LAPORAN PROYEK
SISTEM INFORMASI PENYEWAAN KOST

Disusun Oleh :

Faishal Faruq Nawawi	2200018012
Aqief Idlan Hakimi	2200018051
Rafif Ghani Widiandhi	2200018053
M. Dzaka Al Fikri	2200018057
Farrel Zacky As Syahid R	2200018061

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS AHMAD DAHLAN

2025

HALAMAN PENGESAHAN

SISTEM INFORMASI PENYEWAAN KOST

Faishal Faruq Nawawi	2200018012	Project Manager
Aqief Idlan Hakimi	2200018051	Desainer UI/UX
Rafif Ghani Widiandhi	2200018053	Sistem Analis
M. Dzaka Al Fikri	2200018057	Back-end Developer
Farrel Zacky As Syahid R	2200018061	Front-end Developer

PEMBIMBING : Bambang Robiin, S.T., M.T.
NIP. 197907202005011002 ..... 7/9/2025 .....

PENGUJI : Rusydi Umar, S.T., M.T., Ph.D.
NIPM.
19961104199107111067754O ..... 8/9/2025 .....

Yogyakarta, Sabtu, 23 Agustus 2025

Kaprodi S1 Informatika

Dr. Murinto, S.Si., M.Kom.
NIPM. 19730710200409111095129S

KATA PENGANTAR

Puji syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga kami dapat menyusun proposal proyek dengan judul “Proyek Pengembangan Sistem Informasi Penyewaan Kost”. Proposal ini dibuat sebagai langkah awal dalam mewujudkan solusi digital yang efektif dan efisien untuk mengatasi permasalahan dalam proses penyewaan kost, khususnya bagi pemilik kost pribadi.

Perkembangan teknologi informasi yang pesat memberikan peluang besar untuk meningkatkan sistem penyewaan kost yang selama ini dilakukan secara konvensional. Proyek ini dirancang untuk memberikan kemudahan bagi calon penyewa dalam menemukan kost yang sesuai dengan kebutuhan mereka sekaligus membantu pemilik kost memaksimalkan potensi pemasaran dan pengelolaan properti.

Dalam penyusunan proposal ini, kami berusaha memberikan informasi yang komprehensif, mulai dari latar belakang permasalahan, tujuan proyek, rencana kerja, hingga estimasi biaya yang diperlukan. Kami berharap proyek ini dapat memberikan dampak positif bagi semua pihak terkait, termasuk mendukung transformasi digital di sektor properti.

Kami menyadari bahwa proposal ini masih memiliki kekurangan. Oleh karena itu, kami terbuka untuk menerima masukan dan saran yang membangun demi penyempurnaan proyek ini. Semoga proposal ini dapat memberikan manfaat yang signifikan dan menjadi landasan yang kokoh bagi pelaksanaan proyek ke depannya.

Yogyakarta, 23 Agustus 2025

Tim

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

A. Latar Belakang

Kebutuhan akan hunian sementara seperti kost semakin meningkat, terutama di daerah perkotaan dan pusat pendidikan. Mahasiswa, pekerja, dan pendatang yang membutuhkan tempat tinggal seringkali menghadapi kesulitan menemukan kost yang sesuai dengan preferensi dan anggaran mereka. Permasalahan ini semakin kompleks karena kurangnya transparansi informasi tentang fasilitas, harga, dan ketersediaan kost, serta sulitnya mencapai lokasi tertentu tanpa kunjungan langsung.

Mengandalkan iklan fisik atau rekomendasi dari mulut ke mulut untuk mencari kost seringkali memakan waktu dan tidak efisien. Sebagai pemilik kost, Anda juga menghadapi masalah tambahan seperti tidak dapat menghubungi banyak calon penyewa.

Dengan perkembangan teknologi, pengembangan sistem informasi penyewaan kost berbasis website menjadi solusi modern yang efektif. Website ini dirancang untuk mempermudah penyewa dalam menemukan kost dengan informasi yang jelas dan lengkap terkait fasilitas, harga, lokasi, dan ketersediaan. Sistem ini memberikan keuntungan dalam mengelola data kost secara terpusat, menjangkau calon penyewa secara lebih luas, dan meningkatkan efisiensi proses pemasaran.

B. Project Charter

1. Tujuan

Proyek ini bertujuan untuk merancang dan mengembangkan sebuah website penyewaan kost yang intuitif, informatif, dan efisien. Dengan adanya sistem ini, diharapkan dapat meningkatkan kemudahan

penyewaan kost, memaksimalkan pendapatan, serta menciptakan hubungan yang lebih baik antara pemilik kost dan penyewa.

Dengan ini, website yang menyediakan informasi lengkap dan akurat tentang kost milik pribadi, termasuk fasilitas, harga, lokasi, dan ketersediaan. Memberikan pemilik kost kemampuan untuk mengelola data kost secara efisien, termasuk pembaruan informasi dan pengelolaan ketersediaan.

2. Ruang Lingkup

Tabel 1. 1 Ruang Lingkup

	Kost Putri MEWAH Patemon	
Project Manager Faisal Faruq Nawawi, +62 821 5876 3410		Project Date 02 October 2024
Project Objective	Proyek ini merupakan bidang Real Estate (Properti tak bergerak) dengan sasaran yang ditujukan untuk individu yang mencari tempat tinggal bersifat sementara. Tujuan dari proyek ini adalah menyediakan sarana visual untuk menunjukkan sarana dan prasarana yang dimiliki oleh client serta memberikan sistem yang lebih baik dalam melakukan transaksi sewa properti yang dimiliki oleh client. Sistem yang diberikan memiliki kelebihan dalam kemudahan manajemen keuangan dengan antarmuka yang minimalis namun disertai fitur yang lengkap.	

Deliverables	Dalam proposal proyek ini, tim proyek berkomitmen untuk menyediakan deliverables yang akan mendukung kesuksesan manajemen property client. 1. Tim proyek akan mengembangkan sistem manajemen sewa properti yang intuitif dan mudah digunakan, dilengkapi dengan fitur manajemen keuangan yang lengkap untuk memudahkan pengelolaan aset. 2. Kami akan menghadirkan presentasi visual yang menarik untuk memperlihatkan fasilitas properti, dengan tujuan menarik lebih banyak calon penyewa. 3. Tim proyek akan mengelola website yang memungkinkan pengguna untuk melakukan transaksi dengan mudah dan cepat, serta menciptakan mekanisme umpan balik dari pengguna dan admin untuk memastikan operasional berjalan optimal. 4. Strategi pemasaran digital yang terarah juga akan dirancang oleh tim proyek untuk menarik segmen penyewa sementara seperti mahasiswa, dokter, dan pekerja. Tim proyek menargetkan penyelesaian proyek ini dalam jangka waktu 4-6 bulan, dengan jaminan kompensasi apabila terjadi keterlambatan pengerjaan.

3. Stakeholder

Tabel 1. 2 Stakeholder

	Stakeholder				
	Faisal Faruq	Aqief I Hakimi	M. Dzaka A	Rafif Ghani W	Farrel Zacky
Jabatan	Project Manager	Project Team	Project Team	Project Team	Project Team
Peran	Pemimpin Proyek	Desainer UI/UX	Back-end Developer	Perancang Sistem	Front-end Developer
Fakta	Memiliki ketrampilan kepemimpinan yang baik	Ketrampilan manajemen waktu dan kreatifitas tinggi	Ahli dalam logika pemrograman	Ahli dalam perancangan sistem aplikasi yang terstruktur	Ahli dalam layouting website

BAB II
PERENCANAAN PROYEK

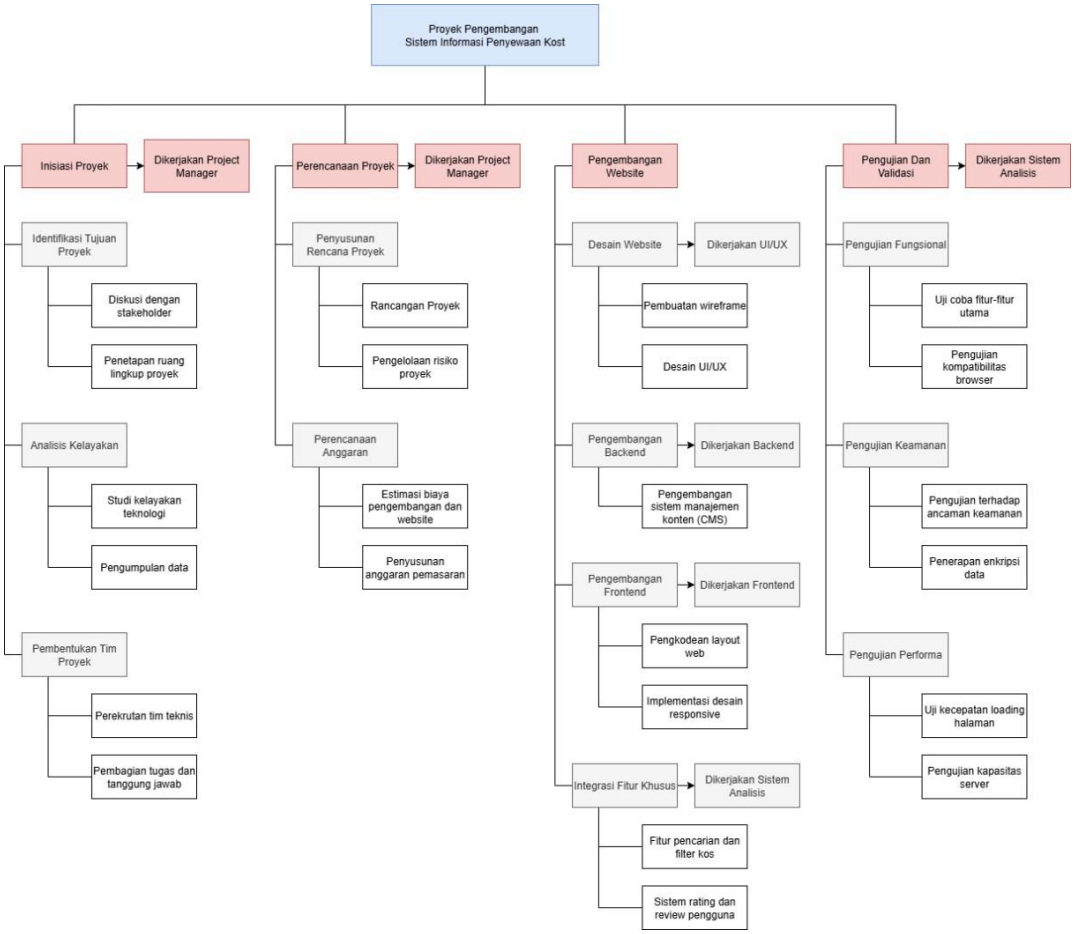
A. Analisis Kelayakan

Tabel 2. 1 SWOT

STRENGTHS	WEAKNESS
- Sistem website mudah dioperasikan bagi pemilik kos dan pengguna. - Berfokus pada satu kos saja, sehingga fokus pada kebutuhan pengguna lokal. - Desain website yang menarik dan fungsional.	- Kurangnya sumber daya dalam pemeliharaan dan pembaharuan.
OPPORTUNITIES	TREATS
- Memperluas peluang untuk meningkatkan pembayaran online. - Berpotensi dikembangkan menjadi sistem multi-kost.	- Risiko kurangnya promosi dapat membuat sistem jarang diakses. - Risiko gangguan teknis seperti server down, masalah bug, atau masalah koneksi internet.

B. Work Breakdown Structure

Gambar 2. 1 Work Breakdown Structure



C. Kebutuhan Sumber Daya

1. Sumber Daya Manusia

Tabel 2. 2 Sumber Daya Manusia

Name	Role	Responsibilities
Faisal Faruq Nawawi	Project Manager	Pemimpin Proyek
Aqief Idlan Hakimi	Project Team	Desainer UI/UX
M. Dzaka Al Fikri	Project Team	Back-end Developer
Rafif Ghani Widiandhi	Project Team	Perancang Sistem
Farrel Zacky A. R	Project Team	Front-end Developer

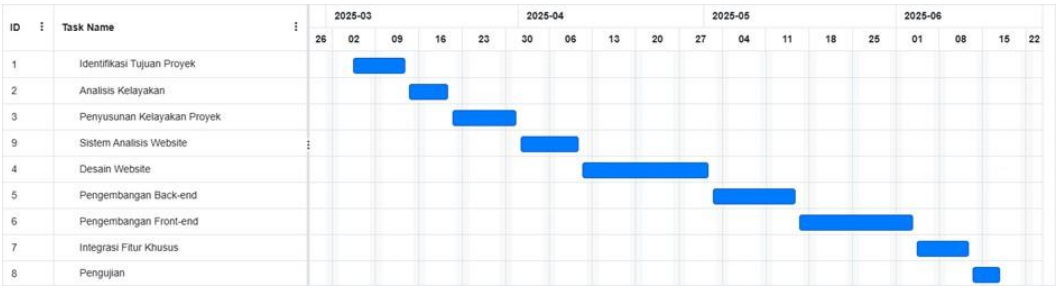
2. Sumber Daya Fisik

Tabel 2. 3 Sumber Daya Fisik

	Nama Alat
Hardware	Wireless Fidelity
	PC
	Laptop
Software	Visual Studio Code
	XAMPP
	Figma
	Adobe Illustrator
	Postman
	GitHub

D. Rencana Jadwal Pelaksanaan Proyek

Gambar 2. 2 Gantt Chart



E. Rencana Nilai Proyek

1. Biaya Alat dan Bahan

Tabel 2. 4 Biaya Alat dan Bahan

No	Keterangan	Jumlah	Harga (Rp)
1	Hosting (Shared Hosting, 25 GB SSD Storage)	Per Tahun	750.000
2	Domain	Per Tahun	150.000
Total			900.000

2. Biaya Tenaga Kerja

Tabel 2. 5 Biaya Tenaga Kerja

No	Keterangan	Jam Kerja	Harga perJam (Rp)	Total Biaya (RP/Jam)
1	Sistem Analis	10 Jam	20.000	200.000
2	Desainer	14 Jam	25.000	350.000
3	Programmer, Tester	15 Jam	30.000	450.000
Total				1.000.000

3. Rekapitulasi

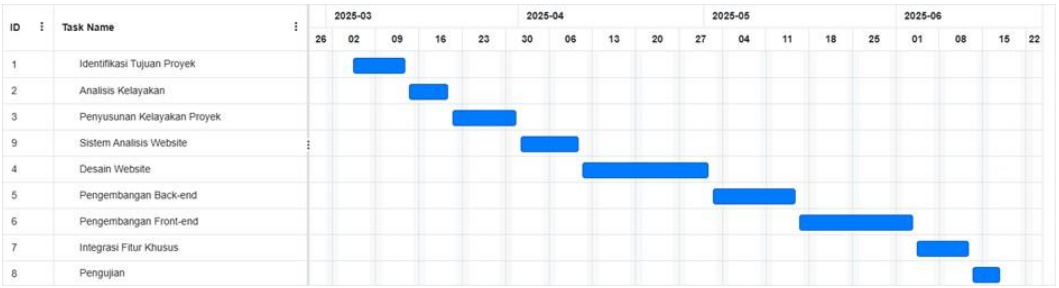
Tabel 2. 6 Rekapitulasi

No	Keterangan		Harga (Rp)
1	Biaya Alat dan Bahan	Database	900.000
2	Biaya Tenaga Kerja	Tenaga Kerja	1.000.000
Total			1.900.000

BAB III
PELAKSANAAN PROYEK

A. Realisasi Jadwal Pelaksanaan

Gambar 3. 1 Realisasi Jadwal Pelaksanaan



B. Analisis Kebutuhan Sistem

Ada dua jenis kebutuhan sistem, yaitu kebutuhan fungsinal dan non fungsional. Analisis kebutuhan fungsional adalah pernyataan layanan yang harus disediakan untuk sistem untuk melakukan input tertentu dalam keadaan tertentu. Analisis kebutuhan non-fungsional mencakup batasan pada layanan atau fungsi yang disediakan oleh sistem. Adapun analisis kebutuhan fungsional dan non-fungsional dijabarkan sebagai berikut:

Memastikan proyek dapat dijalankan dengan sumber daya, waktu, dan anggaran yang tersedia. Analisis ini melibatkan studi kelayakan teknis untuk mengevaluasi teknologi yang digunakan pada Tabel 3.1.

Tabel 3. 1 Non-Fungsional

Kegiatan	Keterangan
Kelayakan Operasi	Menjalankan sistem ini tidak diperlukan tenaga atau user yang benar - benar ahli. Sistem mudah dioperasikan bahkan oleh orang yang awam terhadap teknologi.

Kelayakan Teknis	Kost memiliki fasilitas dan sumber daya yang memadai.
Kelayakan Ekonomi	Estimasi biaya, proyeksi keuntungan, model monetisasi.

Pengujian dilakukan untuk memastikan sistem berjalan sesuai dengan spesifikasi dan kebutuhan pengguna pada Tabel 3.2.

Tabel 3. 2 Fungsional

Kegiatan	Keterangan
Fungsional	Fitur-fitur utama, seperti pencarian kamar, transaksi pembayaran, dan manajemen data kost.
Kompatibilitas	Memastikan website dapat diakses melalui berbagai browser dan perangkat.
Keamanan	Simulasi serangan untuk mengidentifikasi potensi kerentanan dan penerapan enkripsi data.
Performa	Mengevaluasi kecepatan loading halaman dan kapasitas server dalam menangani lalu lintas pengguna.

C. Realisasi Hasil Pekerjaan

Tabel 3. 3 Tabel Realisasi Hasil Pekerjaan

Nama	Jabatan	Realisasi hasil pekerjaan
Faishal Faruq Nawawi	Project Manager	1. Mengoordinasikan tugas antaranggota, dan menyatukan tim.

		2. Mencatat notulen rapat, dan menyusun laporan perkembangan proyek. 3. Melakukan komunikasi dengan stakeholder, dan dosen pembimbing. 4. Mengarsipkan dokumen dan versi final file proyek.
Aqief Idlan Hakimi	Desainer UI/UX	1. Membuat wireframe dan mockup seluruh halaman. 2. Melakukan riset antarmuka dan pengalaman pengguna agar website mudah digunakan. 3. Menentukan palet warna, tipografi, dan ikonografi yang konsisten. 4. Melakukan pengujian usability secara terbatas dan menyempurnakan desain.

Rafif Ghani Widiandhi	Sistem Analis	1. Mengumpulkan kebutuhan sistem dari stakeholder. 2. Menyusun dokumen kebutuhan fungsional dan non-fungsional. 3. Mendesain arsitektur sistem, relasi antar modul, dan alur data. 4. Berkoordinasi dengan frontend dan backend untuk memastikan implementasi sesuai analisis.
M. Dzaka Al Fikri	Back-end Developer	1. Membangun server menggunakan Node.js dan Express.js. 2. Membuat endpoint untuk registrasi, login, reservasi, dan pengelolaan data penyewa. 3. Mengelola basis data MySQL untuk menyimpan informasi pengguna dan transaksi. 4. Mengimplementasikan middleware validasi dan keamanan akses endpoint.

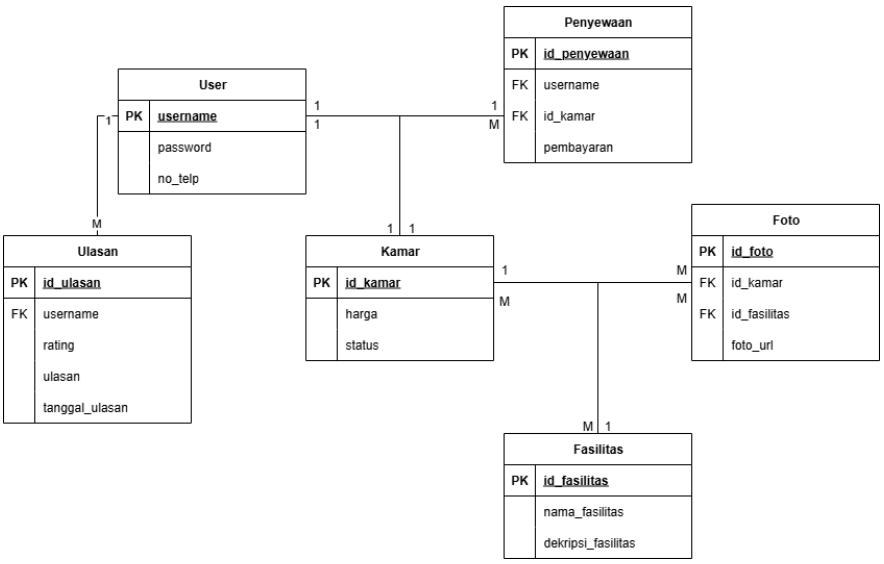
Farrel Zacky A. R	Front-end Developer	1. Menerjemahkan desain UI/UX ke dalam kode HTML, CSS, dan JavaScript. 2. Membangun halaman dinamis. 3. Mengintegrasikan API dari backend untuk menampilkan data real-time. 4. Mengoptimalkan tampilan agar responsif di berbagai perangkat.

1. Sistem Analisis

1.1. Membuat Entity Relationship Diagram (ERD)

Basis data untuk sistem penyewaan kamar kost. Rancangan struktur database untuk sistem ini harus dirancang dengan cermat agar mendukung efisiensi, akurasi, dan keamanan dalam mengelola data keuangan yang sensitif. Pada Gambar 3.1 terdapat 6 tabel dalam database, user, penyewaan, ulasan, kamar, fasilitas, foto.

Gambar 3. 2 ERD

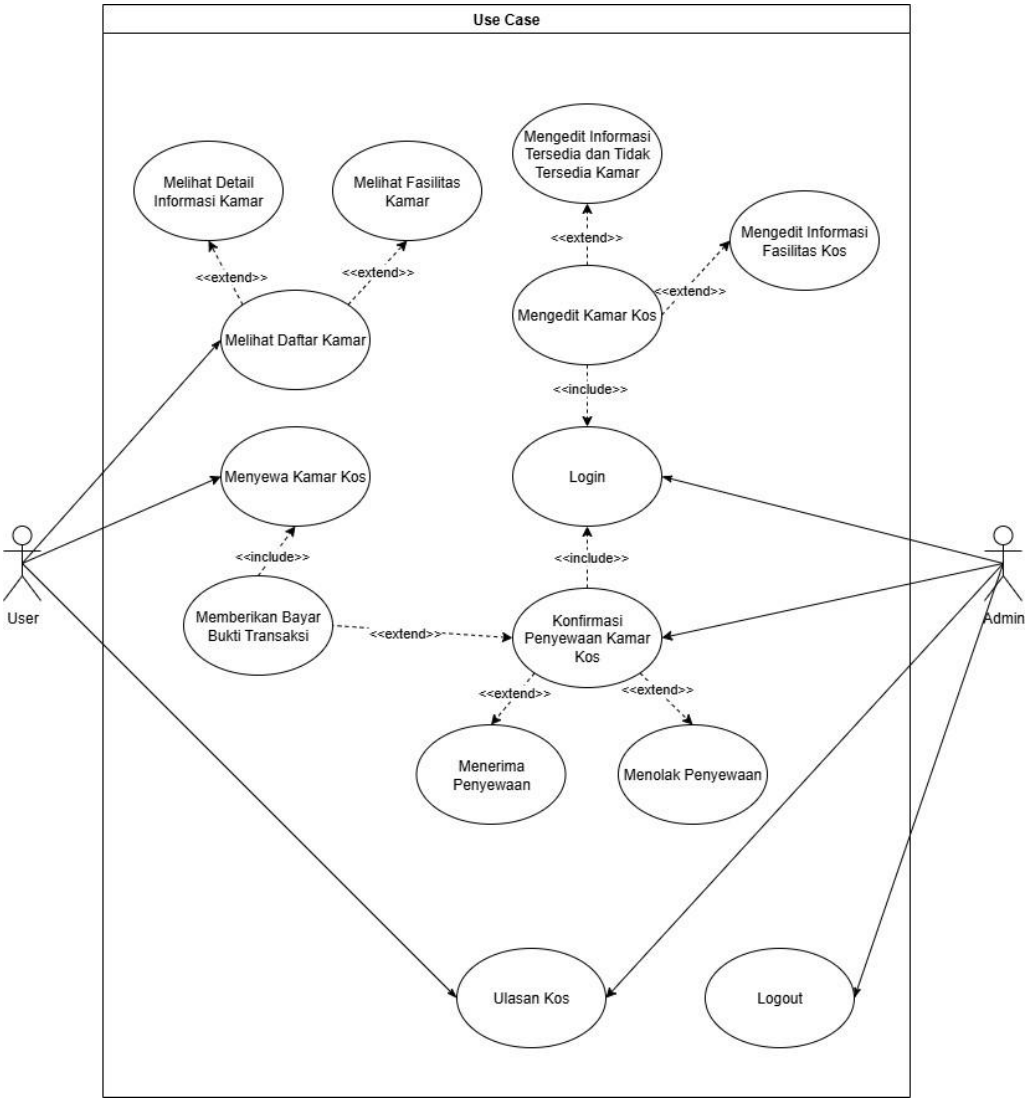


1.2. Membuat Use Case Diagram

Use case diagram di bawah dengan Gambar 3.2 menggambarkan alur fungsional dari sistem informasi kost yang melibatkan dua aktor utama, yaitu User (pengunjung atau calon penyewa kost) dan Admin (pemilik atau pengelola kost). Pada sisi user, sistem memungkinkan pengguna untuk melihat daftar kamar yang tersedia dan tidak tersedia, melihat fasilitas kamar kost dan kost, memberikan ulasan terhadap kost yang pernah digunakan, serta melakukan penyewaan kamar kost.

Sementara itu, admin memiliki akses yang lebih luas karena berfungsi sebagai pengelola data utama. Admin perlu melakukan login terlebih dahulu sebelum dapat menambahkan kamar yang tersedia maupun menambahkan fasilitas kost. Setelah menambahkan kamar, admin juga dapat memperluas fungsinya dengan mengedit data kamar yang tersedia, atau menghapus kamar dari sistem. Begitu pula pada manajemen fasilitas, admin memiliki opsi untuk mengedit informasi fasilitas atau menghapus fasilitas kost.

Gambar 3. 3 Use Case



1.3. Membuat Activity Diagram

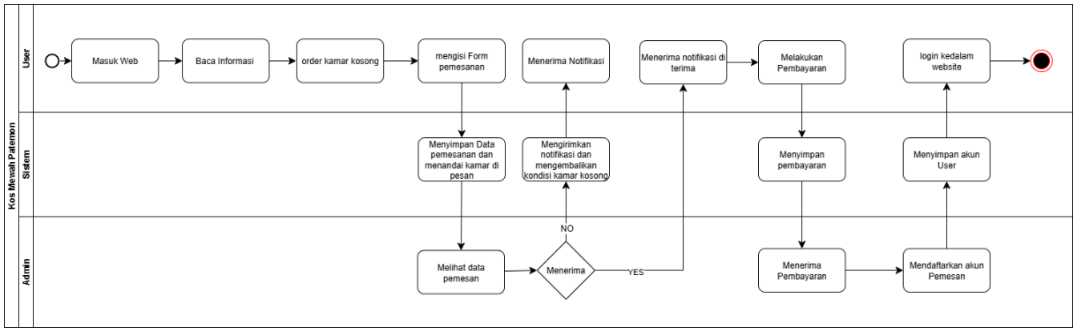
Activity diagram merupakan pemodelan untuk melihat bagaimana cara suratu calon aplikasi bekerja menjalankan fungsi-fungsinya. Dalam activity diagram system informasi ini terdapat 3 activity yaitu user melakukan penyewaan kamar kost, user melakukan pembayaran, user keluar dari kamar kost.

a. User Melakukan Penyewaan Kamar Kost

Gambar 3.3 merupakan activity diagram pada saat user masuk ke website, kemudian akan tampil berbagai ketersediaan kamar. User dapat melakukan penyewaan kamar kost, jika kamar

tersebut tersedia dengan mengisi form pemesanan (reservasi) dan melakukan pembayaran jika diterima admin melalui notif email dari admin. Setelah user melakukan pembayaran dan diterima admin, user dapat login pada website kost.

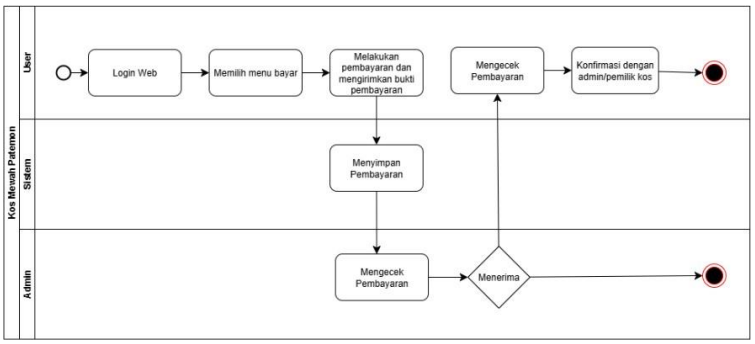
Gambar 3. 4 Activity Diagram User Melakukan Penyewaan Kamar Kost



b. User Melakukan Pembayaran

Gambar 3.4 merupakan activity diagram pada saat user melakukan pembayaran dan akan diterima oleh admin. Admin akan memberikan notifikasi email penerimaan penyewa.

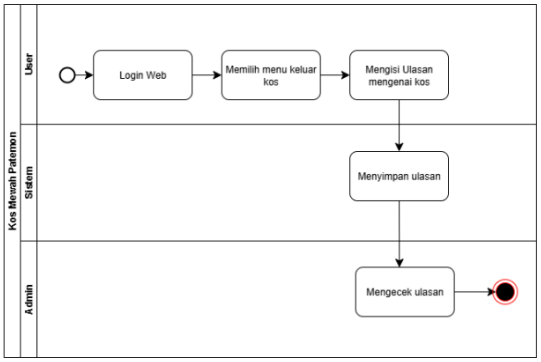
Gambar 3. 5 Activity Diagram User Melakukan Pembayaran



c. User Keluar Dari Kamar Kost

Gambar 3.5 merupakan activity diagram pada saat user keluar dari kamar kost yang telah ditempatinya. User akan menerima feedback ulasan mengenai kost yang telah ditempatinya.

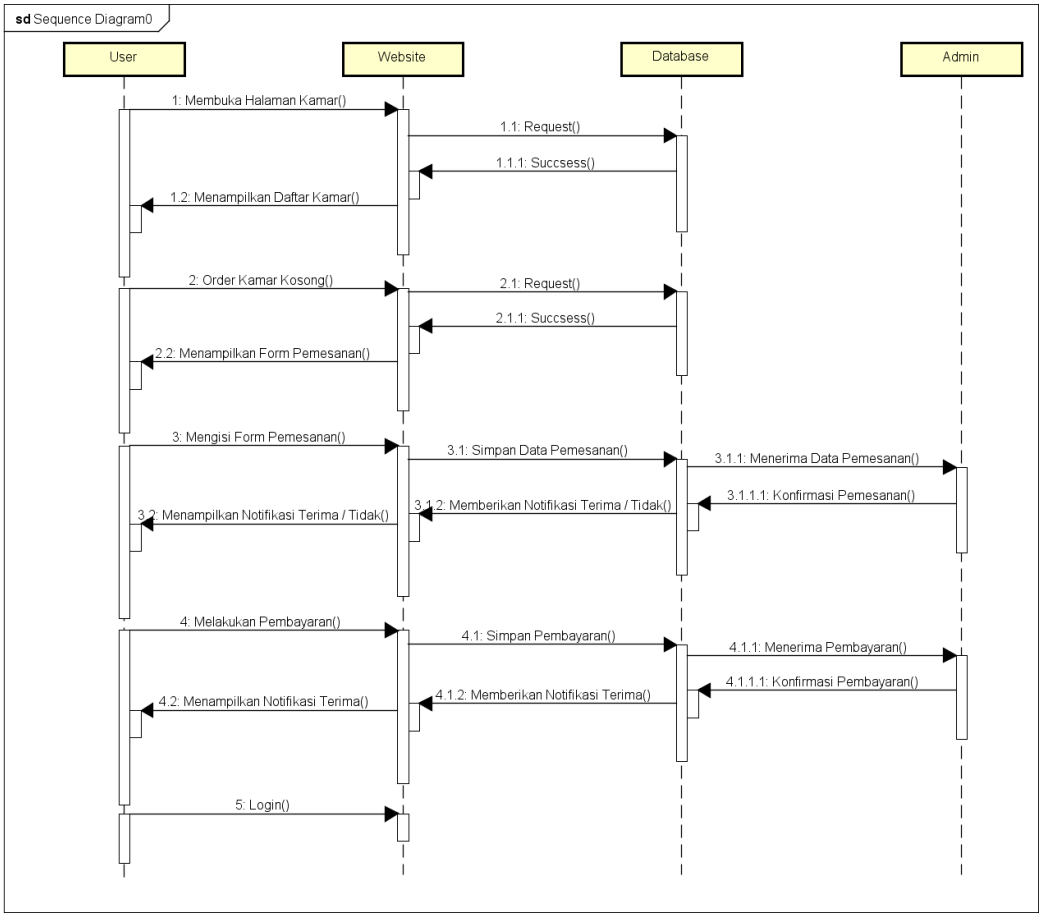
Gambar 3. 6 Activity Diagram User Keluar Dari Kamar Kost



1.4. Membuat Sequence Diagram

Sequence diagram pada Gambar 3.6 menggambarkan alur interaksi antara aktor User, sistem Website, Database, dan Admin dalam proses pemesanan kamar kost secara online. Diagram ini secara jelas menunjukkan urutan komunikasi dan alur logis antara komponen sistem, mulai dari interaksi awal user hingga proses backend oleh admin, sehingga membantu memahami implementasi fitur pemesanan kamar dalam sistem informasi kost secara menyeluruh.

Gambar 3. 7 Sequence Diagram

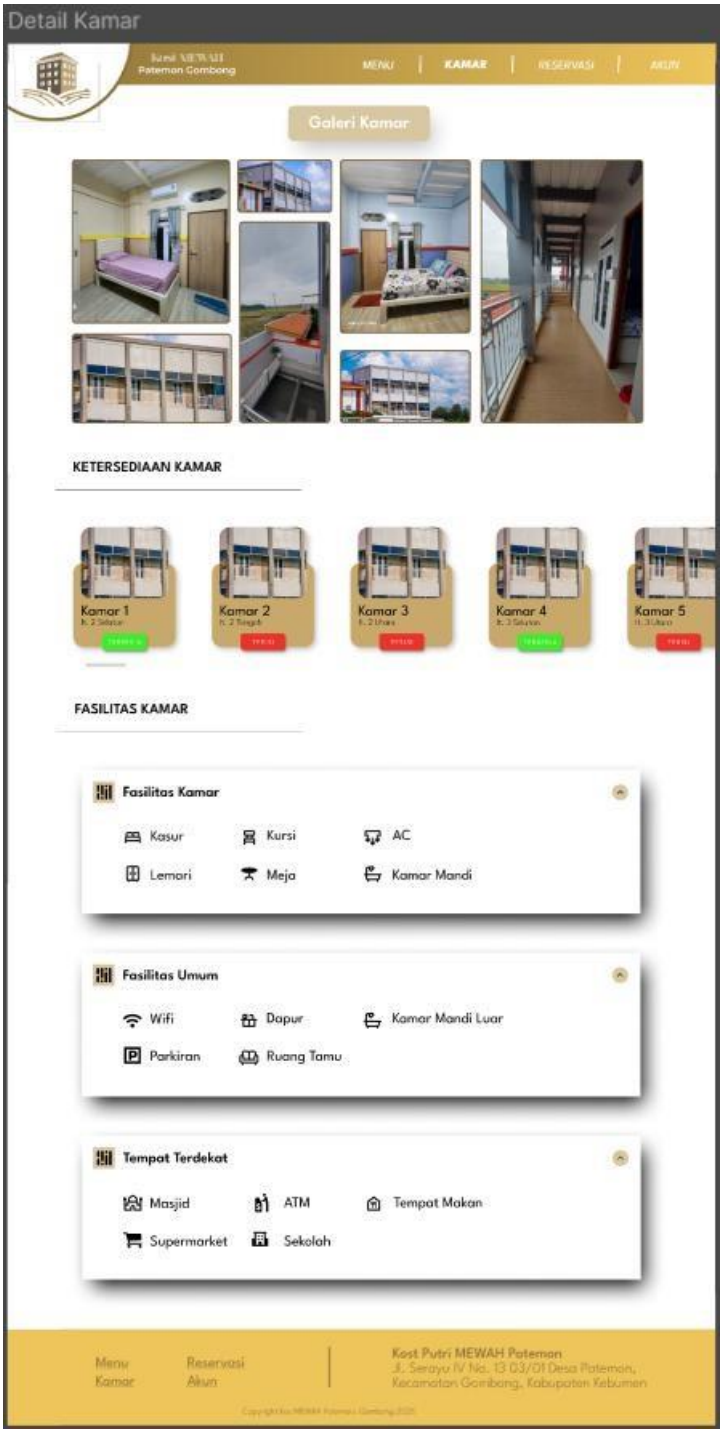


2. UI/UX

Gambar 3. 8 Design Halaman Utama

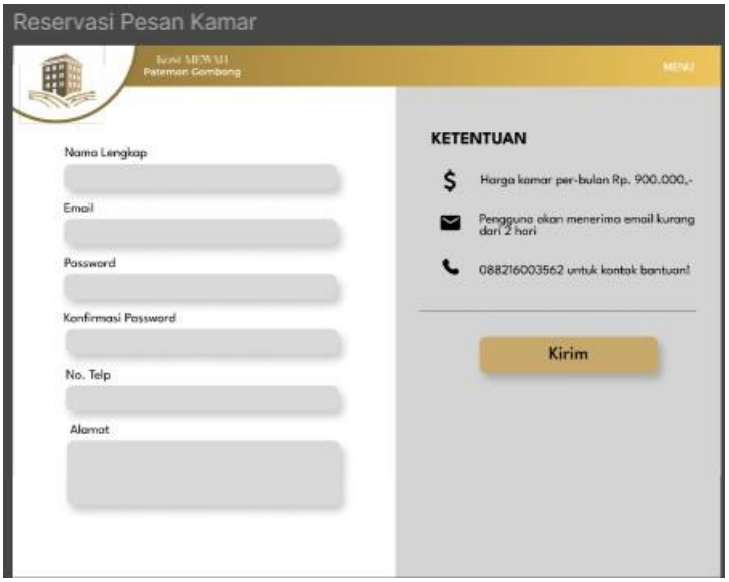
Gambar 3.7 merupakan halaman utama website menampilkan keunggulan kos, ulasan kos dari pengguna-pengguna sebelumnya, lokasi tempat kos berada.

Gambar 3. 9 Design Halaman Kamar



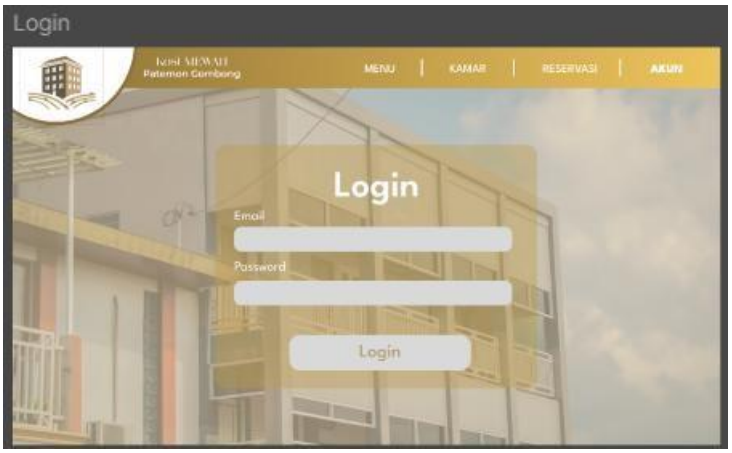
Pada Gambar 3.8 halaman kamar, website menampilkan beberapa kamar yang tersedia dan fasilitas kost, seperti fasilitas kamar, fasilitas umum, tempat terdekat dari lokasi kost berada. User dapat Memesan kamar kost yang tersedia, pada halaman ketersediaan kamar.

Gambar 3. 10 Desgin Reservasi Pesanan Kamar



Gambar 3.9 merupakan Halaman reservasi merupakan dimana user mengisi form data pribadi dengan memasukkan nama lengkap, email, password, nomor telepon, alamat sebelum melakukan transaksi penyewaan kamar kost yang nanti akan diberikan melalui email.

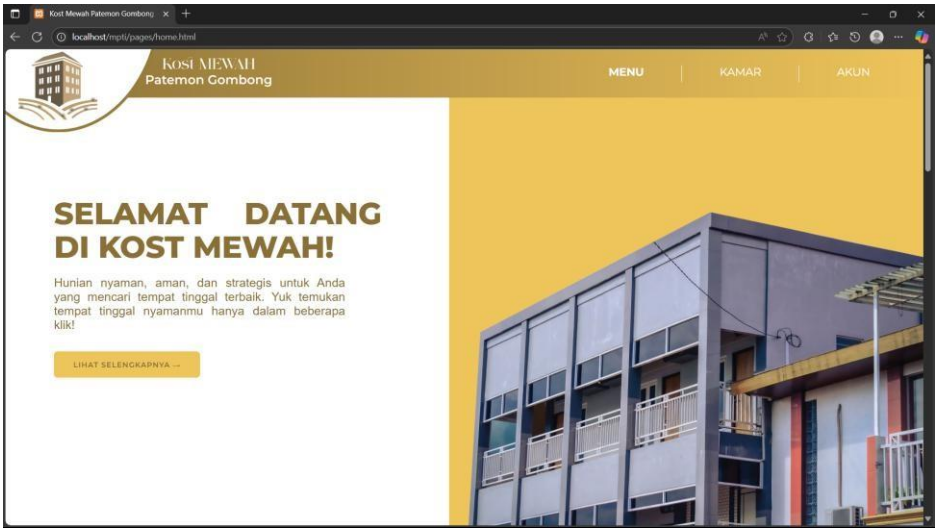
Gambar 3. 11 Design Halaman Login

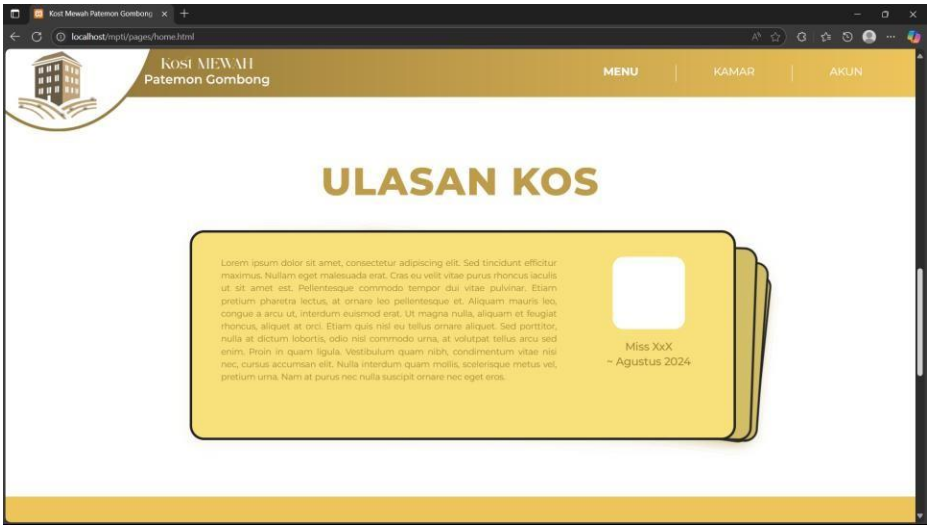


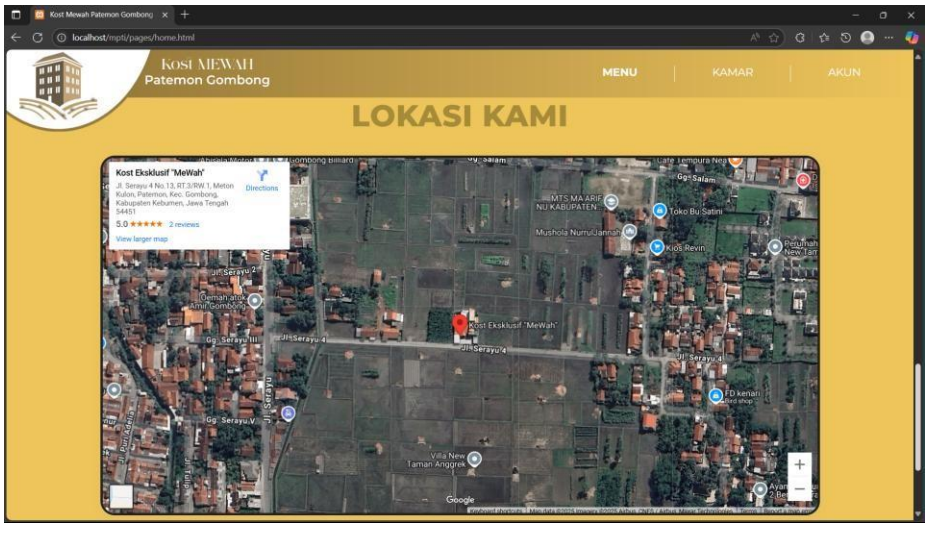
Gambar 3.10 merupakan Halaman login merupakan halaman dimana user login ketika sudah diterima oleh admin sebagai penyewa.

3. Front-End Development

Gambar 3. 12 Halaman Utama

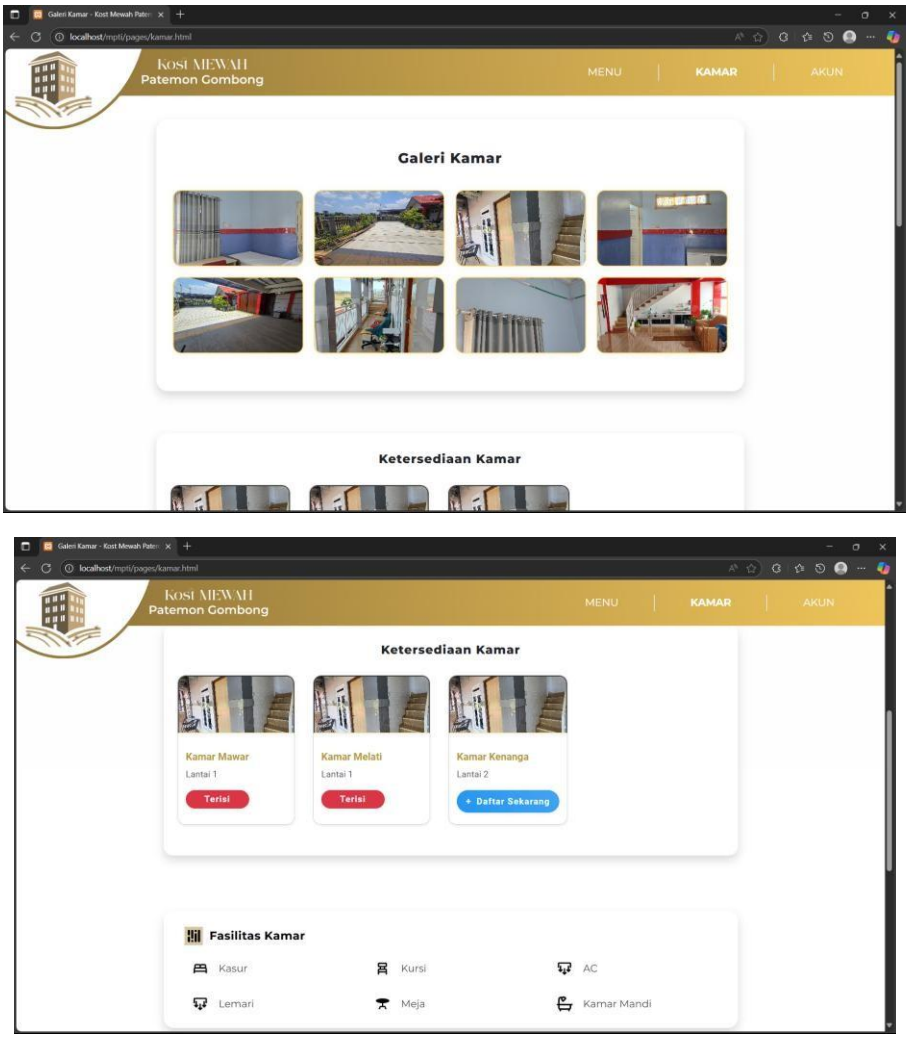


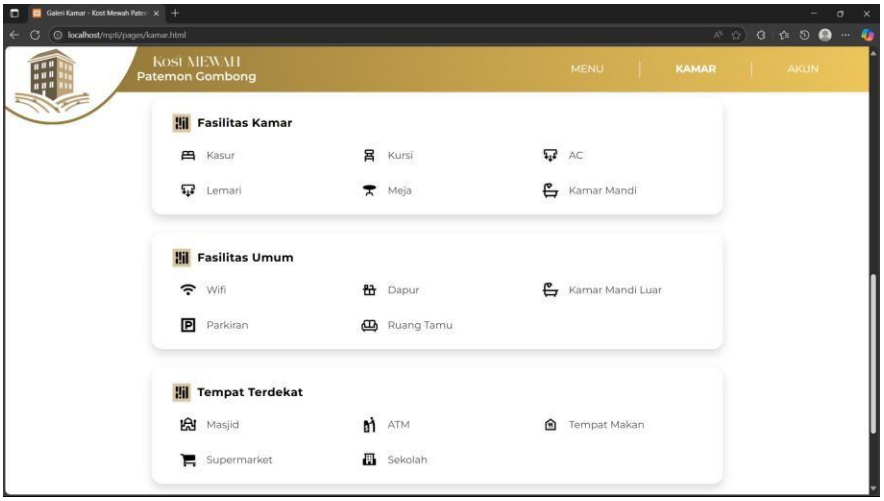




Gambar 3.11 merupakan halaman home ini menampilkan struktur utama dari website Kost Mewah Patemon Gombong, dengan elemen navigasi, konten pengenalan, fitur, ulasan penyewa, dan lokasi. Bagian utama menampilkan sambutan selamat datang dengan gambar ilustratif dan tombol menuju bagian fitur. Fitur-fitur kost ditampilkan dalam bentuk slide menggunakan komponen Swiper seperti parkiran luas, kenyamanan, dan lainnya. Di bawahnya, terdapat bagian ulasan dari penyewa yang juga dikemas dengan Swiper slide agar menarik dan dinamis. Terakhir, website menampilkan lokasi kost yang memberikan kemudahan pengguna untuk mengetahui alamat secara akurat.

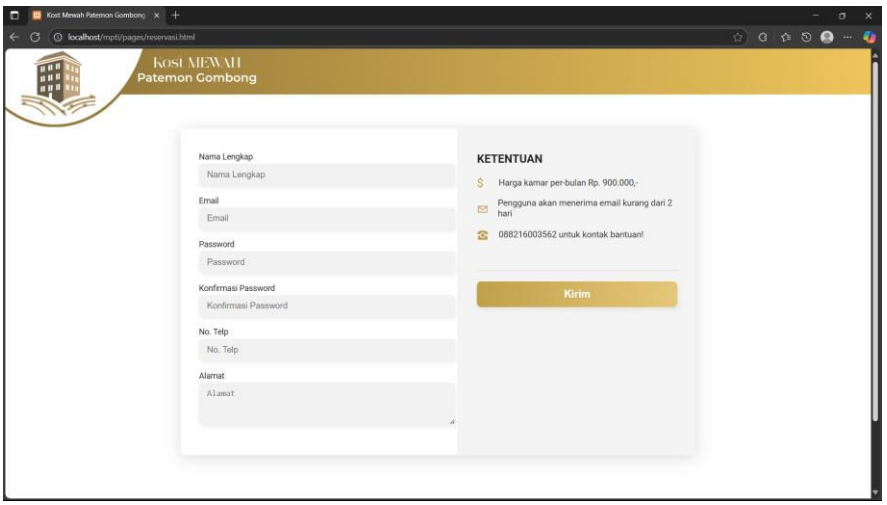
Gambar 3. 13 Halaman Kamar





Gambar 3.12 merupakan halaman kamar berfungsi untuk menampilkan detail lengkap seputar kamar kost. Pada bagian pertama, ditampilkan Galeri Kamar dalam bentuk grid gambar, memungkinkan pengunjung melihat kondisi visual kamar melalui berbagai foto. Bagian kedua menampilkan Ketersediaan Kamar, menggunakan Swiper slider interaktif. Bagian Fasilitas Kost menyajikan daftar kelengkapan kamar seperti kasur, meja, atau lemari dalam format ikon dan label teks, disusun dalam sistem grid yang responsif.

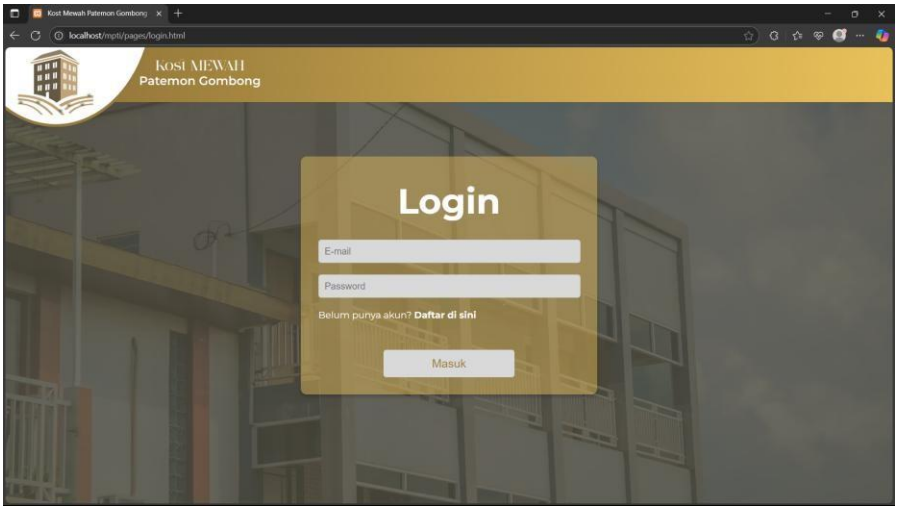
Gambar 3. 14 Halaman Reservasi



Gambar 3.13 merupakan halaman reservasi merupakan halaman form reservasi kamar yang terdiri dari beberapa bagian utama. Bagian pertama adalah form pengisian data penyewa, yang mencakup input seperti nama, email,

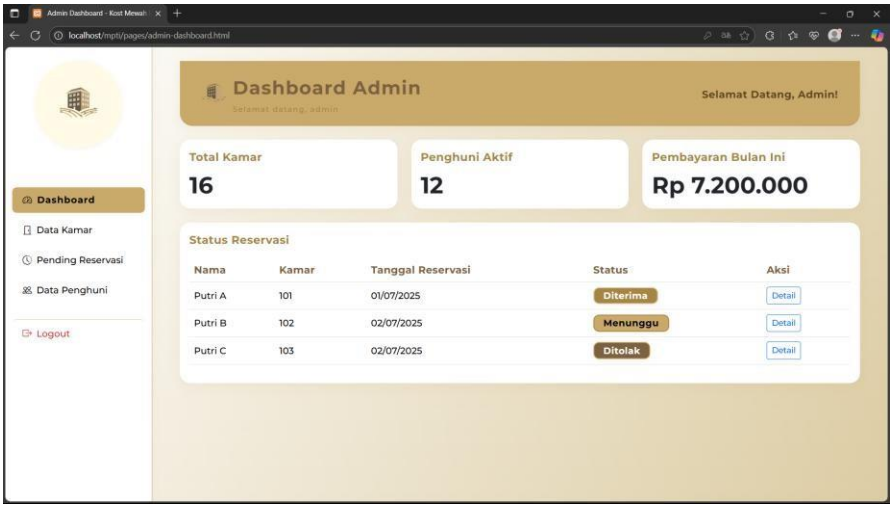
password, nomor telepon, dan alamat lengkap. Form ini digunakan untuk merekam data profil pengguna sebelum melakukan reservasi. Di sisi kanan, terdapat bagian informasi ketentuan. Setelah mengirimkan data, akan muncul modal pembayaran yang menampilkan instruksi pembayaran secara detail.

Gambar 3. 15 Halaman Login



Gambar 3.14 merupakan halaman login berfungsi sebagai gerbang autentikasi. Terdapat form login utama yang meminta pengguna memasukkan email dan password untuk masuk ke sistem.

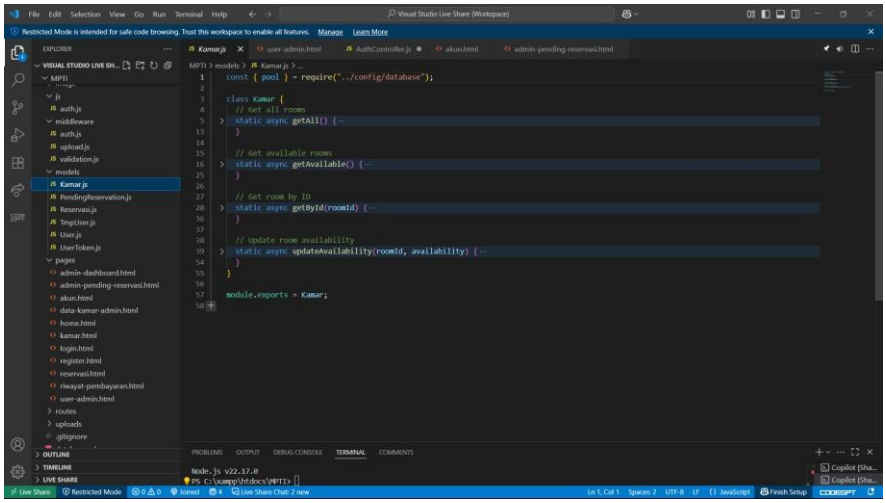
Gambar 3. 16 Halaman Dashboard Admin



Dashboard admin pada Gambar 3.15 merupakan halaman utama bagi admin untuk sistem manajemen kost. Admin memiliki fitur data kamar, pending reservasi, data penghuni. Di bagian konten utama terdapat tiga kartu ringkasan statistik, yaitu total kamar, penghuni aktif, dan total pembayaran bulan ini.

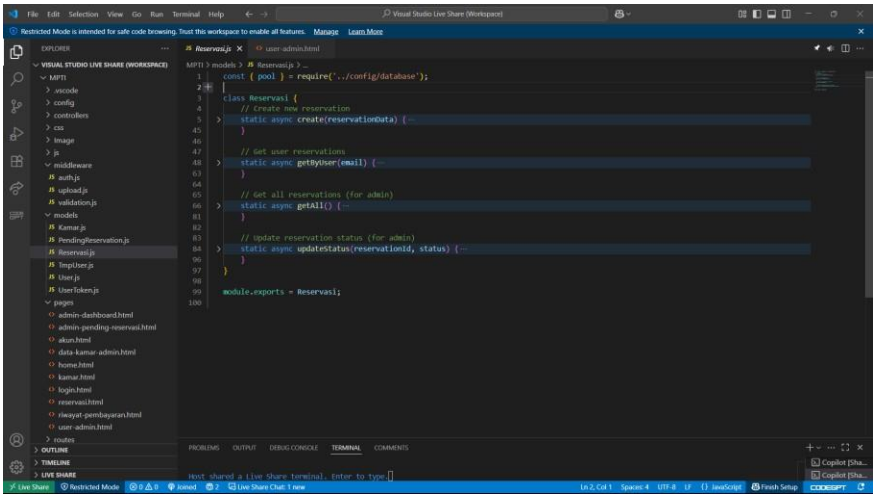
4. Back-End Development

Gambar 3. 17 Menangani Kamar



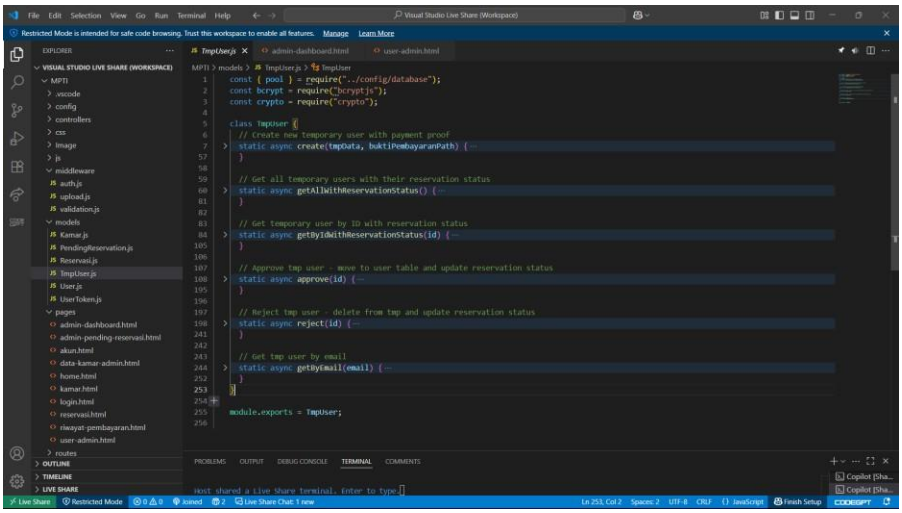
Gambar 3.16 menangani seluruh proses interaksi pada halaman Reservasi, mulai dari membuka modal pembayaran, validasi form, hingga pengiriman data ke server. Saat pengguna menekan tombol Kirim, modal instruksi pembayaran muncul. Setelah mengisi semua data dan mengunggah bukti transfer, form akan divalidasi.

Gambar 3. 18 Menangani Reservasi



Model Reservasi pada Gambar 3.17 mengelola seluruh logika terkait proses reservasi kamar kost. Dengan metode seperti create, getByUser, getAll, dan updateStatus, model ini memastikan hanya kamar yang tersedia yang bisa dipesan, mendukung pelacakan reservasi oleh pengguna dan admin, serta memungkinkan pembaruan status reservasi secara fleksibel. Struktur ini menjaga integritas data dan mempermudah koordinasi antara sistem backend dan antarmuka pengguna.

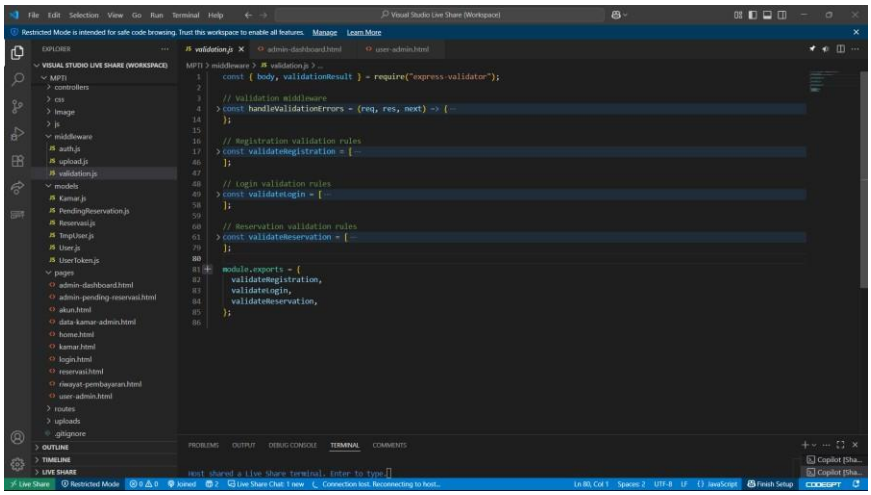
Gambar 3. 19 Menangani Calon Penghuni



Gambar 3.18 mengelola seluruh alur registrasi calon penghuni kost secara sementara hingga disetujui oleh admin. Fungsi create() menyimpan data penyewa

dan bukti pembayaran ke tabel tmp, sekaligus membuat entri reservasi dengan status Menunggu. Admin dapat meninjau dan memutuskan apakah akan menyetujui (dengan memindahkan data ke tabel user, mengubah status reservasi, menandai kamar sebagai terisi, serta memberikan token login) atau menolak (menghapus data dan memperbarui status menjadi Ditolak). Model ini juga mendukung pengambilan data berdasarkan ID atau email, dan memastikan validasi serta transaksi data dilakukan secara aman dan konsisten.

Gambar 3. 20 Menangani Validasi



Sistem secara sistematis memastikan integritas dan kualitas data pengguna untuk proses registrasi, login, dan reservasi. Pada Gambar 3.19 dengan menggunakan middleware dari express-validator, setiap input diperiksa sesuai kriteria yang relevan, seperti format email, panjang password, validitas nomor telepon Indonesia, serta tanggal reservasi yang tidak boleh berada di masa lalu. Jika terdapat kesalahan, sistem segera menghentikan proses dan mengembalikan respon JSON berisi daftar kesalahan. Pendekatan ini membantu menjaga keamanan aplikasi dan pengalaman pengguna yang lebih baik dengan mencegah data yang tidak valid masuk ke sistem.

5. Testing

Testing yang disajikan menggunakan metode pengujian Black Box

Tabel 3. 4 Pengujian Pada Users

NO	Pengujian	Skenario	Hasil Yang di harapkan	Kesimpulan
1.	Saat klit menu	Klit menu	Akan menampilkan menu informasi mengenai kost yang berisi keunggulan kos kami, ulasan kos , dan lokasi kos.	Valid
2.	Saat klik kamar	Klik kamar	Akan menampilkan informari mengenai kamar seperti galeri kamar, ketersediaan kamar, fasilitas kamar, fasilitas umum, dan tempat terdekat.	Valid
3.	Saat klik daftar sekarang pada bottom ketersediaan kamar	Klik daftar sekarang	Akan memasuki menu pendaftaran yang berisi banyak data diri yang di perlukan.	Valid
4.	Saat klik bottom kirim reservasi	Klik kirim reservasi	Akan mengirimkan perdaftaran ke admin untuk di proses	Valid
5.	Saat klik akun	Klik akun	Akan menampilkan form login karena menjadi syarat keamanan data.	Valid
6.	Saat klik bottom Ganti password	Klik Ganti password	Akan memunculkan bagian halaman kecil untuk merubah dari password yang lama dan memasukan password yang baru	Valid

7.	Saat klik pembayaran bulanan	Klik pembayaran bulanan	Akan muncul halaman kecil seperti rubah password tetapi berisi tatacara membayar dan jumlah tagihan.	Valid
8.	Saat klik Riwayat pembayaran	klik Riwayat pembayaran	Akan berpindah ke halaman yang berisi informasi pembayaran , status pembayaran sudah di terima atau masih menunggu verifikasi dari admin.	Valid
9.	Saat klik tombol filter Riwayat dengan sebelumnya memasukan pilihan bulan, status, dan tahun	Klik filter	Akan memunculkan table yang berisi pembayaran yang telah terjadi dari yang telah kita tentukan bulan tahun dan status nya.	Valid
10.	Saat klik detail pada Riwayat pendaftaran	Klik detail	Akan menampilkan detail dari pembayaran tersebut dari bukti dan lain-lain.	Valid
11.	Saat klik pada tulis ulasan	Klik tulis ulasan	Akan memunculkan halaman kecil Dimana pemakai kos bisa memberikan ulasan dan rating untuk kepuasan.	Valid
12.	Saat klik logout	Klik logout	Akan membuat keluar dari akun yang tekah masuki	Valid
13.	Saat klik edit profil	Klik edit profil	Akan memunculkan halaman kecil yang berisi informasi dari akun atau diri kita dan bisa kita ubah di dalamnya.	Valid

Tabel 3. 5 Pengujian Pada Admin

NO	Pengujian	Skenario	Hasil Yang di harapkan	Kesimpulan
1.	Saat klik dashboard	Klit dasboard	Akan menampilkan banyak informasi seperti total kamar, penghuni aktif, total pembayaran yang sudah masuk di bulan tersebut, dan masih banyak lagi.	Valid
2.	Saat klik pending reservsi	Klik pending reservasi	Akan menampilkan halaman yang berisi reservasi yang masih pending.	Valid
3.	Saat klik reservasi	Klik reservasi	Akan menampilkan siapa saja yang telah melakukan reservasi.	Valid
4.	Saat klik pembayaran	Klik pembayaran	Akan menampilkan informasi siapa saja yang belom membayar dan di halaman ini admin dapat memberikan konfirmasi apakah pembayaran sudah atau belom dengan klik bottom bergambar pensil , dan untuk melihat saja bisa klik bottom bergambar mata.	Valid
5.	Saat klik data kamar	Klik data kamar	Akan menampilkan halaman yang berisi informasi mengenai setiap kamar , di halaman ini admin bisa merubah status kamar sudah terpakai atau belom.	Valid

6.	Di dalam data kamar klik tambah kamar baru	Klik kamar baru	Akan memunculkan halaman kecil Dimana admin bisa menambahkan kamar baru.	Valid
7.	Saat klik data akun	Klik data akun	Akan menampilkan data dari setiap akun penghuni kamar .	Valid
8.	Saat klik logout	Klik logout	Akan membuat keluar dari akun yang tekah masuki	Valid

HASIL PERHITUNGAN

$$hasil = \frac{total\ jumlah\ pertanyaan\ valid}{total\ keseluruhan\ pertanyaan}\ x\ 100\% \quad (3.1)$$

$$hasil = \frac{21}{21}\ x\ 100\% = 100\%$$

D. Penjaminan Kualitas Proyek

Penjaminan kualitas pada proyek ini difokuskan pada penerapan validasi data yang ketat serta pengujian berkelanjutan terhadap fungsionalitas sistem. Salah satu pendekatan utama dalam menjaga kualitas aplikasi adalah penerapan middleware validasi menggunakan express-validator, yang diterapkan pada endpoint registrasi, login, dan reservasi. Validasi ini memastikan bahwa setiap data yang masuk ke sistem telah sesuai dengan format, batasan panjang karakter, dan logika bisnis yang ditentukan, seperti validasi email, panjang password minimal enam karakter, nomor telepon yang sesuai standar Indonesia, serta tanggal reservasi yang tidak boleh berada di masa lampau.

Selain validasi input, penjaminan kualitas juga dilaksanakan melalui pengujian manual dengan pendekatan black-box testing untuk memastikan bahwa setiap fitur berjalan sesuai dengan fungsinya. Tim pengembang melakukan pengecekan dari sisi klien dan server untuk menghindari kesalahan komunikasi data. Pengujian dilakukan secara iteratif setiap kali terdapat perubahan fungsi atau perbaikan bug, untuk meminimalkan risiko terjadinya regresi.

Penggunaan kontrol versi dengan Git juga membantu dalam memantau setiap perubahan yang terjadi pada proyek, serta memudahkan rollback jika ditemukan kesalahan. Dengan demikian, kualitas aplikasi tetap terjaga sepanjang siklus pengembangan, mulai dari tahap implementasi hingga pengujian dan deployment akhir.

E. Keberlanjutan Proyek

Keberlanjutan proyek sistem manajemen kost ini dirancang dengan mempertimbangkan aspek teknis, operasional, dan pengembangan jangka panjang agar sistem dapat terus berfungsi dan berkembang sesuai kebutuhan pengguna. Beberapa strategi keberlanjutan meliputi:

1) Pemeliharaan Sistem (Maintenance)

- Pemeliharaan Korektif : Setiap bug atau kesalahan yang muncul akan segera diperbaiki berdasarkan laporan pengguna atau hasil monitoring.
- Pemeliharaan Adaptif : Sistem akan disesuaikan jika ada perubahan teknologi, seperti pembaruan pada versi Node.js, MySQL, atau dependensi lain.
- Pemeliharaan Preventif : Pengecekan berkala terhadap performa server dan database untuk menghindari gangguan.

BAB IV
PENUTUPAN

A. Kesimpulan

Proyek pembangunan sistem manajemen kost berbasis web ini bertujuan untuk memberikan solusi digital bagi pemilik dan penyewa kost dalam mengelola data kamar, reservasi, serta informasi pengguna secara efisien. Dengan menggunakan arsitektur berbasis Node.js untuk backend dan MySQL sebagai basis data, sistem ini berhasil menyediakan fitur utama seperti manajemen data kost, registrasi pengguna, autentikasi, dan pencatatan transaksi. Proyek ini juga mengadopsi prinsip-prinsip manajemen proyek yang sistematis, mulai dari tahap perencanaan, pelaksanaan, pemantauan, hingga penjaminan kualitas dan keberlanjutan. Berdasarkan hasil implementasi dan pengujian, sistem telah berjalan sesuai dengan kebutuhan pengguna dan menunjukkan performa yang stabil.

B. Saran

Untuk pengembangan selanjutnya, beberapa hal yang dapat dipertimbangkan dalam proyek ini adalah:

1) Integrasi Payment Gateway : Menambahkan fitur pembayaran online untuk memudahkan proses transaksi sewa kamar.
2) Pengembangan Aplikasi Mobile : Mengembangkan versi mobile berbasis Android/iOS untuk meningkatkan aksesibilitas pengguna.
3) Monitoring dan Logging : Mengintegrasikan sistem log dan monitoring untuk mendeteksi kesalahan lebih dini dan menjaga performa sistem.
4) Penguatan Keamanan : Menerapkan enkripsi lebih lanjut pada data pengguna dan memperkuat proteksi terhadap serangan seperti SQL Injection atau XSS.
5) Uji Coba dengan Pengguna Nyata : Melakukan uji coba sistem dengan pemilik kost dan penyewa secara langsung untuk mendapatkan umpan balik nyata yang bisa dijadikan dasar perbaikan dan inovasi.

Dengan penerapan saran-saran tersebut, diharapkan sistem ini dapat terus berkembang dan memberikan manfaat yang lebih besar, serta menjadi solusi digital yang handal dan berkelanjutan dalam pengelolaan kost.

LAMPIRAN

A. Proposal Proyek

PROPOSAL PROYEK
PENGANTAR MANAJEMEN DAN PRINSIP PROYEK

"Proyek Pengembangan Sistem Informasi Penyewaan Kost"

Disusun Oleh:

Faishal Faruq Nawawi	2200018012
Aqief Idlan Hakimi	2200018051
Rafif Ghani Widiandhi	2200018053
Muhammad Dzaka Alfikri	2200018057
Farrel Zacky A. R	2200018061

UNIVERSITAS AHMAD DAHLAN
FAKULTAS TEKNOLOGI INDUSTRI
PROGRAM STUDI TEKNIK INFORMATIKA

2024

Lembaran Pengesahan

Untuk memenuhi kebutuhan pengembangan sistem berbasis digital untuk meningkatkan efisiensi dan daya saing bisnis di era modern, proposal ini berjudul:

"Proyek Pengembangan Sistem Informasi Penyewaan Kost"

Proposal ini telah disusun oleh:

Faishal Faruq Nawawi	2200018012
Aqief Idlan Hakimi	2200018051
Rafif Ghani Widiandhi	2200018053
Muhammad Dzaka Alfikri	2200018057
Farrel Zacky A. R	2200018061

Program Studi : Teknik Informatika

Fakultas : Fakultas Teknologi Industri

Universitas : Universitas Ahmad Dahlan Yogyakarta

Dengan demikian, kami menyatakan bahwa proposal ini merupakan hasil dari pemikiran dan ide yang telah kami ciptakan sesuai dengan tujuan proyek. Kami berharap proposal ini akan menawarkan solusi praktis dan manfaat untuk mendukung pertumbuhan bisnis kost melalui media digital.

Yogyakarta, 1 Januari 2025

Project Manager	Pemilik Usaha Kost
Faishal Faruq Nawawi	Rahayu Astuti, S.Kep, Ners.
2200018012	

I. Latar Belakang Permasalahan

Kebutuhan akan hunian sementara seperti kost semakin meningkat, terutama di daerah perkotaan dan pusat pendidikan. Mahasiswa, pekerja, dan pendatang yang membutuhkan tempat tinggal seringkali menghadapi kesulitan menemukan kost yang sesuai dengan preferensi dan anggaran mereka. Permasalahan ini semakin kompleks karena kurangnya transparansi informasi tentang fasilitas, harga, dan ketersediaan kost, serta sulitnya mencapai lokasi tertentu tanpa kunjungan langsung.

Mengandalkan iklan fisik atau rekomendasi dari mulut ke mulut untuk mencari kost seringkali memakan waktu dan tidak efisien. Sebagai pemilik kost, Anda juga menghadapi masalah tambahan seperti tidak dapat menghubungi banyak calon penyewa.

Dengan perkembangan teknologi, pengembangan sistem informasi penyewaan kost berbasis website menjadi solusi modern yang efektif. Website ini dirancang untuk mempermudah penyewa dalam menemukan kost dengan informasi yang jelas dan lengkap terkait fasilitas, harga, lokasi, dan ketersediaan. Sistem ini memberikan keuntungan dalam mengelola data kost secara terpusat, menjangkau calon penyewa secara lebih luas, dan meningkatkan efisiensi proses pemasaran.

II. Tujuan

Proyek ini bertujuan untuk merancang dan mengembangkan sebuah website penyewaan kost yang intuitif, informatif, dan efisien. Dengan adanya sistem ini, diharapkan dapat meningkatkan kemudahan penyewaan kost, memaksimalkan pendapatan, serta menciptakan hubungan yang lebih baik antara pemilik kost dan penyewa.

Dengan ini, website yang menyediakan informasi lengkap dan akurat tentang kost milik pribadi, termasuk fasilitas, harga, lokasi, dan ketersediaan.

Memberikan pemilik kost kemampuan untuk mengelola data kost secara efisien, termasuk pembaruan informasi dan pengelolaan ketersediaan.

III. Ruang Lingkup Proyek

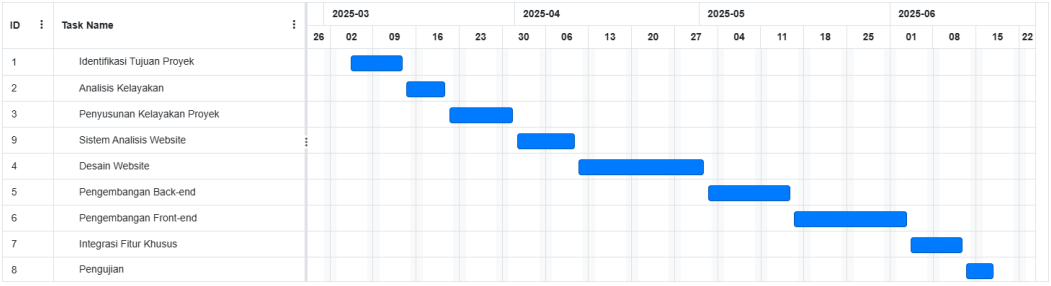
Project Title	Kost Putri MEWAH Patemon	
Project Manager Faisal Faruq Nawawi, +62 821 5876 3410		Project Date 02 October 2024
Project Objective	Proyek ini merupakan bidang Real Estate (Properti tak bergerak) dengan sasaran yang ditujukan untuk individu yang mencari tempat tinggal bersifat sementara. Tujuan dari proyek ini adalah menyediakan sarana visual untuk menunjukkan sarana dan prasarana yang dimiliki oleh client serta memberikan sistem yang lebih baik dalam melakukan transaksi sewa properti yang dimiliki oleh client. Sistem yang diberikan memiliki kelebihan dalam kemudahan manajemen keuangan dengan antarmuka yang minimalis namun disertai fitur yang lengkap.	
Deliverables	Dalam proposal proyek ini, tim proyek berkomitmen untuk menyediakan deliverables yang akan mendukung kesuksesan manajemen property client. 1. Tim proyek akan mengembangkan sistem manajemen sewa properti yang intuitif dan mudah digunakan, dilengkapi dengan fitur manajemen keuangan yang lengkap untuk memudahkan pengelolaan aset. 2. Kami akan menghadirkan presentasi visual yang menarik untuk memperlihatkan fasilitas properti, dengan tujuan menarik lebih banyak calon penyewa. 3. Tim proyek akan mengelola website yang memungkinkan pengguna untuk melakukan transaksi dengan mudah dan cepat, serta menciptakan mekanisme umpan balik dari pengguna dan admin untuk memastikan operasional berjalan optimal. 4. Strategi pemasaran digital yang terarah juga akan dirancang oleh tim proyek untuk menarik segmen penyewa sementara seperti mahasiswa, dokter, dan pekerja. Tim proyek menargetkan penyelesaian proyek ini dalam jangka waktu 4-6 bulan, dengan jaminan kompensasi apabila terjadi keterlambatan pengerjaan.	

IV. Kebutuhan SDM

Name	Role	Responsibilities
Faisal Faruq Nawawi	Project Manager	Pemimpin Proyek
Aqief Idlan Hakimi	Project Team	Desainer UI/UX
Muhammad Dzaka Alfikri	Project Team	Back-end Developer
Rafif Ghani Widiandhi	Project Team	Perancang Sistem
Farrel Zacky A. R	Project Team	Front-end Developer

	Stakeholder				
	Faisal Faruq	Aqief I Hakimi	M. Dzaka A	Rafif Ghani W	Farrel Zacky
Jabatan	Project Manager	Project Team	Project Team	Project Team	Project Team
Peran	Penguasa Proyek	Desainer UI/UX	Back-end Developer	Perancang Sistem	Front-end Developer
Fakta	Memiliki ketrampilan kepemimpinan yang baik	Ketrampilan manajemen waktu dan kreatifitas tinggi	Ahli dalam logika pemrograman	Ahli dalam perancangan sistem aplikasi yang terstruktur	Ahli dalam layouting website

V. Gantt Chart



B. MOU

Surat Perjanjian Kerjasama

Pada hari ini, Minggu, bertempat di Meton Kulon, Patemon, Kec. Gombong, Kabupaten Kebumen, telah disepakati Surat Perjanjian Kerjasama antara:

Nama : Rahayu Astuti, S.Kep, Ners.

NIK 3305191610680001

Alamat : Gombong No.275 06/01, Meton Kulon, Patemon, Kec. Gombong, Kabupaten Kebumen, Jawa Tengah 54416

Bertindak sebagai pemilik kost putri, yang selanjutnya dalam perjanjian ini disebut sebagai **Pihak Pertama**.

Nama : Faisal Faruq Nawawi

NIK 6471050206040003

Alamat : Winong KG II/307

Bertindak sebagai ketua tim pengembang website, yang mewakili timnya sesuai dengan Memorandum of Understanding (MoU) sebelumnya, yang selanjutnya dalam perjanjian ini disebut sebagai **Pihak Kedua**.

Kedua belah pihak sepakat untuk mengadakan kerjasama dalam pengembangan website penyewaan kost berbasis online dengan ketentuan sebagai berikut:

Pasal 1 Tujuan

Tujuan Kerjasama ini bertujuan untuk merancang dan mengembangkan website penyewaan kost yang intuitif, informatif, dan efisien guna:

1. Mempermudah penyewaan kost secara online.
2. Meningkatkan pendapatan Pihak Pertama.
3. Memberikan informasi lengkap dan akurat kepada calon penyewa.

Pasal 2 Ruang Lingkup

Ruang Lingkup Pekerjaan Pihak Kedua akan:

1. Membuat website dengan fitur informasi kost, pengelolaan ketersediaan, dan fitur pencarian.
2. Menyediakan pelatihan kepada Pihak Pertama untuk pengelolaan mandiri.
3. Memberikan dukungan teknis selama masa garansi.

Pasal 3 Hak dan Kewajiban

3.1 Hak Pihak Pertama:

1. Memiliki website yang berfungsi sesuai dengan spesifikasi yang disepakati.
2. Mendapatkan pelatihan untuk pengelolaan website.

3.2 Kewajiban Pihak Pertama:

1. Memberikan informasi lengkap terkait kost untuk dimuat di website.
2. Membayar biaya pengembangan sesuai dengan kesepakatan.

3.3 Hak Pihak Kedua:

1. Mendapatkan pembayaran sesuai dengan tahapan yang telah ditentukan.
2. Mendapatkan umpan balik selama pengembangan.

3.4 Kewajiban Pihak Kedua:

1. Menyelesaikan proyek sesuai dengan spesifikasi, tenggat waktu, dan kesepakatan.
2. Memberikan dukungan teknis selama masa garansi.

Pasal 4 Jangka Waktu

Jangka Waktu Perjanjian ini berlaku selama 4 (empat) bulan sejak tanggal penandatanganan, yaitu terhitung mulai tanggal 3 Maret 2025 sampai dengan 20 Juli 2025. Proyek harus selesai dalam jangka waktu tersebut dengan pembagian tahapan sebagai berikut:

1. Tahap 1: Penyelesaian desain (Maret 2025).
2. Tahap 2: Pengembangan sistem (April – Mei 2025).
3. Tahap 3: Pengujian dan serah terima (Juni – Pertengahan Juli 2025).

Pasal 5 Biaya dan Pembayaran

1. Total biaya pengembangan adalah Rp 1.900.000.
2. Pembayaran dilakukan dalam tiga tahap:
    a. RP 900.000 saat penandatanganan perjanjian.
    b. Rp 1.000.000 setelah penyelesaian tahap desain.
    c. Pelunasan setelah penyelesaian proyek dan serah terima.

Pasal 6 Kerahasiaan

Kerahasiaan Kedua belah pihak sepakat untuk menjaga kerahasiaan informasi terkait proyek ini, kecuali jika informasi tersebut dibutuhkan untuk pelaksanaan pekerjaan.

Pasal 7 Penyelesaian Perselisihan

Penyelesaian Sengketa Apabila terjadi perselisihan, kedua belah pihak sepakat untuk menyelesaikannya secara musyawarah. Jika tidak mencapai kesepakatan, maka sengketa akan diselesaikan sesuai hukum.

Pasal 8 Penutup

Perjanjian ini mengikat secara hukum setelah ditandatàngani oleh kedua belah pihak. Perubahan perjanjian harus disepakati secara tertulis oleh kedua belah pihak.

Demikian Surat perjanjian ini dibuat dengan sebenar-benarnya dan untuk dapat dipergunakan sebagaimana mestinya.

Yogyakarta, ...

Pihak Pertama	**Pihak Kedua**
Rahayu Astuti	Faishal Faruq Nawawi
(.....)	(.....)

C. Logbook Kelompok

LOG BOOK MANAJEMEN PROYEK TEKNOLOGI INFORMASI MAHASISWA

PROGRAM STUDI S1 INFORMATIKA UNIVERSITAS AHMAD DAHLAN T.A 2024/2025

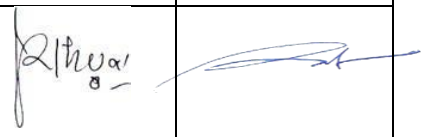
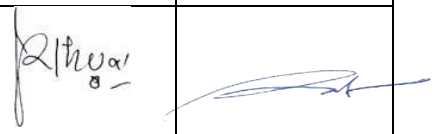
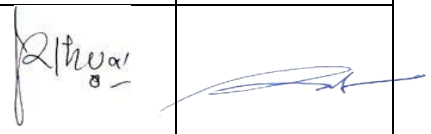
Nama Kelompok : Aaliyyah

No	NIM	Nama
1	2200018012	Faishal Faruq Nawawi
2	2200018051	Aqief Idlan Hakimi
3	2200018053	Rafif Ghani Widiandhi
4	2200018057	Muhammad Dzaka Alfikri
5	2200018061	Farrel Zacky A. R

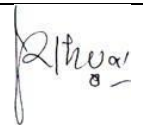
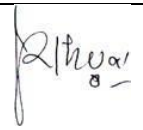
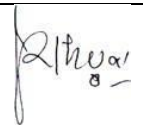
Judul Manajemen Tugas Proyek : Sistem Informasi Penyewaan Kost

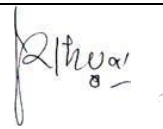
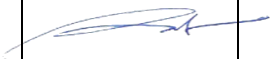
Dosen Pembimbing : Bambang Robi'in, S.T., M.T.

Logbook Minggu 4 sd 7 (Sebelum UTS)

No	Kegiatan	Waktu Pelaksanaan		Hasil	Kendala, Rencana Perubahan (Jika Ada)	Paraf MITRA / CLIENT	Paraf Dosen Pembimbing MPTI
		Hari / TGL	Durasi (Jam)				
1	Pembahasan mengenai website seperti apa yang ingin kita kembangkan	12 Maret 2025	1 jam	Fitur utama website kost dan perencanaan desain untuk website kost	-		
2	Rapat mengenai proposal dan studi kebutuhan pengguna	19 Maret 2025	2 jam	Proposal, MOU, dan gambaran untuk proyek tersedia pada draft	-		
3	Perancangan ERD, Use Case diagram, sistem bisnis	26 Maret 2025	3 jam	ERD & Use Case selesai dengan entitas pengguna, kamar, fasilitas, ulasan, foto	-		
4	Penerapan UI/UX berbasis web dan pembuatan desain logo	16 April 2025	5 jam	Antarmuka yang menarik tertera pada figma dan desain logo disetujui	-		

Logbook Minggu 9 sd 11 (Setelah UTS)

No	Kegiatan	Waktu Pelaksanaan		Hasil	Kendala, Rencana Perubahan (Jika Ada)	Paraf MITRA / CLIENT	Paraf Dosen Pembimbing MPTI
		Hari / TGL	Durasi (Jam)				
1	Mulai implementasi front-end (HTML, CSS, JavaScript)	19 Juni 2025	4 jam	Halaman beranda dan halaman kamar berhasil dibangun sesuai desain. Komponen form, validasi dasar, dan layout responsif sudah berfungsi.	-		
2	Mulai pengembangan back-end dan Integrasi backend dengan front-end	26 Juni 2025	4 jam	Halaman reservasi dan data kamar berhasil ditampilkan dan dikirim ke database.	-		
3	Validasi input dan koneksi database	3 Juli 2025	4 jam	Sistem dapat menyimpan data penyewa dan reservasi ke database secara stabil.	-		

4	Proses finalisasi dan perapian tampilan. Deployment ke layanan hosting.	17 April 2025	4 jam	Konsistensi warna, spasi, dan struktur halaman disempurnakan. Website diakses secara publik.	-		

Catatan Pembimbing Lapangan/Dosen Pembimbing MPTI/Dosen Pengampu Kelas MPTI :

……
……
……
……
……
……

Yogyakarta, ... 2025

Dosen Pengampu Kelas MPTI

Ketua MPTI

……..

NIY. …….

(Faishal Faruq Nawawi

D. Logbook Individu

LOG BOOK MANAJEMEN PROYEK TEKNOLOGI INFORMASI MAHASISWA

PROGRAM STUDI S1 INFORMATIKA UNIVERSITAS AHMAD DAHLAN T.A 2024/2025

Nama Kelompok : Aaliyyah			
	No	NIM	Nama
	1	2200018012	Faishal Faruq Nawawi
Sebagai : Ketua Tim			
Judul Manajemen Tugas Proyek : Sistem Informasi Penyewaan Kost			
Dosen Pembimbing : Bambang Robi'in, S.T., M.T.			

Logbook Individu

No	Kegiatan	Waktu Pelaksanaan		Hasil	Kendala, Rencana Perubahan (Jika Ada)
		Hari / TGL	Durasi (Jam)		

1	Memimpin diskusi awal tim tentang arah dan fitur utama website kost	12 Maret 2025	1 jam	Arahan awal proyek ditentukan dan fitur utama website	-
2	Mendistribusikan tugas pengumpulan kebutuhan pengguna & melengkapi proposal	19 Maret 2025	2 Jam	Proposal, MOU, dan gambaran untuk proyek tersedia pada draft	-
3	Membantu sistem analis dalam perancangan ERD & Use Case diagram	26 Maret 2025	2 Jam	ERD & Use Case selesai dengan entitas pengguna, kamar, fasilitas, ulasan, foto	-
4	Melanjutkan perancangan ERD & Use Case, Memastikan desain logo sesuai target	16 April 2025	3 Jam	Pembuatan logo diterima client	-
5	Membantu UI/UX dalam tampilan home UI/UX Berbasis Web	23 April 2025	5 jam	Antarmuka yang menarik tertera pada figma	-
6	Evaluasi progress UI/UX dan sistem	12 Juni 2025	2 jam	Progres desain dan analisis sistem sudah baik	-
7	Koordinasi integrasi frontend dengan backend	10 Juli 2025	4 jam	Integrasi berjalan lancar sesuai arsitektur sistem	-

8	Menyelesaikan laporan dan uji coba akhir beserta verifikasi	24 Juli 2025	4 jam	Proyek siap untuk deployment dan presentasi	-

LOG BOOK MANAJEMEN PROYEK TEKNOLOGI INFORMASI MAHASISWA

PROGRAM STUDI S1 INFORMATIKA UNIVERSITAS AHMAD DAHLAN T.A 2024/2025

Nama Kelompok : Aaliyyah			
	No	NIM	Nama
	1	2200018053	Rafif Ghani Widiandhi
Sebagai : Sistem Analisis			
Judul Manajemen Tugas Proyek : Sistem Informasi Penyewaan Kost			
Dosen Pembimbing : Bambang Robi'in, S.T., M.T.			

Logbook Individu

No	Kegiatan	Waktu Pelaksanaan	Hasil	Kendala, Rencana Perubahan (Jika Ada)

		Hari / TGL	Durasi (Jam)		
1	Mengikuti diskusi awal tim dan mengajukan ide tentang arah dan fitur utama website kost	12 Maret 2025	1 jam	Arah proyek ditentukan dan fitur utama website	-
2	Melaksanakan tugas pengumpulan kebutuhan pengguna serta berkontribusi dalam penyusunan dan pelengkapan proposal.	19 Maret 2025	2 Jam	Proposal, MOU, dan gambaran untuk proyek tersedia pada draft	-
3	Membuat sistem analis dalam perancangan ERD & Use Case diagram	26 Maret 2025	2 Jam	ERD & Use Case selesai dengan entitas pengguna, kamar, fasilitas, ulasan, foto	-
4	Membantu UI/UX dalam tampilan login UI/UX Berbasis Web	16 April 2025	3 Jam	Antarmuka yang menarik tertera pada figma	-
5	Membantu pengembangan FrontEnd dalam penerapan tampilan UI/UX untuk bagian Contact Us Berbasis Web	23 April 2025	6 Jam	Tampilan halaman Login dan Header Homepage yang sesuai dengan desain UI/UX yang diinginkan	-

6	Pemilihan teknologi stack bersama tim	12 Juni 2025	2 jam	Stack disepakati adalah, Node.js, Express, MySQL	-
7	Verifikasi rancangan sistem dengan project manager dan developer	19 Juni 2025	4 jam	Sistem siap diimplementasikan oleh tim frontend dan backend	-
8	Melakukan testing pada sistem	24 Juli 2025	4 jam	Menggunakan blackbox dan SUS dalam pembuatan testing	-

LOG BOOK MANAJEMEN PROYEK TEKNOLOGI INFORMASI MAHASISWA

PROGRAM STUDI S1 INFORMATIKA UNIVERSITAS AHMAD DAHLAN T.A 2024/2025

Nama Kelompok: Aaliyyah			
	No	NIM	Nama
	1	2200018051	Aqief Idlan Hakimi
Sebagai : UI/UX Designer			
Judul Manajemen Tugas Proyek: Sistem Informasi Penyewaan Kost			

Dosen Pembimbing: Bambang Robi'in, S.T., M.T.

Logbook Individu

No	Kegiatan	Waktu Pelaksanaan		Hasil	Kendala, Rencana Perubahan (Jika Ada)
		Hari / TGL	Durasi (Jam)		
1	Mengikuti arahan diskusi yang disampaikan ketua kelompok dalam membahas tugas dan arah berjalannya proyek.	12 Maret 2025	1 jam	Penerimaan tugas sesuai jobdesk sebagai front-end developer.	-
2	Menerima dan mengerjakan tugas terkait sesuai dengan arahan ketua kelompok. Dalam kasus ini, saya membantu mengerjakan salah satu dari diagram yang tersaji di proposal.	19 Maret 2025	1 Jam	Proposal, MOU, dan gambaran untuk proyek tersedia pada draft.	-
3	Merancang UI menggunakan aplikasi figma sesuai dengan kriteria yang diberikan oleh klien	26 Maret 2025	2 Jam	Hasil kasar UI dari rancangan kriteria klien.	-

4	Membuat logo berdasarkan kriteria yang diberikan oleh klien.	16 April 2025	3 Jam	Proses bisnis & logo telah selesai dibuat	-
5	Membuat dan membantu rancangan UI/UX dalam tampilan home UI/UX Berbasis Web menggunakan aplikasi figma.	23 April 2025	5 Jam	Antarmuka yang menarik tertera pada figma	-
6	Menyelesaikan desain dan tampilan interaktif	19 Juni 2025	4 jam	Antarmuka selesai dan siap diimplementasi ke frontend	-
7	Revisi berdasarkan feedback dari tim dev dan pengguna	26 Juni 2025	4 jam	Desain final siap diimplementasi ke frontend	-
8	Dokumentasi guideline desain UI	3 Juli 2025	2 jam	Frontend memiliki panduan style yang konsisten	-

Nama Kelompok : Aaliyyah			
	No	NIM	Nama
	1	2200018061	Farrel Zacky As Syahid Ramadhani
Sebagai : Front-End Developer			
Judul Manajemen Tugas Proyek : Sistem Informasi Penyewaan Kost			
Dosen Pembimbing : Bambang Robi'in, S.T., M.T.			

Logbook Individu

No	Kegiatan	Waktu Pelaksanaan		Hasil	Kendala, Rencana Perubahan (Jika Ada)
		Hari / TGL	Durasi (Jam)		
1	Mengikuti diskusi awal tim dan mengembangkan gagasan tentang tujuan dan fitur utama website kost	12 Maret 2025	1 jam	Menerima arahan proyek yang sudah ditetapkan dan fitur utama website.	-

2	Di arah pemimpin kelompok, mengambil dan melakukan tugas - tugas terkait. Dalam hal ini, saya membantu dengan salah satu diagram yang tersaji dalam proposal.	19 Maret 2025	2 Jam	Proposal, MOU, dan gambaran untuk proyek tersedia pada draft.	-
3	Membantu sistem analis dalam perancangan diagram system bisnis dan ERD	26 Maret 2025	2 Jam	System bisnis dan ERD selesai dengan entitas pengguna, kamar, fasilitas, ulasan, foto	-
4	Membantu UI/UX dalam tampilan Beberapa menu UI/UX Berbasis Web	16 April 2025	4 Jam	Antarmuka yang menarik tertera pada aplikasi figma	-
5	Membantu pengembangan Front-End dalam penerapan tampilan UI/UX untuk Login Page dan Header di Homepage Berbasis Web	23 April 2025	6 Jam	Tampilan halaman Login dan Header Homepage yang sesuai dengan desain UI/UX yang diinginkan	-
6	Implementasi halaman beranda, kamar, reservasi, dan validasi form	26 Juni 2025	4 jam	Tampilan responsive dan navigasi berfungsi. Reservasi dapat digunakan dengan validasi input	-
7	Implementasi halaman akun penyewa dan tampilan pembayaran	10 Juli 2025	4 jam	Penyewa dapat melihat status dan riwayat bayar	-

8	Testing UI + integrasi awal backend	17 Juli 2025	2 jam	Data tersambung dengan server backend	-

LOG BOOK MANAJEMEN PROYEK TEKNOLOGI INFORMASI MAHASISWA

PROGRAM STUDI S1 INFORMATIKA UNIVERSITAS AHMAD DAHLAN T.A 2024/2025

Nama Kelompok : Aaliyyah			
	No	NIM	Nama
	1	2200018057	M. Dzaka Al Fikri
Sebagai : Back-End Developer			
Judul Manajemen Tugas Proyek : Sistem Informasi Penyewaan Kost			
Dosen Pembimbing : Bambang Robi'in, S.T., M.T.			

Logbook Individu

No	Kegiatan	Waktu Pelaksanaan	Hasil	Kendala, Rencana Perubahan (Jika Ada)

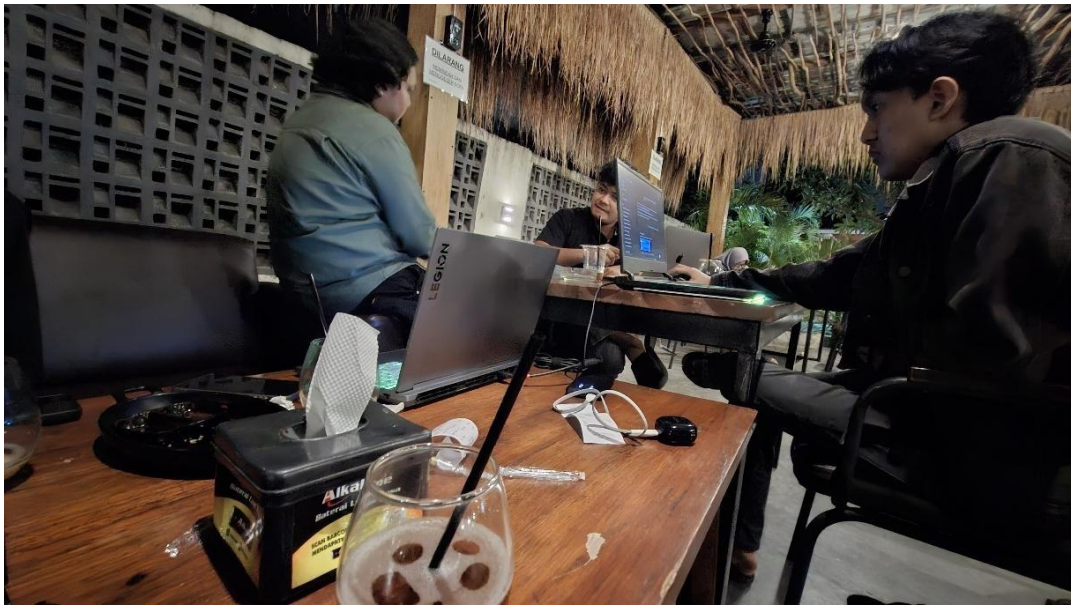
		Hari / TGL	Durasi (Jam)		
1	Mengikuti diskusi awal tim dan mengajukan ide tentang arah dan fitur utama website kost	12 Maret 2025	1 jam	Arah proyek ditentukan dan fitur utama website	-
2	Melaksanakan tugas pengumpulan kebutuhan pengguna serta berkontribusi dalam penyusunan dan pelengkapan proposal.	19 Maret 2025	2 Jam	Proposal, MOU, dan gambaran untuk proyek tersedia pada draft	-
3	Membantu sistem analis dalam perancangan ERD & Use Case diagram	26 Maret 2025	2 Jam	ERD & Use Case selesai dengan entitas pengguna, kamar, fasilitas, ulasan, foto	-
4	Membantu UI/UX dalam tampilan login UI/UX Berbasis Web	16 April 2025	3 Jam	Antarmuka yang menarik tertera pada figma	-

5	Membantu pengembangan FrontEnd dalam penerapan tampilan UI/UX untuk LoginPage dan Header di Homepage Berbasis Web	23 April 2025	6 Jam	Tampilan halaman Login dan Header Homepage yang sesuai dengan desain UI/UX yang diinginkan	-
6	Setup project backend (Node.js, Express)	26 Juni 2025	2 jam	Struktur folder backend dan koneksi DB sudah siap	-
7	Pembuatan API reservasi, kamar, dan login. Middleware validasi dan keamanan	17 Juli 2025	4 jam	Data berhasil ditampilkan dan disimpan. Validasi input aktif, proteksi endpoint dengan autentikasi	-
8	Finalisasi backend & koneksi MySQL	24 Juli 2025	4 jam	Backend stabil dan siap dideploy	-

E. Dokumentasi

Gambar 4. 1 Dokumentasi Rapat Online

Gambar 4. 2 Dokumentasi Rapat Offine



F. Berita Acara

BERITA ACARA SERAH TERIMA WEBSITE SISTEM INFORMASI KOST

Pada hari ini HARI, Tanggal TANGGAL yang bertanda tangan di bawah ini :

I.	Nama	:	Rahayu Astuti, S.Kep, Ners.
	NIK	:	3305191610680001
	Alamat	:	Gombong No.275 06/01, Meton Kulon, Patemon, Kec. Gombong, Kabupaten Kebumen, Jawa Tengah 54416 ***Disebut PIHAK KESATU (I)***
II.	Nama	:	Faishal Faruq Nawawi
	NIK	:	6471050206040003
	Alamat	:	Winong KG II/307
			Disebut PIHAK KEDUA (II)

Dengan ini **PIHAK KEDUA** menyerahkan dokumen kepada **PIHAK KESATU** dan **PIHAK KEDUA** barang berupa :

1. Source Code Website Sistem Informasi Kost
2. Website Yang Telah Di Hosting

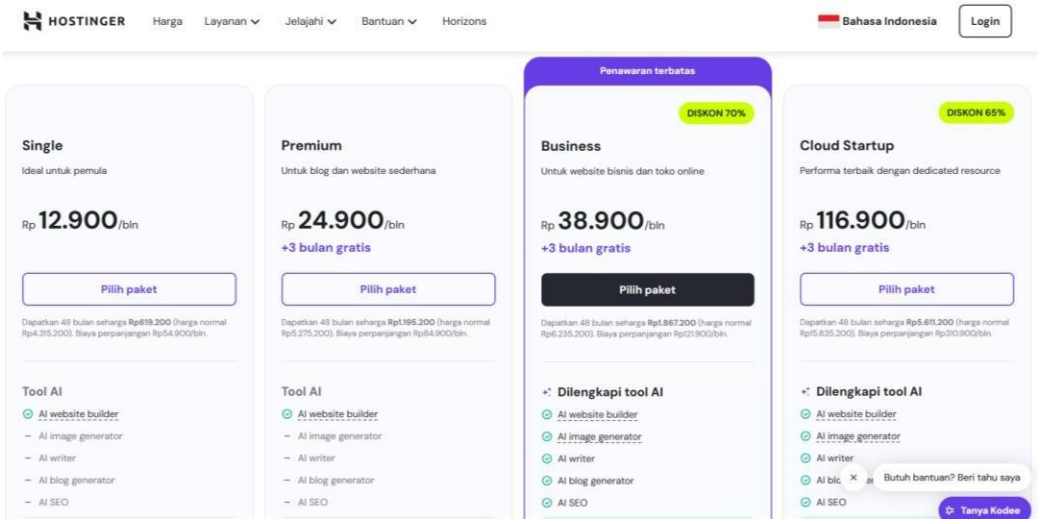
Demikian berita acara serah terima ini dibuat dan ditanda tangani bersama kedua belah pihak untuk dapaat dipergunakan sebagaimana mestinya.

Dibuat di Yogyakarta
Tanggal : 28 Juli 2025

Yang Menerima	Yang Menyerahkan
PIHAK KESATU	**PIHAK KEDUA**
Rahayu Astuti, S.Kep, Ners.	**Faishal Faruq Nawawi**

G. Bukti Pembiayaan

Gambar 4. 3 Tujuan Hosting



H. Source Code

Berikut adalah full code terlampir pada GitHub:

https://github.com/kuahcurry/kost-patemon.git

Tabel 4. 1 Codingan home.html

	Home.html

```
<meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
   <title>Home - Kost Mewah Patemon Gombong</title>
   <link rel="icon" href="../image/logo.png" />
   <link

href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,10
0..900;1,100..900&family=Roboto:ital,wght@0,100..900;1,100..900&display
=swap"
     rel="stylesheet"
   />
   <link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/swiper@11/swiper-bundle.min.css"
   />
   <link rel="stylesheet" type="text/css" href="../css/home.css" />
   <link
    rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.2/css/all.min.css"
   />
<body>
   <header class="header">
    <div class="logo">
     <a href="home.html" class="logo-link">
      <div class="circle-container">
       <div class="circle">
        <img
         src="../Image/logo.png"
         alt="Kost Mewah Patemon Gombong Logo"
        />
       </div>
      </div>
     </a>
    </div>
    <div class="title">
     <a href="home.html" class="title-link">
```

```
        <img src="../Image/logoTeks.png" alt="Title Image" />
      </a>
    </div>
    <div class="menu">
      <nav>
        <nav class="menu-list">
          <li class="active"><a href="home.html">MENU</a></li>
          <li class="border"></li>
          <li><a href="kamar.html">KAMAR</a></li>
          <li class="border"></li>
          <li><a href="akun.html">AKUN</a></li>
        </nav>
      </nav>
    </div>
  </header>
  <main>
    <div class="content-1">
      <div class="left-section">
        <div class="left-section-content">
          <h2>SELAMAT DATANG DI KOST MEWAH!</h2>
          <p>
            Hunian nyaman, aman, dan strategis untuk Anda yang mencari
tempat
            tinggal terbaik. Yuk temukan tempat tinggal nyamanmu hanya dalam
            beberapa klik!
          </p>
          <a href="#content-2" class="btn-secondary">LIHAT SELENGKAPNYA
→</a>
        </div>
      </div>
      <div class="right-section">
        <img src="../Image/kos.png" alt="Building exterior of Kost Mewah" />
      </div>
    </div>

    <div class="content-2" id="content-2">
      <h2>KEUNGGULAN KOS KAMI</h2>
      <div class="feature">
        <swiper-container
          class="mySwiper"
          pagination="true"
          pagination-clickable="false"
          space-between="30"
          slides-per-view="3"
        >
```

```
          <swiper-slide>
            <img src="../image/parkiran1.png" />
            <h3>Parkir Motor dan Mobil Luas</h3>
            <p>
              Parkiran kendaraan dengan ruang yang luas untuk lebih dari lima
              motor dan tiga mobil dengan halaman yang melegakan.
            </p>
          </swiper-slide>
          <swiper-slide>
            <img src="../image/kamar_mandi1.png" />
            <h3>Air Conditioner dan WIFI</h3>
            <p>
              Air Conditioner dengan ukuran 1/2 PK yang dapat mendinginkan
              satu kamar dengan cepat dan WIFI dengan kecepatan yang cukup
              untuk kebutuhan Anda sehari-hari.
            </p>
          </swiper-slide>
          <swiper-slide>
            <img src="../image/dummy.png" />
            <h3>Servis Laptop dan Komputer</h3>
            <p>
              Servis Laptop dan Komputer yang dapat dihubungi dengan nomor
              yang sama dengan admin kos, memudahkan Anda untuk
memperbaiki PC
              Anda!
            </p>
          </swiper-slide>
          <swiper-slide>
            <img src="../image/dummy.png" />
            <h3>Keamanan 24/7</h3>
            <p>
              Kos kami dilengkapi dengan CCTV untuk setiap sudut parkiran,
              dengan pantauan 24 Jam selama seminggu.
            </p>
          </swiper-slide>
          <swiper-slide>
            <img src="../image/dummy.png" />
            <h3>Lokasi Strategis</h3>
            <p>
              Kos dengan jarak yang relatif dekat dengan RSU Palang Biru,
              Alun-Alun Manunggal, Pasar Gombong, Minimarket Alfamart, RS
PKU
              Muhammadiyah, Cafe.
            </p>
          </swiper-slide>
```

```
          <swiper-slide>
            <img src="../image/dummy.png" />
            <h3>Pemandangan Memukau</h3>
            <p>
              Dengan hamparan sawah sepanjang mata memandang dan vibes
              pedesaan, pemandangan pagi dari kos MEWAH Putri Patemon
              memanjakan mata Anda terutama di pagi hari!
            </p>
          </swiper-slide>
        </swiper-container>
      </div>
    </div>
    <div class="content-3">
      <h2 class="review-title">ULASAN KOS</h2>
      <div class="swiper mySwiper">
        <div class="swiper-wrapper">
          <div class="swiper-slide">
            <div class="review-card-content">
              <div class="review-text">
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed
                tincidunt efficitur maximus. Nullam eget malesuada erat. Cras
                eu velit vitae purus rhoncus iaculis ut sit amet est.
                Pellentesque commodo tempor dui vitae pulvinar. Etiam pretium
                pharetra lectus, at ornare leo pellentesque et. Aliquam mauris
                leo, congue a arcu ut, interdum euismod erat. Ut magna nulla,
                aliquam et feugiat rhoncus, aliquet at orci. Etiam quis nisl
                eu tellus ornare aliquet. Sed porttitor, nulla at dictum
                lobortis, odio nisl commodo urna, at volutpat tellus arcu sed
                enim. Proin in quam ligula. Vestibulum quam nibh, condimentum
                vitae nisi nec, cursus accumsan elit. Nulla interdum quam
                mollis, scelerisque metus vel, pretium urna. Nam at purus nec
                nulla suscipit ornare nec eget eros.
              </div>
<div class="content-4">
      <h2>LOKASI KAMI</h2>
      <div class="map-container">
        <iframe

src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d1163.
741172622468!2d109.52734478612243!3d-
7.6118970434925!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0
x2e654ba47793dda1%3A0xc10b0c777099307a!2sKost%20Eksklusif%20%22
MeWah%22!5e1!3m2!1sen!2sid!4v1750948646724!5m2!1sen!2sid"
          width="1200"
          height="600"
```

```
          style="border: 0"
          allowfullscreen=""
          loading="lazy"
          referrerpolicy="no-referrer-when-downgrade"
        ></iframe>
      </div>
    </div>
  </main>
  <footer class="footer">
    <div class="footer-container">
      <div class="footer-menu">
        <div class="footer-menu-col">
          <a href="home.html">Menu</a>
          <a href="kamar.html">Kamar</a>
        </div>
        <div class="footer-menu-col">
          <a href="akun.html">Akun</a>
          <a href="login.html">Login</a>
        </div>
      </div>
      <div class="footer-divider"></div>
      <div class="footer-info">
        <div class="footer-title">Kost Putri MEWAH Patemon</div>
        <div class="footer-address">
          Jl. Serayu IV No. 13 03/01 Desa Patemon,<br />Kecamatan Gombong,
          Kabupaten Kebumen
        </div>
      </div>
    </div>
    <div class="footer-copyright">
      Copyright Kos MEWAH Patemon, Gombong 2025
    </div>
  </footer>
  <script src="https://cdn.jsdelivr.net/npm/swiper@11/swiper-element-
bundle.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/swiper@11/swiper-
bundle.min.js"></script>
  <script src="../js/auth.js"></script>
  <script>
    // Toast Message Functions
    function showToast(message, type = "success", duration = 4000) {
      // Remove existing toast if any
      const existingToast = document.querySelector(".toast");
      if (existingToast) {
        existingToast.remove();
```

```
    }

    // Create toast element
    const toast = document.createElement("div");
    toast.className = `toast ${type}`;

    // Set icon based on type
    let iconClass = "fa-check-circle";
    switch (type) {
      case "error":
        iconClass = "fa-exclamation-triangle";
        break;
      case "warning":
        iconClass = "fa-exclamation-circle";
        break;
      case "info":
        iconClass = "fa-info-circle";
        break;
      case "welcome":
        iconClass = "fa-user-check";
        break;
      default:
        iconClass = "fa-check-circle";
    }

    toast.innerHTML = `
      <div class="toast-icon">
        <i class="fas ${iconClass}"></i>
      </div>
      <div class="toast-message">${message}</div>
      <button class="toast-close" onclick="hideToast()">
        <i class="fas fa-times"></i>
      </button>
    `;

    // Add to body
    document.body.appendChild(toast);

    // Show toast with animation
    setTimeout(() => {
      toast.classList.add("show");
    }, 100);

    // Auto hide after specified duration
    setTimeout(() => {
```

```
      hideToast();
    }, duration);
  }

  // Function to show welcome toast
  function showWelcomeToast(message) {
  showToast(message, "welcome", 5000);
  }

  // Function to hide toast
  function hideToast() {
    const toast = document.querySelector(".toast");
    if (toast) {
      toast.classList.add("hide");
      setTimeout(() => {
        toast.remove();
      }, 400);
    }
  }

  // Function to show success toast
  function showSuccessToast(message) {
  showToast(message, "success");
  }

  // Function to show error toast
  function showErrorToast(message) {
  showToast(message, "error");
  }

  // Function to show warning toast
  function showWarningToast(message) {
  showToast(message, "warning");
  }

  // Function to show info toast
  function showInfoToast(message) {
  showToast(message, "info");
  }

  var swiper = new Swiper(".mySwiper", {
    effect: "cards",
    grabCursor: true,
  });
</script>
```

```
</body>
```

Tabel 4. 2 Codingan kamar.html

Kamar.html

```
<meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
  <title>Kamar - Kost Mewah Patemon Gombong</title>
  <link rel="icon" href="../image/logo.png" />
  <link

href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;
700;800&display=swap"
    rel="stylesheet"
  />
  <!-- Modern Bootstrap & Custom Style -->
  <link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.
css"
    rel="stylesheet"
  />
  <link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/swiper@11/swiper-bundle.min.css"
  />
<link
    rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.2/css/all.min.css"
  />
 </head>
 <body>
  <header class="header">
   <div
    class="logo"
    style="cursor: pointer"
    onclick="window.location.href='home.html'"
   >
    <div class="circle-container">
     <div class="circle">
      <img src="../Image/logo.png" alt="Kost Mewah Patemon Gombong"
/>
     </div>
```

```
    </div>
   </div>
   <div
    class="title"
    style="cursor: pointer"
    onclick="window.location.href='home.html'"
   >
    <img src="../Image/logoTeks.png" alt="Title Image" />
   </div>
   <div class="menu">
    <nav>
     <nav class="menu-list">
      <li><a href="home.html">MENU</a></li>
      <li class="border"></li>
      <li class="active"><a href="kamar.html">KAMAR</a></li>
      <li class="border"></li>
      <li><a href="akun.html">AKUN</a></li>
     </nav>
    </nav>
   </div>
  </header>
  <main>
   <section class="kamar-section">
    <div class="page-header">
     <div class="page-content">
      <div class="breadcrumb">
       <a href="home.html" class="breadcrumb-link">
        <i class="fas fa-home"></i>
        Beranda
       </a>
       <span class="breadcrumb-separator">/</span>
       <span class="breadcrumb-current">
        <i class="fas fa-bed"></i>
        Kamar
       </span>
      </div>

      <div class="hero-main">
       <div class="hero-icon">
        <i class="fas fa-bed"></i>
       </div>
       <div class="hero-text">
        <h1 class="page-title">Galeri Kamar</h1>
        <p class="page-subtitle">
         Jelajahi berbagai pilihan kamar yang tersedia dengan fasilitas
```

```
            terbaik untuk kenyamanan Anda
          </p>
        </div>
      </div>
    </div>
  </div>

  <!-- Galeri Kamar Section -->
  <section class="gallery-section">
    <div class="gallery-bg">
      <span class="gallery-title">Galeri Kamar</span>
      <div class="gallery-grid">
        <img src="../Image/kamar1.png" alt="Kamar 1" />
        <img src="../Image/hal_depan1.png" alt="Kamar 2" />
        <img src="../Image/teras2.png" alt="Kamar 3" />
        <img src="../Image/kamar2.png" alt="Kamar 4" />
        <img src="../Image/parkiran1.png" alt="Kamar 5" />
        <img src="../Image/teras1.png" alt="Kamar 6" />
        <img src="../Image/kamar3.png" alt="Kamar 7" />
        <img src="../Image/ruang_tamu1.png" alt="Kamar 8" />
      </div>
    </div>
  </section>

  <!-- Ketersediaan Kamar Section -->
  <section class="fasilitas-section">
    <div class="swiper-container">
      <div class="gallery-title">Ketersediaan Kamar</div>
      <swiper-container
        class="kamar-swiper"
        space-between="16"
        slides-per-view="3"
        id="kamar-swiper"
      >
        <!-- Swiper-slide akan diisi otomatis oleh JS -->
      </swiper-container>
    </div>
  </section>

  <!-- Fasilitas Sections -->
  <section class="fasilitas-section">
    <div class="fasilitas-card">
      <div class="card-title">
        <img src="../Image/logofasilitas.png" alt="Fasilitas Kamar" />
        Fasilitas Kamar
```

```
        </div>
        <div class="fasilitas-list">
          <div class="fasilitas-item">
            <img src="../Image/kasur.png" alt="Kasur" /> Kasur
          </div>
          <div class="fasilitas-item">
            <img src="../Image/kursi.png" alt="Kursi" /> Kursi
          </div>
          <div class="fasilitas-item">
            <img src="../Image/ac.png" alt="AC" /> AC
          </div>
          <div class="fasilitas-item">
            <img src="../Image/lemari.png" alt="Lemari" /> Lemari
          </div>
          <div class="fasilitas-item">
            <img src="../Image/meja.png" alt="Meja" /> Meja
          </div>
          <div class="fasilitas-item">
            <img src="../Image/kamarMandi.png" alt="Kamar Mandi" /> Kamar
            Mandi
          </div>
        </div>
      </div>

      <div class="fasilitas-card">
        <div class="card-title">
          <img src="../Image/logofasilitas.png" alt="Fasilitas Umum" />
          Fasilitas Umum
        </div>
        <div class="fasilitas-list">
          <div class="fasilitas-item">
            <img src="../Image/wifi.png" alt="Wifi" /> Wifi
          </div>
          <div class="fasilitas-item">
            <img src="../Image/dapur.png" alt="Dapur" /> Dapur
          </div>
          <div class="fasilitas-item">
            <img src="../Image/kamarMandi.png" alt="Kamar Mandi Luar" />
            Kamar Mandi Luar
          </div>
          <div class="fasilitas-item">
            <img src="../Image/parkiran.png" alt="Parkiran" /> Parkiran
          </div>
          <div class="fasilitas-item">
```

```
            <img src="../Image/ruangTamu.png" alt="Ruang Tamu" /> Ruang
Tamu
          </div>
         </div>
        </div>

        <div class="fasilitas-card">
         <div class="card-title">
          <img src="../Image/logofasilitas.png" alt="Tempat Terdekat" />
          Tempat Terdekat
         </div>
         <div class="fasilitas-list">
          <div class="fasilitas-item">
           <img src="../Image/masjid.png" alt="Masjid" /> Masjid
          </div>
          <div class="fasilitas-item">
           <img src="../Image/atm.png" alt="ATM" /> ATM
          </div>
          <div class="fasilitas-item">
           <img src="../Image/tempatMakan.png" alt="Tempat Makan" />
Tempat
            Makan
          </div>
          <div class="fasilitas-item">
           <img src="../Image/supermarket.png" alt="Supermarket" />
           Supermarket
          </div>
          <div class="fasilitas-item">
           <img src="../Image/sekolah.png" alt="Sekolah" /> Sekolah
          </div>
         </div>
        </div>
       </section>
      </section>
    </main>

    <!-- Custom Modal -->
    <div id="confirmModal" class="custom-modal">
     <div class="modal-content">
      <div class="modal-header">
       <div class="modal-icon">
        <i class="fas fa-user-check"></i>
       </div>
       <h3 class="modal-title">Konfirmasi Akses</h3>
      </div>
```

```
      <div class="modal-body">
        <div class="modal-message">
          Anda akan diarahkan ke halaman akun untuk melihat status dan
          informasi lengkap mengenai kost Anda.
        </div>
        <div class="modal-user-info" id="modalUserInfo">
          <i class="fas fa-user"></i>
          <span id="modalUsername"></span>
        </div>
      </div>
      <div class="modal-actions">
        <button
          class="modal-btn modal-btn-cancel"
          onclick="hideConfirmModal()"
        >
          Batal
        </button>
        <button
          class="modal-btn modal-btn-confirm"
          onclick="confirmGoToAccount()"
        >
          Lanjutkan
        </button>
      </div>
    </div>
  </div>

  <footer class="footer">
    <div class="footer-container">
      <div class="footer-menu">
        <div class="footer-menu-col">
          <a href="home.html">Menu</a>
          <a href="kamar.html">Kamar</a>
        </div>
        <div class="footer-menu-col">
          <a href="akun.html">Akun</a>
          <a href="login.html">Login</a>
        </div>
      </div>
      <div class="footer-divider"></div>
      <div class="footer-info">
        <div class="footer-title">Kost Putri MEWAH Patemon</div>
        <div class="footer-address">
          Jl. Serayu IV No. 13 03/01 Desa Patemon,<br />Kecamatan Gombong,
          Kabupaten Kebumen
```

```
        </div>
       </div>
      </div>
      <div class="footer-copyright">
       Copyright Kos MEWAH Patemon, Gombong 2025
      </div>
    </footer>
    <script src="https://cdn.jsdelivr.net/npm/swiper@11/swiper-element-
bundle.min.js"></script>
    <script src="../js/auth.js"></script>
    <script>
      // Toast Message Functions
      function showToast(message, type = "success", duration = 4000) {
       // Remove existing toast if any
       const existingToast = document.querySelector(".toast");
       if (existingToast) {
        existingToast.remove();
       }

       // Create toast element
       const toast = document.createElement("div");
       toast.className = `toast ${type}`;

       // Set icon based on type
       let iconClass = "fa-check-circle";
       switch (type) {
        case "error":
         iconClass = "fa-exclamation-triangle";
         break;
        case "warning":
         iconClass = "fa-exclamation-circle";
         break;
        case "info":
         iconClass = "fa-info-circle";
         break;
        case "welcome":
         iconClass = "fa-user-check";
         break;
        default:
         iconClass = "fa-check-circle";
       }

       toast.innerHTML = `
        <div class="toast-icon">
         <i class="fas ${iconClass}"></i>
```

```
    </div>
    <div class="toast-message">${message}</div>
    <button class="toast-close" onclick="hideToast()">
      <i class="fas fa-times"></i>
    </button>
  `;

  // Add to body
  document.body.appendChild(toast);

  // Show toast with animation
  setTimeout(() => {
    toast.classList.add("show");
  }, 100);

  // Auto hide after specified duration
  setTimeout(() => {
    hideToast();
  }, duration);
}

// Function to show welcome toast
function showWelcomeToast(message) {
showToast(message, "welcome", 5000);
}

// Function to hide toast
function hideToast() {
  const toast = document.querySelector(".toast");
  if (toast) {
    toast.classList.add("hide");
    setTimeout(() => {
      toast.remove();
    }, 400);
  }
}

// Function to show success toast
function showSuccessToast(message) {
showToast(message, "success");
}

// Function to show error toast
function showErrorToast(message) {
showToast(message, "error");
```

```
}

// Function to show warning toast
function showWarningToast(message) {
showToast(message, "warning");
}

// Function to show info toast
function showInfoToast(message) {
showToast(message, "info");
}

// Custom Modal Functions
function showConfirmModal(userName) {
  const modal = document.getElementById("confirmModal");
  const usernameSpan = document.getElementById("modalUsername");

  usernameSpan.textContent = `Masuk sebagai: ${userName}`;
  modal.classList.add("show");

  // Prevent body scrolling
  document.body.style.overflow = "hidden";
}

function hideConfirmModal() {
  const modal = document.getElementById("confirmModal");
  modal.classList.remove("show");

  // Restore body scrolling
  document.body.style.overflow = "";
}

function confirmGoToAccount() {
  hideConfirmModal();
  showInfoToast("Mengarahkan ke halaman akun...");

  // Small delay for better UX
  setTimeout(() => {
    window.location.href = "akun.html";
  }, 500);
}

// Close modal when clicking outside
document.addEventListener("click", function (e) {
  const modal = document.getElementById("confirmModal");
```

```
    if (e.target === modal) {
      hideConfirmModal();
    }
  });

  // Close modal with Escape key
  document.addEventListener("keydown", function (e) {
    if (e.key === "Escape") {
      hideConfirmModal();
    }
  });

  async function loadKamar() {
    const container = document.getElementById("kamar-swiper");
    try {
      const res = await fetch("http://localhost:3000/api/kamar");
      const data = await res.json();
      if (!data.success) throw new Error("Gagal mengambil data kamar");

      // Hapus slide lama jika ada
      container.innerHTML = "";

      // Cek status login untuk menentukan style tombol
      const userLoggedIn = isUserLoggedIn();
      const user = getLoggedInUser();

      data.data.forEach((kamar) => {
        const slide = document.createElement("swiper-slide");
        slide.className = "kamar-slide";

        // Tentukan style dan text tombol berdasarkan status login
        let buttonHTML = "";
        if (kamar.Ketersediaan == 1) {
          if (userLoggedIn) {
            buttonHTML = `<button class="btn btn-success rounded-pill px-3
py-2 shadow-sm btn-booking-logged-in"
onclick="showConfirmModal('${user.nama}')">Lihat  Status</button>`;
          } else {
            buttonHTML = `<button class="btn btn-primary rounded-pill px-3
py-2 shadow-sm btn-booking-guest"
onclick="handleBooking('${encodeURIComponent(
              kamar.No_Kamar
            )}', '${encodeURIComponent(
              kamar.Nama_Kamar
            )}')">Daftar Sekarang</button>`;
```

```
          }
        } else {
          buttonHTML = `<span class="badge rounded-pill bg-danger px-3 py-2
shadow-sm">Terisi</span>`;
        }

        slide.innerHTML = `
      <div class="card h-100 shadow-sm kamar-modern-card">
        <img src="${
          kamar.gambar || "../Image/kamar1.png"
        }" class="card-img-top" alt="${kamar.Nama_Kamar}">
        <div class="card-body d-flex flex-column align-items-start">
          <div class="fw-bold">${kamar.Nama_Kamar}</div>
          <div class="text-muted">${kamar.Letak}</div>
          ${buttonHTML}
        </div>
      </div>
    `;
        container.appendChild(slide);
      });
    } catch (err) {
      container.innerHTML = `<div style="color:red;padding:24px;">Gagal
memuat data kamar</div>`;
      showErrorToast("Gagal memuat data kamar. Silakan coba lagi nanti.");
    }
  }

  // Fungsi untuk handle klik booking
  function handleBooking(kamarId, namaKamar) {
    // Cek apakah user sudah login
    if (isUserLoggedIn()) {
      // Jika sudah login, langsung arahkan ke halaman akun
      const user = getLoggedInUser();

      // Tampilkan toast dengan opsi untuk melihat akun
      showInfoToast(
        `Halo ${user.nama}! Anda sudah terdaftar sebagai penghuni kost.
Silakan cek status akun Anda.`
      );

      // Redirect setelah 3 detik
      setTimeout(() => {
        window.location.href = "akun.html";
      }, 3000);
    } else {
```

```
        // Jika belum login, arahkan ke halaman reservasi untuk mendaftar
        showWarningToast(
          "Anda perlu mendaftar terlebih dahulu untuk melakukan booking."
        );

        // Redirect ke halaman reservasi setelah 2 detik
        setTimeout(() => {
          window.location.href = `reservasi.html?id=${kamarId}`;
        }, 2000);
      }
    }

    // Setup menu akun link berdasarkan status login
    document.addEventListener("DOMContentLoaded", function () {
    loadKamar();

      // Show welcome toast if user is logged in
      const user = getLoggedInUser();
      if (user) {
        showWelcomeToast(
          `Selamat datang, ${user.nama}! Anda sudah terdaftar sebagai
penghuni kost.`
        );
      }

      // Update menu AKUN berdasarkan status login
      const menuAkunItems =
document.querySelectorAll('a[href="akun.html"]');
      const footerAkunItems = document.querySelectorAll(
        'a[href="#"]:not([href*="maps"])'
      );

      // Cari link akun di footer dan update
      footerAkunItems.forEach((item) => {
        if (item.textContent.trim().toLowerCase() === "akun") {
          if (isUserLoggedIn()) {
            item.href = "akun.html";
          } else {
            item.href = "login.html";
          }
        }
      });

      // Update menu akun di header
      menuAkunItems.forEach((item) => {
```

```
        item.addEventListener("click", function (e) {
          if (!isUserLoggedIn()) {
            e.preventDefault();
            showWarningToast(
              "Anda harus login terlebih dahulu untuk mengakses halaman
akun!"
            );
            setTimeout(() => {
              window.location.href = "login.html";
            }, 1000);
          }
        });
      });

      // Tampilkan indikator jika sudah login
      if (user) {
        menuAkunItems.forEach((item) => {
          if (item.textContent.trim() === "AKUN") {
            item.style.color = "#ffffff";
            item.title = `Logged in as ${user.nama}`;
          }
        });
      }
    });
  </script>
 </body>
```

Tabel 4. 3 Codingan login.html

Login.html

```
<meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
  <title>Login - Kost Mewah Patemon Gombong</title>
  <link rel="icon" href="../image/logo.png" />
  <link

href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;
600;700;800;900&display=swap"
    rel="stylesheet"
  />
  <link
    rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.2/css/all.min.css"
```

```
    />
  <body>
    <header class="header">
     <div class="logo" onclick="window.location.href='home.html'">
      <div class="circle-container">
       <div class="circle">
        <img
         src="../Image/logo.png"
         alt="Kost Mewah Patemon Gombong Logo"
        />
       </div>
      </div>
     </div>
     <div class="title" onclick="window.location.href='home.html'">
      <img src="../Image/logoTeks.png" alt="Title Image" />
     </div>
    </header>

    <div class="login-container">
     <div class="login-box">
      <div class="login-content">
       <div class="login-header">
        <h1 class="login-title">Selamat Datang</h1>
        <p class="login-subtitle">
         Silakan masuk ke akun Anda untuk melanjutkan
        </p>
       </div>

       <form id="loginForm" action="#" method="POST">
        <div class="form-group">
         <label for="email">
          <i class="fas fa-envelope"></i>
          Email
         </label>
         <input
          type="email"
          name="email"
          id="email"
          placeholder="Masukkan email Anda"
          required
         />
        </div>

        <div class="form-group">
         <label for="password">
```

```
            <i class="fas fa-lock"></i>
            Password
          </label>
          <div class="input-wrapper">
            <input
              type="password"
              name="password"
              id="password"
              placeholder="Masukkan password Anda"
              required
            />
            <button
              type="button"
              class="toggle-password"
              onclick="togglePassword()"
            >
              <i class="fas fa-eye" id="passwordIcon"></i>
            </button>
          </div>
        </div>

        <button type="submit" class="login-btn">
          <i class="fas fa-sign-in-alt"></i>
          <span id="loginBtnText">Masuk</span>
        </button>
      </form>

      <!-- Error message (hidden by default, using toast messages instead) --
>
      <div id="errorMessage" class="error-message" style="display: none">
        <i class="fas fa-exclamation-triangle"></i>
        <span id="errorText"></span>
      </div>

      <!-- Success message (hidden by default, using toast messages instead)
-->
      <div
        id="successMessage"
        class="success-message"
        style="display: none"
      >
        <i class="fas fa-check-circle"></i>
        <span id="successText"></span>
      </div>
    </div>
```

```
  </div>
</div>

<script src="../js/auth.js"></script>
<script>
  // Fungsi untuk toggle password visibility
  function togglePassword() {
    const passwordField = document.getElementById("password");
    const passwordIcon = document.getElementById("passwordIcon");

    if (passwordField.type === "password") {
      passwordField.type = "text";
      passwordIcon.classList.remove("fa-eye");
      passwordIcon.classList.add("fa-eye-slash");
    } else {
      passwordField.type = "password";
      passwordIcon.classList.remove("fa-eye-slash");
      passwordIcon.classList.add("fa-eye");
    }
  }

  // Toast Message Functions
  function showToast(message, type = "success", duration = 4000) {
    // Remove existing toast if any
    const existingToast = document.querySelector(".toast");
    if (existingToast) {
      existingToast.remove();
    }

    // Create toast element
    const toast = document.createElement("div");
    toast.className = `toast ${type}`;

    // Set icon based on type
    let iconClass = "fa-check-circle";
    switch (type) {
      case "error":
        iconClass = "fa-exclamation-triangle";
        break;
      case "warning":
        iconClass = "fa-exclamation-circle";
        break;
      case "info":
        iconClass = "fa-info-circle";
        break;
```

```
    case "welcome":
      iconClass = "fa-user-check";
      break;
    default:
      iconClass = "fa-check-circle";
  }

  toast.innerHTML = `
    <div class="toast-icon">
      <i class="fas ${iconClass}"></i>
    </div>
    <div class="toast-message">${message}</div>
    <button class="toast-close" onclick="hideToast()">
      <i class="fas fa-times"></i>
    </button>
  `;

  // Add to body
  document.body.appendChild(toast);

  // Show toast with animation
  setTimeout(() => {
    toast.classList.add("show");
  }, 100);

  // Auto hide after specified duration
  setTimeout(() => {
    hideToast();
  }, duration);
}

// Function to show welcome toast
function showWelcomeToast(message) {
showToast(message, "welcome", 5000);
}

// Function to hide toast
function hideToast() {
  const toast = document.querySelector(".toast");
  if (toast) {
    toast.classList.add("hide");
    setTimeout(() => {
      toast.remove();
    }, 400);
  }
```

```
    }

    // Function to show success toast
    function showSuccessToast(message) {
    showToast(message, "success");
    }

    // Function to show error toast
    function showErrorToast(message) {
    showToast(message, "error");
    }

    // Function to show warning toast
    function showWarningToast(message) {
    showToast(message, "warning");
    }

    // Function to show info toast
    function showInfoToast(message) {
    showToast(message, "info");
    }

    // Fungsi untuk menampilkan pesan error (now using toast)
    function showError(message) {
      // Use toast message instead of inline error for better UX
      showErrorToast(message);

      // Hide inline error message since we're using toast
      const errorDiv = document.getElementById("errorMessage");
      if (errorDiv) {
        errorDiv.style.display = "none";
      }
    }

    // Fungsi untuk menampilkan pesan success (deprecated - use showToast
instead)
    function showSuccess(message) {
      showToast(message, "success");
    }

    // Fungsi untuk menampilkan loading
    function showLoading(show) {
      const submitBtn = document.querySelector(".login-btn");
      const btnText = document.getElementById("loginBtnText");
      const btnIcon = submitBtn.querySelector("i");
```

```
  if (show) {
    submitBtn.disabled = true;
    btnText.textContent = "Sedang Login...";
    btnIcon.className = "loading-spinner";
  } else {
    submitBtn.disabled = false;
    btnText.textContent = "Masuk";
    btnIcon.className = "fas fa-sign-in-alt";
  }
}

// Fungsi untuk menyimpan data login ke localStorage
function saveLoginData(responseData) {
  localStorage.setItem("isLoggedIn", "true");
  localStorage.setItem("userId", responseData.data.user.email);
  localStorage.setItem("userName", responseData.data.user.nama);
  localStorage.setItem("userEmail", responseData.data.user.email);
  localStorage.setItem("userRole", responseData.data.user.role);
  localStorage.setItem("token", responseData.data.token);
  localStorage.setItem("authToken", responseData.data.token);

  // Set login timestamp for welcome message tracking
  localStorage.setItem("login_time", new Date().getTime().toString());

  // Simpan data tambahan jika ada
  if (responseData.data.user.no_telp) {
    localStorage.setItem("userPhone", responseData.data.user.no_telp);
  }
  if (responseData.data.user.alamat) {
    localStorage.setItem("userAddress", responseData.data.user.alamat);
  }
  if (responseData.data.user.foto) {
    localStorage.setItem("userFoto", responseData.data.user.foto);
  }
}

// Fungsi untuk redirect berdasarkan role
function redirectUser(role) {
  if (role === "admin") {
    window.location.href = "admin-dashboard.html";
  } else if (role === "penyewa") {
    window.location.href = "home.html";
  } else {
    window.location.href = "home.html";
```

```
  }
}

// Event listener untuk form login
document
  .getElementById("loginForm")
  .addEventListener("submit", async function (e) {
    e.preventDefault();

    // Ambil data dari form
    const email = document.getElementById("email").value.trim();
    const password = document.getElementById("password").value.trim();

    // Validasi input
    if (!email || !password) {
      showErrorToast("Email dan password harus diisi!");
      return;
    }

    // Validasi format email
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
      showErrorToast("Format email tidak valid!");
      return;
    }

    // Tampilkan loading
    showLoading(true);

    try {
      // Kirim request login ke API
      const response = await fetch(
        "http://localhost:3000/api/auth/login",
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            Email: email,
            Password: password,
          }),
        }
      );
```

```
      const data = await response.json();

      // Sembunyikan loading
      showLoading(false);

      if (response.ok && data.success) {
        // Login berhasil
        saveLoginData(data);

        // Tampilkan pesan sukses
        showSuccess(
          "Login berhasil! Selamat datang, " + data.data.user.nama
        );

        // Redirect berdasarkan role setelah 1.5 detik
        setTimeout(() => {
          redirectUser(data.data.user.role);
        }, 1500);
      } else {
        // Login gagal
        showErrorToast(
          data.message || "Login gagal! Periksa email dan password Anda."
        );
      }
    } catch (error) {
      // Sembunyikan loading
      showLoading(false);

      // Handle error network atau server
      console.error("Error:", error);
      showErrorToast("Terjadi kesalahan jaringan. Silakan coba lagi.");
    }
  });

// Event listener untuk input fields (Enter key)
document
  .getElementById("email")
  .addEventListener("keypress", function (e) {
    if (e.key === "Enter") {
      document.getElementById("password").focus();
    }
  });

document
  .getElementById("password")
```

```
      .addEventListener("keypress", function (e) {
        if (e.key === "Enter") {
          document
            .getElementById("loginForm")
            .dispatchEvent(new Event("submit"));
        }
      });

    // Cek apakah user sudah login saat halaman dimuat
    window.addEventListener("load", function () {
      const isLoggedIn = localStorage.getItem("isLoggedIn");
      const userRole = localStorage.getItem("userRole");

      if (isLoggedIn === "true" && userRole) {
        // Jika sudah login, redirect ke halaman yang sesuai
        redirectUser(userRole);
      }
    });

    // Add click handlers for logo and title
    document.addEventListener("DOMContentLoaded", function () {
      const logo = document.querySelector(".logo");
      const title = document.querySelector(".title");

      if (logo) {
        logo.addEventListener("click", function () {
          window.location.href = "home.html";
        });
      }

      if (title) {
        title.addEventListener("click", function () {
          window.location.href = "home.html";
        });
      }
    });
  </script>
</body>
```

Tabel 4. 4 Codingan reservasi.html

	Reservasi.html
	<meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0" />

```
    <title>Reservasi - Kost Mewah Patemon Gombong</title>
    <link rel="icon" href="../image/logo.png" />
    <link

href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,10
0..900;1,100..900&family=Roboto:ital,wght@0,100..900;1,100..900&display
=swap"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/swiper@11/swiper-bundle.min.css"
    />
    <link rel="stylesheet" type="text/css" href="../css/Reservasi.css" />
    <link rel="stylesheet" type="text/css" href="../css/home.css" />
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.2/css/all.min.css"
    />
    <!-- Bootstrap CSS -->
    <link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.
css"
      rel="stylesheet"
    />
  <body>
    <header class="header">
      <div class="logo">
        <a href="home.html" class="logo-link">
          <div class="circle-container">
            <div class="circle">
              <img
                src="../Image/logo.png"
                alt="Kost Mewah Patemon Gombong Logo"
              />
            </div>
          </div>
        </a>
      </div>
      <div class="title">
        <a href="home.html" class="title-link">
          <img src="../Image/logoTeks.png" alt="Title Image" />
        </a>
```

```
  </div>
</header>

<main>
  <section class="reservasi-section">
    <div class="page-header">
      <div class="page-content">
        <div class="breadcrumb">
          <a href="home.html" class="breadcrumb-link">
            <i class="fas fa-home"></i>
            Beranda
          </a>
          <span class="breadcrumb-separator">/</span>
          <a href="kamar.html" class="breadcrumb-link">
            <i class="fas fa-bed"></i>
            Kamar
          </a>
          <span class="breadcrumb-separator">/</span>
          <span class="breadcrumb-current">
            <i class="fas fa-calendar-plus"></i>
            Reservasi Kamar
          </span>
        </div>

        <div class="hero-main">
          <div class="hero-icon">
            <i class="fas fa-calendar-plus"></i>
          </div>
          <div class="hero-text">
            <h1 class="page-title">Reservasi Kamar</h1>
            <p class="page-subtitle">
              Daftar dan dapatkan kamar impian Anda di Kost Mewah Patemon
              Gombong
            </p>
          </div>
        </div>
      </div>
    </div>

    <div class="reservasi-layout">
      <div class="form-card">
        <div class="form-header">
          <h2 class="form-title">
            <i class="fas fa-user-plus"></i>
            Formulir Pendaftaran
```

```
          </h2>
          <p class="form-subtitle">
            Lengkapi data diri Anda untuk melakukan reservasi kamar
          </p>
        </div>

        <div class="form-content">
          <form class="reservasi-form">
            <div class="form-group">
              <label for="nama" class="form-label">
                <i class="fas fa-user"></i>
                Nama Lengkap
              </label>
              <input
                type="text"
                id="nama"
                name="nama"
                class="form-control"
                placeholder="Masukkan nama lengkap Anda"
                required
              />
            </div>

            <div class="form-group">
              <label for="email" class="form-label">
                <i class="fas fa-envelope"></i>
                Email
              </label>
              <input
                type="email"
                id="email"
                name="email"
                class="form-control"
                placeholder="Masukkan email Anda"
                required
              />
            </div>

            <div class="form-group">
              <label for="password" class="form-label">
                <i class="fas fa-lock"></i>
                Password
              </label>
              <input
                type="password"
```

```
                id="password"
                name="password"
                class="form-control"
                placeholder="Masukkan password"
                required
              />
            </div>

            <div class="form-group">
              <label for="confirmPassword" class="form-label">
                <i class="fas fa-lock"></i>
                Konfirmasi Password
              </label>
              <input
                type="password"
                id="confirmPassword"
                name="confirmPassword"
                class="form-control"
                placeholder="Konfirmasi password"
                required
              />
            </div>

            <div class="form-group">
              <label for="noTelp" class="form-label">
                <i class="fas fa-phone"></i>
                No. Telepon
              </label>
              <input
                type="tel"
                id="noTelp"
                name="noTelp"
                class="form-control"
                placeholder="Masukkan nomor telepon"
                required
              />
            </div>

            <div class="form-group">
              <label for="alamat" class="form-label">
                <i class="fas fa-map-marker-alt"></i>
                Alamat
              </label>
              <textarea
                id="alamat"
```

```
                name="alamat"
                class="form-control"
                placeholder="Masukkan alamat lengkap"
                rows="4"
                required
              ></textarea>
            </div>
          </form>
        </div>
      </div>

      <div class="info-card">
        <div class="info-header">
          <h2 class="info-title">
            <i class="fas fa-info-circle"></i>
            Informasi Reservasi
          </h2>
        </div>

        <div class="info-content">
          <div class="price-section">
            <div class="price-card">
              <div class="price-header">
                <i class="fas fa-tag"></i>
                <span>Harga Kamar</span>
              </div>
              <div class="price-amount">
                Rp. 900.000<span class="price-period">/bulan</span>
              </div>
            </div>
          </div>

          <div class="ketentuan-section">
            <h3 class="ketentuan-title">
              <i class="fas fa-clipboard-list"></i>
              Ketentuan Reservasi
            </h3>
            <ul class="ketentuan-list">
              <li class="ketentuan-item">
                <i class="fas fa-dollar-sign"></i>
                <span>Harga kamar per-bulan Rp. 900.000,-</span>
              </li>
              <li class="ketentuan-item">
                <i class="fas fa-envelope"></i>
                <span>Pengguna akan menerima email kurang dari 2 hari</span>
```

```
            </li>
            <li class="ketentuan-item">
              <i class="fas fa-phone"></i>
              <span>088216003562 untuk kontak bantuan!</span>
            </li>
            <li class="ketentuan-item">
              <i class="fas fa-shield-alt"></i>
              <span>Jaminan keamanan 24 jam</span>
            </li>
            <li class="ketentuan-item">
              <i class="fas fa-wifi"></i>
              <span>WiFi gratis untuk semua penghuni</span>
            </li>
            <li class="ketentuan-item">
              <i class="fas fa-parking"></i>
              <span>Parkir motor gratis tersedia</span>
            </li>
          </ul>
        </div>

        <div class="submit-section">
          <button class="btn-submit">
            <i class="fas fa-paper-plane"></i>
            <span>Kirim Reservasi</span>
          </button>
        </div>
      </div>
    </div>
  </div>
 </section>
</main>

<!-- Modal Pembayaran -->
<div
 class="modal fade"
 id="modalPembayaran"
 tabindex="-1"
 aria-labelledby="paymentModalLabel"
 aria-hidden="true"
>
 <div class="modal-dialog modal-lg">
  <div class="modal-content">
   <div class="modal-header">
    <h5
     class="modal-title fw-bold"
```

```
            id="paymentModalLabel"
            style="
              color: #6d5a2b;
              font-size: 1.7rem;
              font-weight: 800;
              text-shadow: 0 1px 2px rgba(0, 0, 0, 0.1);
            "
          >
            <i class="fas fa-receipt me-2"></i>Instruksi Pembayaran
          </h5>
        </div>
        <div class="modal-body">
          <div id="paymentContent">
            <div
              class="rekening-card mb-4"
              style="
                background: linear-gradient(135deg, #f8f9fa, #e9ecef);
                border-radius: 15px;
                padding: 25px;
                border: 2px solid #eec55b;
                box-shadow: 0 5px 15px rgba(238, 197, 91, 0.1);
              "
            >
              <div
                style="
                  display: flex;
                  align-items: center;
                  margin-bottom: 15px;
                  font-size: 16px;
                "
              >
                <span
                  style="color: #eec55b; margin-right: 12px; font-size: 20px"
                >
                  <i class="fa-solid fa-building-columns"></i>
                </span>
                <strong>Bank:</strong>
                <span style="margin-left: 8px">BCA</span>
              </div>
              <div
                style="
                  display: flex;
                  align-items: center;
                  margin-bottom: 15px;
                  font-size: 16px;
```

```
            "
          >
            <span
              style="color: #eec55b; margin-right: 12px; font-size: 20px"
            >
              <i class="fa-solid fa-credit-card"></i>
            </span>
            <strong>No. Rekening:</strong>
            <span style="margin-left: 8px">1234567890</span>
          </div>
          <div
            style="display: flex; align-items: center; font-size: 16px"
          >
            <span
              style="color: #eec55b; margin-right: 12px; font-size: 20px"
            >
              <i class="fa-solid fa-id-card"></i>
            </span>
            <strong>Atas Nama:</strong>
            <span style="margin-left: 8px">Kos Putri Melati</span>
          </div>
        </div>

        <div
          class="total-bayar mb-4"
          style="
            text-align: center;
            background: linear-gradient(135deg, #eec55b, #f8e07a);
            color: #6d5a2b;
            padding: 15px;
            border-radius: 12px;
            font-size: 20px;
            font-weight: bold;
            box-shadow: 0 3px 10px rgba(238, 197, 91, 0.3);
          "
        >
          Total Bayar: <span style="color: #8a7037">Rp. 900.000</span>
        </div>

        <!-- Accordion Instruksi Pembayaran -->
        <div class="accordion-instruksi mb-4">
          <button
            class="accordion-btn w-100 p-3"
            type="button"
            id="accordionInstruksiBtn"
```

```
                aria-expanded="false"
                aria-controls="instruksiContent"
                style="
                  background: #f8f9fa;
                  border: 2px solid #eec55b;
                  border-radius: 10px;
                  font-weight: bold;
                color: #6d5a2b;
                transition: all 0.3s;
                "
              >
                <i
                  class="fa-solid fa-chevron-down me-2"
                  id="instruksiIcon"
                ></i>
                Lihat Instruksi Pembayaran ATM &amp;amp; M-Banking
              </button>
              <div class="accordion-content" id="instruksiContent">
                <div class="instruksi-detail">
                  <div class="mb-4">
                    <strong style="color: #6d5a2b; font-size: 16px"
                      >Instruksi via ATM BCA:</strong
                    >
                    <ol
                      style="
                        margin-top: 12px;
                        padding-left: 20px;
                        line-height: 1.6;
                      "
                    >
                      <li>Masukkan kartu ATM dan PIN Anda.</li>
                      <li>
                        Pilih menu <strong>Transaksi Lainnya</strong> →
                        <strong>Transfer</strong> →
                        <strong>Ke Rekening BCA</strong>.
                      </li>
                      <li>
                        Masukkan nomor rekening: <strong>1234567890</strong>
                      </li>
                      <li>Masukkan nominal transfer sesuai tagihan.</li>
                      <li>Ikuti instruksi hingga transaksi selesai.</li>
                    </ol>
                  </div>
                  <div class="mb-3">
                    <strong style="color: #6d5a2b; font-size: 16px"
```

```
                >Instruksi via M-Banking BCA:</strong
              >
              <ol
                style="
                  margin-top: 12px;
                  padding-left: 20px;
                  line-height: 1.6;
                "
              >
                <li>Login aplikasi BCA Mobile.</li>
                <li>
                  Pilih menu <strong>m-Transfer</strong> →
                  <strong>Antar Rekening BCA</strong>.
                </li>
                <li>
                  Masukkan nomor rekening: <strong>1234567890</strong>
                </li>
                <li>Masukkan nominal transfer sesuai tagihan.</li>
                <li>Ikuti instruksi hingga transaksi selesai.</li>
              </ol>
            </div>
          </div>
        </div>
      </div>

      <!-- Form Upload Bukti -->
      <form id="formPembayaran" class="form-bukti">
        <div class="mb-3">
          <label for="buktiPembayaran" class="form-label fw-bold"
            >Upload Bukti Pembayaran:</label
          >
          <input
            type="file"
            class="form-control"
            id="buktiPembayaran"
            name="buktiPembayaran"
            accept="image/*"
            required
            style="
              border: 2px solid #eec55b;
              border-radius: 8px;
              padding: 12px;
            "
          />
          <div class="form-text">Format: JPG, JPEG, PNG (Max 5MB)</div>
```

```
            </div>
            <input type="hidden" id="idKamar" name="idKamar" />
            <div class="modal-btn-row d-flex gap-3">
              <button
                type="submit"
                class="btn flex-fill btn-gradient"
                style="
                  background: linear-gradient(135deg, #eec55b, #f8e07a);
                  border: none;
                  color: #6d5a2b;
                  font-weight: bold;
                  padding: 12px;
                  border-radius: 8px;
                  transition: all 0.3s;
                "
              >
                <i class="fas fa-upload me-2"></i>Kirim Bukti Pembayaran
              </button>
              <button
                type="button"
                class="btn btn-outline-secondary flex-fill"
                id="batalPembayaran"
              >
                Batal
              </button>
            </div>
          </form>

          <div
            class="modal-note mt-3"
            style="
              text-align: center;
              color: #6c757d;
              font-size: 14px;
              font-style: italic;
            "
          >
            Setelah pembayaran dan upload bukti, pembayaran Anda akan
segera
            diproses dalam 1x24 jam.
          </div>
        </div>
      </div>
    </div>
  </div>
```

```
</div>

<script src="../js/auth.js"></script>
<script>
 // No page protection needed - this page is for new user registration
 document.addEventListener("DOMContentLoaded", function () {
  // Initialize page without authentication requirements
  console.log("Reservasi page loaded - ready for new user registration");
 });

 // Toast Message Functions
 function showToast(message, type = "success", duration = 4000) {
  // Remove existing toast if any
  const existingToast = document.querySelector(".toast");
  if (existingToast) {
   existingToast.remove();
  }

  // Create toast element
  const toast = document.createElement("div");
  toast.className = `toast ${type}`;

  // Set icon based on type
  let iconClass = "fa-check-circle";
  switch (type) {
   case "error":
    iconClass = "fa-exclamation-triangle";
    break;
   case "warning":
    iconClass = "fa-exclamation-circle";
    break;
   case "info":
    iconClass = "fa-info-circle";
    break;
   case "welcome":
    iconClass = "fa-user-check";
    break;
   default:
    iconClass = "fa-check-circle";
  }

  toast.innerHTML = `
   <div class="toast-icon">
    <i class="fas ${iconClass}"></i>
   </div>
```

```
    <div class="toast-message">${message}</div>
    <button class="toast-close" onclick="hideToast()">
      <i class="fas fa-times"></i>
    </button>
  `;

  // Add to body
  document.body.appendChild(toast);

  // Show toast with animation
  setTimeout(() => {
    toast.classList.add("show");
  }, 100);

  // Auto hide after specified duration
  setTimeout(() => {
    hideToast();
  }, duration);
}

// Function to show welcome toast
function showWelcomeToast(message) {
showToast(message, "welcome", 5000);
}

// Function to hide toast
function hideToast() {
  const toast = document.querySelector(".toast");
  if (toast) {
    toast.classList.add("hide");
    setTimeout(() => {
      toast.remove();
    }, 400);
  }
}

// Function to show success toast
function showSuccessToast(message) {
showToast(message, "success");
}

// Function to show error toast
function showErrorToast(message) {
showToast(message, "error");
}
```

```
// Function to show warning toast
function showWarningToast(message) {
showToast(message, "warning");
}

// Function to show info toast
function showInfoToast(message) {
showToast(message, "info");
}

document
  .querySelector(".btn-submit")
  .addEventListener("click", function (e) {
    e.preventDefault();

    // Validasi form sebelum membuka modal
    const Nama = document.getElementById("nama").value.trim();
    const Email = document.getElementById("email").value.trim();
    const Password = document.getElementById("password").value;
    const ConfirmPassword =
      document.getElementById("confirmPassword").value;
    const No_telp = document.getElementById("noTelp").value.trim();
    const Alamat = document.getElementById("alamat").value.trim();

    // Cek apakah semua field sudah diisi
    if (
      !Nama ||
      !Email ||
      !Password ||
      !ConfirmPassword ||
      !No_telp ||
      !Alamat
    ) {
      showErrorToast(
        "Semua kolom harus diisi sebelum melanjutkan ke pembayaran!"
      );
      return;
    }

    // Validasi format email
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(Email)) {
      showErrorToast("Format email tidak valid!");
      return;
```

```
    }

    // Validasi password
    if (Password !== ConfirmPassword) {
      showErrorToast("Password dan konfirmasi password tidak sama!");
      return;
    }

    if (Password.length < 6) {
      showErrorToast("Password minimal 6 karakter!");
      return;
    }

    // Jika semua validasi berhasil, buka modal pembayaran
    const modal = new bootstrap.Modal(
      document.getElementById("modalPembayaran")
    );
    modal.show();
  });

document
  .getElementById("batalPembayaran")
  .addEventListener("click", function () {
    const modal = bootstrap.Modal.getInstance(
      document.getElementById("modalPembayaran")
    );
    if (modal) {
      modal.hide();
    }
  });

// Ambil id kamar dari URL dan set ke input hidden
const urlParams = new URLSearchParams(window.location.search);
const idKamar = urlParams.get("id");
console.log("ID Kamar dari URL:", idKamar);
if (idKamar) {
  document.getElementById("idKamar").value = idKamar;
  console.log(
    "ID Kamar berhasil di-set:",
    document.getElementById("idKamar").value
  );
} else {
  console.warn("ID Kamar tidak ditemukan di URL!");
}
```

```
// Event listener untuk accordion button
document
  .getElementById("accordionInstruksiBtn")
  .addEventListener("click", function (e) {
    e.preventDefault();
    const content = document.getElementById("instruksiContent");
    const icon = document.getElementById("instruksiIcon");
    const button = this;

    // Toggle the accordion content with smooth animation
    if (content.classList.contains("show")) {
      // Closing animation
      content.classList.remove("show");
      icon.classList.remove("fa-chevron-up");
      icon.classList.add("fa-chevron-down");
      icon.style.transform = "rotate(0deg)";
      button.setAttribute("aria-expanded", "false");
    } else {
      // Opening animation
      content.classList.add("show");
      icon.classList.remove("fa-chevron-down");
      icon.classList.add("fa-chevron-up");
      icon.style.transform = "rotate(180deg)";
      button.setAttribute("aria-expanded", "true");
    }
  });

document
  .getElementById("formPembayaran")
  .addEventListener("submit", async function (e) {
    e.preventDefault();

    // Validasi form registrasi
    const form = document.querySelector(".reservasi-form");
    const Nama = document.getElementById("nama").value.trim();
    const Email = document.getElementById("email").value.trim();
    const Password = document.getElementById("password").value;
    const ConfirmPassword =
      document.getElementById("confirmPassword").value;
    const No_telp = document.getElementById("noTelp").value.trim();
    const Alamat = document.getElementById("alamat").value.trim();
    const No_Kamar = document.getElementById("idKamar").value;
    const buktiPembayaran =
      document.getElementById("buktiPembayaran").files[0];
```

```
        console.log("Data form:", {
          Nama,
          Email,
          Password: Password ? "Ada" : "Kosong",
          ConfirmPassword: ConfirmPassword ? "Ada" : "Kosong",
          No_telp,
          Alamat,
          No_Kamar,
          buktiPembayaran: buktiPembayaran ? "Ada" : "Kosong",
        });

        // Validasi input
        if (
          !Nama ||
          !Email ||
          !Password ||
          !ConfirmPassword ||
          !No_telp ||
          !Alamat ||
          !No_Kamar
        ) {
          console.log("Validasi gagal:", {
            Nama: Nama,
            Email: Email,
            Password: Password ? "Ada" : "Kosong",
            ConfirmPassword: ConfirmPassword ? "Ada" : "Kosong",
            No_telp: No_telp,
            Alamat: Alamat,
            No_Kamar: No_Kamar,
          });
          showErrorToast("Semua field harus diisi!");
          return;
        }

        // Validasi format email
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(Email)) {
          showErrorToast("Format email tidak valid!");
          return;
        }

        if (Password !== ConfirmPassword) {
          showErrorToast("Password dan konfirmasi password tidak sama!");
          return;
        }
```

```
if (Password.length < 6) {
  showErrorToast("Password minimal 6 karakter!");
  return;
}

if (!buktiPembayaran) {
  showErrorToast("Bukti pembayaran harus diupload!");
  return;
}

// Disable submit button
const submitBtn = document.querySelector(".btn-gradient");
const originalText = submitBtn.innerHTML;
submitBtn.disabled = true;
submitBtn.innerHTML =
  '<i class="fa-solid fa-spinner fa-spin"></i> Mengirim...';

try {
  // Buat FormData untuk kirim file
  const formData = new FormData();
  formData.append("Nama", Nama);
  formData.append("Email", Email);
  formData.append("Password", Password);
  formData.append("No_telp", No_telp);
  formData.append("Alamat", Alamat);
  formData.append("No_Kamar", No_Kamar);
  formData.append("Tanggal_Reservasi", new Date().toISOString());
  formData.append("buktiPembayaran", buktiPembayaran);

  // Kirim tmp user data
  const response = await fetch(
    "http://localhost:3000/api/tmp-users",
    {
      method: "POST",
      body: formData,
    }
  );

  const result = await response.json();

  if (result.success) {
    showSuccessToast(
    result.message +
```

```
            ". Anda akan menerima notifikasi email setelah admin menyetujui
reservasi Anda."
          );

          // Reset form
          document.querySelector(".reservasi-form").reset();
          document.getElementById("buktiPembayaran").value = "";
          const modal = bootstrap.Modal.getInstance(
            document.getElementById("modalPembayaran")
          );
          if (modal) {
            modal.hide();
          }

          // Redirect ke home after toast
          setTimeout(() => {
            window.location.href = "home.html";
          }, 2000);
        } else {
          showErrorToast(
            result.message || "Terjadi kesalahan saat mengirim reservasi!"
          );
        }
      } catch (error) {
        console.error("Error:", error);
        showErrorToast(
          "Terjadi kesalahan saat mengirim reservasi. Silakan coba lagi."
        );
      } finally {
        // Re-enable submit button
        submitBtn.disabled = false;
        submitBtn.innerHTML = originalText;
      }
    });
  </script>

  <!-- Bootstrap JS -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle
.min.js"></script>
 </body>
```

Tabel 4. 5 Codingan AuthController.js

	AuthController.js

```
const User = require("../models/User");
const UserToken = require("../models/UserToken");
const jwt = require("jsonwebtoken");

class AuthController {
 // User Registration
 static async register(req, res) {
  try {
   const result = await User.create(req.body);

   if (!result.success) {
    return res.status(400).json(result);
   }

   res.status(201).json({
    success: true,
    message: "User registered successfully",
    data: {
     userId: result.userId,
    },
   });
  } catch (error) {
   console.error("Registration error:", error);
   res.status(500).json({
    success: false,
    message: "Internal server error",
   });
  }
 }

 // User Login
 static async login(req, res) {
  try {
   const { Email, Password } = req.body;

   // Find user
   const user = await User.findByEmail(Email);
   if (!user) {
    return res.status(401).json({
     success: false,
     message: "Invalid email or password",
    });
   }

   // Verify password
```

```
    const isValidPassword = await User.verifyPassword(
      Password,
      user.Password
    );
    if (!isValidPassword) {
      return res.status(401).json({
        success: false,
        message: "Invalid email or password",
      });
    }

    // Generate JWT token
    const token = jwt.sign(
      { email: user.Email, role: user.Role },
      process.env.JWT_SECRET,
      { expiresIn: "24h" }
    );

    res.json({
      success: true,
      message: "Login successful",
      data: {
        token,
        user: {
          nama: user.Nama,
          email:  user.Email,
          role: user.Role,
        },
      },
    });
  } catch (error) {
    console.error("Login error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Token-based login for approved reservations
static async loginWithToken(req, res) {
  try {
    const { token } = req.body;

    if (!token) {
```

```
      return res.status(400).json({
        success: false,
        message: "Token is required",
      });
    }

    // Verify token
    const tokenData = await UserToken.verifyToken(token);
    if (!tokenData) {
      return res.status(401).json({
        success: false,
        message: "Invalid or expired token",
      });
    }

    // Find user
    const user = await User.findByEmail(tokenData.Email);
    if (!user) {
      return res.status(401).json({
        success: false,
        message: "User not found",
      });
    }

    // Mark token as used
    await UserToken.markTokenAsUsed(token);

    // Generate JWT token
    const jwtToken = jwt.sign(
      { email: user.Email, role: user.Role },
      process.env.JWT_SECRET,
      { expiresIn: "24h" }
    );

    res.json({
      success: true,
      message: "Login successful",
      data: {
        token: jwtToken,
        user: {
          nama: user.Nama,
          email:  user.Email,
          role: user.Role,
        },
      },
```

```
    });
  } catch (error) {
    console.error("Token login error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get User Profile
static async getProfile(req, res) {
  try {
    const profile = await User.getProfile(req.user.Email);

    if (!profile) {
      return res.status(404).json({
        success: false,
        message: "User not found",
      });
    }

    res.json({
      success: true,
      data: profile,
    });
  } catch (error) {
    console.error("Get profile error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Update User Profile
static async updateProfile(req, res) {
  try {
    const updateData = { ...req.body };

    // If file was uploaded, add file path to update data
    if (req.file) {
      // Store relative path for frontend access
      updateData.Foto = `/uploads/profiles/${req.file.filename}`;
    }
```

```
    const result = await User.updateProfile(req.user.Email, updateData);

    if (!result.success) {
      return res.status(404).json(result);
    }

    res.json(result);
  } catch (error) {
    console.error("Update profile error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Change Password
static async changePassword(req, res) {
  try {
    const { currentPassword, newPassword, confirmPassword } = req.body;

    // Validation
    if (!currentPassword || !newPassword || !confirmPassword) {
      return res.status(400).json({
        success: false,
        message: "Semua field password harus diisi",
      });
    }

    if (newPassword !== confirmPassword) {
      return res.status(400).json({
        success: false,
        message: "Password baru dan konfirmasi password tidak sama",
      });
    }

    if (newPassword.length < 6) {
      return  res.status(400).json({
      success: false,
        message: "Password baru minimal 6 karakter",
      });
    }

    const result = await User.changePassword(
```

```
      req.user.Email,
      currentPassword,
      newPassword
    );

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json(result);
  } catch (error) {
    console.error("Change password error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get all reservations with user data for admin
static async getAllUsersWithReservations(req, res) {
try {
    const filters = {
      status: req.query.status || '',
      kamar: req.query.kamar || '',
      periode: req.query.periode || ''
    };

    const reservations = await User.getAllReservationsWithUserData(filters);

    res.json({
      success: true,
      data: reservations
    });
  } catch (error) {
    console.error("Get reservations with user data error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error"
    });
  }
}

// Get filter options for admin interface
static async getFilterOptions(req, res) {
```

```
    try {
      const options = await User.getFilterOptions();

      res.json({
        success: true,
        data: options
      });
    } catch (error) {
      console.error("Get filter options error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error"
      });
    }
  }

  // Update reservation status
  static async updateReservationStatus(req, res) {
    try {
      const { reservasiId } = req.params;
      const { status, keterangan } = req.body;

      if (!status) {
        return res.status(400).json({
          success: false,
          message: "Status is required"
        });
      }

      const result = await User.updateReservationStatus(reservasiId, status,
keterangan);

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json(result);
    } catch (error) {
      console.error("Update reservation status error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error"
      });
    }
  }
```

```
// Get all users for admin interface
static async getAllUsers(req, res) {
try {
   const users = await User.getAll();

   res.json({
     success: true,
     data: users
   });
  } catch (error) {
   console.error("Get all users error:", error);
   res.status(500).json({
     success: false,
     message: "Internal server error"
   });
  }
}

// Get dashboard statistics for admin
static async getDashboardStats(req, res) {
try {
   const stats = await User.getDashboardStatistics();

   res.json({
     success: true,
     data: stats
   });
  } catch (error) {
   console.error("Get dashboard stats error:", error);
   res.status(500).json({
     success: false,
     message: "Internal server error"
   });
  }
}

// Get all payments for admin pembayaran page
static async getAllPayments(req, res) {
  try {
   const filters = {
     status: req.query.status || '',
     bulan: req.query.bulan || '',
     tahun: req.query.tahun || '',
     kamar: req.query.kamar || ''
```

```
    };

    const payments = await User.getAllPaymentsForAdmin(filters);

    res.json({
      success: true,
      data: payments
    });
  } catch (error) {
    console.error("Get all payments error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error"
    });
  }
 }

 // Update payment status (admin only)
 static async updatePaymentStatus(req, res) {
 try {
    const { paymentId } = req.params;
    const { status, keterangan } = req.body;

    const validStatuses = ['Diterima', 'Menunggu', 'Ditolak'];
    if (!validStatuses.includes(status)) {
      return res.status(400).json({
        success: false,
        message: "Status tidak valid. Harus salah satu dari: " +
validStatuses.join(', ')
      });
    }

    const result = await User.updatePaymentStatus(paymentId, status,
keterangan);

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json(result);
  } catch (error) {
    console.error("Update payment status error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error"
```

```
    });
   }
  }
}

module.exports = AuthController;
```

Tabel 4. 6 Codingan KamarController.js

KamarController.js

```
const Kamar = require("../models/Kamar");

class KamarController {
 // Get all rooms
 static async getAll(req, res) {
  try {
   const rooms = await Kamar.getAll();

   res.json({
    success: true,
    data: rooms,
   });
  } catch (error) {
   console.error("Get all rooms error:", error);
   res.status(500).json({
    success: false,
    message: "Internal server error",
   });
  }
 }

 // Get available rooms
 static async getAvailable(req, res) {
  try {
   const rooms = await Kamar.getAvailable();

   res.json({
    success: true,
    data: rooms,
   });
  } catch (error) {
   console.error("Get available rooms error:", error);
   res.status(500).json({
    success: false,
    message: "Internal server error",
```

```
    });
  }
 }

 // Get room by ID
 static async getById(req, res) {
  try {
   const { id } = req.params;
   const room = await Kamar.getById(id);

   if (!room) {
    return res.status(404).json({
     success: false,
     message: "Room not found",
    });
   }

   res.json({
    success: true,
    data: room,
   });
  } catch (error) {
   console.error("Get room by ID error:", error);
   res.status(500).json({
    success: false,
    message: "Internal server error",
   });
  }
 }

 // Update room availability (Admin only)
 static async updateAvailability(req, res) {
 try {
   const { id } = req.params;
   const { availability } = req.body;

   // Validate availability
   if (availability !== 0 && availability !== 1) {
    return res.status(400).json({
     success: false,
     message: "Availability must be 0 (not available) or 1 (available)",
    });
   }

   const result = await Kamar.updateAvailability(id, availability);
```

```
    if (!result.success) {
      return res.status(404).json(result);
    }

    res.json(result);
  } catch (error) {
    console.error("Update room availability error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Create new room (Admin only)
static async create(req, res) {
  try {
    const { Nama_Kamar, Letak, Ketersediaan } = req.body;

    if (!Nama_Kamar || !Letak) {
      return  res.status(400).json({
      success: false,
        message: "Nama kamar dan letak harus diisi",
      });
    }

    const result = await Kamar.create({
      Nama_Kamar,
      Letak,
      Ketersediaan: Ketersediaan !== undefined ? parseInt(Ketersediaan) : 1,
    });

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.status(201).json(result);
  } catch (error) {
    console.error("Create room error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
```

```
}

// Update room (Admin only)
static async update(req, res) {
try {
  const { id } = req.params;
  const { Nama_Kamar, Letak, Ketersediaan } = req.body;

  if (!Nama_Kamar || !Letak) {
    return  res.status(400).json({
    success: false,
      message: "Nama kamar dan letak harus diisi",
    });
  }

  const result = await Kamar.update(id, {
    Nama_Kamar,
    Letak,
    Ketersediaan: parseInt(Ketersediaan),
  });

  if (!result.success) {
    return res.status(400).json(result);
  }

  res.json(result);
 } catch (error) {
  console.error("Update room error:", error);
  res.status(500).json({
    success: false,
    message: "Internal server error",
  });
 }
}

// Delete room (Admin only)
static async delete(req, res) {
try {
  const { id } = req.params;

  const result = await Kamar.delete(id);

  if (!result.success) {
    return res.status(400).json(result);
  }
```

```
    res.json(result);
   } catch (error) {
    console.error("Delete room error:", error);
    res.status(500).json({
     success: false,
     message: "Internal server error",
    });
   }
  }
}

module.exports = KamarController;
```

Tabel 4. 7 Codingan PaymentController.js

PaymentController.js

```
const Payment = require("../models/Payment");
const TmpUser = require("../models/TmpUser");
const path = require("path");
const fs = require("fs");

class PaymentController {
 // Get all payments for admin (Admin only)
 static async getAllPayments(req, res) {
  try {
   const result = await Payment.getAllPayments();

   if (!result.success) {
    return res.status(400).json(result);
   }

   res.json({
    success: true,
    data: result.data,
   });
  } catch (error) {
   console.error("Get all payments error:", error);
   res.status(500).json({
    success: false,
    message: "Internal server error",
   });
  }
 }
```

```
  // Upload payment proof (User)
  static async uploadPaymentProof(req, res) {
    try {
      const { reservationId } = req.params;

      if (!req.file) {
        return res.status(400).json({
          success: false,
          message: "Bukti pembayaran is required",
        });
      }

      const buktiPembayaranPath = path
        .join("uploads", "bukti_pembayaran", req.file.filename)
        .replace(/\\/g, "/");

      const result = await Payment.createMonthlyPayment(
        reservationId,
        buktiPembayaranPath
      );

      if (!result.success) {
        // Delete uploaded file if creation failed
        if (fs.existsSync(buktiPembayaranPath)) {
        fs.unlinkSync(buktiPembayaranPath);
        }
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        message:
          "Payment proof uploaded successfully. Waiting for admin
verification.",
        data: {
          paymentId: result.paymentId,
        },
      });
    } catch (error) {
      console.error("Upload payment proof error:", error);

      // Delete uploaded file if there was an error
      if (req.file && fs.existsSync(req.file.path)) {
      fs.unlinkSync(req.file.path);
      }
```

```
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Update payment proof (User)
static async updatePaymentProof(req, res) {
  try {
    const { paymentId } = req.params;
    if (!req.file) {
      return res.status(400).json({
        success: false,
        message: "Bukti pembayaran is required",
      });
    }
    const buktiPembayaranPath = path
      .join("uploads", "bukti_pembayaran", req.file.filename)
      .replace(/\\/g, "/");
    // Get current payment data to delete old file
    const currentPayment = await Payment.getPaymentById(paymentId);
    if (
      currentPayment.success &&
      currentPayment.data &&
      currentPayment.data.Bukti_Pembayaran
    ) {
      const oldFilePath = currentPayment.data.Bukti_Pembayaran;
      if (fs.existsSync(oldFilePath)) {
        fs.unlinkSync(oldFilePath);
      }
    }
    const result = await Payment.updatePaymentProof(
      paymentId,
      buktiPembayaranPath
    );
    if (!result.success) {
      // Delete uploaded file if update failed
      if (fs.existsSync(buktiPembayaranPath)) {
        fs.unlinkSync(buktiPembayaranPath);
      }
      return res.status(400).json(result);
    }
    res.json({
```

```
      success: true,
      message: "Bukti pembayaran berhasil diupdate",
      data: result.data,
    });
  } catch (error) {
    console.error("Update payment proof error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get unpaid payment by reservation ID
static async getUnpaidPaymentByReservation(req, res) {
  try {
    const { reservationId } = req.params;
    const result = await
Payment.getUnpaidPaymentByReservation(reservationId);
    if (!result.success || !result.data) {
      return res.status(404).json({
        success: false,
        message: "No unpaid payment found for this reservation",
      });
    }
    res.json({
      success: true,
      data: result.data,
    });
  } catch (error) {
    console.error("Get unpaid payment error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Update payment status (Admin only)
static async updatePaymentStatus(req, res) {
try {
    const { paymentId } = req.params;
    const { status } = req.body;
    if (!status) {
      return res.status(400).json({
```

```
        success: false,
        message: "Status pembayaran harus diisi",
      });
    }
    const result = await Payment.updatePaymentStatus(paymentId, status);
    if (!result.success) {
      return res.status(400).json(result);
    }
    res.json({
      success: true,
      message: `Payment status updated to ${status} successfully`,
      data: result.data,
    });
  } catch (error) {
    console.error("Update payment status error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get all pending payments (Admin only)
static async getAllPendingPayments(req, res) {
try {
    const result = await Payment.getAllPendingPayments();

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json({
      success: true,
      data: result.data,
    });
  } catch (error) {
    console.error("Get pending payments error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get payment history for a reservation (User/Admin)
```

```
  static async getPaymentHistory(req, res) {
    try {
      const { reservationId } = req.params;

      const result = await Payment.getPaymentHistory(reservationId);

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        data: result.data,
      });
    } catch (error) {
      console.error("Get payment history error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }

  // Update reservation statuses (Admin/Cron job)
  static async updateReservationStatuses(req, res) {
  try {
      const result = await TmpUser.updateReservationStatus();

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        message: "Reservation statuses updated successfully",
      });
    } catch (error) {
      console.error("Update reservation statuses error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }
```

```
// Mark user as checkout (Admin only)
static async markUserCheckout(req, res) {
try {
   const { reservationId } = req.params;
   const { reason } = req.body;

   const result = await TmpUser.markAsCheckout(reservationId, reason);

   if (!result.success) {
     return res.status(400).json(result);
   }

   res.json({
     success: true,
     message: "User marked as checkout successfully",
   });
 } catch (error) {
   console.error("Mark user checkout error:", error);
   res.status(500).json({
     success: false,
     message: "Internal server error",
   });
 }
}

// Serve payment proof image
static async getPaymentProof(req, res) {
 try {
   const { paymentId } = req.params;

   const result = await Payment.getPaymentById(paymentId);

   if (!result.success) {
     return res.status(404).json(result);
   }

   const payment = result.data;

   if (!payment.Bukti_Pembayaran) {
     return res.status(404).json({
     success: false,
       message: "Payment proof not found",
     });
   }
```

```
    const imagePath = path.resolve(payment.Bukti_Pembayaran);

    if (!fs.existsSync(imagePath)) {
      return res.status(404).json({
      success: false,
        message: "Payment proof file not found",
      });
    }

    res.sendFile(imagePath);
  } catch (error) {
    console.error("Get payment proof error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get user's payment history (User)
static async getMyPaymentHistory(req, res) {
  try {
    const userEmail = req.user?.Email || req.user?.email;
    if (!userEmail) {
      return res.status(401).json({
        success: false,
        message: "Authentication required",
      });
    }

    const result = await Payment.getUserPaymentHistory(userEmail);

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json({
      success: true,
      data: result.data,
    });
  } catch (error) {
    console.error("Get user payment history error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
```

```
    });
  }
}

// Get monthly payment history with filters (User)
static async getMonthlyPaymentHistory(req, res) {
try {
  const userEmail = req.user?.Email || req.user?.email;
  if (!userEmail) {
    return res.status(401).json({
      success: false,
      message: "Authentication required",
    });
  }

  const { year, month, status } = req.query;

  const result = await Payment.getMonthlyPaymentHistory(userEmail, {
    year,
    month,
    status,
  });

  if (!result.success) {
    return res.status(400).json(result);
  }

  res.json({
    success: true,
    data: result.data,
  });
 } catch (error) {
  console.error("Get monthly payment history error:", error);
  res.status(500).json({
    success: false,
    message: "Internal server error",
  });
 }
}

// Create payment (User - legacy support)
static async createPayment(req, res) {
 try {
  const paymentData = req.body;
```

```
    // Add bukti pembayaran file path if uploaded
    if (req.file) {
      paymentData.Bukti_Pembayaran = path
        .join("uploads", "bukti_pembayaran", req.file.filename)
        .replace(/\\/g, "/");
    }

    const result = await Payment.createPaymentLegacy(paymentData);

    if (!result.success) {
      // Delete uploaded file if creation failed
      if (req.file && fs.existsSync(req.file.path)) {
        fs.unlinkSync(req.file.path);
      }
      return res.status(400).json(result);
    }

    res.status(201).json({
      success: true,
      message: "Pembayaran berhasil disubmit! Menunggu verifikasi admin.",
      data: {
        paymentId: result.paymentId,
      },
    });
  } catch (error) {
    console.error("Create payment error:", error);

    // Delete uploaded file if there was an error
    if (req.file && fs.existsSync(req.file.path)) {
    fs.unlinkSync(req.file.path);
    }

    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Generate monthly payments automatically (Admin only)
static async generateMonthlyPayments(req, res) {
  try {
    console.log(
      `[${new Date().toISOString()}] Manual monthly payment generation
triggered by admin`
```

```
    );

    // Get current date info
    const now = new Date();
    const currentMonth = now.getMonth() + 1; // 1-12
    const currentYear = now.getFullYear();

    const result = await Payment.generateMonthlyPaymentsForAllUsers(
      currentMonth,
      currentYear
    );

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json({
      success: true,
      message: `Successfully generated ${result.generated} monthly
payments`,
      data: {
        generated: result.generated,
        errors: result.errors || [],
        period: `${currentMonth}/${currentYear}`,
      },
    });
  } catch (error) {
    console.error("Generate monthly payments error:", error);
    res.status(500).json({
      success: false,
      message: "Failed to generate monthly payments",
      error: error.message,
    });
  }
 }

 // Generate payments for specific month/year (Admin only)
 static async generateManualPayments(req, res) {
  try {
    const { month, year } = req.body;

    if (!month || !year) {
      return res.status(400).json({
        success: false,
        message: "Month and year are required",
```

```
      });
    }

    if (month < 1 || month > 12) {
      return  res.status(400).json({
      success: false,
        message: "Month must be between 1 and 12",
      });
    }

    console.log(
      `Manual payment generation for ${month}/${year} triggered by admin`
    );

    const result = await Payment.generateMonthlyPaymentsForAllUsers(
      month,
      year
    );

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json({
      success: true,
      message: `Successfully generated ${result.generated} payments for
${month}/${year}`,
      data: {
        generated: result.generated,
        errors: result.errors || [],
        period: `${month}/${year}`,
      },
    });
  } catch (error) {
    console.error("Generate manual payments error:", error);
    res.status(500).json({
      success: false,
      message: "Failed to generate manual payments",
      error: error.message,
    });
  }
 }

 // Get payment generation history (Admin only)
 static async getGenerationHistory(req, res) {
```

```
    try {
      const result = await Payment.getGenerationHistory();

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        data: result.data,
      });
    } catch (error) {
      console.error("Get generation history error:", error);
      res.status(500).json({
        success: false,
        message: "Failed to get generation history",
        error: error.message,
      });
    }
  }

  // Verify payment (Admin only)
  static async verifyPayment(req, res) {
  try {
    const { paymentId } = req.params;
    const { status = "Diterima" } = req.body;

    const result = await Payment.updatePaymentStatus(paymentId, status);

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json({
      success: true,
      message: "Payment verified successfully",
      data: result.data,
    });
  } catch (error) {
    console.error("Verify payment error:", error);
    res.status(500).json({
      success: false,
      message: "Failed to verify payment",
      error: error.message,
    });
```

```
    }
  }
}

module.exports = PaymentController;
```

Tabel 4. 8 Codingan PendingReservasiController.js

PendingReservasiController.js

```
const PendingReservation = require("../models/PendingReservation");
const UserToken = require("../models/UserToken");
const path = require("path");
const fs = require("fs");

class PendingReservasiController {
  // Create new pending reservation with payment proof
  static async create(req, res) {
    try {
      const pendingData = req.body;

      // Check if bukti pembayaran file was uploaded
      if (!req.file) {
        return res.status(400).json({
          success: false,
          message: "Bukti pembayaran is required",
        });
      }

      const buktiPembayaranPath = req.file.path;

      const result = await PendingReservation.create(
        pendingData,
        buktiPembayaranPath
      );

      if (!result.success) {
        // Delete uploaded file if reservation creation failed
        if (fs.existsSync(buktiPembayaranPath)) {
          fs.unlinkSync(buktiPembayaranPath);
        }
        return res.status(400).json(result);
      }

      res.status(201).json({
        success: true,
```

```
      message:
        "Reservasi berhasil dikirim! Silakan menunggu konfirmasi admin.",
      data: {
        pendingId: result.pendingId,
      },
    });
  } catch (error) {
    console.error("Create pending reservation error:", error);

    // Delete uploaded file if there was an error
    if (req.file && fs.existsSync(req.file.path)) {
    fs.unlinkSync(req.file.path);
    }

    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get all pending reservations (Admin only)
static async getAll(req, res) {
  try {
    const pendingReservations = await PendingReservation.getAll();

    res.json({
      success: true,
      data: pendingReservations,
    });
  } catch (error) {
    console.error("Get all pending reservations error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get pending reservation by ID (Admin only)
static async getById(req, res) {
  try {
    const { id } = req.params;
    const pendingReservation = await PendingReservation.getById(id);
```

```
    if (!pendingReservation) {
      return res.status(404).json({
      success: false,
        message: "Pending reservation not found",
      });
    }

    res.json({
      success: true,
      data: pendingReservation,
    });
  } catch (error) {
    console.error("Get pending reservation by ID error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Approve pending reservation (Admin only)
static async approve(req, res) {
  try {
    const { id } = req.params;

    const result = await PendingReservation.approve(id);

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json({
      success: true,
      message:
        "Reservasi berhasil disetujui! User dapat login menggunakan token
yang telah diberikan.",
      data: result.data,
    });
  } catch (error) {
    console.error("Approve pending reservation error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
```

```
  }

  // Reject pending reservation (Admin only)
  static async reject(req, res) {
    try {
      const { id } = req.params;

      const result = await PendingReservation.reject(id);

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        message: "Reservasi telah ditolak.",
      });
    } catch (error) {
      console.error("Reject pending reservation error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }

  // Delete pending reservation (Admin only)
  static async delete(req, res) {
    try {
      const { id } = req.params;

      // Get pending reservation to delete associated file
      const pendingReservation = await PendingReservation.getById(id);

      const result = await PendingReservation.delete(id);

      if (!result.success) {
        return res.status(404).json(result);
      }

      // Delete associated bukti pembayaran file
      if (pendingReservation && pendingReservation.Bukti_Pembayaran) {
        const filePath = pendingReservation.Bukti_Pembayaran;
        if (fs.existsSync(filePath)) {
          fs.unlinkSync(filePath);
```

```
      }
    }

    res.json({
      success: true,
      message: "Pending reservation deleted successfully",
    });
  } catch (error) {
    console.error("Delete pending reservation error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Serve bukti pembayaran image (Admin only)
static async getBuktiPembayaran(req, res) {
  try {
    const { id } = req.params;

    const pendingReservation = await PendingReservation.getById(id);

    if (!pendingReservation || !pendingReservation.Bukti_Pembayaran) {
      return res.status(404).json({
        success: false,
        message: "Bukti pembayaran not found",
      });
    }

    const filePath = pendingReservation.Bukti_Pembayaran;

    if (!fs.existsSync(filePath)) {
      return res.status(404).json({
        success: false,
        message: "File not found",
      });
    }

    res.sendFile(path.resolve(filePath));
  } catch (error) {
    console.error("Get bukti pembayaran error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
```

```
    });
   }
  }
}

module.exports = PendingReservasiController;
```

Tabel 4. 9 Codingan ReservasiController.js

ReservasiController.js

```
const Reservasi = require("../models/Reservasi");

class ReservasiController {
 // Create new reservation
 static async create(req, res) {
 try {
    const reservationData = {
     ...req.body,
     Email: req.user.Email, // Use authenticated user's email
    };

    const result = await Reservasi.create(reservationData);

    if (!result.success) {
     return res.status(400).json(result);
    }

    res.status(201).json({
     success: true,
     message: "Reservation created successfully",
     data: {
      reservationId: result.reservationId,
     },
    });
  } catch (error) {
    console.error("Create reservation error:", error);
    res.status(500).json({
     success: false,
     message: "Internal server error",
    });
  }
 }

 // Get all reservations with payment info (Admin only)
 static async getReservationsWithPayments(req, res) {
```

```
    try {
      const { status, kamar, periode } = req.query;

      const filters = {};
      if (status) filters.status = status;
      if (kamar) filters.kamar = kamar;
      if (periode) filters.periode = periode;

      const reservations = await Reservasi.getAllWithPayments(filters);

      // Process reservations to update status based on payment status
      const processedReservations = reservations.map((reservation) => {
        // If latest payment status is not 'Diterima', set reservation status to
'Telat/Belum Bayar'
        // and clear the payment period display
        if (reservation.latest_payment_status !== "Diterima") {
          reservation.Status = "Telat/Belum Bayar";
          reservation.last_payment_period = null; // This will show "belum ada
pembayaran"
        } else if (
          reservation.latest_payment_status === "Diterima" &&
          reservation.Status !== "Aktif/Lunas"
        ) {
          // If payment is accepted but reservation status is not updated, update
it
          reservation.Status = "Aktif/Lunas";
        }

        return reservation;
      });

      res.json({
        success: true,
        data: processedReservations,
      });
    } catch (error) {
      console.error("Get reservations with payments error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }

  // Update reservation status (Admin only)
```

```
  static async updateStatus(req, res) {
    try {
      const { id } = req.params;
      const { status, keterangan } = req.body;

      const result = await Reservasi.updateStatus(id, status, keterangan);

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        message: "Reservation status updated successfully",
      });
    } catch (error) {
      console.error("Update reservation status error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }

  // Get user's reservations
  static async getUserReservations(req, res) {
    try {
      const reservations = await Reservasi.getByUser(req.user.Email);

      res.json({
        success: true,
        data: reservations,
      });
    } catch (error) {
      console.error("Get user reservations error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }

  // Get reservations by user email (for admin or user themselves)
  static async getReservasiByUser(req, res) {
    try {
```

```
    const { userID } = req.params;

    // Check if user is admin or accessing their own data
    // Since we use Email as identifier, userID here should be an email
    if (!req.user.isAdmin && req.user.Email !== userID) {
      return res.status(403).json({
        success: false,
        message: "Access denied. You can only view your own reservations.",
      });
    }

    // Get reservations with payment info and calculated status
    const reservations = await Reservasi.getByUserIdWithPayments(userID);

    res.json({
      success: true,
      data: reservations,
    });
  } catch (error) {
    console.error("Get reservations by user email error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get all reservations (Admin only)
static async getAll(req, res) {
  try {
    const reservations = await Reservasi.getAll();

    res.json({
      success: true,
      data: reservations,
    });
  } catch (error) {
    console.error("Get all reservations error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}
```

```
  // Update reservation status (Admin only)
  static async updateStatus(req, res) {
    try {
      const { id } = req.params;
      const { status } = req.body;

      // Validate status
      const validStatuses = ["Menunggu", "Diterima", "Ditolak"];
      if (!validStatuses.includes(status)) {
        return res.status(400).json({
          success: false,
          message:
            "Invalid status. Must be one of: Menunggu, Diterima, Ditolak",
        });
      }

      const result = await Reservasi.updateStatus(id, status);

      if (!result.success) {
        return res.status(404).json(result);
      }

      res.json(result);
    } catch (error) {
      console.error("Update reservation status error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }
}

module.exports = ReservasiController;
```

Tabel 4. 10 Codingan TmpUserController.js

	TmpUserController.js

	const TmpUser = require("../models/TmpUser");
	const UserToken = require("../models/UserToken");
	const path = require("path");
	const fs = require("fs");
	
	class TmpUserController {
	  // Create new tmp user with payment proof

```
static async create(req, res) {
  try {
    const tmpData = req.body;

    // Check if bukti pembayaran file was uploaded
    if (!req.file) {
      return res.status(400).json({
        success: false,
        message: "Bukti pembayaran is required",
      });
    }

    const buktiPembayaranPath = req.file.path;

    const result = await TmpUser.create(tmpData, buktiPembayaranPath);

    if (!result.success) {
      // Delete uploaded file if creation failed
      if (fs.existsSync(buktiPembayaranPath)) {
      fs.unlinkSync(buktiPembayaranPath);
      }
      return res.status(400).json(result);
    }

    res.status(201).json({
      success: true,
      message:
        "Reservasi berhasil dikirim! Silakan menunggu konfirmasi admin.",
      data: {
        tmpId: result.tmpId,
      },
    });
  } catch (error) {
    console.error("Create tmp user error:", error);

    // Delete uploaded file if there was an error
    if (req.file && fs.existsSync(req.file.path)) {
    fs.unlinkSync(req.file.path);
    }

    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
```

```
  }

  // Get all tmp users with reservation status (Admin only)
  static async getAll(req, res) {
    try {
      const tmpUsers = await TmpUser.getAllWithReservationStatus();

      res.json({
        success: true,
        data: tmpUsers,
      });
    } catch (error) {
      console.error("Get all tmp users error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }

  // Get tmp user by ID (Admin only)
  static async getById(req, res) {
    try {
      const { id } = req.params;
      const tmpUser = await TmpUser.getByIdWithReservationStatus(id);

      if (!tmpUser) {
        return res.status(404).json({
          success: false,
          message: "Temporary user not found",
        });
      }

      res.json({
        success: true,
        data: tmpUser,
      });
    } catch (error) {
      console.error("Get tmp user by ID error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }
```

```
  // Approve tmp user (Admin only)
  static async approve(req, res) {
    try {
      const { id } = req.params;

      const result = await TmpUser.approve(id);

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        message:
          "Reservasi berhasil disetujui! User telah dibuat, token login telah
disiapkan, dan catatan pembayaran telah dibuat.",
        data: result.data,
      });
    } catch (error) {
      console.error("Approve tmp user error:", error);
      res.status(500).json({
        success: false,
        message: "Internal server error",
      });
    }
  }

  // Reject tmp user (Admin only)
  static async reject(req, res) {
    try {
      const { id } = req.params;

      const result = await TmpUser.reject(id);

      if (!result.success) {
        return res.status(400).json(result);
      }

      res.json({
        success: true,
        message: "User telah ditolak.",
      });
    } catch (error) {
      console.error("Reject tmp user error:", error);
```

```
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Serve bukti pembayaran image (Admin only)
static async getBuktiPembayaran(req, res) {
  try {
    const { id } = req.params;

    const tmpUser = await TmpUser.getByIdWithReservationStatus(id);

    if (!tmpUser || !tmpUser.Bukti_Pembayaran) {
      console.log("Bukti pembayaran not found for ID:", id);
      console.log("TmpUser data:", tmpUser);
      return res.status(404).json({
        success: false,
        message: "Bukti pembayaran not found",
      });
    }

    const filePath = tmpUser.Bukti_Pembayaran;
    console.log("Trying to serve file:", filePath);

    if (!fs.existsSync(filePath)) {
      console.log("File does not exist:", filePath);
      return res.status(404).json({
        success: false,
        message: "File not found",
      });
    }

    console.log("Serving file:", filePath);
    res.sendFile(path.resolve(filePath));
  } catch (error) {
    console.error("Get bukti pembayaran error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}
```

```
// Get all approved users/reservations from User table (Admin only)
static async getAllReservations(req, res) {
 try {
  const {
  status,
  kamar,
   search,
   sortBy = "Tanggal_Reservasi",
   sortOrder = "DESC",
  } = req.query;

  const reservations = await TmpUser.getAllReservationsWithFilters({
   status,
   kamar,
   search,
   sortBy,
   sortOrder,
  });

  res.json({
   success: true,
   data: reservations,
   total: reservations.length,
  });
 } catch (error) {
  console.error("Get all reservations error:", error);
  res.status(500).json({
   success: false,
   message: "Internal server error",
  });
 }
}

// Update reservation status (Admin only)
static async updateReservationStatus(req, res) {
try {
  const { id } = req.params;
  const { status, notes } = req.body;

  const validStatuses = [
   "Menunggu",
   "Diterima",
   "Ditolak",
   "Aktif/Lunas",
   "Telat/Belum Bayar",
```

```
      "Keluar",
    ];
    if (!validStatuses.includes(status)) {
      return res.status(400).json({
      success: false,
        message:
          "Invalid status. Valid statuses: " + validStatuses.join(", "),
      });
    }

    const result = await TmpUser.updateReservationStatus(id, status, notes);

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json({
      success: true,
      message: `Status reservasi berhasil diubah menjadi ${status}`,
      data: result.data,
    });
  } catch (error) {
    console.error("Update reservation status error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Get filter options for admin (kamar list, status options)
static async getFilterOptions(req, res) {
  try {
    const filterOptions = await TmpUser.getFilterOptions();

    res.json({
      success: true,
      data: filterOptions,
    });
  } catch (error) {
    console.error("Get filter options error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
```

```
    }
  }
}

module.exports = TmpUserController;
```

Tabel 4. 11 Codingan UserController.js

UserController.js

```
const User = require("../models/User");

class UserController {
  // Get all users (Admin only)
  static async getAllUsers(req, res) {
  try {
     const users = await User.getAllUsers();

     res.json({
       success: true,
       data: users,
     });
   } catch (error) {
     console.error("Get all users error:", error);
     res.status(500).json({
       success: false,
       message: "Internal server error",
     });
   }
  }

  // Get user by email (Admin only)
  static async getUserByEmail(req, res) {
  try {
     const { email } = req.params;
     const user = await User.findByEmail(email);

     if (!user) {
       return res.status(404).json({
         success: false,
         message: "User not found",
       });
     }

     // Remove password from response
     delete user.Password;
```

```
    res.json({
      success: true,
      data: user,
    });
  } catch (error) {
    console.error("Get user by email error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Create new user (Admin only)
static async createUser(req, res) {
try {
    const { Nama, Email, Password, No_telp, Alamat, Role } = req.body;

    if (!Nama || !Email || !Password || !No_telp || !Alamat) {
      return res.status(400).json({
        success: false,
        message: "Semua field harus diisi",
      });
    }

    // Validate email format
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(Email)) {
      return res.status(400).json({
        success: false,
        message: "Format email tidak valid",
      });
    }

    // Validate phone number format (Indonesian)
    const phoneRegex = /^(08|62)\d{8,13}$/;
    if (!phoneRegex.test(No_telp)) {
      return res.status(400).json({
      success: false,
        message: "Format nomor telepon tidak valid",
      });
    }

    const result = await User.createByAdmin({
```

```
      Nama,
      Email,
      Password,
      No_telp,
      Alamat,
      Role: Role || "penyewa",
    });

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.status(201).json(result);
  } catch (error) {
    console.error("Create user error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Update user (Admin only)
static async updateUser(req, res) {
  try {
    const { email } = req.params;
    const { Nama, No_telp, Alamat, Role } = req.body;

    if (!Nama || !No_telp || !Alamat) {
      return res.status(400).json({
      success: false,
        message: "Nama, nomor telepon, dan alamat harus diisi",
      });
    }

    // Validate phone number format (Indonesian)
    const phoneRegex = /^(08|62)\d{8,13}$/;
    if (!phoneRegex.test(No_telp)) {
      return res.status(400).json({
      success: false,
        message: "Format nomor telepon tidak valid",
      });
    }

    const result = await User.updateUser(email, {
```

```
      Nama,
      No_telp,
      Alamat,
      Role: Role || "penyewa",
    });

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json(result);
  } catch (error) {
    console.error("Update user error:", error);
    res.status(500).json({
      success: false,
      message: "Internal server error",
    });
  }
}

// Delete user (Admin only)
static async deleteUser(req, res) {
  try {
    const { email } = req.params;

    // Prevent admin from deleting themselves
    const currentUserEmail = req.user?.Email || req.user?.email;
    if (email === currentUserEmail) {
      return res.status(400).json({
        success: false,
        message: "Tidak dapat menghapus akun sendiri",
      });
    }

    const result = await User.deleteUser(email);

    if (!result.success) {
      return res.status(400).json(result);
    }

    res.json(result);
  } catch (error) {
    console.error("Delete user error:", error);
    res.status(500).json({
      success: false,
```

```
      message: "Internal server error",
    });
  }
 }
}

module.exports = UserController;
```

Tabel 4. 12 Codingan Kamar.js

Kamar.js

```
const { pool } = require("../config/database");

class Kamar {
 // Get all rooms
 static async getAll() {
 try {
    const query = "SELECT * FROM kamar ORDER BY No_Kamar";
    const [rows] = await pool.execute(query);
    return rows;
  } catch (error) {
    throw error;
  }
 }

 // Get available rooms
 static async getAvailable() {
 try {
    const query =
      "SELECT * FROM kamar WHERE Ketersediaan = 1 ORDER BY
No_Kamar";
    const [rows] = await pool.execute(query);
    return rows;
  } catch (error) {
    throw error;
  }
 }

 // Get room by ID
 static async getById(roomId) {
  try {
    const query = "SELECT * FROM kamar WHERE No_Kamar = ?";
    const [rows] = await pool.execute(query, [roomId]);
    return rows[0] || null;
  } catch (error) {
```

```
    throw error;
   }
  }

  // Update room availability
  static async updateAvailability(roomId, availability) {
   try {
    const query = "UPDATE kamar SET Ketersediaan = ? WHERE No_Kamar =
?";
    const [result] = await pool.execute(query, [availability, roomId]);

    return {
     success: result.affectedRows > 0,
     message:
      result.affectedRows > 0
       ? "Room availability updated"
       : "Room not found",
    };
   } catch (error) {
    throw error;
   }
  }

  // Create new room
  static async create(roomData) {
   try {
    const { Nama_Kamar, Letak, Ketersediaan = 1 } = roomData;

    const query = `
     INSERT INTO kamar (Nama_Kamar, Letak, Ketersediaan)
     VALUES (?, ?, ?)
    `;

    const [result] = await pool.execute(query, [
     Nama_Kamar,
     Letak,
     Ketersediaan,
    ]);

    return {
     success: true,
     message: "Room created successfully",
     roomId: result.insertId,
     data: {
      No_Kamar: result.insertId,
```

```
        Nama_Kamar,
        Letak,
        Ketersediaan,
      },
    };
  } catch (error) {
    if (error.code === "ER_DUP_ENTRY") {
      return {
        success: false,
        message: "Room name already exists",
      };
    }
    return {
      success: false,
      message: "Failed to create room: " + error.message,
    };
  }
}

// Update room data
static async update(roomId, roomData) {
  try {
    const { Nama_Kamar, Letak, Ketersediaan } = roomData;

    const query = `
      UPDATE kamar
      SET Nama_Kamar = ?, Letak = ?, Ketersediaan = ?
      WHERE No_Kamar = ?
    `;

    const [result] = await pool.execute(query, [
      Nama_Kamar,
      Letak,
      Ketersediaan,
      roomId,
    ]);

    if (result.affectedRows === 0) {
      return {
        success: false,
        message: "Room not found",
      };
    }

    return {
```

```
      success: true,
      message: "Room updated successfully",
      data: {
        No_Kamar: roomId,
        Nama_Kamar,
        Letak,
        Ketersediaan,
      },
    };
  } catch (error) {
    if (error.code === "ER_DUP_ENTRY") {
      return {
        success: false,
        message: "Room name already exists",
      };
    }
    return {
      success: false,
      message: "Failed to update room: " + error.message,
    };
  }
}

// Delete room
static async delete(id) {
  try {
    // Check if room has active reservations
    const checkQuery = `
      SELECT COUNT(*) as count
      FROM reservasi
      WHERE No_Kamar = ? AND Status IN ('Telat/Belum Bayar', 'Aktif/Lunas')
    `;
    const [checkResult] = await pool.execute(checkQuery, [id]);

    if (checkResult[0].count > 0) {
      return {
        success: false,
        message: "Cannot delete room with active reservations",
      };
    }

    // Delete room
    const query = "DELETE FROM kamar WHERE No_Kamar = ?";
    const [result] = await pool.execute(query, [id]);
```

```
    if (result.affectedRows === 0) {
      return {
        success: false,
        message: "Room not found",
      };
    }

    return {
      success: true,
      message: "Room deleted successfully",
    };
  } catch (error) {
    console.error("Delete room error:", error);
    return {
      success: false,
      message: "Failed to delete room: " + error.message,
    };
  }
 }
}

module.exports = Kamar;
```

Tabel 4. 13 Payment.js

	Payment.js

```
const { pool } = require("../config/database");

class Payment {
 // Create new payment record
 static async createPayment(paymentData) {
  try {
    const {
      ID_Reservasi,
      Tanggal_Bayar,
      Jumlah,
      Jumlah_Bayar, // Alternative field name
      Bukti_Pembayaran,
      bukti_pembayaran, // Alternative field name
      Status = "Menunggu Verifikasi",
      periode,
    } = paymentData;

    // Validate required fields and handle undefined values
    if (!ID_Reservasi) {
```

```
  return {
    success: false,
    message: "ID_Reservasi is required",
  };
}

// Handle amount field (could be Jumlah or Jumlah_Bayar)
const amount = Jumlah || Jumlah_Bayar || null;
if (!amount) {
  return {
    success: false,
    message: "Payment amount is required",
  };
}

// Handle bukti pembayaran field (could be different names)
const buktiPath = Bukti_Pembayaran || bukti_pembayaran || null;

// Handle date - use current date if not provided
const paymentDate =
  Tanggal_Bayar || new Date().toISOString().split("T")[0];

// Generate periode if not provided
const currentDate = new Date();
const monthNames = [
  "Januari",
  "Februari",
  "Maret",
  "April",
  "Mei",
  "Juni",
  "Juli",
  "Agustus",
  "September",
  "Oktober",
  "November",
  "Desember",
];
const defaultPeriode = `${
  monthNames[currentDate.getMonth()]
} ${currentDate.getFullYear()}`;
const paymentPeriode = periode || defaultPeriode;

const query = `
  INSERT INTO pembayaran (
```

```
        ID_Reservasi,
        Tanggal_Bayar,
        Jumlah,
        Bukti_Pembayaran,
        Status
      ) VALUES (?, ?, ?, ?, ?)
    `;

    const [result] = await pool.execute(query, [
      ID_Reservasi,
      paymentDate,
      amount,
      buktiPath,
      Status,
    ]);

    return {
      success: true,
      paymentId: result.insertId,
      message: "Payment record created successfully",
    };
  } catch (error) {
    console.error("Create payment error:", error);
    return {
      success: false,
      message: error.message,
    };
  }
 }

 // Update payment status (for admin verification)
 static async updatePaymentStatus(paymentId, status, buktiPembayaran =
null) {
  try {
    let query, params;

    if (buktiPembayaran) {
      query = `
        UPDATE pembayaran
        SET Status = ?, Bukti_Pembayaran = ?, Tanggal_Bayar = NOW()
        WHERE ID_Pembayaran = ?
      `;
      params = [status, buktiPembayaran, paymentId];
    } else {
      query = `
```

```
      UPDATE pembayaran
      SET Status = ?
      WHERE ID_Pembayaran = ?
    `;
    params = [status, paymentId];
  }

  const [result] = await pool.execute(query, params);

  if (result.affectedRows === 0) {
    return {
      success: false,
      message: "Payment not found",
    };
  }

  // If payment is marked as 'Diterima', update reservation status
  if (status === "Diterima") {
    const [payment] = await pool.execute(
      `
      SELECT ID_Reservasi FROM pembayaran WHERE ID_Pembayaran = ?
    `,
      [paymentId]
    );

    if (payment.length > 0) {
      await pool.execute(
        `
        UPDATE reservasi
        SET Status = 'Aktif/Lunas'
        WHERE ID_Reservasi = ?
      `,
        [payment[0].ID_Reservasi]
      );
    }
  }

  return {
    success: true,
    message: `Payment status updated to ${status}`,
  };
} catch (error) {
  return {
    success: false,
    message: error.message,
```

```
    };
  }
}

// Get payment history for a reservation
static async getPaymentHistory(reservationId) {
  try {
    const query = `
      SELECT p.*, r.Email, u.Nama
      FROM pembayaran p
      JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
      JOIN user u ON r.Email = u.Email
      WHERE p.ID_Reservasi = ?
      ORDER BY p.Created_At DESC
    `;

    const [payments] = await pool.execute(query, [reservationId]);
    return {
      success: true,
      data: payments,
    };
  } catch (error) {
    return {
      success: false,
      message: error.message,
    };
  }
}

// Get all payments for admin
static async getAllPayments() {
try {
    const query = `
      SELECT
        p.*,
        r.Email,
        r.No_Kamar,
        u.Nama,
        u.No_telp,
        k.Nama_Kamar,
        MONTH(p.Created_At) as periode_bulan,
        YEAR(p.Created_At) as periode_tahun
      FROM pembayaran p
      JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
      JOIN user u ON r.Email = u.Email
```

```
      JOIN kamar k ON r.No_Kamar = k.No_Kamar
      ORDER BY p.Created_At DESC
    `;

    const [payments] = await pool.execute(query);
    return {
      success: true,
      data: payments,
    };
  } catch (error) {
    return {
      success: false,
      message: error.message,
    };
  }
}

// Get all pending payments for admin
static async getAllPendingPayments() {
try {
    const query = `
      SELECT
        p.*,
        r.Email,
        r.No_Kamar,
        u.Nama,
        k.Nama_Kamar,
        MONTH(p.Created_At) as periode_bulan,
        YEAR(p.Created_At) as periode_tahun
      FROM pembayaran p
      JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
      JOIN user u ON r.Email = u.Email
      JOIN kamar k ON r.No_Kamar = k.No_Kamar
      WHERE p.Status = 'Menunggu'
      ORDER BY p.Created_At DESC
    `;

    const [payments] = await pool.execute(query);
    return {
      success: true,
      data: payments,
    };
  } catch (error) {
    return {
      success: false,
```

```
      message: error.message,
    };
  }
}

// Get payment by ID
static async getPaymentById(paymentId) {
  try {
    const query = `
      SELECT
        p.*,
        r.Email,
        r.No_Kamar,
        u.Nama,
        k.Nama_Kamar
      FROM pembayaran p
      JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
      JOIN user u ON r.Email = u.Email
      JOIN kamar k ON r.No_Kamar = k.No_Kamar
      WHERE p.ID_Pembayaran = ?
    `;

    const [payments] = await pool.execute(query, [paymentId]);

    if (payments.length === 0) {
      return {
        success: false,
        message: "Payment not found",
      };
    }

    return {
      success: true,
      data: payments[0],
    };
  } catch (error) {
    return {
      success: false,
      message: error.message,
    };
  }
}

// Penghuni: Update payment proof, status, and payment date
static async updatePaymentProof(paymentId, buktiPath) {
```

```
    try {
      const updateQuery = `
        UPDATE pembayaran
        SET Bukti_Pembayaran = ?,
            Status = 'Menunggu',
            Tanggal_Bayar = CURRENT_DATE,
            updated_at = CURRENT_TIMESTAMP
        WHERE ID_Pembayaran = ?
      `;
      const [result] = await pool.execute(updateQuery, [buktiPath,
paymentId]);
      if (result.affectedRows === 0) {
        return {
          success: false,
          message: "Payment not found or no changes made",
        };
      }
      const updatedPayment = await this.getPaymentById(paymentId);
      return {
        success: true,
        message: "Payment proof updated successfully",
        data: updatedPayment.data,
      };
    } catch (error) {
      console.error("Error updating payment proof:", error);
      return {
        success: false,
        message: "Failed to update payment proof",
      };
    }
  }

  // Admin: Update payment status only
  static async updatePaymentStatus(paymentId, status) {
    try {
      const updateQuery = `
        UPDATE pembayaran
        SET Status = ?,
            updated_at = CURRENT_TIMESTAMP
        WHERE ID_Pembayaran = ?
      `;
      const [result] = await pool.execute(updateQuery, [status, paymentId]);
      if (result.affectedRows === 0) {
        return {
          success: false,
```

```
        message: "Payment not found or no changes made",
      };
    }
    const updatedPayment = await this.getPaymentById(paymentId);
    return {
      success: true,
      message: `Payment status updated to ${status}`,
      data: updatedPayment.data,
    };
  } catch (error) {
    console.error("Error updating payment status:", error);
    return {
      success: false,
      message: "Failed to update payment status",
    };
  }
}

// Get unpaid payment by reservation ID
static async getUnpaidPaymentByReservation(reservationId) {
  try {
    const query = `
      SELECT
        p.*,
        r.Email,
        r.No_Kamar,
        u.Nama,
        k.Nama_Kamar
      FROM pembayaran p
      JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
      JOIN user u ON r.Email = u.Email
      JOIN kamar k ON r.No_Kamar = k.No_Kamar
      WHERE p.ID_Reservasi = ?
      AND p.Status = 'Belum Bayar'
      ORDER BY p.created_at DESC
      LIMIT 1
    `;

    const [payments] = await pool.execute(query, [reservationId]);

    if (payments.length === 0) {
      return {
        success: false,
        message: "No unpaid payment found for this reservation",
      };
```

```
    }

    return {
      success: true,
      data: payments[0],
    };
  } catch (error) {
    console.error("Error getting unpaid payment:", error);
    return {
      success: false,
      message: "Failed to get unpaid payment",
    };
  }
}

// Create monthly payment record for user
static async createMonthlyPayment(reservationId, buktiPembayaran) {
  try {
    const currentDate = new Date();

    // Check if payment already exists for current month
    const [existing] = await pool.execute(
      `
      SELECT * FROM pembayaran
      WHERE ID_Reservasi = ?
      AND YEAR(Created_At) = ?
      AND MONTH(Created_At) = ?
    `,
      [reservationId, currentDate.getFullYear(), currentDate.getMonth() + 1]
    );

    if (existing.length > 0) {
      // Update existing payment
      const [result] = await pool.execute(
        `
        UPDATE pembayaran
        SET Bukti_Pembayaran = ?, Status = 'Menunggu', Tanggal_Bayar =
NOW()
        WHERE ID_Pembayaran = ?
      `,
        [buktiPembayaran, existing[0].ID_Pembayaran]
      );

      return {
        success: true,
```

```
        paymentId: existing[0].ID_Pembayaran,
        message: "Payment proof uploaded and status updated to pending",
       };
     } else {
      // Create new payment
      return await this.createPayment({
       ID_Reservasi: reservationId,
       Tanggal_Bayar: new Date(),
       Jumlah: 900000,
       Bukti_Pembayaran: buktiPembayaran,
       Status: "Menunggu",
      });
     }
   } catch (error) {
     return {
      success: false,
      message: error.message,
     };
   }
 }

 // Get user payment history
 static async getUserPaymentHistory(userEmail) {
   try {
     const query = `
       SELECT
         p.ID_Pembayaran,
         p.ID_Reservasi,
         YEAR(p.Created_At) as Periode_Tahun,
         MONTH(p.Created_At) as Periode_Bulan,
         p.Tanggal_Bayar,
         p.Jumlah,
         p.Status,
         r.No_Kamar,
         k.Nama_Kamar,
         u.Nama as Nama_Penyewa,
         r.Tanggal_Reservasi,
         MONTHNAME(p.Created_At) as nama_bulan
       FROM pembayaran p
       JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
       JOIN kamar k ON r.No_Kamar = k.No_Kamar
       JOIN user u ON r.Email = u.Email
       WHERE r.Email = ?
       ORDER BY Periode_Tahun DESC, Periode_Bulan DESC, p.Tanggal_Bayar
DESC
```

```
      LIMIT 12
    `;

    const [rows] = await pool.execute(query, [userEmail]);

    return {
      success: true,
      data: rows,
    };
  } catch (error) {
    return {
      success: false,
      message: error.message,
    };
  }
}

// Get monthly payment history with filters
static async getMonthlyPaymentHistory(userEmail, filters = {}) {
  try {
    const { year, month, status } = filters;

    let query = `
      SELECT
        p.ID_Pembayaran as id,
        p.ID_Reservasi as reservasiId,
        YEAR(p.Created_At) as tahun,
        MONTH(p.Created_At) as bulan,
        p.Tanggal_Bayar as tanggalBayar,
        p.Jumlah as jumlah,
        p.Status as status,
        r.No_Kamar as nomorKamar,
        k.Nama_Kamar as namaKamar,
        u.Nama as namaPenyewa,
        r.Tanggal_Reservasi as tanggalReservasi,
        MONTHNAME(p.Created_At) as namaBulan,
        CASE
          WHEN p.Status = 'Diterima' THEN 'lunas'
          WHEN p.Status = 'Belum Bayar' THEN 'belum'
          WHEN p.Status = 'Menunggu' THEN 'pending'
          ELSE LOWER(p.Status)
        END as statusFormatted
      FROM pembayaran p
      JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
      JOIN kamar k ON r.No_Kamar = k.No_Kamar
```

```
  JOIN user u ON r.Email = u.Email
  WHERE r.Email = ?
`;

const params = [userEmail];

// Add filters
if (year) {
  query += ` AND YEAR(p.Created_At) = ?`;
  params.push(year);
}

if (month) {
  query += ` AND MONTH(p.Created_At) = ?`;
  params.push(month);
}

if (status) {
  const statusMap = {
    lunas: "Diterima",
    belum: "Belum Bayar",
    pending: "Menunggu",
  };
  if (statusMap[status]) {
    query += ` AND p.Status = ?`;
    params.push(statusMap[status]);
  }
}

query += ` ORDER BY tahun DESC, bulan DESC, p.Tanggal_Bayar DESC`;

const [rows] = await pool.execute(query, params);

// Transform data to match frontend format
const transformedRows = rows.map((row) => {
return {
   id: row.id,
   periode: row.namaBulan
     ? `${row.namaBulan} ${row.tahun}`
     : `${row.bulan}/${row.tahun}`,
   bulan: row.bulan ? row.bulan.toString().padStart(2, "0") : "00",
   tahun: row.tahun ? row.tahun.toString() : "0000",
   jumlah: parseFloat(row.jumlah || 0),
   status: row.statusFormatted,
   tanggalBayar: row.tanggalBayar
```

```
          ? new Date(row.tanggalBayar).toISOString().split("T")[0]
          : null,
        tanggalJatuhTempo: row.tanggalJatuhTempo
          ? new Date(row.tanggalJatuhTempo).toISOString().split("T")[0]
          : null,
        metodePembayaran: row.metodePembayaran || "Transfer Bank",
        nomorReferensi:
          row.nomorReferensi ||
          `TRX${row.tahun || "2025"}${(row.bulan || 1)
            .toString()
            .padStart(2, "0")}${row.id.toString().padStart(6, "0")}`,
        namaKamar: row.namaKamar,
        nomorKamar: row.nomorKamar,
      };
    });

    return {
      success: true,
      data: transformedRows,
    };
  } catch (error) {
    return {
      success: false,
      message: error.message,
    };
  }
}

// Legacy payment creation (for backwards compatibility)
static async createPaymentLegacy(paymentData) {
  try {
    // This is a wrapper for legacy createPayment calls
    return await this.createPayment(paymentData);
  } catch (error) {
    return {
      success: false,
      message: error.message,
    };
  }
}

// Get payment proof file path
static async getPaymentProof(paymentId) {
  try {
    const query = `
```

```
      SELECT Bukti_Pembayaran
      FROM pembayaran
      WHERE ID_Pembayaran = ?
    `;

    const [rows] = await pool.execute(query, [paymentId]);

    if (rows.length === 0) {
      return {
        success: false,
        message: "Payment not found",
      };
    }

    if (!rows[0].Bukti_Pembayaran) {
      return {
        success: false,
        message: "Payment proof not found",
      };
    }

    return {
      success: true,
      filePath: rows[0].Bukti_Pembayaran,
    };
  } catch (error) {
    return {
      success: false,
      message: error.message,
    };
  }
}

// Generate monthly payments for all active users
static async generateMonthlyPaymentsForAllUsers(month, year) {
  try {
    console.log(`Starting payment generation for ${month}/${year}`);

    // Get all active tenants with reservation details
    const getUsersQuery = `
      SELECT DISTINCT
        u.Nama,
        u.Email,
        r.ID_Reservasi,
        k.Nama_Kamar
```

```
      FROM user u
      JOIN reservasi r ON u.Email = r.Email
      JOIN kamar k ON r.No_Kamar = k.No_Kamar
      WHERE u.Role = 'penyewa'
      AND r.Status != 'Keluar'
      AND r.Status = 'Aktif/Lunas'
    `;

    const [users] = await pool.execute(getUsersQuery);
    console.log(`Found ${users.length} active tenants`);

    if (users.length === 0) {
      return {
        success: true,
        message: "No active tenants found",
        generated: 0,
        errors: [],
      };
    }

    let generatedCount = 0;
    let errors = [];

    // Generate payment for each active user
    for (const user of users) {
      try {
        // Check if payment already exists for this period using DATE functions
on Created_At
        const checkExistingQuery = `
          SELECT ID_Pembayaran
          FROM pembayaran
          WHERE ID_Reservasi = ?
          AND MONTH(Created_At) = ?
          AND YEAR(Created_At) = ?
        `;

        const [existing] = await pool.execute(checkExistingQuery, [
          user.ID_Reservasi,
          month,
          year,
        ]);

        if (existing.length > 0) {
          console.log(
            `Payment already exists for user ${user.Email} for ${month}/${year}`
```

```
          );
          continue;
        }

        // Create new payment record with Created_At set to target month
        // This allows us to track the payment period using DATE functions
        const targetDate = new Date(year, month - 1, 21); // 21st of target
month

        const insertPaymentQuery = `
          INSERT INTO pembayaran (
          ID_Reservasi,
            Jumlah,
            Status,
            Created_At
          ) VALUES (?, ?, 'Belum Bayar', ?)
        `;

        await pool.execute(insertPaymentQuery, [
          user.ID_Reservasi,
          900000, // Default price as per database structure
          targetDate,
        ]);

        generatedCount++;
        console.log(
          `Generated payment for ${user.Nama} (${user.Email}) - Room:
${user.Nama_Kamar} - Amount: 900000`
        );
      } catch (userError) {
        console.error(
          `Error generating payment for user ${user.Email}:`,
          userError
        );
        errors.push({
          userEmail: user.Email,
          userName: user.Nama,
          error: userError.message,
        });
      }
    }

    console.log(
      `Payment generation completed. Generated: ${generatedCount}, Errors:
${errors.length}`
```

```
    );

    return {
      success: true,
      message: `Successfully generated ${generatedCount} payments`,
      generated: generatedCount,
      errors: errors,
    };
  } catch (error) {
    console.error("Error in generateMonthlyPaymentsForAllUsers:", error);
    return {
      success: false,
      message: "Failed to generate monthly payments",
      error: error.message,
    };
  }
}

// Get payment generation history
static async getGenerationHistory() {
try {
    const query = `
      SELECT
        DATE(Created_At) as generation_date,
        COUNT(*) as payments_generated,
        MONTH(Created_At) as month,
        YEAR(Created_At) as year
      FROM pembayaran
      WHERE DATE(Created_At) >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
      GROUP BY DATE(Created_At), MONTH(Created_At), YEAR(Created_At)
      ORDER BY Created_At DESC
      LIMIT 10
    `;

    const [results] = await pool.execute(query);

    return {
      success: true,
      data: results,
    };
  } catch (error) {
    console.error("Error in getGenerationHistory:", error);
    return {
      success: false,
      message: "Failed to get generation history",
```

```
      error: error.message,
    };
   }
  }
}

module.exports = Payment;
```

Tabel 4. 14 Codingan PendingReservation.js

PendingReservation.js

```
const { pool } = require("../config/database");
const bcrypt = require("bcryptjs");
const crypto = require("crypto");

class PendingReservation {
 // Create new pending reservation with payment proof
 static async create(pendingData, buktiPembayaranPath) {
 const {
    Nama,
    Email,
    Password,
    No_telp,
    Alamat,
    No_Kamar,
    Tanggal_Reservasi,
  } = pendingData;

  try {
   // Hash password
   const hashedPassword = await bcrypt.hash(Password, 10);

   const query = `
    INSERT INTO pending_reservations (
     Nama, No_telp, Alamat, Email, Password, No_Kamar,
     Tanggal_Reservasi, Bukti_Pembayaran, Status
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, 'Menunggu')
   `;

   const [result] = await pool.execute(query, [
    Nama,
    No_telp,
    Alamat,
    Email,
    hashedPassword,
```

```
      No_Kamar,
      Tanggal_Reservasi,
      buktiPembayaranPath,
    ]);

    return {
      success: true,
      message: "Pending reservation created successfully",
      pendingId: result.insertId,
    };
  } catch (error) {
    if (error.code === "ER_DUP_ENTRY") {
      return {
        success: false,
        message: "Email already has a pending reservation",
      };
    }
    throw error;
  }
}

// Get all pending reservations (Admin only)
static async getAll() {
  try {
    const query = `
      SELECT
        pr.*,
        k.Nama_Kamar,
        k.Letak
      FROM pending_reservations pr
      JOIN kamar k ON pr.No_Kamar = k.No_Kamar
      ORDER BY pr.Created_At DESC
    `;

    const [rows] = await pool.execute(query);
    return rows;
  } catch (error) {
    throw error;
  }
}

// Get pending reservation by ID
static async getById(id) {
  try {
    const query = `
```

```
      SELECT
        pr.*,
        k.Nama_Kamar,
        k.Letak
      FROM pending_reservations pr
      JOIN kamar k ON pr.No_Kamar = k.No_Kamar
      WHERE pr.ID_Pending = ?
    `;

    const [rows] = await pool.execute(query, [id]);
    return rows[0] || null;
  } catch (error) {
    throw error;
  }
 }

 // Approve pending reservation and create user + reservation
 static async approve(id) {
  const connection = await pool.getConnection();

  try {
    await connection.beginTransaction();

    // Get pending reservation data
    const [pendingRows] = await connection.execute(
      "SELECT * FROM pending_reservations WHERE ID_Pending = ? AND
Status = 'Menunggu'",
      [id]
    );

    if (pendingRows.length === 0) {
      await connection.rollback();
      return {
        success: false,
        message: "Pending reservation not found or already processed",
      };
    }

    const pending = pendingRows[0];

    // 1. Create user in users table
    const userQuery = `
      INSERT INTO user (Nama, No_telp, Alamat, Email, Password, Role)
      VALUES (?, ?, ?, ?, ?, 'penyewa')
    `;
```

```
    await connection.execute(userQuery, [
      pending.Nama,
      pending.No_telp,
      pending.Alamat,
      pending.Email,
      pending.Password, // Already hashed
    ]);

    // 2. Create reservation
    const reservasiQuery = `
      INSERT INTO reservasi (No_Kamar, Email, Tanggal_Reservasi, Status)
      VALUES (?, ?, ?, 'Diterima')
    `;

    const [reservasiResult] = await connection.execute(reservasiQuery, [
      pending.No_Kamar,
      pending.Email,
      pending.Tanggal_Reservasi,
    ]);

    // 3. Generate temporary token (valid for 1 day)
    const token = crypto.randomBytes(32).toString("hex");
    const expiresAt = new Date();
    expiresAt.setDate(expiresAt.getDate() + 1); // 1 day from now

    const tokenQuery = `
      INSERT INTO user_tokens (Email, Token, Expires_At)
      VALUES (?, ?, ?)
    `;

    await connection.execute(tokenQuery, [pending.Email, token,
expiresAt]);

    // 4. Update pending reservation status
    await connection.execute(
      "UPDATE pending_reservations SET Status = 'Diterima' WHERE
ID_Pending = ?",
      [id]
    );

    // 5. Update room availability
    await connection.execute(
      "UPDATE kamar SET Ketersediaan = 0 WHERE No_Kamar = ?",
      [pending.No_Kamar]
```

```
      );

      await connection.commit();

      return {
        success: true,
        message: "Reservation approved successfully",
        data: {
          userId: pending.Email,
          reservasiId: reservasiResult.insertId,
          token: token,
          expiresAt: expiresAt,
        },
      };
    } catch (error) {
      await connection.rollback();
      if (error.code === "ER_DUP_ENTRY") {
        return {
          success: false,
          message: "User with this email already exists",
        };
      }
      throw error;
    } finally {
      connection.release();
    }
  }

  // Reject pending reservation
  static async reject(id) {
    try {
      const query =
        "UPDATE pending_reservations SET Status = 'Ditolak' WHERE
ID_Pending = ? AND Status = 'Menunggu'";
      const [result] = await pool.execute(query, [id]);

      if (result.affectedRows === 0) {
        return {
          success: false,
          message: "Pending reservation not found or already processed",
        };
      }

      return {
        success: true,
```

```
      message: "Reservation rejected successfully",
     };
    } catch (error) {
     throw error;
    }
  }

  // Delete pending reservation
  static async delete(id) {
   try {
    const query = "DELETE FROM pending_reservations WHERE ID_Pending
= ?";
    const [result] = await pool.execute(query, [id]);

    if (result.affectedRows === 0) {
     return {
      success: false,
      message: "Pending reservation not found",
     };
    }

    return {
     success: true,
     message: "Pending reservation deleted successfully",
    };
   } catch (error) {
    throw error;
   }
  }
}

module.exports = PendingReservation;
```

Tabel 4. 15 Codingan Reservasi.js

	Reservasi.js

	const { pool } = require("../config/database");
	
	class Reservasi {
	  // Create new reservation
	  static async create(reservationData) {
	  const { No_Kamar, Email, Tanggal_Reservasi } = reservationData;
	
	  try {
	  // Check if room is available

```
    const roomQuery = "SELECT Ketersediaan FROM kamar WHERE
No_Kamar = ?";
    const [roomRows] = await pool.execute(roomQuery, [No_Kamar]);

    if (roomRows.length === 0) {
      return {
        success: false,
        message: "Room not found",
      };
    }

    if (roomRows[0].Ketersediaan === 0) {
      return {
        success: false,
        message: "Room is not available",
      };
    }

    // Create reservation
    const query = `
            INSERT INTO reservasi (No_Kamar, Email, Tanggal_Reservasi,
Status)
            VALUES (?, ?, ?, 'Menunggu')
          `;

    const [result] = await pool.execute(query, [
      No_Kamar,
      Email,
      Tanggal_Reservasi,
    ]);

    return {
      success: true,
      message: "Reservation created successfully",
      reservationId: result.insertId,
    };
  } catch (error) {
    throw error;
  }
 }

 // Get user reservations
 static async getByUser(email) {
  try {
    const query = `
```

```
            SELECT r.*, k.Nama_Kamar, k.Letak
            FROM reservasi r
            JOIN kamar k ON r.No_Kamar = k.No_Kamar
            WHERE r.Email = ?
            ORDER BY r.Tanggal_Reservasi DESC
        `;

    const [rows] = await pool.execute(query, [email]);
    return rows;
  } catch (error) {
    throw error;
  }
}

// Get all reservations (for admin)
static async getAll() {
  try {
    const query = `
            SELECT r.*, k.Nama_Kamar, k.Letak, u.Nama as Nama_User
            FROM reservasi r
            JOIN kamar k ON r.No_Kamar = k.No_Kamar
            JOIN user u ON r.Email = u.Email
            ORDER BY r.Tanggal_Reservasi DESC
        `;

    const [rows] = await pool.execute(query);
    return rows;
  } catch (error) {
    throw error;
  }
}

// Get all reservations with payment info (for admin)
static async getAllWithPayments(filters = {}) {
  try {
    let query = `
            SELECT
                r.*,
                k.Nama_Kamar,
                k.Letak,
                u.Nama,
                u.No_telp,
                u.Alamat,
                (
```

```
                SELECT CONCAT(YEAR(p.Created_At), '-',
LPAD(MONTH(p.Created_At), 2, '0'))
                FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                ORDER BY p.Created_At DESC
                LIMIT 1
            ) as last_payment_period,
            (
                SELECT p.Status
                FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                ORDER BY p.Created_At DESC
                LIMIT 1
            ) as latest_payment_status,
            (
                SELECT COUNT(*)
                FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                AND p.Status = 'Diterima'
            ) as total_payments,
            CASE
                WHEN (
                    SELECT p.Status
                    FROM pembayaran p
                    WHERE p.ID_Reservasi = r.ID_Reservasi
                    ORDER BY p.Created_At DESC
                    LIMIT 1
                ) = 'Diterima' THEN r.Status
                WHEN (
                    SELECT COUNT(*)
                    FROM pembayaran p
                    WHERE p.ID_Reservasi = r.ID_Reservasi
                ) > 0 THEN 'Telat/Belum Bayar'
                ELSE r.Status
            END as calculated_status
        FROM reservasi r
        JOIN kamar k ON r.No_Kamar = k.No_Kamar
        JOIN user u ON r.Email = u.Email
    `;

  const whereConditions = [];
  const queryParams = [];

  if (filters.status) {
    whereConditions.push("r.Status = ?");
```

```
      queryParams.push(filters.status);
    }

    if (filters.kamar) {
      whereConditions.push("r.No_Kamar = ?");
      queryParams.push(filters.kamar);
    }

    if (filters.periode) {
      whereConditions.push(`
              EXISTS (
                SELECT 1 FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                AND CONCAT(YEAR(p.Created_At), '-',
LPAD(MONTH(p.Created_At), 2, '0')) = ?
              )
            `);
      queryParams.push(filters.periode);
    }

    if (whereConditions.length > 0) {
      query += " WHERE " + whereConditions.join(" AND ");
    }

    query += " ORDER BY r.Tanggal_Reservasi DESC";

    const [rows] = await pool.execute(query, queryParams);
    return rows;
  } catch (error) {
    throw error;
  }
 }

 // Get reservations by user email with payment info and calculated status
 static async getByUserIdWithPayments(userEmail) {
  try {
    const query = `
          SELECT
            r.*,
            k.Nama_Kamar,
            k.Letak,
            u.Nama,
            u.No_telp,
            u.Alamat,
            (
```

```
                SELECT CONCAT(YEAR(p.Created_At), '-',
LPAD(MONTH(p.Created_At), 2, '0'))
                FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                ORDER BY p.Created_At DESC
                LIMIT 1
            ) as periode,
            (
                SELECT p.Jumlah
                FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                ORDER BY p.Created_At DESC
                LIMIT 1
            ) as Harga_Kamar,
            (
                SELECT p.Status
                FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                ORDER BY p.Created_At DESC
                LIMIT 1
            ) as latest_payment_status,
            (
                SELECT COUNT(*)
                FROM pembayaran p
                WHERE p.ID_Reservasi = r.ID_Reservasi
                AND p.Status = 'Diterima'
            ) as total_payments,
            CASE
                WHEN (
                    SELECT p.Status
                    FROM pembayaran p
                    WHERE p.ID_Reservasi = r.ID_Reservasi
                    ORDER BY p.Created_At DESC
                    LIMIT 1
                ) = 'Diterima' THEN 'Aktif/Lunas'
                WHEN (
                    SELECT COUNT(*)
                    FROM pembayaran p
                    WHERE p.ID_Reservasi = r.ID_Reservasi
                ) > 0 THEN 'Telat/Belum Bayar'
                ELSE r.Status
            END as calculated_status
        FROM reservasi r
        JOIN kamar k ON r.No_Kamar = k.No_Kamar
        JOIN user u ON r.Email = u.Email
```

```
            WHERE u.Email = ?
            ORDER BY r.Tanggal_Reservasi DESC
          `;

    const [rows] = await pool.execute(query, [userEmail]);
    return rows;
   } catch (error) {
    throw error;
   }
  }

  // Update reservation status with notes (for admin)
  static async updateStatus(reservationId, status, keterangan = null) {
   try {
    const query = "UPDATE reservasi SET Status = ? WHERE ID_Reservasi =
?";
    const [result] = await pool.execute(query, [status, reservationId]);

    return {
     success: result.affectedRows > 0,
     message:
      result.affectedRows > 0
       ? "Reservation status updated"
       : "Reservation not found",
    };
   } catch (error) {
    throw error;
   }
  }
}

module.exports = Reservasi;
```

Tabel 4. 16 Codingan TmpUser.js

	TmpUser.js

```
const { pool } = require("../config/database");
const bcrypt = require("bcryptjs");
const crypto = require("crypto");
const Payment = require("./Payment");

class TmpUser {
  // Create new temporary user with payment proof
  static async create(tmpData, buktiPembayaranPath) {
    const { Nama, Email, Password, No_telp, Alamat, No_Kamar } = tmpData;
```

```
  try {
    // Hash password
    const hashedPassword = await bcrypt.hash(Password, 10);

    const query = `
      INSERT INTO tmp (
        Nama, No_telp, Alamat, Email, Password, No_Kamar,
        Bukti_Pembayaran, Role
      ) VALUES (?, ?, ?, ?, ?, ?, ?, 'penyewa')
    `;

    const [result] = await pool.execute(query, [
      Nama,
      No_telp,
      Alamat,
      Email,
      hashedPassword,
      No_Kamar,
      buktiPembayaranPath,
    ]);

    // Note: Reservation will be created when admin approves the tmp user
    // The tmp table already contains No_Kamar for tracking which room
was requested

    return {
      success: true,
      message:
        "Temporary user created successfully. Waiting for admin approval",
      tmpId: result.insertId,
    };
  } catch (error) {
    if (error.code === "ER_DUP_ENTRY") {
      return {
        success: false,
        message: "Email already has a pending reservation",
      };
    }
    throw error;
  }
 }

 // Get all temporary users with their reservation status
 static async getAllWithReservationStatus() {
```

```
    try {
      const query = `
        SELECT
          t.*,
          k.Nama_Kamar,
          k.Letak,
          r.Status as Reservation_Status,
          r.ID_Reservasi,
          r.Tanggal_Reservasi
        FROM tmp t
        JOIN kamar k ON t.No_Kamar = k.No_Kamar
        LEFT JOIN reservasi r ON t.Email = r.Email AND t.No_Kamar =
r.No_Kamar
        ORDER BY t.Created_At DESC
      `;

      const [rows] = await pool.execute(query);
      return rows;
    } catch (error) {
      throw error;
    }
  }

  // Get temporary user by ID with reservation status
  static async getByIdWithReservationStatus(id) {
    try {
      const query = `
        SELECT
          t.*,
          k.Nama_Kamar,
          k.Letak,
          r.Status as Reservation_Status,
          r.ID_Reservasi,
          r.Tanggal_Reservasi
        FROM tmp t
        JOIN kamar k ON t.No_Kamar = k.No_Kamar
        LEFT JOIN reservasi r ON t.Email = r.Email AND t.No_Kamar =
r.No_Kamar
        WHERE t.ID_Tmp = ?
      `;

      const [rows] = await pool.execute(query, [id]);
      return rows[0] || null;
    } catch (error) {
      throw error;
```

```
  }
}

// Approve tmp user - move to user table and create active reservation
static async approve(id) {
  const connection = await pool.getConnection();

  try {
    await connection.beginTransaction();

    // Get tmp user data
    const [tmpRows] = await connection.execute(
      "SELECT * FROM tmp WHERE ID_Tmp = ?",
      [id]
    );

    if (tmpRows.length === 0) {
      await connection.rollback();
      return {
        success: false,
        message: "Temporary user not found",
      };
    }

    const tmpUser = tmpRows[0];

    // 1. Create user in users table
    const userQuery = `
      INSERT INTO user (Nama, No_telp, Alamat, Email, Password, Role)
      VALUES (?, ?, ?, ?, ?, ?)
    `;

    await connection.execute(userQuery, [
      tmpUser.Nama,
      tmpUser.No_telp,
      tmpUser.Alamat,
      tmpUser.Email,
      tmpUser.Password, // Already hashed
      tmpUser.Role,
    ]);

    // 2. Create reservation with 'Aktif/Lunas' status (as paid first month)
    const reservasiQuery = `
      INSERT INTO reservasi (No_Kamar, Email, Tanggal_Reservasi, Status)
      VALUES (?, ?, NOW(), 'Aktif/Lunas')
```

```
    `;

    const [reservasiResult] = await connection.execute(reservasiQuery, [
      tmpUser.No_Kamar,
      tmpUser.Email,
    ]);

    // 3. Create payment record for first month
    const currentDate = new Date();
    const currentMonth = String(currentDate.getMonth() + 1).padStart(2,
"0");
    const currentYear = currentDate.getFullYear().toString();
    const monthYear = currentYear + "-" + currentMonth;

    const paymentQuery = `
      INSERT INTO pembayaran (
      ID_Reservasi,
      Tanggal_Bayar,
        Jumlah,
        Bukti_Pembayaran,
        Status
      ) VALUES (?, NOW(), 900000, ?, 'Diterima')
    `;

    await connection.execute(paymentQuery, [
      reservasiResult.insertId,
      tmpUser.Bukti_Pembayaran,
    ]);

    // 4. Generate temporary token (valid for 1 day)
    const token = crypto.randomBytes(32).toString("hex");
    const expiresAt = new Date();
    expiresAt.setDate(expiresAt.getDate() + 1); // 1 day from now

    const tokenQuery = `
      INSERT INTO user_tokens (Email, Token, Expires_At)
      VALUES (?, ?, ?)
    `;

    await connection.execute(tokenQuery, [tmpUser.Email, token,
expiresAt]);

    // 5. Update room availability
    await connection.execute(
      "UPDATE kamar SET Ketersediaan = 0 WHERE No_Kamar = ?",
```

```
      [tmpUser.No_Kamar]
    );

    // 6. Delete from tmp table
    await connection.execute("DELETE FROM tmp WHERE ID_Tmp = ?", [id]);

    await connection.commit();

    return {
      success: true,
      message: "User approved successfully and moved to active status",
      data: {
        reservationId: reservasiResult.insertId,
        email: tmpUser.Email,
        token: token,
        expiresAt: expiresAt,
        status: "Aktif/Lunas",
      },
    };
  } catch (error) {
    await connection.rollback();
    if (error.code === "ER_DUP_ENTRY") {
      return {
        success: false,
        message: "User with this email already exists",
      };
    }
    throw error;
  } finally {
    connection.release();
  }
}

// Reject tmp user - delete from tmp and update reservation status
static async reject(id) {
  const connection = await pool.getConnection();

  try {
    await connection.beginTransaction();

    // Get tmp user data
    const [tmpRows] = await connection.execute(
      "SELECT * FROM tmp WHERE ID_Tmp = ?",
      [id]
    );
```

```
    if (tmpRows.length === 0) {
      await connection.rollback();
      return {
        success: false,
        message: "Temporary user not found",
      };
    }

    const tmpUser = tmpRows[0];

    // 1. Delete from tmp table (no reservation to update since it was never
created)
    await connection.execute("DELETE FROM tmp WHERE ID_Tmp = ?", [id]);

    await connection.commit();

    return {
      success: true,
      message: "User rejected successfully",
    };
  } catch (error) {
    await connection.rollback();
    throw error;
  } finally {
    connection.release();
  }
 }

 // Get tmp user by email
 static async getByEmail(email) {
  try {
    const query = "SELECT * FROM tmp WHERE Email = ?";
    const [rows] = await pool.execute(query, [email]);
    return rows[0] || null;
  } catch (error) {
    throw error;
  }
 }

 // Auto update reservation status based on payment dates
 static async updateReservationStatus() {
  const connection = await pool.getConnection();

  try {
```

```
// Get current date
const now = new Date();
const currentYear = now.getFullYear();
const currentMonth = now.getMonth() + 1;
const currentDay = now.getDate();

// Define payment deadline (example: day 5 of each month)
const PAYMENT_DEADLINE = 5;

// Get all active reservations
const [reservations] = await connection.execute(`
  SELECT r.*, u.Email, u.Nama
  FROM reservasi r
  JOIN user u ON r.Email = u.Email
  WHERE r.Status IN ('Aktif/Lunas', 'Telat/Belum Bayar')
`);

for (const reservation of reservations) {
  // Check latest payment for this reservation
  const [payments] = await connection.execute(
    `
    SELECT * FROM pembayaran
    WHERE ID_Reservasi = ?
    ORDER BY Tanggal_Bayar DESC
    LIMIT 1
  `,
    [reservation.ID_Reservasi]
  );

  if (payments.length === 0) {
    // No payment found, set status to late
    await connection.execute(
      `
      UPDATE reservasi
      SET Status = 'Telat/Belum Bayar'
      WHERE ID_Reservasi = ?
    `,
      [reservation.ID_Reservasi]
    );
    continue;
  }

  const lastPayment = payments[0];
  const lastPaymentDate = new Date(
    lastPayment.Tanggal_Bayar || lastPayment.Created_At
```

```
    );
    const lastPaymentMonth = lastPaymentDate.getMonth() + 1;
    const lastPaymentYear = lastPaymentDate.getFullYear();
    const lastPaymentStatus = lastPayment.Status;

    // Check if payment is for current month AND status is 'Diterima'
    const isCurrentMonthPaid =
      lastPaymentYear === currentYear &&
      lastPaymentMonth === currentMonth &&
      lastPaymentStatus === "Diterima";

    // Check if past deadline for current month
    const isPastDeadline = currentDay > PAYMENT_DEADLINE;

    let newStatus = reservation.Status;

    if (isCurrentMonthPaid) {
      // Paid current month and accepted - set to active
      newStatus = "Aktif/Lunas";
    } else if (
      isPastDeadline ||
      (lastPaymentYear === currentYear &&
        lastPaymentMonth === currentMonth &&
        lastPaymentStatus !== "Diterima")
    ) {
      // Past deadline OR current month payment not accepted - set to late
      newStatus = "Telat/Belum Bayar";

      // Create pending payment record for current month if not exists
      const currentMonthStr =
        currentYear + "-" + String(currentMonth).padStart(2, "0");

      const [existingPayment] = await connection.execute(
        `
        SELECT * FROM pembayaran
        WHERE ID_Reservasi = ?
        AND YEAR(Created_At) = ?
        AND MONTH(Created_At) = ?
      `,
        [reservation.ID_Reservasi, currentYear, currentMonth]
      );

      if (existingPayment.length === 0) {
        // Create pending payment record
        await connection.execute(
```

```
          `
          INSERT INTO pembayaran (
            ID_Reservasi,
            Tanggal_Bayar,
            Jumlah,
            Bukti_Pembayaran,
            Status
          ) VALUES (?, NULL, 900000, NULL, 'Belum Bayar')
        `,
          [reservation.ID_Reservasi]
        );
       }
      }

      // Update reservation status if changed
      if (newStatus !== reservation.Status) {
      await connection.execute(
        `
        UPDATE reservasi
        SET Status = ?
        WHERE ID_Reservasi = ?
       `,
        [newStatus, reservation.ID_Reservasi]
       );
      }
    }

    await connection.commit();
    return {
     success: true,
     message: "Reservation statuses updated successfully",
    };
   } catch (error) {
    await connection.rollback();
    throw error;
   } finally {
    connection.release();
   }
}

// Function to mark user as checkout/keluar
static async markAsCheckout(reservationId, reason = "") {
  try {
    const [result] = await pool.execute(
      `
```

```
      UPDATE reservasi
      SET Status = 'Keluar', Keterangan = ?
      WHERE ID_Reservasi = ?
    `,
      [reason, reservationId]
    );

    if (result.affectedRows === 0) {
      return {
        success: false,
        message: "Reservation not found",
      };
    }

    // Update room availability
    const [reservation] = await pool.execute(
      `
      SELECT No_Kamar FROM reservasi WHERE ID_Reservasi = ?
    `,
      [reservationId]
    );

    if (reservation.length > 0) {
      await pool.execute(
        `
        UPDATE kamar SET Ketersediaan = 1 WHERE No_Kamar = ?
      `,
        [reservation[0].No_Kamar]
      );
    }

    return {
      success: true,
      message: "User marked as checkout successfully",
    };
   } catch (error) {
    throw error;
   }
  }
}

module.exports = TmpUser;
```

Tabel 4. 17 Codingan User.js

	User.js

```
const { pool } = require("../config/database");
const bcrypt = require("bcryptjs");

class User {
 // Create new user (Registration)
 static async create(userData) {
  const { Nama, Email, Password, No_telp, Alamat } = userData;

  try {
   // Hash password
   const hashedPassword = await bcrypt.hash(Password, 10);

   const query = `
         INSERT INTO user (Nama, No_telp, Alamat, Email, Password, Role)
         VALUES (?, ?, ?, ?, ?, 'penyewa')
       `;

   const [result] = await pool.execute(query, [
    Nama,
    No_telp,
    Alamat,
    Email,
    hashedPassword,
   ]);

   return {
    success: true,
    message: "User created successfully",
    userId: result.insertId,
   };
  } catch (error) {
   if (error.code === "ER_DUP_ENTRY") {
    return {
     success: false,
     message: "Email already exists",
    };
   }
   throw error;
  }
 }

 // Find user by email
 static async findByEmail(email) {
```

```
    try {
      const query = "SELECT * FROM user WHERE Email = ?";
      const [rows] = await pool.execute(query, [email]);
      return rows[0] || null;
    } catch (error) {
      throw error;
    }
  }

  // Get all users (Admin only)
  static async getAllUsers() {
  try {
      const query = "SELECT Nama, Email, No_telp, Alamat, Role, Foto FROM
user ORDER BY Nama";
      const [rows] = await pool.execute(query);
      return rows;
    } catch (error) {
      throw error;
    }
  }

  // Verify password
  static async verifyPassword(plainPassword, hashedPassword) {
    return await bcrypt.compare(plainPassword, hashedPassword);
  }

  // Get user profile (without password)
  static async getProfile(email) {
    try {
      const query =
        "SELECT Nama, No_telp, Alamat, Email, Foto, Role FROM user WHERE
Email = ?";
      const [rows] = await pool.execute(query, [email]);
      return rows[0] || null;
    } catch (error) {
      throw error;
    }
  }

  // Update user profile
  static async updateProfile(email, updateData) {
    try {
      const { Nama, No_telp, Alamat, Foto } = updateData;

      // Build dynamic query based on what fields are being updated
```

```
    let setClause = [];
    let queryParams = [];

    if (Nama !== undefined) {
      setClause.push("Nama = ?");
      queryParams.push(Nama);
    }

    if (No_telp !== undefined) {
      setClause.push("No_telp = ?");
      queryParams.push(No_telp);
    }

    if (Alamat !== undefined) {
      setClause.push("Alamat = ?");
      queryParams.push(Alamat);
    }

    if (Foto !== undefined) {
      setClause.push("Foto = ?");
      queryParams.push(Foto);
    }

    if (setClause.length === 0) {
      return {
        success: false,
        message: "No valid fields to update",
      };
    }

    const query = `UPDATE user SET ${setClause.join(", ")} WHERE Email = ?`;
    queryParams.push(email);

    const [result] = await pool.execute(query, queryParams);

    if (result.affectedRows > 0) {
      // Return updated user data
      const updatedUser = await this.findByEmail(email);
      return {
        success: true,
        message: "Profile updated successfully",
        data: {
          Nama: updatedUser.Nama,
          Email: updatedUser.Email,
          No_telp: updatedUser.No_telp,
```

```
          Alamat: updatedUser.Alamat,
          Foto: updatedUser.Foto,
          Role: updatedUser.Role,
        },
      };
    } else {
      return {
        success: false,
        message: "User not found",
      };
    }
  } catch (error) {
    throw error;
  }
}

// Change user password
static async changePassword(email, currentPassword, newPassword) {
  try {
    // Get current user data
    const user = await this.findByEmail(email);
    if (!user) {
      return {
        success: false,
        message: "User not found",
      };
    }

    // Verify current password
    const isCurrentPasswordValid = await this.verifyPassword(
      currentPassword,
      user.Password
    );
    if (!isCurrentPasswordValid) {
      return {
        success: false,
        message: "Password lama tidak benar",
      };
    }

    // Hash new password
    const hashedNewPassword = await bcrypt.hash(newPassword, 10);

    // Update password in database
    const query = "UPDATE user SET Password = ? WHERE Email = ?";
```

```
    const [result] = await pool.execute(query, [hashedNewPassword, email]);

    if (result.affectedRows > 0) {
      return {
        success: true,
        message: "Password berhasil diubah",
      };
    } else {
      return {
        success: false,
        message: "Gagal mengubah password",
      };
    }
  } catch (error) {
    throw error;
  }
}

// Get all reservations with user data for admin
static async getAllReservationsWithUserData(filters = {}) {
  try {
    let query = `
      SELECT
        r.ID_Reservasi,
        r.No_Kamar,
        r.Email,
        r.Tanggal_Reservasi,
        r.Status,
        u.Nama,
        u.No_telp,
        u.Alamat,
        u.Role,
        u.Foto,
        k.Nama_Kamar,
        k.Letak,
        k.Ketersediaan
      FROM reservasi r
      LEFT JOIN user u ON r.Email = u.Email
      LEFT JOIN kamar k ON r.No_Kamar = k.No_Kamar
      WHERE 1=1
    `;

    let queryParams = [];

    // Apply filters
```

```
    if (filters.status && filters.status !== "") {
      query += " AND r.Status = ?";
      queryParams.push(filters.status);
    }

    if (filters.kamar && filters.kamar !== "") {
      query += " AND r.No_Kamar = ?";
      queryParams.push(filters.kamar);
    }

    if (filters.periode && filters.periode !== "") {
      query += ' AND DATE_FORMAT(r.Tanggal_Reservasi, "%Y-%m") = ?';
      queryParams.push(filters.periode);
    }

    query += " ORDER BY r.Tanggal_Reservasi DESC, u.Nama ASC";

    const [rows] = await pool.execute(query, queryParams);
    return rows;
  } catch (error) {
    throw error;
  }
 }

 // Get filter options for admin interface
 static async getFilterOptions() {
  try {
    const [statusRows] = await pool.execute(`
      SELECT DISTINCT Status
      FROM reservasi
      WHERE Status IS NOT NULL
      ORDER BY Status
    `);

    const [kamarRows] = await pool.execute(`
      SELECT DISTINCT No_Kamar
      FROM kamar
      ORDER BY No_Kamar
    `);

    const [periodeRows] = await pool.execute(`
      SELECT DISTINCT DATE_FORMAT(Tanggal_Reservasi, "%Y-%m") as
periode
      FROM reservasi
      WHERE Tanggal_Reservasi IS NOT NULL
```

```
      ORDER BY periode DESC
    `);

    return {
      statuses: statusRows.map((row) => row.Status),
      kamars: kamarRows.map((row) => row.No_Kamar),
      periodes: periodeRows.map((row) => row.periode),
    };
  } catch (error) {
    throw error;
  }
}

// Update reservation status
static async updateReservationStatus(
  reservasiId,
  newStatus,
  keterangan = null
) {
  try {
    let query = "UPDATE reservasi SET Status = ?";
    let queryParams = [newStatus];

    query += " WHERE ID_Reservasi = ?";
    queryParams.push(reservasiId);

    const [result] = await pool.execute(query, queryParams);

    if (result.affectedRows > 0) {
      return {
        success: true,
        message: "Status reservasi berhasil diupdate",
      };
    } else {
      return {
        success: false,
        message: "Reservasi tidak ditemukan",
      };
    }
  } catch (error) {
    throw error;
  }
}

// Get all users (for admin interface)
```

```
  static async getAll() {
    try {
      const query = `
        SELECT Nama, Email, No_telp, Alamat, Role, Foto
        FROM user
        ORDER BY Nama ASC
      `;
      const [rows] = await pool.execute(query);
      return rows;
    } catch (error) {
      throw error;
    }
  }

  // Get dashboard statistics
  static async getDashboardStatistics() {
    try {
      // Get total kamar count
      const [kamarResult] = await pool.execute(
        "SELECT COUNT(*) as total_kamar FROM kamar"
      );
      const totalKamar = kamarResult[0].total_kamar;

      // Get active users count (penyewa role only)
      const [userResult] = await pool.execute(
        "SELECT COUNT(*) as total_users FROM user WHERE Role = 'penyewa'"
      );
      const totalUsers = userResult[0].total_users;

      // Get monthly payment total based on actual payments with 'Diterima'
status for current month
      const currentMonth = new Date().toISOString().slice(0, 7); // YYYY-MM
format
      const [paymentResult] = await pool.execute(
        `
        SELECT COALESCE(SUM(p.Jumlah), 0) as monthly_payment, COUNT(*)
as paid_count
        FROM pembayaran p
        WHERE p.Status = 'Diterima'
        AND DATE_FORMAT(p.Tanggal_Bayar, '%Y-%m') = ?
      `,
        [currentMonth]
      );

      const monthlyPayment = paymentResult[0].monthly_payment || 0;
```

```
    const paidUsers = paymentResult[0].paid_count || 0;

    // Calculate occupancy percentage
    const occupancyPercentage =
      totalKamar > 0 ? Math.round((totalUsers / totalKamar) * 100) : 0;

    return {
      totalKamar,
      totalUsers,
      monthlyPayment,
      occupancyPercentage,
      paidUsers,
    };
  } catch (error) {
    throw error;
  }
 }

 // Get all payments for admin interface (adapted for current database
schema)
 static async getAllPaymentsForAdmin(filters = {}) {
  try {
    let query = `
      SELECT
        p.ID_Pembayaran as id,
        u.Nama as nama,
        CONCAT('Kamar ', r.No_Kamar) as kamar,
        r.No_Kamar as no_kamar,
        COALESCE(DATE_FORMAT(p.Tanggal_Bayar, '%m'), '07') as bulan,
        COALESCE(DATE_FORMAT(p.Tanggal_Bayar, '%Y'), '2025') as tahun,
        p.Jumlah as jumlah,
        p.Tanggal_Bayar as tanggal_bayar,
        CASE
          WHEN p.Status = 'Lunas' THEN 'Diterima'
          WHEN p.Status = 'Belum Bayar' THEN 'Menunggu'
          WHEN p.Status = 'Terlambat' THEN 'Ditolak'
          ELSE p.Status
        END as status,
        NULL as bukti_pembayaran,
        u.Email as email,
        u.No_telp as no_telp
      FROM pembayaran p
      INNER JOIN reservasi r ON p.ID_Reservasi = r.ID_Reservasi
      INNER JOIN user u ON r.Email = u.Email
      WHERE 1=1
```

```
    `;

    const params = [];

    // Apply filters
    if (filters.status && filters.status !== "") {
      const dbStatus =
        filters.status === "Diterima"
          ? "Lunas"
          : filters.status === "Menunggu"
          ? "Belum Bayar"
          : filters.status === "Ditolak"
          ? "Terlambat"
          : filters.status;
      query += " AND p.Status = ?";
      params.push(dbStatus);
    }

    if (filters.bulan && filters.bulan !== "") {
      query +=
        ' AND (p.Tanggal_Bayar IS NULL OR DATE_FORMAT(p.Tanggal_Bayar,
"%m") = ?)';
      params.push(filters.bulan.padStart(2,  "0"));
    }

    if (filters.tahun && filters.tahun !== "") {
      query +=
        ' AND (p.Tanggal_Bayar IS NULL OR DATE_FORMAT(p.Tanggal_Bayar,
"%Y") = ?)';
      params.push(filters.tahun);
    }

    if (filters.kamar && filters.kamar !== "") {
      query += " AND r.No_Kamar = ?";
      params.push(filters.kamar);
    }

    query += " ORDER BY p.Tanggal_Bayar DESC, p.ID_Pembayaran DESC";

    const [rows] = await pool.execute(query, params);
    return rows;
  } catch (error) {
    throw error;
  }
 }
```

```
// Update payment status
static async updatePaymentStatus(paymentId, status, keterangan = null) {
 try {
  // Map admin status to database status
  const dbStatus =
   status === "Diterima"
    ? "Lunas"
    : status === "Menunggu"
    ? "Belum Bayar"
    : status === "Ditolak"
    ? "Terlambat"
    : status;

  const query = `
   UPDATE pembayaran
   SET Status = ?
   WHERE ID_Pembayaran = ?
  `;

  const [result] = await pool.execute(query, [dbStatus, paymentId]);

  if (result.affectedRows === 0) {
   return {
    success: false,
    message: "Pembayaran tidak ditemukan",
   };
  }

  // If payment is accepted, update the associated reservation status
  if (status === "Diterima") {
   await pool.execute(
    `
    UPDATE reservasi r
    INNER JOIN pembayaran p ON r.ID_Reservasi = p.ID_Reservasi
    SET r.Status = 'Diterima'
    WHERE p.ID_Pembayaran = ?
   `,
    [paymentId]
   );
  }

  return {
   success: true,
   message: `Status pembayaran berhasil diubah menjadi ${status}`,
```

```
    };
   } catch (error) {
    throw error;
   }
  }

  // Create new user by admin
  static async createByAdmin(userData) {
   const { Nama, Email, Password, No_telp, Alamat, Role = "penyewa" } =
userData;

   try {
    // Hash password
    const hashedPassword = await bcrypt.hash(Password, 10);

    const query = `
     INSERT INTO user (Nama, No_telp, Alamat, Email, Password, Role)
     VALUES (?, ?, ?, ?, ?, ?)
    `;

    const [result] = await pool.execute(query, [
     Nama,
     No_telp,
     Alamat,
     Email,
     hashedPassword,
     Role,
    ]);

    return {
     success: true,
     message: "User created successfully",
     userId: result.insertId,
     data: {
      Nama,
      Email,
      No_telp,
      Alamat,
      Role,
     },
    };
   } catch (error) {
    if (error.code === "ER_DUP_ENTRY") {
     return {
      success: false,
```

```
        message: "Email already exists",
      };
    }
    return {
      success: false,
      message: "Failed to create user: " + error.message,
    };
  }
}

// Update user data
static async updateUser(email, userData) {
  try {
    const { Nama, No_telp, Alamat, Role } = userData;

    const query = `
      UPDATE user
      SET Nama = ?, No_telp = ?, Alamat = ?, Role = ?
      WHERE Email = ?
    `;

    const [result] = await pool.execute(query, [
      Nama,
      No_telp,
      Alamat,
      Role,
      email,
    ]);

    if (result.affectedRows === 0) {
      return {
        success: false,
        message: "User not found",
      };
    }

    return {
      success: true,
      message: "User updated successfully",
      data: {
        Nama,
        Email: email,
        No_telp,
        Alamat,
        Role,
```

```
      },
    };
  } catch (error) {
    return {
      success: false,
      message: "Failed to update user: " + error.message,
    };
  }
}

// Delete user
static async deleteUser(email) {
  try {
    // Check if user has active reservations
    const checkQuery = `
      SELECT COUNT(*) as count
      FROM reservasi
      WHERE Email = ? AND Status IN ('Telat/Belum Bayar', 'Aktif/Lunas')
    `;
    const [checkResult] = await pool.execute(checkQuery, [email]);

    if (checkResult[0].count > 0) {
      return {
        success: false,
        message: "Cannot delete user with active reservations",
      };
    }

    const query = "DELETE FROM user WHERE Email = ?";
    const [result] = await pool.execute(query, [email]);

    if (result.affectedRows === 0) {
      return {
        success: false,
        message: "User not found",
      };
    }

    return {
      success: true,
      message: "User deleted successfully",
    };
  } catch (error) {
    return {
      success: false,
```

```
      message: "Failed to delete user: " + error.message,
    };
  }
 }
}

module.exports = User;
```

Tabel 4. 18 Codingan UserToken.js

UserToken.js

```
const { pool } = require("../config/database");

class UserToken {
 // Verify if token is valid and not expired
 static async verifyToken(token) {
  try {
   const query = `
    SELECT * FROM user_tokens
    WHERE Token = ? AND Expires_At > NOW() AND Used = 0
   `;

   const [rows] = await pool.execute(query, [token]);
   return rows[0] || null;
  } catch (error) {
   throw error;
  }
 }

 // Mark token as used
 static async markTokenAsUsed(token) {
  try {
   const query = "UPDATE user_tokens SET Used = 1 WHERE Token = ?";
   const [result] = await pool.execute(query, [token]);

   return {
    success: result.affectedRows > 0,
    message:
     result.affectedRows > 0 ? "Token marked as used" : "Token not found",
   };
  } catch (error) {
   throw error;
  }
 }
```

```
  // Get token by email
  static async getByEmail(email) {
    try {
      const query = `
        SELECT * FROM user_tokens
        WHERE Email = ? AND Expires_At > NOW() AND Used = 0
        ORDER BY Created_At DESC
        LIMIT 1
      `;

      const [rows] = await pool.execute(query, [email]);
      return rows[0] || null;
    } catch (error) {
      throw error;
    }
  }

  // Clean up expired tokens
  static async cleanupExpiredTokens() {
    try {
      const query = "DELETE FROM user_tokens WHERE Expires_At <=
NOW()";
      const [result] = await pool.execute(query);

      return {
        success: true,
        deletedCount: result.affectedRows,
      };
    } catch (error) {
      throw error;
    }
  }

  // Get all tokens for admin view
  static async getAll() {
    try {
      const query = `
        SELECT
          ut.*,
          u.Nama
        FROM user_tokens ut
        LEFT JOIN user u ON ut.Email = u.Email
        ORDER BY ut.Created_At DESC
      `;
```

```
    const [rows] = await pool.execute(query);
    return rows;
   } catch (error) {
    throw error;
   }
  }
}

module.exports = UserToken;
```

Tabel 4. 19 Codingan Auth.js

	Auth.js

```
const express = require("express");
const router = express.Router();
const AuthController = require("../controllers/AuthController");
const {
  validateRegistration,
  validateLogin,
} = require("../middleware/validation");
const { authenticateToken, requireAdmin } = require("../middleware/auth");
const { handleUpload } = require("../middleware/upload");

// Public routes
router.post("/register", validateRegistration, AuthController.register);
router.post("/login", validateLogin, AuthController.login);
router.post("/login-token", AuthController.loginWithToken); // New token-
based login

// Protected routes
router.get("/profile", authenticateToken, AuthController.getProfile);
router.put(
  "/profile",
  authenticateToken,
  handleUpload,
  AuthController.updateProfile
);
router.put(
  "/change-password",
  authenticateToken,
  AuthController.changePassword
);

// Admin routes for user management
```

```
router.get("/admin/users", authenticateToken, requireAdmin,
AuthController.getAllUsers);
router.get("/admin/users-reservations", authenticateToken, requireAdmin,
AuthController.getAllUsersWithReservations);
router.get("/admin/payments", authenticateToken, requireAdmin,
AuthController.getAllPayments);
router.get("/admin/filter-options", authenticateToken, requireAdmin,
AuthController.getFilterOptions);
router.get("/admin/dashboard-stats", authenticateToken, requireAdmin,
AuthController.getDashboardStats);
router.put("/admin/reservations/:reservasiId/status", authenticateToken,
requireAdmin, AuthController.updateReservationStatus);
router.put("/admin/payments/:paymentId/status", authenticateToken,
requireAdmin, AuthController.updatePaymentStatus);

module.exports = router;
```

Tabel 4. 20 Codingan kamar.js

kamar.js

```
const express = require("express");
const router = express.Router();
const KamarController = require("../controllers/KamarController");
const { authenticateToken, requireAdmin } = require("../middleware/auth");

// Public routes
router.get("/", KamarController.getAll);
router.get("/available", KamarController.getAvailable);
router.get("/:id", KamarController.getById);

// Admin routes (protected + admin only)
router.post("/", authenticateToken, requireAdmin, KamarController.create);
router.put("/:id", authenticateToken, requireAdmin,
KamarController.update);
router.delete("/:id", authenticateToken, requireAdmin,
KamarController.delete);
router.put(
  "/:id/availability",
  authenticateToken,
  requireAdmin,
  KamarController.updateAvailability
);

module.exports = router;
```

Tabel 4. 21 Codingan payments.js

	payments.js

	const express = require("express");
	const PaymentController = require("../controllers/PaymentController");
	const { authenticateToken, requireAdmin } = require("../middleware/auth");
	const { uploadPaymentProof, upload } = require("../middleware/upload");
	
	const router = express.Router();
	
	// User routes
	router.post(
	"/reservation/:reservationId/upload-proof",
	authenticateToken,
	uploadPaymentProof,
	PaymentController.uploadPaymentProof
	);
	
	// Update payment proof
	
	// Penghuni: update bukti pembayaran
	router.put(
	"/:paymentId/upload-proof",
	authenticateToken,
	upload.single("bukti_pembayaran"),
	PaymentController.updatePaymentProof
	);
	
	// Admin: update status pembayaran
	router.put(
	"/:paymentId/update-status",
	authenticateToken,
	requireAdmin,
	PaymentController.updatePaymentStatus
	);
	
	router.get(
	"/reservation/:reservationId/history",
	authenticateToken,
	PaymentController.getPaymentHistory
	);
	
	router.get(
	"/unpaid/:reservationId",
	authenticateToken,
	PaymentController.getUnpaidPaymentByReservation

```
);

router.get(
  "/my-payments",
  authenticateToken,
  PaymentController.getMyPaymentHistory
);

router.get(
  "/my-monthly-payments",
  authenticateToken,
  PaymentController.getMonthlyPaymentHistory
);

// Legacy payment creation support
router.post(
  "/payments",
  authenticateToken,
  upload.single("buktiPembayaran"),
  PaymentController.createPayment
);

// Admin routes
router.get(
  "/all",
  authenticateToken,
  requireAdmin,
  PaymentController.getAllPayments
);

router.get(
  "/pending",
  authenticateToken,
  requireAdmin,
  PaymentController.getAllPendingPayments
);

// Monthly payment generation routes
router.post(
  "/generate-monthly",
  authenticateToken,
  requireAdmin,
  PaymentController.generateMonthlyPayments
);
```

```
router.post(
  "/generate-manual",
  authenticateToken,
  requireAdmin,
  PaymentController.generateManualPayments
);

router.get(
  "/generation-history",
  authenticateToken,
  requireAdmin,
  PaymentController.getGenerationHistory
);

router.post(
  "/:paymentId/verify",
  authenticateToken,
  requireAdmin,
  PaymentController.verifyPayment
);

router.post(
  "/update-statuses",
  authenticateToken,
  requireAdmin,
  PaymentController.updateReservationStatuses
);

router.post(
  "/reservation/:reservationId/checkout",
  authenticateToken,
  requireAdmin,
  PaymentController.markUserCheckout
);

// File serving routes
router.get(
  "/:paymentId/proof",
  authenticateToken,
  PaymentController.getPaymentProof
);

module.exports = router;
```

Tabel 4. 22 Codingan pending-reservasi.js

	pending-reservasi.js

```
const express = require("express");
const multer = require("multer");
const path = require("path");
const PendingReservasiController =
require("../controllers/PendingReservasiController");
const { requireAuth, requireAdmin } = require("../middleware/auth");

const router = express.Router();

// Configure multer for file uploads
const storage = multer.diskStorage({
destination: function (req, file, cb) {
   const uploadDir = "uploads/bukti_pembayaran";
   // Create directory if it doesn't exist
   const fs = require("fs");
   if (!fs.existsSync(uploadDir)) {
     fs.mkdirSync(uploadDir, { recursive: true });
   }
   cb(null, uploadDir);
 },
 filename: function (req, file, cb) {
   // Generate unique filename with timestamp
   const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() *
1e9);
   cb(null, "bukti-" + uniqueSuffix + path.extname(file.originalname));
 },
});

const fileFilter = (req, file, cb) => {
 // Accept only image files
 if (file.mimetype.startsWith("image/")) {
   cb(null, true);
 } else {
   cb(new Error("Only image files are allowed!"), false);
 }
};

const upload = multer({
 storage: storage,
 fileFilter: fileFilter,
 limits: {
   fileSize: 5 * 1024 * 1024, // 5MB limit
 },
```

```
});

// Public routes
router.post(
  "/",
  upload.single("buktiPembayaran"),
  PendingReservasiController.create
);

// Admin only routes
router.get("/", requireAuth, requireAdmin,
PendingReservasiController.getAll);
router.get(
  "/:id",
  requireAuth,
  requireAdmin,
  PendingReservasiController.getById
);
router.put(
  "/:id/approve",
  requireAuth,
  requireAdmin,
  PendingReservasiController.approve
);
router.put(
  "/:id/reject",
  requireAuth,
  requireAdmin,
  PendingReservasiController.reject
);
router.delete(
  "/:id",
  requireAuth,
  requireAdmin,
  PendingReservasiController.delete
);
router.get(
  "/:id/bukti-pembayaran",
  requireAuth,
  requireAdmin,
  PendingReservasiController.getBuktiPembayaran
);

module.exports = router;
```

Tabel 4. 23 Codingan reservasi.js

	reservasi.js

	const express = require("express");
	const router = express.Router();
	const ReservasiController = require("../controllers/ReservasiController");
	const { authenticateToken, requireAdmin } = require("../middleware/auth");
	const { validateReservation } = require("../middleware/validation");
	
	// User routes (protected)
	router.post(
	"/",
	authenticateToken,
	validateReservation,
	ReservasiController.create
	);
	router.get(
	"/my-reservations",
	authenticateToken,
	ReservasiController.getUserReservations
	);
	
	// Get reservations by user ID (for admin or user themselves)
	router.get(
	"/user/:userID",
	authenticateToken,
	ReservasiController.getReservasiByUser
	);
	
	// Admin routes (protected + admin only)
	router.get("/all", authenticateToken, requireAdmin,
	ReservasiController.getAll);
	router.get(
	"/admin/with-payments",
	authenticateToken,
	requireAdmin,
	ReservasiController.getReservationsWithPayments
	);
	router.put(
	"/:id/status",
	authenticateToken,
	requireAdmin,
	ReservasiController.updateStatus
	);
	
	module.exports = router;

Tabel 4. 24 Codingan scheduler-control.js

scheduler-control.js

```
const express = require("express");
const router = express.Router();
const auth = require("../middleware/auth");
const paymentScheduler = require("../services/payment-scheduler");

// Get scheduler status (admin only)
router.get("/status", auth.authenticateToken, auth.requireAdmin, (req, res)
=> {
  try {
    const status = paymentScheduler.getDetailedStatus();
    res.json({
      success: true,
      data: status,
    });
  } catch (error) {
    res.status(500).json({
    success: false,
      message: "Failed to get scheduler status",
      error: error.message,
    });
  }
});

// Manual trigger payment generation (admin only)
router.post(
  "/trigger-manual",
  auth.authenticateToken,
  auth.requireAdmin,
  async (req, res) => {
    try {
      const { month, year } = req.body;

      if (!month || !year) {
        return res.status(400).json({
          success: false,
          message: "Month and year are required",
        });
      }

      const result = await paymentScheduler.triggerManualGeneration(
        month,
```

```
      year
    );
    res.json(result);
  } catch (error) {
    res.status(500).json({
    success: false,
      message: "Failed to trigger manual generation",
      error: error.message,
    });
  }
 }
);

// Start scheduler (admin only)
router.post("/start", auth.authenticateToken, auth.requireAdmin, (req, res)
=> {
 try {
  paymentScheduler.start();
  res.json({
    success: true,
    message: "Payment scheduler started",
  });
 } catch (error) {
  res.status(500).json({
  success: false,
    message: "Failed to start scheduler",
    error: error.message,
  });
 }
});

// Stop scheduler (admin only)
router.post("/stop", auth.authenticateToken, auth.requireAdmin, (req, res)
=> {
 try {
  paymentScheduler.stop();
  res.json({
    success: true,
    message: "Payment scheduler stopped",
  });
 } catch (error) {
  res.status(500).json({
  success: false,
    message: "Failed to stop scheduler",
    error: error.message,
```

```
    });
  }
});

module.exports = router;
```

Tabel 4. 25 Codingan tmp-users.js

tmp-users.js

```
const express = require("express");
const multer = require("multer");
const path = require("path");
const TmpUserController = require("../controllers/TmpUserController");
const { requireAuth, requireAdmin } = require("../middleware/auth");

const router = express.Router();

// Configure multer for file uploads
const storage = multer.diskStorage({
destination: function (req, file, cb) {
   const uploadDir = "uploads/bukti_pembayaran";
   // Create directory if it doesn't exist
   const fs = require("fs");
   if (!fs.existsSync(uploadDir)) {
     fs.mkdirSync(uploadDir, { recursive: true });
   }
   cb(null, uploadDir);
  },
  filename: function (req, file, cb) {
   // Generate unique filename with timestamp
   const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() *
1e9);
   cb(null, "bukti-" + uniqueSuffix + path.extname(file.originalname));
  },
});

const fileFilter = (req, file, cb) => {
  // Accept only image files
  if (file.mimetype.startsWith("image/")) {
   cb(null, true);
  } else {
   cb(new Error("Only image files are allowed!"), false);
  }
};
```

```
const upload = multer({
  storage: storage,
  fileFilter: fileFilter,
  limits: {
    fileSize: 5 * 1024 * 1024, // 5MB limit
  },
});

// Public routes
router.post("/", upload.single("buktiPembayaran"),
TmpUserController.create);

// Admin only routes
router.get("/", requireAuth, requireAdmin, TmpUserController.getAll);
router.get("/:id", requireAuth, requireAdmin, TmpUserController.getById);
router.put(
  "/:id/approve",
  requireAuth,
  requireAdmin,
  TmpUserController.approve
);
router.put("/:id/reject", requireAuth, requireAdmin,
TmpUserController.reject);
router.get(
  "/:id/bukti-pembayaran",
  requireAuth,
  requireAdmin,
  TmpUserController.getBuktiPembayaran
);

// New reservation management routes (Admin only)
router.get("/admin/reservations", requireAuth, requireAdmin,
TmpUserController.getAllReservations);
router.put("/admin/reservations/:id/status", requireAuth, requireAdmin,
TmpUserController.updateReservationStatus);
router.get("/admin/filter-options", requireAuth, requireAdmin,
TmpUserController.getFilterOptions);

module.exports = router;
```

users.js

```
const express = require("express");
const router = express.Router();
const UserController = require("../controllers/UserController");
const { authenticateToken, requireAdmin } = require("../middleware/auth");
```

```
// Admin routes (protected + admin only)
router.get("/", authenticateToken, requireAdmin,
UserController.getAllUsers);
router.get("/:email", authenticateToken, requireAdmin,
UserController.getUserByEmail);
router.post("/", authenticateToken, requireAdmin,
UserController.createUser);
router.put("/:email", authenticateToken, requireAdmin,
UserController.updateUser);
router.delete("/:email", authenticateToken, requireAdmin,
UserController.deleteUser);

module.exports = router;
```

I. Poster

Gambar 4. 4 Poster

J. Video Teaser

https://drive.google.com/drive/folders/1GoRAiEMlrTjPmvMjwuuSBS9eric-UJhT?usp=sharing

K. Slide Presentasi

https://drive.google.com/drive/folders/1dqDMkH0Nzd-FLSY5-1cIqBennKtpC5Z4?usp=sharing