Final Report Pengembangan Aplikasi Telematika

Full PAT Project

Oleh:

Nicholas ChandraNRP: C11220003

PROGRAM STUDI TEKNIK ELEKTRO

FAKULTAS TEKNOLOGI

INDUSTRI UNIVERSITAS

KRISTEN PETRA

SURABAYA

2025

I. Imagine a System

1. Manajemen Reservasi Ruangan

Sistem telematika untuk **manajemen reservasi ruangan** memanfaatkan teknologi jaringan, sensor, serta platform perangkat lunak untuk memantau, mengelola, dan mengoptimalkan proses pemesanan dan penggunaan ruangan di berbagai fasilitas seperti kantor, gedung pertemuan, universitas, atau hotel. Tujuan utamanya adalah untuk menyediakan visibilitas real-time terhadap ketersediaan ruangan, efisiensi penggunaan ruang, serta kemudahan akses bagi pengguna dan pengelola fasilitas.

Entitas yang Terlibat :

- **Perangkat Pengakses (Smartphone / Kiosk):** Digunakan oleh pengguna untuk melihat ketersediaan dan melakukan reservasi.
- **Ruangan Fisik:** Lokasi-lokasi yang dapat dipesan, seperti ruang rapat, ruang kelas, atau aula.
- **Sistem Informasi Manajemen Ruangan:** Aplikasi pusat yang mengelola data reservasi, jadwal, dan ketersediaan ruangan.
- **Administrator Fasilitas:** Pihak yang mengatur konfigurasi ruangan, kapasitas, dan jadwal operasional.
- **Pengguna (Staff / Publik / Mahasiswa):** Individu yang melakukan pemesanan ruangan untuk keperluan tertentu.
- **Layar Informasi Ruangan (Display Panel):** Monitor yang menampilkan status dan jadwal penggunaan ruangan secara langsung di depan pintu.
- **Check In QRIS (Opsional):** Memudahkan cara check in ruangan memanfaatkan fitur scan QRIS melalui aplikasi.

Fungsi Utama :

- **Reservasi Ruangan Real-time:** Pengguna dapat melihat ketersediaan dan memesan ruangan secara instan.
- **Manajemen Jadwal dan Ketersediaan:** Sistem mengelola waktu penggunaan dan menghindari konflik jadwal.
- **Pemeriksaan Kehadiran Otomatis:** Integrasi sensor atau QR code untuk mencatat kehadiran pengguna yang telah memesan.
- **Notifikasi dan Pengingat:** Memberikan peringatan terkait waktu reservasi, pembatalan, atau konflik jadwal.
- **Laporan Penggunaan Ruangan:** Menyediakan data statistik penggunaan ruangan untuk evaluasi efisiensi.
- **Kontrol Akses Otomatis (opsional):** Integrasi dengan sistem kunci pintar untuk membuka ruangan hanya bagi yang terdaftar.
- **Integrasi Kalender:** Sinkronisasi dengan kalender internal organisasi (Google Calendar, Outlook, dll).

II. Design a System

1. Pengantar

Dalam lingkungan kerja modern, pengelolaan ruang meeting yang efisien menjadi tantangan bagi banyak perusahaan. Sistem Manajemen Reservasi Ruang Meeting dirancang untuk membantu karyawan dalam mencari, memesan, dan mengelola ruang meeting dengan mudah melalui aplikasi desktop dan mobile yang terhubung ke server. Sistem ini dipilih karena sering terjadi bentrokan jadwal pemakaian ruang, kurangnya visibilitas ketersediaan ruangan, serta kebutuhan akan sistem terpusat yang dapat diakses dengan mudah oleh seluruh karyawan.

2. Analisa Kebutuhan

Sistem ini memiliki 3 entitas utama, yaitu :

1) **Server :**
    a) Database untuk menyimpan data ruang, reservasi, pengguna, dan jadwal.
    b) API untuk komunikasi dengan aplikasi desktop dan mobile.
    c) Keamanan data dengan autentikasi Username dan Password.
2) **Aplikasi Desktop :**
    a) Login dan autentikasi administrator atau IT.
    b) Input, update, dan penghapusan data ruang meeting.
    c) Pembuatan laporan penggunaan ruang meeting.
    d) Dashboard untuk melihat statistik reservasi.
3) **Aplikasi Mobile :**
    a) Login dan autentikasi staff perusahaan.
    b) Pengecekan ketersediaan ruang meeting secara real-time.
    c) Reservasi ruang meeting.
    d) Notifikasi pengingat jadwal meeting.

3. Desain Arsitektur

Server berfungsi sebagai pusat data utama dalam sistem ini, dimana bekerja dalam menyimpan informasi ruang meeting, jadwal reservasi, data pengguna, serta menangani proses autentikasi dan otorisasi. Selain itu, server juga bertanggung jawab dalam menangani permintaan dari aplikasi desktop dan mobile melalui API berbasis

REST, memastikan bahwa informasi yang diberikan selalu up-to-date dan dapat diakses dengan aman. Sistem keamanan seperti autentikasi berbasis token dan enkripsi data juga diterapkan untuk menjaga integritas dan keamanan data pengguna.

Aplikasi desktop dirancang untuk administrator atau tim IT dalam melakukan pemantauan dan pengelolaan sistem. Administrator dapat mengakses dashboard untuk melihat status ruang meeting secara real-time, mengelola daftar ruang meeting dengan menambahkan, mengedit, atau menghapus data ruang, serta memeriksa dan mengelola jadwal reservasi. Selain itu, administrator juga dapat melihat log aktivitas pengguna, menganalisis laporan penggunaan ruang meeting, dan memastikan bahwa sistem berjalan dengan optimal. Dengan adanya fitur pemantauan dan manajemen ini, administrator memiliki kontrol penuh terhadap sistem untuk memastikan kelancaran operasional.

Aplikasi mobile ditujukan bagi pengguna atau staf perusahaan agar mereka dapat dengan mudah mencari dan memesan ruang meeting yang tersedia. Pengguna dapat melihat daftar ruang meeting beserta informasi ketersediaan, melakukan reservasi dengan UI yang mudah dimengerti, serta menerima notifikasi pengingat sebelum waktu meeting dimulai. Terdapat fitur tambahan seperti pembatalan reservasi, pemberian feedback terhadap fasilitas ruang meeting, serta integrasi dengan kalender perusahaan juga disediakan untuk meningkatkan pengalaman pengguna dalam mengelola kebutuhan meeting mereka.

4. Desain Layanan

1. Autentikasi Pengguna (AuthAPI)	
End Point	app/api/auth/login
Method	POST

Parameter	{
	email : Staff@api.com
	password : 123
	}

Response	{
	key / token : gK3w (Untuk akses masuk ke suatu halaman private)
	role : Staff
	}

2. Melihat Daftar Ruang Meeting (RoomAPI)	
End Point	app/api/rooms
Method	GET

Response	{
	room_id : 0001
	name : Ruang Direktur
	capacity : 4
	facilities : AC, LCD, Meja Direktur
	status : Free
	}

3. Reservasi Ruangan (ReserveAPI)	
End Point	app/api/reservation
Method	POST

Parameter	{
	user_id : b25013
	room_id : 0022
	date : 2024-12-22
	start_time : 12:00
	end_time : 13:00
	}

Response	{
	status : reserved
	reservations_id : 002
	}

4. Membatalkan Reservasi (CancelAPI)	
End Point	app/api/reservation/{id}
Method	DELETE

Response	{
	status : canceled
	}

5. Melihat Jadwal Reservasi (CheckRoomAPI)	
End Point	app/api/rooms/{room_id}/reservations
Method	GET

Response	{
	reservation_id : 212
	room_name : Ruang Tamu
	date : 2025-01-24
	start_time : 08:30
	end_time : 10:00
	}

6. Check In Reservasi (CheckInAPI)	
End Point	app/api/checkin
Method	POST

Parameter	{
	user_id : d13002
	reservation_id : 212
	}

Response	{
	status : Occupied
	}

7. Melihat Laporan Penggunaan Ruang Meeting (ReportAPI)	
End Point	app/api/reports
Method	GET

Parameter	{
	room_id : 0010
	start_date : 2024-01-01
	end_date : 2025-01-01
	}

Response	{
	room_name : Auditorium
	total_reservations : 18
	usage_hours : 72
	}

8. Memberikan Feedback Ruang Meeting (FeedbackAPI)	
End Point	app/api/feedback
Method	POST

Parameter	{
	user_id : Staff
	room_id : 123
	rating : 5
	comments : Kurang dingin
	}

Response	{
	status : success
	}

5. Desain Protokol

1) Menggunakan format komunikasi JSON melalui via HTTP(S)
2) Autentikasi suatu user menggunakan Json Web Tokens (JWT) yang dikirim dalam header Authorization. Token ini berfungsi sebagai key access dalam membuka halaman atau page yang berisi data bersifat private
3) Proses Reservasi :
    a. Pengguna (Staff) login dengan email dan password.
    b. Sistem mengembalikan token autentikasi.
    c. Mobile apps memberikan tampilan UI yang berisi daftar ruang meeting yang tersedia dan kategori ruangannya, seperti kapasitas, atribut, dan status ruangan tersebut.
    d. User kemudian dapat memilih ruang dan waktu reservasi.
    e. Mobile apps mengirimkan permintaan reservasi ke server.
    f. Server memeriksa ketersediaan dan menyimpan data reservasi.
    g. User mendapatkan notifikasi dan kode reservasi jika prosesnya berhasil.
    h. User bisa melakukan pembatalan pada reservasi ruangan pada mobile apps.
    i. Pembatalan reservasi pada mobile apps, akan mengirimkan permintaan dalam mencari reservasi yang ingin dibatalkan dan mendelete data mengenai reservasi tersebut di database server.
    j. Pengguna melakukan check-in dengan memasukkan kode reservasi pada saat masuk ke ruang meeting.
    k. Setelah penggunaan selesai, pengguna dapat memberikan feedback.
    l. Hasil dari reservasi yang dilakukan antara mobile apps dengan server dapat dipantau melalui desktop apps yang dipegang oleh seorang IT atau administrator.

III. Server Implementation

1. Daftar API

No	Endpoint	Metode	Deskripsi	Parameter	Akses
1.	/app/api/auth/register	POST	Mendaftarkan user baru	email, password, name, department	Publik
2.	/app/api/auth/login	POST	Autentikasi user	email, password	Publik
3.	/app/api/rooms	GET	Mendapatkan daftar semua ruangan	-	Semua User
4.	/app/api/rooms/:id	GET	Mendapatkan detail ruangan berdasarkan ID	id(path)	Semua User
5.	/app/api/reservation	POST	Membuat reservasi ruangan baru	user_id, room_id, date, start_time, end_time	Semua User
6.	/app/api/reservation/:id	DELETE	Membatalkan reservasi	id(path)	Semua User
7.	/app/api/rooms/:room_id/reservations	GET	Melihat reservasi ruangan tertentu	room_id(path)	Semua User
8.	/app/api/checkin	POST	Melakukan check-in	user_id, reservation_i	Semua User

			reservasi	d	
9.	/app/api/reports	GET	Mendapatkan laporan penggunaan ruangan	room_id, start_date, end_date (query)	Admin
10.	/app/api/feedback	POST	Memberikan feedback terhadap ruangan	user_id, room_id, rating, comments	Semua User
11.	/app/api/rooms	POST	Menambahkan ruangan baru	name, capacity, facilities	Admin
12.	/app/api/rooms/:id	PUT	Memperbarui informasi ruangan	name, capacity, facilities	Admin
13.	/app/api/rooms/:id	DELETE	Menghapus ruangan	id(path)	Admin
14.	/app/api/users/:id	GET	Mendapatkan informasi pengguna berdasarkan ID	id(path)	Admin, User Sendiri
15.	/app/api/users	GET	Mendapatkan daftar semua pengguna	-	Admin
16.	/app/api/users/:id	PUT	Memperbarui informasi pengguna	email, password, name, department, role	Admin
17.	/app/api/users/:id	DELETE	Menghapus pengguna	id(path)	Admin

2. Kode Server

CUPLIKAN KODE

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 4000;
const JWT_SECRET = process.env.JWT_SECRET || 'reservationsystem2025';

// Middleware
app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// MongoDB Connection
mongoose.connect(process.env.MONGO_URI || 'mongodb://localhost:27017/reservation-system', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => {
  console.log('Connected to MongoDB');
}).catch(err => {
  console.error('MongoDB connection error:', err);
  process.exit(1);
});

// Models
const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['Staff', 'Admin', 'IT'], default: 'Staff' },
```

```
// Models
const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['Staff', 'Admin', 'IT'], default: 'Staff' },
  name: String,
  department: String
});

const roomSchema = new mongoose.Schema({
  room_id: { type: String, required: true, unique: true },
  name: { type: String, required: true },
  capacity: { type: Number, required: true },
  facilities: [String],
  status: { type: String, enum: ['Free', 'Occupied', 'Maintenance'], default: 'Free' }
});

const reservationSchema = new mongoose.Schema({
  reservation_id: { type: String, required: true, unique: true },
  user_id: { type: String, required: true },
  room_id: { type: String, required: true },
  room_name: String,
  date: { type: Date, required: true },
  start_time: { type: String, required: true },
  end_time: { type: String, required: true },
  status: { type: String, enum: ['Reserved', 'Canceled', 'Occupied', 'Completed'], default: 'Reserved' },
  created_at: { type: Date, default: Date.now }
});

const feedbackSchema = new mongoose.Schema({
  user_id: { type: String, required: true },
  room_id: { type: String, required: true },
  reservation_id: String,
  rating: { type: Number, required: true, min: 1, max: 5 },
  comments: String,
  created_at: { type: Date, default: Date.now }
});

const User = mongoose.model('User', userSchema);
const Room = mongoose.model('Room', roomSchema);
const Reservation = mongoose.model('Reservation', reservationSchema);
const Feedback = mongoose.model('Feedback', feedbackSchema);
```

authenticateToken 13 of 16

```
// Authentication Middleware
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) return res.status(401).json({ message: 'Access token required' });

  jwt.verify(token, JWT_SECRET, (err, user) => {
    if (err) return res.status(403).json({ message: 'Invalid or expired token' });
    req.user = user;
    next();
  });
};

// Role-based authorization middleware
const authorize = (roles = []) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Forbidden: Insufficient privileges' });
    }
    next();
  };
};

// Generate unique ID
const generateUniqueId = (prefix) => {
  return `${prefix}${Date.now().toString(36)}${Math.random().toString(36).substr(2, 5)}`;
};

// API Routes

app.get('/', (req, res) => {
    res.send('Server berjalan!');
  });

// Add this after your existing APIs but before the server start code

// 1. User Registration API
app.post('/app/api/auth/register', async (req, res) => {
  try {
    const { email, password, name, department } = req.body;
```

authenticateToken 13 of 16

```
// 1. User Registration API
app.post('/app/api/auth/register', async (req, res) => {
  try {
    const { email, password, name, department } = req.body;

    // Validate required fields
    if (!email || !password) {
      return res.status(400).json({ message: 'Email and password are required' });
    }

    // Check if user already exists
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(409).json({ message: 'User with this email already exists' });
    }

    // Create new user (default role is Staff as defined in schema)
    const newUser = new User({
      email,
      password, // In production, this should be hashed
      name: name || '',
      department: department || '',
      role: 'Staff' // Default role for new registrations
    });

    await newUser.save();

    // Generate JWT token for immediate login
    const token = jwt.sign({
      userId: newUser._id,
      email: newUser.email,
      role: newUser.role
    }, JWT_SECRET, { expiresIn: '24h' });

    return res.status(201).json({
      message: 'Registration successful',
      token,
      role: newUser.role
    });

  } catch (error) {
    console.error('Registration error:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
});
```

authenticateToken 13 of 16

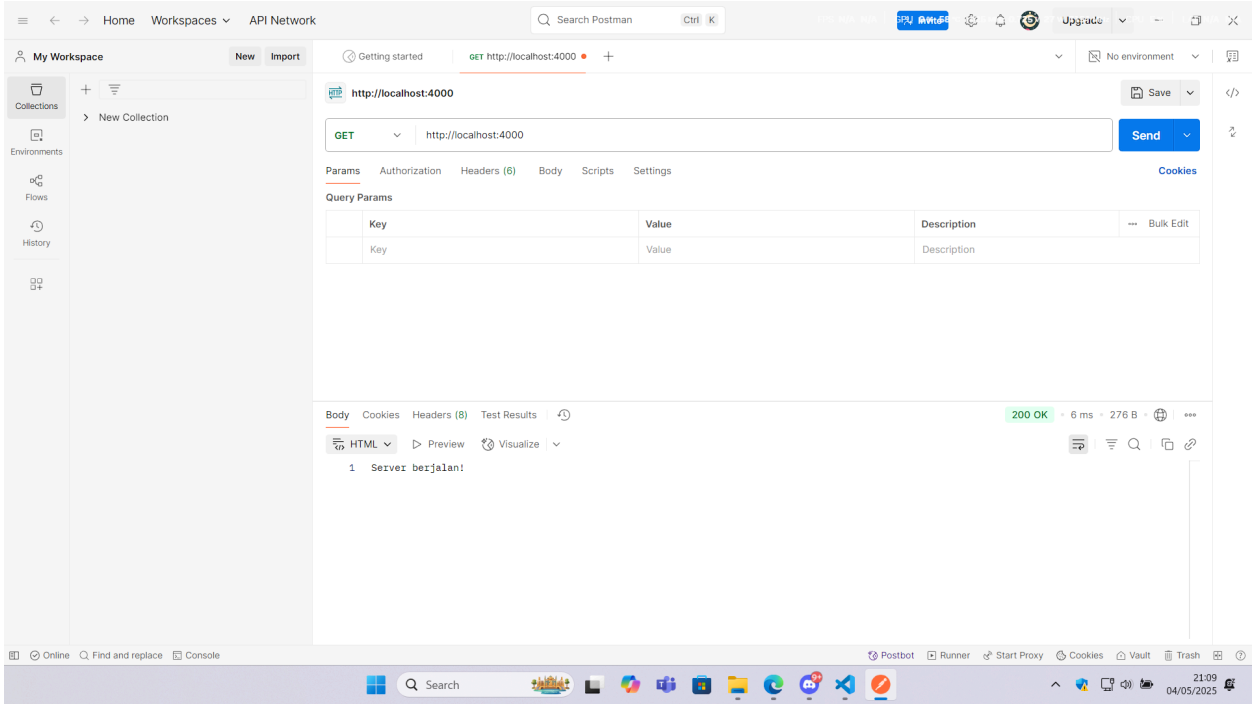
```
// 3. Room API - Get all rooms
app.get('/app/api/rooms', authenticateToken, async (req, res) => {
  try {
    const rooms = await Room.find({});
    return res.status(200).json(rooms);
  } catch (error) {
    console.error('Error fetching rooms:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
});

// Get room details by ID
app.get('/app/api/rooms/:id', authenticateToken, async (req, res) => {
  try {
    const room = await Room.findOne({ room_id: req.params.id });
    if (!room) {
      return res.status(404).json({ message: 'Room not found' });
    }
    return res.status(200).json(room);
  } catch (error) {
    console.error('Error fetching room details:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
});

// 4. Reservation API
app.post('/app/api/reservation', authenticateToken, async (req, res) => {
  try {
    const { user_id, room_id, date, start_time, end_time } = req.body;

    if (!user_id || !room_id || !date || !start_time || !end_time) {
      return res.status(400).json({ message: 'All fields are required' });
    }

    // Check if room exists
    const room = await Room.findOne({ room_id });
    if (!room) {
      return res.status(404).json({ message: 'Room not found' });
    }

    // Check if room is available at requested time
    const reservationDate = new Date(date);
    const existingReservations = await Reservation.find({
      room_id,
      date: {
```

authenticateToken ? of 16

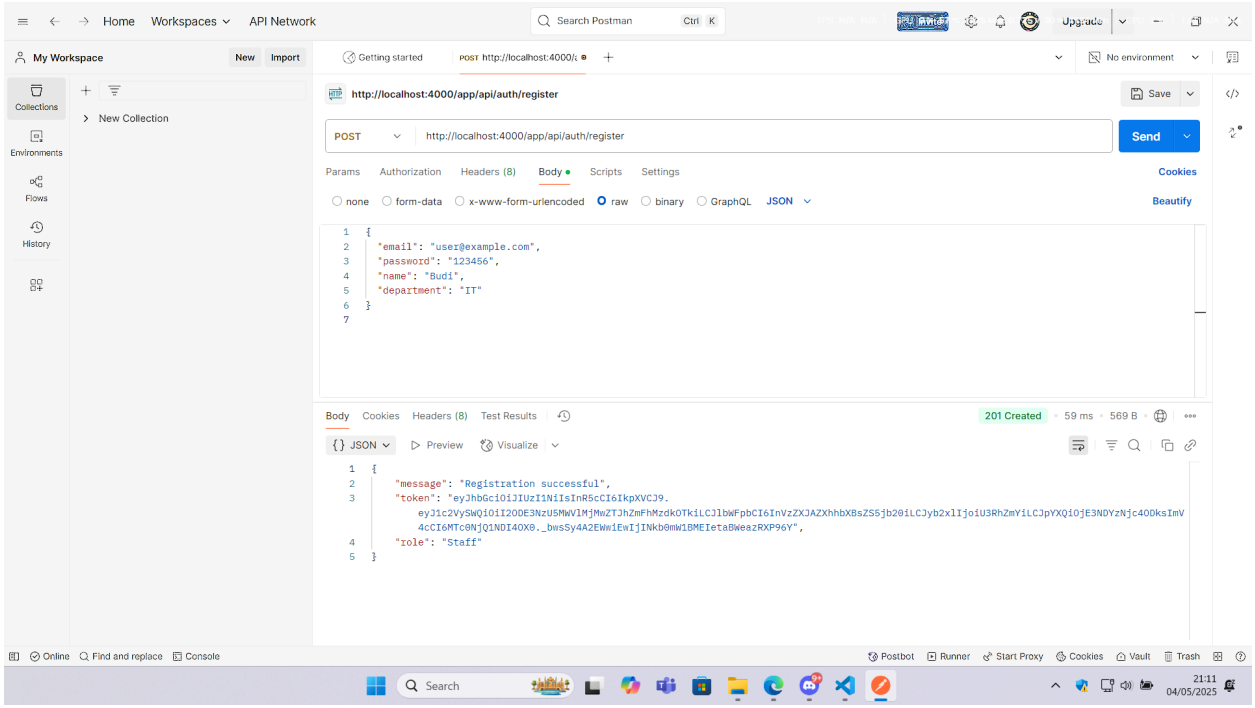
```
// Add new room
app.post('/app/api/rooms', authenticateToken, authorize(['Admin', 'IT']), async (req, res) => {
  try {
    const { name, capacity, facilities } = req.body;

    if (!name || !capacity) {
      return res.status(400).json({ message: 'Room name and capacity are required' });
    }

    const room_id = generateUniqueId('rm');
    const newRoom = new Room({
      room_id,
      name,
      capacity,
      facilities: facilities ? facilities.split(',').map(f => f.trim()) : [],
      status: 'Free'
    });

    await newRoom.save();

    return res.status(201).json({
      status: 'success',
      room_id
    });

  } catch (error) {
    console.error('Error adding room:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
});

// Update room
app.put('/app/api/rooms/:id', authenticateToken, authorize(['Admin', 'IT']), async (req, res) => {
  try {
    const { name, capacity, facilities } = req.body;

    const room = await Room.findOne({ room_id: req.params.id });
    if (!room) {
      return res.status(404).json({ message: 'Room not found' });
    }

    if (name) room.name = name;
    if (capacity) room.capacity = capacity;
    if (facilities) room.facilities = facilities.split(',').map(f => f.trim());
```

III. PENGUJIAN API MELALUI POSTMAN

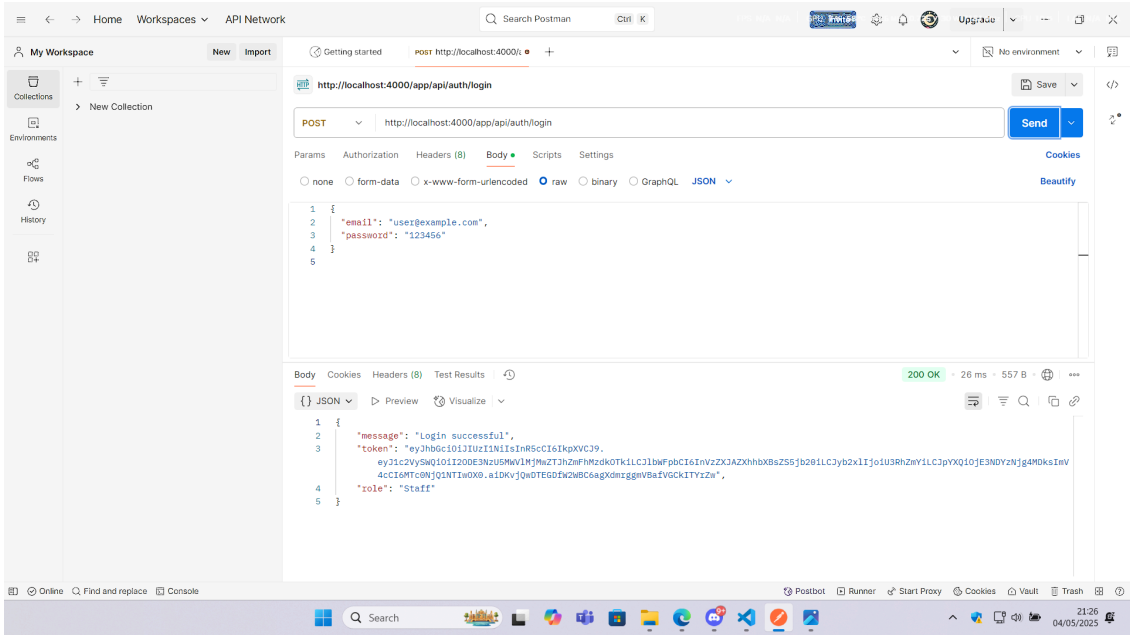
Testing koneksi terhadap sistem



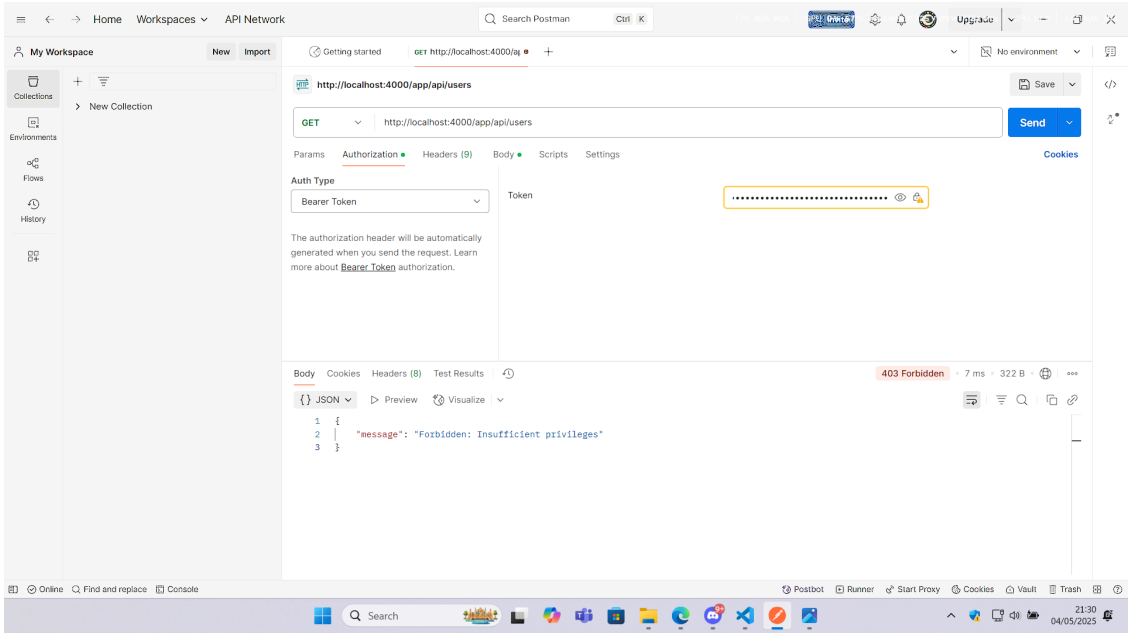
Testing melakukan registrasi user baru untuk masuk ke dalam Database (Proses registrasi hanya berlaku terhadap role “Staff”, sedangkan Admin sudah terdaftar di Database MongoDB melalui Seed Data)



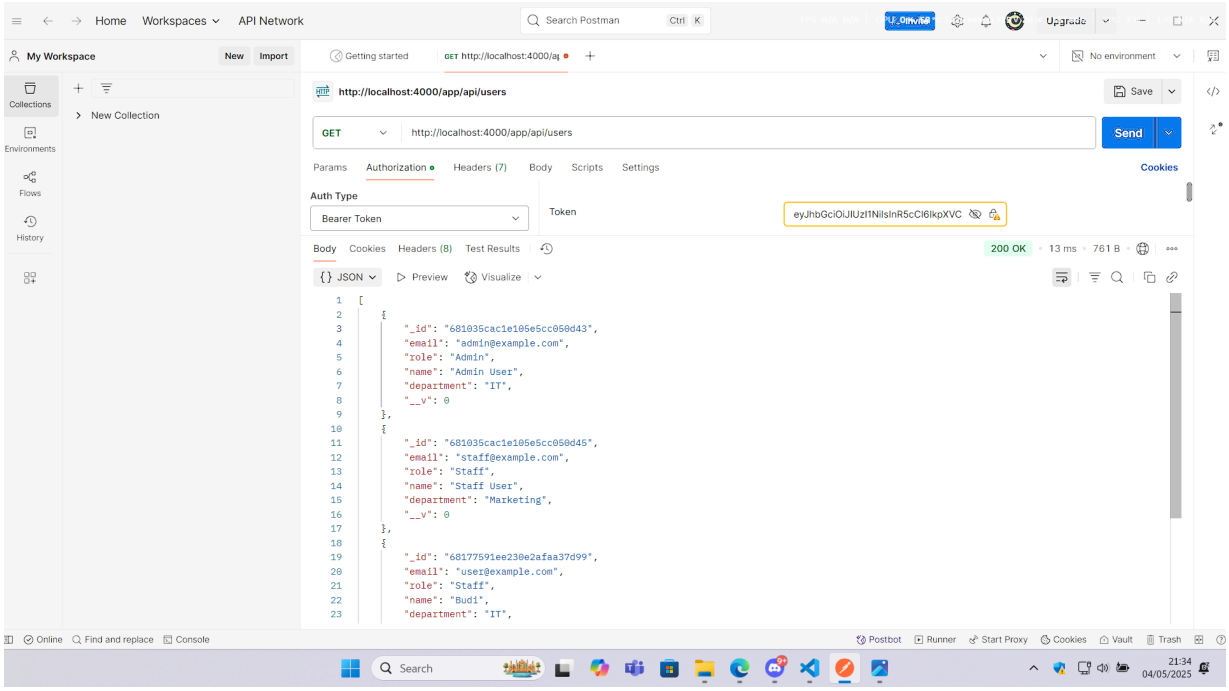
Testing login dengan autentikasi user yang baru saja melakukan registrasi



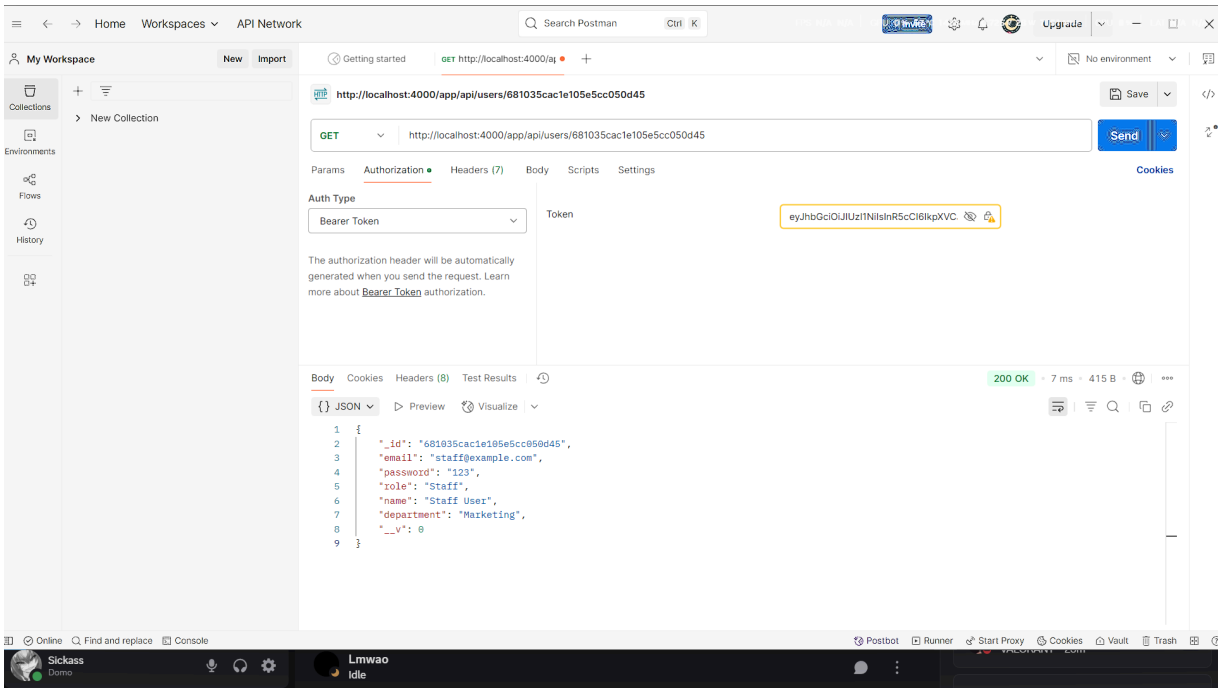
Testing melihat akun atau user siapa saja yang terdaftar di Database, contoh di bawah menggunakan token otoritas user yang memiliki role “Staff”, sehingga akses melihat user di tolak karena hanya user yang memiliki role “Admin” yang bisa melihat data semua user.



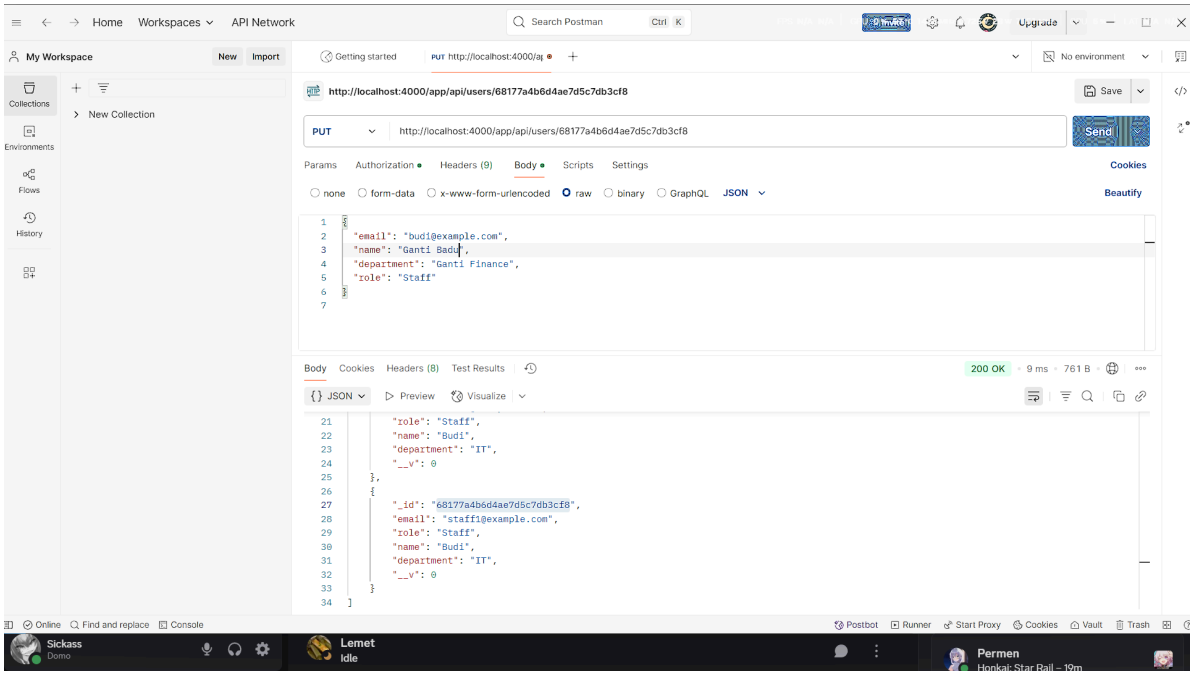
Testing melihat data semua user menggunakan token otoritas “Admin”

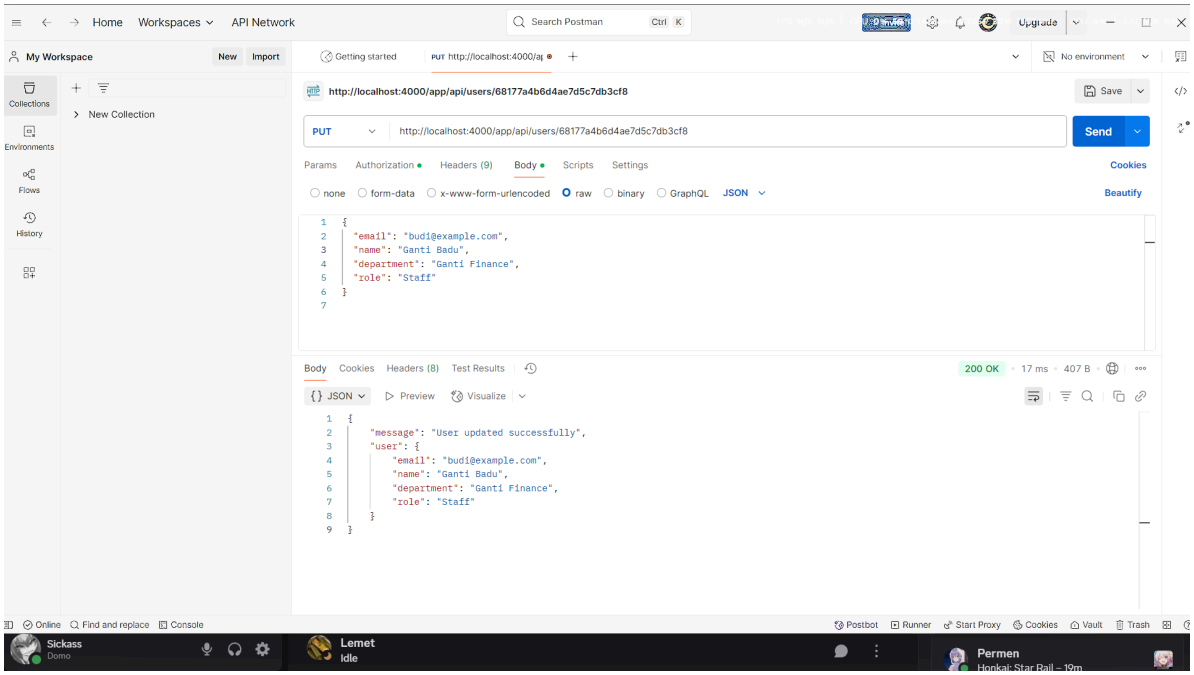


Testing melihat data seorang user melalui ID, hal ini bisa dilakukan dengan token otoritas “Admin” atau user itu sendiri untuk melihat profil diri.

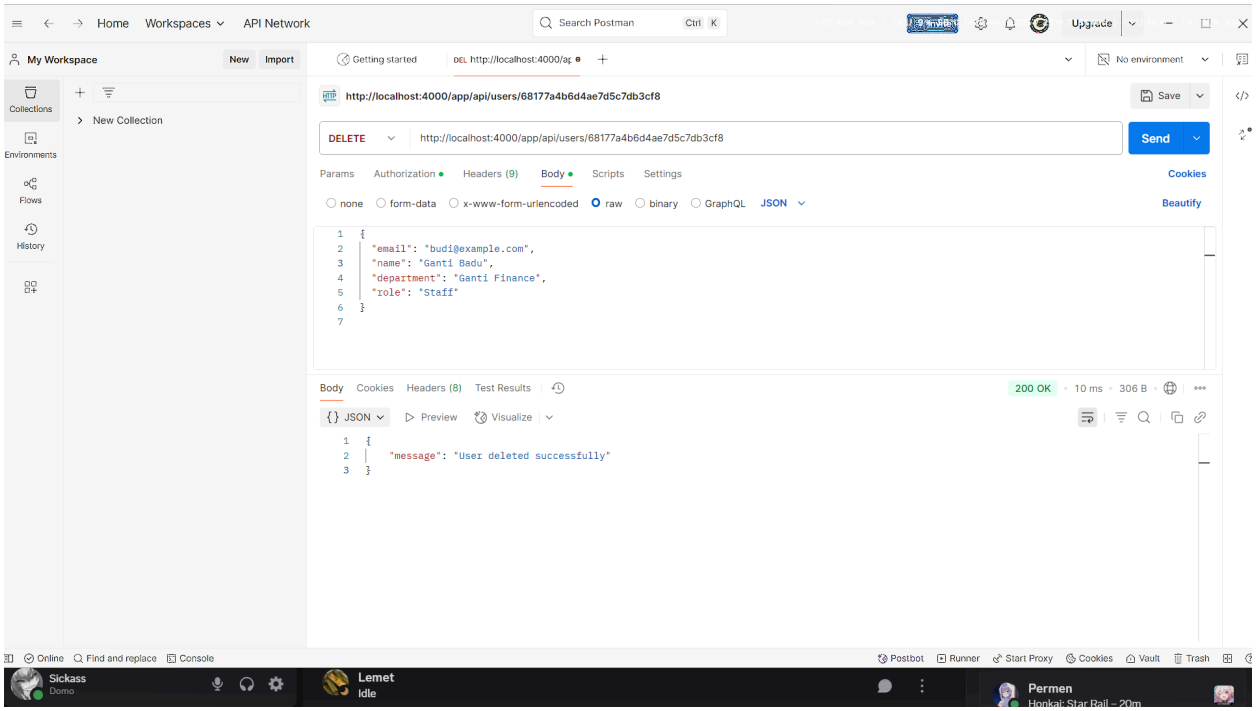


Testing mengganti data seorang user di database melalui ID seorang user dan menggunakan token otoritas “Admin” (tidak bisa diubah oleh “Staff”)

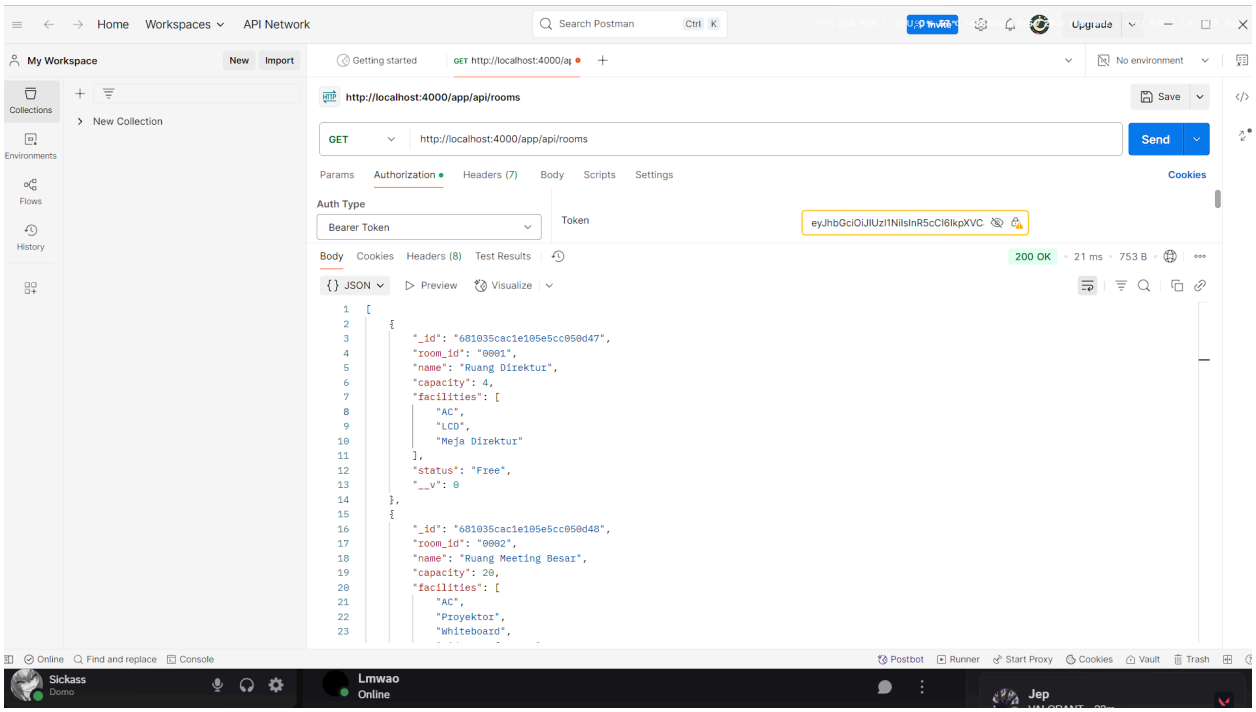




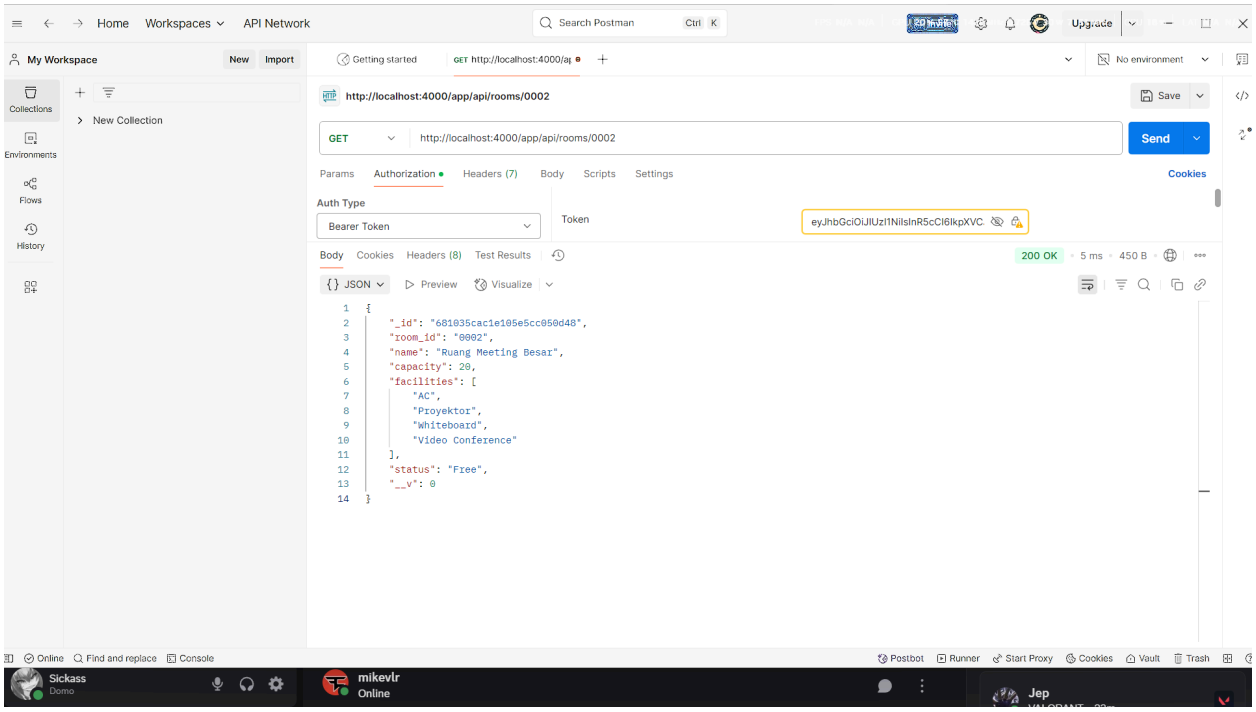
Testing menghapus data user di Database melalui ID dan menggunakan token otoritas “Admin” (tidak bisa diubah oleh “Staff”)



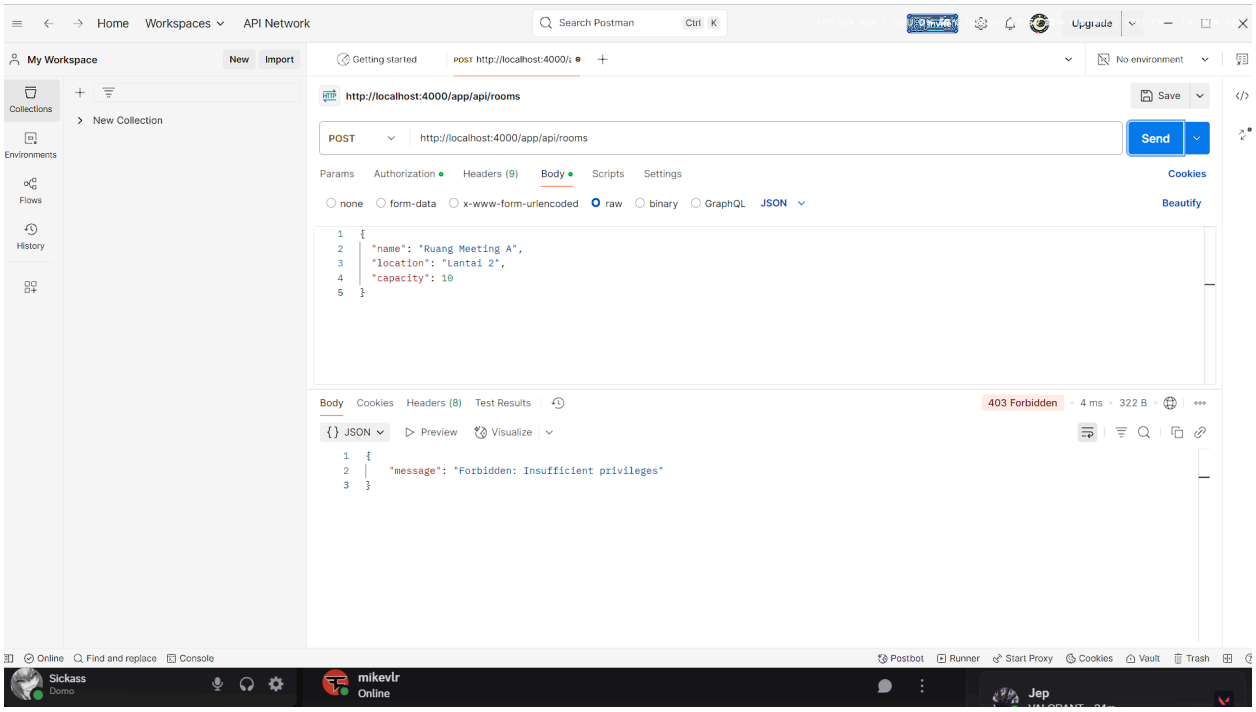
Testing API yang menyertakan ruangan apa saja yang tersedia, deskripsi, dan status ruangan tersebut di Sistem atau Database (Bisa diakses oleh semua user, melalui bearer token)



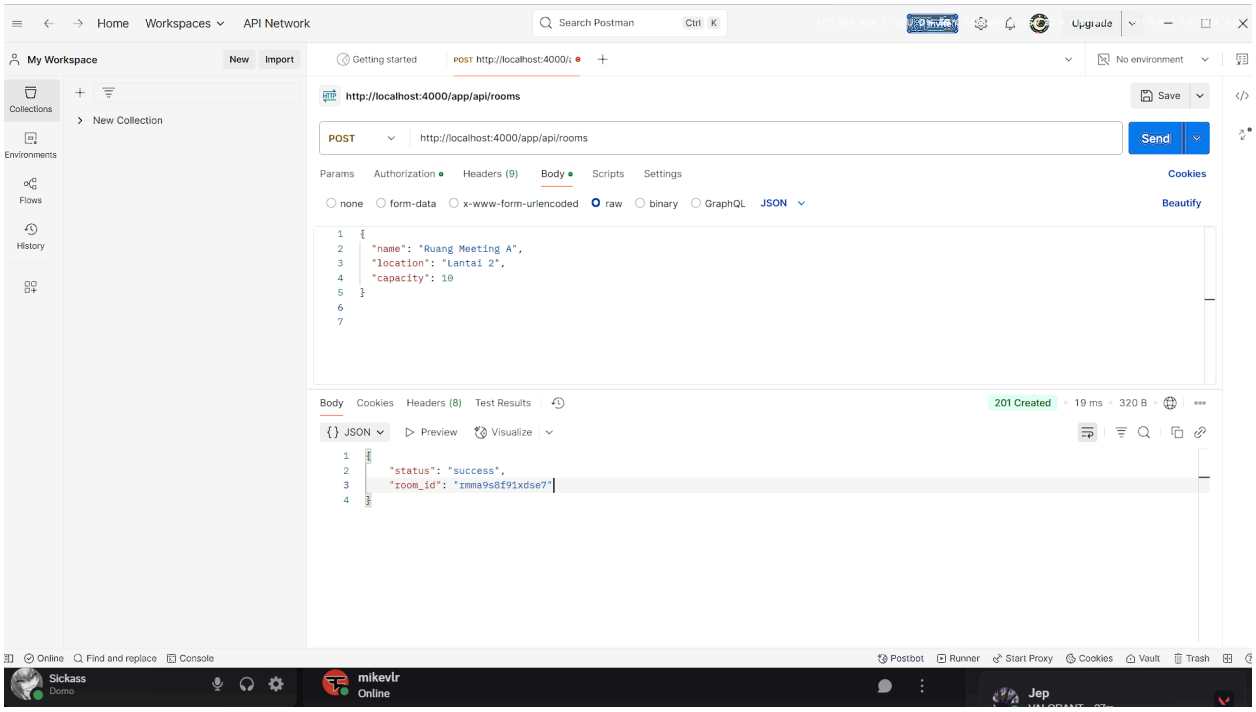
Testing mencari ruangan, deskripsi, dan statusnya melalui Room_ID (Bisa diakses oleh semua user, melalui bearer token)



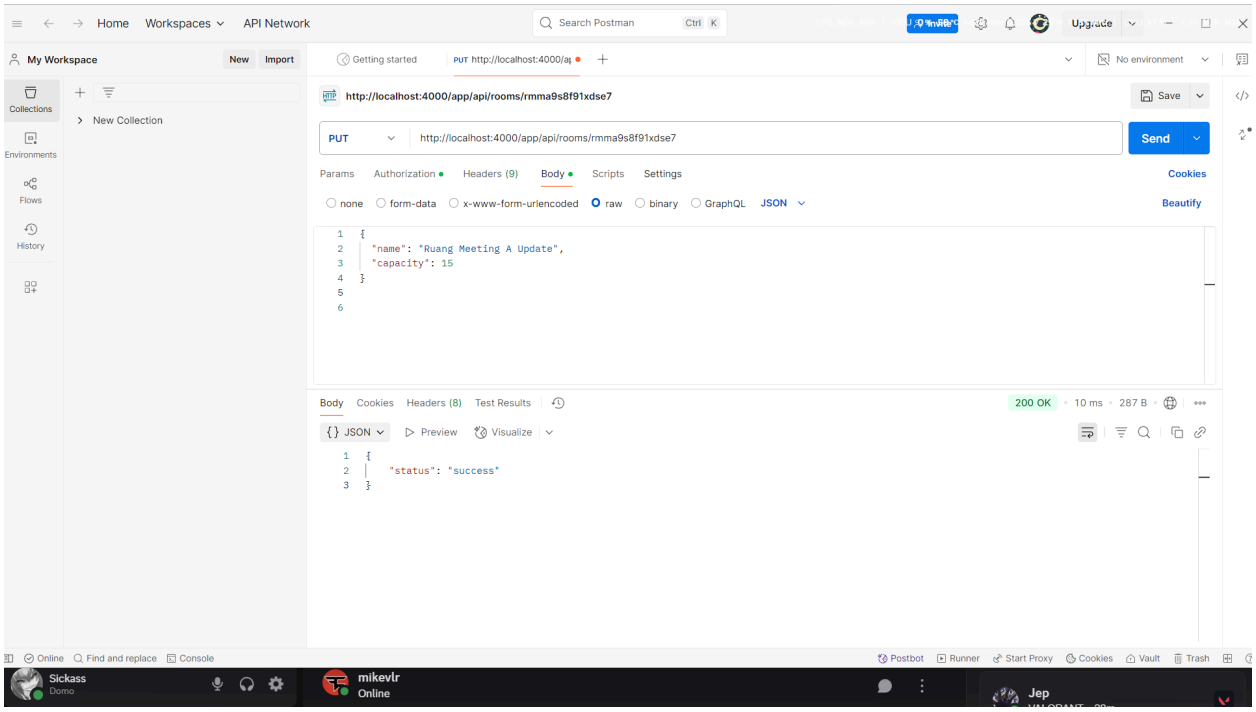
Testing melakukan penambahan ruangan ke dalam Sistem atau Database (Gagal, karena hanya admin yang berwenang dalam pengaturan ruangan)

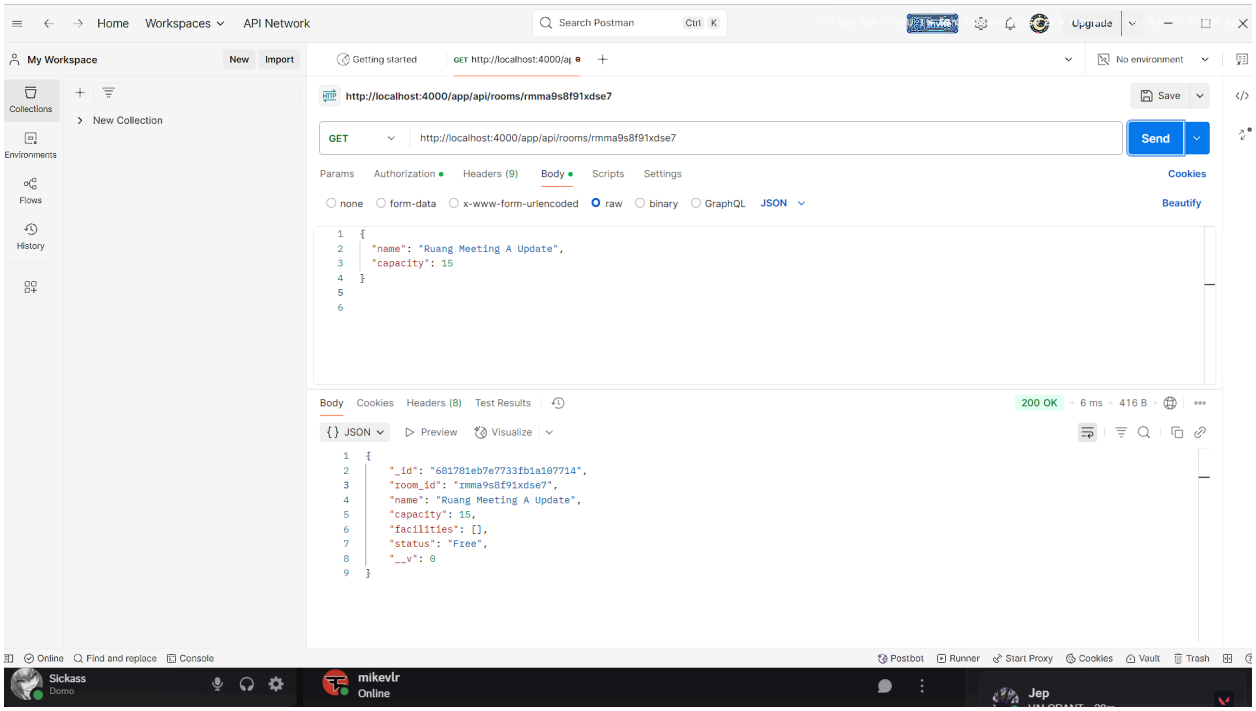


Testing melakukan penambahan ruangan ke dalam Sistem atau Database (Berhasil, karena token otoritas “Admin” dikenali oleh Sistem atau Database)

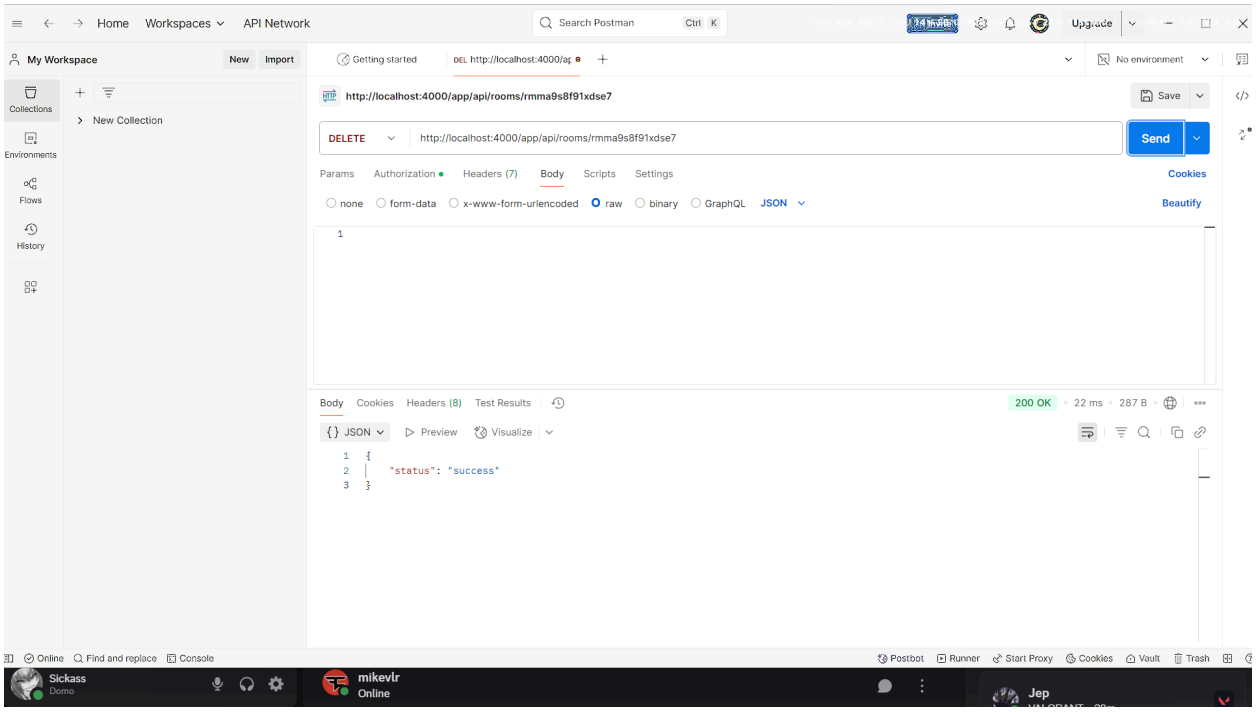


Testing mengubah deskripsi dari suatu ruangan dengan menggunakan token otoritas “Admin”

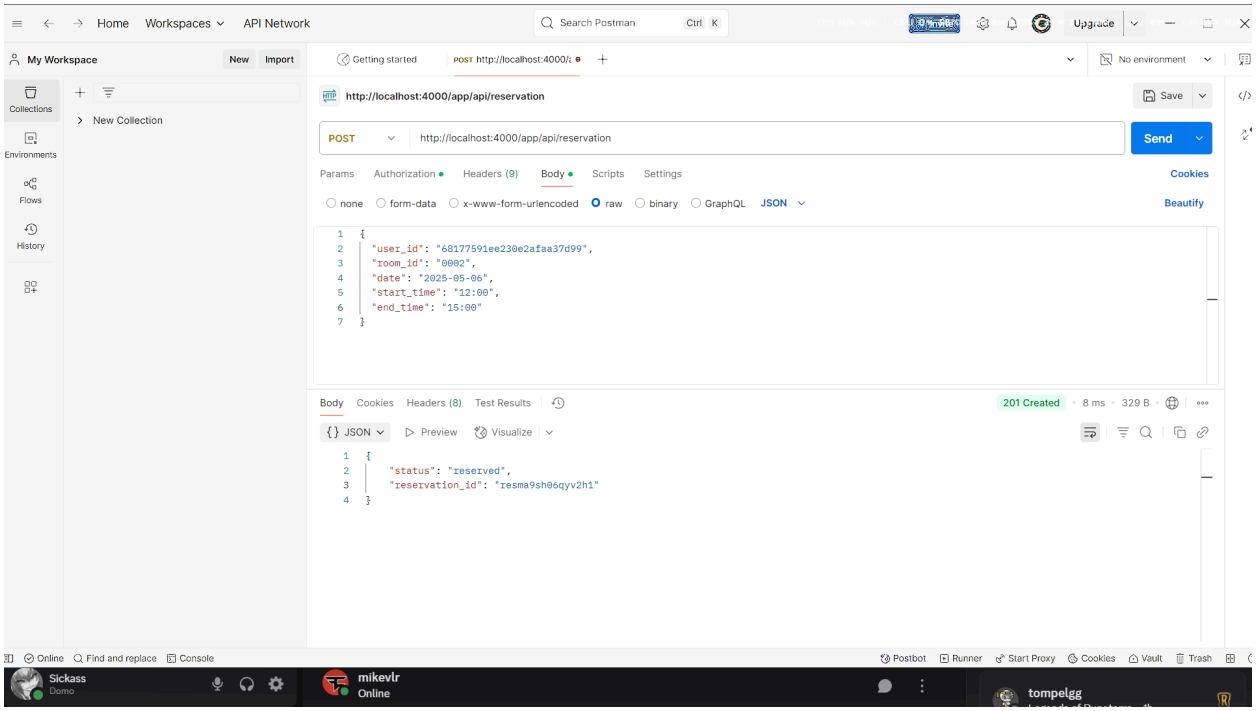




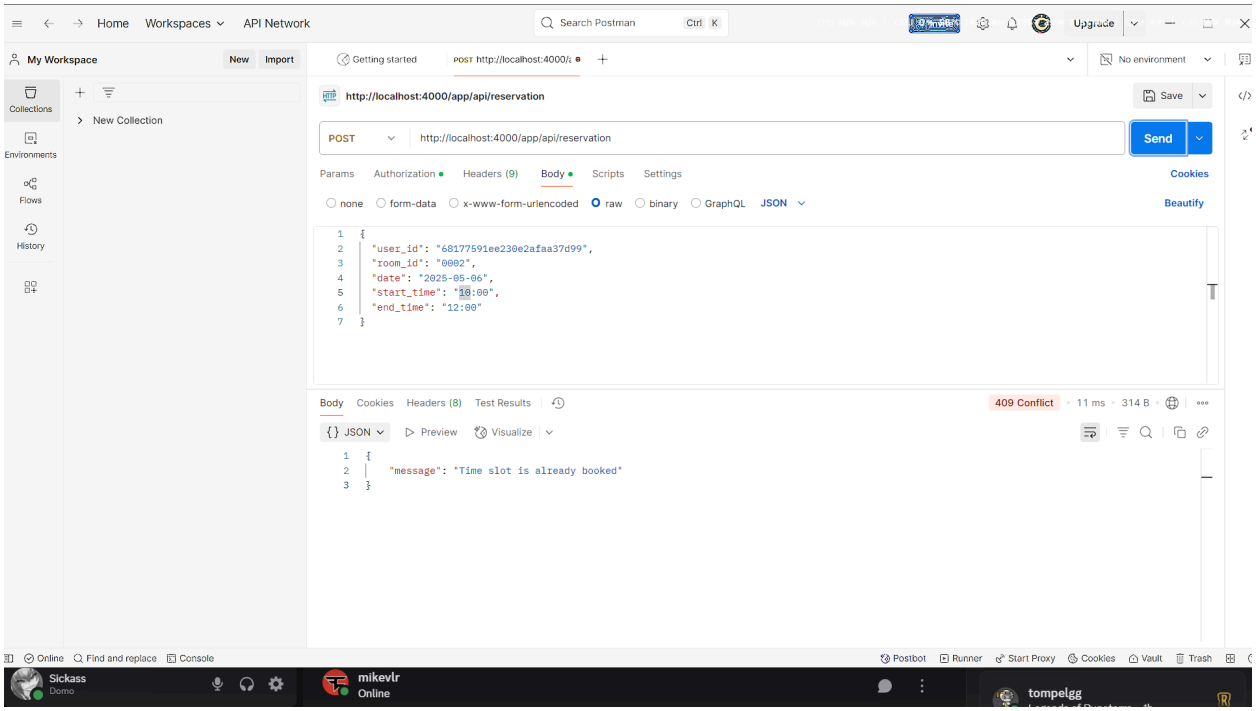
Testing menghapus ruangan dari daftar ruang yang berada di Sistem atau Database melalui Room_ID menggunakan token otoritas “Admin”



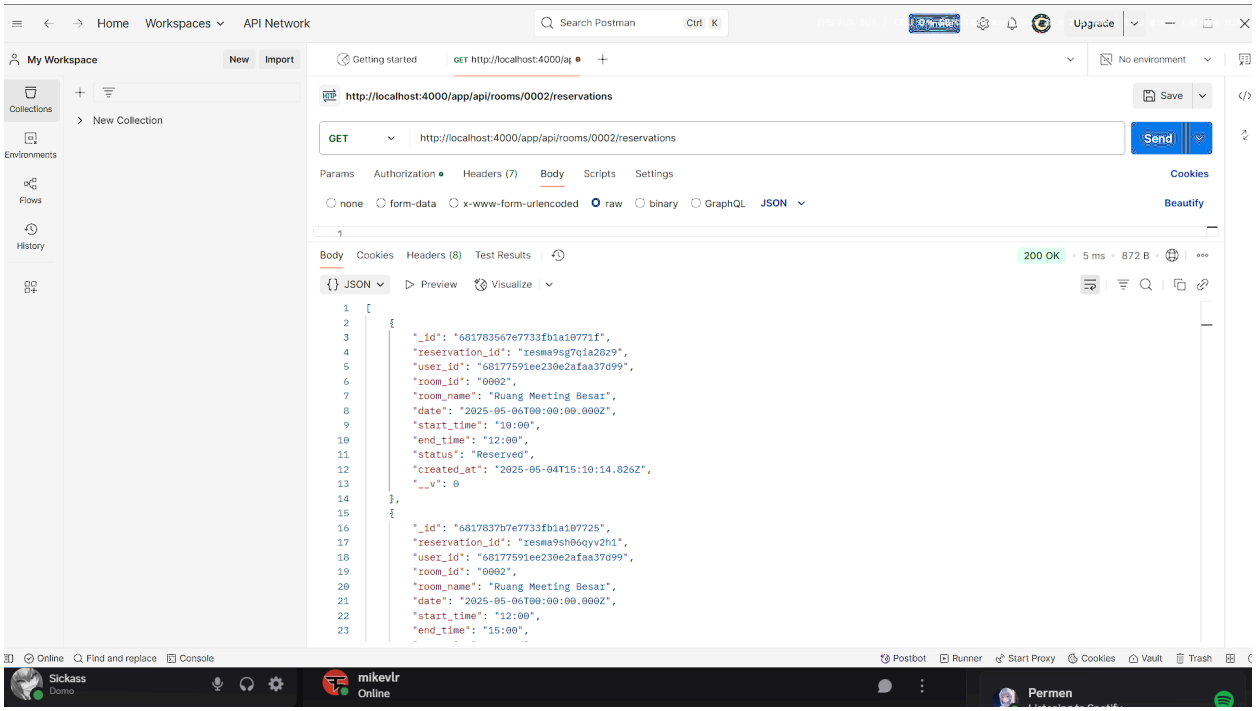
Testing melakukan reservasi ruangan dengan User_ID (Ruangan berubah statusnya menjadi reserved)



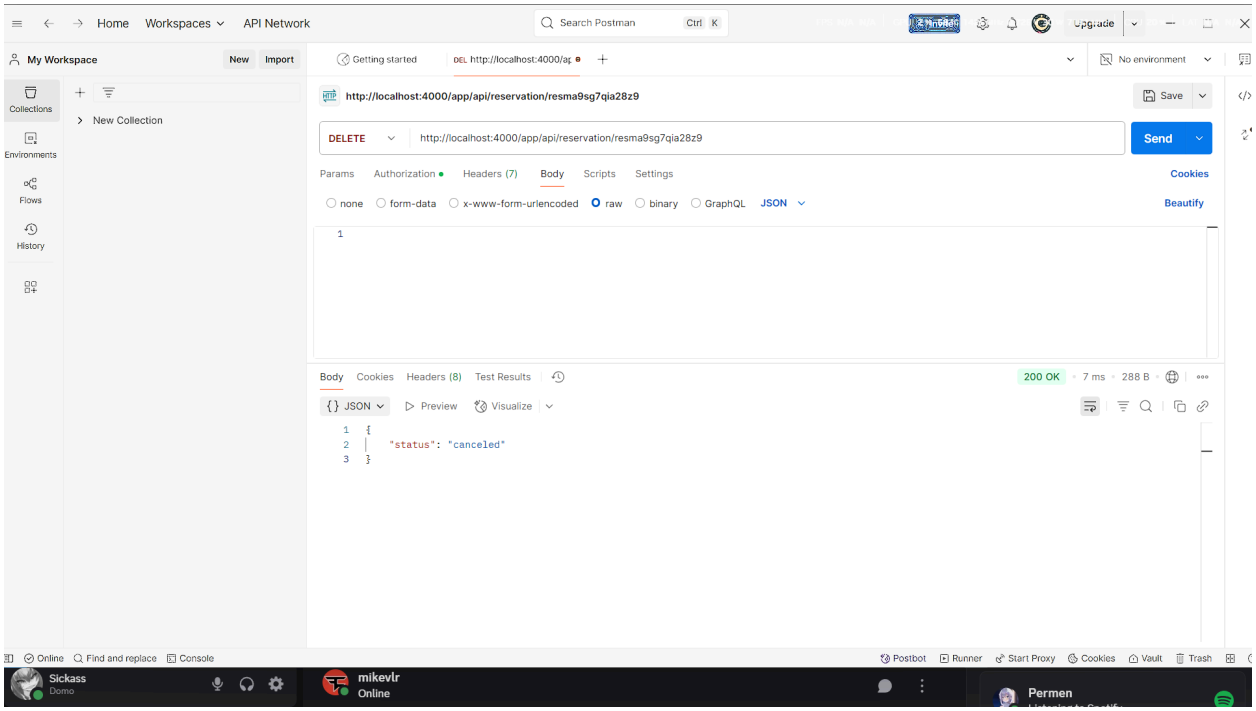
Testing melakukan reservasi apabila ruangan tersebut sudah terbooking



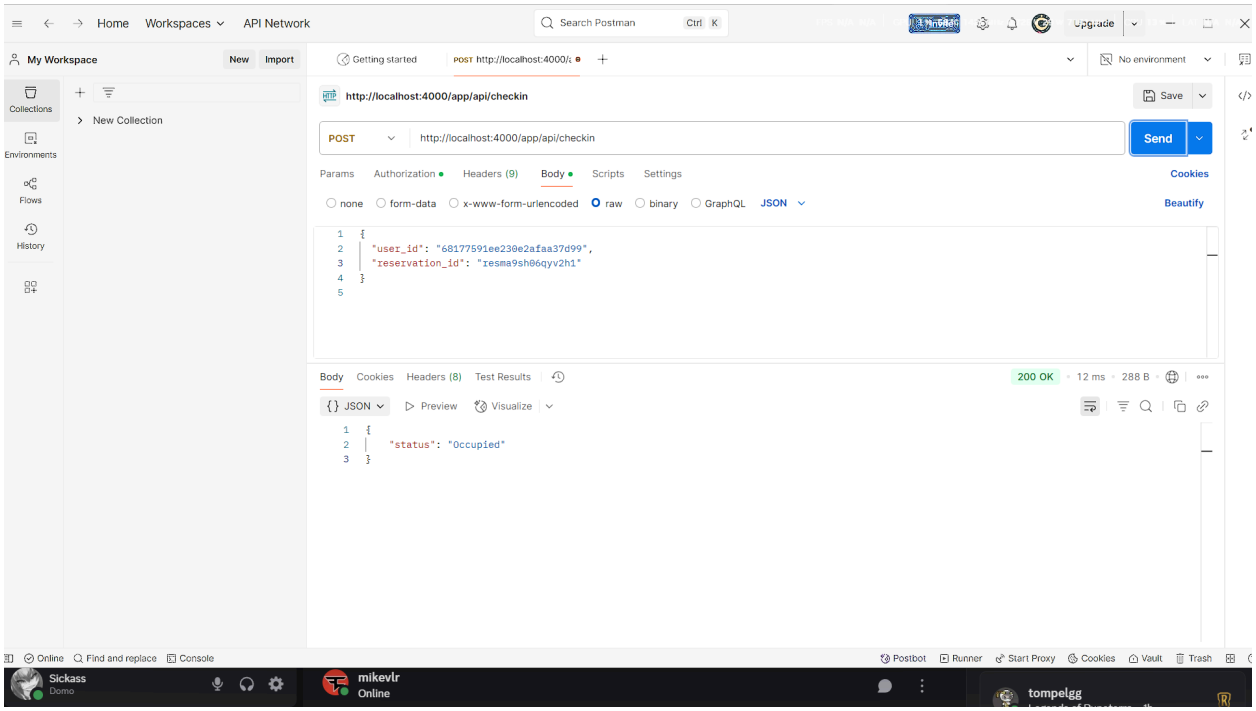
Testing untuk melihat jadwal reservasi dari suatu ruangan melalui Room_ID



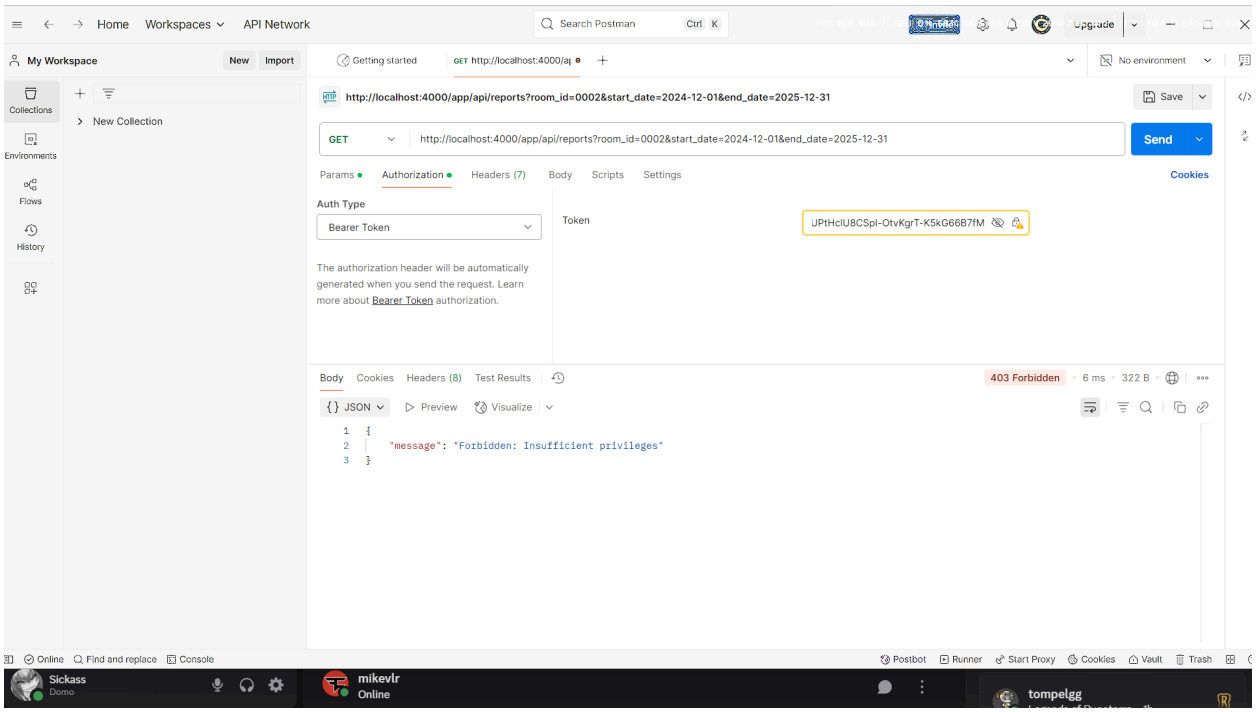
Testing melakukan pembatalan reservasi suatu ruangan melalui Reservation_ID



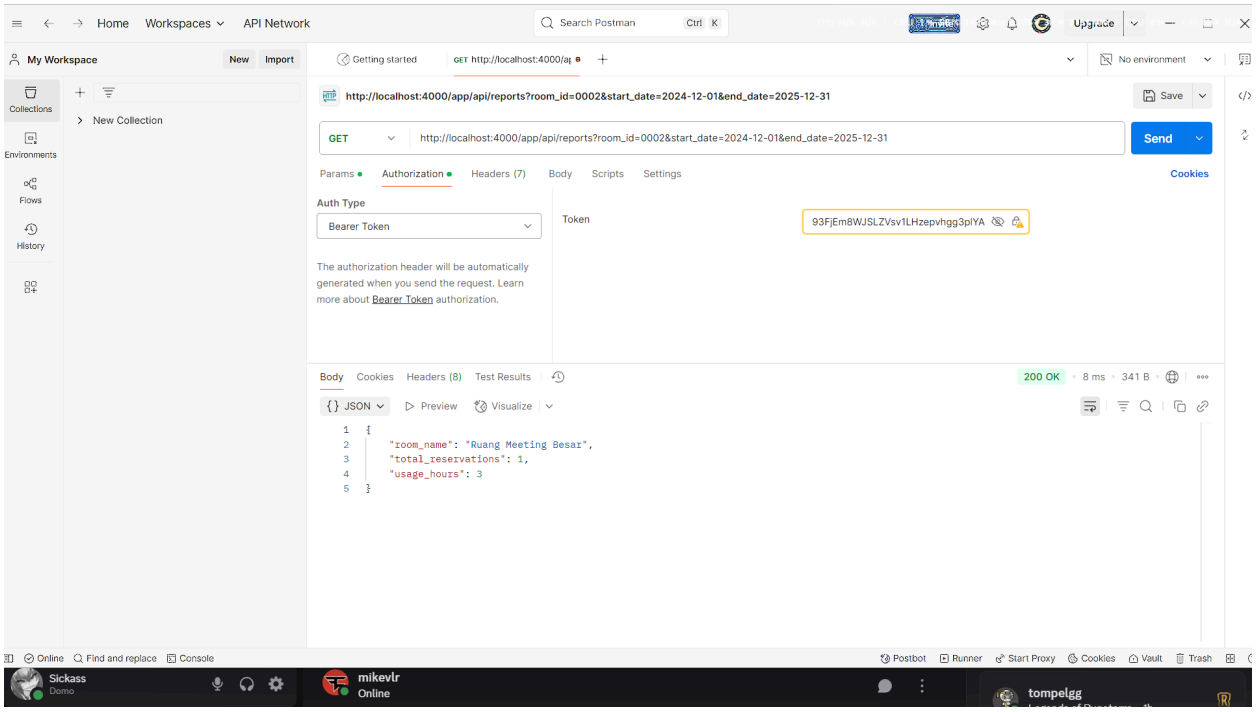
Testing melakukan check-in ruangan menggunakan User_ID dan Reservation_ID (Status Ruangan berubah menjadi Occupied)



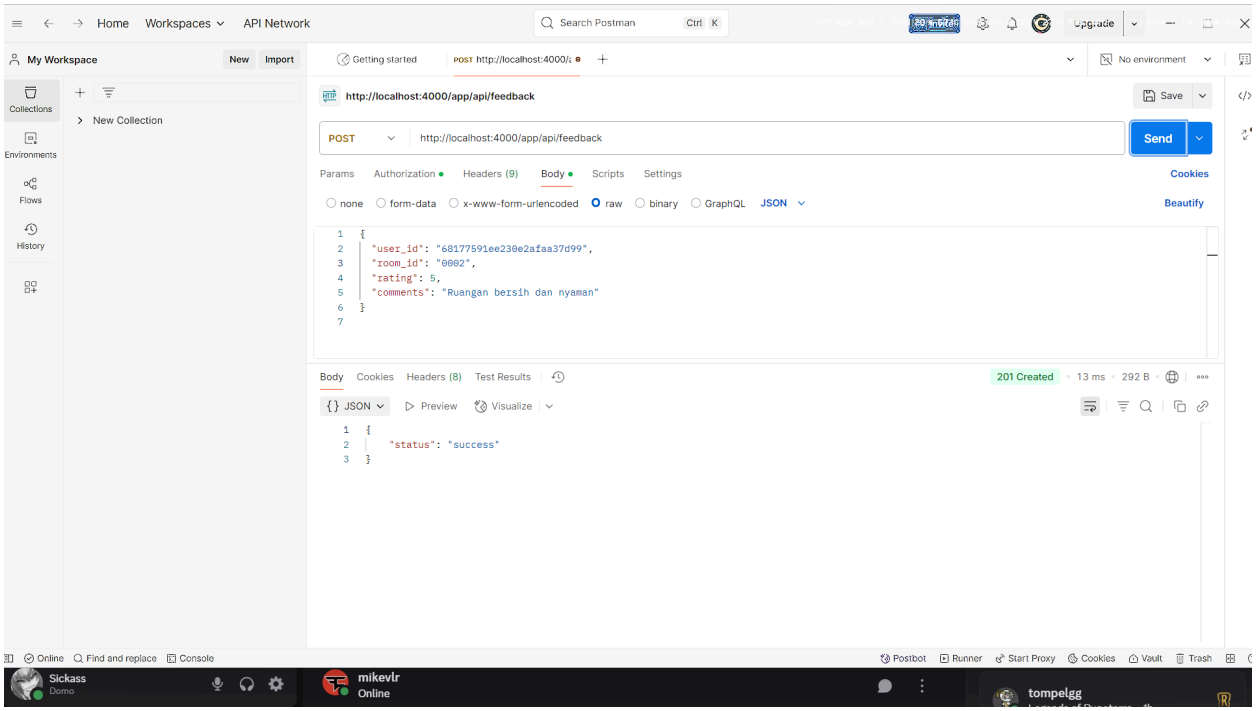
Testing pencarian riwayat penggunaan suatu ruangan menggunakan query Room_ID, start_date, dan end_date (Ditolak, karena memerlukan token otoritas “Admin”)



Testing pencarian riwayat penggunaan suatu ruangan menggunakan query Room_ID, start_date, dan end_date (Mengeluarkan nama ruangan dan jumlah pemakaian dari kurun waktu sesuai query, karena token otoritas “Admin” dikenali oleh Sistem atau Database)



Testing melakukan feedback atau penilaian terhadap suatu ruangan



IV. Desktop Client

A. Penjelasan Aplikasi

Room Reservation Admin Client adalah aplikasi desktop berbasis Java Swing yang berfungsi sebagai panel administrasi untuk sistem reservasi ruangan. Aplikasi ini memungkinkan administrator untuk mengelola ruangan, pengguna, dan melihat laporan penggunaan ruangan.

STRUKTUR APLIKASI

```
├── Main.java                    # Entry point aplikasi
├── RoomReservationAdminClient.java    # Main application class
└── config.properties            # Configuration file
```

B. Desain UI

2.1 Layout Structure

Aplikasi menggunakan CardLayout untuk mengelola multiple panels:

```
MainFrame (1000x600)
├── CardPanel (CardLayout)
    ├── LoginPanel
    ├── DashboardPanel
    ├── RoomsPanel
    ├── UsersPanel
    └── ReportsPanel
```

2.2 Panel Components

A. Login Panel

- **Layout: BoxLayout (Vertical)**
- **Components:**
  - **Title Label ("Admin Login")**
  - **Form Panel (GridBagLayout)**
    - **Email TextField**
    - **Password Field**
    - **Login Button**
    - **Status Label**
- **Features: Input validation dan error display**

B. Dashboard Panel

- **Layout: BorderLayout**
- **Components:**
    - **Navigation Bar (North)**
    - **Content Panel (Center)**
        - **Welcome Message**
        - **Statistics Grid (2x2)**
            - **Total Rooms**
            - **Total Users**
            - **Active Reservations**
            - **Average Rating**

C. Rooms Management Panel

- **Layout: BorderLayout**
- **Components:**
    - **Navigation Bar (North)**
    - **Header Panel dengan Add Button (North)**
    - **JTable dengan custom cell renderers (Center)**
- **Table Columns: Room ID, Name, Capacity, Facilities, Status, Actions**

D. Users Management Panel

- **Layout: BorderLayout**
- **Components:**
    - **Navigation Bar (North)**
    - **Header Panel (North)**
    - **JTable dengan custom cell renderers (Center)**
- **Table Columns: User ID, Email, Name, Department, Role, Actions**

E. Reports Panel

C. **Layout: BorderLayout**
D. **Components:**
    A. **Navigation Bar (North)**
    B. **Form Panel untuk filter (North)**
    C. **Results Panel (Center)**

E. **Penjelasan Source Code**

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Check if configuration exists
        try {
            if (!Files.exists(Paths.get(first:"config.properties"))) {
                // Create default config file
                try (FileWriter writer = new FileWriter(fileName:"config.properties")) {
                    writer.write(str:"# Room Reservation Admin Client Configuration\n");
                    writer.write(str:"api.base.url=http://localhost:5000/app/api\n");
                    writer.write(str:"# Default admin credentials for testing\n");
                }
                JOptionPane.showMessageDialog(parentComponent:null,
                    message:"Created default configuration file.\nPlease edit config.properties if needed.",
                    title:"First Run Setup",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(parentComponent:null,
                "Error creating configuration file: " + e.getMessage(),
                title:"Configuration Error",
                JOptionPane.ERROR_MESSAGE);
        }

        // Start the application
        new RoomReservationAdminClient();
    }
}
```

Aplikasi ini dimulai dari kelas Main.java, yang bertugas melakukan inisialisasi awal sebelum menjalankan antarmuka pengguna. Program akan memeriksa apakah file konfigurasi config.properties tersedia di direktori proyek. Jika tidak ditemukan, sistem secara otomatis akan membuat file konfigurasi default berisi URL dasar API backend. Setelah file konfigurasi dipastikan tersedia, objek utama RoomReservationAdminClient akan dijalankan untuk memuat seluruh tampilan dan fungsionalitas.

```java
// Add action listener to login button
loginButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String email = emailField.getText();
        String password = new String(passwordField.getPassword());

        if (email.isEmpty() || password.isEmpty()) {
            statusLabel.setText(text:"Please enter email and password");
            return;
        }

        try {
            // Create login request JSON
            JSONObject loginData = new JSONObject();
            loginData.put(key:"email", email);
            loginData.put(key:"password", password);

            // Send login request
            JSONObject response = sendPostRequest(endpoint:"/auth/login", loginData.toJSONString(), token:null);

            if (response != null) {
                String token = (String) response.get(key:"token");
                userRole = (String) response.get(key:"role");

                if (token != null && (userRole.equals(anObject:"Admin") || userRole.equals(anObject:"IT"))) {
                    jwtToken = token;
                    statusLabel.setText(text:"");
                    loadDashboardData();
                    cardLayout.show(cardPanel, name:"dashboard");
```

Fitur otentikasi berbasis JWT (JSON Web Token) diimplementasikan di halaman login. Pengguna diminta memasukkan email dan password yang kemudian dikirim sebagai data JSON melalui permintaan HTTP POST ke endpoint /auth/login. Jika respon server

mengandung token dan peran pengguna adalah Admin atau IT, token disimpan untuk digunakan dalam permintaan berikutnya, dan aplikasi akan memuat tampilan dashboard. Jika pengguna tidak memiliki hak akses administratif, sistem akan menampilkan pesan error yang sesuai. Hal ini mencerminkan adanya kontrol akses berbasis peran.

```java
    private JSONObject sendPostRequest(String endpoint, String jsonData, String token) throws Exception {
    URL url = new URL(apiBaseUrl + endpoint);
    System.out.println("Making POST request to: " + url.toString());

    HttpURLConnection connection = (HttpURLConnection) url.openConnection();

    connection.setRequestMethod(method:"POST");
    connection.setRequestProperty(key:"Content-Type", value:"application/json");
    connection.setRequestProperty(key:"Accept", value:"application/json");
    connection.setDoOutput(dooutput:true);
    connection.setConnectTimeout(timeout:10000); // 10 seconds timeout
    connection.setReadTimeout(timeout:10000); // 10 seconds timeout

    if (token != null && !token.isEmpty()) {
        connection.setRequestProperty(key:"Authorization", "Bearer " + token);
        System.out.println(x:"Authorization header added");
    } else {
        System.out.println(x:"WARNING: No authorization token provided");
    }

    // Send the JSON data
    try (DataOutputStream wr = new DataOutputStream(connection.getOutputStream())) {
        wr.writeBytes(jsonData);
        wr.flush();
        System.out.println("Request data sent: " + jsonData);
    }

    int responseCode = connection.getResponseCode();
    System.out.println("Response code: " + responseCode);
```

Untuk komunikasi dengan backend API, aplikasi menggunakan metode HTTP standar yang dikemas ke dalam fungsi seperti sendPutRequest(), sendPostRequest(), sendGetRequest(). Fungsi-fungsi ini mengatur header permintaan, seperti tipe konten dan token otorisasi, serta memastikan penanganan error berdasarkan kode respon HTTP.

```java
    private void loadRoomsData() {
        try {
            JSONArray rooms = sendGetRequest(endpoint:"/rooms");

            if (rooms != null) {
                // Clear the table
                roomsTableModel.setRowCount(rowCount:0);

                for (Object obj : rooms) {
                    JSONObject room = (JSONObject) obj;
                    String roomId = (String) room.get(key:"room_id");
                    String name = (String) room.get(key:"name");
                    long capacity = (long) room.get(key:"capacity");
                    JSONArray facilities = (JSONArray) room.get(key:"facilities");
                    String facilitiesStr = String.join(delimiter:", ", facilities);
                    String status = (String) room.get(key:"status");

                    // Add row to table
                    roomsTableModel.addRow(new Object[]{roomId, name, capacity, facilitiesStr, status, "Edit/Delete"});
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(mainFrame, "Error loading rooms data: " + e.getMessage(), title:"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Sistem memuat data secara dinamis ke dalam tampilan, terutama pada tabel ruangan dan daftar reservasi. Contohnya, metode loadRoomsData() mengambil data dari endpoint /rooms, mengurai JSON yang diterima, dan mengisinya ke dalam TableModel

untuk ditampilkan di UI. Proses ini memungkinkan pengguna untuk melihat informasi terkini tanpa harus memulai ulang aplikasi.

```java
775     private void showAddRoomDialog() {
776         JDialog dialog = new JDialog(mainFrame, title:"Add New Room", modal:true);
777         dialog.setSize(width:400, height:300);
778         dialog.setLocationRelativeTo(mainFrame);
779
780         JPanel panel = new JPanel(new GridBagLayout());
781         GridBagConstraints gbc = new GridBagConstraints();
782         gbc.fill = GridBagConstraints.HORIZONTAL;
783         gbc.insets = new Insets(top:5, left:5, bottom:5, right:5);
784
785         JLabel nameLabel = new JLabel(text:"Room Name:");
786         JTextField nameField = new JTextField(columns:20);
787
788         JLabel capacityLabel = new JLabel(text:"Capacity:");
789         JTextField capacityField = new JTextField(columns:20);
790
791         JLabel facilitiesLabel = new JLabel(text:"Facilities (comma separated):");
792         JTextField facilitiesField = new JTextField(columns:20);
793
794         JButton saveButton = new JButton(text:"Save");
795         JButton cancelButton = new JButton(text:"Cancel");
796
797         JLabel statusMessageLabel = new JLabel(text:"");
798         statusMessageLabel.setForeground(Color.RED);
799
800         gbc.gridx = 0;
801         gbc.gridy = 0;
802         panel.add(nameLabel, gbc);
803
804         gbc.gridx = 1;
805         gbc.gridy = 0;
806         panel.add(nameField, gbc);
```

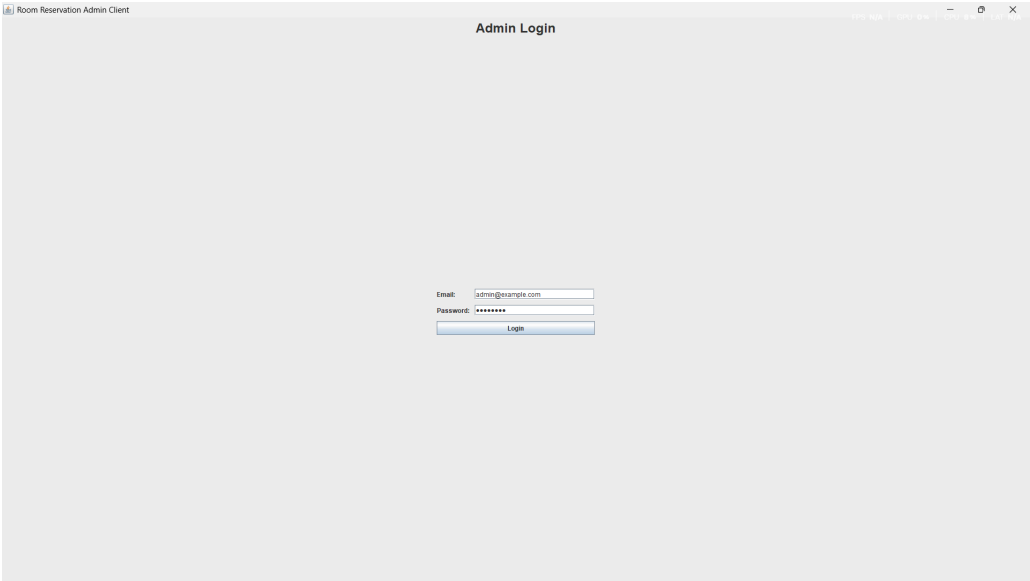
Fitur CRUD (Create, Read, Update, Delete) untuk manajemen ruangan juga disediakan, dengan antarmuka berupa JDialog yang interaktif. Saat menambahkan ruangan baru, sistem memvalidasi input pengguna (seperti nama ruangan dan kapasitas), kemudian mengubah data menjadi format JSON untuk dikirim ke server. Fasilitas ruangan dikonversi dari string menjadi array JSON, dan data dikirim melalui HTTP POST. Setelah ruangan berhasil ditambahkan, sistem otomatis menyegarkan tabel untuk menampilkan data terbaru.

```java
596     private void generateReport(String roomId, String startDate, String endDate, JLabel roomNameLabel, JLabel totalReservationsLabel, JLabel usageHoursLabel) {
597     try {
598         String url = "/reports?room_id=" + roomId + "&start_date=" + startDate + "&end_date=" + endDate;
599
600         // Gunakan method yang baru untuk mendapatkan JSONObject
601         JSONObject report = sendGetRequestForObject(url);
602
603         if (report != null) {
604             // Periksa apakah field yang diperlukan ada
605             String roomName = report.get(key:"room_name") != null ? (String) report.get(key:"room_name") : "Unknown";
606
607             // Handle total_reservations (bisa Long atau Integer)
608             Object totalReservationsObj = report.get(key:"total_reservations");
609             long totalReservations = 0;
610             if (totalReservationsObj instanceof Long) {
611                 totalReservations = (Long) totalReservationsObj;
612             } else if (totalReservationsObj instanceof Integer) {
613                 totalReservations = ((Integer) totalReservationsObj).longValue();
614             }
615
616             // Handle usage_hours (bisa Double, Integer, atau Long)
617             Object usageHoursObj = report.get(key:"usage_hours");
618             double usageHours = 0.0;
619             if (usageHoursObj instanceof Double) {
620                 usageHours = (Double) usageHoursObj;
621             } else if (usageHoursObj instanceof Integer) {
622                 usageHours = ((Integer) usageHoursObj).doubleValue();
623             } else if (usageHoursObj instanceof Long) {
624                 usageHours = ((Long) usageHoursObj).doubleValue();
```

Sistem juga memiliki fitur pembuatan laporan penggunaan ruangan berdasarkan ID ruangan dan rentang waktu tertentu. Fungsi generateReport() melakukan permintaan GET ke endpoint /reports dengan parameter filter seperti ID ruangan, tanggal mulai, dan tanggal akhir. Respon yang diterima diproses dan ditampilkan di antarmuka, termasuk nama ruangan, total reservasi, dan total jam penggunaan. Proses ini

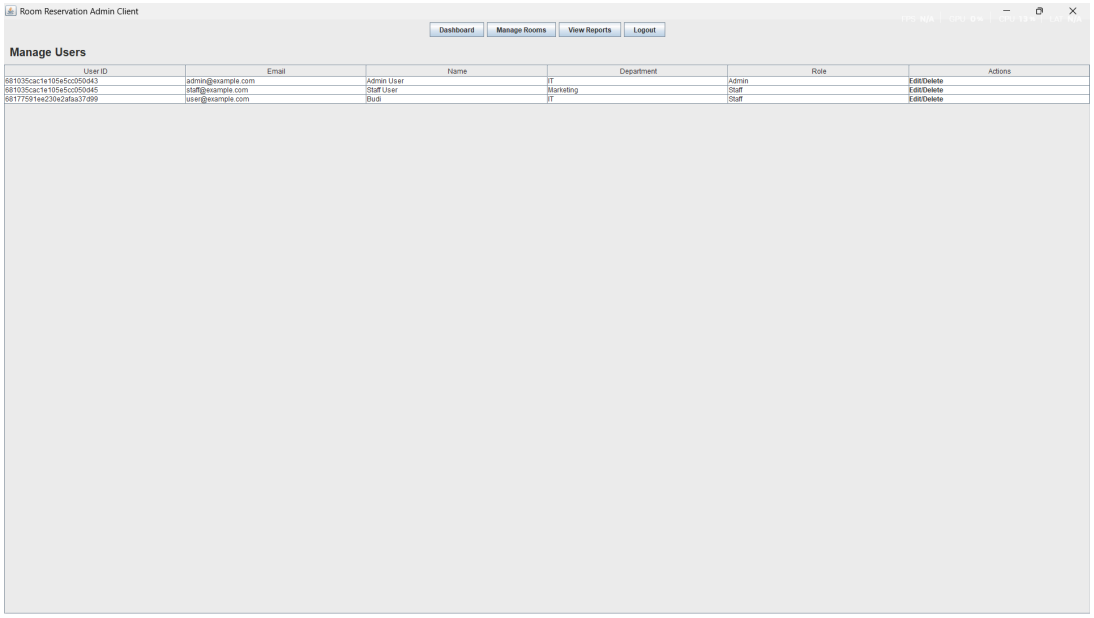
memperhatikan variasi tipe data numerik dari hasil parsing JSON dan memberikan fallback untuk memastikan stabilitas aplikasi.

F. **Hasil Pengujian**

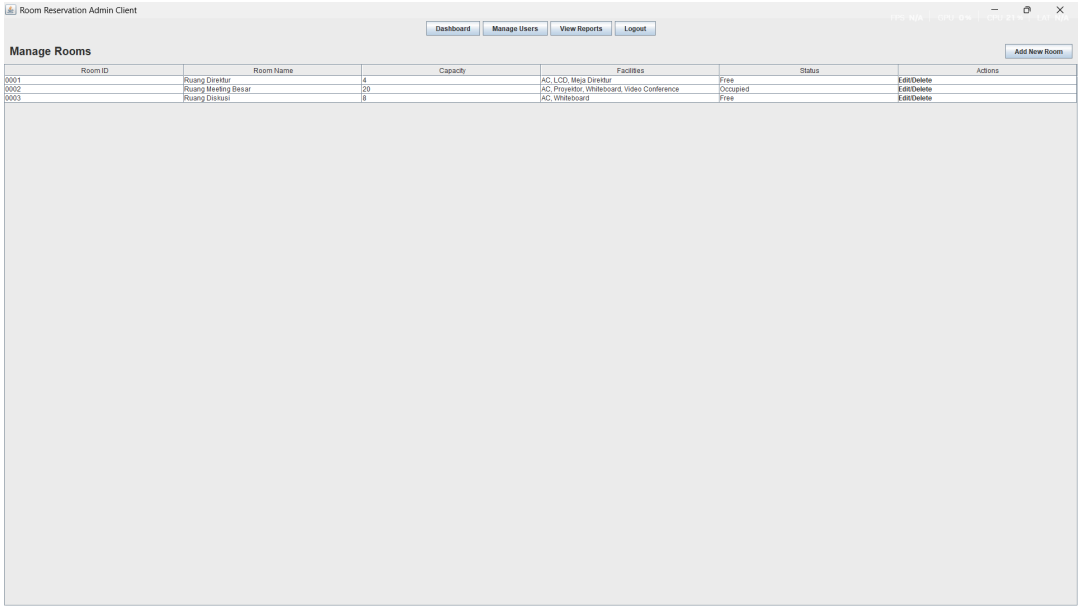


Setelah menjalankan kode java, muncul sebuah ui yang menampilkan login panel, login ini hanya bisa masuk untuk username dan database yang dikenali sebagai akses admin pada aplikasi

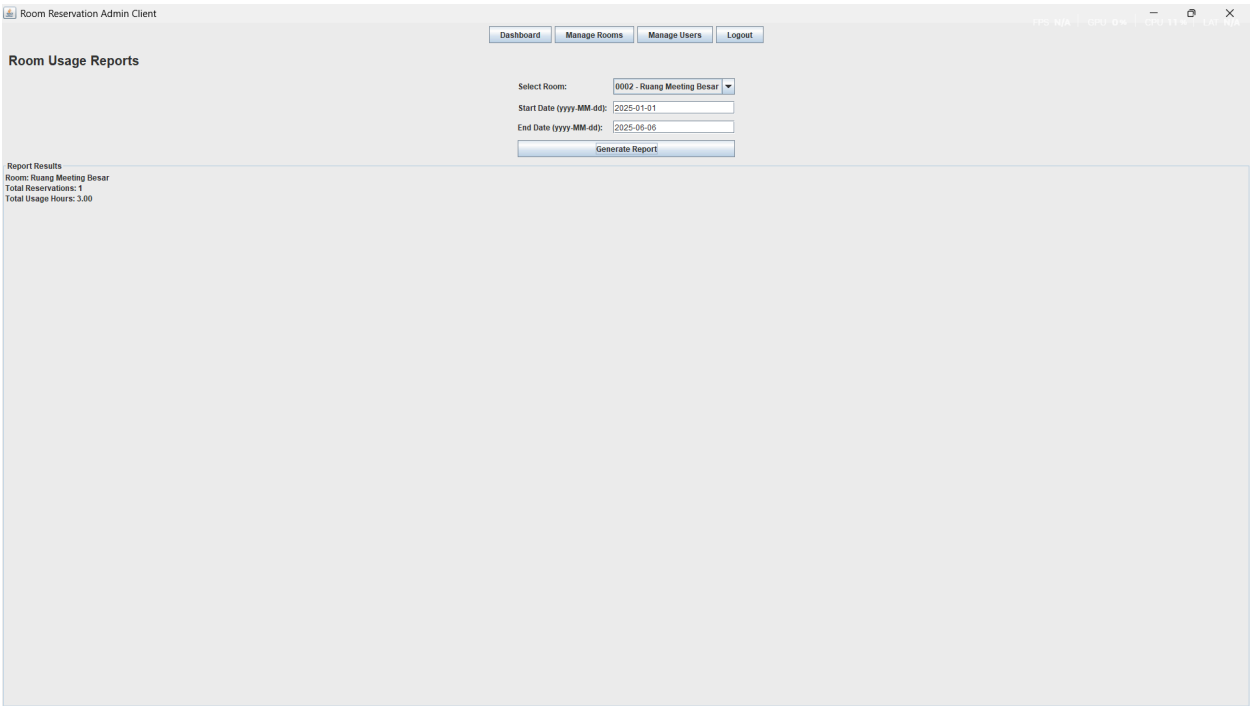
Setelah melakukan login kita diberikan tampilan dashboard yang menampilkan total ruangan dan user yang terdaftar di database server. Dashboard juga menampilkan jumlah reservation yang sedang berlangsung, dan rata - rata rating setiap ruangan yang tersedia



Bagian manage users berisi tabel mengenai user siapa saja yang terdaftar dalam server, dengan spesifikasi seperti email, user id, nama, departement, role. Sebagai admin tentu kita bisa melakukan edit atau delete pada data user diatas.



Manage rooms, berfungsi sebagai tampilan detail mengenai room apa saja yang terdaftar di dalam server, dan terdapat tambahan untuk menambah ruangan baru yang bisa didaftarkan ke dalam server.



V. Mobile Client Implementation

1. Deskripsi Aplikasi dan Teknologi

Room Reservation Android Application merupakan aplikasi mobile native yang dikembangkan untuk memfasilitasi sistem manajemen pemesanan ruangan secara digital. Aplikasi ini dirancang dengan tujuan utama untuk mempermudah pengguna dalam melakukan reservasi ruangan melalui perangkat mobile mereka, mulai dari proses registrasi akun, autentikasi pengguna, hingga pengelolaan reservasi yang komprehensif. Fitur-fitur utama yang disediakan meliputi sistem login dan registrasi yang aman, tampilan daftar ruangan yang tersedia dengan informasi detail, sistem pemesanan yang intuitif dengan pemilihan tanggal dan waktu, manajemen reservasi pribadi yang memungkinkan pengguna melihat dan membatalkan reservasi mereka, fitur check-in digital untuk konfirmasi kehadiran, serta sistem feedback yang memungkinkan pengguna memberikan penilaian terhadap fasilitas yang telah digunakan.

Dari segi teknologi, aplikasi ini dibangun menggunakan Android Native dengan bahasa pemrograman Java sebagai fondasi utama pengembangan. Pemilihan platform Android Native bertujuan untuk memberikan performa optimal dan integrasi yang sempurna dengan sistem operasi Android. Untuk manajemen data, aplikasi menggunakan kombinasi SQLite untuk penyimpanan lokal dan API Backend untuk sinkronisasi data real-time dengan server. User interface dirancang menggunakan Android XML Layout yang memungkinkan desain yang responsif dan konsisten di berbagai ukuran layar. Untuk komunikasi dengan server, aplikasi mengimplementasikan HTTP Client yang menangani semua operasi networking termasuk autentikasi, pengambilan data ruangan, dan proses reservasi. Arsitektur aplikasi mengadopsi pola Model-View-Controller (MVC) yang memisahkan logika bisnis, presentasi data, dan interaksi pengguna secara terstruktur, sehingga memudahkan maintenance dan pengembangan fitur baru di masa mendatang.

2. Struktur Projek

```
app/src/main/java/com/example/Room_Reservation/
├── activities/
│   ├── MainActivity.java
│   ├── RegisterActivity.java
│   ├── DashboardActivity.java
│   ├── RoomsActivity.java
│   ├── MakeReservationActivity.java
│   ├── MyReservationsActivity.java
│   ├── ReservationDetailActivity.java
│   ├── CheckInActivity.java
├── models/
│   ├── User.java
│   ├── Room.java
│   ├── Reservation.java
├── adapters/
│   ├── RoomAdapter.java
│   └── ReservationAdapter.java
└── utils/
    ├── ApiConfig.java
    ├── NetworkUtils.java
    ├── SharedPrefManager.java
    └── DateTimeUtils.java
```

Layout Files (app/src/main/res/layout/) Activity Layouts

- activity_main.xml - Login screen layout
- activity_register.xml - Registration screen layout
- activity_dashboard.xml - Dashboard layout
- activity_rooms.xml - Rooms list layout
- activity_make_reservation.xml - Make reservation layout
- activity_my_reservations.xml - User reservations layout
- activity_reservation_detail.xml - Reservation detail layout
- activity_check_in.xml - Check-in layout
- item_room.xml - Room list item layout
- Item_reservation.xml - Reservation list item layout

3. Deskripsi Class dan Layout

3.1 Deskripsi Layout

- **Login Screen (activity_main)**
  Berfungsi sebagai gerbang utama aplikasi dengan desain yang clean dan user-friendly. Layar ini menampilkan logo aplikasi di bagian atas untuk branding, diikuti dengan dua field input utama yaitu EditText untuk username atau email dan EditText untuk password yang dilengkapi dengan fitur hide/show password. Bagian bawah layar memuat tombol Login yang prominent dengan warna kontras tinggi untuk menarik perhatian pengguna, serta TextView yang berfungsi sebagai link menuju halaman registrasi bagi pengguna baru.

- **Registration Screen (activity_register)**
  Proses onboarding pengguna baru dengan form yang komprehensif namun tidak overwhelming. Layar ini menampilkan serangkaian EditText yang tersusun vertikal untuk input data pribadi meliputi nama lengkap, alamat email, nomor telepon, password, dan konfirmasi password. Setiap field dilengkapi dengan validasi real-time dan icon yang relevan untuk meningkatkan user experience. Tombol Register ditempatkan di bagian bawah dengan desain yang konsisten dengan tombol di layar lainnya, dan terdapat link untuk kembali ke halaman login jika pengguna sudah memiliki akun.

- **Dashboard Screen (activity_dashboard)**
  Pusat navigasi utama setelah pengguna berhasil login, menampilkan interface yang intuitif dengan welcome message yang dipersonalisasi menggunakan nama pengguna. Layar ini menggunakan card-based layout yang menampilkan berbagai fitur utama aplikasi dalam bentuk card atau button yang dapat diklik,

termasuk "View Rooms" untuk melihat daftar ruangan, "My Reservations" untuk mengelola reservasi pribadi, dan "Profile" untuk pengaturan akun. Navigation drawer atau bottom navigation bar menyediakan akses cepat ke fitur-fitur sekunder dan pengaturan aplikasi, dengan indikator visual yang jelas menunjukkan menu yang sedang aktif.

- **Rooms List Screen (activity_room + item_room)**
  Menampilkan daftar ruangan yang tersedia dalam format RecyclerView yang scrollable dan responsif. Bagian atas layar dilengkapi dengan SearchView yang memungkinkan pengguna mencari ruangan berdasarkan nama atau karakteristik tertentu, serta filter options yang dapat diakses melalui icon filter untuk menyaring ruangan berdasarkan kapasitas, tipe, atau fasilitas. Setiap item ruangan dalam list menampilkan informasi penting seperti nama ruangan, tipe, kapasitas, dan harga dalam layout yang terstruktur dan mudah dipahami. FloatingActionButton di pojok kanan bawah menyediakan fungsi refresh untuk memperbarui data ruangan secara manual.

- **Make Reservation Screen (activity_make_reservation)**
  Proses pembuatan reservasi dengan layout yang step-by-step dan user-friendly. Bagian atas layar menampilkan detail ruangan yang dipilih termasuk foto, nama, fasilitas, dan harga untuk memberikan konfirmasi visual kepada pengguna. Area utama layar berisi komponen interaktif untuk pemilihan tanggal menggunakan DatePicker dengan kalender yang intuitif, dan TimePicker terpisah untuk waktu mulai dan selesai yang memungkinkan pengguna memilih durasi reservasi dengan tepat. EditText untuk catatan atau keperluan khusus ditempatkan di bagian tengah, diikuti dengan ringkasan total biaya yang dihitung secara otomatis. Tombol "Make Reservation" di bagian bawah layar dilengkapi dengan loading indicator dan confirmation dialog untuk memastikan akurasi data sebelum proses reservasi dilakukan.

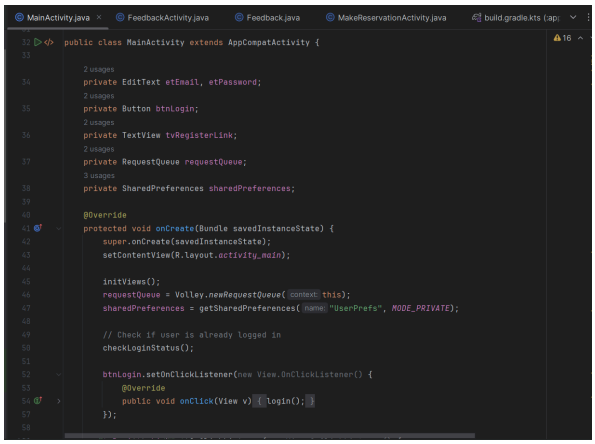
- **My Reservations Screen (activity_my_reservations + item_reservation)**
  Daftar reservasi pengguna dalam format RecyclerView yang terorganisir berdasarkan status dan tanggal. Setiap item reservasi menampilkan informasi penting seperti nama ruangan, tanggal dan waktu, status reservasi (upcoming, ongoing, completed, cancelled), dan action buttons yang sesuai dengan status tersebut. Tab atau segmented control di bagian atas memungkinkan pengguna memfilter reservasi berdasarkan status untuk memudahkan navigasi dan manajemen.

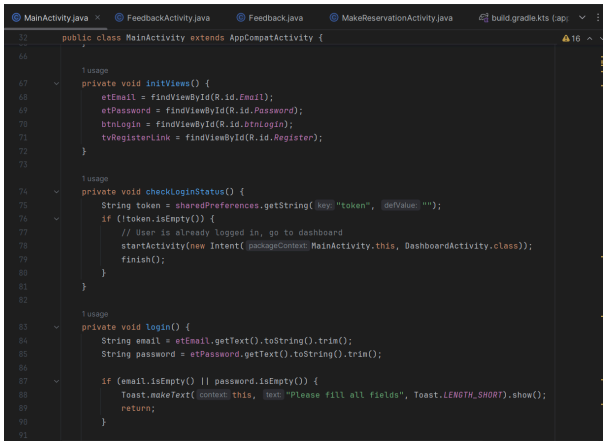
- **Reservation Detail Screen (activity_reservation_detail)**
  Memberikan view detail setiap reservasi dengan layout yang detail dan informatif. Layar ini menampilkan semua informasi reservasi termasuk detail ruangan dengan foto, informasi waktu dan durasi, total biaya, dan catatan khusus. Action buttons seperti "Cancel Reservation", "Modify", atau "Check In" ditempatkan strategis berdasarkan status reservasi dan waktu yang tersisa, dengan konfirmasi dialog untuk setiap action yang memiliki konsekuensi penting.

- **Check-in Screen (CheckInActivity)**
  Proses check-in yang cepat dan efisien dengan interface yang minimal namun informatif. Layar ini menampilkan informasi reservasi yang akan di-check-in, termasuk detail ruangan dan waktu reservasi, diikuti dengan tombol check-in yang prominent dan mudah diakses. Fitur tambahan seperti QR code scanner atau location verification dapat diintegrasikan untuk meningkatkan keamanan dan akurasi proses check-in.

3.2 Deskripsi Class

- **MainActivity**





MainActivity merupakan activity utama yang berfungsi sebagai entry point aplikasi dan menangani proses autentikasi pengguna. Class ini meng-extends AppCompatActivity untuk mendapatkan fitur-fitur modern Android yang kompatibel dengan versi lama. Variabel instance yang dideklarasikan meliputi Username dan Password sebagai EditText untuk input kredensial, btnLogin sebagai tombol untuk memulai proses login, tvRegister sebagai TextView yang berfungsi sebagai link menuju halaman registrasi, dan sharedPrefManager untuk mengelola data session pengguna.

Method onCreate() mengatur lifecycle activity dengan memanggil setContentView() untuk menghubungkan layout XML, kemudian menjalankan initViews() untuk menginisialisasi komponen UI, dan checkLoginStatus() untuk memeriksa apakah pengguna sudah login sebelumnya. Method login() menangani logika autentikasi dengan mengambil input dari EditText, kemudian memanggil NetworkUtils.login() dengan callback interface NetworkCallback yang menangani response sukses dengan menyimpan data pengguna ke SharedPreferences dan berpindah ke activity_dashboard, serta menampilkan error message jika login gagal.

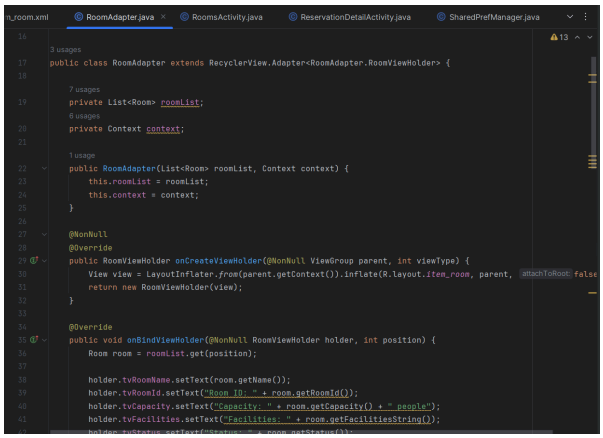
- **Room**

```java
public class Room {
    3 usages
    private String roomId;
    3 usages
    private String name;
    3 usages
    private int capacity;
    6 usages
    private List<String> facilities;
    3 usages
    private String status;

    public Room() {
    }

    public Room(String roomId, String name, int capacity, List<String> facilities, String status) {
        this.roomId = roomId;
        this.name = name;
        this.capacity = capacity;
        this.facilities = facilities;
        this.status = status;
    }

    // Getters and Setters
    7 usages
    public String getRoomId() { return roomId; }

    2 usages
    public void setRoomId(String roomId) { this.roomId = roomId; }
```

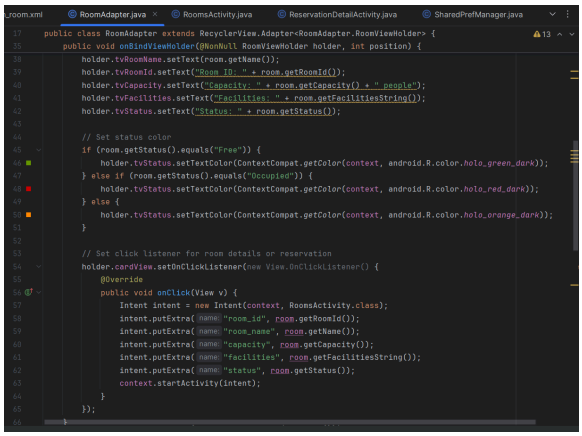
Room.java merupakan model class yang merepresentasikan entitas ruangan dalam aplikasi dengan menerapkan konsep encapsulation dari Object-Oriented Programming. Class ini mendeklarasikan variabel private untuk menyimpan data ruangan meliputi id sebagai identifier unik, name untuk nama ruangan, capacity untuk kapasitas maksimal, facilities untuk fasilitas barang yang tersedia, status untuk status ketersediaan.

Constructor default tanpa parameter disediakan untuk keperluan serialization dan deserialization, sementara constructor dengan parameter lengkap memungkinkan inisialisasi object dengan data lengkap sekaligus. Method getter dan setter mengikuti convention Java Bean yang memungkinkan akses controlled terhadap variabel private, dengan setter yang dapat dilengkapi validasi data dan getter yang memberikan akses read-only ke variabel internal. Implementasi List<String> facilities memungkinkan penambahan dan penghapusan fasilitas secara dinamis, sedangkan penggunaan boolean status memudahkan filtering ruangan berdasarkan availability status.

- **RoomAdapter**



```java
3 usages
public class RoomAdapter extends RecyclerView.Adapter<RoomAdapter.RoomViewHolder> {

    7 usages
    private List<Room> roomList;
    6 usages
    private Context context;

    1 usage
    public RoomAdapter(List<Room> roomList, Context context) {
        this.roomList = roomList;
        this.context = context;
    }

    @NonNull
    @Override
    public RoomViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_room, parent, attachToRoot: false
        return new RoomViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull RoomViewHolder holder, int position) {
        Room room = roomList.get(position);

        holder.tvRoomName.setText(room.getName());
        holder.tvRoomId.setText("Room ID: " + room.getRoomId());
        holder.tvCapacity.setText("Capacity: " + room.getCapacity() + " people");
        holder.tvFacilities.setText("Facilities: " + room.getFacilitiesString());
        holder.tvStatus.setText("Status: " + room.getStatus());
```



```java
public class RoomAdapter extends RecyclerView.Adapter<RoomAdapter.RoomViewHolder> {
    public void onBindViewHolder(@NonNull RoomViewHolder holder, int position) {
        holder.tvRoomName.setText(room.getName());
        holder.tvRoomId.setText("Room ID: " + room.getRoomId());
        holder.tvCapacity.setText("Capacity: " + room.getCapacity() + " people");
        holder.tvFacilities.setText("Facilities: " + room.getFacilitiesString());
        holder.tvStatus.setText("Status: " + room.getStatus());

        // Set status color
        if (room.getStatus().equals("Free")) {
            holder.tvStatus.setTextColor(ContextCompat.getColor(context, android.R.color.holo_green_dark));
        } else if (room.getStatus().equals("Occupied")) {
            holder.tvStatus.setTextColor(ContextCompat.getColor(context, android.R.color.holo_red_dark));
        } else {
            holder.tvStatus.setTextColor(ContextCompat.getColor(context, android.R.color.holo_orange_dark));
        }

        // Set click listener for room details or reservation
        holder.cardView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(context, RoomsActivity.class);
                intent.putExtra( name: "room_id", room.getRoomId());
                intent.putExtra( name: "room_name", room.getName());
                intent.putExtra( name: "capacity", room.getCapacity());
                intent.putExtra( name: "facilities", room.getFacilitiesString());
                intent.putExtra( name: "status", room.getStatus());
                context.startActivity(intent);
            }
        });
```

RoomAdapter.java merupakan implementasi RecyclerView Adapter yang dirancang khusus untuk menampilkan daftar ruangan dalam aplikasi Room Reservation dengan menggunakan CardView sebagai container utama setiap item. Class ini meng-extends RecyclerView.Adapter dengan generic type RoomViewHolder dan mengimplementasikan pattern ViewHolder untuk optimasi performa rendering list yang panjang. Adapter ini memiliki dua variabel instance utama yaitu roomList bertipe List<Room> yang berfungsi sebagai data source containing room objects, dan context bertipe Context yang diperlukan untuk operasi UI seperti inflating layout dan color resource access.

Constructor RoomAdapter menerima dua parameter essential yaitu List<Room> dan Context, yang kemudian diassign ke instance variables untuk digunakan dalam lifecycle adapter. Method onCreateViewHolder() menjalankan proses inflating layout item_room.xml menggunakan LayoutInflater dan mengembalikan instance RoomViewHolder yang baru dibuat, sedangkan onBindViewHolder() bertanggung jawab untuk melakukan data binding antara Room object dengan view components di setiap posisi dalam list.

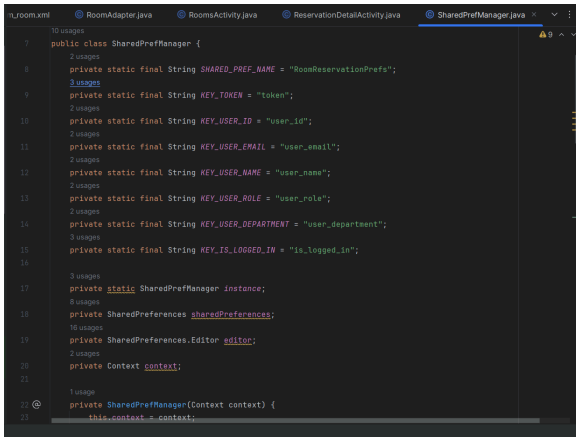
Dalam proses data binding, adapter menampilkan informasi comprehensive room details termasuk nama ruangan, room ID dengan prefix "Room ID: ", kapasitas dengan format "Capacity: X people", fasilitas yang tersedia melalui getFacilitiesString(), dan status ruangan. Implementasi visual feedback yang menarik diterapkan melalui conditional coloring pada status text, dimana status "Free" ditampilkan dengan warna hijau (holo_green_dark), "Occupied" dengan warna merah (holo_red_dark), dan status lainnya dengan warna orange (holo_orange_dark) menggunakan ContextCompat.getColor() untuk compatibility across different Android versions.

Fitur interaktivity diimplementasikan melalui OnClickListener pada CardView yang memungkinkan user untuk melakukan tap pada item ruangan. Ketika user melakukan click, adapter membuat Intent baru menuju RoomsActivity dengan membawa data room details melalui putExtra() method, termasuk room_id, room_name, capacity, facilities, dan status yang kemudian dapat diakses di destination activity untuk keperluan detail view atau reservation process.

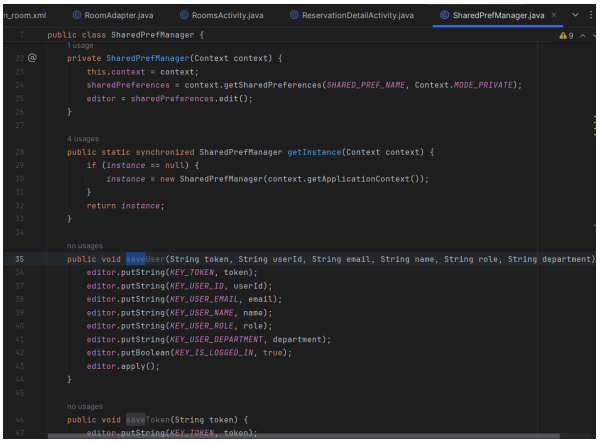
Adapter dilengkapi dengan utility methods untuk dynamic data manipulation, yaitu updateRoomList() yang memungkinkan replacement complete data source dengan clearing existing list dan adding new data, kemudian memanggil notifyDataSetChanged() untuk refresh entire view. Method updateRoomStatus() memberikan functionality untuk mengupdate status specific room berdasarkan roomId dengan melakukan iterasi through roomList, finding matching room, updating status, dan memanggil notifyItemChanged() untuk efficient partial update tanpa re-rendering entire list.

Inner class RoomViewHolder meng-extends RecyclerView.ViewHolder dan berisi references ke semua UI components dalam item layout termasuk CardView sebagai container utama, serta TextView components untuk room name, room ID, capacity, facilities, dan status display. Constructor ViewHolder menjalankan findViewById() operations untuk menginisialisasi semua view references yang diperlukan untuk data binding process, dengan menggunakan itemView parameter yang merepresentasikan inflated layout untuk setiap item dalam RecyclerView.

- **SharedPrefManager**



```
n_room.xml   RoomAdapter.java   RoomsActivity.java   ReservationDetailActivity.java   SharedPrefManager.java
                                                                                                          9
    10 usages
7   public class SharedPrefManager {
        2 usages
8       private static final String SHARED_PREF_NAME = "RoomReservationPrefs";
        3 usages
9       private static final String KEY_TOKEN = "token";
        2 usages
10      private static final String KEY_USER_ID = "user_id";
        2 usages
11      private static final String KEY_USER_EMAIL = "user_email";
        2 usages
12      private static final String KEY_USER_NAME = "user_name";
        2 usages
13      private static final String KEY_USER_ROLE = "user_role";
        2 usages
14      private static final String KEY_USER_DEPARTMENT = "user_department";
        3 usages
15      private static final String KEY_IS_LOGGED_IN = "is_logged_in";
16
        3 usages
17      private static SharedPrefManager instance;
        8 usages
18      private SharedPreferences sharedPreferences;
        16 usages
19      private SharedPreferences.Editor editor;
        2 usages
20      private Context context;
21
        1 usage
22      private SharedPrefManager(Context context) {
23          this.context = context;
```

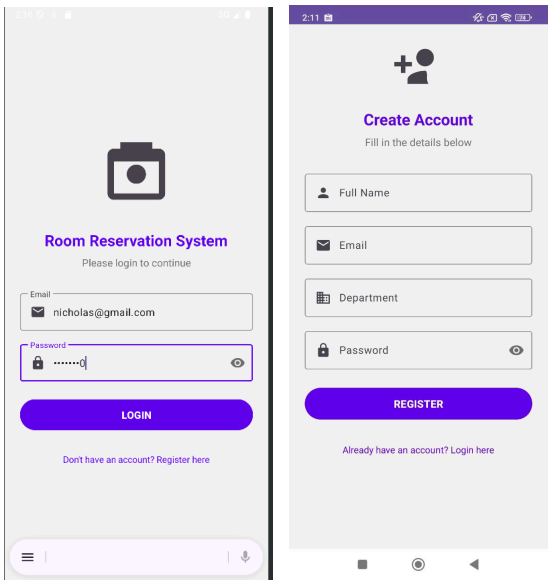


```
n_room.xml   RoomAdapter.java   RoomsActivity.java   ReservationDetailActivity.java   SharedPrefManager.java
                                                                                                          9
7   public class SharedPrefManager {
        1 usage
22      private SharedPrefManager(Context context) {
23          this.context = context;
24          sharedPreferences = context.getSharedPreferences(SHARED_PREF_NAME, Context.MODE_PRIVATE);
25          editor = sharedPreferences.edit();
26      }
27
        4 usages
28      public static synchronized SharedPrefManager getInstance(Context context) {
29          if (instance == null) {
30              instance = new SharedPrefManager(context.getApplicationContext());
31          }
32          return instance;
33      }
34
        no usages
35      public void saveUser(String token, String userId, String email, String name, String role, String department)
36          editor.putString(KEY_TOKEN, token);
37          editor.putString(KEY_USER_ID, userId);
38          editor.putString(KEY_USER_EMAIL, email);
39          editor.putString(KEY_USER_NAME, name);
40          editor.putString(KEY_USER_ROLE, role);
41          editor.putString(KEY_USER_DEPARTMENT, department);
42          editor.putBoolean(KEY_IS_LOGGED_IN, true);
43          editor.apply();
44      }
45
        no usages
46      public void saveToken(String token) {
47          editor.putString(KEY_TOKEN, token);
```

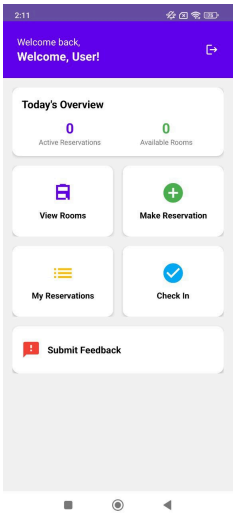
SharedPrefManager adalah class yang berfungsi sebagai wrapper untuk mengelola penyimpanan data pengguna secara lokal menggunakan SharedPreferences Android. Kelas ini menyimpan informasi penting seperti token autentikasi, ID pengguna, email, nama, peran, dan departemen dalam file preferensi bernama "RoomReservationPrefs". Implementasi singleton memastikan hanya ada satu instance dari kelas ini di seluruh aplikasi, yang diakses melalui method getInstance() dengan parameter context untuk inisialisasi yang aman menggunakan getApplicationContext().

Fungsionalitas utama kelas ini mencakup operasi CRUD (Create, Read, Update, Delete) untuk data pengguna, dengan method seperti saveUser() untuk menyimpan data lengkap pengguna, saveToken() untuk menyimpan token secara terpisah, dan berbagai getter method untuk mengambil data spesifik. Kelas ini juga menyediakan method isLoggedIn() untuk mengecek status login, getUser() yang mengembalikan objek User yang sudah terpopulasi, getAuthorizationHeader() untuk mendapatkan header Bearer token yang siap digunakan dalam API calls, serta method clear() dan logout() untuk menghapus semua data tersimpan saat user logout. Logging juga diterapkan pada method getToken() untuk debugging purposes.

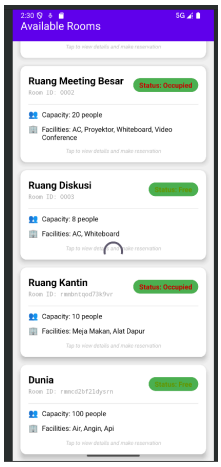
4. Hasil Pengujian



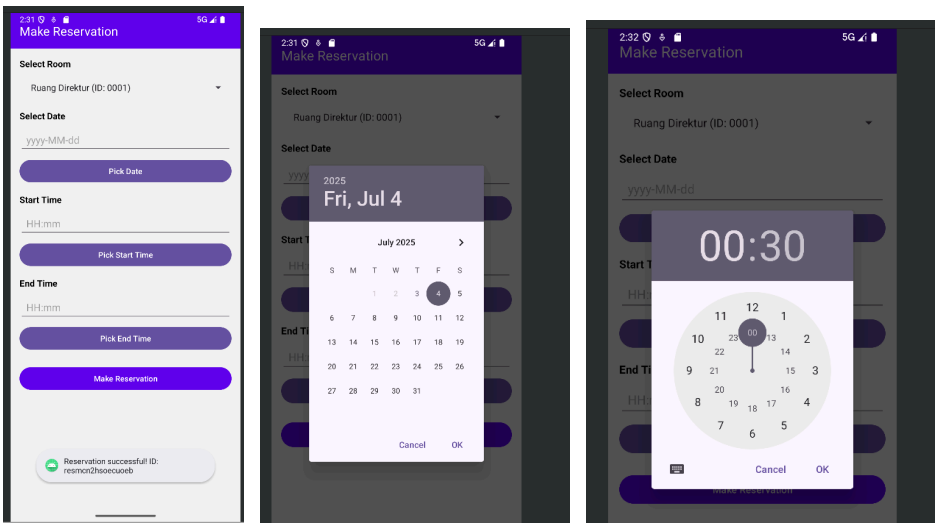
Halaman Login dari MainActivity beserta link register yang bisa terhubung dengan halaman registrasi akun ke dalam database



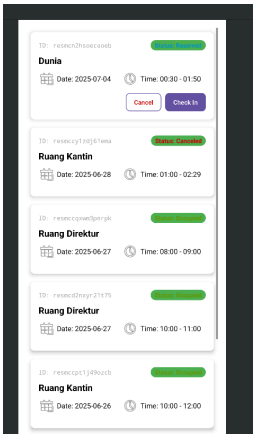
Tampilan dashboard dalam room management reservation, View Room digunakan untuk melihat list apa saja room yang tersedia, Make Reservation memberikan form yang bisa kita isi untuk melakukan reservasi ruangan, My Reservations dapat mengecek riwayat ruangan apa saja yang pernah user lakukan reservasi, Check in Merupakan halaman untuk mengecek ruang reservasi yang dibooking user, sekaligus melakukan check in ke dalam ruangan.



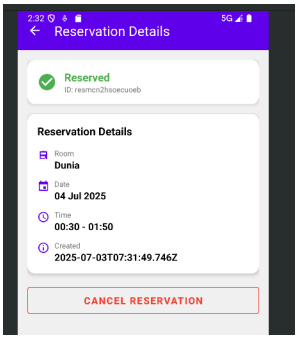
List card view Ruangan yang tersedia dari server



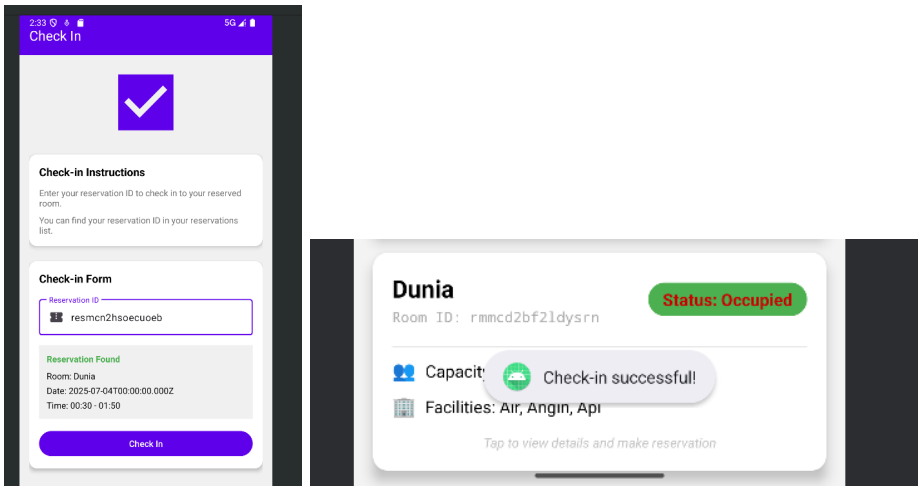
Form pengisian data mengenai reservasi ruangan



Cek perubahan riwayat reservasi ruangan pada My reservations



Tampilan Detail pada reservasi ruangan yang dimiliki



Pengecekan kode check in terhadap reservasi ruangan yang tersedia, sekaligus check in mengubah status ruangan dari reserved ke occupied