**UJIAN TENGAH SEMESTER**

**PEMROGRAMAN PLATFORM KHUSUS**

# **LAYANAN WEB SERVICE: SISTEM PERIZINAN MAHASISWA**

Dosen Pengampu: Ibnu Santoso



Disusun oleh

Nama : Titha Nur Izzah

NIM : 222212898

Kelas : 3SI1

**POLITEKNIK STATISTIKA STIS**

**TAHUN 2024/2025**

# Daftar Isi

[LAYANAN WEB SERVICE: SISTEM PERIZINAN MAHASISWA 1](#_heading=h.2aim2sif1lv2)

[Daftar Isi 2](#_heading=h.a03f1w9yclft)

[A. Pendahuluan 3](#_heading=h.e7fndqhhji6d)

[B. Proses Bisnis 3](#_heading=h.e4us1y6rgy8a)

[C. Use Case dan Activity Diagram 5](#_heading=h.452e477oy3)

[1. Use Case 5](#_heading=h.ssvxg7bimgtd)

[2. Use Case Deskripsi 5](#_heading=h.bkey0z6whqpj)

[D. Penjelasan Kode Program 6](#_heading=h.upsfpju1mrwl)

[1. Persiapan Program 6](#_heading=h.t9tusinuss1g)

[2. Entitas 7](#_heading=h.bf46vj5x0ny4)

[3. DTO (Data Transfer Object) 11](#_heading=h.8v5ei97cwbr4)

[4. Repository 14](#_heading=h.yoecvrcyww0i)

[5. Controller 16](#_heading=h.4qfcqzwjs91q)

[6. Service 22](#_heading=h.6uamxy4kukpr)

[7. Security 27](#_heading=h.a10k0rpkifw1)

[E. Dokumentasi 33](#_heading=h.2lssd2mndm5k)

[F. Pengujian Program 33](#_heading=h.fhhrn8tp5dqf)

[G. Link Program 34](#_heading=h.ek31onrjjf7)

# **Pendahuluan**

Dalam dunia perguruan tinggi, sistem manajemen perizinan mahasiswa merupakan komponen penting untuk mengelola pengajuan izin ketidakhadiran, izin sakit, dan keperluan lainnya secara efisien. Proses perizinan yang konvensional seringkali membutuhkan proses manual yang tidak hanya memakan waktu tetapi juga berpotensi menimbulkan kesalahan administrasi. Oleh karena itu, dibutuhkan sistem digital yang dapat membantu mahasiswa dalam mengajukan izin dengan mudah, serta memungkinkan pihak dosen untuk memantau dan memproses pengajuan tersebut secara efektif.

Proyek ini bertujuan untuk mengembangkan aplikasi yang mampu memfasilitasi proses perizinan mahasiswa dengan fitur-fitur utama, seperti pengajuan perizinan, manajemen pengguna, otorisasi berbasis peran, dan pelacakan status perizinan. Sistem ini didesain untuk memiliki dua peran utama: *Student* (mahasiswa), *Admin* (pihak pengelola sistem) sekaligus *Dosen BAAK* (sebagai pemberi keputusan pada pengajuan izin). Setiap peran memiliki akses dan hak yang berbeda sesuai dengan kebutuhan proses bisnis. Mahasiswa dapat mengajukan izin dan memantau status pengajuannya, sedangkan dosen BAAK memiliki otoritas untuk meninjau, menyetujui, atau menolak permohonan izin yang diajukan mahasiswa.

Sistem ini juga menggunakan autentikasi berbasis *token* (JWT) untuk menjamin keamanan akses, serta dilengkapi dokumentasi API menggunakan *Swagger*, yang memudahkan pengembangan dan pengujian layanan. Dengan adanya sistem ini, diharapkan proses perizinan mahasiswa dapat berlangsung lebih cepat, lebih transparan, dan lebih terstruktur, sehingga meningkatkan efisiensi pengelolaan izin di Politeknik Statistika STIS.

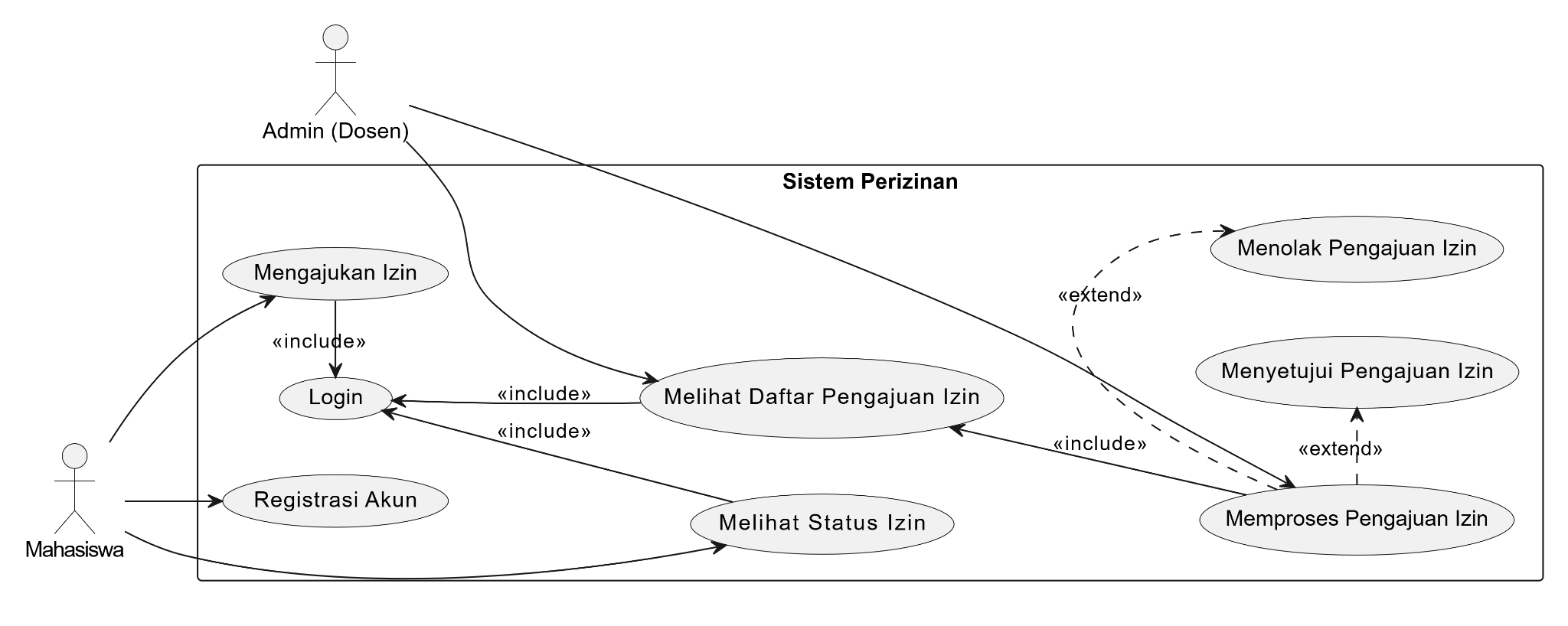
# **Proses Bisnis**

* 1. Registrasi Mahasiswa:
     + Hanya mahasiswa yang dapat melakukan registrasi melalui endpoint yang telah disediakan. Setelah registrasi, pengguna otomatis diberikan role "STUDENT".
  2. Pengajuan Perizinan oleh Mahasiswa:
     + Mahasiswa yang terdaftar dapat mengajukan perizinan, seperti izin sakit atau keperluan lain, melalui endpoint /api/perizinan/{username}.
     + Setiap pengajuan akan disimpan dengan status awal sebagai "Pending".
  3. Melihat Daftar Pengajuan Sendiri (Mahasiswa):
     + Mahasiswa hanya dapat melihat pengajuan yang mereka buat melalui endpoint /api/perizinan/{username}. Mereka tidak dapat mengakses pengajuan mahasiswa lain.
  4. Akses Admin (DOSEN) untuk Melihat dan Mengelola Pengajuan:
     + Admin dengan role "DOSEN" (username dan password ditetapkan secara manual, tidak melalui proses registrasi) memiliki wewenang penuh untuk melihat seluruh pengajuan perizinan dari semua mahasiswa.
     + Admin dapat melihat daftar semua pengajuan melalui endpoint yang baru ditambahkan (/api/perizinan), yang khusus diakses oleh pengguna dengan role "DOSEN".
  5. Pemrosesan Pengajuan oleh Admin (DOSEN):
     + Admin dapat memperbarui status pengajuan izin melalui endpoint /api/perizinan/{id}/status, di mana admin bisa menyetujui atau menolak permohonan.
     + Mahasiswa kemudian bisa melihat status perizinan mereka yang telah diproses oleh admin.
  6. Keamanan dan Kontrol Akses:
     + Setiap endpoint dibatasi sesuai dengan role pengguna. Mahasiswa hanya bisa mengakses endpoint untuk membuat dan melihat pengajuan mereka sendiri.
     + Admin dapat mengakses semua pengajuan mahasiswa dan memproses statusnya, memberikan otoritas penuh atas persetujuan atau penolakan izin yang diajukan.

Dengan alur ini, sistem mengatur dengan jelas batasan dan wewenang setiap peran (STUDENT dan DOSEN), mendukung alur persetujuan izin yang terstruktur serta aman.

# **Use Case dan Use Case Description**

## Use Case



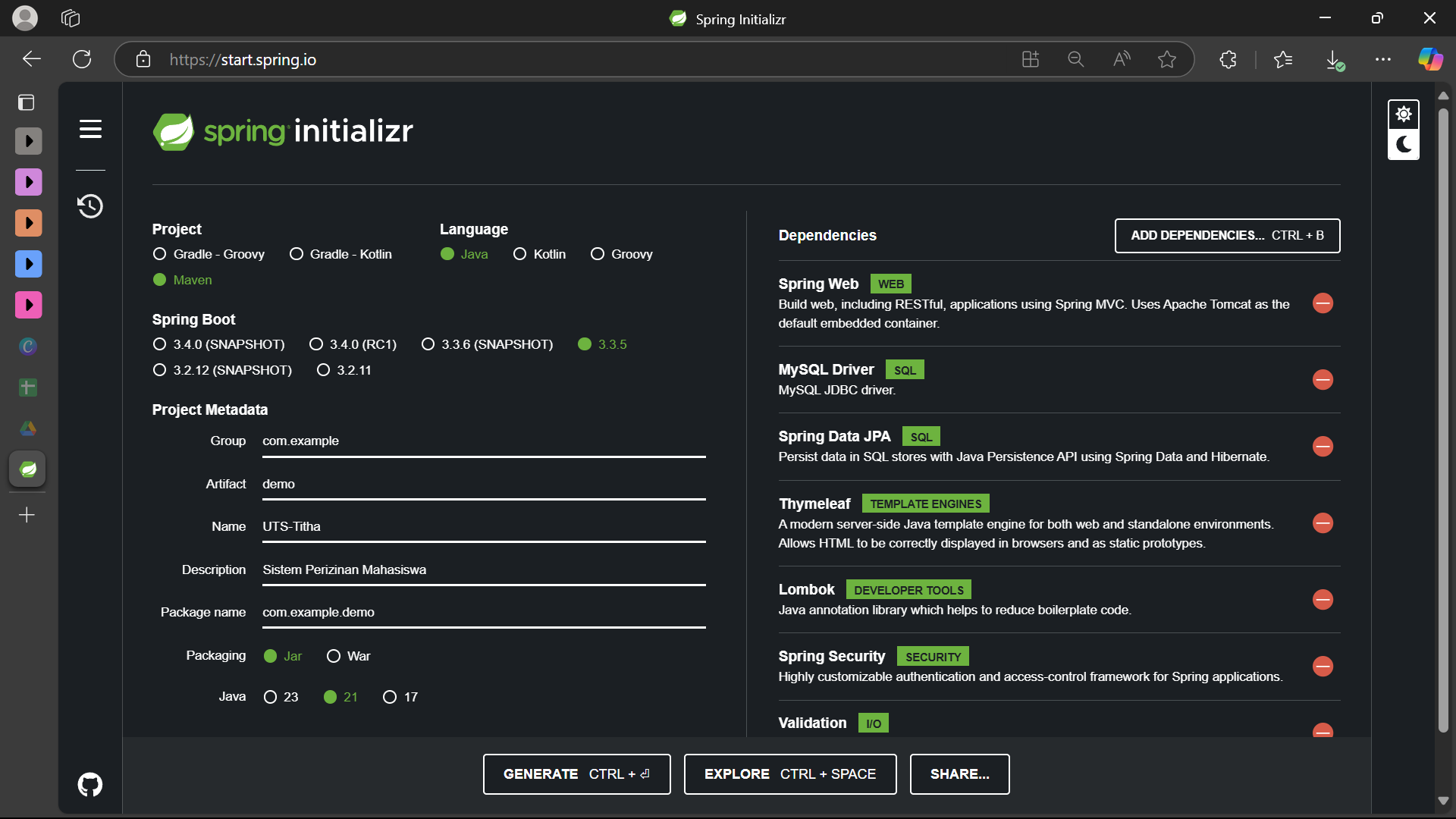
## Use Case Description

| Use Case | Aktor | Deskripsi | Pre-Condition | Post-Condition |
| --- | --- | --- | --- | --- |
| Registrasi Akun | Mahasiswa | Mahasiswa membuat akun untuk dapat mengakses sistem. | Mahasiswa mengakses endpoint | Akun mahasiswa dibuat dengan role "STUDENT". |
| Mengajukan Izin | Mahasiswa | Mahasiswa mengajukan izin untuk suatu keperluan, seperti sakit. | Mahasiswa terdaftar dan login | Pengajuan izin dibuat dengan status "Pending". |
| Melihat Status Izin | Mahasiswa | Mahasiswa dapat melihat status dari izin yang telah diajukannya. | Mahasiswa memiliki pengajuan izin | Daftar izin mahasiswa ditampilkan. |
| Melihat Daftar Pengajuan Izin | Dosen (Admin) | Admin melihat semua pengajuan izin dari seluruh mahasiswa. | Admin login ke sistem | Semua pengajuan izin ditampilkan untuk admin. |
| Memproses Pengajuan Izin | Dosen (Admin) | Admin dapat menyetujui atau menolak pengajuan izin mahasiswa. | Pengajuan izin berstatus "Pending" | Status izin diubah menjadi "Approved" atau "Rejected". |

# **Penjelasan Kode Program**

## Persiapan Program

Pertama-tama saya mengenerate file pada spring initializr juga menambahkan dependensinya



Selain itu, saya menambahkan dependensi yang belum tersedia di pom.xml

<!-- JWT for Token-Based Authentication -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<!-- SpringDoc OpenAPI for Swagger documentation -->

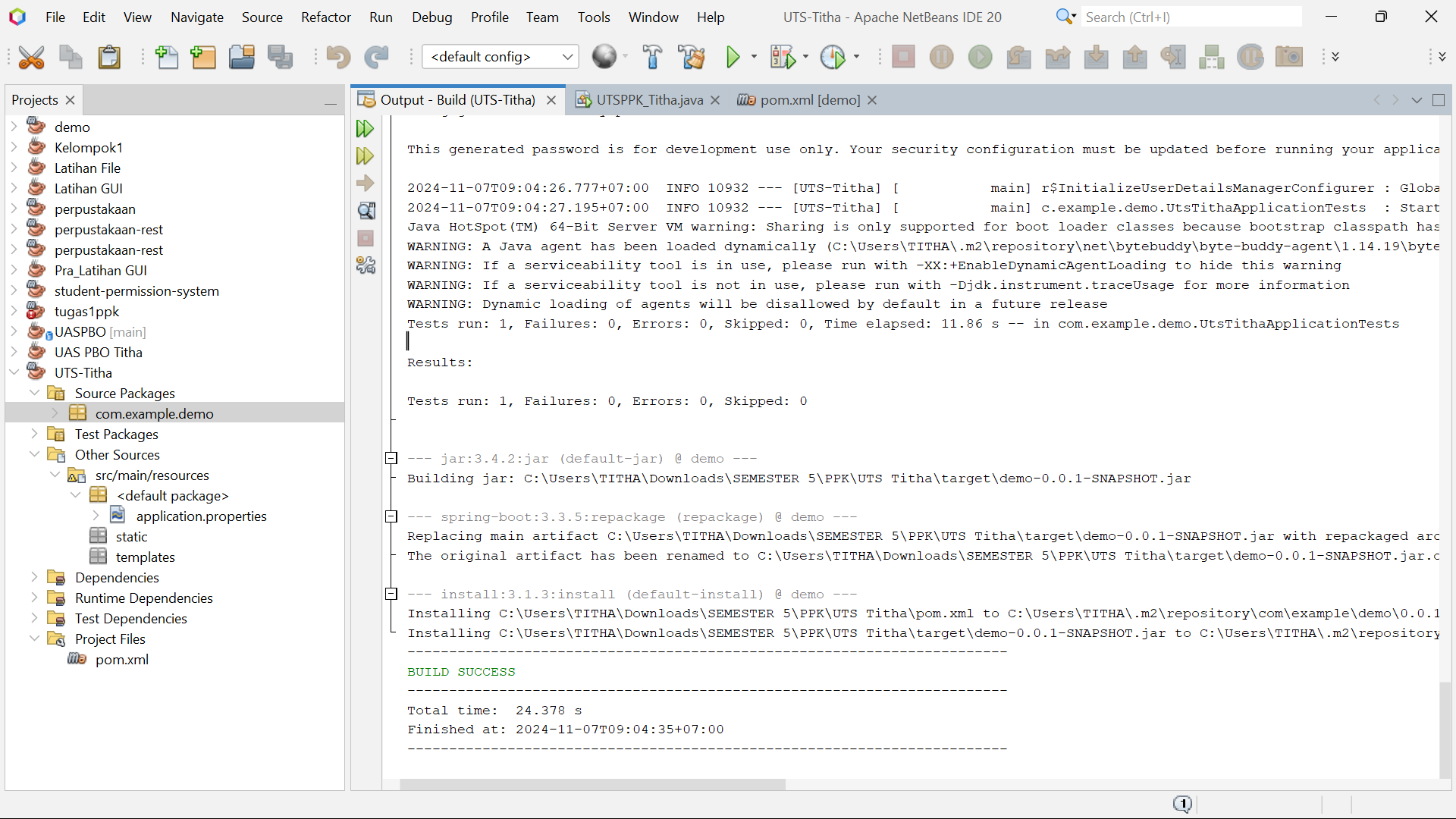
<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.1.0</version>

</dependency>



Setelah itu, lakukan build with dependencies untuk mengaktifkan dependensi yang terdapat pada pom.xml

## Entitas

Menyimpan definisi struktur data yang disimpan dalam basis data. Setiap class dalam package ini akan direpresentasikan sebagai tabel dalam basis data. Anotasi JPA digunakan untuk memetakan atribut di class dengan kolom dalam tabel basis data.

* User

Entitas User merepresentasikan pengguna aplikasi, baik itu mahasiswa maupun admin.

package com.example.perizinan.entity;

import jakarta.persistence.\*;

import lombok.\*;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;

import java.util.List;

import java.util.stream.Collectors;

@Entity

@Table(name = "users")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class User implements UserDetails {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String username;

@Column(nullable = false)

private String password;

@Column(nullable = false)

private String email;

@Column(nullable = false)

private String role; // ROLE\_USER atau ROLE\_ADMIN

// Relasi satu user bisa memiliki banyak permissions (izin yang diajukan)

@OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

private List<Perizinan> permissions;

// Relasi satu user bisa memiliki banyak token (untuk autentikasi)

@OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

private List<Token> tokens;

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

// Mengembalikan daftar role sebagai otoritas, misalnya "ROLE\_USER" atau "ROLE\_ADMIN"

return List.of(new SimpleGrantedAuthority(role));

}

@Override

public String getPassword() {

return password;

}

@Override

public String getUsername() {

return username;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

}

* Perizinan

Entitas Permission merepresentasikan izin yang diajukan mahasiswa, seperti izin sakit, dispensasi, lupa presensi, dll.

package com.example.perizinan.entity;

import jakarta.persistence.\*;

import lombok.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "permissions")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Perizinan {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String tipe; // Contoh: "Sakit", "Dispensasi", "Lupa Presensi", dll.

@Column(nullable = false)

private String alasan; // Alasan pengajuan izin

@Column(nullable = false)

private LocalDateTime startDate; // Tanggal mulai izin

@Column(nullable = false)

private LocalDateTime endDate; // Tanggal akhir izin

@Column(nullable = false)

private String status; // Status: "Pending", "Approved", "Rejected"

// Relasi banyak permission diajukan oleh satu user

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "user\_id", nullable = false)

private User user;

}

* Token

Entitas Token menyimpan informasi token JWT yang digunakan untuk autentikasi pengguna.

package com.example.perizinan.entity;

import jakarta.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "tokens")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Token {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String tokenValue;

@Column(nullable = false)

private boolean isExpired = false; // Apakah token sudah kadaluarsa

@Column(nullable = false)

private boolean isRevoked = false; // Apakah token telah dicabut

// Relasi banyak token dimiliki oleh satu user

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "user\_id", nullable = false)

private User user;

}

## DTO (Data Transfer Object)

Package ini berisi class yang digunakan untuk mengirim data antara lapisan (layer) tanpa mengekspose langsung entitas (entity) dari basis data. Hal ini membantu dalam menjaga keamanan dan integritas data yang dikirim dan diterima.

* LoginDTO

LoginDTO digunakan untuk transfer data saat pengguna melakukan login. DTO ini hanya membutuhkan username dan password sebagai informasi login.

package com.example.perizinan.dto;

import lombok.Data;

@Data

public class LoginDTO {

private String username;

private String password;

}

* PerizinanDTO

PermissionDTO digunakan untuk transfer data terkait izin yang diajukan pengguna. DTO ini berisi informasi izin seperti tipe, alasan, tanggal mulai, dan akhir izin.

package com.example.perizinan.dto;

import lombok.Data;

import jakarta.validation.constraints.\*;

import java.time.LocalDateTime;

@Data

public class PerizinanDTO {

private Long id;

@NotNull

private String tipe; // Contoh: "Sakit", "Dispensasi", "Lupa Presensi", dll.

@NotNull

private String alasan;

@NotNull

private LocalDateTime startDate;

@NotNull

private LocalDateTime endDate;

@NotNull

private Status status; // Enum untuk Status

}

enum Status {

PENDING, APPROVED, REJECTED

}

* UserDTO

UserDTO digunakan untuk transfer data pengguna antara client dan server. DTO ini berisi informasi umum pengguna yang aman untuk dibagikan.

package com.example.perizinan.dto;

import lombok.Data;

import jakarta.validation.constraints.\*;

@Data

public class UserDTO {

private Long id;

@NotNull

@Size(min = 3, max = 20, message = "Username harus terdiri dari 3 hingga 20 karakter.")

private String username;

@NotNull

@Email(message = "Format email tidak valid.")

private String email;

@NotNull

private String role; // ROLE\_USER atau ROLE\_ADMIN

}

* UserRegisterDTO

package com.example.perizinan.dto;

import lombok.Data;

import jakarta.validation.constraints.\*;

@Data

public class UserRegisterDTO {

@NotNull

@Size(min = 3, max = 20, message = "Username harus terdiri dari 3 hingga 20 karakter.")

private String username;

@NotNull

@Email(message = "Format email tidak valid.")

private String email;

@NotNull

@Size(min = 6, message = "Password harus terdiri dari minimal 6 karakter.")

private String password;

@NotNull

private String role; // ROLE\_USER atau ROLE\_ADMIN

}

## Repository

Package ini berisi interface yang mengatur akses data ke basis data menggunakan JPA (Java Persistence API). Repository digunakan untuk mempermudah akses CRUD (Create, Read, Update, Delete) dan pengelolaan data lainnya di basis data.

* PerizinanRepository

PerizinanRepository adalah repository untuk entitas Perizinan. Repository ini juga menggunakan JpaRepository.

package com.example.perizinan.repository;

import com.example.perizinan.entity.Perizinan;

import com.example.perizinan.entity.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface PerizinanRepository extends JpaRepository<Perizinan, Long> {

List<Perizinan> findByUser(User user);

List<Perizinan> findByStatus(String status);

List<Perizinan> findByUserAndStatus(User user, String status); // Tambahan untuk mencari izin spesifik pengguna

}

* TokenRepository

Repository ini memungkinkan pengelolaan token autentikasi, seperti penyimpanan dan pencarian token, serta pengecekan status token untuk autentikasi pengguna.

package com.example.perizinan.repository;

import com.example.perizinan.entity.Token;

import com.example.perizinan.entity.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository

public interface TokenRepository extends JpaRepository<Token, Long> {

Optional<Token> findByTokenValue(String tokenValue);

void deleteByUser(User user); // Metode tambahan untuk menghapus semua token pengguna

}

* UserRepository

UserRepository adalah repository untuk entitas User. Repository ini menggunakan antarmuka JpaRepository yang disediakan oleh Spring Data JPA.

package com.example.perizinan.repository;

import com.example.perizinan.entity.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username); // Cari user berdasarkan username

boolean existsByUsername(String username); // Cek apakah username sudah ada

boolean existsByEmail(String email); // Cek apakah email sudah ada

}

## Controller

Package ini berfungsi untuk menangani request HTTP dari pengguna dan mengirimkan response kembali. Setiap endpoint diatur dalam class controller yang sesuai dengan fungsionalitas spesifik, seperti authentikasi, perizinan, dan pengelolaan pengguna.

* AuthController

AuthController mengelola autentikasi, seperti login dan logout.

package com.example.perizinan.controllers;

import com.example.perizinan.dto.LoginDTO;

import com.example.perizinan.entity.Token;

import com.example.perizinan.service.AuthService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.Optional;

@RestController

@RequestMapping("/api/auth")

public class AuthController {

@Autowired

private AuthService authService;

@PostMapping("/login")

public ResponseEntity<Token> login(@RequestBody LoginDTO loginDTO) {

Optional<Token> tokenOpt = authService.login(loginDTO);

return tokenOpt.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.status(401).build());

}

@PostMapping("/logout")

public ResponseEntity<Void> logout(@RequestHeader("Authorization") String tokenValue) {

authService.logout(tokenValue.replace("Bearer ", ""));

return ResponseEntity.noContent().build();

}

}

* PerizinanController

PerizinanController mengelola operasi terkait perizinan, seperti pengajuan izin, melihat izin, dan mengubah status izin.

package com.example.perizinan.controllers;

import com.example.perizinan.dto.PerizinanDTO;

import com.example.perizinan.entity.Perizinan;

import com.example.perizinan.entity.User;

import com.example.perizinan.service.PerizinanService;

import com.example.perizinan.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/perizinan")

public class PerizinanController {

@Autowired

private PerizinanService perizinanService;

@Autowired

private UserService userService;

// Membuat pengajuan perizinan baru (hanya mahasiswa yang bisa mengakses)

@PostMapping("/{username}")

@PreAuthorize("hasRole('STUDENT')")

public ResponseEntity<Perizinan> createPerizinan(@PathVariable String username, @RequestBody PerizinanDTO perizinanDTO) {

Optional<User> userOpt = userService.findByUsername(username);

if (userOpt.isPresent()) {

Perizinan perizinan = new Perizinan();

perizinan.setUser(userOpt.get());

perizinan.setTipe(perizinanDTO.getTipe());

perizinan.setAlasan(perizinanDTO.getAlasan());

perizinan.setStartDate(perizinanDTO.getStartDate());

perizinan.setEndDate(perizinanDTO.getEndDate());

perizinan.setStatus("Pending"); // Status awal adalah "Pending"

return ResponseEntity.ok(perizinanService.createPerizinan(perizinan));

}

return ResponseEntity.notFound().build();

}

// Mendapatkan daftar perizinan pengguna tertentu (hanya mahasiswa yang bisa mengakses untuk melihat izin mereka sendiri)

@GetMapping("/{username}")

@PreAuthorize("hasRole('STUDENT')")

public ResponseEntity<List<Perizinan>> getPerizinanByUser(@PathVariable String username) {

Optional<User> userOpt = userService.findByUsername(username);

if (userOpt.isPresent()) {

return ResponseEntity.ok(perizinanService.getPerizinanByUser(userOpt.get()));

}

return ResponseEntity.notFound().build();

}

// Mendapatkan semua pengajuan izin (hanya bisa diakses oleh dosen/admin)

@GetMapping("/all")

@PreAuthorize("hasRole('DOSEN')")

public ResponseEntity<List<Perizinan>> getAllPerizinan() {

List<Perizinan> allPerizinan = perizinanService.getAllPerizinan();

return ResponseEntity.ok(allPerizinan);

}

// Mengupdate status perizinan (menyetujui atau menolak - hanya dosen yang bisa mengakses)

@PutMapping("/{id}/status")

@PreAuthorize("hasRole('DOSEN')")

public ResponseEntity<Perizinan> updatePerizinanStatus(@PathVariable Long id, @RequestBody String status) {

Optional<Perizinan> perizinanOpt = perizinanService.getPerizinanById(id);

if (perizinanOpt.isPresent()) {

Perizinan perizinan = perizinanOpt.get();

perizinan.setStatus(status); // Dosen mengubah status (misalnya "Approved" atau "Rejected")

return ResponseEntity.ok(perizinanService.updatePerizinan(perizinan));

}

return ResponseEntity.notFound().build();

}

}

* UserController

UserController mengelola operasi terkait pengguna, seperti registrasi, pengaturan profil, dan penghapusan akun.

package com.example.perizinan.controllers;

import com.example.perizinan.dto.UserDTO;

import com.example.perizinan.dto.UserRegisterDTO;

import com.example.perizinan.entity.User;

import com.example.perizinan.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.Optional;

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserService userService;

@PostMapping("/register")

public ResponseEntity<User> registerUser(@RequestBody UserRegisterDTO userRegisterDTO) {

User user = new User();

user.setUsername(userRegisterDTO.getUsername());

user.setPassword(userRegisterDTO.getPassword()); // pastikan password dienkripsi di UserService

user.setEmail(userRegisterDTO.getEmail());

user.setRole(userRegisterDTO.getRole());

User registeredUser = userService.registerUser(user);

return ResponseEntity.ok(registeredUser);

}

@GetMapping("/{username}")

public ResponseEntity<UserDTO> getUserProfile(@PathVariable String username) {

Optional<User> userOpt = userService.findByUsername(username);

if (userOpt.isPresent()) {

User user = userOpt.get();

UserDTO userDTO = new UserDTO();

userDTO.setId(user.getId());

userDTO.setUsername(user.getUsername());

userDTO.setEmail(user.getEmail());

userDTO.setRole(user.getRole());

return ResponseEntity.ok(userDTO);

}

return ResponseEntity.notFound().build();

}

@PutMapping("/{username}")

public ResponseEntity<UserDTO> updateUserProfile(@PathVariable String username, @RequestBody UserDTO userDTO) {

Optional<User> userOpt = userService.findByUsername(username);

if (userOpt.isPresent()) {

User user = userOpt.get();

user.setEmail(userDTO.getEmail());

User updatedUser = userService.updateUser(user);

userDTO.setId(updatedUser.getId());

userDTO.setUsername(updatedUser.getUsername());

userDTO.setEmail(updatedUser.getEmail());

userDTO.setRole(updatedUser.getRole());

return ResponseEntity.ok(userDTO);

}

return ResponseEntity.notFound().build();

}

@DeleteMapping("/{username}")

public ResponseEntity<Void> deleteUser(@PathVariable String username) {

Optional<User> userOpt = userService.findByUsername(username);

if (userOpt.isPresent()) {

userService.deleteUser(userOpt.get().getId());

return ResponseEntity.noContent().build();

}

return ResponseEntity.notFound().build();

}

}

## Service

Package service berisi komponen utama untuk implementasi business logic aplikasi. Dalam arsitektur layered, package ini berperan sebagai jembatan antara controller dan repository. Package ini menerima data dari controller, melakukan pengolahan sesuai aturan bisnis yang diperlukan, dan berkomunikasi dengan repository untuk penyimpanan atau pengambilan data dari basis data.

* AuthService

AuthService bertanggung jawab atas autentikasi pengguna, termasuk login dan pengelolaan token.

package com.example.perizinan.service;

import com.example.perizinan.dto.LoginDTO;

import com.example.perizinan.entity.Token;

import com.example.perizinan.entity.User;

import com.example.perizinan.repository.TokenRepository;

import com.example.perizinan.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class AuthService {

@Autowired

private UserRepository userRepository;

@Autowired

private TokenRepository tokenRepository;

@Autowired

private PasswordEncoder passwordEncoder;

// Login

public Optional<Token> login(LoginDTO loginDTO) {

Optional<User> userOpt = userRepository.findByUsername(loginDTO.getUsername());

if (userOpt.isPresent() && passwordEncoder.matches(loginDTO.getPassword(), userOpt.get().getPassword())) {

Token token = new Token();

token.setTokenValue(generateToken()); // Implementasikan metode generateToken()

token.setUser(userOpt.get());

tokenRepository.save(token);

return Optional.of(token);

}

return Optional.empty();

}

// Logout

public void logout(String tokenValue) {

Optional<Token> tokenOpt = tokenRepository.findByTokenValue(tokenValue);

tokenOpt.ifPresent(token -> {

token.setExpired(true);

tokenRepository.save(token);

});

}

// Metode untuk menghasilkan token (Anda bisa mengganti implementasi sesuai kebutuhan)

private String generateToken() {

return java.util.UUID.randomUUID().toString();

}

}

* PerizinanService

PerizinanService bertanggung jawab atas operasi yang berkaitan dengan perizinan mahasiswa, seperti pengajuan izin dan update status.

package com.example.perizinan.service;

import com.example.perizinan.entity.Perizinan;

import com.example.perizinan.entity.User;

import com.example.perizinan.repository.PerizinanRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class PerizinanService {

@Autowired

private PerizinanRepository perizinanRepository;

// Membuat pengajuan izin baru

public Perizinan createPerizinan(Perizinan perizinan) {

return perizinanRepository.save(perizinan);

}

// Mendapatkan daftar izin berdasarkan pengguna (untuk mahasiswa melihat izin mereka sendiri)

public List<Perizinan> getPerizinanByUser(User user) {

return perizinanRepository.findByUser(user);

}

// Mendapatkan izin berdasarkan ID

public Optional<Perizinan> getPerizinanById(Long id) {

return perizinanRepository.findById(id);

}

// Mengupdate izin (menyimpan perubahan status atau detail lainnya)

public Perizinan updatePerizinan(Perizinan perizinan) {

return perizinanRepository.save(perizinan);

}

// Mendapatkan semua pengajuan izin (untuk admin atau dosen melihat seluruh izin yang diajukan)

public List<Perizinan> getAllPerizinan() {

return perizinanRepository.findAll();

}

}

* UserService

UserService bertanggung jawab atas operasi yang berkaitan dengan pengguna, seperti pendaftaran dan manajemen profil.

package com.example.perizinan.service;

import com.example.perizinan.entity.User;

import com.example.perizinan.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import java.util.Collections;

import java.util.Optional;

@Service

public class UserService implements UserDetailsService {

private final UserRepository userRepository;

private final PasswordEncoder passwordEncoder;

@Autowired

public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {

this.userRepository = userRepository;

this.passwordEncoder = passwordEncoder;

}

// Load user by username for authentication

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found with username: " + username));

return new org.springframework.security.core.userdetails.User(

user.getUsername(),

user.getPassword(),

Collections.singletonList(new SimpleGrantedAuthority("ROLE\_" + user.getRole()))

);

}

// Register a new user

public User registerUser(User user) {

// Encrypt password before saving

user.setPassword(passwordEncoder.encode(user.getPassword()));

return userRepository.save(user);

}

// Update user information

public User updateUser(User user) {

// Additional logic for updating user can be added here

return userRepository.save(user);

}

// Delete user by ID

public void deleteUser(Long userId) {

userRepository.deleteById(userId);

}

// Find user by username

public Optional<User> findByUsername(String username) {

return userRepository.findByUsername(username);

}

}

## Security

Package ini mengelola konfigurasi keamanan aplikasi, termasuk filter untuk autentikasi JWT, pengaturan role, dan konfigurasi keamanan Spring Security.

* + - JwtAuthenticationFilter

Filter ini digunakan untuk menangkap dan memverifikasi JWT dari header Authorization setiap permintaan.

package com.example.perizinan.security;

import com.example.perizinan.service.UserService;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

public class JwtAuthenticationFilter extends OncePerRequestFilter {

private final JwtUtil jwtUtil;

private final UserService userService;

public JwtAuthenticationFilter(JwtUtil jwtUtil, UserService userService) {

this.jwtUtil = jwtUtil;

this.userService = userService;

}

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = userService.loadUserByUsername(username);

if (jwtUtil.isTokenValid(jwt, userDetails.getUsername())) {

UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

chain.doFilter(request, response);

}

}

* + - JwtUtil

Untuk mendukung JWT, pertama kita memerlukan utilitas untuk menghasilkan dan memverifikasi token JWT.

package com.example.perizinan.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.function.Function;

@Component

public class JwtUtil {

private String SECRET\_KEY = "your\_secret\_key";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

public Boolean isTokenValid(String token, String username) {

final String extractedUsername = extractUsername(token);

return (extractedUsername.equals(username) && !isTokenExpired(token));

}

private Boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10)) // 10 jam

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY)

.compact();

}

}

* + - SecurityConfig

Di konfigurasi ini, kita mengatur agar hanya endpoint login dan register yang dapat diakses tanpa autentikasi. Endpoint lain membutuhkan autentikasi dengan token JWT.

package com.example.perizinan.security;

import com.example.perizinan.service.UserService;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.AuthenticationProvider;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

private final UserService userService;

private final PasswordEncoder passwordEncoder;

// Constructor injection untuk UserService dan PasswordEncoder

public SecurityConfig(UserService userService, PasswordEncoder passwordEncoder) {

this.userService = userService;

this.passwordEncoder = passwordEncoder;

}

@Bean

public JwtAuthenticationFilter jwtAuthenticationFilter() {

return new JwtAuthenticationFilter(jwtUtil(), userService);

}

@Bean

public JwtUtil jwtUtil() {

return new JwtUtil();

}

@Bean

public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {

AuthenticationManagerBuilder auth = http.getSharedObject(AuthenticationManagerBuilder.class);

auth.userDetailsService(userService).passwordEncoder(passwordEncoder);

return auth.build();

}

@Bean

public AuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(userService);

authProvider.setPasswordEncoder(passwordEncoder);

return authProvider;

}

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.csrf().disable()

.authorizeHttpRequests()

.requestMatchers("/swagger-ui/\*\*", "/v3/api-docs/\*\*", "/swagger-ui.html").permitAll() // Mengizinkan akses tanpa autentikasi untuk Swagger

.requestMatchers("/api/auth/login", "/api/users/register").permitAll()

.requestMatchers("/api/permissions/\*\*").hasRole("DOSEN") // Hanya bisa diakses oleh dosen

.requestMatchers(HttpMethod.GET, "/api/public/\*\*").permitAll()

.anyRequest().authenticated()

.and()

.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authenticationProvider(authenticationProvider())

.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

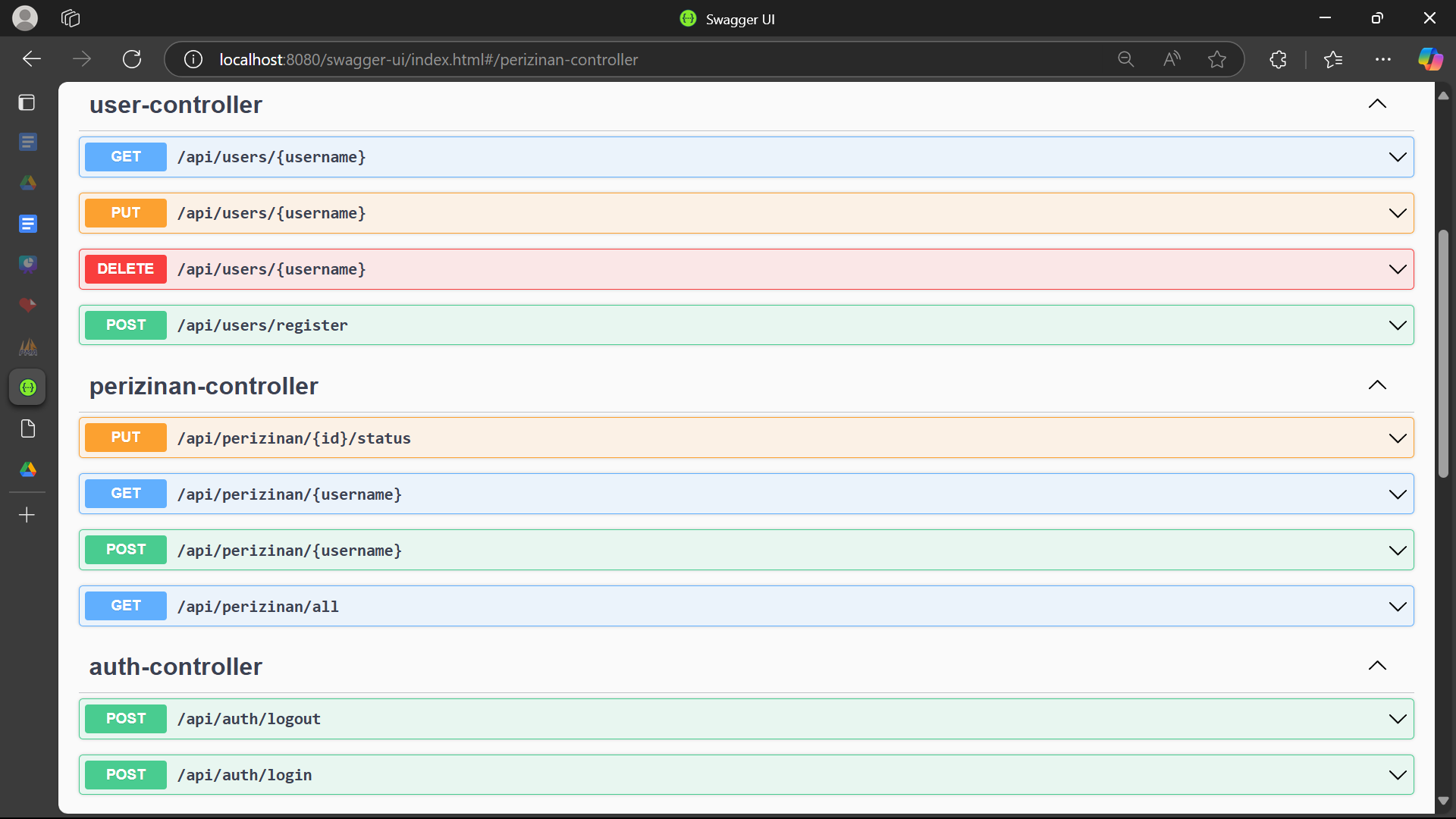
return http.build();

}

}

# **Dokumentasi**

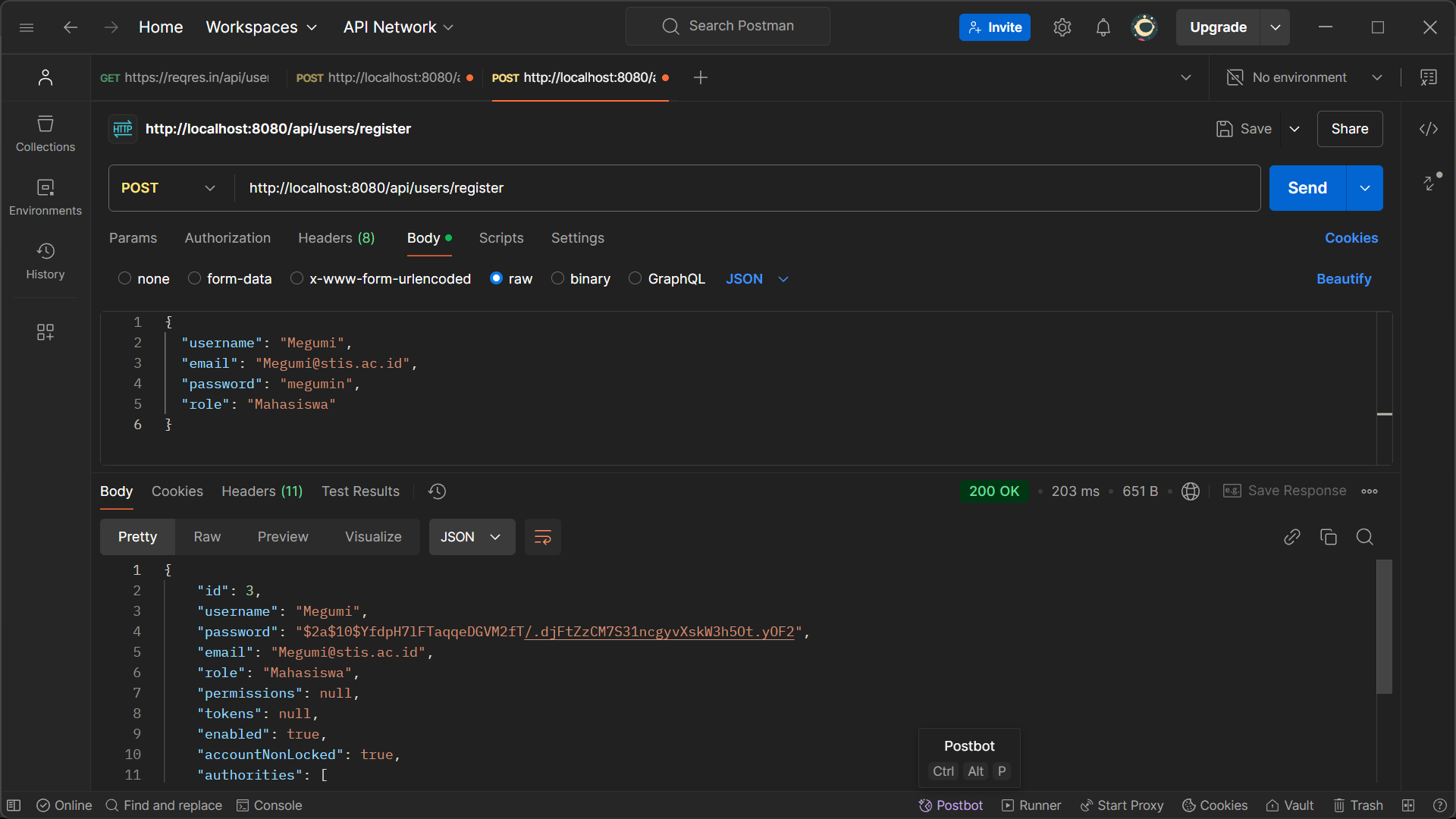
Berikut adalah Dokumentasi menggunakan swagger



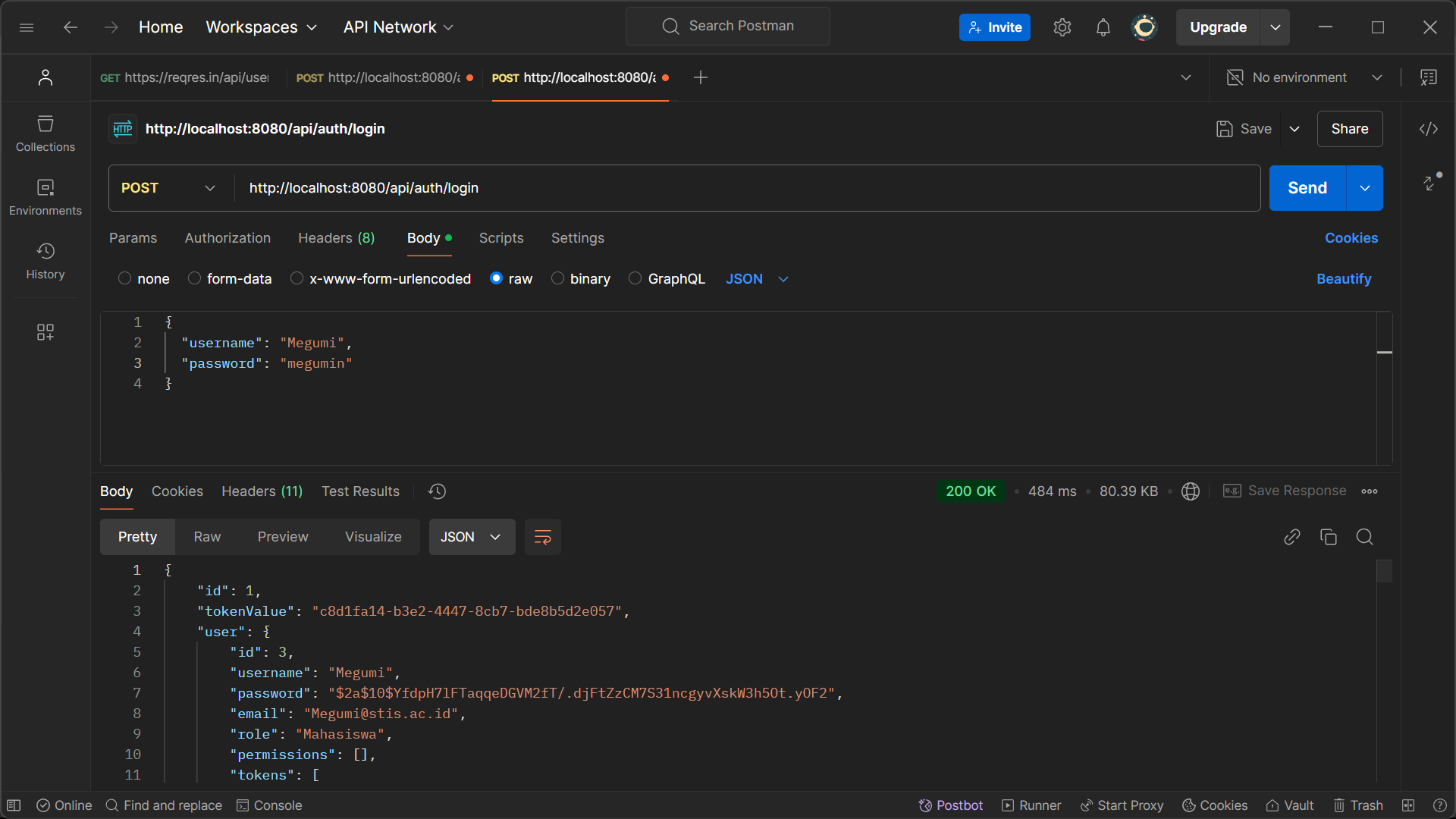
# **Pengujian Program**

1. Mahasiswa

* Registrasi Akun Mahasiswa



* Login Mahasiswa



* Pengajuan Izin
* Melihat Daftar Pengajuan
* Mengedit Profil Pengguna
* Menghapus Akun

1. Admin/BAAK

* Login Admin
* Melihat Semua Pengajuan Izin
* Menyetujui/Menolak Pengajuan Izin
* Mengedit Profil Pengguna

# **Link Program**