

Laporan Ujian Tengah Semester

Pemrograman Platform Khusus



Oleh :

Pamungkas Dero Ivano

NIM : 222212813

Kelas : 3SI1

DIV Komputasi Statistik

Politeknik Statistika STIS

Tahun 2024

1. Latar Belakang

Digiexam adalah sebuah sistem manajemen ujian online yang dikembangkan untuk menggantikan sistem ujian berbasis Google Form yang saat ini digunakan di Polstat-STIS. Meskipun Google Form merupakan alat yang mudah digunakan untuk menyelenggarakan ujian secara online, namun terdapat berbagai keterbatasan dalam hal fungsionalitas dan pengelolaan ujian yang lebih kompleks. Sistem ujian berbasis Google Form tidak memberikan kemudahan dalam hal pengelolaan soal ujian yang terstruktur, verifikasi identitas peserta ujian, serta pengaturan waktu ujian yang lebih fleksibel. Selain itu, proses pengolahan nilai dan pencatatan hasil ujian masih sangat manual, sehingga membutuhkan banyak waktu dan tenaga untuk melakukan verifikasi dan perhitungan.

Dengan menggunakan Digiexam, diharapkan seluruh proses ujian bisa lebih terorganisir, efisien, dan terintegrasi, mulai dari pembuatan soal, penjadwalan ujian, pelaksanaan ujian, hingga pengolahan hasil ujian. Digiexam dirancang untuk memungkinkan pengelola ujian untuk lebih mudah mengelola soal-soal ujian dan bank soal secara terpusat, sementara peserta ujian dapat melakukan ujian secara lebih terstruktur dan aman dengan autentikasi berbasis token. Selain itu, Digiexam juga dilengkapi dengan fitur manajemen pengguna yang memungkinkan setiap peserta ujian untuk memiliki akun yang terverifikasi, serta dapat mengakses dan mengedit profil mereka, mengganti password, atau bahkan menghapus akun mereka sesuai kebutuhan. Sistem ini diharapkan dapat meningkatkan kualitas dan efisiensi pelaksanaan ujian di Polstat-STIS dengan mengatasi keterbatasan yang ada pada sistem ujian sebelumnya.

2. Tujuan Pembuatan Aplikasi

Pembuatan aplikasi Digiexam bertujuan untuk:

- a. Membuat layanan web berbasis RESTful API menggunakan Java dan Spring Boot.
- b. Mengimplementasikan manajemen pengguna, termasuk registrasi, login, profil, edit profil, ganti password, dan hapus akun.
- c. Mengembangkan proses bisnis utama yaitu manajemen ujian, manajemen soal, dan penyimpanan hasil ujian.
- d. Menggunakan token-based authentication untuk semua endpoint yang memerlukan autentikasi pengguna.
- e. Menyediakan dokumentasi API menggunakan Swagger UI dan SpringDoc OpenAPI.

3. Deskripsi Proses Bisnis

Proses bisnis yang diimplementasikan adalah Manajemen Ujian Online & Bank Soal di Polstat-STIS, yang mencakup:

- a. Manajemen pengguna: registrasi, login, pengelolaan profil, perubahan role oleh admin, perubahan password oleh pengguna, dan autentikasi.
- b. Manajemen soal: pembuatan soal, pencarian soal, dan pengambilan data soal untuk ujian.
- c. Manajemen ujian: pembuatan ujian, pencarian ujian, dan pengambilan data ujian.
- d. Manajemen hasil ujian: penghitungan otomatis nilai ujian mahasiswa, pengiriman hasil ujian, pencarian hasil ujian pengguna.

4. Implementasi

Implementasi sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS ini bertujuan untuk mempermudah pengelolaan ujian dan soal ujian secara digital. Aplikasi ini dibangun menggunakan bahasa pemrograman Java dengan framework Spring Boot untuk sisi backend, serta menggunakan JWT (JSON Web Token) untuk autentikasi pengguna. Pendekatan yang digunakan dalam pengembangan aplikasi ini adalah Model-View-Controller (MVC), yang memungkinkan pemisahan logika bisnis, tampilan antarmuka, dan kontrol alur aplikasi, sehingga kode menjadi lebih terstruktur dan mudah dikelola.

Pada bagian Model, entitas utama yang diimplementasikan adalah User, Exam, dan Question. Entitas User bertanggung jawab untuk menyimpan data pengguna, seperti nama, email, dan kata sandi, yang digunakan dalam proses registrasi, login, dan manajemen profil. Entitas Exam digunakan untuk menyimpan informasi tentang ujian, seperti judul ujian, deskripsi, serta waktu pelaksanaan ujian. Sedangkan entitas Question menyimpan data terkait soal ujian, termasuk pertanyaan dan pilihan jawaban. Setiap entitas ini berinteraksi dengan database menggunakan JPA (Java Persistence API), yang mempermudah pengelolaan data melalui repository, seperti UserRepository, ExamRepository, dan QuestionRepository.

Bagian View dalam aplikasi ini tidak memiliki tampilan antarmuka pengguna langsung karena aplikasi ini berbasis RESTful API. Pengguna berinteraksi dengan sistem melalui permintaan HTTP yang dikirimkan melalui alat seperti Postman atau aplikasi frontend lain. Aplikasi ini menyediakan berbagai endpoint, seperti POST /api/exams untuk membuat ujian baru, GET /api/exams untuk mengambil daftar ujian yang tersedia, dan POST /api/questions untuk menambah soal ujian baru. Semua endpoint ini diproteksi dengan

autentikasi token berbasis JWT untuk memastikan bahwa hanya pengguna yang terautentikasi yang dapat mengaksesnya.

Pada bagian Controller, setiap controller bertanggung jawab untuk menangani permintaan HTTP dan mengelola alur aplikasi. Misalnya, AuthController menangani proses registrasi dan login pengguna, serta pembuatan token JWT, sementara ExamController mengelola pengambilan dan pembuatan ujian. Controller lainnya, seperti QuestionController, menangani pengelolaan soal ujian yang akan digunakan dalam ujian.

Untuk pengelolaan autentikasi, aplikasi ini menggunakan JWT untuk memastikan bahwa hanya pengguna yang terautentikasi yang dapat mengakses endpoint yang dilindungi. Setelah pengguna berhasil login, mereka akan menerima token yang harus disertakan dalam header permintaan untuk mengakses endpoint yang membutuhkan otorisasi. Token ini divalidasi di setiap permintaan untuk memastikan bahwa pengguna yang mengakses sistem adalah pengguna yang sah.

Selain itu, aplikasi ini juga mendukung pengelolaan hasil ujian. Setelah ujian selesai dilakukan oleh mahasiswa, hasil ujian akan langsung dikalkulasi nilainya dan disimpan dalam entitas Result, yang mencatat skor pengguna, ID ujian, ID pengguna, dan waktu penyelesaian ujian. Hasil ujian ini dapat diakses oleh pengguna yang telah mengikuti ujian tersebut untuk melihat skor mereka.

Untuk mempermudah pengujian dan penggunaan API, dokumentasi API disediakan melalui Swagger UI. Swagger UI secara otomatis mendokumentasikan semua endpoint yang ada dalam aplikasi dan memungkinkan pengembang untuk menguji setiap endpoint secara langsung melalui antarmuka web. Dokumentasi ini mempermudah pengguna dan pengembang dalam memahami cara menggunakan layanan yang disediakan oleh aplikasi.

Dengan penerapan pola desain MVC dan penggunaan JWT untuk autentikasi, aplikasi Manajemen Ujian Online & Bank Soal ini dirancang agar mudah dikembangkan, dipelihara, dan digunakan. Aplikasi ini memberikan solusi yang efisien untuk pengelolaan ujian dan soal ujian di Polstat-STIS, serta memenuhi persyaratan ujian tengah semester yang mencakup pembuatan layanan web berbasis RESTful dengan autentikasi dan dokumentasi yang baik.

5. Struktur Proyek dan Alur Kerja Umum

Struktur proyek ini dibagi menjadi beberapa modul dan paket utama, yang meliputi entitas (entities), repositori (repositories), layanan (services), pengontrol (controllers), serta komponen terkait autentikasi JWT. Struktur direktori proyek adalah sebagai berikut:

```
src/main/java/com/polstatstis/digiexam
├── config/
│   └── SecurityConfig.java
├── controller/
│   ├── AuthController.java
│   ├── ExamController.java
│   ├── ExamResultController.java
│   ├── QuestionController.java
│   └── UserController.java
├── dto/
│   ├── AnswerDTO.java
│   ├── ChangePasswordDTO.java
│   ├── ExamDTO.java
│   ├── ExamResultDTO.java
│   ├── ExamSubmissionDTO.java
│   ├── QuestionDTO.java
│   ├── UserDTO.java
│   ├── UserLogin.java
│   └── UserRegistrationDTO.java
├── entity/
│   ├── Exam.java
│   ├── ExamResult.java
│   ├── Question.java
│   └── User.java
├── exception/
│   ├── ExamNotFoundException.java
│   ├── ExamResultNotFoundException.java
│   ├── JwtAuthenticationException.java
│   ├── QuestionNotFoundException.java
│   └── UserNotFoundExceptionHandler.java
├── mapper/
│   ├── ExamMapper.java
│   ├── ExamResultMapper.java
│   ├── QuestionMapper.java
│   ├── UserLoginMapper.java
│   └── UserMapper.java
├── repository/
│   ├── ExamRepository.java
│   ├── ExamResultRepository.java
│   ├── QuestionRepository.java
│   └── UserRepository.java
├── security/
│   ├── CustomUserDetails.java
│   ├── JwtAuthenticationEntryPoint.java
│   ├── JwtAuthenticationFilter.java
│   └── JwtService.java
├── service/
│   ├── AuthService.java
│   ├── ExamResultService.java
│   ├── ExamService.java
│   ├── QuestionService.java
│   ├── UsersDetailsServiceImpl.java
│   └── UserService.java
└── DIgiexamApplication.java
```

Setiap kali pengguna melakukan permintaan HTTP (seperti **POST**, **GET**, **PUT**, atau **DELETE**) melalui aplikasi ini, request tersebut akan diproses melalui beberapa lapisan dalam aplikasi sebelum memberikan respons kembali kepada pengguna. Alur kerja umum ketika endpoint dipanggil adalah sebagai berikut:

1. **Pengguna mengirimkan request HTTP** (misalnya, melakukan registrasi, login, atau membuat ujian).
2. **Controller menerima request** dan memanggil **service layer** untuk melakukan logika bisnis.
3. **Service layer** mengelola logika aplikasi dan berkomunikasi dengan **repository** untuk mengambil atau menyimpan data ke dalam database.
4. **Repository layer** bertanggung jawab untuk melakukan operasi basis data menggunakan **JPA (Java Persistence API)** atau query yang sesuai.
5. Setelah **repository** mengembalikan hasil, **service layer** mengembalikannya ke **controller**, yang kemudian mengirimkan respons kepada pengguna.

6. Penjelasan Package Utama

6.1. Package Entity

Kelas entitas mendefinisikan struktur data atau model yang akan disimpan di database. Setiap kelas entitas berhubungan langsung dengan tabel dalam database. Entitas ini digunakan untuk mewakili data yang akan dikelola dalam aplikasi. Beberapa entity utama yang diimplementasikan adalah sebagai berikut:

6.1.1. Entity Exam

Kode berikut mendefinisikan entitas Exam yang merepresentasikan ujian dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Menggunakan JPA (Java Persistence API), kelas ini dipetakan ke tabel exams dalam basis data dan menggunakan anotasi Lombok untuk mengurangi boilerplate code. Kunci utama id dihasilkan secara otomatis, sementara title, description, dan date disimpan sebagai kolom yang tidak boleh null. Daftar pertanyaan questions dikelola sebagai koleksi elemen dengan tabel exam_questions, dan referensi ke entitas User yang membuat ujian (createdBy) dihubungkan melalui kolom created_by. Anotasi Lombok seperti `@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`, dan `@Builder` digunakan untuk menghasilkan metode getter, setter, konstruktor, dan builder pattern secara otomatis. Dengan pendekatan ini, kelas Exam mempermudah pengelolaan data ujian dan integrasi dengan sistem basis data.

```

package com.polstatstis.digiexam.entity;

import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDateTime;
import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "exams")
public class Exam {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String description;

    @Column(nullable = false)
    private LocalDateTime date;

    @ElementCollection
    @CollectionTable(name = "exam_questions", joinColumns = @JoinColumn(name = "exam_id"))
    @Column(name = "question_id")
    private List<Long> questions;

    @ManyToOne
    @JoinColumn(name = "created_by", nullable = false)
    private User createdBy;
}

```

6.1.2. Entity ExamResult

Kode ini mendefinisikan entitas ExamResult, yang mewakili hasil ujian dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Entitas ini dipetakan ke tabel exam_results dalam basis data menggunakan JPA dan memanfaatkan anotasi Lombok untuk mengurangi boilerplate code. Kunci utama id dihasilkan secara otomatis, sementara properti user dan exam merepresentasikan referensi ke entitas User dan Exam, masing-masing dihubungkan melalui kolom user_id dan exam_id. Properti score menyimpan nilai ujian yang diperoleh pengguna. Anotasi @Entity dan @Table(name = "exam_results") memastikan pemetaan yang tepat ke tabel basis data.

```

package com.polstatstis.digiexam.entity;

import jakarta.persistence.*;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "exam_results")
public class ExamResult {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne
    @JoinColumn(name = "exam_id", nullable = false)
    private Exam exam;

    @Column(nullable = false)
    private Integer score;
}

```

6.1.3. Entity Question

Kode ini mendefinisikan entitas Question, yang merepresentasikan pertanyaan ujian dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Entitas ini dipetakan ke tabel questions dalam basis data menggunakan JPA, dan memanfaatkan anotasi Lombok untuk mengurangi boilerplate code. Properti id adalah kunci utama yang dihasilkan otomatis. Properti text menyimpan teks pertanyaan, sedangkan options adalah daftar pilihan jawaban yang dipetakan ke tabel question_options. Properti answer menyimpan jawaban yang benar, dan properti createdBy mengacu pada pengguna yang membuat pertanyaan. Anotasi @Entity dan @Table memastikan pemetaan yang tepat ke tabel basis data.

```

package com.polstatstis.digiexam.entity;

import jakarta.persistence.*;
import lombok.*;

import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "questions")

```



```

public class Question {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String text;

    @ElementCollection
    @CollectionTable(
        name = "question_options",
        joinColumns = @JoinColumn(name = "question_id")
    )
    @Column(name = "option_text") // Diubah dari 'option' menjadi 'option_text'
    private List<String> options;

    @Column(nullable = false)
    private String answer;

    @ManyToOne
    @JoinColumn(name = "created_by", nullable = false)
    private User createdBy;
}

```

6.2. Package Repository Interface

Kelas repository berfungsi untuk mengelola operasi database (CRUD: Create, Read, Update, Delete). Repository ini berinteraksi langsung dengan entitas dan menyediakan metode untuk melakukan pencarian, pengambilan, pembaruan, dan penghapusan data dalam database. Beberapa repository utama yang diimplementasikan adalah sebagai berikut:

6.2.1. ExamRepository

```

package com.polstatstis.digiexam.repository;

import com.polstatstis.digiexam.entity.Exam;
import org.springframework.data.jpa.repository.JpaRepository;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;

import java.util.List;

public interface ExamRepository extends JpaRepository<Exam, Long> {
    @Operation(summary = "Find exams created by a user")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found exams created b
y the user",
            content = {@Content(mediaType = "application/json",
                schema = @Schema(implementation = Exam.class))}),
        @ApiResponse(responseCode = "404", description = "No exams found",
            content = @Content)})
    List<Exam> findByCreatedBy(Long userId);
}

```

ExamResultRepository

```
package com.polstatstis.digiexam.repository;

import com.polstatstis.digiexam.entity.ExamResult;
import org.springframework.data.jpa.repository.JpaRepository;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;

import java.util.List;

public interface ExamResultRepository extends JpaRepository<ExamResult, Long> {
    @Operation(summary = "Find exam results by user ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found exam results",
            content = { @Content(mediaType = "application/json",
                schema = @Schema(implementation = ExamResult.class)) })
        ,
        @ApiResponse(responseCode = "404", description = "Exam results not found",
            content = @Content) })

    List<ExamResult> findById(Long userId);

    @Operation(summary = "Find exam results by exam ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found exam results",
            content = { @Content(mediaType = "application/json",
                schema = @Schema(implementation = ExamResult.class)) })
        ,
        @ApiResponse(responseCode = "404", description = "Exam results not found",
            content = @Content) })

    List<ExamResult> findByExamId(Long examId);
}
```

6.2.2. QuestionRepository

```
package com.polstatstis.digiexam.repository;

import com.polstatstis.digiexam.entity.Question;
import org.springframework.data.jpa.repository.JpaRepository;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;

import java.util.List;

public interface QuestionRepository extends JpaRepository<Question, Long> {
    @Operation(summary = "Find questions by user ID")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found questions",
            content = { @Content(mediaType = "application/json",
                schema = @Schema(implementation = Question.class)) })
    },
```

```

        @ApiResponse(responseCode = "404", description = "Questions not found",
            content = @Content)})

    List<Question> findByCreatedById(Long userId);
}

```

6.2.3. UserRepository

```

package com.polstatstis.digiexam.repository;

import com.polstatstis.digiexam.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {

    @Operation(summary = "Find user by email", description = "Returns a single user
by email", tags = { "user" })
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "successful operation"
, content = @Content(schema = @Schema(implementation = User.class))),
        @ApiResponse(responseCode = "404", description = "User not found") })

    Optional<User> findByEmail(String email);
}

```

6.3. Package DTO (Data Transfer Object)

DTO digunakan untuk mentransfer data antara lapisan aplikasi, terutama antara controller dan service. DTO digunakan untuk menghindari pengiriman data entitas secara langsung, yang bisa mengandung data yang tidak relevan atau sensitif. DTO memungkinkan format data yang lebih terstruktur dan ringkas.. Beberapa DTO utama yang diimplementasikan adalah sebagai berikut:

6.3.1. UserDTO

```

package com.polstatstis.digiexam.dto;

import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UserDTO {

    private Long id;
    private String email;
    private String name;
}

```

```

        private String role;
    }

```

6.3.2. ExamDTO

```

package com.polstatstis.digiexam.dto;

import lombok.*;

import java.time.LocalDateTime;
import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ExamDTO {

    private Long id;
    private String title;
    private String description;
    private LocalDateTime date;
    private List<Long> questionIds;
    private Long createdById;
}

```

6.4. Package Mapper

Mapper digunakan untuk mengonversi antara entitas dan DTO. Biasanya, data yang dikirimkan melalui API dalam bentuk DTO perlu diubah kembali ke bentuk entitas untuk disimpan di database, dan sebaliknya, entitas perlu diubah ke DTO untuk dikirim melalui API. Beberapa mapper utama yang diimplementasikan adalah sebagai berikut:

6.4.1. UserMapper

```

package com.polstatstis.digiexam.mapper;

import com.polstatstis.digiexam.dto.UserDTO;
import com.polstatstis.digiexam.entity.User;

public class UserMapper {

    public static UserDTO toDTO(User user) {
        return UserDTO.builder()
            .id(user.getId())
            .email(user.getEmail())
            .name(user.getName())
            .role(user.getRole().name())
            .build();
    }

    public static User toEntity(UserDTO userDTO) {
        return User.builder()
            .id(userDTO.getId())
            .email(userDTO.getEmail())
            .name(userDTO.getName())

```

```

        .role(User.Role.valueOf(userDTO.getRole()))
        .build();
    }
}

```

6.4.2. ExamMapper

```

package com.polstatstis.digiexam.mapper;

import com.polstatstis.digiexam.dto.ExamDTO;
import com.polstatstis.digiexam.entity.Exam;

public class ExamMapper {

    public static ExamDTO toDTO(Exam exam) {
        return ExamDTO.builder()
            .id(exam.getId())
            .title(exam.getTitle())
            .description(exam.getDescription())
            .date(exam.getDate())
            .questionIds(exam.getQuestions())
            .createdById(exam.getCreatedBy().getId())
            .build();
    }

    public static Exam toEntity(ExamDTO examDTO) {
        return Exam.builder()
            .id(examDTO.getId())
            .title(examDTO.getTitle())
            .description(examDTO.getDescription())
            .date(examDTO.getDate())
            .questions(examDTO.getQuestionIds())
            .build();
    }
}

```

6.5. Package Service

Kelas service berfungsi untuk mengelola logika bisnis aplikasi. Service ini mengatur bagaimana data diproses, dikelola, dan dikombinasikan dari berbagai komponen (repository, entitas, DTO, dan lainnya). Kelas service adalah tempat di mana kebanyakan logika aplikasi berada, seperti validasi, perhitungan, atau pengelolaan alur kerja. Beberapa service utama yang diimplementasikan adalah sebagai berikut

6.5.1. AuthService

Kode ini mendefinisikan layanan AuthService untuk menangani proses registrasi dan login pengguna dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Menggunakan JPA untuk akses data, kelas ini memanfaatkan UserRepository, PasswordEncoder, dan JwtService. Metode register memvalidasi email pengguna, mengenkripsi kata sandi, dan menyimpan pengguna baru, serta mencatat keberhasilan registrasi. Metode login memeriksa keberadaan pengguna berdasarkan

email, memvalidasi kata sandi, dan menghasilkan token JWT jika berhasil, serta mencatat keberhasilan atau kegagalan login menggunakan logger dari SLF4J. Anotasi Lombok digunakan untuk mengurangi boilerplate code.

```
package com.polstatstis.digiexam.service;

import com.polstatstis.digiexam.dto.UserLoginDTO;
import com.polstatstis.digiexam.dto.UserRegistrationDTO;
import com.polstatstis.digiexam.dto.UserDTO;
import com.polstatstis.digiexam.entity.User;
import com.polstatstis.digiexam.exception.UserNotFoundException;
import com.polstatstis.digiexam.mapper.UserMapper;
import com.polstatstis.digiexam.repository.UserRepository;
import com.polstatstis.digiexam.security.JwtService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Service
@RequiredArgsConstructor
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private static final Logger logger = LoggerFactory.getLogger(AuthService.class);

    ;

    public UserDTO register(UserRegistrationDTO registrationDTO) {
        // Validasi email sudah terdaftar
        if (userRepository.existsByEmail(registrationDTO.getEmail().trim().toLowerCase())) {
            throw new IllegalArgumentException("Email already registered");
        }

        User user = User.builder()
            .email(registrationDTO.getEmail().trim().toLowerCase())
            .password(passwordEncoder.encode(registrationDTO.getPassword()))
            .name(registrationDTO.getName())
            .role(User.Role.USER)
            .build();

        User savedUser = userRepository.save(user);
        logger.info("User registered successfully: {}", savedUser.getEmail());
        return UserMapper.toDTO(savedUser);
    }

    public String login(UserLoginDTO loginDTO) {
        String email = loginDTO.getEmail().trim().toLowerCase();

        User user = userRepository.findByEmail(email)
            .orElseThrow(() -> {
                logger.error("Login attempt failed: User not found with email: {}", email);
                return new UserNotFoundException("User not found");
            });

        if (!passwordEncoder.matches(loginDTO.getPassword(), user.getPassword())) {
            logger.error("Login attempt failed: Invalid password for user: {}", email);
            throw new IllegalArgumentException("Invalid password");
        }
    }
}
```

```

    }

    String token = jwtService.generateToken(user.getEmail());
    logger.info("User logged in successfully: {}", email);
    return token;
}
}

```

6.5.2. UserService

Kode ini mendefinisikan layanan UserService yang menangani berbagai operasi terkait pengguna dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Layanan ini menggunakan JPA untuk akses data, dengan UserRepository dan PasswordEncoder sebagai dependensi utama. Metode getUserProfile dan updateUserProfile mengelola pengambilan dan pembaruan profil pengguna, sementara changeUserRole memungkinkan perubahan peran pengguna. Metode changePassword memverifikasi dan memperbarui kata sandi pengguna, dan deleteUser menghapus pengguna dari basis data. Kelas ini menggunakan anotasi Lombok untuk mengurangi boilerplate code.

```

package com.polstatstis.digiexam.service;

import com.polstatstis.digiexam.dto.ChangePasswordDTO;
import com.polstatstis.digiexam.dto.UserDTO;
import com.polstatstis.digiexam.entity.User;
import com.polstatstis.digiexam.mapper.UserMapper;
import com.polstatstis.digiexam.repository.UserRepository;
import com.polstatstis.digiexam.exception.UserNotFoundException;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UserDTO getUserProfile(Long userId) {
        User user = userRepository.findById(userId)
            .orElseThrow(() -> new UserNotFoundException("User not found"));
        return UserMapper.toDTO(user);
    }

    public UserDTO updateUserProfile(Long userId, UserDTO userDTO) {
        User user = userRepository.findById(userId)
            .orElseThrow(() -> new UserNotFoundException("User not found"));
        user.setName(userDTO.getName());
        user.setEmail(userDTO.getEmail());
        userRepository.save(user);
        return UserMapper.toDTO(user);
    }
}

```

```

public UserDTO changeUserRole(Long id, UserDTO userDTO) {
    User user = userRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("User not found"));

    // Konversi String ke enum Role
    User.Role newRole;
    try {
        newRole = User.Role.valueOf(userDTO.getRole().toUpperCase());
    } catch (IllegalArgumentException e) {
        throw new RuntimeException("Invalid role: " + userDTO.getRole());
    }

    // Ubah role pengguna
    user.setRole(newRole);

    // Simpan perubahan ke database
    userRepository.save(user);

    // Kembalikan userDTO yang telah diperbarui
    userDTO.setId(user.getId());
    return userDTO;
}

public void changePassword(Long userId, ChangePasswordDTO changePasswordDTO) {
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new UserNotFoundException("User not found"));

    if (!passwordEncoder.matches(changePasswordDTO.getOldPassword(), user.getPassword())) {
        throw new IllegalArgumentException("Invalid old password");
    }

    user.setPassword(passwordEncoder.encode(changePasswordDTO.getNewPassword()));
    userRepository.save(user);
}

public void deleteUser(Long userId) {
    if (!userRepository.existsById(userId)) {
        throw new UserNotFoundException("User not found");
    }
    userRepository.deleteById(userId);
}
}

```

6.5.3. ExamService

Kode ini mendefinisikan layanan ExamService yang mengelola ujian dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Layanan ini menggunakan JPA untuk akses data dengan ExamRepository, QuestionRepository, UserRepository, dan ExamResultRepository sebagai dependensi utama. Metode createExam membuat ujian baru, getAllExams mengambil semua ujian, getExamById mengambil ujian berdasarkan ID, dan deleteExam menghapus ujian berdasarkan ID. Metode submitExam memproses pengiriman ujian oleh pengguna, menghitung skor berdasarkan jawaban yang diberikan, dan menyimpan hasil ujian.

Kelas ini menggunakan anotasi Lombok untuk mengurangi boilerplate code dan menangani exception secara tepat.

```
package com.polstatstis.digiexam.service;

import com.polstatstis.digiexam.dto.ExamDTO;
import com.polstatstis.digiexam.dto.ExamResultDTO;
import com.polstatstis.digiexam.dto.ExamSubmissionDTO;
import com.polstatstis.digiexam.entity.Exam;
import com.polstatstis.digiexam.entity.User;
import com.polstatstis.digiexam.exception.UserNotFoundException;
import com.polstatstis.digiexam.mapper.ExamMapper;
import com.polstatstis.digiexam.repository.ExamRepository;
import com.polstatstis.digiexam.exception.ExamNotFoundException;
import com.polstatstis.digiexam.repository.UserRepository;
import com.polstatstis.digiexam.repository.ExamResultRepository;
import com.polstatstis.digiexam.repository.QuestionRepository;
import com.polstatstis.digiexam.entity.ExamResult;
import com.polstatstis.digiexam.entity.Question;
import com.polstatstis.digiexam.dto.AnswerDTO;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class ExamService {

    private final ExamRepository examRepository;
    private final QuestionRepository questionRepository;
    private final UserRepository userRepository;
    private final ExamResultRepository examResultRepository;

    public ExamDTO createExam(ExamDTO examDTO, Long userId) {
        User user = userRepository.findById(userId)
            .orElseThrow(() -> new UserNotFoundException("User not found"));

        Exam exam = ExamMapper.toEntity(examDTO);
        exam.setCreatedBy(user); // Set createdBy sebelum menyimpan

        examRepository.save(exam);
        return ExamMapper.toDTO(exam);
    }

    public List<ExamDTO> getAllExams() {
        return examRepository.findAll().stream()
            .map(ExamMapper::toDTO)
            .collect(Collectors.toList());
    }

    public ExamDTO getExamById(Long examId) {
        Exam exam = examRepository.findById(examId)
            .orElseThrow(() -> new ExamNotFoundException("Exam not found"));
        return ExamMapper.toDTO(exam);
    }

    public void deleteExam(Long examId) {
        if (!examRepository.existsById(examId)) {
            throw new ExamNotFoundException("Exam not found");
        }
    }
}
```

```

        examRepository.deleteById(examId);
    }

    public ExamResultDTO submitExam(Long examId, ExamSubmissionDTO submission) {
        User user = userRepository.findById(submission.getUserId())
            .orElseThrow(() -> new UserNotFoundException("User not found"));
        Exam exam = examRepository.findById(examId)
            .orElseThrow(() -> new ExamNotFoundException("Exam not found"));

        Map<Long, String> correctAnswers = questionRepository.findAllById(
            submission.getAnswers().stream()
                .map(AnswerDTO::getQuestionId)
                .collect(Collectors.toList())
        ).stream().collect(Collectors.toMap(Question::getId, Question::getAnswer));

        int score = 0;
        for (AnswerDTO answer : submission.getAnswers()) {
            if (correctAnswers.get(answer.getQuestionId()).equals(answer.getAnswer(
))) {
                score++;
            }
        }

        ExamResult examResult = ExamResult.builder()
            .user(user)
            .exam(exam)
            .score(score)
            .build();

        examResultRepository.save(examResult);
        return new ExamResultDTO(examResult.getId(), user.getId(), exam.getId(), sc
ore);
    }
}

```

6.6. Package Controller

Paket controller dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS bertanggung jawab untuk menangani permintaan HTTP dan mengelola interaksi antara klien dan server. Paket ini mencakup pengontrol utama seperti AuthController, UserController, dan ExamController, yang masing-masing mengelola aspek autentikasi, pengguna, dan ujian. AuthController menyediakan endpoint untuk registrasi dan login pengguna, UserController menangani operasi terkait profil dan manajemen pengguna, sedangkan ExamController mengelola pembuatan, pengambilan, penghapusan, dan pengiriman ujian. Anotasi `@PreAuthorize` digunakan dalam pengontrol ini untuk memastikan bahwa akses ke endpoint tertentu hanya diberikan kepada pengguna dengan peran yang sesuai, seperti ADMIN, LECTURER, atau STUDENT, guna meningkatkan keamanan aplikasi. Secara keseluruhan, paket controller ini mengatur alur data dan operasi bisnis, serta memastikan keamanan melalui pengelolaan peran dan otorisasi.

Beberapa controller utama yang diimplementasikan adalah sebagai berikut:

6.6.1. AuthController

```
package com.polstatstis.digiexam.controller;

import com.polstatstis.digiexam.dto.UserDTO;
import com.polstatstis.digiexam.dto.UserLoginDTO;
import com.polstatstis.digiexam.dto.UserRegistrationDTO;
import com.polstatstis.digiexam.service.AuthService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {

    private final AuthService authService;

    @Operation(summary = "registrasi pengguna.")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "User registered successfully", content = {
            @Content(mediaType = "application/json", schema = @Schema(implementation = Page.class))}),
        @ApiResponse(responseCode = "400", description = "Invalid input", content = @Content)})
    @PostMapping("/register")
    public ResponseEntity<UserDTO> register(@RequestBody UserRegistrationDTO registrationDTO) {
        UserDTO user = authService.register(registrationDTO);
        return ResponseEntity.ok(user);
    }

    @Operation(summary = "User login untuk dapat access token.")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Email and access token", content = {
            @Content(mediaType = "application/json", schema = @Schema(implementation = Page.class))}),
        @ApiResponse(responseCode = "401", description = "Invalid credentials", content = @Content)})
    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestBody UserLoginDTO loginDTO) {
        String token = authService.login(loginDTO);
        return ResponseEntity.ok(token);
    }
}
```

6.6.2. UserController:

```
package com.polstatstis.digiexam.controller;
```

```

import com.polstatstis.digiexam.dto.ChangePasswordDTO;
import com.polstatstis.digiexam.dto.UserDTO;
import com.polstatstis.digiexam.service.UserService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    @Operation(summary = "Mendapatkan profil pengguna")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Found the user",
            content = { @Content(mediaType = "application/json",
                schema = @Schema(implementation = UserDTO.class)) }},
        @ApiResponse(responseCode = "404", description = "User not found",
            content = @Content) })

    @GetMapping("/{id}")
    public ResponseEntity<UserDTO> getUserProfile(@PathVariable Long id) {
        UserDTO user = userService.getUserProfile(id);
        return ResponseEntity.ok(user);
    }

    @Operation(summary = "Update profil pengguna")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "User profile updated",
            content = { @Content(mediaType = "application/json",
                schema = @Schema(implementation = UserDTO.class)) }},
        @ApiResponse(responseCode = "404", description = "User not found",
            content = @Content) })

    @PreAuthorize("#id == authentication.principal.id or hasRole('ADMIN')")
    @PutMapping("/{id}")
    public ResponseEntity<UserDTO> updateUserProfile(@PathVariable Long id, @RequestBody UserDTO userDTO) {
        UserDTO updatedUser = userService.updateUserProfile(id, userDTO);
        return ResponseEntity.ok(updatedUser);
    }

    @Operation(summary = "Mengganti password oleh pengguna")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Password changed successfully",
            content = { @Content(mediaType = "application/json",
                schema = @Schema(implementation = UserDTO.class)) }},
        @ApiResponse(responseCode = "400", description = "Invalid old password",
            content = @Content),
        @ApiResponse(responseCode = "404", description = "User not found",
            content = @Content) })

    @PreAuthorize("#id == authentication.principal.id")
    @PutMapping("/{id}/change-password")

```

```

        public ResponseEntity<Void> changePassword(@PathVariable Long id, @RequestBody
ody ChangePasswordDTO changePasswordDTO) {
            userService.changePassword(id, changePasswordDTO);
            return ResponseEntity.ok().build();
        }

        @Operation(summary = "Delete user oleh admin")
        @ApiResponses(value = {
            @ApiResponse(responseCode = "204", description = "User deleted",
                content = @Content),
            @ApiResponse(responseCode = "404", description = "User not found",
                content = @Content) })

        @PreAuthorize("hasRole('ADMIN')")
        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
            userService.deleteUser(id);
            return ResponseEntity.noContent().build();
        }

        @Operation(summary = "Mengubah role pengguna oleh admin")
        @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "User role changed
successfully",
                content = { @Content(mediaType = "application/json",
                    schema = @Schema(implementation = UserDTO.class)) }
        ),
            @ApiResponse(responseCode = "404", description = "User not found",
                content = @Content) })

        @PreAuthorize("hasRole('ADMIN')")
        @PutMapping("/{id}/role")
        public ResponseEntity<UserDTO> changeUserRole(@PathVariable Long id, @Reque
stBody UserDTO userDTO) {
            UserDTO updatedUser = userService.changeUserRole(id, userDTO);
            return ResponseEntity.ok(updatedUser);
        }
    }
}

```

6.6.3. ExamController

```

package com.polstatstis.digiexam.controller;

import com.polstatstis.digiexam.dto.ExamDTO;
import com.polstatstis.digiexam.dto.ExamResultDTO;
import com.polstatstis.digiexam.dto.ExamSubmissionDTO;
import com.polstatstis.digiexam.service.ExamService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/exams")
@RequiredArgsConstructor
public class ExamController {

```

```

private final ExamService examService;

@Operation(summary = "buat ujian baru oleh dosen")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Exam created successfully"),
    @ApiResponse(responseCode = "400", description = "Invalid input"),
    @ApiResponse(responseCode = "500", description = "Server error")
})

// preauthorize only admin and lecturer can create exam
@PreAuthorize("hasRole('ADMIN') or hasRole('LECTURER')")
@PostMapping
public ResponseEntity<ExamDTO> createExam(@RequestBody ExamDTO examDTO, @RequestParam Long userId) {
    ExamDTO createdExam = examService.createExam(examDTO, userId);
    return ResponseEntity.ok(createdExam);
}

@Operation(summary = "mendapatkan semua ujian")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Exams retrieved successfully"),
    @ApiResponse(responseCode = "500", description = "Server error")
})

// preauthorize only admin and lecturer can get all exams
@PreAuthorize("hasRole('ADMIN') or hasRole('LECTURER')")
@GetMapping
public ResponseEntity<List<ExamDTO>> getAllExams() {
    List<ExamDTO> exams = examService.getAllExams();
    return ResponseEntity.ok(exams);
}

@Operation(summary = "mendapatkan ujian berdasarkan id")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Exam retrieved successfully"),
    @ApiResponse(responseCode = "404", description = "Exam not found"),
    @ApiResponse(responseCode = "500", description = "Server error")
})

@GetMapping("/{id}")
public ResponseEntity<ExamDTO> getExamById(@PathVariable Long id) {
    ExamDTO exam = examService.getExamById(id);
    return ResponseEntity.ok(exam);
}

@Operation(summary = "menghapus ujian oleh dosen")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Exam updated successfully"),
    @ApiResponse(responseCode = "404", description = "Exam not found"),
    @ApiResponse(responseCode = "500", description = "Server error")
})

// preauthorize only admin and lecturer can delete exam
@PreAuthorize("hasRole('ADMIN') or hasRole('LECTURER')")
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteExam(@PathVariable Long id) {
    examService.deleteExam(id);
    return ResponseEntity.noContent().build();
}

@Operation(summary = "Submit ujian oleh mahasiswa")

```

```

        @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Exam submitted successfully",
                content = { @Content(mediaType = "application/json",
                    schema = @Schema(implementation = ExamResultDTO.class)) }),
            @ApiResponse(responseCode = "404", description = "Exam or User not found",
                content = @Content)
        })

        @PreAuthorize("hasRole('STUDENT')")
        @PostMapping("/{id}/submit")
        public ResponseEntity<ExamResultDTO> submitExam(@PathVariable Long id, @RequestBody ExamSubmissionDTO submission) {
            ExamResultDTO result = examService.submitExam(id, submission);
            return ResponseEntity.ok(result);
        }
    }
}

```

6.7. Package Security dan Authentication

Kelas-kelas ini mengelola keamanan aplikasi, termasuk otentikasi dan otorisasi pengguna. Mereka bertanggung jawab untuk memverifikasi identitas pengguna dan memastikan bahwa hanya pengguna yang berwenang yang dapat mengakses sumber daya tertentu.. Beberapa authentication utama dan securityconfig yang diimplementasikan adalah sebagai berikut:

6.7.1. SecurityConfig

Kode ini mendefinisikan konfigurasi keamanan SecurityConfig untuk sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Menggunakan Spring Security, konfigurasi ini mengatur filter autentikasi JWT (JwtAuthenticationFilter), layanan detail pengguna (UserDetailsServiceImpl), dan manajemen sesi stateless. Metode securityFilterChain mengonfigurasi aturan akses HTTP dan menambahkan filter JWT. Metode authenticationProvider mengatur penyedia autentikasi berbasis DAO dengan encoder kata sandi BCrypt. Metode authenticationManager mengonfigurasi manajer autentikasi dengan layanan detail pengguna dan encoder kata sandi. Anotasi seperti @Configuration, @EnableWebSecurity, dan @EnableGlobalMethodSecurity digunakan untuk mengaktifkan fitur keamanan global dan method-level.

```

package com.polstatstis.digiexam.config;

import com.polstatstis.digiexam.security.JwtAuthenticationFilter;
import com.polstatstis.digiexam.service.UserDetailsServiceImpl;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
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.method.configuration.EnableGlobalMethodSecurity;
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.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

```

```

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserDetailsServiceImpl userDetailsService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/docs/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            )
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {
        return http.getSharedObject(AuthenticationManagerBuilder.class)
            .userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder())
            .and()
            .build();
    }
}

```



```
}  
}
```

6.7.1. JwtService

Kode ini mendefinisikan layanan JwtService untuk mengelola token JWT dalam sistem Manajemen Ujian Online & Bank Soal di Polstat-STIS. Menggunakan Jwts dari pustaka io.jsonwebtoken, layanan ini menghasilkan, memvalidasi, dan mengekstrak informasi dari token JWT. Properti secretKey dan expiration diambil dari file konfigurasi dan diinisialisasi dengan encoding Base64. Metode generateToken menghasilkan token baru, extractUsername mengekstrak nama pengguna dari token, dan validateToken memeriksa validitas token berdasarkan masa berlakunya dan kecocokan dengan nama pengguna yang diberikan. Anotasi @Service menandakan bahwa kelas ini adalah komponen layanan Spring.

```
package com.polstatstis.digiexam.security;  
  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import java.util.Base64;  
import java.util.Date;  
  
@Service  
public class JwtService {  
    @Value("${jwt.secret}")  
    private String secretKey;  
  
    @Value("${jwt.expiration}")  
    private Long expiration;  
  
    @PostConstruct  
    protected void init() {  
        // Secret key harus di-encode dengan Base64  
        secretKey = Base64.getEncoder().encodeToString(secretKey.getBytes());  
    }  
  
    public String generateToken(String username) {  
        Date now = new Date();  
        Date validity = new Date(now.getTime() + expiration);  
  
        return Jwts.builder()  
            .setSubject(username)  
            .setIssuedAt(now)  
            .setExpiration(validity)  
            .signWith(SignatureAlgorithm.HS256, secretKey)  
            .compact();  
    }  
  
    public String extractUsername(String token) {  
        return extractAllClaims(token).getSubject();  
    }  
}
```

```

    }

    public boolean validateToken(String token, String username) {
        try {
            String extractedUsername = extractUsername(token);
            return extractedUsername.equals(username) && !isTokenExpired(token);
        } catch (Exception e) {
            return false;
        }
    }

    private boolean isTokenExpired(String token) {
        return extractAllClaims(token).getExpiration().before(new Date());
    }

    private Claims extractAllClaims(String token) {
        return Jwts.parser()
            .setSigningKey(secretKey)
            .parseClaimsJws(token)
            .getBody();
    }
}

```

6.8. Package Exception Handling

Kelas-kelas pengecualian ini menangani situasi kesalahan dalam aplikasi. Mereka digunakan untuk memberikan umpan balik yang berguna kepada pengguna atau pengembang jika terjadi kesalahan seperti data tidak ditemukan atau masalah otentikasi. Beberapa exception handling yang diimplementasikan sama persis, hanya berbeda pada penamaan kelasnya, seperti berikut:

```

package com.polstatstis.digiexam.exception;

public class JwtAuthenticationException extends RuntimeException {
    public JwtAuthenticationException(String message) {
        super(message);
    }
}

```

6.9. Kelas Main Application (DigiexamApplication.java)

Kelas ini adalah titik masuk utama untuk aplikasi Spring Boot. Ini digunakan untuk menjalankan aplikasi dan mengonfigurasi komponen-komponen yang diperlukan agar aplikasi dapat berjalan. Kelas main yang diimplementasikan adalah sebagai berikut:

```

package com.polstatstis.digiexam;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DigiexamApplication {

```

```
public static void main(String[] args) {  
    SpringApplication.run(DigiexamApplication.class, args);  
}  
  
}
```

7. Dokumentasi Endpoint

Dengan dibantu dokumentasi endpoint lengkap dari Swagger-UI yang anotasinya sudah saya sesuaikan secara manual pada program, saya dapat dengan mudah menggunakannya sebagai boilerplate dalam pengujian. Berikut adalah seluruh endpoint.

7.1. Auth-Controller

POST

/api/auth/register

registrasi pengguna.

⌵

Parameters

Try it out

No parameters

Request body

required

application/json

⌵

Example Value

Schema

```
{
  "email": "string",
  "password": "string",
  "name": "string"
}
```

Responses		
Code	Description	Links
200	User registered successfully	No links
<div><div>Media type</div><div>application/json</div><div>⌵</div></div> <div>Controls Accept header.</div> <div><div>Example Value</div><div>Schema</div></div> <div><pre>{ "totalPages": 0, "size": 0, "content": [{}], "number": 0, "sort": { "empty": true, "sorted": true, "unsorted": true }, "numberOfElements": 0, "first": true, "last": true, "pageable": { "offset": 0, "sort": { "empty": true, "sorted": true, "unsorted": true }, "paged": true, "pageNumber": 0, "pageSize": 0, "unpaged": true }, "empty": true }</pre></div>		
400	Invalid input	No links

POST

/api/auth/login

User login untuk dapat access token.

^

Parameters

Try it out

No parameters

Request body

required

application/json

Example Value

Schema

```
{
  "email": "string",
  "password": "string"
}
```

Responses

Code	Description	Links
200	Email and access token	No links
	<div><div>Media type</div><div>application/json</div><div></div></div> <div>Controls Accept header.</div> <div><div>Example Value</div><div>Schema</div></div> <div><pre>{ "totalPages": 0, "size": 0, "content": [{}], "number": 0, "sort": { "empty": true, "sorted": true, "unsorted": true }, "numberOfElements": 0, "first": true, "last": true, "pageable": { "offset": 0, "sort": { "empty": true, "sorted": true, "unsorted": true }, "paged": true, "pageNumber": 0, "pageSize": 0, "unpaged": true }, "empty": true }</pre></div>	
401	Invalid credentials	No links

7.2. User-Controller

GET

/api/users/{id}

Mendapatkan profil pengguna

⌵

Parameters

Try it out

Name	Description
id * required integer(\$int64) (path)	<input type="text" value="id"/>

Responses

Code	Description	Links
200	<div>Found the user</div> <div>Media type</div> <div><div>application/json</div><div>⌵</div></div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "email": "string", "name": "string", "role": "string" }</pre></div>	No links
404	User not found	No links

PUT

/api/users/{id}

Update profil pengguna

⌵

Parameters

Try it out

Name	Description
id * required	
integer(\$int64)	id
(path)	

Request body required

application/json

Example Value | Schema

```
{  "id": 0,  "email": "string",  "name": "string",  "role": "string"}
```

Responses

Code	Description	Links
200	User profile updated	No links
	Media type <div>application/json</div> Controls Accept header.	
	Example Value Schema	
	<pre>{ "id": 0, "email": "string", "name": "string", "role": "string"}</pre>	
404	User not found	No links

DELETE

/api/users/{id}

Delete user oleh admin

⌵

Parameters

Try it out

Name	Description
id * required	
integer(\$int64)	id
(path)	

Responses		
Code	Description	Links
204	User deleted	No links
404	User not found	No links

PUT

/api/users/{id}/role

Mengubah role pengguna oleh admin

^

Parameters

Try it out

Name	Description
id * required	
integer(\$int64)	id
(path)	

Request body **required**

application/json

Example Value | Schema

```
{  "id": 0,  "email": "string",  "name": "string",  "role": "string"}
```

Responses		
Code	Description	Links
200	User role changed successfully	No links
<div>Media type<div>application/json</div></div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "email": "string", "name": "string", "role": "string"}</pre></div>		
404	User not found	No links

PUT

/api/users/{id}/change-password

Mengganti password oleh pengguna

Try it out

Parameters

Name	Description
id * required integer(\$int64) (path)	<input type="text" value="id"/>

Request body required

application/json

Example Value | Schema

```

{
  "oldPassword": "string",
  "newPassword": "string"
}

```

Responses

Code	Description	Links
200	Password changed successfully <div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <pre> { "id": 0, "email": "string", "name": "string", "role": "string" } </pre>	No links
400	Invalid old password	No links
404	User not found	No links

7.3. Question-Controller

GET

/api/questions

Mendapatkan semua pertanyaan

Try it out

Parameters

No parameters

Responses

Code	Description	Links
------	-------------	-------

200	List of questions	No links
-----	-------------------	----------

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "id": 0,
  "text": "string",
  "options": [
    "string"
  ],
  "answer": "string",
  "createdById": 0
}
```

404	No questions found	No links
-----	--------------------	----------

Media type

application/hal+json

Example Value | Schema

```
[
  {
    "id": 0,
    "text": "string",
    "options": [
      "string"
    ],
    "answer": "string",
    "createdById": 0
  }
]
```

POST /api/questions Membuat pertanyaan baru

Parameters

Try it out

Name	Description
------	-------------

userId * required

integer(\$int64)

(query)

userId

Request body required

application/json

Example Value | Schema

```
{
  "id": 0,
  "text": "string",
  "options": [
    "string"
  ],
  "answer": "string",
  "createdById": 0
}
```

Responses

Code	Description	Links
200	<div>Question created</div> <div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "text": "string", "options": ["string"], "answer": "string", "createdById": 0 }</pre></div>	No links
400	<div>Invalid input</div> <div>Media type</div> <div>application/hal+json</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "text": "string", "options": ["string"], "answer": "string", "createdById": 0 }</pre></div>	No links
409	<div>Question already exists</div> <div>Media type</div> <div>application/hal+json</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "text": "string", "options": ["string"], "answer": "string", "createdById": 0 }</pre></div>	No links

Parameters

Try it out

Name	Description
id * required integer(int64) (path)	<div>id</div>

Responses

Code	Description	Links
200	Question found <div>Media type application/json Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "text": "string", "options": ["string"], "answer": "string", "createdById": 0 }</pre></div>	No links
404	Question not found <div>Media type application/hal+json</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "text": "string", "options": ["string"], "answer": "string", "createdById": 0 }</pre></div>	No links

DELETE

/api/questions/{id}

Menghapus pertanyaan

^

📄

Parameters

Try it out

Name	Description
id <small>* required</small> integer(\$int64) (path)	<input type="text" value="id"/>

Responses

Code	Description	Links
204	Question deleted	No links
404	Question not found	No links

7.4. Exam-Controller

GET

/api/exams

mendapatkan semua ujian

^

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	<div>Exams retrieved successfully</div> <div><div>Media type</div><div>application/hal+json</div><div>Controls Accept header.</div><div>Example Value Schema</div><div><pre>[{ "id": 0, "title": "string", "description": "string", "date": "2024-11-15T23:08:48.088Z", "questionIds": [0], "createdById": 0 }]</pre></div></div>	No links
500	<div>Server error</div> <div><div>Media type</div><div>application/hal+json</div><div>Example Value Schema</div><div><pre>[{ "id": 0, "title": "string", "description": "string", "date": "2024-11-15T23:08:48.089Z", "questionIds": [0], "createdById": 0 }]</pre></div></div>	No links

Parameters

Try it out

Name	Description
userId * required integer(\$int64) (query)	<input type="text" value="userId"/>

Request body required

application/json ▾

Example Value | Schema

```
{
  "id": 0,
  "title": "string",
  "description": "string",
  "date": "2024-11-15T23:09:46.846Z",
  "questionIds": [
    0
  ],
  "createdById": 0
}
```

Responses		
Code	Description	Links
200	Exam created successfully	No links
<div><div>Media type</div><div>application/hal+json ▾</div><div>Controls Accept header.</div><div><div>Example Value Schema</div><div><pre>{ "id": 0, "title": "string", "description": "string", "date": "2024-11-15T23:09:46.846Z", "questionIds": [0], "createdById": 0 }</pre></div></div></div>		

400

No links

Invalid input

Media type

application/hal+json ▾

[Example Value](#) | [Schema](#)

```
{
  "id": 0,
  "title": "string",
  "description": "string",
  "date": "2024-11-15T23:09:46.848Z",
  "questionIds": [
    0
  ],
  "createdById": 0
}
```

500

No links

Server error

Media type

application/hal+json ▾

[Example Value](#) | [Schema](#)

```
{
  "id": 0,
  "title": "string",
  "description": "string",
  "date": "2024-11-15T23:09:46.849Z",
  "questionIds": [
    0
  ],
  "createdById": 0
}
```


Parameters

Try it out

Name	Description
id * required	
integer(\$int64)	id
(path)	

Request body required

application/json

Example Value | Schema

```
{  "userId": 0,  "examId": 0,  "answers": [    {      "questionId": 0,      "answer": "string"    }  ]}
```

Responses		
Code	Description	Links
200	Exam submitted successfully	No links
<div>Media type<div>application/json</div></div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <pre>{ "id": 0, "userId": 0, "examId": 0, "score": 0}</pre>		
404	Exam or User not found	No links

Parameters

Try it out

Name	Description
id * required integer(\$int64) (path)	<div>id</div>

Responses

Code	Description	Links
200	<div>Exam retrieved successfully</div> <div>Media type<div>application/hal+json ▾</div><div>Controls Accept header.</div><div>Example Value Schema</div><div><pre>{ "id": 0, "title": "string", "description": "string", "date": "2024-11-15T23:11:32.263Z", "questionIds": [0], "createdById": 0}</pre></div></div>	No links

404

No links

Exam not found

Media type

application/hal+json

Example Value | Schema

```
{
  "id": 0,
  "title": "string",
  "description": "string",
  "date": "2024-11-15T23:11:32.264Z",
  "questionIds": [
    0
  ],
  "createdById": 0
}
```

500

No links

Server error

Media type

application/hal+json

Example Value | Schema

```
{
  "id": 0,
  "title": "string",
  "description": "string",
  "date": "2024-11-15T23:11:32.265Z",
  "questionIds": [
    0
  ],
  "createdById": 0
}
```

DELETE

/api/exams/{id} menghapus ujian oleh dosen

⌵

📋

Parameters

Try it out

Name	Description
<div><div>id * required</div><div>integer(\$int64)</div><div>(path)</div></div>	<div>id</div>

Responses

Code	Description	Links
200	Exam updated successfully	No links
404	Exam not found	No links
500	Server error	No links

7.5. Exam-Result-Controller

POST `/api/exam-results` Menyimpan hasil ujian

Parameters Try it out

No parameters

Request body required application/json

Example Value | Schema

```
{
  "id": 0,
  "userId": 0,
  "examId": 0,
  "score": 0
}
```

Responses

Code	Description	Links
200	Exam result found	No links
	<p>Media type</p> <div>application/json</div> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "id": 0, "userId": 0, "examId": 0, "score": 0}</pre>	
404	Exam result not found	No links

GET

/api/exam-results/user/{userId} Menampilkan hasil ujian berdasarkan id pengguna



Parameters

Try it out

Name	Description
------	-------------

userId * required

integer(\$int64)
(path)

userId

Responses		
Code	Description	Links
200	List of exam results	No links
<div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <pre>{ "id": 0, "userId": 0, "examId": 0, "score": 0 }</pre>		
404	No exam results found	No links

8. Uji Coba Endpoint

Dalam pengujian aplikasi, beberapa endpoint diuji untuk memastikan fungsionalitasnya berjalan dengan baik. Setiap endpoint diuji untuk memverifikasi bahwa permintaan dan responsnya sesuai dengan spesifikasi dan dapat menangani kasus-kasus yang berbeda, seperti autentikasi yang valid, pembuatan ujian baru, dan manajemen soal.

Berikut adalah langkah-langkah dan hasil uji coba untuk beberapa endpoint utama aplikasi:

8.1. Endpoint Registrasi Pengguna (POST /api/auth/register)

- **Tujuan:** Menguji fungsionalitas registrasi pengguna baru.
- **Deskripsi:** Pengguna mengirimkan data pendaftaran (nama, email, password) untuk membuat akun baru.
- **Request Body:**

```

1  {
2      "name": "John Doe Utomo",
3      "email": "utomo@example.com",
4      "password": "password123"
5  }
6
```

- ```
1 {
2 "id": 4,
3 "email": "utomo@example.com",
4 "name": "John Doe Utomo",
5 "role": "USER"
6 }
```

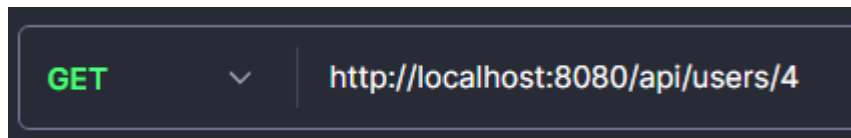
- ## 8.2. Endpoint Login (POST /api/auth/login)

- ```
1 {
2     "name": "John Doe utomo",
3     "email": "utomo@example.com",
4     "password": "password123"
5 }
```

-

- ### 8.3. Endpoint Melihat Profil Pengguna (GET /api/user/{id})

- Authorization: Bearer jwt token string



- **Respons:**
 - Status 200 OK: Menandakan permintaan berhasil.
 - Respons Body:

```
1 {
2   "id": 4,
3   "email": "utomo@example.com",
4   "name": "John Doe Utomo",
5   "role": "USER"
6 }
```

id	email	name	password	role
1	john.doe@example.com	John Doe	\$2a\$10\$5uv.85oQicln6nWK1YUWOAbbcV/PFe...	USER
2	john.doe@example.com	John Doe utomo	\$2a\$10\$LxCgdJ2tYXmJS/BR79y5/.OYMomsM08...	USER
4	utomo@example.com	John Doe Utomo	\$2a\$10\$CeQguFvz6on4zUYuiF.0g.CsGXpG/KFw...	USER
5	utomo@examadplue.com	John Doe	\$2a\$10\$gW/Vvb5KQkWBqGFJMB8f..JmtNJa2m...	USER
7	fajardoe@example.com	Fajar Doe	\$2a\$10\$15ohDuHUmWQn1Qq7vfmqeXrDxsIhE...	USER

- **Hasil Uji:** Endpoint berhasil mengembalikan data profil pengguna berdasarkan token JWT yang valid.

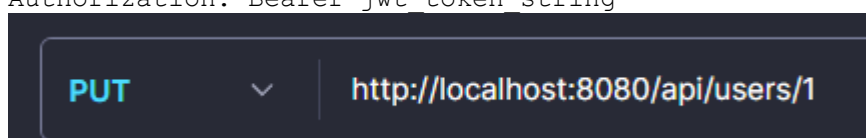
8.4. Endpoint Edit Profil Pengguna (PUT /api/user/{id})

- **Tujuan:** Menguji fungsionalitas untuk mengubah data profil pengguna.
- **Deskripsi:** Pengguna dapat mengubah data profilnya (misalnya, mengganti email atau username).
- **Request Body:**

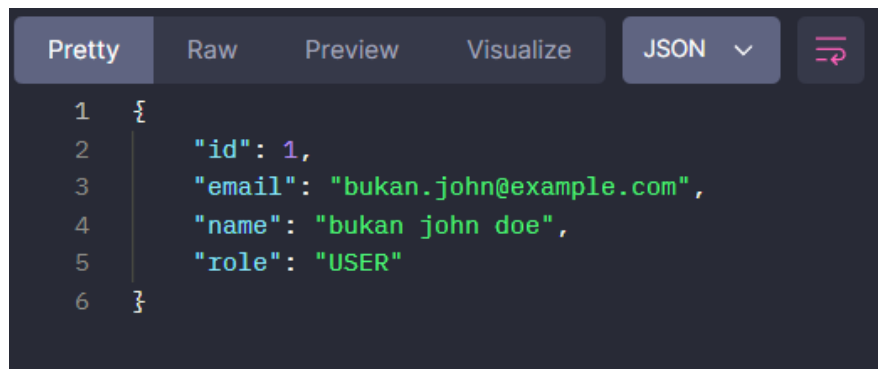
```
1 {
2   "id": 0,
3   "email": "bukan.john@example.com",
4   "name": "bukan john doe",
5   "role": "user"
6 }
```

- **Header:**

Authorization: Bearer jwt token string



- **Respons:**
 - Status 200 OK: Menandakan perubahan berhasil dilakukan.
 - Respons Body:



```

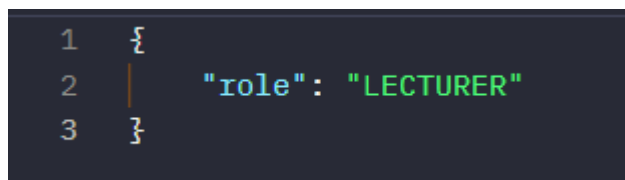
1  {
2      "id": 1,
3      "email": "bukan.john@example.com",
4      "name": "bukan john doe",
5      "role": "USER"
6  }

```

- **Hasil Uji:** Endpoint berhasil memperbarui profil pengguna dan mengembalikan status yang sesuai.

8.5. Endpoint Ganti Role Pengguna (PUT /api/users/{id}/role)

- **Tujuan:** Menguji fungsionalitas untuk mengubah role pengguna.
- **Deskripsi:** Administrator dapat mengubah role pengguna dengan menyertakan data role baru.
- **Request Body:**



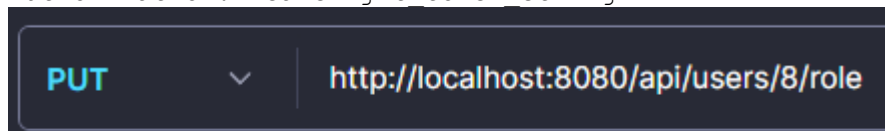
```

1  {
2      "role": "LECTURER"
3  }

```

- **Header:**

Authorization: Bearer jwt token string

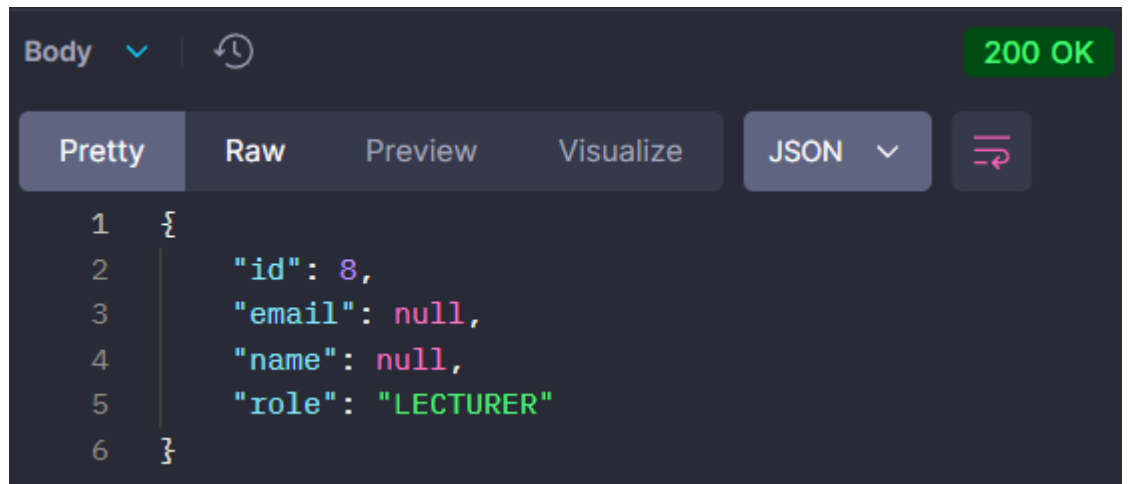


```

PUT http://localhost:8080/api/users/8/role

```

- **Respons:**
 - Status 200 OK: Menandakan perubahan role berhasil dilakukan.
 - Respons Body:



```
Body  [v] [refresh] 200 OK

Pretty Raw Preview Visualize JSON [dropdown] [refresh]

1 {
2   "id": 8,
3   "email": null,
4   "name": null,
5   "role": "LECTURER"
6 }
```

- **Hasil Uji:** Endpoint berhasil mengubah role pengguna dan mengembalikan status yang sesuai.

Sebelum:

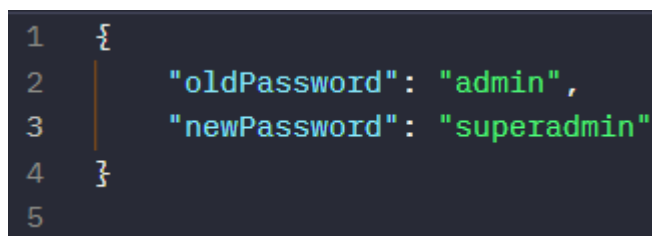
8	dero@example.com	dero	\$2a\$10\$oPDA4MvvHReJHj.XIwn5Au5tjQouRAP...	USER
---	------------------	------	--	------

Setelah:

8	dero@example.com	dero	\$2a\$10\$oPDA4MvvHReJHj.XIwn5Au5tjQouRAP...	LECTURER
---	------------------	------	--	----------

8.6. Endpoint Ganti Password Pengguna (PUT /api/users/{id}/change-password)

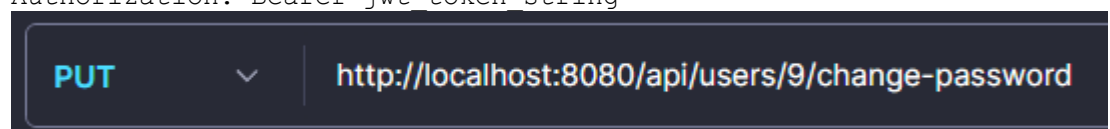
- **Tujuan:** Menguji fungsionalitas untuk mengubah kata sandi pengguna.
- **Deskripsi:** Pengguna dapat mengganti kata sandi mereka dengan menyertakan kata sandi lama dan kata sandi baru.
- **Request Body:**



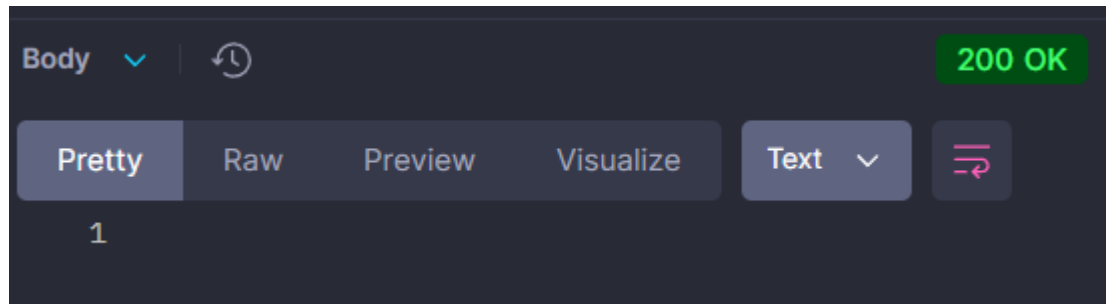
```
1 {
2   "oldPassword": "admin",
3   "newPassword": "superadmin"
4 }
5
```

- **Header:**

Authorization: Bearer jwt token string



- **Respons:**
 - Status 200 OK: Menandakan perubahan kata sandi berhasil dilakukan.
 - Respons Body:



- **Hasil Uji:** Endpoint berhasil mengganti kata sandi pengguna dan mengembalikan status yang sesuai.

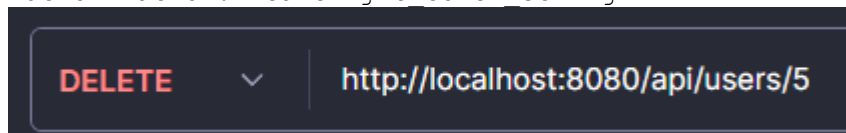
8.7. Endpoint Delete User (DELETE /api/users/{id})

- **Tujuan:** menguji fungsionalitas untuk menghapus pengguna.
- **Deskripsi:** Administrator dapat menghapus pengguna berdasarkan ID pengguna.
- **Request Body:**

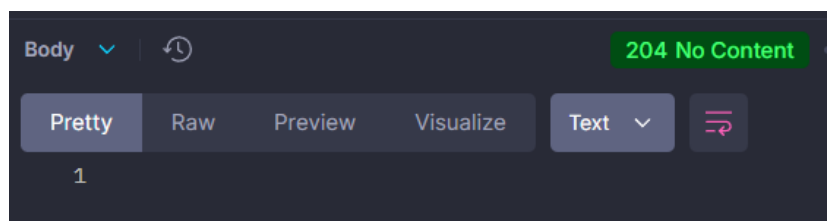
}

- **Header:**

Authorization: Bearer jwt token string



- **Respons:**
 - Status 200 OK: Menandakan pengguna berhasil dihapus.
 - Respons Body:



- **Hasil Uji:** Endpoint berhasil menghapus pengguna dan mengembalikan status yang sesuai.

Sebelum:

digixam.users: 7 rows total (approximately) >> Next Show all Sorting Columns (5/5)

id	email	name	password	role
1	bukan.john@example.com	bukan john doe	\$2a\$10\$5uv.85oQiclun6nWK1YUWOAbbcV/PFe...	USER
2	john.doe@examples.com	John Doe utomo	\$2a\$10\$LxCgdJ2tYXmJS/BR79y5/.OYMomsM08...	USER
4	utomo@example.com	John Doe Utomo	\$2a\$10\$CeQguFvz6on4zUYuiF.0g.CsGXpG/KFw...	USER
5	utomo@examadplue.com	John Doe	\$2a\$10\$gW/Vvb5KQkWQbGFJMB8f..JmtNJa2m...	USER
7	fajardoe@example.com	Fajar Doe	\$2a\$10\$15ohDuHumWQn1Qq7vmqeXrDxsIhE...	USER
8	dero@example.com	dero	\$2a\$10\$oPDA4MwHReJHj.XIwn5Au5tjQouRAP...	LECTURER
9	admin@a.bc	superadmin	\$2a\$10\$5WPD6KWJ0gNIiQtDdTWHTuP1adX2.9...	ADMIN

Setelah:

digixam.users: 6 rows total (approximately) >> Next Show all Sorting Columns (5/5)

id	email	name	password	role
1	bukan.john@example.com	bukan john doe	\$2a\$10\$5uv.85oQiclun6nWK1YUWOAbbcV/PFe...	USER
2	john.doe@examples.com	John Doe utomo	\$2a\$10\$LxCgdJ2tYXmJS/BR79y5/.OYMomsM08...	USER
4	utomo@example.com	John Doe Utomo	\$2a\$10\$CeQguFvz6on4zUYuiF.0g.CsGXpG/KFw...	USER
7	fajardoe@example.com	Fajar Doe	\$2a\$10\$15ohDuHumWQn1Qq7vmqeXrDxsIhE...	USER
8	dero@example.com	dero	\$2a\$10\$oPDA4MwHReJHj.XIwn5Au5tjQouRAP...	LECTURER
9	admin@a.bc	superadmin	\$2a\$10\$5WPD6KWJ0gNIiQtDdTWHTuP1adX2.9...	ADMIN

8.8. Endpoint Pembuatan Soal Ujian (POST /api/questions?userId={id})

POST <http://localhost:8080/api/questions?userId=1>

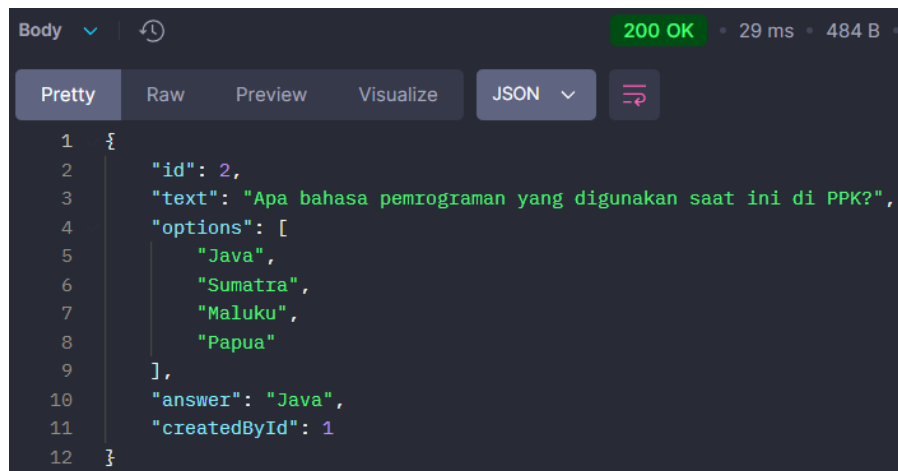
- **Tujuan:** Menguji fungsionalitas untuk menambah soal ke dalam ujian yang sudah dibuat.
- **Deskripsi:** Pengguna dapat menambahkan soal ke dalam ujian yang sudah ada dengan menyertakan ID ujian.
- **Request Body:**

```
1  {
2    "text": "Apa bahasa pemrograman yang digunakan saat ini di PPK?",
3    "options": ["Java", "Sumatra", "Maluku", "Papua"],
4    "answer": "Java"
5  }
6
```

- **Header:**

Authorization: Bearer jwt_token_string

- **Respons:**
 - Status 201 Created: Menandakan soal berhasil ditambahkan.
 - Respons Body:



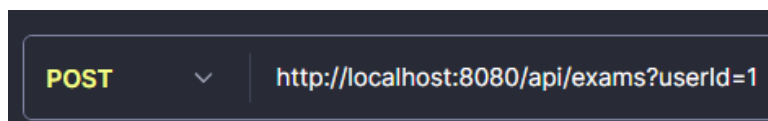
```

1  {
2      "id": 2,
3      "text": "Apa bahasa pemrograman yang digunakan saat ini di PPK?",
4      "options": [
5          "Java",
6          "Sumatra",
7          "Maluku",
8          "Papua"
9      ],
10     "answer": "Java",
11     "createdById": 1
12 }

```

- **Hasil Uji:** Endpoint berhasil menambah soal ke ujian dan mengembalikan status yang sesuai.

8.9. Endpoint Pembuatan Ujian (POST /api/exam/create)

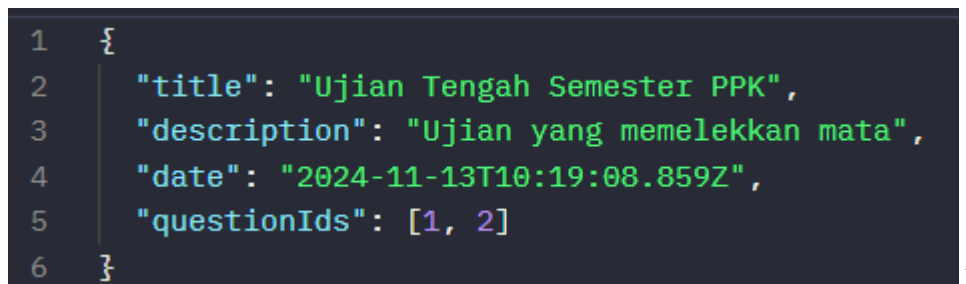


```

POST http://localhost:8080/api/exams?userId=1

```

- **Tujuan:** Menguji fungsionalitas untuk membuat ujian baru.
- **Deskripsi:** Pengguna dengan hak akses admin dapat membuat ujian baru dengan menentukan judul dan deskripsi ujian.
- **Request Body:**



```

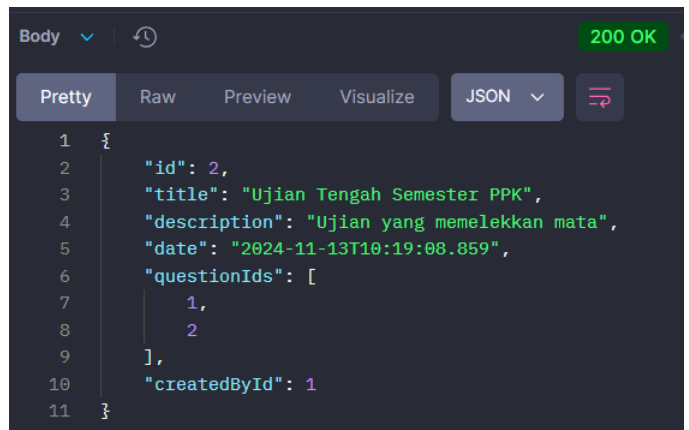
1  {
2      "title": "Ujian Tengah Semester PPK",
3      "description": "Ujian yang melelekan mata",
4      "date": "2024-11-13T10:19:08.859Z",
5      "questionIds": [1, 2]
6  }

```

- **Header:**

Authorization: Bearer jwt_token_string

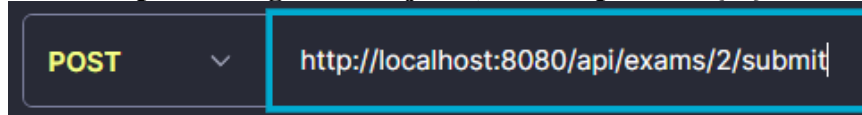
- **Respons:**
 - Status 201 Created: Menandakan ujian berhasil dibuat.
 - Respons Body:



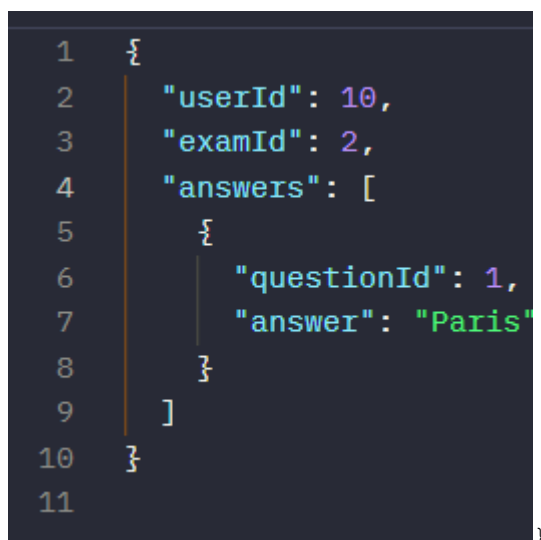
```
1 {
2   "id": 2,
3   "title": "Ujian Tengah Semester PPK",
4   "description": "Ujian yang melelekan mata",
5   "date": "2024-11-13T10:19:08.859",
6   "questionIds": [
7     1,
8     2
9   ],
10  "createdById": 1
11 }
```

- **Hasil Uji:** Endpoint berhasil membuat ujian baru dan mengembalikan status yang sesuai.

8.10. Endpoint Pengiriman Ujian (POST /api/exam/{id}/submit)



- **Tujuan:** Menguji fungsionalitas untuk mengirimkan ujian yang telah diselesaikan oleh pengguna.
- **Deskripsi:** Pengguna dengan hak akses STUDENT dapat mengirimkan jawaban ujian yang telah diselesaikan. Endpoint ini akan menerima jawaban dan menghitung hasil ujian.
- **Request Body:**

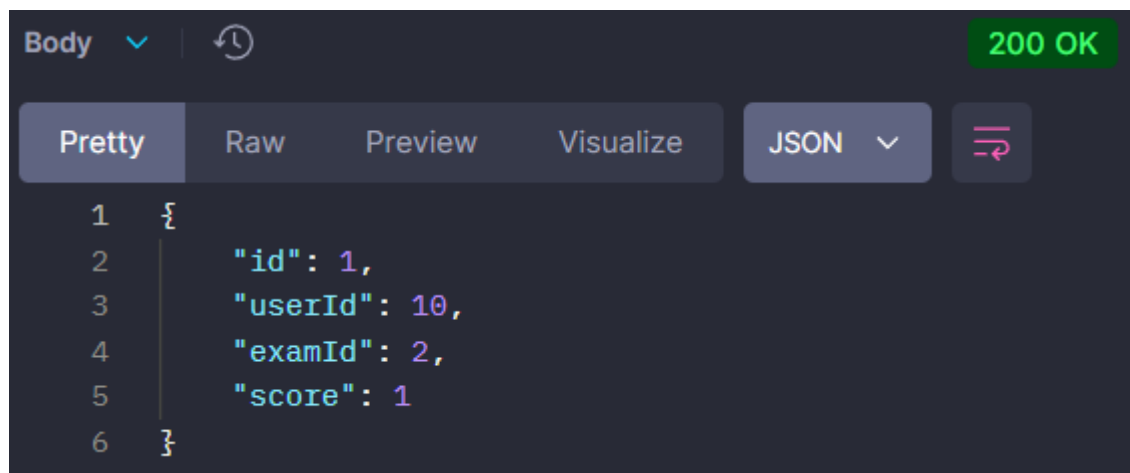


```
1 {
2   "userId": 10,
3   "examId": 2,
4   "answers": [
5     {
6       "questionId": 1,
7       "answer": "Paris"
8     }
9   ]
10 }
11 }
```

- **Header:**

Authorization: Bearer jwt_token_string

- **Respons:**
 - Status 201 Created: Menandakan ujian berhasil dibuat.
 - Respons Body:



- **Hasil Uji:** Endpoint berhasil mengirimkan jawaban ujian, menghitung hasil ujian, dan mengembalikan status serta respons yang sesuai.

9. Kesimpulan

Aplikasi Digiexam telah berhasil diimplementasikan sesuai dengan spesifikasi ujian tengah semester, menggunakan Java, Spring Boot, dan MySQL untuk membangun RESTful API. Aplikasi ini mengelola manajemen pengguna, pembuatan ujian, dan pengelolaan soal dengan menggunakan token-based authentication berbasis JWT untuk keamanan. Pengujian endpoint menunjukkan bahwa seluruh fitur berfungsi dengan baik, dan autentikasi serta otorisasi berjalan lancar. Dengan struktur yang terorganisir dan mematuhi prinsip desain yang baik, aplikasi ini siap digunakan dan dapat diperluas sesuai kebutuhan di masa depan.

Lampiran

1. Repository GIT - [Pamungkas Dero Ivano / STIS DigiExam · GitLab](#)