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;  
 }  
}

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);  
 }  
}

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(), score);  
 }  
}

package com.polstatstis.digiexam.controller;  
  
import com.polstatstis.digiexam.dto.ChangePasswordDTO;  
import com.polstatstis.digiexam.dto.UserDTO;  
import com.polstatstis.digiexam.service.UserService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.\*;  
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;  
  
@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) })  
  
 @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) })  
  
 @PutMapping("/{id}/change-password")  
 public ResponseEntity<Void> changePassword(@PathVariable Long id, @RequestBody 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) })  
  
 @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) })  
  
 @PutMapping("/{id}/role")  
 public ResponseEntity<UserDTO> changeUserRole(@PathVariable Long id, @RequestBody UserDTO userDTO) {  
 UserDTO updatedUser = userService.changeUserRole(id, userDTO);  
 return ResponseEntity.ok(updatedUser);  
 }  
}

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 lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.\*;  
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;  
  
@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("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")  
 })  
  
 @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")  
 })  
  
 @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)  
 })  
  
 @PostMapping("/{id}/submit")  
 public ResponseEntity<ExamResultDTO> submitExam(@PathVariable Long id, @RequestBody ExamSubmissionDTO submission) {  
 ExamResultDTO result = examService.submitExam(id, submission);  
 return ResponseEntity.ok(result);  
 }  
}

package com.polstatstis.digiexam.config;  
  
import com.polstatstis.digiexam.security.JwtAuthenticationFilter;  
import lombok.RequiredArgsConstructor;  
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.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 com.polstatstis.digiexam.service.UserDetailsServiceImpl;  
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/\*\*").permitAll()  
 .requestMatchers("/api/questions/\*\*").hasAnyRole("ADMIN, LECTURER")  
 .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();  
 }  
}

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();  
 }  
}

```java