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);  
 }  
}

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 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, @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 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);  
 }  
}

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();  
 }  
}