```java
// =====================
// app/controller/admin/AdminUserController.java
// =====================
package app.controller.admin;

import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import app.dto.UserDto;
import app.dto.UserSummary;
import app.dto.CreateUserRequest;
import app.dto.AssignRolesRequest;
import app.security.RequiresPermission;
import app.service.UserService;

@RestController
@RequestMapping("/api/admin/users")
public class AdminUserController {

        private final UserService userService;

        public AdminUserController(UserService userService) {
                this.userService = userService;
        }

        // ----------------------------------------------------
        // Create user
        // ----------------------------------------------------
        @PostMapping
        @RequiresPermission("admin.manage")
        public UserDto createUser(@RequestBody CreateUserRequest req) {
                return userService.create(
                                req.name(),
                                req.email(),
                                req.password(),
                                req.roleIds());
        }

        // GET ONE USER
        @GetMapping("/{userId}")
        @RequiresPermission("admin.manage")
        public ResponseEntity<UserDto> getUser(@PathVariable Long userId) {
                return ResponseEntity.ok(userService.getById(userId));
        }

        // ----------------------------------------------------
        // List all users
        // ----------------------------------------------------
        @GetMapping
        @RequiresPermission("admin.manage")
        public List<UserSummary> getUsers() {
                return userService.getAll();
        }

        // ----------------------------------------------------
        // Assign roles to a user
        // ----------------------------------------------------
        @PostMapping("/{userId}/roles")
        @RequiresPermission("admin.manage")
        public UserDto assignRoles(
                        @PathVariable Long userId,
                        @RequestBody AssignRolesRequest req) {
                return userService.assignRoles(userId, req.roleIds());
        }

        // ----------------------------------------------------
        // Remove roles from a user
        // ----------------------------------------------------
        @DeleteMapping("/{userId}/roles")
        @RequiresPermission("admin.manage")
        public UserDto removeRoles(
                        @PathVariable Long userId,
                        @RequestBody AssignRolesRequest req) {
                return userService.removeRoles(userId, req.roleIds());
        }
}

// ====================
// app/dto/ApiResponse.java
// ====================
package app.dto;

public class ApiResponse<T> {
        private String message;
        private T data;

        public ApiResponse(String message) {
                this.message = message;
        }

        public ApiResponse(String message, T data) {
                this.message = message;
                this.data = data;
        }

        // Getters and Setters
        public String getMessage() {
                return message;
        }

        public void setMessage(String message) {
                this.message = message;
        }

        public T getData() {
                return data;
        }

        public void setData(T data) {
                this.data = data;
        }
}

// =====================
// app/dto/AssignRolesRequest.java
// =====================
package app.dto;

import java.util.List;

public record AssignRolesRequest(List<Long> roleIds) {}

// =====================
// app/dto/AssignTicketRequest.java
// =====================
package app.dto;

public record AssignTicketRequest(Long userId) {}

// =====================
// app/dto/CreatePermissionRequest.java
// =====================
package app.dto;

public record CreatePermissionRequest(
        String name,
        String description
) {}

// =====================
// app/dto/CreateRoleRequest.java
// =====================
package app.dto;

import java.util.List;

public record CreateRoleRequest(
        String name,
        String description,
        List<Long> permissionIds
) {}

// =====================
// app/dto/CreateTicketRequest.java
// =====================
package app.dto;

public record CreateTicketRequest(
                String title,
                String body,
                int priorityId
) {}

// =====================
// app/dto/CreateUserRequest.java
// =====================
package app.dto;

import java.util.List;

public record CreateUserRequest(
                String name,
                String email,
                String password,
                List<Long> roleIds
) {}

// =====================
// app/dto/LoginRequest.java
// =====================
package app.dto;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;

public class LoginRequest {

        @Email
        @NotBlank
        private String email;

        @NotBlank
        private String password;

        public String getEmail() {
                return email;
        }
        public void setEmail(String email) {
                this.email = email;
        }

        public String getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
        }
}

// =====================
// app/dto/PermissionDto.java
// =====================
package app.dto;

import app.model.Permission;

public record PermissionDto(
                Long id,
                String name,
                String description
) {
        public static PermissionDto from(Permission p) {
                return new PermissionDto(p.getId(), p.getName(),
p.getDescription());
        }
}
// =====================
// app/controller/TicketController.java
// =====================
package app.controller;

import app.dto.AssignTicketRequest;
import app.dto.CreateTicketRequest;
import app.dto.UpdateTicketRequest;
import app.model.Ticket;
import app.security.Ownership;
import app.security.OwnershipType;
import app.security.RequiresPermission;
import app.service.TicketService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/tickets")
```

```
public class TicketController {

        private final TicketService ticketService;

        public TicketController(TicketService ticketService) {
                this.ticketService = ticketService;
        }

        // ----------------------------------------------------
        // READ – requires ticket.read (ALL) or OWN (SELF)
        // ----------------------------------------------------
        @GetMapping
        @RequiresPermission("ticket.read")
        @Ownership(OwnershipType.ALL_OR_SELF)
        public List<Ticket> getAllTickets() {
                return ticketService.getAllTickets();
        }

        @GetMapping("/{id}")
        @RequiresPermission("ticket.read")
        @Ownership(OwnershipType.ALL_OR_SELF)
        public ResponseEntity<Ticket> getTicketById(@PathVariable Long id) {
                return ticketService.getTicketById(id)
                                .map(ResponseEntity::ok)
                                .orElse(ResponseEntity.notFound().build());
        }

        // ----------------------------------------------------
        // SEARCH / FILTER – still requires ticket.read
        // OwnershipAspect will enforce ALL_OR_SELF automatically
        // ----------------------------------------------------
        @GetMapping("/status/{statusId}")
        @RequiresPermission("ticket.read")
        @Ownership(OwnershipType.ALL_OR_SELF)
        public List<Ticket> getTicketsByStatusId(@PathVariable int statusId) {
                return ticketService.getTicketsByStatusId(statusId);
        }

        @GetMapping("/priority/{priorityId}")
        @RequiresPermission("ticket.read")
        @Ownership(OwnershipType.ALL_OR_SELF)
        public List<Ticket> getTicketsByPriorityId(@PathVariable int priorityId) {
                return ticketService.getTicketsByPriorityId(priorityId);
        }

        @GetMapping("/search")
        @RequiresPermission("ticket.read")
        @Ownership(OwnershipType.ALL_OR_SELF)
        public List<Ticket> searchTicketsByTitle(@RequestParam String keyword) {
                return ticketService.searchTicketsByTitle(keyword);
        }

        // ----------------------------------------------------
        // CREATE – requires ticket.write
        // USER does NOT have ticket.write, but creation is allowed
        // because you give USER the ticket.create permission
        // ----------------------------------------------------
        @PostMapping
        @RequiresPermission("ticket.create")
        public Ticket createTicket(@RequestBody CreateTicketRequest req) {
                return ticketService.createTicket(req.title(), req.body(),
req.priorityId());
        }

        // ----------------------------------------------------
        // UPDATE – requires ticket.write OR SELF ownership
        // ----------------------------------------------------
        @PutMapping("/{id}")
        @RequiresPermission("ticket.write")
        @Ownership(OwnershipType.SELF)
        public ResponseEntity<Ticket> updateTicket(
                        @PathVariable Long id,
                        @RequestBody UpdateTicketRequest req) {
                return ticketService.getTicketById(id)
                                .map(existing -> {
                                        Ticket updated =
ticketService.updateTicketFields(
                                                        id,
                                                        req.title(),
                                                        req.body(),
                                                        req.priorityId());
                                        return ResponseEntity.ok(updated);
                                })
                                .orElse(ResponseEntity.notFound().build());
        }

        // ----------------------------------------------------
        // DELETE – requires ticket.delete (admin/manager only)
        // ----------------------------------------------------
        // TO IMPLEMENT

        // ----------------------------------------------------
        // ASSIGN – requires ticket.assign (support/manager/admin)
        // ----------------------------------------------------
        @PostMapping("/{id}/assign")
        @RequiresPermission("ticket.assign")
        public ResponseEntity<Ticket> assignTicket(
                        @PathVariable Long id,
                        @RequestBody AssignTicketRequest req) {
```

```
                return ResponseEntity.ok(ticketService.assignTicket(id,
req.userId()));
        }
}

// =====================
// app/controller/UserController.java
// =====================
package app.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import app.dto.UserInfo;
import app.security.CustomUserDetails;


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

        @GetMapping("/user")
        public ResponseEntity<UserInfo> currentUser(Authentication authentication)
{
                if (authentication == null || !authentication.isAuthenticated()) {
                        return ResponseEntity.status(401).build();
                }

                Object principal = authentication.getPrincipal();
                if (principal instanceof CustomUserDetails user) {
                        return ResponseEntity.ok(new UserInfo(user));
                }

                return ResponseEntity.status(500).build();
        }
}

// =====================
// app/controller/admin/AdminPermissionController.java
// =====================
package app.controller.admin;

import java.util.List;

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

import app.dto.CreatePermissionRequest;
import app.dto.PermissionDto;
import app.security.RequiresPermission;
import app.service.PermissionService;

@RestController
@RequestMapping("/api/admin/permissions")
public class AdminPermissionController {

        private final PermissionService permissionService;

        public AdminPermissionController(PermissionService permissionService) {
                this.permissionService = permissionService;
        }

        @PostMapping
        @RequiresPermission("admin.manage")
        public PermissionDto createPermission(@RequestBody CreatePermissionRequest
req) {
                return PermissionDto.from(
                                permissionService.create(req.name(),
req.description())
                );
        }

        @GetMapping
        @RequiresPermission("admin.manage")
        public List<PermissionDto> getPermissions() {
                return permissionService.getAll().stream()
                                .map(PermissionDto::from)
                                .toList();
        }
}

// =====================
// app/controller/admin/AdminPriorityController.java
// =====================
package app.controller.admin;

import app.dto.PriorityDto;
import app.service.PriorityService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/admin/priorities")
public class AdminPriorityController {
        @Autowired
        private PriorityService priorityService;

        @GetMapping
        public List<PriorityDto> getAllPriorities() {
                return priorityService.getAllPriorities();
        }

        @GetMapping("/{id}")
        public ResponseEntity<PriorityDto> getPriorityById(@PathVariable int id) {
                PriorityDto dto = priorityService.getPriorityById(id);
                if (dto == null) {
                        return ResponseEntity.notFound().build();
                }
                return ResponseEntity.ok(dto);
        }

        @PostMapping
        public ResponseEntity<PriorityDto> createPriority(@RequestBody PriorityDto
dto) {
                PriorityDto created = priorityService.createPriority(dto);
                return ResponseEntity.ok(created);
        }

        @PutMapping("/{id}")
        public ResponseEntity<PriorityDto> updatePriority(@PathVariable int id,
@RequestBody PriorityDto dto) {
                PriorityDto updated = priorityService.updatePriority(id, dto);
                if (updated == null) {
                        return ResponseEntity.notFound().build();
                }
                return ResponseEntity.ok(updated);
        }

        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deletePriority(@PathVariable int id) {
                boolean deleted = priorityService.deletePriority(id);
                if (!deleted) {
                        return ResponseEntity.notFound().build();
                }
                return ResponseEntity.noContent().build();
        }
}

// =====================
// app/controller/admin/AdminRoleController.java
// =====================
package app.controller.admin;

import java.util.List;
import org.springframework.web.bind.annotation.*;

import app.dto.CreateRoleRequest;
import app.dto.RoleDto;
import app.security.RequiresPermission;
import app.service.RoleService;

@RestController
@RequestMapping("/api/admin/roles")
public class AdminRoleController {

        private final RoleService roleService;

        public AdminRoleController(RoleService roleService) {
                this.roleService = roleService;
        }

        @PostMapping
        @RequiresPermission("admin.manage")
        public RoleDto createRole(@RequestBody CreateRoleRequest req) {
                return RoleDto.from(
                                roleService.create(req.name(), req.description(),
req.permissionIds())
                );
        }

        @GetMapping
        @RequiresPermission("admin.manage")
        public List<RoleDto> getRoles() {
                return roleService.getAll().stream()
                                .map(RoleDto::from)
                                .toList();
        }

        // UPDATE ROLE
        @PutMapping("/{roleId}")
        @RequiresPermission("admin.manage")
        public RoleDto updateRole(@PathVariable Long roleId, @RequestBody
CreateRoleRequest req) {
                return RoleDto.from(
                                roleService.update(roleId, req.name(),
req.description(), req.permissionIds())
                );
        }

        // PATCH ADD PERMISSIONS TO ROLE
        @PatchMapping("/{roleId}/permissions")
        @RequiresPermission("admin.manage")
        public RoleDto addPermissionsToRole(@PathVariable Long roleId, @RequestBody
List<Long> permissionIds) {
                return RoleDto.from(
                                roleService.addPermissions(roleId, permissionIds)
                );
        }
}
// ====================
// app/controller/AuthController.java
// ====================
package app.controller;

import app.dto.ApiResponse;
import app.dto.LoginRequest;
import app.dto.RefreshTokenRequest;
import app.dto.RegisterRequest;
import app.dto.UserInfo;
import app.security.CustomUserDetails;
import app.service.AuthService;
import jakarta.validation.Valid;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;

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

        private static final Logger log =
LoggerFactory.getLogger(AuthController.class);

        private final AuthService authService;

        public AuthController(AuthService authService) {
                this.authService = authService;
        }

        // ----------------------------------------------------
        // Login
        // ----------------------------------------------------
        @PostMapping("/login")
        public ResponseEntity<ApiResponse<?>> login(@Valid @RequestBody
LoginRequest req) {
```

```
                log.info("Login attempt for email: {}", req.getEmail());

                try {
                        Map<String, String> tokens = authService.login(req);

                        // Extract authenticated user from SecurityContext
                        var auth =
SecurityContextHolder.getContext().getAuthentication();
                        var userDetails = (CustomUserDetails) auth.getPrincipal();
                        var userInfo = new UserInfo(userDetails);

                        Map<String, Object> data = new HashMap<>();
                        data.put("user", userInfo);
                        data.put("access_token", tokens.get("access_token"));
                        data.put("refresh_token", tokens.get("refresh_token"));

                        return ResponseEntity.ok(new ApiResponse<>("Login
successful", data));

                } catch (RuntimeException e) {
                        log.error("Login failed for {}: {}", req.getEmail(),
e.getMessage());
                        return ResponseEntity
                                        .status(HttpStatus.UNAUTHORIZED)
                                        .body(new ApiResponse<>(e.getMessage()));
                }
        }

        // ----------------------------------------------------
        // Register
        // ----------------------------------------------------
        @PostMapping("/register")
        public ResponseEntity<ApiResponse<?>> register(@Valid @RequestBody
RegisterRequest req) {
                log.info("Registration attempt for {}", req.getEmail());

                try {
                        var user = authService.register(req);

                        return ResponseEntity
                                        .status(HttpStatus.CREATED)
                                        .body(new ApiResponse<>(
                                                        "User registered
successfully",
                                                        user.getName()
                                        ));

                } catch (RuntimeException e) {
                        log.error("Registration failed for {}: {}", req.getEmail(),
e.getMessage());
```

```
                        return ResponseEntity
                                        .badRequest()
                                        .body(new ApiResponse<>(e.getMessage()));
                }
        }

        // ----------------------------------------------------
        // Refresh Token
        // ----------------------------------------------------
        @PostMapping("/refresh")
        public ResponseEntity<ApiResponse<?>> refresh(@Valid @RequestBody
RefreshTokenRequest req) {
                log.info("Refresh token attempt");

                try {
                        var tokens =
authService.refreshToken(req.getRefreshToken());
                        return ResponseEntity.ok(new ApiResponse<>("Token refreshed
successfully", tokens));

                } catch (RuntimeException e) {
                        log.error("Token refresh failed: {}", e.getMessage());
                        return ResponseEntity
                                        .status(HttpStatus.UNAUTHORIZED)
                                        .body(new ApiResponse<>(e.getMessage()));
                }
        }
}

// =====================
// app/controller/HomeController.java
// =====================
package app.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {
          // Catch-all for non-API routes
        @RequestMapping({"/{path:[^\\.]*}", "/**/{path:[^\\.]*}"})
        public String forward() {
                return "forward:/index.html";
        }
}

// =====================
// app/controller/MessageController.java
// =====================
package app.controller;
```

```
import app.model.Message;
import app.security.Ownership;
import app.security.OwnershipType;
import app.security.RequiresPermission;
import app.service.MessageService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/conversations")
public class MessageController {

        private final MessageService messageService;

        public MessageController(MessageService messageService) {
                this.messageService = messageService;
        }

        // ----------------------------------------------------
        // GET messages for a conversation
        // USER can only view their own conversation
        // ----------------------------------------------------
        @GetMapping("/{conversationId}/messages")
        @RequiresPermission("conversation.read")
        @Ownership(OwnershipType.ALL_OR_SELF)
        public List<Message> getMessages(@PathVariable int conversationId) {
                return messageService.getMessages(conversationId);
        }

        // ----------------------------------------------------
        // POST message (reply)
        // USER can only reply to their own conversation
        // ----------------------------------------------------
        @PostMapping("/{conversationId}/messages")
        @RequiresPermission("conversation.reply")
        @Ownership(OwnershipType.ALL_OR_SELF)
        public ResponseEntity<Message> addMessage(
                        @PathVariable int conversationId,
                        @RequestBody String body
        ) {
                return ResponseEntity.ok(
                                messageService.addMessage(conversationId, body)
                );
        }
}

// ====================
// app/controller/PriorityController.java
// ====================
package app.controller;

import app.dto.PriorityDto;
import app.service.PriorityService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
@RequestMapping("/api/priorities")
public class PriorityController {

        @Autowired
        private PriorityService priorityService;

        @GetMapping
        public List<PriorityDto> getAllPriorities() {
                return priorityService.getAllPriorities();
        }
}

// ====================
// app/controller/StatusController.java
// ====================
package app.controller;

import app.dto.StatusDto;
import app.mapper.StatusMapper;
import app.model.Status;
import app.security.RequiresPermission;
import app.service.StatusService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/status")
public class StatusController {

        private final StatusService statusService;

        public StatusController(StatusService statusService) {
                this.statusService = statusService;
        }
        @GetMapping
        @RequiresPermission({"status.read", "status.manage"})
        public List<StatusDto> getAll() {
                return statusService.findAll()
                                .stream()
                                .map(StatusMapper::toDTO)
                                .toList();
        }

        @GetMapping("/{id}")
        @RequiresPermission({"status.read", "status.manage"})
        public ResponseEntity<StatusDto> getById(@PathVariable int id) {
                return statusService.findById(id)
                                .map(StatusMapper::toDTO)
                                .map(ResponseEntity::ok)
                                .orElse(ResponseEntity.notFound().build());
        }

        @PostMapping
        @RequiresPermission({"status.write", "status.manage"})
        public ResponseEntity<StatusDto> create(@RequestBody StatusDto dto) {
                Status saved = statusService.save(StatusMapper.toEntity(dto));
                return ResponseEntity.ok(StatusMapper.toDTO(saved));
        }

        @PutMapping("/{id}")
        @RequiresPermission({"status.write", "status.manage"})
        public ResponseEntity<StatusDto> update(@PathVariable int id, @RequestBody
StatusDto dto) {
                return statusService.findById(id)
                                .map(existing -> {
                                        existing.setName(dto.getName());
                                        existing.setType(dto.getType());
                                        Status updated =
statusService.save(existing);
                                        return
ResponseEntity.ok(StatusMapper.toDTO(updated));
                                })
                                .orElse(ResponseEntity.notFound().build());
        }

        @DeleteMapping("/{id}")
        @RequiresPermission({"status.delete", "status.manage"})
        public ResponseEntity<Void> delete(@PathVariable int id) {
                statusService.deleteById(id);
                return ResponseEntity.noContent().build();
        }
}
// =====================
// app/Application.java
// =====================
package app;

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

@SpringBootApplication
public class Application {

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

}

// =====================
// app/config/CorsConfig.java
// =====================
package app.config;

import java.util.List;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.env.Environment;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

@Configuration
public class CorsConfig {

	@Value("${app.security.allowed-origin:http://localhost:8100}")
	private String allowedOrigins;

	private final Environment env;

	public CorsConfig(Environment env) {
		this.env = env;
	}

	@Bean
	public CorsConfigurationSource corsConfigurationSource() {
		CorsConfiguration configuration = new CorsConfiguration();

		String profile = env.getProperty("spring.profiles.active", "prod");
		if (profile.equalsIgnoreCase("dev")) {
			// In dev, allow origins from env
configuration.setAllowedOrigins(List.of(allowedOrigins.split(",\s*")));
                        configuration.setAllowCredentials(true);
                        configuration.setAllowedMethods(List.of("GET", "POST",
"PUT", "DELETE", "OPTIONS", "PATCH"));
                        configuration.setAllowedHeaders(List.of("Authorization",
"Content-Type", "Accept"));
                        configuration.setExposedHeaders(List.of());
                        configuration.setMaxAge(3600L);
                } else {
                        // In prod, disable all external origins
                        configuration.setAllowedOrigins(List.of());
                        configuration.setAllowCredentials(false);
                        configuration.setAllowedMethods(List.of("GET"));
                        configuration.setAllowedHeaders(List.of());
                        configuration.setExposedHeaders(List.of());
                        configuration.setMaxAge(0L);
                }

                UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
                source.registerCorsConfiguration("/**", configuration);
                return source;
        }
}

// ====================
// app/config/DataInitializer.java
// ====================
package app.config;

import app.model.Permission;
import app.model.Role;
import app.model.User;
import app.repository.PermissionRepository;
import app.repository.RoleRepository;
import app.repository.UserRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.password.PasswordEncoder;

import java.util.Set;

@Configuration
public class DataInitializer {

        @Bean
        CommandLineRunner initData(
                        UserRepository userRepo,
                        RoleRepository roleRepo,
                        PermissionRepository permRepo,
                        PasswordEncoder passwordEncoder
        ) {
                return args -> {

                        // Only run if no users exist
                        if (userRepo.count() > 0) {
                                return;
                        }

                        // 1. Create base permissions
                        Permission adminManage = permRepo.save(new
Permission("admin.manage", "Full admin access"));
                        Permission userManage = permRepo.save(new
Permission("user.manage", "Manage users"));

                        // 2. Create ADMIN role with permissions
                        Role adminRole = new Role();
                        adminRole.setName("ADMIN");
                        adminRole.setDescription("System administrator");
                        adminRole.setPermissions(Set.of(adminManage, userManage));
                        adminRole = roleRepo.save(adminRole);

                        // 3. Create the first admin user
                        User admin = new User();
                        admin.setName("Administrator");
                        admin.setEmail("admin@example.com");
                        admin.setPassword(passwordEncoder.encode("admin123")); //
change later
                        admin.setRoles(Set.of(adminRole));

                        userRepo.save(admin);

                        System.out.println("✔ First admin user created:
admin@example.com / admin123");
                };
        }
}

// =====================
// app/config/JpaConfig.java
// =====================
package app.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@Configuration
@EnableJpaAuditing
public class JpaConfig {
}

// ====================
// app/config/SecurityConfig.java
// ====================
package app.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
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.UsernamePasswordAuthenticationFilte
r;

import app.security.JwtAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

        private final JwtAuthenticationFilter jwtAuthenticationFilter;

        public SecurityConfig(JwtAuthenticationFilter jwtAuthenticationFilter) {
                this.jwtAuthenticationFilter = jwtAuthenticationFilter;
        }

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

        @Bean
        public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {

                // CORS handled by CorsConfig
                http.cors(cors -> {});

                // Disable CSRF (API-only backend)
                http.csrf(csrf -> csrf.disable());

                // Use stateless JWT auth
                http.sessionManagement(sm ->
sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
```

```
                http.authorizeHttpRequests(auth -> auth
                                // React static build
                                .requestMatchers(
                                                "/",
                                                "/index.html",
                                                "/assets/**",
                                                "/favicon.svg",
                                                "/static/**"
                                ).permitAll()

                                // Public API endpoints
                                .requestMatchers("/api/auth/**").permitAll()

                                // Everything else requires authentication
                                .anyRequest().authenticated()
                        );

                // Disable form login + HTTP Basic (API uses tokens or custom auth)
                http.httpBasic(basic -> basic.disable());
                http.formLogin(form -> form.disable());

                        // Attach JWT auth filter
                        http.addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);

                return http.build();
        }
}
```