Users API

Project Setup

Prerequisites

* Java: JDK 21 or higher
* Spring Boot: Version 3.x
* PostgreSQL: Installed and running (e.g., on localhost:5432)
* Maven: For dependency management
* IDE: IntelliJ IDEA, Eclipse, or VS Code
* Postman: For API testing

Create a Spring Boot Project

* Use Spring Initializr [(https://start.spring.io/)](https://start.spring.io/) to create a new project.
* Select:
  + Project: Maven
  + Language: Java
  + Spring Boot: 3.x
  + Dependencies:
    - Spring Web
    - Spring Data JPA
    - PostgreSQL Driver
    - Spring Security
    - Validation
    - Lombok (optional, for reducing boilerplate code)
* Generate and import the project into your IDE.

Dependencies

Add the following dependencies to your pom.xml:

Xml (Latest: <https://github.com/expertmartin/kingtan/tree/main/api-users>)

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.5.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.kingtan</groupId>  
 <artifactId>users</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>users</name>  
 <description>Kingtan Farm</description>  
 <url/>  
 <licenses>  
 <license/>  
 </licenses>  
 <developers>  
 <developer/>  
 </developers>  
 <scm>  
 <connection/>  
 <developerConnection/>  
 <tag/>  
 <url/>  
 </scm>  
 <properties>  
 <java.version>21</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-validation</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <!-- H2 Database for testing -->  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <!-- Jackson for JSON serialization -->  
 <dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-api</artifactId>  
 <version>0.11.5</version>  
 </dependency>  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-impl</artifactId>  
 <version>0.11.5</version>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-jackson</artifactId>  
 <version>0.11.5</version>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-mail</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
 <version>2.8.9</version>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.13.0</version>  
 <configuration>  
 <release>21</release>  
 <source>21</source>  
 <target>21</target>  
 <annotationProcessorPaths>  
 <path>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>${lombok.version}</version>  
 </path>  
 </annotationProcessorPaths>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <excludes>  
 <exclude>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </exclude>  
 </excludes>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

Database Schema PostgreSQL Setup

* On Docker Desktop,download PostgreSQL docker image and start the container pg at port 5432



application.yml settings for PostgreSQL:

*########### Postgres Begin ###################* datasource:  
 url: jdbc:postgresql://192.168.1.75:5432/db\_users  
 username: postgres  
 password: hulun  
 driver-class-name: org.postgresql.Driver  
  
*##### JPA/Hibernate configuration* jpa:  
 properties:  
 hibernate:  
 dialect: org.hibernate.dialect.PostgreSQLDialect  
 format\_sql: true  
 ddl-auto: update  
 show-sql: true  
  
*##### Enable to run sql script, such as schema.sql and dada.sql* sql:  
 init:  
 mode: always

* Create a new database with PGAdmin on Docker Desktop:

A screenshot of a computer

AI-generated content may be incorrect.

* Create a non-admin user for better security: (not executed yet. Should do it later)

sql

CREATE USER app\_user WITH PASSWORD 'securepassword';

GRANT ALL PRIVILEGES ON DATABASE user\_management TO app\_user;

Database Schema

Create tables for users and roles with a many-to-many relationship. Scripts are in the file:

**api-users/src/main/resources/schema.sql**

-- CREATE DATABASE db\_users;  
  
-- CREATE USER app\_user WITH PASSWORD 'securepassword';  
-- GRANT ALL PRIVILEGES ON DATABASE user\_management TO app\_user;  
  
 -- Users table  
 CREATE TABLE IF NOT EXISTS users (  
 id BIGSERIAL PRIMARY KEY,  
 username VARCHAR(50) UNIQUE NOT NULL,  
 email VARCHAR(100) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL,  
 enabled BOOLEAN NOT NULL DEFAULT TRUE  
 );  
  
 -- Roles table  
 CREATE TABLE IF NOT EXISTS roles (  
 id BIGSERIAL PRIMARY KEY,  
 name VARCHAR(50) UNIQUE NOT NULL  
 );  
  
 -- Junction table for user-role relationship  
 CREATE TABLE IF NOT EXISTS user\_roles (  
 user\_id BIGINT NOT NULL,  
 role\_id BIGINT NOT NULL,  
 PRIMARY KEY (user\_id, role\_id),  
 FOREIGN KEY (user\_id) REFERENCES users(id),  
 FOREIGN KEY (role\_id) REFERENCES roles(id)  
 );  
  
 CREATE TABLE IF NOT EXISTS password\_reset\_token (  
 id BIGSERIAL PRIMARY KEY,  
 token VARCHAR(255) NOT NULL,  
 user\_id BIGINT NOT NULL,  
 expiry\_date TIMESTAMP NOT NULL,  
 FOREIGN KEY (user\_id) REFERENCES users(id)  
 );

Seed Initial Data Insert default roles (e.g., ROLE\_USER and ROLE\_ADMIN):

**api-users/src/main/resources/data.sql**

INSERT INTO roles (name) VALUES ('ROLE\_USER') ON CONFLICT (NAME) DO NOTHING;  
INSERT INTO roles (name) VALUES ('ROLE\_ADMIN') ON CONFLICT (NAME) DO NOTHING;

Spring Boot Configuration

Application Properties Configure the database connection and JPA settings in src/main/resources/application.yml:

spring:  
 application:  
 name: users  
  
*########### Postgres Begin ###################* datasource:  
 url: jdbc:postgresql://192.168.1.75:5432/db\_users  
 username: postgres  
 password: hulun  
 driver-class-name: org.postgresql.Driver  
  
*##### JPA/Hibernate configuration* jpa:  
 properties:  
 hibernate:  
 dialect: org.hibernate.dialect.PostgreSQLDialect  
 format\_sql: true  
 ddl-auto: update  
 show-sql: true  
  
*##### Enable to run sql script, such as schema.sql and dada.sql* sql:  
 init:  
 mode: always  
  
*##### Email configuration* mail:  
 host: smtp.gmail.com  
 port: 587  
 username: kingtan.farm@gmail.com  
 password: fcnc$30n  
 properties:  
 mail:  
 smtp:  
 auth: true  
 starttls:  
 enable: true  
   
*##### JWT configuration (for authentication)*app:  
 jwt:  
 secret: tanyuanwodixinganerwoainiwoxiangrinidebitiannidebizairiyicicaonide1meilide1mapismshzainenggenniriyicimapitnayuanninameilidemapi  
 expiration: 86400000  
  
springdoc:  
 api-docs:  
 path: /api-docs *# Endpoint for OpenAPI JSON* swagger-ui:  
 path: /swagger-ui.html *# Swagger UI endpoint  
  
#api:  
# users-path: /api/v1/users  
# auth-path: /api/v1/auth*

* ddl-auto=update: Updates the schema based on entities without dropping existing data.
* jwt.secret: A secure key for signing JWT tokens.
* jwt.expiration: Token expiration time in milliseconds (e.g., 24 hours).

Entity Classes

Create entity classes to map to the database tables. The new API, record, is used to implemented DTOs.

User

java

package com.kingtan.users.model;  
  
import jakarta.persistence.\*;  
import lombok.Data;  
import java.util.HashSet;  
import java.util.Set;  
  
@Data  
@Entity  
@Table(name = "users")  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(unique = true, nullable = false)  
 private String username;  
  
 @Column(unique = true, nullable = false)  
 private String email;  
  
 @Column(nullable = false)  
 private String password;  
  
// @Column  
// private String phone; // For 2FA  
  
 @Column(nullable = false)  
 private boolean enabled = true;  
  
 @ManyToMany(fetch = FetchType.*EAGER*)  
 @JoinTable(  
 name = "user\_roles",  
 joinColumns = @JoinColumn(name = "user\_id"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id")  
 )  
 private Set<Role> roles = new HashSet<>();  
}

Role

Java

package com.kingtan.users.model;  
  
import jakarta.persistence.\*;  
import lombok.Data;  
  
@Data  
@Entity  
@Table(name = "roles")  
public class Role {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(unique = true, nullable = false)  
 private String name;  
}

## PasswordResetToken Entity

package com.kingtan.users.model;  
  
import jakarta.persistence.\*;  
import lombok.Data;  
  
import java.time.LocalDateTime;  
  
@Data  
@Entity  
public class PasswordResetToken {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false)  
 private String token;  
  
 @OneToOne  
 @JoinColumn(name = "user\_id", nullable = false)  
 private User user;  
  
 @Column(nullable = false)  
 private LocalDateTime expiryDate;  
}

TwoFactorCode Entity

package com.kingtan.users.model;  
  
import jakarta.persistence.\*;  
import lombok.Data;  
import java.time.LocalDateTime;  
  
@Data  
@Entity  
@Table(name = "two\_factor\_codes")  
public class TwoFactorCode {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false)  
 private String code;  
  
 @Column(nullable = false)  
 private Long userId;  
  
 @Column(nullable = false)  
 private LocalDateTime expiresAt;  
  
 @Column(nullable = false)  
 private boolean used = false;  
}

DTO Classes

Use DTOs (Data Transfer Objects) to transfer data between layers and avoid exposing entity details. The new API, record, is used to implement DTOs.

UserDTO

java

package com.kingtan.users.dto;  
  
import jakarta.validation.constraints.Email;  
import jakarta.validation.constraints.NotBlank;  
import java.util.Set;  
  
public record UserDTO(  
 Long id,  
  
 @NotBlank(message = "Username cannot be blank")  
 String username,  
  
 @Email(message = "Email must be valid")  
 @NotBlank(message = "Email cannot be blank")  
 String email,  
  
 Set<String> roles  
) {}

SignupRequest

java

package com.kingtan.users.dto;  
  
import jakarta.validation.constraints.Email;  
import jakarta.validation.constraints.NotBlank;  
  
public record SignupRequest (  
 @NotBlank(message = "Username is required")  
 String username,  
  
 @NotBlank(message = "Email is required")  
 @Email(message = "Email should be valid")  
 String email,  
  
 @NotBlank(message = "Password is required")  
 String password  
) {}

LoginRequest

java

package com.kingtan.users.dto;  
  
import jakarta.validation.constraints.NotBlank;  
  
public record LoginRequest (  
 @NotBlank(message = "Username is required")  
 String username,  
  
 @NotBlank(message = "Password is required")  
 String password  
) {}

Repository Layer

UserRepository

Java

package com.kingtan.users.repository;  
  
import com.kingtan.users.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import java.util.Optional;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
 Optional<User> findByUsername(String username);  
 Optional<User> findByEmail(String email);  
 boolean existsByUsername(String username);  
 boolean existsByEmail(String email);  
}

RoleRepository

Java

package com.kingtan.users.repository;  
  
import com.kingtan.users.model.Role;  
import org.springframework.data.jpa.repository.JpaRepository;  
import java.util.Optional;  
  
public interface RoleRepository extends JpaRepository<Role, Long> {  
 Optional<Role> findByName(String name);  
}

Service Layer

## UserService

The service layer contains business logic, including user registration, login, and CRUD operations.

java

package com.kingtan.users.service;  
  
import com.kingtan.users.dto.SignupRequest;  
import com.kingtan.users.dto.UserDTO;  
import com.kingtan.users.model.Role;  
import com.kingtan.users.model.User;  
import com.kingtan.users.repository.RoleRepository;  
import com.kingtan.users.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
import java.util.stream.Collectors;  
  
@Service  
public class UserService {  
  
 private final UserRepository userRepository;  
 private final RoleRepository roleRepository;  
 private final PasswordEncoder passwordEncoder;  
  
 @Autowired  
 public UserService(UserRepository userRepository, RoleRepository roleRepository, PasswordEncoder passwordEncoder) {  
 this.userRepository = userRepository;  
 this.roleRepository = roleRepository;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 @Transactional  
 public UserDTO registerUser(SignupRequest request) {  
 if (userRepository.existsByUsername(request.username())) {  
 throw new RuntimeException("Username is already taken");  
 }  
 if (userRepository.existsByEmail(request.email())) {  
 throw new RuntimeException("Email is already in use");  
 }  
  
 User user = new User();  
 user.setUsername(request.username());  
 user.setEmail(request.email());  
 user.setPassword(passwordEncoder.encode(request.password()));  
  
 // Assign default role (e.g., ROLE\_USER)  
 Role userRole = roleRepository.findByName("ROLE\_USER")  
 .orElseThrow(() -> new RuntimeException("Role not found"));  
 user.setRoles(new HashSet<>(Set.*of*(userRole)));  
  
 user = userRepository.save(user);  
 return mapToDTO(user);  
 }  
  
 public UserDTO findByUsername(String username) {  
 User user = userRepository.findByUsername(username)  
 .orElseThrow(() -> new RuntimeException("User not found"));  
 return mapToDTO(user);  
 }  
  
 public List<UserDTO> findAllUsers() {  
 return userRepository.findAll().stream()  
 .map(this::mapToDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 @Transactional  
 public UserDTO updateUser(Long id, UserDTO userDTO) {  
 User user = userRepository.findById(id)  
 .orElseThrow(() -> new RuntimeException("User not found"));  
 user.setUsername(userDTO.username());  
 user.setEmail(userDTO.email());  
  
 // Update roles if provided  
 if (userDTO.roles() != null) {  
 Set<Role> roles = userDTO.roles().stream()  
 .map(roleName -> roleRepository.findByName(roleName)  
 .orElseThrow(() -> new RuntimeException("Role not found: " + roleName)))  
 .collect(Collectors.*toSet*());  
 user.setRoles(roles);  
 }  
  
 user = userRepository.save(user);  
 return mapToDTO(user);  
 }  
  
 @Transactional  
 public void deleteUser(Long id) {  
 if (!userRepository.existsById(id)) {  
 throw new RuntimeException("User not found");  
 }  
 userRepository.deleteById(id);  
 }  
  
 private UserDTO mapToDTO(User user) {  
 UserDTO dto = new UserDTO(user.getId(), user.getUsername(), user.getEmail(), user.getRoles().stream()  
 .map(Role::getName)  
 .collect(Collectors.*toSet*()));  
 return dto;  
 }  
}

## PasswordResetService

package com.kingtan.users.service;  
  
import com.kingtan.users.model.PasswordResetToken;  
import com.kingtan.users.model.User;  
import com.kingtan.users.repository.PasswordResetTokenRepository;  
import com.kingtan.users.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.mail.javamail.MimeMessageHelper;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import jakarta.mail.MessagingException;  
import jakarta.mail.internet.MimeMessage;  
import java.time.LocalDateTime;  
import java.util.UUID;  
  
@Service  
public class PasswordResetService {  
  
 private final UserRepository userRepository;  
 private final PasswordResetTokenRepository tokenRepository;  
 private final JavaMailSender mailSender;  
 private final PasswordEncoder passwordEncoder; // Add PasswordEncoder field  
  
 @Autowired  
 public PasswordResetService(UserRepository userRepository,  
 PasswordResetTokenRepository tokenRepository,  
 JavaMailSender mailSender,  
 PasswordEncoder passwordEncoder) { // Inject PasswordEncoder  
 this.userRepository = userRepository;  
 this.tokenRepository = tokenRepository;  
 this.mailSender = mailSender;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 @Transactional  
 public void createPasswordResetToken(String email) throws MessagingException {  
 User user = userRepository.findByEmail(email)  
 .orElseThrow(() -> new RuntimeException("User not found"));  
  
 String token = UUID.*randomUUID*().toString();  
 PasswordResetToken resetToken = new PasswordResetToken();  
 resetToken.setToken(token);  
 resetToken.setUser(user);  
 resetToken.setExpiryDate(LocalDateTime.*now*().plusHours(1));  
 tokenRepository.save(resetToken);  
  
 sendEmail(user.getEmail(), token);  
 }  
  
 private void sendEmail(String to, String token) throws MessagingException {  
 MimeMessage message = mailSender.createMimeMessage();  
 MimeMessageHelper helper = new MimeMessageHelper(message);  
 helper.setTo(to);  
 helper.setSubject("Password Reset Request");  
 helper.setText("Use this token to reset your password: " + token);  
 mailSender.send(message);  
 }  
  
 @Transactional  
 public void resetPassword(String token, String newPassword) {  
 PasswordResetToken resetToken = tokenRepository.findByToken(token)  
 .orElseThrow(() -> new RuntimeException("Invalid token"));  
 if (resetToken.getExpiryDate().isBefore(LocalDateTime.*now*())) {  
 throw new RuntimeException("Token expired");  
 }  
  
 User user = resetToken.getUser();  
 user.setPassword(passwordEncoder.encode(newPassword));  
 userRepository.save(user);  
 tokenRepository.delete(resetToken);  
 }  
}

Controller Layer

UserController

Expose REST endpoints for user management.

java

package com.kingtan.users.controller;  
  
import com.kingtan.users.dto.SignupRequest;  
import com.kingtan.users.dto.UserDTO;  
import com.kingtan.users.service.UserService;  
import io.swagger.v3.oas.annotations.Operation;  
import io.swagger.v3.oas.annotations.tags.Tag;  
import jakarta.validation.Valid;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/v1/users")  
@Tag(name = "Users", description = "Operations for managing users")  
public class UserController {  
  
 private final UserService userService;  
  
 @Autowired  
 public UserController(UserService userService) {  
 this.userService = userService;  
 }  
  
 @PostMapping("/register")  
 @Operation(summary = "Register user")  
 public ResponseEntity<UserDTO> register(@Valid @RequestBody SignupRequest request) {  
 UserDTO userDTO = userService.registerUser(request);  
 return ResponseEntity.*ok*(userDTO);  
 }  
  
 @Operation(summary = "Get user by ID")  
 @GetMapping("/{username}")  
 @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")  
 public ResponseEntity<UserDTO> getUser(@PathVariable String username) {  
 UserDTO userDTO = userService.findByUsername(username);  
 return ResponseEntity.*ok*(userDTO);  
 }  
  
 @GetMapping  
 @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")  
 @Operation(summary = "Get all users")  
 public ResponseEntity<List<UserDTO>> getAllUsers() {  
 List<UserDTO> users = userService.findAllUsers();  
 return ResponseEntity.*ok*(users);  
 }  
  
 @PutMapping("/{id}")  
 @PreAuthorize("hasRole('ADMIN') or #id == authentication.principal.id")  
 @Operation(summary = "Update user by ID")  
 public ResponseEntity<UserDTO> updateUser(@PathVariable Long id, @Valid @RequestBody UserDTO userDTO) {  
 UserDTO updatedUser = userService.updateUser(id, userDTO);  
 return ResponseEntity.*ok*(updatedUser);  
 }  
  
 @DeleteMapping("/{id}")  
 @PreAuthorize("hasRole('ADMIN')")  
 @Operation(summary = "Delete user by ID")  
 public ResponseEntity<Void> deleteUser(@PathVariable Long id) {  
 userService.deleteUser(id);  
 return ResponseEntity.*noContent*().build();  
 }  
}

## AuthController

package com.kingtan.users.controller;  
  
import com.kingtan.users.dto.LoginRequest;  
import com.kingtan.users.security.JwtUtil;  
import io.swagger.v3.oas.annotations.Operation;  
import io.swagger.v3.oas.annotations.tags.Tag;  
import jakarta.validation.Valid;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
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/v1/auth")  
@Tag(name = "Users", description = "Operations for login")  
public class AuthController {  
  
 private final AuthenticationManager authenticationManager;  
 private final JwtUtil jwtUtil;  
  
 @Autowired  
 public AuthController(AuthenticationManager authenticationManager, JwtUtil jwtUtil) {  
 this.authenticationManager = authenticationManager;  
 this.jwtUtil = jwtUtil;  
 }  
  
 @PostMapping("/login")  
 @Operation(summary = "User login")  
 public ResponseEntity<String> login(@Valid @RequestBody LoginRequest request) {  
 System.*out*.printf("request: %s\n", request);  
 Authentication authentication = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(request.username(), request.password()));  
 String token = jwtUtil.generateToken(authentication.getName());  
 System.*out*.printf("token: %s\n", token);  
 return ResponseEntity.*ok*(token);  
 }  
}

## PasswordResetController

package com.kingtan.users.controller;  
  
import com.kingtan.users.service.PasswordResetService;  
import io.swagger.v3.oas.annotations.Operation;  
import io.swagger.v3.oas.annotations.tags.Tag;  
import jakarta.mail.MessagingException;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
@RestController  
@RequestMapping("/api/v1/auth")  
@Tag(name = "Users", description = "Operations for password reset")  
public class PasswordResetController {  
  
 private final PasswordResetService passwordResetService;  
  
 @Autowired  
 public PasswordResetController(PasswordResetService passwordResetService) {  
 this.passwordResetService = passwordResetService;  
 }  
  
 @PostMapping("/password/reset")  
 @Operation(summary = "Request password reset")  
 public ResponseEntity<String> requestPasswordReset(@RequestParam String email) throws MessagingException {  
 passwordResetService.createPasswordResetToken(email);  
 return ResponseEntity.*ok*("Password reset email sent");  
 }  
  
 @PostMapping("/password/reset/confirm")  
 @Operation(summary = "Reset password confirm")  
 public ResponseEntity<String> resetPassword(@RequestParam String token, @RequestParam String newPassword) {  
 passwordResetService.resetPassword(token, newPassword);  
 return ResponseEntity.*ok*("Password reset successful");  
 }  
}

Security Configuration

JWT Authentication

Implement JWT-based authentication for secure access.

JWTUtility

package com.kingtan.users.security;  
  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
  
@Component  
public class JwtUtil {  
 @Value("${app.jwt.secret}")  
 private String jwtSecret;  
  
 @Value("${app.jwt.expiration}")  
 private long jwtExpiration;  
  
 public String generateToken(String username) {  
 return Jwts.*builder*()  
 .setSubject(username)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + jwtExpiration))  
 .signWith(SignatureAlgorithm.*HS512*, jwtSecret)  
 .compact();  
 }  
  
 public String getUsernameFromToken(String token) {  
 return Jwts.*parser*().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();  
 }  
  
 public boolean validateToken(String token) {  
 try {  
 Jwts.*parser*().setSigningKey(jwtSecret).parseClaimsJws(token);  
 return true;  
 } catch (Exception e) {  
 return false;  
 }  
 }  
}

### JWTAuthenticationFilter

java

package com.kingtan.users.security;  
  
import jakarta.servlet.FilterChain;  
import jakarta.servlet.ServletException;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;  
import org.springframework.stereotype.Component;  
import org.springframework.web.filter.OncePerRequestFilter;  
  
import java.io.IOException;  
  
@Component  
public class JwtAuthenticationFilter extends OncePerRequestFilter {  
  
 private final JwtUtil jwtUtil;  
 private final CustomUserDetailsService userDetailsService;  
  
 @Autowired  
 public JwtAuthenticationFilter(JwtUtil jwtUtil, CustomUserDetailsService userDetailsService) {  
 this.jwtUtil = jwtUtil;  
 this.userDetailsService = userDetailsService;  
 }  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  
 throws ServletException, IOException {  
 String path = request.getRequestURI();  
 if (path.equals("/api/v1/users/register") || path.equals("/api/v1/auth/login") ||  
 path.equals("/api/v1/auth/password/reset") || path.equals("/api/v1/auth/password/reset/confirm")) {  
 filterChain.doFilter(request, response);  
 return;  
 }  
 String header = request.getHeader("Authorization");  
 if (header != null && header.startsWith("Bearer ")) {  
 String token = header.substring(7);  
 try {  
 if (jwtUtil.validateToken(token)) {  
 String username = jwtUtil.getUsernameFromToken(token);  
 UserDetails userDetails = userDetailsService.loadUserByUsername(username);  
 UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(  
 userDetails, null, userDetails.getAuthorities());  
 SecurityContextHolder.*getContext*().setAuthentication(auth);  
 }  
 } catch (Exception e) {  
 logger.error("JWT authentication failed", e);  
 }  
 }  
 filterChain.doFilter(request, response);  
 }  
}

SecurityConfiguration

java

package com.kingtan.users.security;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;  
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;  
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;  
  
import static org.springframework.security.config.Customizer.*withDefaults*;  
  
@Configuration  
@EnableWebSecurity  
@EnableMethodSecurity(prePostEnabled = true)  
public class SecurityConfig {  
  
 private final JwtUtil jwtUtil;  
 private final CustomUserDetailsService userDetailsService;  
 private final JwtAuthenticationFilter jwtAuthenticationFilter;  
  
 public SecurityConfig(JwtUtil jwtUtil, CustomUserDetailsService userDetailsService, JwtAuthenticationFilter jwtAuthenticationFilter) {  
 this.jwtUtil = jwtUtil;  
 this.userDetailsService = userDetailsService;  
 this.jwtAuthenticationFilter = jwtAuthenticationFilter;  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 // Disable CSRF using Customizer  
 .csrf(csrf -> csrf.disable())  
 // Configure session management to be stateless  
 .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.*STATELESS*))  
 // Configure authorization rules  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/api/v1/users/register", "/api/v1/auth/login", "/api/v1/auth/password/reset", "/api/v1/auth/password/reset/confirm").permitAll()  
 .requestMatchers("/api/v1/users/\*\*").hasRole("USER")  
 .anyRequest().authenticated()  
 )  
 // Enable HTTP Basic authentication with defaults  
 .httpBasic(*withDefaults*())  
 // Add custom JWT filter  
 .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);  
  
 return http.build();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {  
 return config.getAuthenticationManager();  
 }  
}

## CustomUserDetailsService

java

package com.kingtan.users.security;  
  
import com.kingtan.users.model.User;  
import com.kingtan.users.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
import java.util.stream.Collectors;  
  
@Service  
public class CustomUserDetailsService implements UserDetailsService {  
  
 private final UserRepository userRepository;  
  
 @Autowired  
 public CustomUserDetailsService(UserRepository userRepository) {  
 this.userRepository = userRepository;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username)  
 .orElseThrow(() -> new UsernameNotFoundException("User not found: " + username));  
System.*out*.printf("CustomUserDetailsService.loadUserByUsername(%s)\n, user = %s\n", username, user);  
 return new org.springframework.security.core.userdetails.User(  
 user.getUsername(),  
 user.getPassword(),  
 user.isEnabled(),  
 true, // accountNonExpired  
 true, // credentialsNonExpired  
 true, // accountNonLocked  
 user.getRoles().stream()  
 .map(role -> new SimpleGrantedAuthority(role.getName()))  
 .collect(Collectors.*toList*())  
 );  
 }  
}

Password Management. Password ResetImplement a password reset feature with email verification.

* Add JavaMailSender Dependency:

xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-mail</artifactId>

</dependency>

* Configure Email in application.yml:

Properties

*##### Email configuration* mail:  
 host: smtp.gmail.com  
 port: 587  
 username: kingtan.farm@gmail.com  
 password: fcnc$30n  
 properties:  
 mail:  
 smtp:  
 auth: true  
 starttls:  
 enable: true

* Password Reset Token Entity:

java class: com.kingtan.users.model.PasswordResetToken

* Password Reset Service:

Java class: com.kingtan.users.service. PasswordResetService

* Password Reset Controller:

Java class: com.kingtan.users.controller. PasswordResetController.

Testing with Postman

## Register a User

To use the application, a user needs to register first. The info provided is stored in users database.

* + Endpoint: POST localhost:8080/api/v1/users/register
  + Example: bodies for 3 users

1.

{

    "username": "martin",

    "email": "testuser@example.com",

    "password": "hulunbeier09"

}

2.

{

    "username": "yuantan ",

    "email": "tanyuan@live.com",

    "password": "ribihz2009"

}

3.

{

    "username": "guojing",

    "email": "guojing37@hotmail.com",

    "password": "ribihz2009"

}

4.

{

    "username": "chunni",

    "email": "chunni24@hotmail.com",

    "password": "ribihz2009"

}

* + Response: 200 OK with user details. Example:

{

    "id": 4,

    "username": "chunni",

    "email": "chunni24@hotmail.com",

    "roles": [

        "ROLE\_USER"

    ]

}

## Login:

Use the username and password to login. Return a JWT token:

* + Endpoint: POST localhost:8080/api/v1/auth/login
  + Body:

json

{

    "username": "martin",

    "password": "hulunbeier09"

}

* + Response: 200 OK with JWT token. Example:

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJtYXJ0aW4iLCJpYXQiOjE3NTkyNjA5NzYsImV4cCI6MTc1OTM0NzM3Nn0.LnQh5kVJ-OP7OwdwqQO5K8xLADClaFBWZc\_ghRJv5EwH6XvftYybp96LwSE7SHd4eTvIXTjBLUTJNlyfE3tjnA

## Get one User Profile:

* + Endpoint: GET http://localhost:8080/api/v1/users/yuantan
  + Header: Authorization: Bearer <JWT\_TOKEN>. Example:

A screenshot of a computer

AI-generated content may be incorrect.

* + Response: 200 OK with user details:

{

    "id": 2,

    "username": "yuantan",

    "email": "tanyuan@live.com",

    "roles": [

        "ROLE\_USER"

    ]

}

## Get All Users (Admin only):

* + Endpoint: GET http://localhost:8080/api/v1/users
  + Header: Authorization: Bearer <JWT\_TOKEN>
  + Response: 200 OK with list of users.

## Password Reset Request:

(There are issues with email. Will update later)

* + Endpoint: POST /api/auth/password/reset?email=testuser@example.com
  + Response: 200 OK with confirmation message.

## Confirm Password Reset:

(There are issues with email. Will update later)

* + Endpoint: POST /api/auth/password/reset/confirm?token=<TOKEN>&newPassword=newpassword123
  + Response: 200 OK with success message.

# Unit Test

See code at GitHub: <https://github.com/expertmartin/kingtan/tree/main/api-users>