Spring Boot JWT Auth Backend (User & Admin) – Full Implementation

This is a **complete Spring Boot (3.x)** backend for JWT authentication with **User** and **Admin** roles, **registration**, **login**, and **role-based authorization** using **PostgreSQL**. It includes a clean folder structure, code for every class, and a step-by-step flow description. You can paste these files into a fresh Spring Boot project and run.

> Java 17+, Spring Boot 3.2/3.3+, Gradle or Maven. Example here uses **Maven**.

---

1) Folder Structure

```
src
└── main
    ├── java
    │   └── com/example/jwtapp
    │       ├── JwtAppApplication.java
    │       ├── config
    │       │   ├── SecurityConfig.java
    │       │   └── CorsConfig.java
    │       ├── security
    │       │   ├── JwtAuthFilter.java
    │       │   ├── JwtService.java
    │       │   └── CustomUserDetailsService.java
    │       ├── domain
    │       │   ├── Role.java
    │       │   └── User.java
    │       ├── dto
    │       │   ├── AuthRequest.java
    │       │   ├── AuthResponse.java
    │       │   └── RegisterRequest.java
    │       ├── repository
    │       │   └── UserRepository.java
    │       ├── service
    │       │   └── UserService.java
    │       ├── web
    │       │   ├── AuthController.java
    │       │   ├── UserController.java
    │       │   └── AdminController.java
    │       └── util
    │           └── ApiError.java
```

```
    └── resources
        ├── application.yml
        └── schema.sql   (optional: unique constraints / indices)
```

---

2) `pom.xml`

```
<?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 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>jwt-app</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>jwt-app</name>
    <description>Spring Boot JWT Auth (User & Admin)</description>

    <properties>
        <java.version>17</java.version>
        <spring.boot.version>3.3.2</spring.boot.version>
        <jjwt.version>0.11.5</jjwt.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring.boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</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-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>

    <!-- JWT -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-api</artifactId>
        <version>${jjwt.version}</version>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-impl</artifactId>
        <version>${jjwt.version}</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-jackson</artifactId>
        <version>${jjwt.version}</version>
        <scope>runtime</scope>
    </dependency>

    <!-- DB -->
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Lombok (optional) -->
    <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>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

---

3) `application.yml`

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/jwtapp
    username: postgres
    password: postgres
  jpa:
    hibernate:
      ddl-auto: update  # for dev; use validate in prod
    properties:
      hibernate:
        format_sql: true
    show-sql: true

app:
  jwt:
    secret: "change-me-to-a-very-long-random-string-change-me"
    expiration-ms: 3600000  # 1 hour

server:
  port: 8080
```

> **Note:** In production, keep the secret in an environment variable or a vault.

---

4) Domain Layer

`Role.java`

```
package com.example.jwtapp.domain;

public enum Role {
    USER, ADMIN
}
```

`User.java`

```
package com.example.jwtapp.domain;

import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import lombok.*;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.time.Instant;
import java.util.Collection;
import java.util.List;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "users", indexes = {
        @Index(name = "idx_users_email", columnList = "email", unique = true)
})
public class User implements UserDetails {

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

    @NotBlank
    private String fullName;

    @Email
```

```
    @Column(nullable = false, unique = true)
    private String email;

    @NotBlank
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Role role;

    private boolean enabled = true;

    private Instant createdAt;

    @PrePersist
    void onCreate() {
        createdAt = Instant.now();
    }

    // --- UserDetails ---
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority("ROLE_" + role.name()));
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() { return true; }
    @Override
    public boolean isAccountNonLocked() { return true; }
    @Override
    public boolean isCredentialsNonExpired() { return true; }
    @Override
    public boolean isEnabled() { return enabled; }
}
```

---

5) DTOs

`AuthRequest.java`

```
package com.example.jwtapp.dto;

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

public record AuthRequest(
        @Email String email,
        @NotBlank String password
) {}
```

`RegisterRequest.java`

```
package com.example.jwtapp.dto;

import com.example.jwtapp.domain.Role;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;

public record RegisterRequest(
        @NotBlank String fullName,
        @Email String email,
        @Size(min = 6, message = "Password must be at least 6 chars") String
password,
        Role role // optional; defaults to USER if null
) {}
```

`AuthResponse.java`

```
package com.example.jwtapp.dto;

import com.example.jwtapp.domain.Role;

public record AuthResponse(String token, Long userId, String fullName, String
email, Role role) {}
```

---

6) Repository

`UserRepository.java`

```
package com.example.jwtapp.repository;

import com.example.jwtapp.domain.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
    boolean existsByEmail(String email);
}
```

---

7) Security Layer

`JwtService.java`

```
package com.example.jwtapp.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.security.Key;
import java.util.Date;
import java.util.Map;
import java.util.function.Function;

@Service
public class JwtService {

    @Value("${app.jwt.secret}")
    private String secret;

    @Value("${app.jwt.expiration-ms}")
```

```
    private long jwtExpirationMs;

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver)
{
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails) {
        return generateToken(Map.of(), userDetails);
    }

    public String generateToken(Map<String, Object> extraClaims, UserDetails
userDetails) {
        return Jwts.builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() +
jwtExpirationMs))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();
    }

    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !
isTokenExpired(token));
    }

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

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
```

```
                .getBody();
    }

    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secret);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

> **Important:** `app.jwt.secret` must be **Base64-encoded**. For quick dev, you can set a long random string and base64 it. Example: `echo -n 'super-secret-very-long' | base64`.

`CustomUserDetailsService.java`

```
package com.example.jwtapp.security;

import com.example.jwtapp.repository.UserRepository;
import lombok.RequiredArgsConstructor;
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;

@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        return userRepository.findByEmail(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not
found: " + username));
    }
}
```

`JwtAuthFilter.java`

```
package com.example.jwtapp.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
```

```
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
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
@RequiredArgsConstructor
public class JwtAuthFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final CustomUserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws
ServletException, IOException {
        final String authHeader = request.getHeader("Authorization");
        final String jwt;
        final String userEmail;

        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwt = authHeader.substring(7);
        try {
            userEmail = jwtService.extractUsername(jwt);
        } catch (Exception e) {
            filterChain.doFilter(request, response);
            return;
        }

        if (userEmail != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails =
userDetailsService.loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new
```

```
UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities()
                );
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

SecurityConfig.java

```
package com.example.jwtapp.config;

import com.example.jwtapp.security.JwtAuthFilter;
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.configuration.AuthenticationConfigu
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
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
@RequiredArgsConstructor
@EnableMethodSecurity
public class SecurityConfig {

    private final JwtAuthFilter jwtAuthFilter;
    private final UserDetailsService userDetailsService;

    @Bean
```

```
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(sm ->
sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated()
            )
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthFilter,
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(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}
```

(Optional) `CorsConfig.java`

```
package com.example.jwtapp.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;
```

```
import java.util.List;

@Configuration
public class CorsConfig {
    @Bean
    public CorsFilter corsFilter() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOriginPatterns(List.of("*"));

config.setAllowedMethods(List.of("GET","POST","PUT","DELETE","PATCH","OPTIONS"));
        config.setAllowedHeaders(List.of("*"));
        config.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return new CorsFilter(source);
    }
}
```

---

8) Service Layer

`UserService.java`

```
package com.example.jwtapp.service;

import com.example.jwtapp.domain.Role;
import com.example.jwtapp.domain.User;
import com.example.jwtapp.dto.RegisterRequest;
import com.example.jwtapp.repository.UserRepository;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
```

```
    @Transactional
    public User register(RegisterRequest req) {
        if (userRepository.existsByEmail(req.email())) {
            throw new IllegalArgumentException("Email already registered");
        }
        Role role = req.role() == null ? Role.USER : req.role();
        User user = User.builder()
                .fullName(req.fullName())
                .email(req.email())
                .password(passwordEncoder.encode(req.password()))
                .role(role)
                .enabled(true)
                .build();
        return userRepository.save(user);
    }

    public List<User> findAll() {
        return userRepository.findAll();
    }
}
```

---

9) Web Layer (Controllers)

`AuthController.java`

```
package com.example.jwtapp.web;

import com.example.jwtapp.domain.User;
import com.example.jwtapp.dto.*;
import com.example.jwtapp.repository.UserRepository;
import com.example.jwtapp.security.JwtService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

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

```
@RequiredArgsConstructor
public class AuthController {

    private final AuthenticationManager authenticationManager;
    private final JwtService jwtService;
    private final com.example.jwtapp.service.UserService userService;
    private final UserRepository userRepository;

    @PostMapping("/register")
    public ResponseEntity<AuthResponse> register(@Valid @RequestBody
RegisterRequest request) {
        User saved = userService.register(request);
        String token = jwtService.generateToken(saved);
        return ResponseEntity.ok(new AuthResponse(token, saved.getId(),
saved.getFullName(), saved.getEmail(), saved.getRole()));
    }

    @PostMapping("/login")
    public ResponseEntity<AuthResponse> login(@Valid @RequestBody AuthRequest
request) {
        try {
            Authentication auth = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(request.email(),
request.password())
            );
            User principal = (User) auth.getPrincipal();
            String token = jwtService.generateToken(principal);
            return ResponseEntity.ok(new AuthResponse(token, principal.getId(),
principal.getFullName(), principal.getEmail(), principal.getRole()));
        } catch (BadCredentialsException ex) {
            throw new BadCredentialsException("Invalid email or password");
        }
    }

    @GetMapping("/me")
    public ResponseEntity<AuthResponse> me(@AuthenticationPrincipal User
current) {
        String token = jwtService.generateToken(current);
        return ResponseEntity.ok(new AuthResponse(token, current.getId(),
current.getFullName(), current.getEmail(), current.getRole()));
    }
}
```

`UserController.java`

```java
package com.example.jwtapp.web;

import com.example.jwtapp.domain.User;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

    @GetMapping("/profile")
    public User profile(@AuthenticationPrincipal User user) {
        return user;
    }

    @GetMapping("/hello")
    public String helloUser() {
        return "Hello, authenticated USER or ADMIN";
    }
}
```

`AdminController.java`

```java
package com.example.jwtapp.web;

import com.example.jwtapp.domain.User;
import com.example.jwtapp.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.access.prepost.PreAuthorize;
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/admin")
@RequiredArgsConstructor
public class AdminController {

    private final UserService userService;
```

```
    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/users")
    public List<User> allUsers() {
        return userService.findAll();
    }

    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/hello")
    public String helloAdmin() {
        return "Hello, ADMIN";
    }
}
```

---

10) Error Handling Helper (Optional)

`ApiError.java`

```
package com.example.jwtapp.util;

import org.springframework.http.HttpStatus;

import java.time.Instant;

public record ApiError(HttpStatus status, String message, Instant timestamp) {
    public static ApiError of(HttpStatus status, String message) {
        return new ApiError(status, message, Instant.now());
    }
}
```

(You can add a `@ControllerAdvice` if you want custom JSON error envelopes.)

---

11) Application Entry

`JwtAppApplication.java`

```
package com.example.jwtapp;

import com.example.jwtapp.domain.Role;
import com.example.jwtapp.domain.User;
import com.example.jwtapp.repository.UserRepository;
import org.springframework.boot.CommandLineRunner;
```

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.password.PasswordEncoder;

@SpringBootApplication
public class JwtAppApplication {

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

    // Seed one admin (dev only)
    @Bean
    CommandLineRunner seedAdmin(UserRepository repo, PasswordEncoder encoder) {
        return args -> {
            repo.findByEmail("admin@techcombo.in").orElseGet(() -> {
                User admin = User.builder()
                        .fullName("TechCombo Admin")
                        .email("admin@techcombo.in")
                        .password(encoder.encode("Admin@123"))
                        .role(Role.ADMIN)
                        .enabled(true)
                        .build();
                return repo.save(admin);
            });
        };
    }
}
```

---

12) How the Flow Works (Request → Response)

1. **Register** `POST /api/auth/register`
2. Payload: `{ fullName, email, password, role? }` (role optional; default USER)
3. Service encodes password (BCrypt), saves `User` with `Role.USER` or `Role.ADMIN`.
4. Returns **JWT** + user info.
5. **Login** `POST /api/auth/login`
6. Payload: `{ email, password }`
7. Spring `AuthenticationManager` authenticates using `DaoAuthenticationProvider` + `UserDetailsService`.
8. On success, generates JWT (subject = email) and returns.

9. **Authenticated Requests**

10. Client sends header: `Authorization: Bearer <token>`.
11. `JwtAuthFilter` extracts token, validates, sets `SecurityContext`.
12. Controllers access current user via `@AuthenticationPrincipal` or method security.

13. **Role-Based Access**

14. `@PreAuthorize("hasRole('ADMIN')")` guards admin endpoints.
15. Other endpoints require authentication (configured in `SecurityConfig`).

---

13) Test Quickly with cURL (or Postman)

```
# 1) Register a user
curl -X POST http://localhost:8080/api/auth/register
  -H "Content-Type: application/json"
  -d '{"fullName":"Achal
Turkar","email":"achal@example.com","password":"Secret@123"}'

# 2) Register an admin (dev)
curl -X POST http://localhost:8080/api/auth/register
  -H "Content-Type: application/json"
  -d '{"fullName":"Admin
One","email":"admin1@example.com","password":"Admin@123","role":"ADMIN"}'

# 3) Login
curl -X POST http://localhost:8080/api/auth/login
  -H "Content-Type: application/json"
  -d '{"email":"achal@example.com","password":"Secret@123"}'

# 4) Authenticated user endpoint
curl http://localhost:8080/api/users/profile
  -H "Authorization: Bearer <PASTE_TOKEN>"

# 5) Admin-only endpoint
curl http://localhost:8080/api/admin/users
  -H "Authorization: Bearer <ADMIN_TOKEN>"
```

---

14) Notes & Production Tips

- Change `app.jwt.secret` to a **Base64** string from a strong random key.
- Set `ddl-auto: validate` and manage schema with migrations (Flyway/Liquibase).

- Consider **refresh tokens** and **logout/blacklist** for long-lived sessions.
- Add a `@ControllerAdvice` to wrap errors consistently.
- Lock down admin registration in production (e.g., only admins can create admins, or via seeding/console).
- Always hash passwords (BCrypt) and never log them.

---

15) Endpoints Summary

- `POST /api/auth/register` – Register USER or ADMIN, returns JWT.
- `POST /api/auth/login` – Login, returns JWT.
- `GET /api/auth/me` – Returns current user info + a fresh token.
- `GET /api/users/profile` – Any authenticated user.
- `GET /api/users/hello` – Any authenticated user (smoke test).
- `GET /api/admin/users` – **ADMIN only** list of users.
- `GET /api/admin/hello` – **ADMIN only**.

You now have a clean, production-ready base to plug into your Next.js/React frontend.