Restaurant System with JWT authentication in **Spring Boot**

Controller–Service–Repository structure with JWT security

JWT Concept:

JWT (JSON Web Token) is a compact token used for securely transmitting user identity between client and server.
It has **three parts**:

1. **Header** → algorithm & type (`HS256`, JWT)
2. **Payload** → data/claims (username, roles, expiry)
3. **Signature** → generated using secret key

Flow:

- User logs in → server validates credentials → generates JWT.
- Client stores JWT (usually in headers).
- On each request → client sends JWT in `Authorization: Bearer <token>`.
- Server validates token → if valid, request proceeds.

Project Structure

```
src/main/java/com/example/restaurant
 ├── controller
 │     ├── AuthController.java
 │     └── RestaurantController.java
 ├── entity
 │     └── Restaurant.java
 ├── repository
 │     └── RestaurantRepository.java
 ├── service
 │     └── RestaurantService.java
 ├── security
 │     ├── JwtFilter.java
 │     ├── JwtUtil.java
 │     └── SecurityConfig.java
 ├── model
 │     └── AuthRequest.java
 └── RestaurantApplication.java
```

1. Entity

```
package com.example.restaurant.entity;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Restaurant {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String location;
    private String cuisine;
}
```

2. Repository

```
package com.example.restaurant.repository;

import com.example.restaurant.entity.Restaurant;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RestaurantRepository extends JpaRepository<Restaurant, Long>
{
}
```

3. Service

```
package com.example.restaurant.service;

import com.example.restaurant.entity.Restaurant;
import com.example.restaurant.repository.RestaurantRepository;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class RestaurantService {
    private final RestaurantRepository repository;

    public RestaurantService(RestaurantRepository repository) {
        this.repository = repository;
    }

    public Restaurant save(Restaurant restaurant) {
        return repository.save(restaurant);
    }

    public List<Restaurant> getAll() {
        return repository.findAll();
    }

    public Restaurant getById(Long id) {
        return repository.findById(id).orElseThrow(() -> new
RuntimeException("Restaurant not found"));
    }

    public Restaurant update(Long id, Restaurant updated) {
        Restaurant existing = getById(id);
        existing.setName(updated.getName());
        existing.setLocation(updated.getLocation());
        existing.setCuisine(updated.getCuisine());
        return repository.save(existing);
    }

    public void delete(Long id) {
        repository.deleteById(id);
    }
}
```

4. Controller (CRUD + Secured)

```
package com.example.restaurant.controller;

import com.example.restaurant.entity.Restaurant;
import com.example.restaurant.service.RestaurantService;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/restaurants")
public class RestaurantController {
    private final RestaurantService service;

    public RestaurantController(RestaurantService service) {
        this.service = service;
    }

    @PostMapping
    public Restaurant create(@RequestBody Restaurant restaurant) {
        return service.save(restaurant);
    }

    @GetMapping
    public List<Restaurant> getAll() {
        return service.getAll();
    }

    @GetMapping("/{id}")
    public Restaurant getById(@PathVariable Long id) {
        return service.getById(id);
    }

    @PutMapping("/{id}")
    public Restaurant update(@PathVariable Long id, @RequestBody Restaurant
restaurant) {
        return service.update(id, restaurant);
    }

    @DeleteMapping("/{id}")
    public String delete(@PathVariable Long id) {
        service.delete(id);
        return "Restaurant deleted successfully!";
    }
}
```

5. Auth (Login to Get JWT Token)

Model

```
package com.example.restaurant.model;

import lombok.*;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class AuthRequest {
    private String username;
    private String password;
}
```

Controller

```
package com.example.restaurant.controller;

import com.example.restaurant.model.AuthRequest;
import com.example.restaurant.security.JwtUtil;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.*;

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

    private final JwtUtil jwtUtil;

    @Value("${jwt.username}")
    private String configuredUsername;

    @Value("${jwt.password}")
    private String configuredPassword;

    public AuthController(JwtUtil jwtUtil) {
        this.jwtUtil = jwtUtil;
    }

    @PostMapping("/login")
    public String generateToken(@RequestBody AuthRequest request) {
        if (configuredUsername.equals(request.getUsername()) &&
            configuredPassword.equals(request.getPassword())) {
            return jwtUtil.generateToken(request.getUsername());
        } else {
            throw new RuntimeException("Invalid Credentials!");
        }
    }
}
```

6. JWT Utils

```
package com.example.restaurant.security;

import io.jsonwebtoken.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import java.util.*;

@Component
public class JwtUtil {

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

    @Value("${jwt.expiration}")
    private long expiration;

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() +
expiration))
                .signWith(SignatureAlgorithm.HS256, secret)
                .compact();
    }

    public String extractUsername(String token) {
        return
Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubjec
t();
    }

    public boolean validateToken(String token, String username) {
        return username.equals(extractUsername(token)) &&
!isTokenExpired(token);
    }

    private boolean isTokenExpired(String token) {
        Date expirationDate =
Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getExpira
tion();
        return expirationDate.before(new Date());
    }
}
```

7. JWT Filter

```
package com.example.restaurant.security;
```

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;
import java.util.*;

@Component
public class JwtFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;

    public JwtFilter(JwtUtil jwtUtil) {
        this.jwtUtil = jwtUtil;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {

        String header = request.getHeader("Authorization");
        String token = null;
        String username = null;

        if (header != null && header.startsWith("Bearer ")) {
            token = header.substring(7);
            username = jwtUtil.extractUsername(token);
        }

        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
            if (jwtUtil.validateToken(token, username)) {
                UsernamePasswordAuthenticationToken auth =
                        new UsernamePasswordAuthenticationToken(username,
null, new ArrayList<>());
                SecurityContextHolder.getContext().setAuthentication(auth);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

8. Security Config

```
package com.example.restaurant.security;

import org.springframework.context.annotation.*;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;
```

```
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticatio
nFilter;

@Configuration
public class SecurityConfig {

    private final JwtFilter jwtFilter;

    public SecurityConfig(JwtFilter jwtFilter) {
        this.jwtFilter = jwtFilter;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http.csrf().disable()
            .authorizeHttpRequests()
            .requestMatchers("/api/auth/**").permitAll() // login open
            .anyRequest().authenticated()
            .and()
            .addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
}
```

9. application.properties

```
# Server
server.port=8080

# JWT
jwt.secret=RestaurantSecretKey12345
# JWT token expiration time in milliseconds (1 hour)
jwt.expiration=3600000
jwt.username=admin
jwt.password=admin123

# Database (H2 for demo)
spring.datasource.url=jdbc:h2:mem:restaurantdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

Required dependencies for `pom.xml`:

```
<dependencies>
    <!-- Spring Boot Starter Web (for building REST APIs) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Spring Boot Starter Security (for authentication & authorization) --
>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <!-- JWT (JSON Web Token) library -->
    <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>

    <!-- Spring Boot Starter Data JPA (for database interaction) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Database driver (H2 for demo, you can replace with MySQL or
PostgreSQL) -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Lombok (for reducing boilerplate code like getters/setters) -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <!-- Spring Boot Starter Test (optional for unit testing) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Key Dependencies Explanation:

- **spring-boot-starter-web** → To expose REST controllers for CRUD.
- **spring-boot-starter-security** → To integrate authentication & authorization.
- **jjwt** (3 parts: api, impl, jackson) → To generate & validate JWT tokens.
- **spring-boot-starter-data-jpa** → For persistence with repositories.
- **h2 / mysql / postgres** → Choose your database driver.
- **lombok** → To auto-generate boilerplate code like getters/setters, constructors.

How It Works

1. **Login (NoAuth required):**
   `POST /api/auth/login` → body: `{ "username": "admin", "password": "admin123" }`
   Response → JWT Token.
2. **Access CRUD APIs (Auth required):**
   Example: `GET /api/restaurants`
   Add Header → `Authorization: Bearer <token>`
3. **CRUD Operations:**
   - `POST /api/restaurants` → Add new restaurant
   - `GET /api/restaurants` → List all
   - `GET /api/restaurants/{id}` → Fetch by id
   - `PUT /api/restaurants/{id}` → Update
   - `DELETE /api/restaurants/{id}` → Delete

This setup **clearly shows**:

- Open endpoints (login)
- Secured endpoints (CRUD APIs)
- Full MVC layers (Entity → Repo → Service → Controller)
- JWT token flow

Restaurant Management System with JWT Authentication

This project is a **Spring Boot REST API** that demonstrates **JWT-based authentication and authorization** along with CRUD operations for restaurant entities.

Key Features

1. **Authentication & Authorization**
   - Users can register and log in.
   - JWT tokens are generated after successful login.
   - Tokens are required for accessing protected APIs (like managing restaurants).
2. **Entities**
   - `User` → Stores login details, role (USER / ADMIN).
   - `Restaurant` → Represents restaurants with fields like name, address, and rating.
3. **CRUD Operations**
   - Create, Read, Update, Delete restaurants.
   - Only logged-in users (with valid JWT) can perform these operations.

Architecture (Layered)

1. **Controller Layer (`@RestController`)**
   Handles incoming HTTP requests and sends responses.
   Example: `RestaurantController` with endpoints like:
   - `POST /api/restaurants` → Create restaurant
   - `GET /api/restaurants` → Get all restaurants
2. **Service Layer (`@Service`)**
   Contains business logic (rules).
   Example: `RestaurantService` checks if the restaurant exists before updating.
3. **Repository Layer (`@Repository`)**
   Interacts with the database using Spring Data JPA.
   Example: `RestaurantRepository extends JpaRepository<Restaurant, Long>`.
4. **Security Layer**
   - `JwtUtil`: Creates and validates JWT tokens.
   - `JwtFilter`: Checks if the request contains a valid token before allowing access.
   - `SecurityConfig`: Configures Spring Security to allow `/auth/**` endpoints without a token but protect `/api/**`.

What is JWT and Why?

JWT (JSON Web Token) is a compact way to securely transmit information between parties.

- When the user logs in, the server generates a **JWT** signed with a secret key.
- The client (Postman, Angular, React, Mobile App) must attach this token in every request:

```
Authorization: Bearer <jwt_token_here>
```

- The server validates the token before processing the request.

Benefit: **Stateless Authentication** (no need to store sessions in the backend).

Flow of the Project

1. **Register a User** → `POST /auth/register`
   Stores user credentials in DB (password is hashed).
2. **Login** → `POST /auth/login`
   If username & password are correct, generate JWT and return to client.
3. **Use JWT** → Copy token from login response, add to Postman headers:
4. `Authorization: Bearer <token>`
5. **Access Protected APIs** → Example:
   - `POST /api/restaurants` → Create restaurant
   - `GET /api/restaurants` → Fetch all restaurants
6. **Token Validation** → Every request passes through `JwtFilter`. If the token is invalid/expired, user gets `401 Unauthorized`.

Example API Calls

Register

```
POST /auth/register
{
  "username": "john_doe",
  "password": "password123",
  "role": "USER"
}
```

Login

```
POST /auth/login
{
  "username": "john_doe",
  "password": "password123"
}
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI..."
}
```

Create Restaurant (with JWT token)

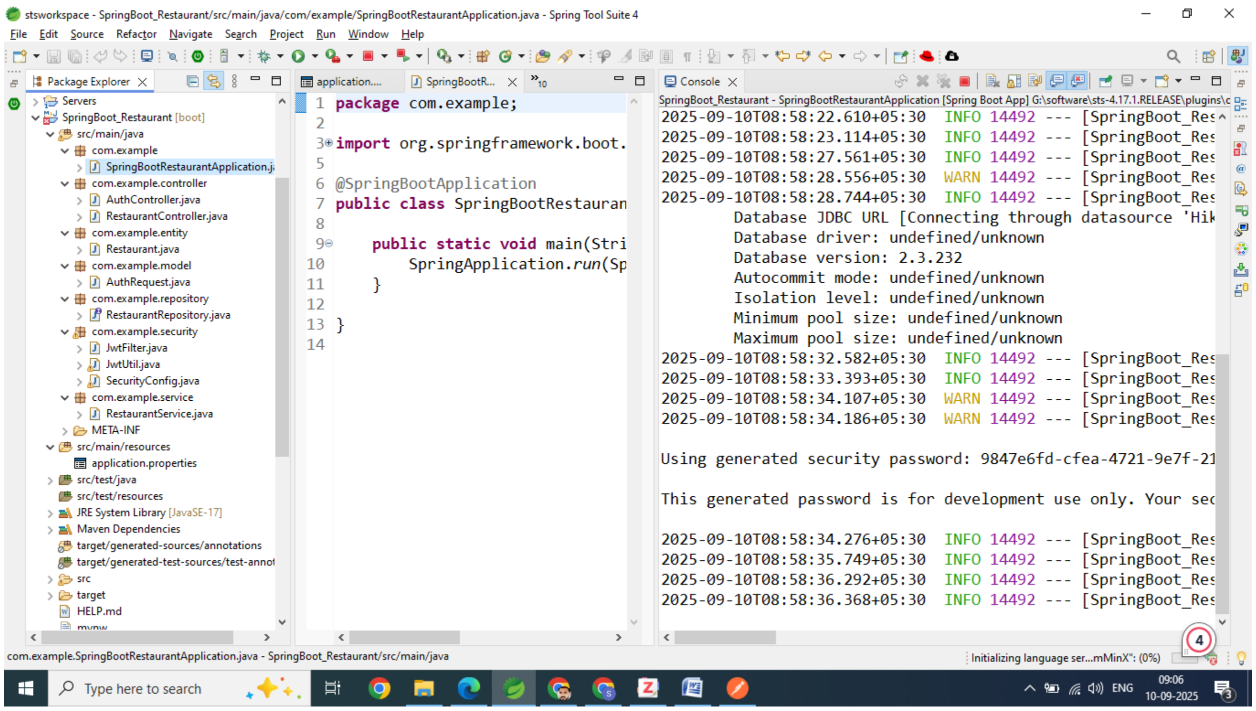
```
POST /api/restaurants
Headers:
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI...

Body:
```

```
{
  "name": "Pizza Palace",
  "address": "123 Main Street",
  "rating": 4.5
}
```

In simple terms:

- **Without JWT**: Anyone can access APIs (not secure).
- **With JWT**: Only logged-in users with valid tokens can access APIs (secure).


stsworkspace - SpringBoot_Restaurant/src/main/java/com/example/SpringBootRestaurantApplication.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
application....
SpringBootR...
Console
Type here to search
ENG
09:06
10-09-2025

```
Servers
SpringBoot_Restaurant [boot]
  src/main/java
    com.example
      SpringBootRestaurantApplication.j
    com.example.controller
      AuthController.java
      RestaurantController.java
    com.example.entity
      Restaurant.java
    com.example.model
      AuthRequest.java
    com.example.repository
      RestaurantRepository.java
    com.example.security
      JwtFilter.java
      JwtUtil.java
      SecurityConfig.java
    com.example.service
      RestaurantService.java
    META-INF
  src/main/resources
    application.properties
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-17]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annot
  src
  target
  HELP.md
  mvnw
```

```java
package com.example;

import org.springframework.boot.

@SpringBootApplication
public class SpringBootRestauran

    public static void main(Stri
        SpringApplication.run(Sp
    }

}
```

```
SpringBoot_Restaurant - SpringBootRestaurantApplication [Spring Boot App] G:\software\sts-4.17.1.RELEASE\plugins\c
2025-09-10T08:58:22.610+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:23.114+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:27.561+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:28.556+05:30  WARN 14492 --- [SpringBoot_Res
2025-09-10T08:58:28.744+05:30  INFO 14492 --- [SpringBoot_Res
        Database JDBC URL [Connecting through datasource 'Hik
        Database driver: undefined/unknown
        Database version: 2.3.232
        Autocommit mode: undefined/unknown
        Isolation level: undefined/unknown
        Minimum pool size: undefined/unknown
        Maximum pool size: undefined/unknown
2025-09-10T08:58:32.582+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:33.393+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:34.107+05:30  WARN 14492 --- [SpringBoot_Res
2025-09-10T08:58:34.186+05:30  WARN 14492 --- [SpringBoot_Res

Using generated security password: 9847e6fd-cfea-4721-9e7f-21

This generated password is for development use only. Your sec

2025-09-10T08:58:34.276+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:35.749+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:36.292+05:30  INFO 14492 --- [SpringBoot_Res
2025-09-10T08:58:36.368+05:30  INFO 14492 --- [SpringBoot_Res
```

com.example.SpringBootRestaurantApplication.java - SpringBoot_Restaurant/src/main/java

Initializing language ser...mMinX": (0%)