Create authentication service that returns JWT

As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.

Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.

Request

```
curl -s -u user:pwd http://localhost:8090/authenticate
```

Response

```
{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE
1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}
```

This can be incorporated as three major steps:

- Create authentication controller and configure it in SecurityConfig
- Read Authorization header and decode the username and password
- Generate token based on the user retrieved in the previous step

Let incorporate the above as separate hands on exercises.

Implementation

Step 1: Create a Spring Boot Project

Use Spring Initializr or your IDE (like Spring Tool Suite or IntelliJ) to create a Maven-based Spring Boot project with the following dependencies:

- spring-boot-starter-web
- spring-boot-starter-security
- jjwt-api
- jjwt-impl
- jjwt-jackson

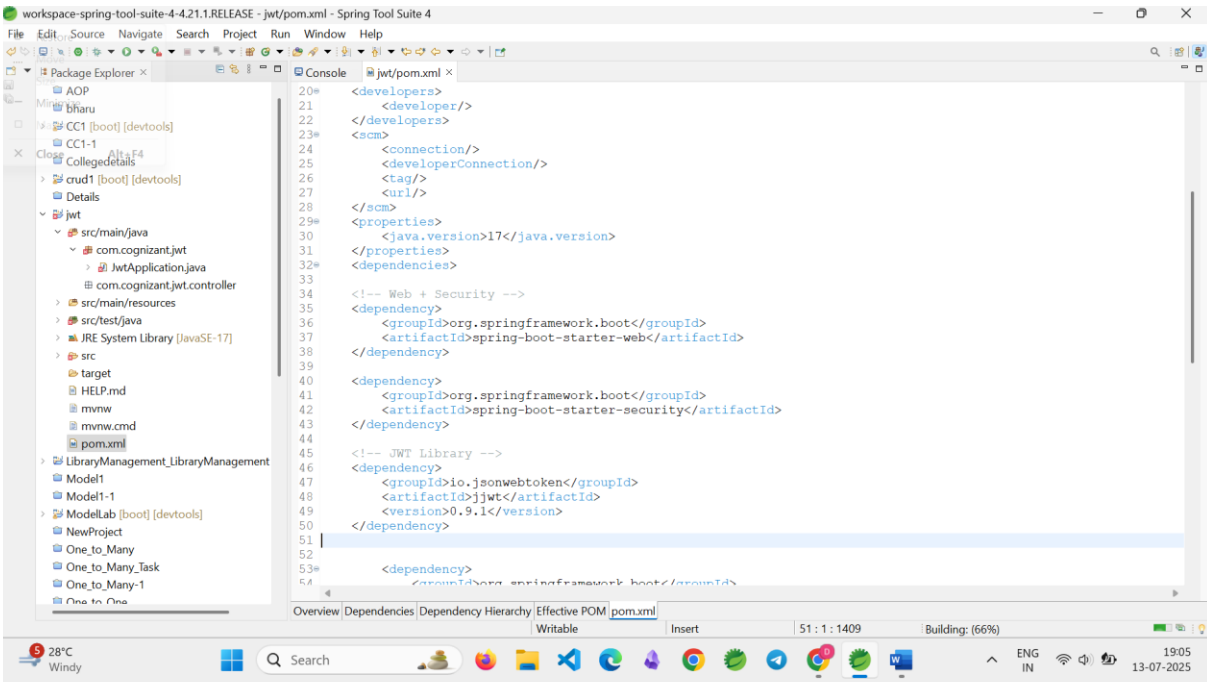
Sample pom.xml Dependencies

xml

CopyEdit

```xml
<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>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>
</dependencies>
```


workspace-spring-tool-suite-4-4.21.1.RELEASE - jwt/pom.xml - Spring Tool Suite 4

```
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>

    <!-- Web + Security -->
    <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>

    <!-- JWT Library -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>


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

Step 2: Configure Application Port

Open src/main/resources/application.properties and add the following line:

```
server.port=8091
```

This sets the application to run on port 8091.

Step 3: Create the Main Application Class

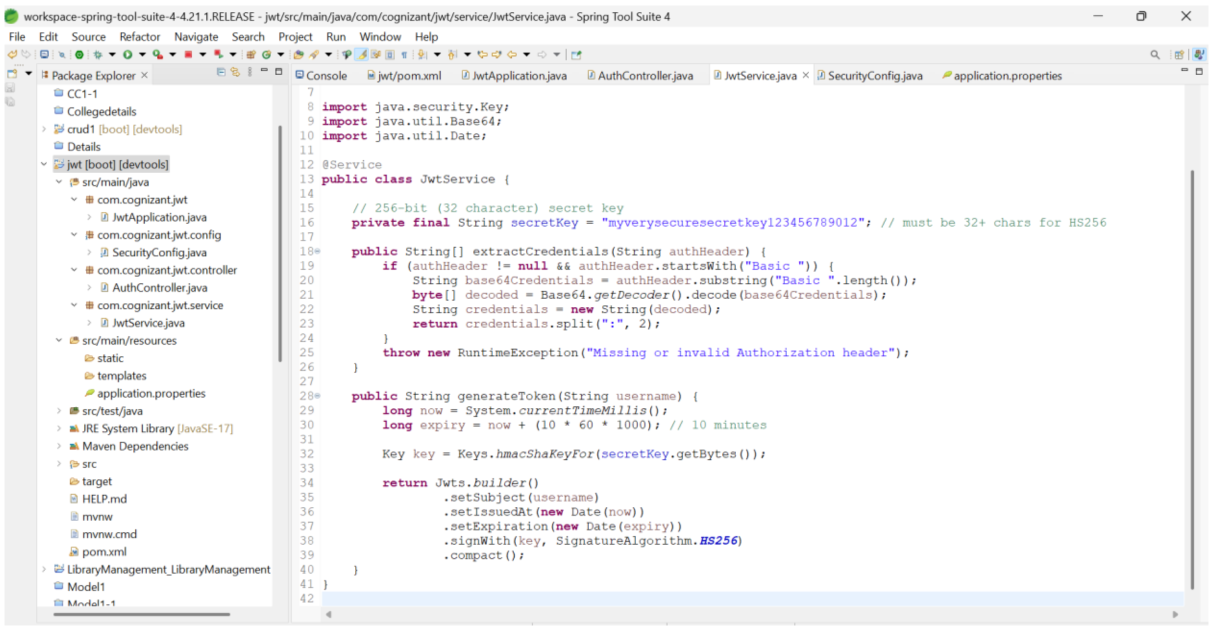
Create a main class named JwtApplication.java in the base package:

```
@SpringBootApplication
public class JwtApplication {
    public static void main(String[] args) {
        SpringApplication.run(JwtApplication.class, args);
    }
}
```

Step 4: Create the JWT Service

Create a class JwtService.java under com.cognizant.jwt.service.

```
@Service
public class JwtService {

    private final String secretKey = "myverysecuresecretkey123456789012";

    public String[] extractCredentials(String authHeader) {
        if (authHeader != null && authHeader.startsWith("Basic ")) {
            String base64Credentials = authHeader.substring("Basic ".length());
            byte[] decoded = Base64.getDecoder().decode(base64Credentials);
            String credentials = new String(decoded);
            return credentials.split(":", 2);
        }
        throw new RuntimeException("Missing or invalid Authorization header");
    }

    public String generateToken(String username) {
        long now = System.currentTimeMillis();
        long expiry = now + (10 * 60 * 1000);

        Key key = Keys.hmacShaKeyFor(secretKey.getBytes());
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date(now))
                .setExpiration(new Date(expiry))
                .signWith(key, SignatureAlgorithm.HS256)
                .compact();
    }
}
```



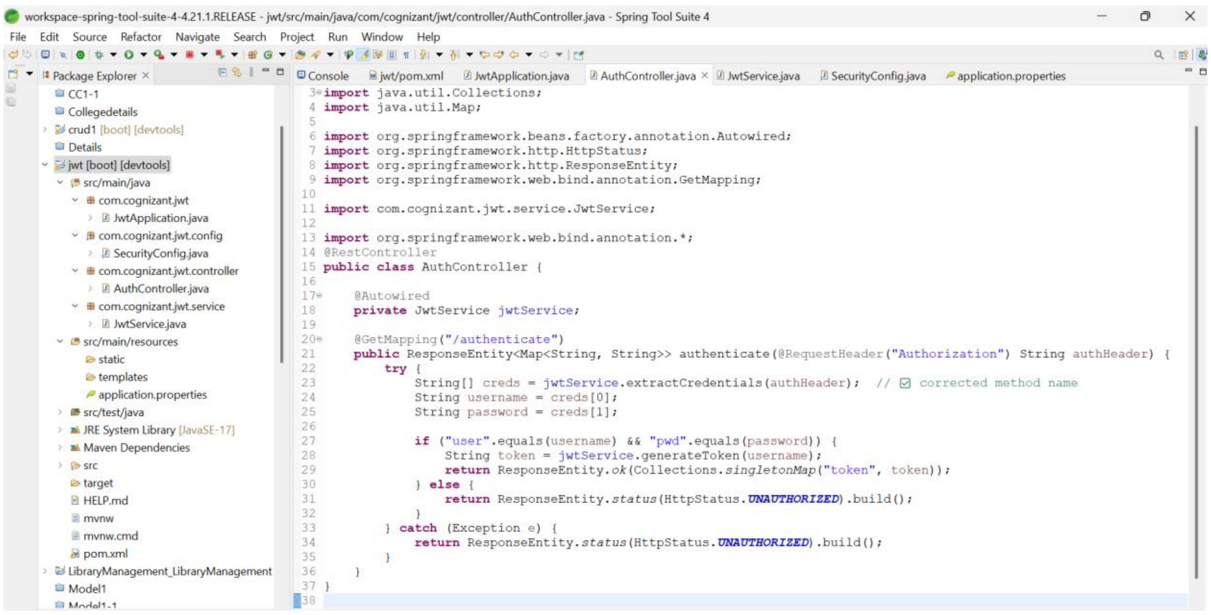
```

import java.security.Key;
import java.util.Base64;
import java.util.Date;

@Service
public class JwtService {

    // 256-bit (32 character) secret key
    private final String secretKey = "myverysecuresecretkey123456789012"; // must be 32+ chars for HS256

    public String[] extractCredentials(String authHeader) {
        if (authHeader != null && authHeader.startsWith("Basic ")) {
            String base64Credentials = authHeader.substring("Basic ".length());
            byte[] decoded = Base64.getDecoder().decode(base64Credentials);
            String credentials = new String(decoded);
            return credentials.split(":", 2);
        }
        throw new RuntimeException("Missing or invalid Authorization header");
    }

    public String generateToken(String username) {
        long now = System.currentTimeMillis();
        long expiry = now + (10 * 60 * 1000); // 10 minutes

        Key key = Keys.hmacShaKeyFor(secretKey.getBytes());

        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date(now))
                .setExpiration(new Date(expiry))
                .signWith(key, SignatureAlgorithm.HS256)
                .compact();
    }
}
```

Step 5: Create the Authentication Controller

Create a class named AuthController.java under com.cognizant.jwt.controller.

```
@RestController
public class AuthController {


    @Autowired
    private JwtService jwtService;
```

```java
    @GetMapping("/authenticate")
    public ResponseEntity<Map<String, String>>
authenticate(@RequestHeader("Authorization") String authHeader) {
        try {
            String[] creds = jwtService.extractCredentials(authHeader);
            String username = creds[0];
            String password = creds[1];

            if ("user".equals(username) && "pwd".equals(password)) {
                String token = jwtService.generateToken(username);
                return ResponseEntity.ok(Collections.singletonMap("token", token));
            } else {
                return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
            }
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }
}
```



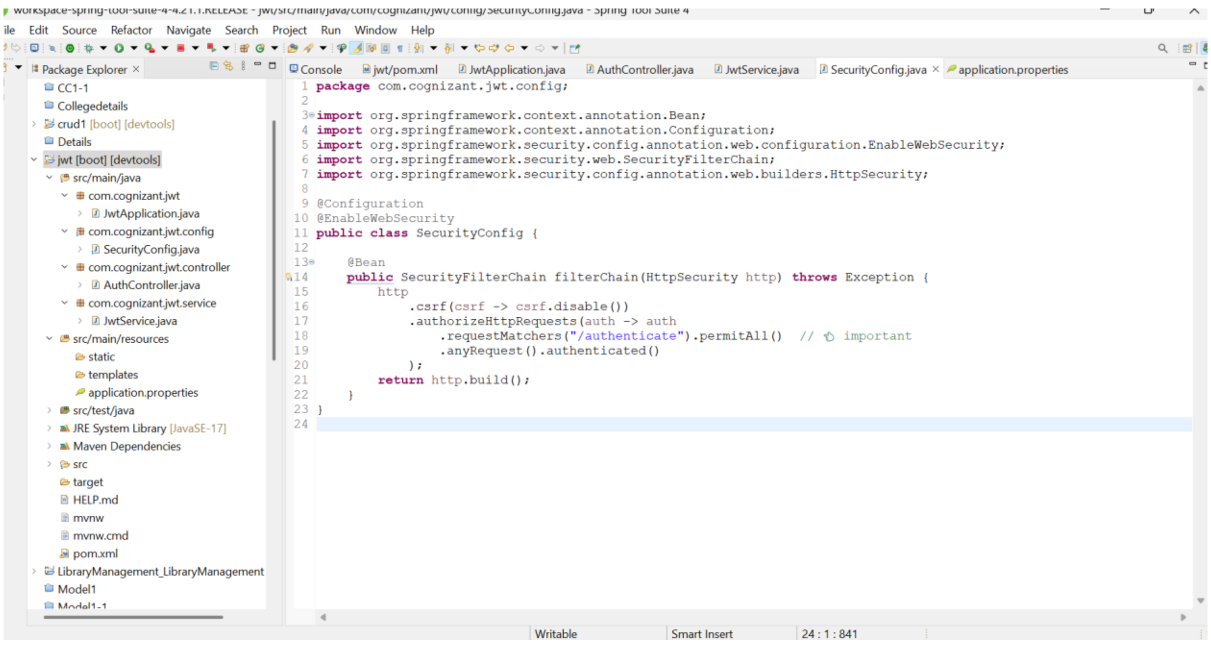
```java
import java.util.Collections;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;

import com.cognizant.jwt.service.JwtService;

import org.springframework.web.bind.annotation.*;
@RestController
public class AuthController {

    @Autowired
    private JwtService jwtService;

    @GetMapping("/authenticate")
    public ResponseEntity<Map<String, String>> authenticate(@RequestHeader("Authorization") String authHeader) {
        try {
            String[] creds = jwtService.extractCredentials(authHeader);  // ☑ corrected method name
            String username = creds[0];
            String password = creds[1];

            if ("user".equals(username) && "pwd".equals(password)) {
                String token = jwtService.generateToken(username);
                return ResponseEntity.ok(Collections.singletonMap("token", token));
            } else {
                return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
            }
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }
}

```

Step 6: Configure Spring Security

Create a configuration class SecurityConfig.java under com.cognizant.jwt.config.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/authenticate").permitAll()
                .anyRequest().authenticated());
        return http.build();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        return username -> null;
    }
}
```

This disables CSRF, allows access to the /authenticate endpoint without authentication, and disables Spring Boot’s default user authentication.



```
package com.cognizant.jwt.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/authenticate").permitAll()  // 👈 important
                .anyRequest().authenticated()
            );
        return http.build();
    }
}
```

Step 7: Test the JWT Authentication Endpoint

Use curl or Postman to test the /authenticate endpoint.

```
curl -u user:pwd http://localhost:8091/authenticate
```

Expected Output:

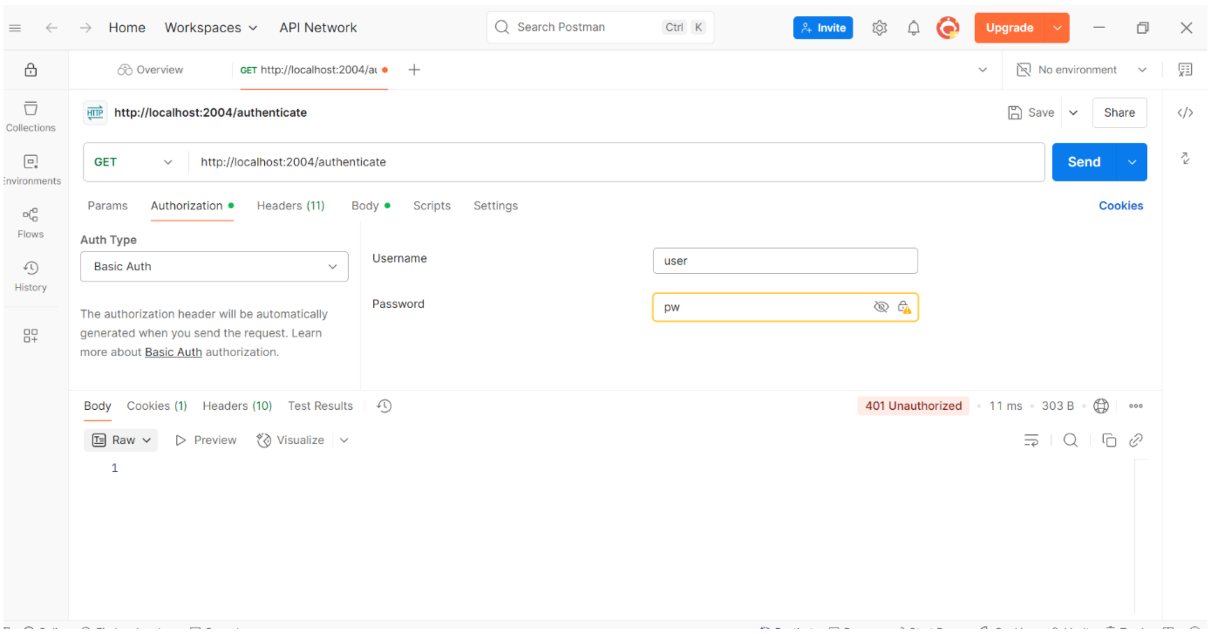
```
{
  "token": "eyJhbGciOiJIUzI1NiJ9..."
}
```

Postman:

- Method: GET
- URL: http://localhost:8091/authenticate
- Authorization tab: Basic Auth (username = user, password = pwd)

If password wrong

    401 error Unauthorized



http://localhost:2004/authenticate

GET http://localhost:2004/authenticate

Params · Authorization · Headers (11) · Body · Scripts · Settings

Auth Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

Username: user

Password: pw

Body · Cookies (1) · Headers (10) · Test Results

401 Unauthorized · 11 ms · 303 B

Raw · Preview · Visualize