Lab 10

Part A:

Copy and Paste Lab 9 Part A from the previous lab where you did the Book Controller application. Now you will add basic authentication security to the book controller.

The first thing you need to add is the spring security stater dependency to the project in its POM file:

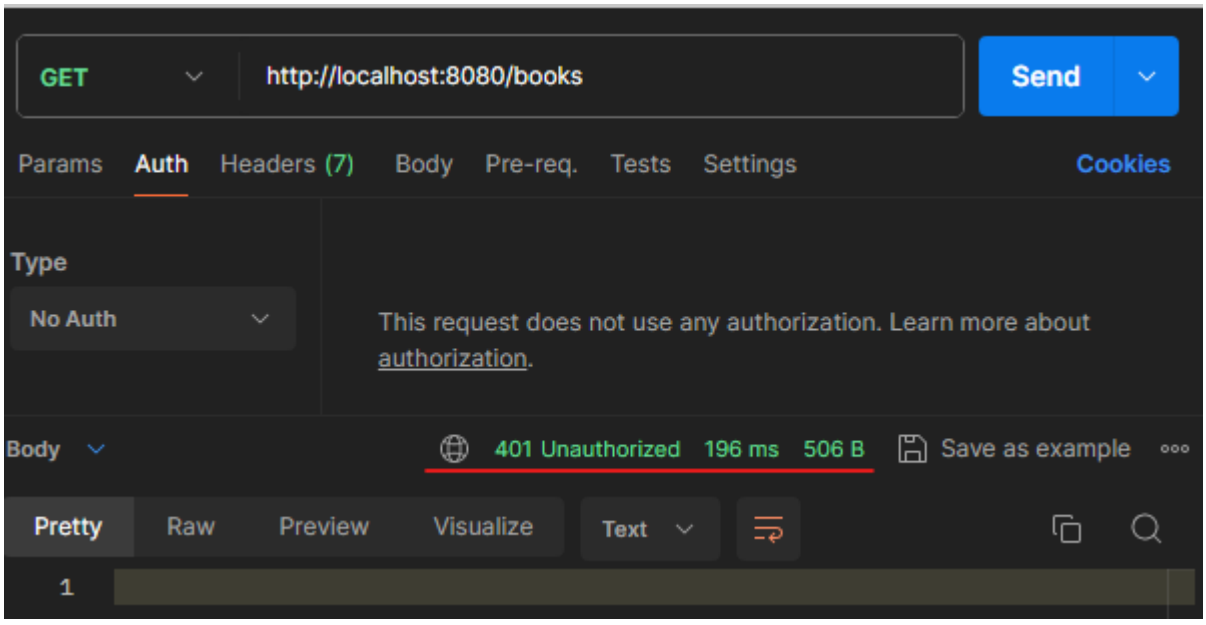
```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

When you run the application, you should see this log:

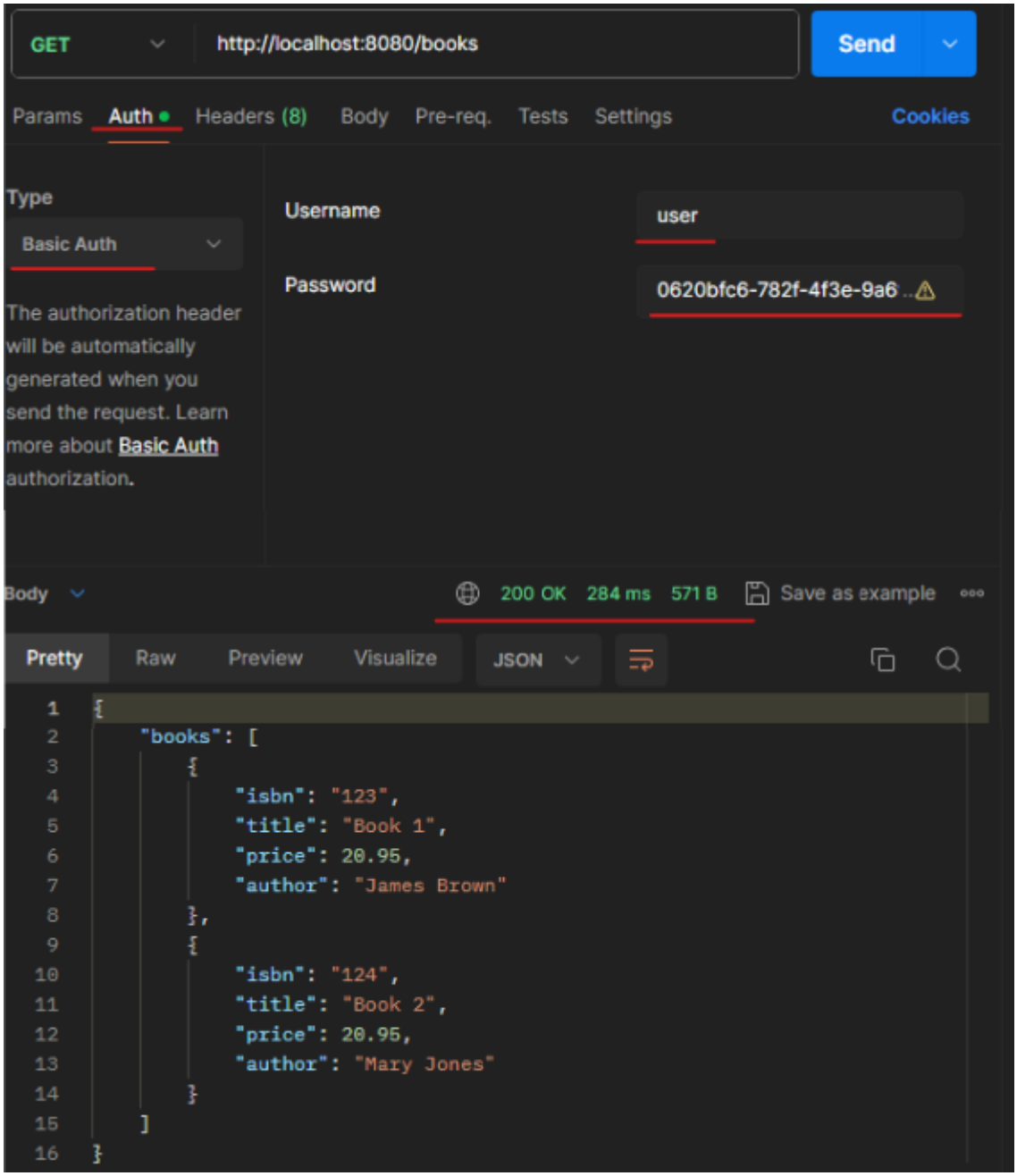
“Using generated security password: 0620bfc6-782f-4f3e-9a69-1086472425de

This generated password is for development use only. Your security configuration must be updated before running your application in production.”

If you try to call the list of Books from your Postman, you will receive an unauthorized response:



To be able to get the list of books you need to authenticate now against the book controller application using the given password Spring Boot prints in the console for you:



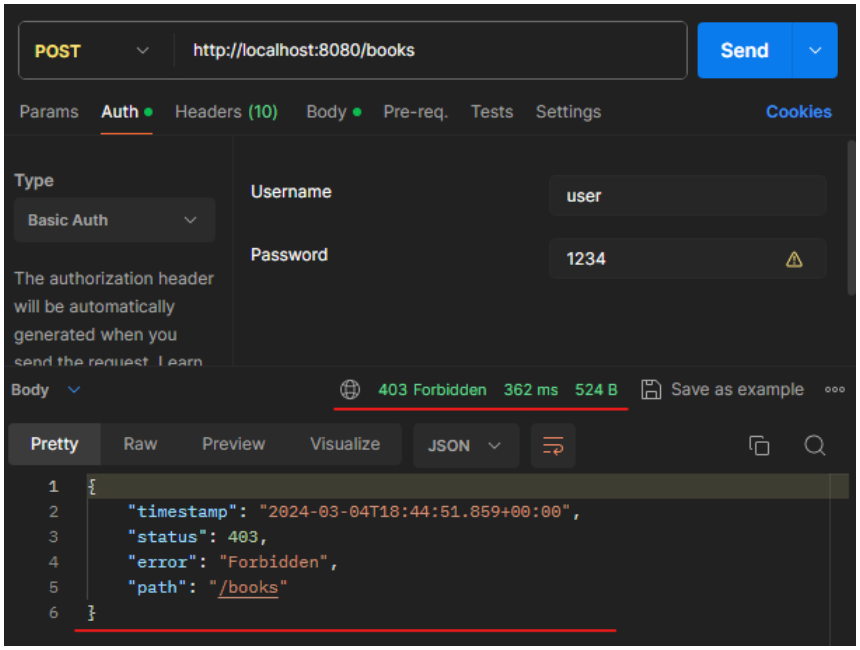
For the first part of lab A, you need to create a basic authentication user, and get rid of the basic authentication Spring provides by default. For this, you need to create your own **WebSecurityConfig** class where you create your user in memory creating a **UserDetailService** **@Bean** with a "**InMemoryUserDetailsManager**" object.

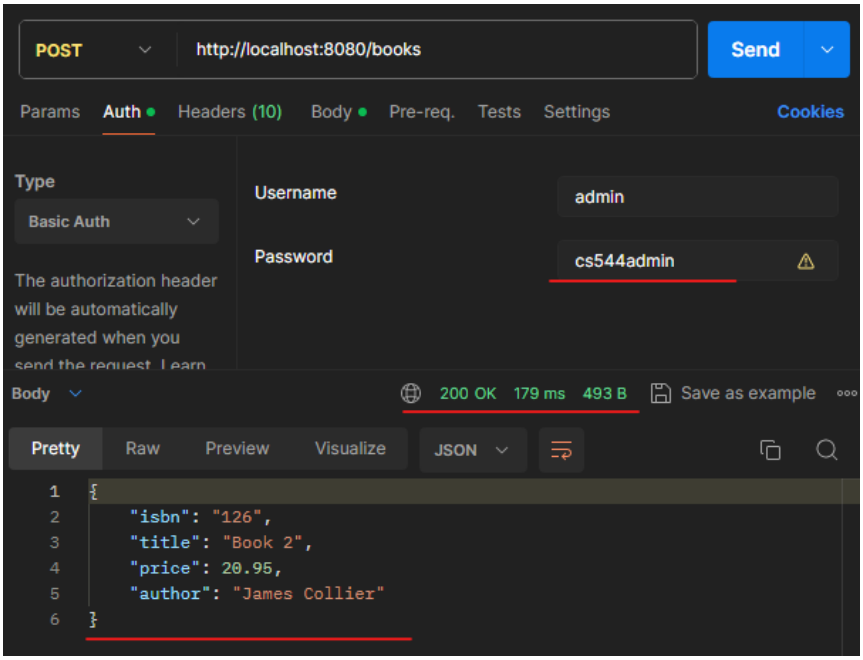
You can read the Spring Security documentation to achieve this: [Spring Security Authentication Doc]

In the second part of Lab A, you need now to implement some security in your endpoint mappings creating a **SecurityFilterChain** @Bean. You need to add a role to your user called "USER" and create a new user "admin" with the role "ADMIN". Now you need to filter your endpoints based on these two roles:

- "USER" Role should have access to your GET endpoints "/books", "/books/{isbn}", "/searchbooks/{author}" (or whatever names you have for your GET endpoints).
- "ADMIN" Role should have access to your Book Controller application's POST, PUT, and DELETE endpoints.

If you do it properly you should have a Forbidden response if you try to call any of the POST, PUT, or DELETE endpoints with your user account:





Part B:

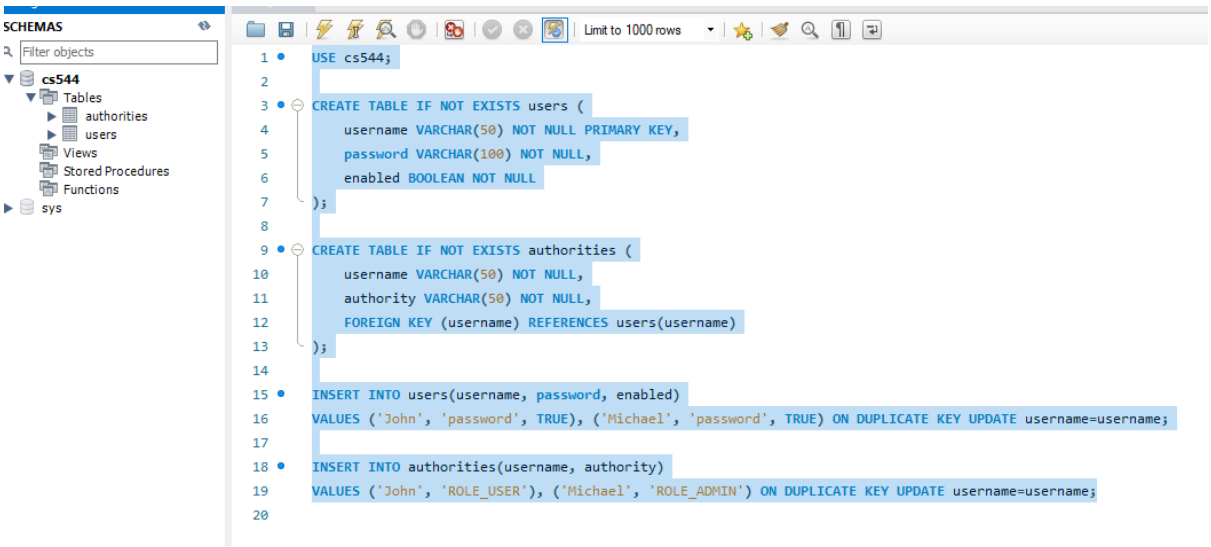
From Lab 9 Part B, Copy and Paste your Bank Application where you implemented your Account Controller and now do the same as you did in Part A, but this time we will use a simple JDBC approach instead of users in memory. To create your users and authority tables, you can run this query in your MySQL database:

USE cs544; // Here you can use any name you have for your DB.

```
CREATE TABLE IF NOT EXISTS users (
   username VARCHAR(50) NOT NULL PRIMARY KEY,
   password VARCHAR(100) NOT NULL,
   enabled BOOLEAN NOT NULL
);

CREATE TABLE IF NOT EXISTS authorities (
   username VARCHAR(50) NOT NULL,
   authority VARCHAR(50) NOT NULL,
   FOREIGN KEY (username) REFERENCES users(username)
);

INSERT INTO users(username, password, enabled)
VALUES ('John', 'user', TRUE), ('Michael', 'admin', TRUE) ON DUPLICATE KEY
UPDATE username=username;

INSERT INTO authorities(username, authority)
VALUES ('John', 'ROLE_USER'), ('Michael', 'ROLE_ADMIN') ON DUPLICATE KEY
UPDATE username=username;
```

Now you should have your tables created with Jonh and Michael users:

- Note: You can use any username and password that you want.



For this lab the WebSecurityConfig class is given:

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    private final DataSource dataSource;

    public WebSecurityConfig(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {
        auth.jdbcAuthentication()
                .dataSource(dataSource)
                .passwordEncoder(noOpPasswordEncoder());
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(requests -> requests
                        .anyRequest().authenticated()
                )
                .httpBasic(Customizer.withDefaults());
        return http.build();
    }

    @SuppressWarnings("deprecation")
    @Bean
    public static PasswordEncoder noOpPasswordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

}
```

As you noticed, this time we have not secured the endpoints of the Account Controller class in our SecurityFilterChain Bean. We will now secure our endpoints using the MethodSecurity approach at Class level in the AccountController class directly. To do so, please consult the Spring Security Servlet Authorization Documentation. Link: Spring Security Servlet Authorization Doc

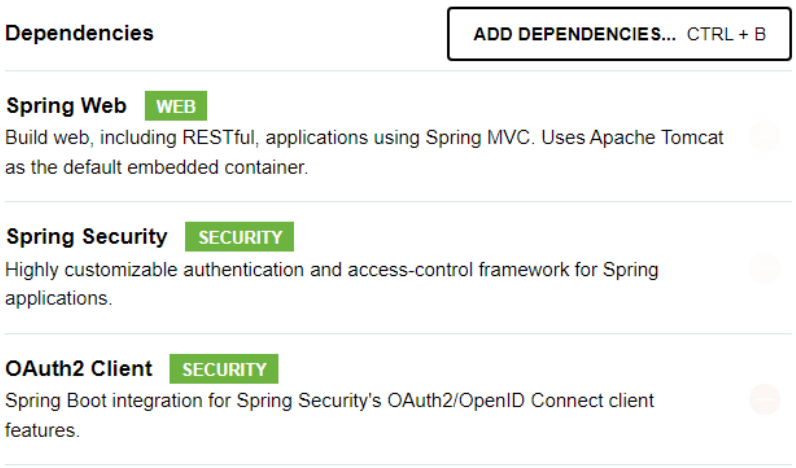
These is the criteria to secure your endpoints:

- POST Mappings createAccount() and accountOperation() has role “USER”
- GET Mappings getAccount() and getAllAccounts() has role “ADMIN”

Part C: (Optional)

For this part, we are going to use Oauth2 SSO Authentication using Google. First thing you need to do is go to the Spring Initializr Io and create your project. All you need are three dependencies:

- Spring Web
- Spring Security
- Spring OAuth2 Client



In your Spring Boot application, you need to create a RestController class with two endpoints. One is accessible without authentication and the other one with authentication. No need for Authorization for this lab.

You will need to create a Project in Google with Client ID, Client Secret, and Redirect URI. To do so you need to visit Google Developers Site.

In your application.yml file, all you need is this configuration:

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: <clientId>
            client-secret: <clientSecret>
```

You can make use of the WebSecurityConfig class from Lab 10 Part A to secure your endpoint. You only need the SecurityFilterChain Bean for this lab. For more information please visit: Spring Security OAuth2 Doc.

Once you implement your RestController and WebSecurityConfig classes and run your application, you should be able to see a response similar to this if you use the Principal/OidcUser object to get your information:

Welcome Samuel Bartolome Valiente to Spring Boot Google SSO

If you find issues creating your application using Google SSO you can visit this tutorial: Spring Boot Google OAuth2 OpenID Tutorial. It should give you an idea about what you need.

Please make your project as simple as possible. All you need is what I have explained to you above.

What to hand in:

1. A separate zip file with the solution of part A
2. A separate zip file with the solution of part B
3. A separate zip file with the solution of part C (Optional Lab)