Introduction to demoMFA Project

The demoMFA project is a Spring Boot application that integrates Okta for OAuth 2.0 authentication and Multi-Factor Authentication (MFA). This guide explains how to set up and run the project.

Step 1: Setting Up the Project

1. **Install Spring Tool Suite (STS) or Use Eclipse with Spring Extensions**:
    - If you're using Eclipse, install the Spring Tools plugin from the Eclipse Marketplace.
    - Alternatively, download and install Spring Tool Suite (STS) from the Spring website.
2. **Create a New Spring Starter Project**:
    - Open STS or Eclipse with Spring Tools installed.
    - Go to **File** > **New** > **Spring Starter Project**.
    - Choose **Project**:
        - Select "Spring" under the "Project" section.
        - Choose "Spring Starter Project" and click **Next**.
    - Choose **Project Metadata**:
        - Enter your project details (e.g., Group, Artifact, Name, Description).
        - Choose "Web", "Security", and "Okta" under the dependencies if available; otherwise, add them manually later.
        - Click **Finish**.

Step 2: Add Dependencies

1. **Update pom.xml or build.gradle**:
    - Open your pom.xml file if you're using Maven or build.gradle if you're using Gradle.

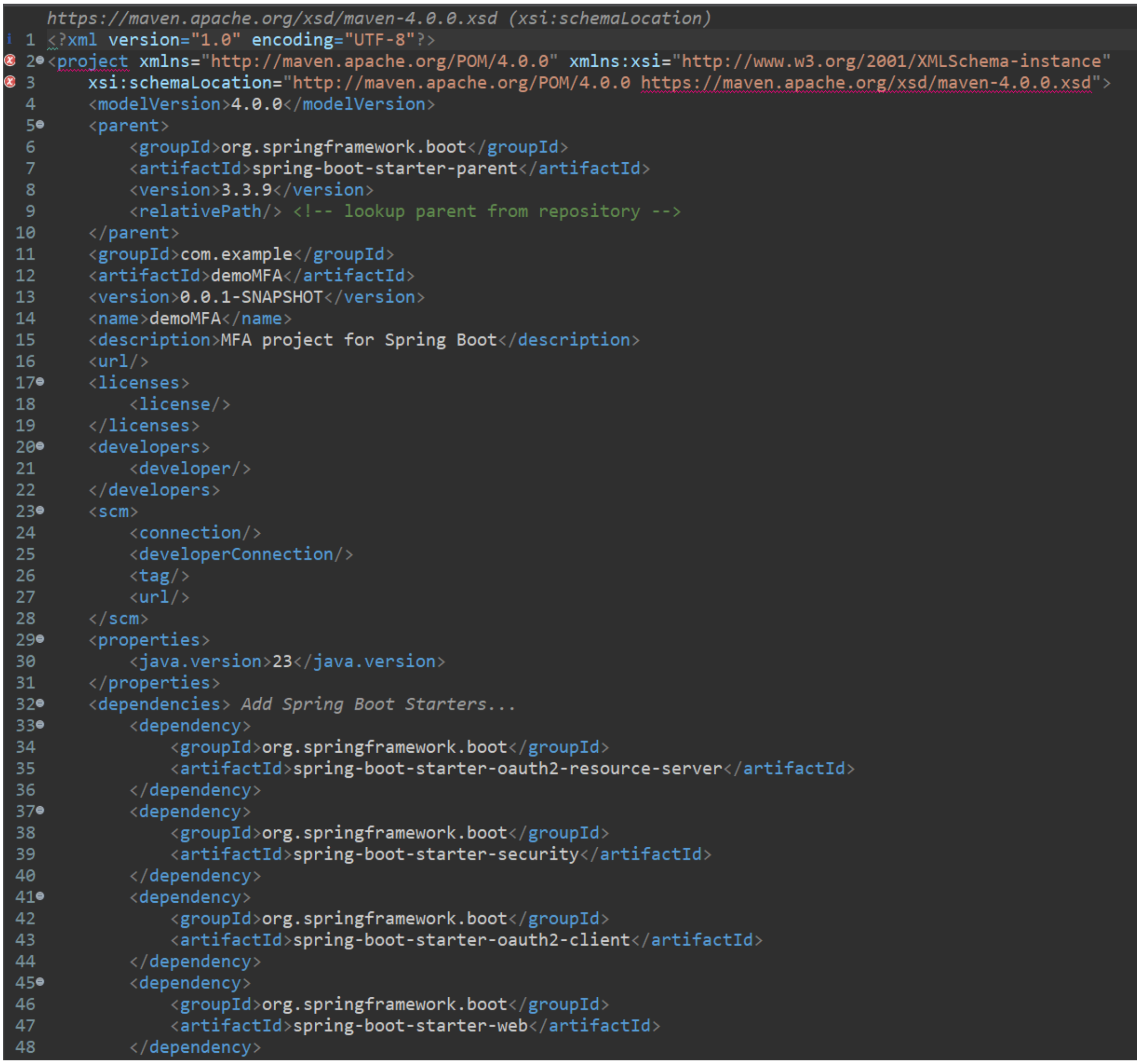
- Add the following dependencies for Okta and OAuth:

Maven (pom.xml):

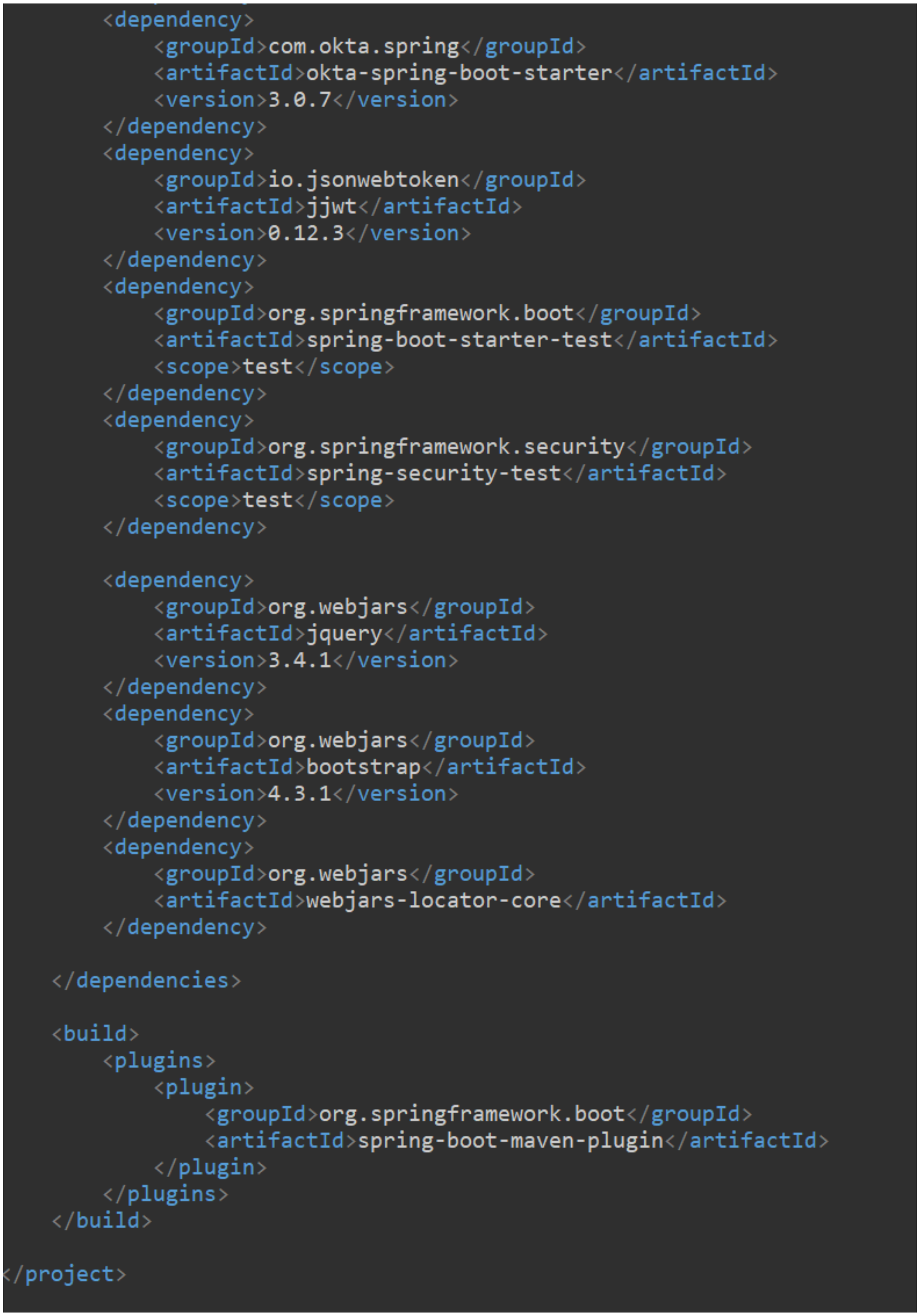
xml

```
<dependencies>
    <!-- Spring Security OAuth2 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-client</artifactId>
    </dependency>
    <!-- Okta -->
    <dependency>
        <groupId>com.okta.spring</groupId>
        <artifactId>okta-spring-boot-starter</artifactId>
        <version>3.0.7</version> <!-- Check for the latest version -->
    </dependency>
    <!-- JWT -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.12.3</version> <!-- Check for the latest version -->
    </dependency>
</dependencies>
```

Screenshots of pom.yml file:



```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<?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>3.3.9</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demoMFA</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demoMFA</name>
    <description>MFA project for Spring Boot</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>23</java.version>
    </properties>
    <dependencies> Add Spring Boot Starters...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-oauth2-resource-server</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-oauth2-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```



```xml
        <dependency>
            <groupId>com.okta.spring</groupId>
            <artifactId>okta-spring-boot-starter</artifactId>
            <version>3.0.7</version>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.12.3</version>
        </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>

        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>jquery</artifactId>
            <version>3.4.1</version>
        </dependency>
        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>bootstrap</artifactId>
            <version>4.3.1</version>
        </dependency>
        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>webjars-locator-core</artifactId>
        </dependency>

    </dependencies>

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

</project>
```

Gradle (build.gradle):

groovy

```
implementation 'org.springframework.boot:spring-boot-starter-security'

implementation 'org.springframework.boot:spring-boot-starter-oauth2-client'

implementation 'com.okta.spring:okta-spring-boot-starter:3.0.7' // Check for the latest version

implementation 'io.jsonwebtoken:jjwt:0.12.3' // Check for the latest version
```

2. **Refresh Dependencies**:
    - Right-click your project in the Project Explorer and select **Maven** > **Update Project** (for Maven) or **Gradle** > **Refresh Gradle Project** (for Gradle).

Step 3: Configure Okta

1. **Create an Okta Developer Account**:
    - Go to Okta Developer and sign up for a free account.
2. **Create an Okta Application**:
    - Log in to your Okta dashboard.
    - Go to **Applications** > **Add Application**.
    - Choose **Web** and click **Next**.
    - Set up your application with the following settings:
        - **Login redirect URIs**: https://localhost:8080/login/oauth2/code/okta
        - **Logout redirect URIs**: https://localhost:8080
    - Click **Done**.
3. **Configure Okta in Your Application**:
    - In your application.properties or application.yml file, add the following configuration:

application.properties:

text

okta.oauth2.issuer=https://your-okta-domain.com/oauth2/default

okta.oauth2.client-id=your-client-id

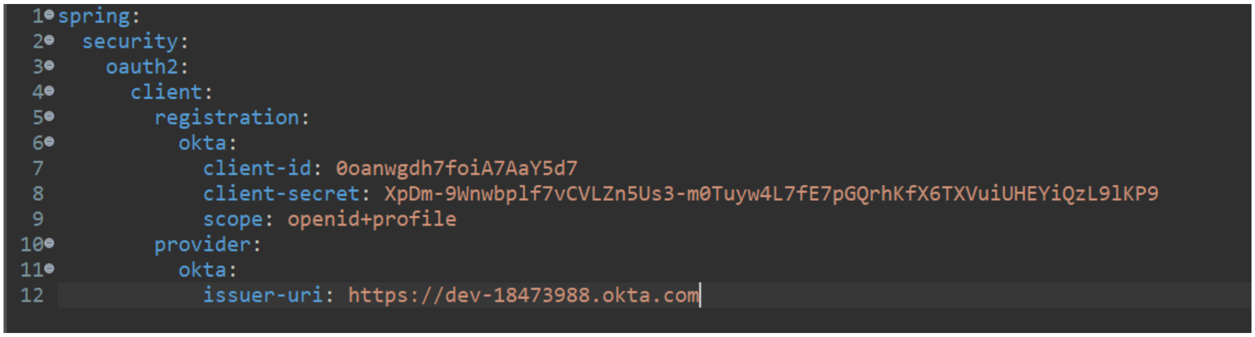
okta.oauth2.client-secret=your-client-secret

application.yml:

text

```
okta:
  oauth2:
    issuer: https://your-okta-domain.com/oauth2/default
    client-id: your-client-id
    client-secret: your-client-secret
```

Screenshots of application.yml file:



```
spring:
  security:
    oauth2:
      client:
        registration:
          okta:
            client-id: 0oanwgdh7foiA7AaY5d7
            client-secret: XpDm-9Wnwbplf7vCVLZn5Us3-m0Tuyw4L7fE7pGQrhKfX6TXVuiUHEYiQzL9lKP9
            scope: openid+profile
        provider:
          okta:
            issuer-uri: https://dev-18473988.okta.com
```

Step 4: Implement REST Endpoints

1. **Create a REST Controller**:
   - Create a new Java class named HelloController.java in your project's package (e.g., com.example.demoMFA).

java

package com.example.demoMFA;

import org.springframework.web.bind.annotation.GetMapping;

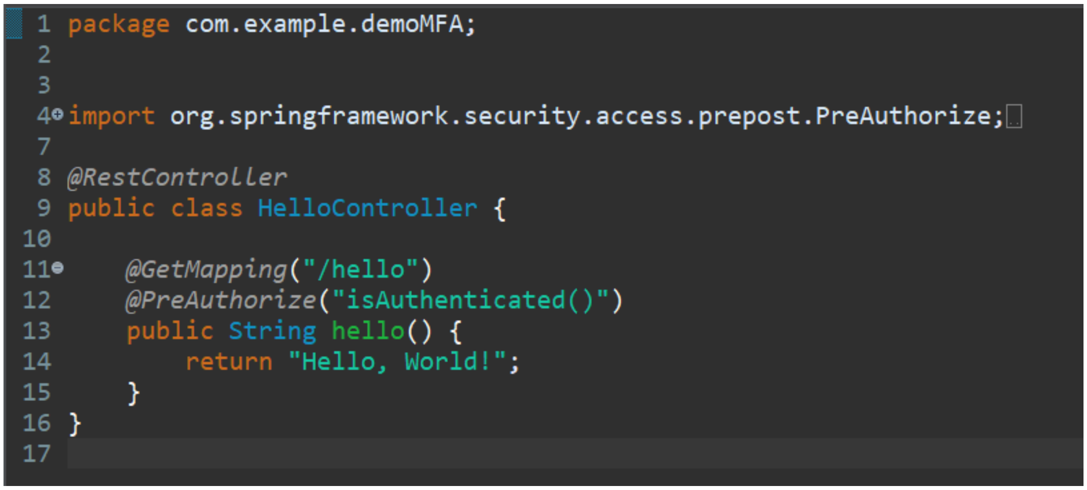
```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }
}
```

2. **Secure the REST Endpoint**:
    - Modify the HelloController to require authentication:

java

```
package com.example.demoMFA;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello")
    @PreAuthorize("isAuthenticated()")
    public String hello() {
        return "Hello, World!";
    }
}
```

Screenshots of HelloController.java file:



```
package com.example.demoMFA;


import org.springframework.security.access.prepost.PreAuthorize;

@RestController
public class HelloController {

    @GetMapping("/hello")
    @PreAuthorize("isAuthenticated()")
    public String hello() {
        return "Hello, World!";
    }
}

```

Step 5: Implement JWT

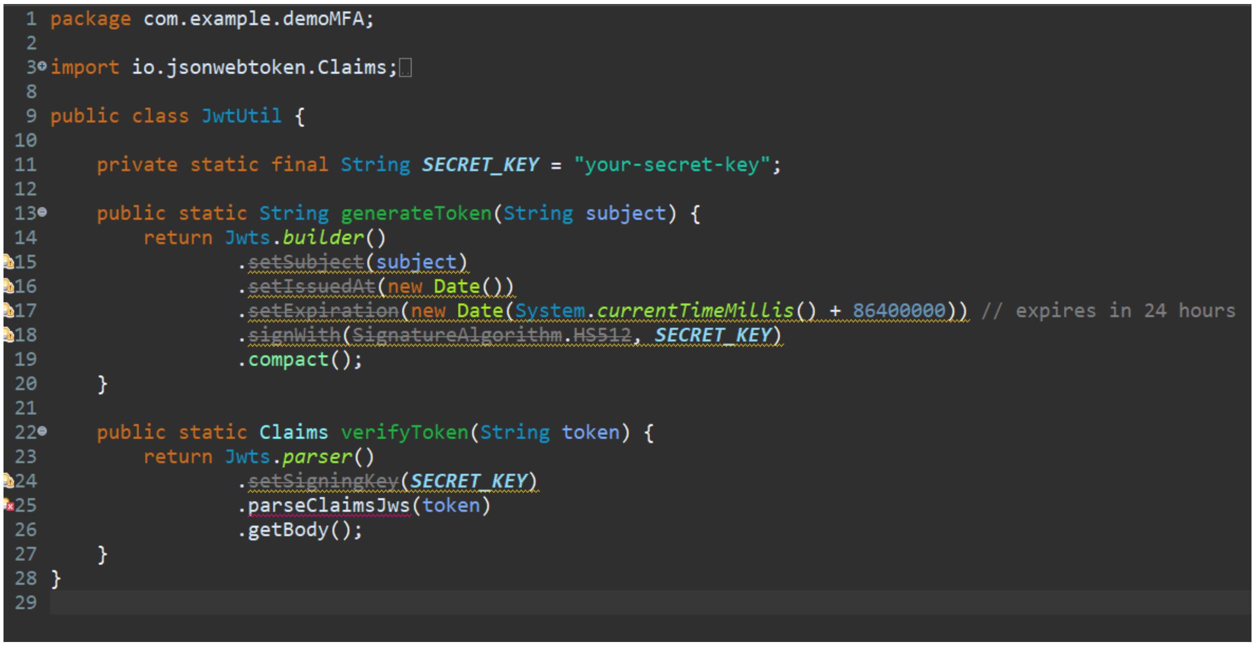
1. **Generate JWT Tokens**:
    - Create a new Java class named JwtUtil.java to handle JWT token generation and verification.

java

```
package com.example.demoMFA;


import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
```

```java
import java.util.Date;

public class JwtUtil {

  private static final String SECRET_KEY = "your-secret-key";

  public static String generateToken(String subject) {
    return Jwts.builder()
        .setSubject(subject)
        .setIssuedAt(new Date())
        .setExpiration(new Date(System.currentTimeMillis() + 86400000)) // expires in 24 hours
        .signWith(SignatureAlgorithm.HS512, SECRET_KEY)
        .compact();
  }

  public static Claims verifyToken(String token) {
    return Jwts.parser()
        .setSigningKey(SECRET_KEY)
        .parseClaimsJws(token)
        .getBody();
  }
}
```

Screenshots of JwtUtil.java file:



```java
package com.example.demoMFA;

import io.jsonwebtoken.Claims;

public class JwtUtil {

    private static final String SECRET_KEY = "your-secret-key";

    public static String generateToken(String subject) {
        return Jwts.builder()
                .setSubject(subject)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + 86400000)) // expires in 24 hours
                .signWith(SignatureAlgorithm.HS512, SECRET_KEY)
                .compact();
    }

    public static Claims verifyToken(String token) {
        return Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody();
    }
}

```

2. **Use JWT in Your Application**:
    - Modify your authentication logic to generate and verify JWT tokens.

Step 6: Implement MFA

1. **Configure MFA in Okta**:
    - Log in to your Okta dashboard.
    - Go to **Security** > **Auth Policies**.
    - Create or edit a policy to require MFA for your application.
2. **Integrate MFA in Your Application**:
    - Use Okta's APIs to check if a user has completed MFA during authentication.
    - This may involve customizing your authentication flow to handle MFA checks.

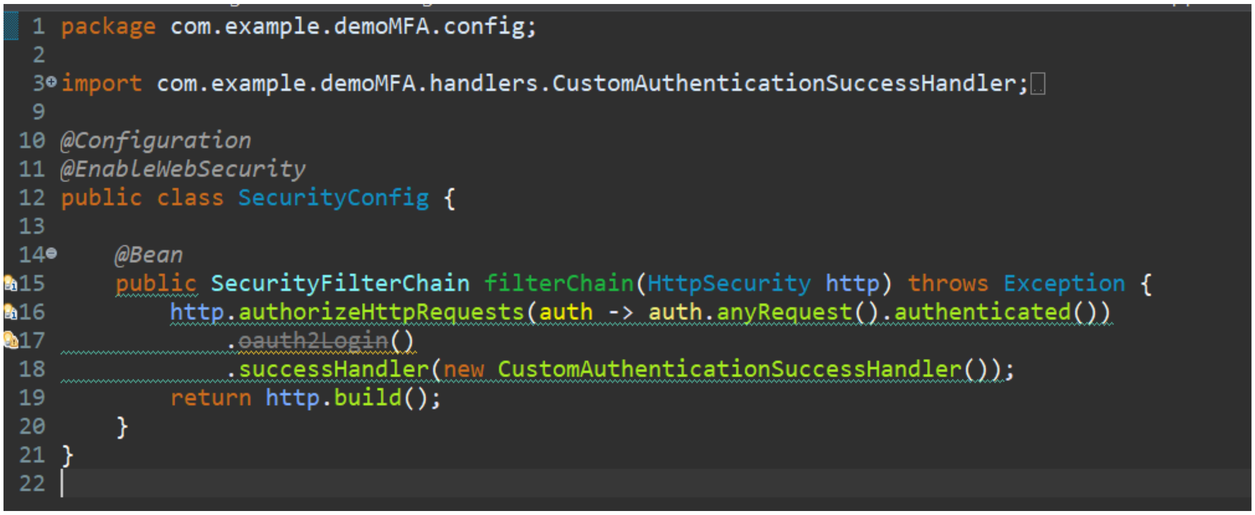
Step 7: Configure Security

1. **Create or Update SecurityConfig.java**:
    - Ensure that your SecurityConfig.java file is correctly configured to use OAuth 2 with Okta.

java

```
package com.example.demoMFA.config;


import com.example.demoMFA.handlers.CustomAuthenticationSuccessHandler;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;


@Configuration
@EnableWebSecurity
public class SecurityConfig {


    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(auth -> auth.anyRequest().authenticated())
            .oauth2Login()
            .successHandler(new CustomAuthenticationSuccessHandler());
        return http.build();
    }
}
```

Screenshots of SecurityConfig.java file:



```
package com.example.demoMFA.config;

import com.example.demoMFA.handlers.CustomAuthenticationSuccessHandler;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(auth -> auth.anyRequest().authenticated())
            .oauth2Login()
            .successHandler(new CustomAuthenticationSuccessHandler());
        return http.build();
    }
}
```

Step 8: Implement Custom Authentication Success Handler

1. **Create a Custom Success Handler**:

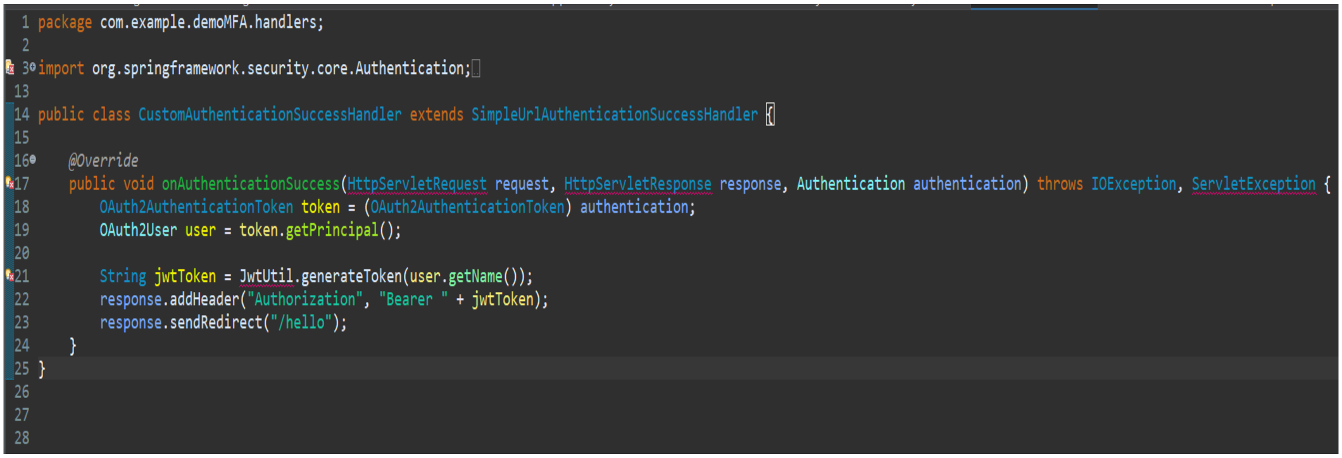
    - Create a new Java class named CustomAuthenticationSuccessHandler.java to handle the logic after successful authentication.

java

```
package com.example.demoMFA.handlers;


import org.springframework.security.core.Authentication;

import org.springframework.security.oauth2.client.authentication.OAuth2AuthenticationToken;

import org.springframework.security.oauth2.core.user.OAuth2User;
```

```
import
org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import
org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccess
Handler;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class CustomAuthenticationSuccessHandler extends
SimpleUrlAuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws
IOException, ServletException {
        OAuth2AuthenticationToken token = (OAuth2AuthenticationToken)
authentication;
        OAuth2User user = token.getPrincipal();

        // Generate JWT token
        String jwtToken = JwtUtil.generateToken(user.getName());
        response.addHeader("Authorization", "Bearer " + jwtToken);
        // Redirect to protected endpoint
        response.sendRedirect("/hello");
    }
}
```

Screenshots of CustomAuthenticationSuccessHandler.java file:



```
package com.example.demoMFA.handlers;

import org.springframework.security.core.Authentication;

public class CustomAuthenticationSuccessHandler extends SimpleUrlAuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {
        OAuth2AuthenticationToken token = (OAuth2AuthenticationToken) authentication;
        OAuth2User user = token.getPrincipal();

        String jwtToken = JwtUtil.generateToken(user.getName());
        response.addHeader("Authorization", "Bearer " + jwtToken);
        response.sendRedirect("/hello");
    }
}
```

Step 9: Create Logout Page

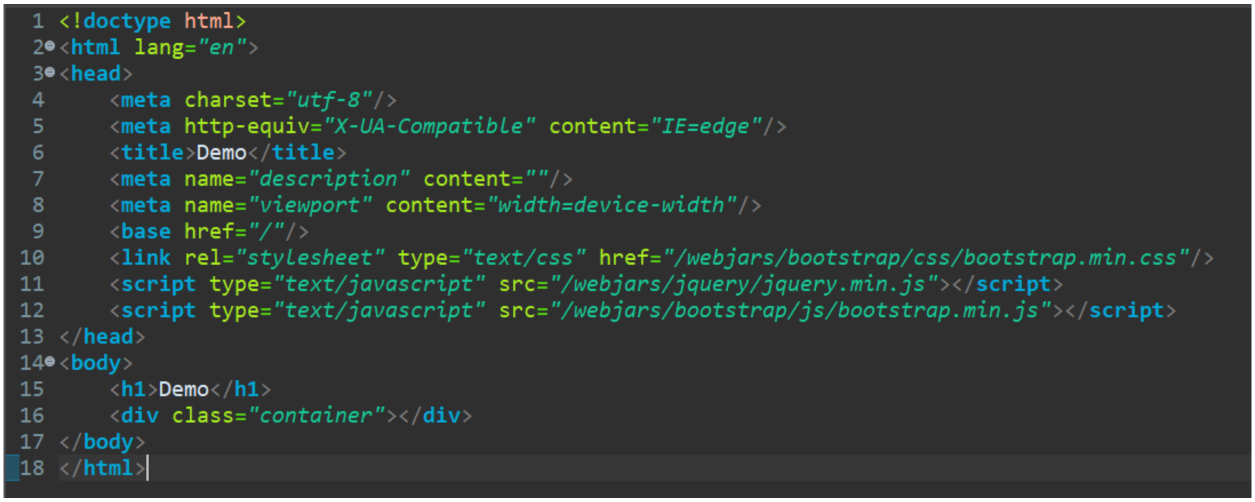
1. **Create an HTML File for Logout**:

    - Create a new HTML file named logout.html in your project's resources directory (e.g., src/main/resources/static).

xml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <title>Logout</title>
    <meta name="description" content=""/>
    <meta name="viewport" content="width=device-width"/>
    <base href="/"/>
    <link rel="stylesheet" type="text/css" href="/webjars/bootstrap/css/bootstrap.min.css"/>
    <script type="text/javascript" src="/webjars/jquery/jquery.min.js"></script>
    <script type="text/javascript" src="/webjars/bootstrap/js/bootstrap.min.js"></script>
</head>
<body>
    <h1>Logged Out Successfully</h1>
    <div class="container"></div>
</body>
</html>
```

Screenshots of index.html file inside static package:



```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <title>Demo</title>
    <meta name="description" content=""/>
    <meta name="viewport" content="width=device-width"/>
    <base href="/"/>
    <link rel="stylesheet" type="text/css" href="/webjars/bootstrap/css/bootstrap.min.css"/>
    <script type="text/javascript" src="/webjars/jquery/jquery.min.js"></script>
    <script type="text/javascript" src="/webjars/bootstrap/js/bootstrap.min.js"></script>
</head>
<body>
    <h1>Demo</h1>
    <div class="container"></div>
</body>
</html>
```

Step 10: Enabling MFA in Okta

Enabling Multi-Factor Authentication (MFA) in Okta involves setting up policies that enforce additional security checks during the login process. Here's a step-by-step guide on how to do it:

Step 1: Log in to Okta Dashboard

1. **Access Okta Dashboard**:
    - Navigate to your Okta organization's URL and log in as an administrator.

Step 2: Create or Edit an Auth Policy

1. **Go to Security > Auth Policies**:
    - Click on **Security** from the top navigation menu.
    - Select **Auth Policies**.
2. **Create a New Policy**:
    - Click **Add Policy**.
    - Choose **Sign On** as the policy type.
    - Select **Okta Password** as the rule type (or another type if you prefer).
    - Click **Next**.
3. **Edit an Existing Policy**:
    - If you already have a policy, find it in the list and click on its name to edit it.

Step 3: Add an MFA Rule

1. **Add Rule**:
    - In the policy settings, look for the **Rules** section.
    - Click **Add Rule**.
    - Choose **Require MFA** as the rule type.

    - Select the MFA factor you want to enforce (e.g., SMS, Authenticator App, etc.).

2. **Configure MFA Settings**:
    - Set up the MFA rule to apply under specific conditions, such as:
        - **User is accessing from an unknown location**.
        - **User is accessing a sensitive application**.
    - You can also set exceptions for certain users or groups.
3. **Save the Rule**:
    - Once you've configured the MFA rule, click **Save**.

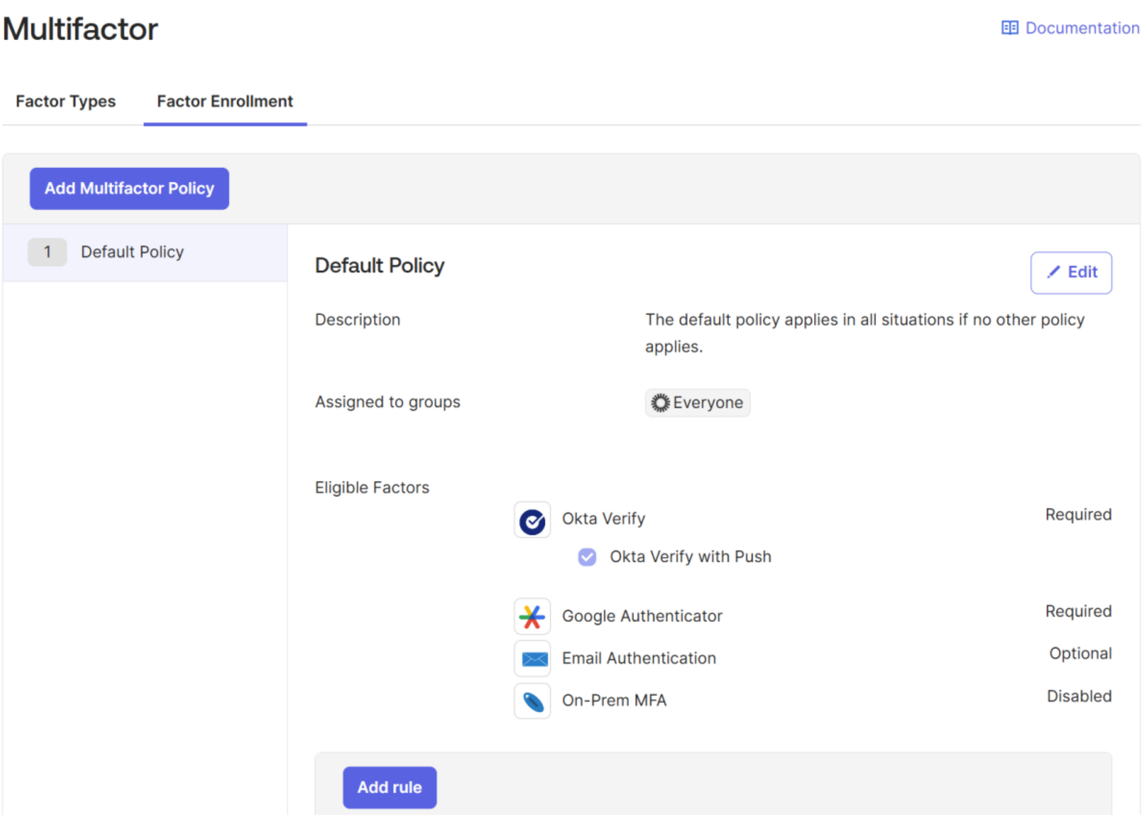
Step 4: Assign the Policy

1. **Assign the Policy**:
    - Ensure that the policy is assigned to the correct users or groups.
    - You can do this by specifying the target users or groups in the policy settings.
2. **Save and Activate the Policy**:
    - After configuring and assigning the policy, click **Save**.
    - If necessary, activate the policy by clicking **Activate**.

Step 5: Test MFA

1. **Test the MFA Flow**:
    - Log out of your Okta account and log back in to test the MFA flow.
    - Ensure that you are prompted for the additional MFA factor you configured.

By following these steps, you can effectively enable MFA in Okta to enhance the security of your applications and user accounts.



Default Policy

Description: The default policy applies in all situations if no other policy applies.

Assigned to groups: Everyone

Eligible Factors

Okta Verify	Required
✓ Okta Verify with Push	
Google Authenticator	Required
Email Authentication	Optional
On-Prem MFA	Disabled

In My Okta MFA settings, I have used Okta Verify and Google Authenticator as required and Email Authentication as optional.

Step 11: Creating Test Users in Okta:

To create test users in Okta, follow these steps:

Step 1: Log in to Okta Dashboard

1. **Access Okta Dashboard**:
   - Navigate to your Okta organization's URL and log in as an administrator.

Step 2: Go to Directory > People

1. **Navigate to People Section**:

- Click on **Directory** from the top navigation menu.
- Select **People**.

Step 3: Create a New User

1. **Add a New User**:
    - Click the **Add Person** button.
    - Fill in the user's details:
        - **First Name** and **Last Name**.
        - **Primary Email** (this will be the username).
        - **Login** (if different from the email).
    - Click **Save**.

Step 4: Assign Groups or Roles

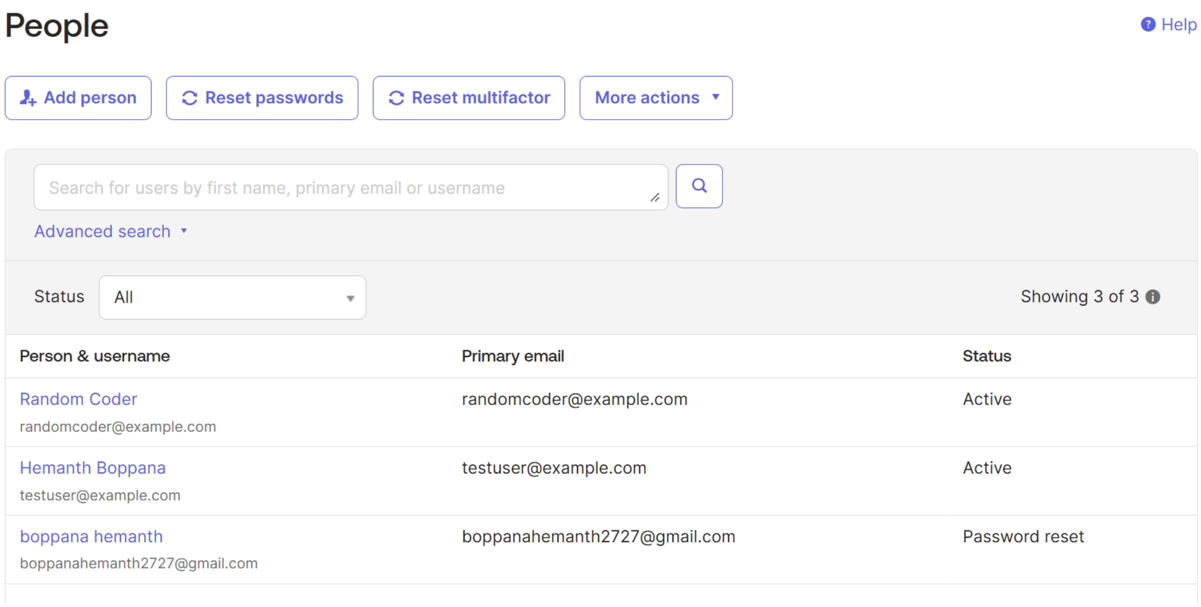
1. **Assign Groups or Roles**:
    - If necessary, assign the user to specific groups or roles by clicking on the **Groups** or **Roles** tab.

Step 5: Activate the User

1. **Activate the User**:
    - Ensure that the user is activated by checking the status.
    - If not activated, click **Activate** to enable the user account.

Step 6: Set Password

1. **Set Password**:
    - You can either set a password for the user or have them set it themselves through a password reset link.



Status: All

Showing 3 of 3

Person & username	Primary email	Status
Random Coder randomcoder@example.com	randomcoder@example.com	Active
Hemanth Boppana testuser@example.com	testuser@example.com	Active
boppana hemanth boppanahemanth2727@gmail.com	boppanahemanth2727@gmail.com	Password reset

I have added 2 additional test users along with the default user.

Step 12: Run Your Application

1. **Run Your Spring Boot Application**:
   - Right-click your project in the Project Explorer and select **Run As** > **Spring Boot App**.
   - Alternatively, use the Spring Boot Dashboard to start your application.
2. **Test Your Application**:
   - Open a web browser and navigate to https://localhost:8080/hello.
   - You should be redirected to Okta for authentication.
   - After successful authentication and MFA completion, you should see the "Hello, World!" message.

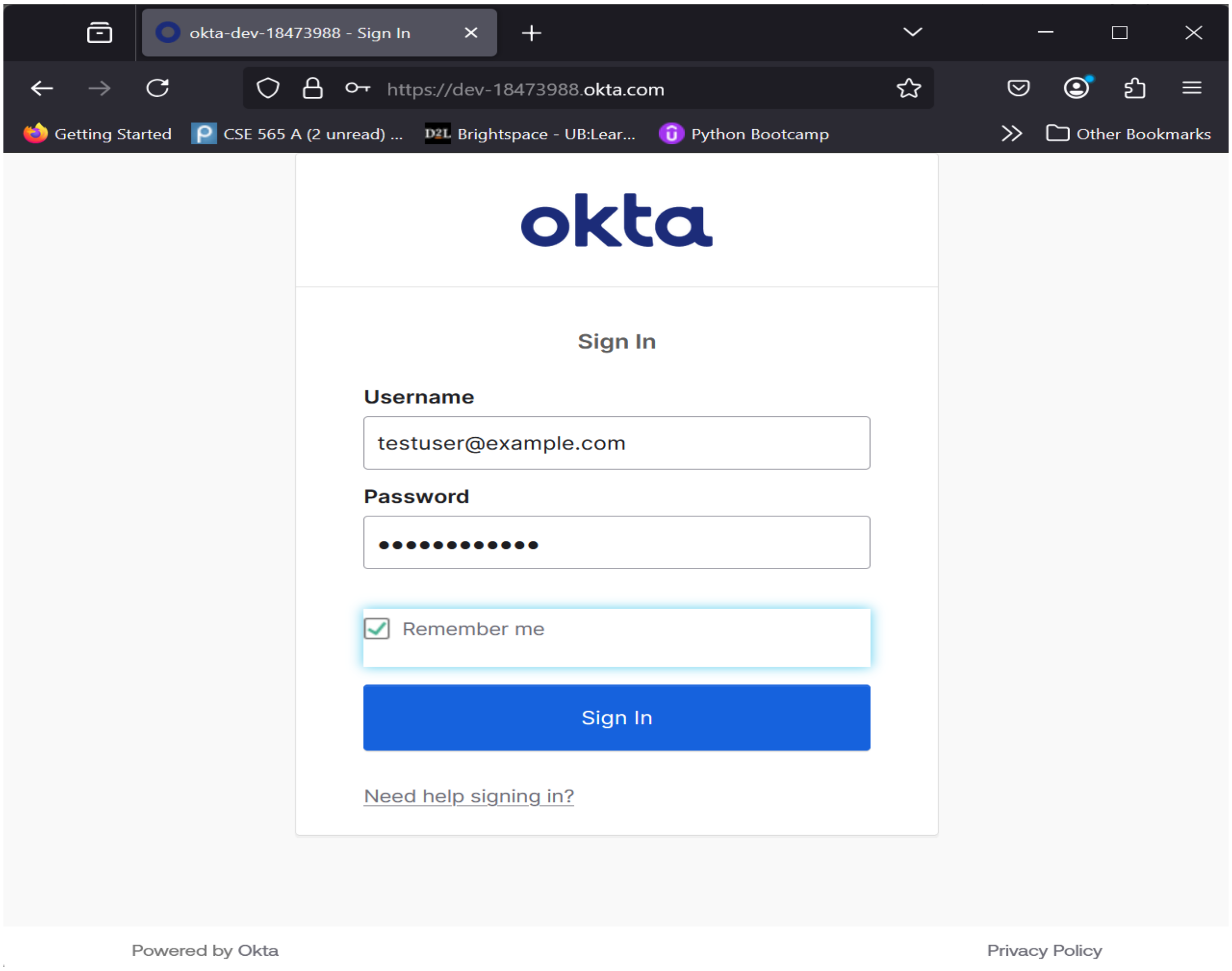
Troubleshooting

- **Okta Issuer URL Mismatch**:
  Ensure that the Okta issuer URL in your application properties matches the one in your Okta configuration.
- **SecurityFilterChain Issue**:
  Verify that you have the correct version of Spring Security in your project.

Conclusion

By following these steps, you can successfully set up a Spring Boot application with Okta and MFA integration. This project demonstrates how to secure a REST endpoint using OAuth 2.0 and JWT tokens, while also enforcing MFA for enhanced security.

Screenshots of Hand on process, step by step:

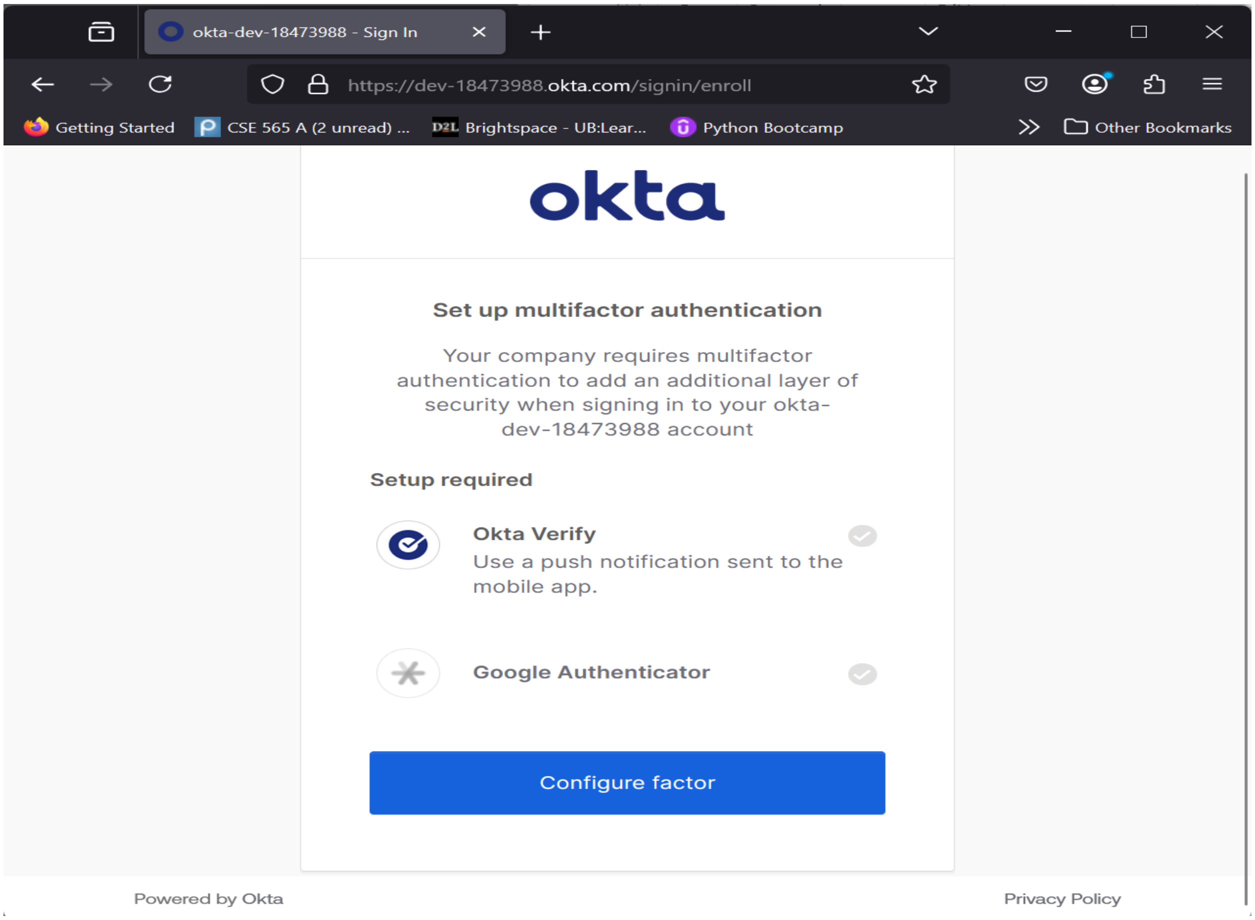


Sign In

Username: testuser@example.com

Password: ••••••••••••

☑ Remember me



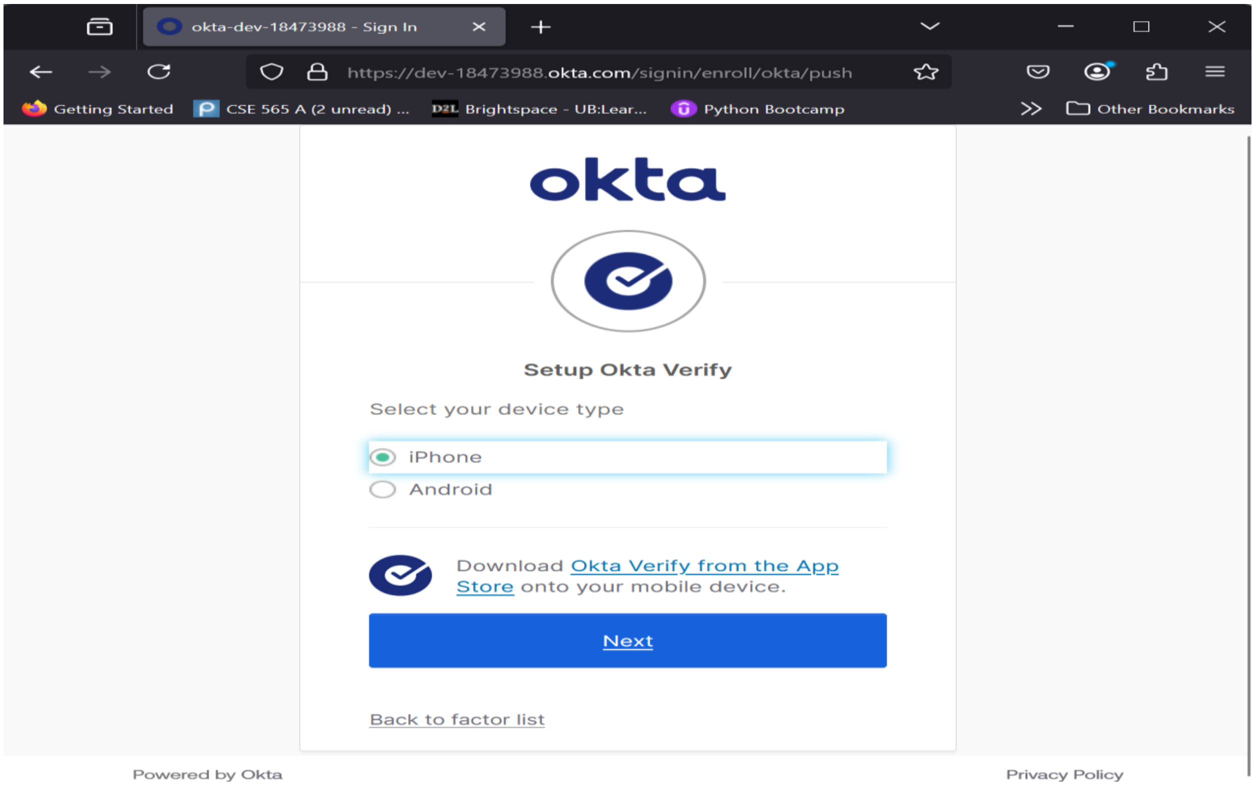
Set up multifactor authentication

Your company requires multifactor authentication to add an additional layer of security when signing in to your okta-dev-18473988 account

Setup required

Okta Verify ✓
Use a push notification sent to the mobile app.

Google Authenticator ✓

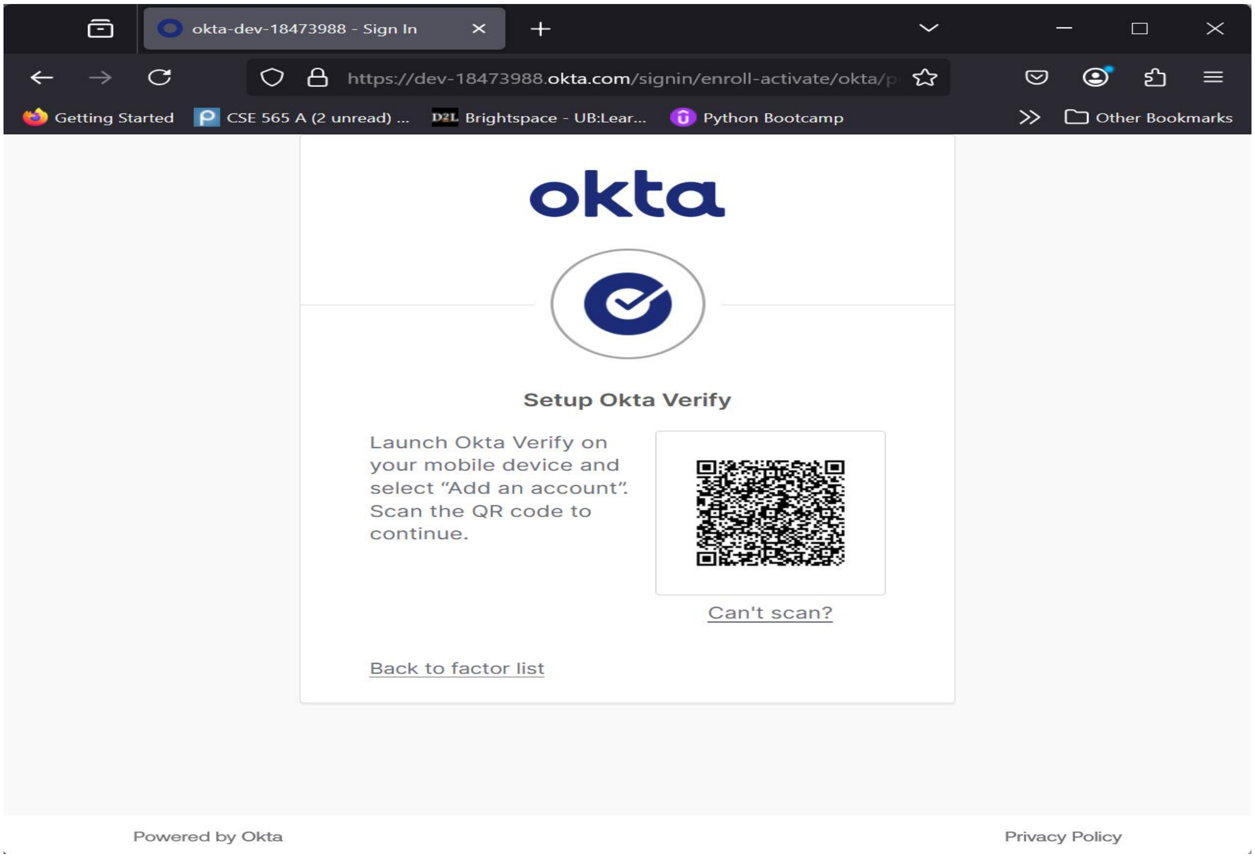


Setup Okta Verify

Select your device type

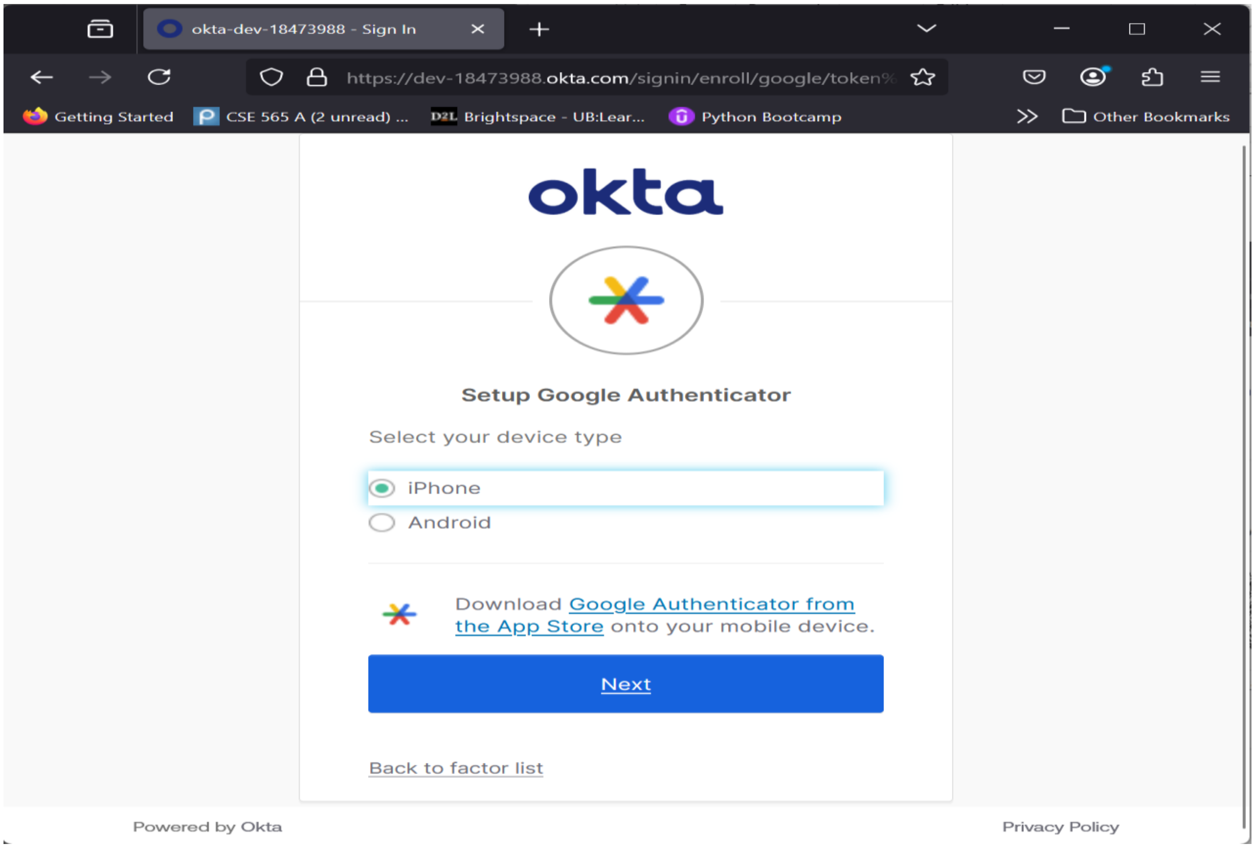
● iPhone
○ Android

Download Okta Verify from the App Store onto your mobile device.



Setup Okta Verify

Launch Okta Verify on your mobile device and select "Add an account". Scan the QR code to continue.



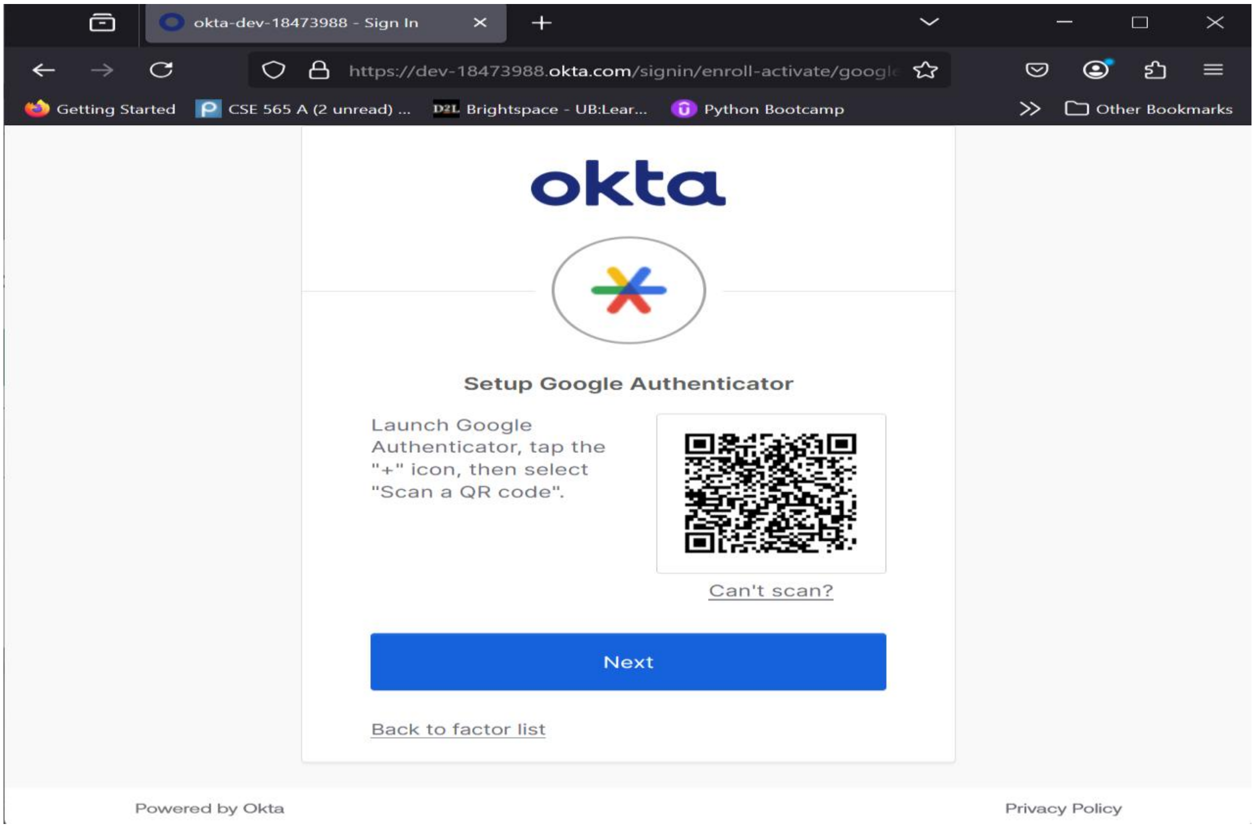
Setup Google Authenticator

Select your device type

● iPhone

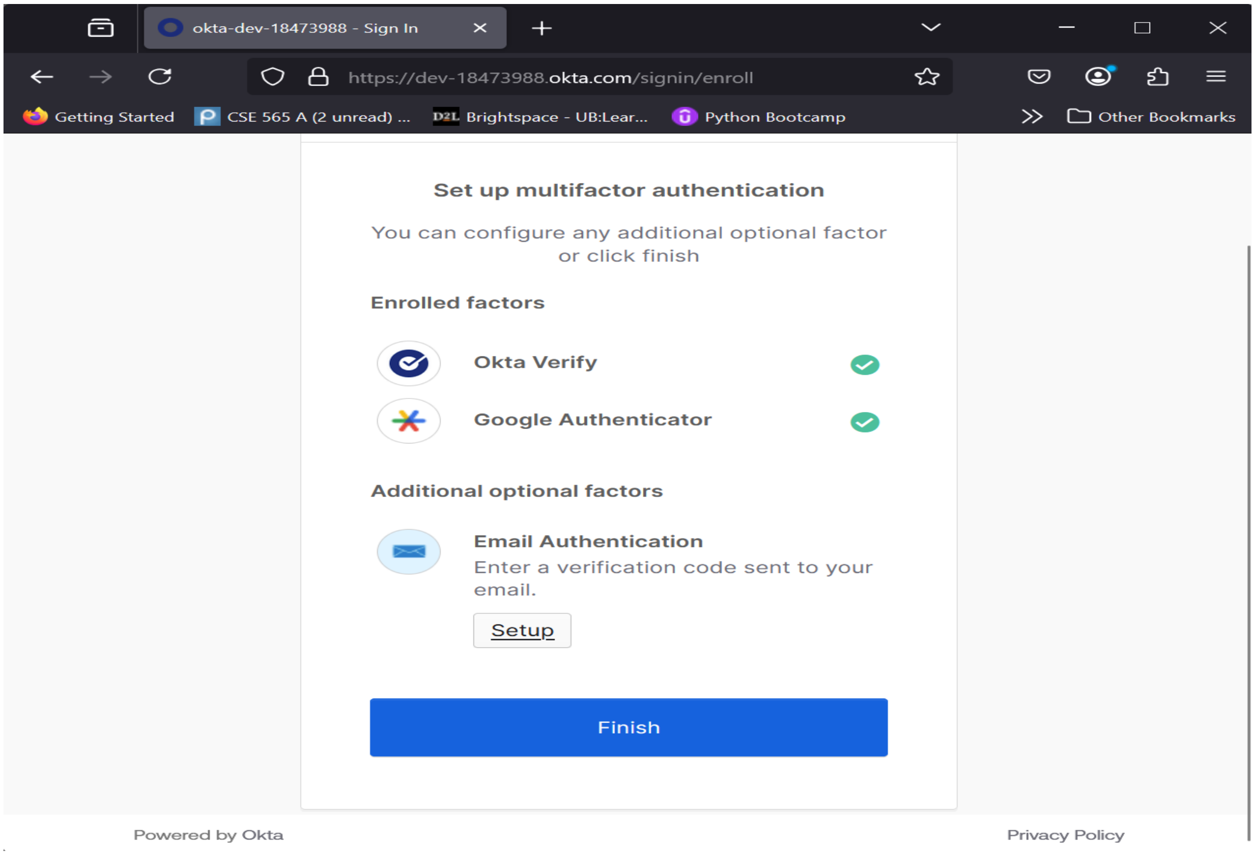
○ Android

Download Google Authenticator from the App Store onto your mobile device.



Setup Google Authenticator

Launch Google Authenticator, tap the "+" icon, then select "Scan a QR code".



Set up multifactor authentication

You can configure any additional optional factor or click finish

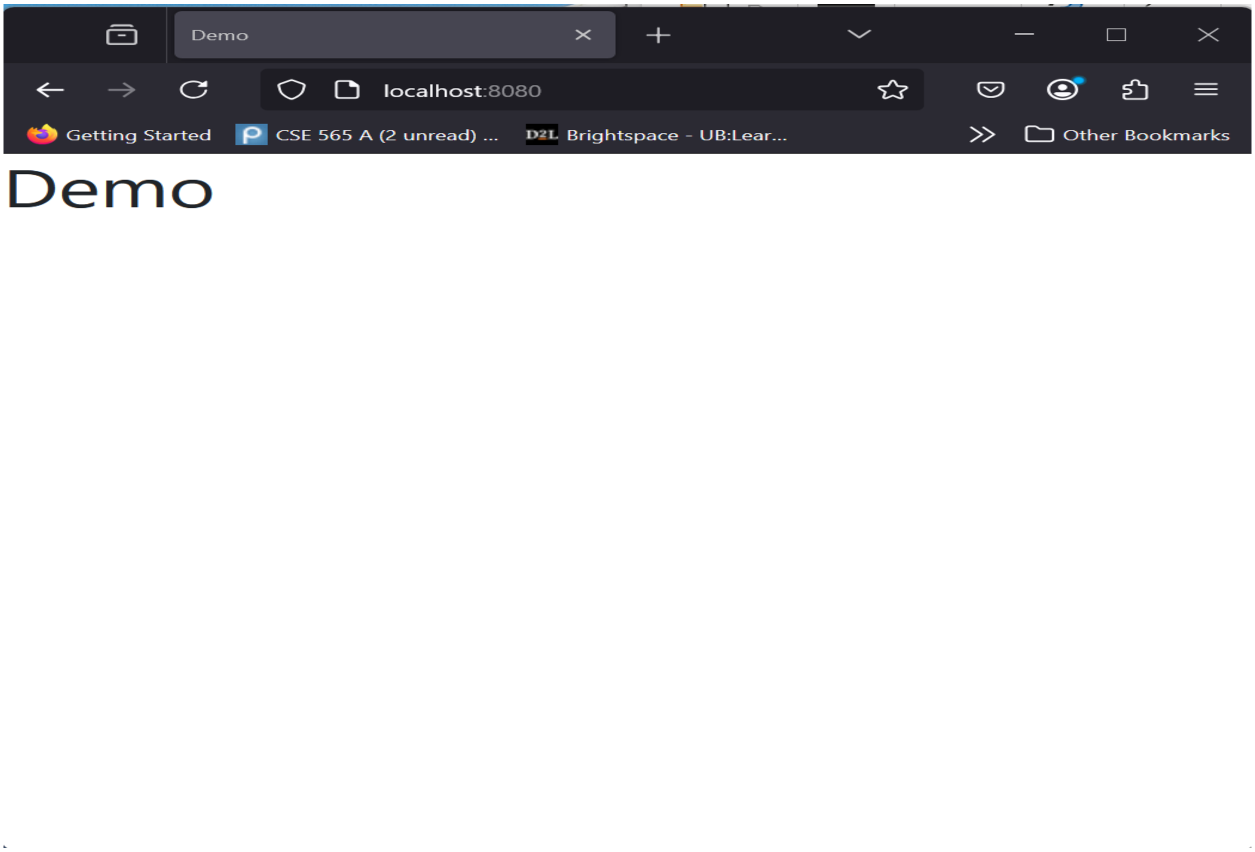
Enrolled factors

Okta Verify ✓

Google Authenticator ✓

Additional optional factors

Email Authentication
Enter a verification code sent to your email.



Demo

This is the index.html file that is being displayed after I have successfully logged in using the multifactor authentication process and then okta is redirecting me to this index.html page as I have configured this URL as my Sign-out redirect URI.