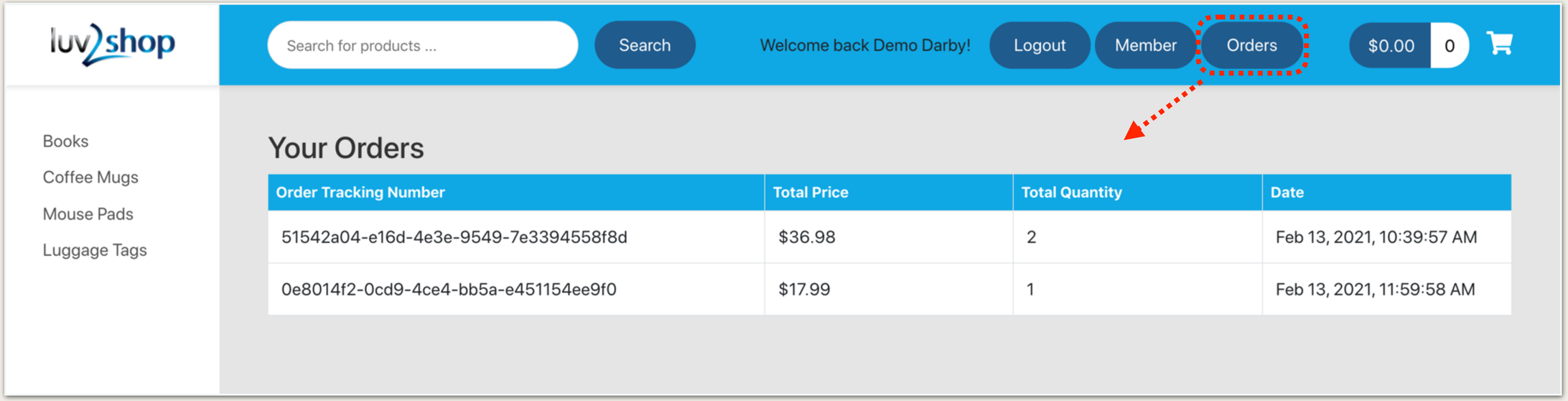
Secure Order History Backend

View Order History

- The Order History functionality is working
- However, the REST API for `/api/orders` is unsecured



Secure Order History Backend

- Let's secure the backend
- `/api/orders` should only available to logged in users

HTTP Method	Endpoint
`GET`	`/api/orders/search/findByCustomerEmail?email=demo@luv2code.com`

Development Process - Spring Boot

1. Add Okta Spring Boot Starter to Maven pom.xml
2. Create an App at the Okta Developer website
3. In Spring Boot app, set application properties
4. Protect endpoints in Spring Security configuration class

Step 1: Add Okta Spring Boot Starter to Maven pom.xml

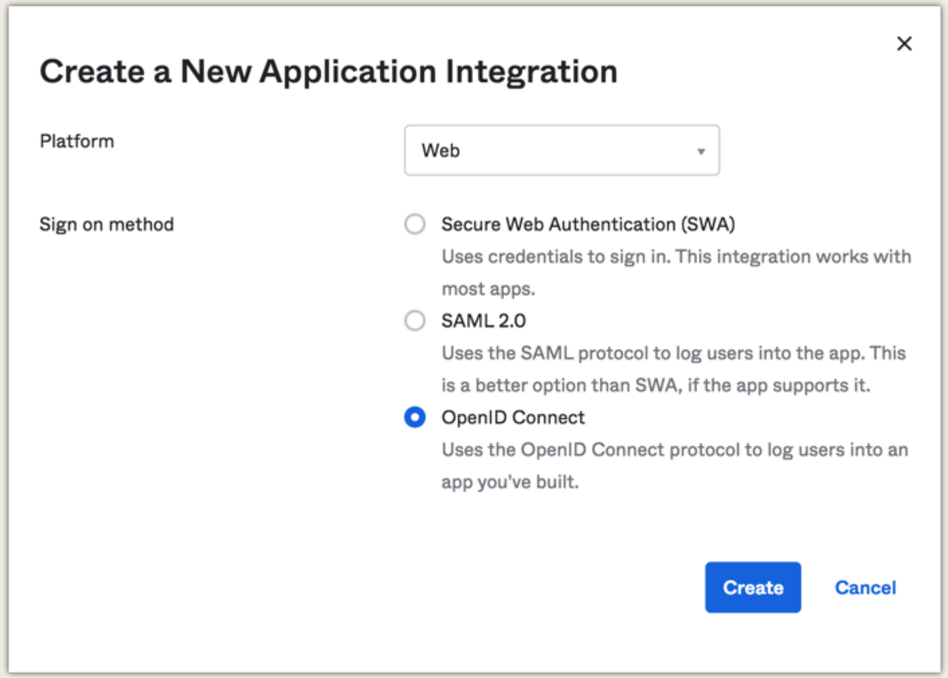
- Okta provides a Spring Boot Starter for OAuth 2 / OpenID Connect
- Simplifies integration and configuration of Spring Security and Okta

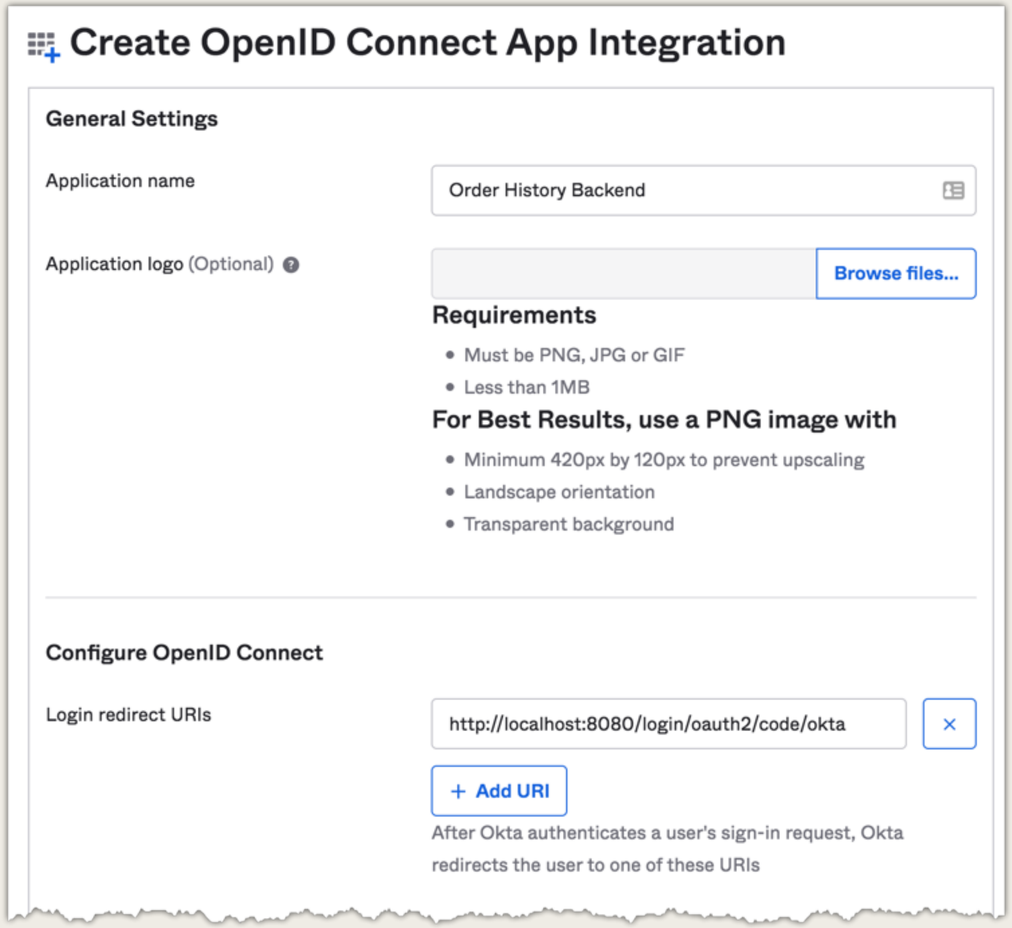
File: pom.xml

```
...
 <dependency>
    <groupId>com.okta.spring</groupId>
    <artifactId>okta-spring-boot-starter</artifactId>
    <version>2.0.1</version>
 </dependency>
...
```

https://github.com/okta/okta-spring-boot

Step 2: Create an App in Okta Developer website

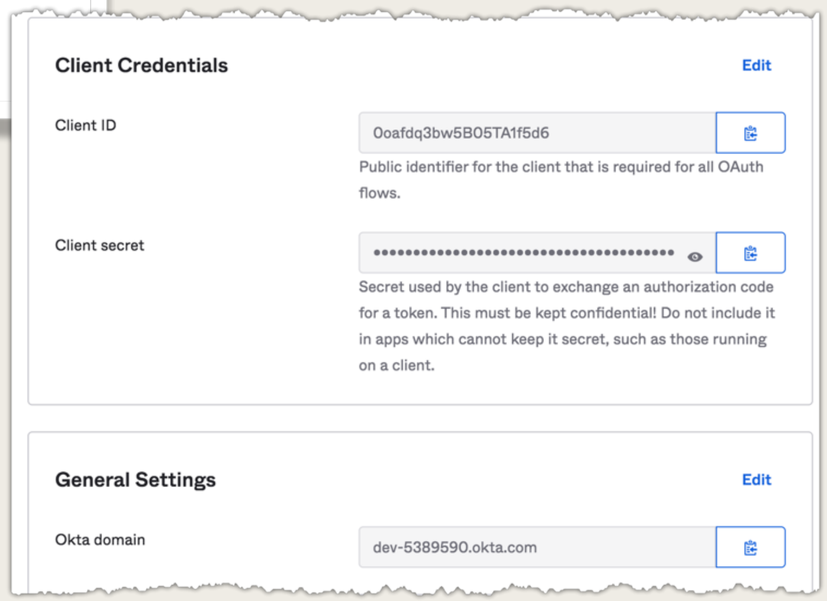




Step 3: In Spring Boot app, set application properties

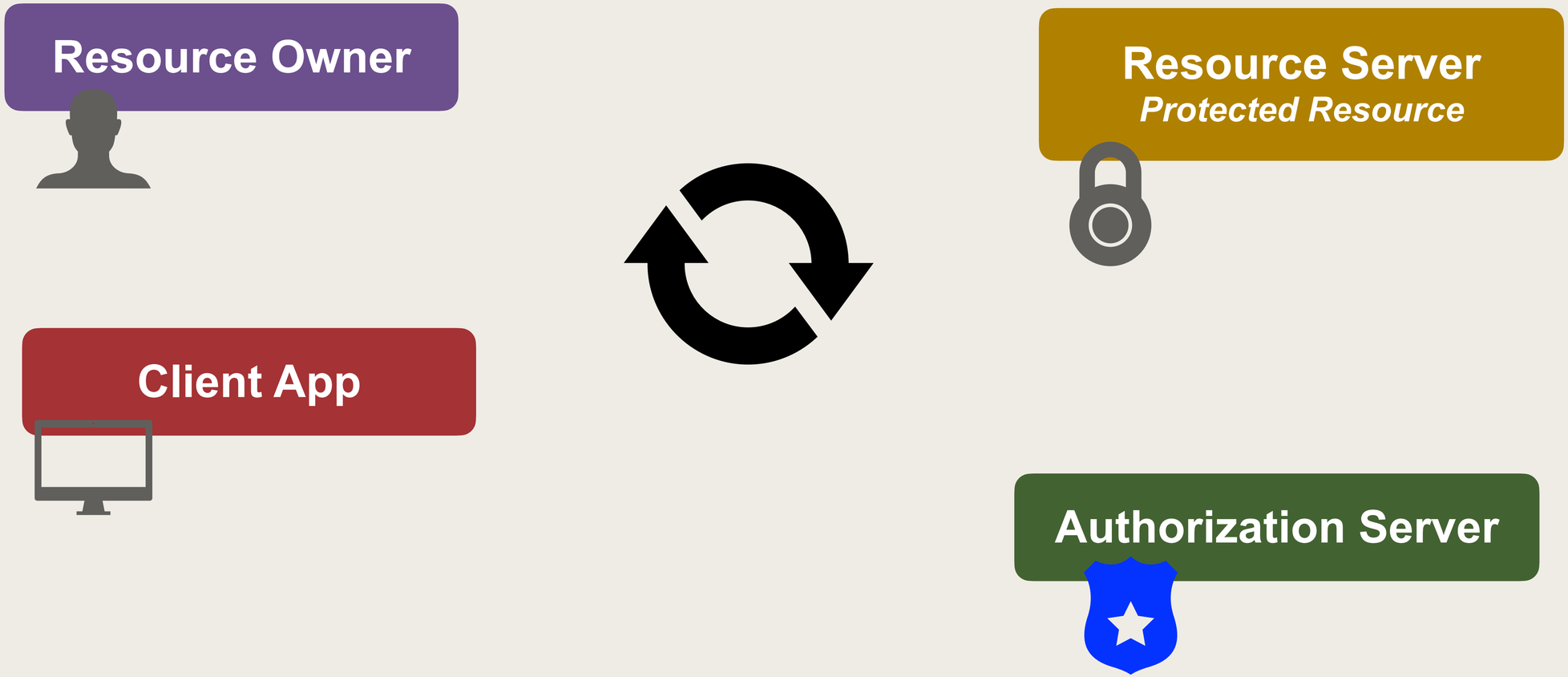
File: application.properties

```
...
okta.oauth2.client-id={yourClientId}
okta.oauth2.client-secret={yourClientSecret}
okta.oauth2.issuer=https://{yourOktaDomain}/oauth2/default
```



Spring Boot application will use these properties to verify / validate JWT access tokens

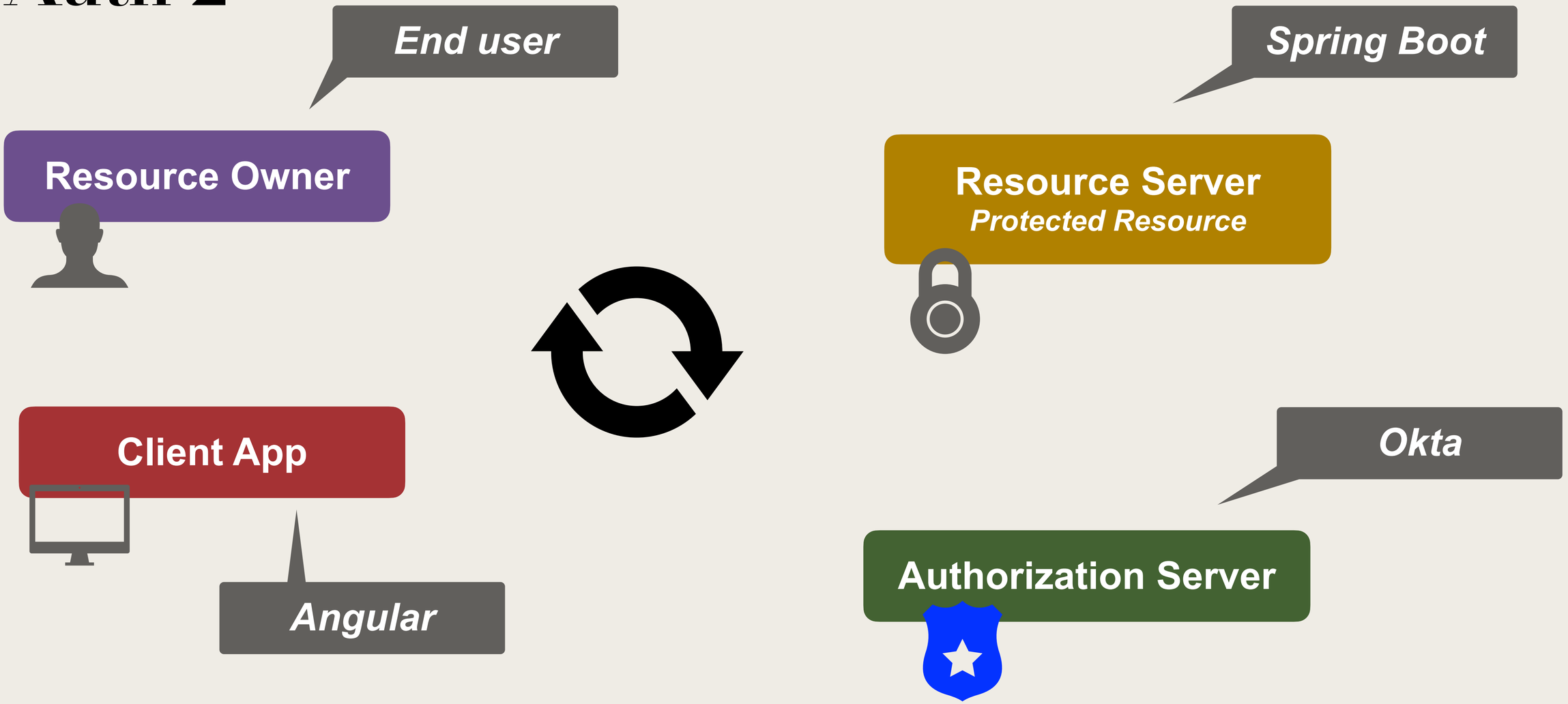
OAuth 2



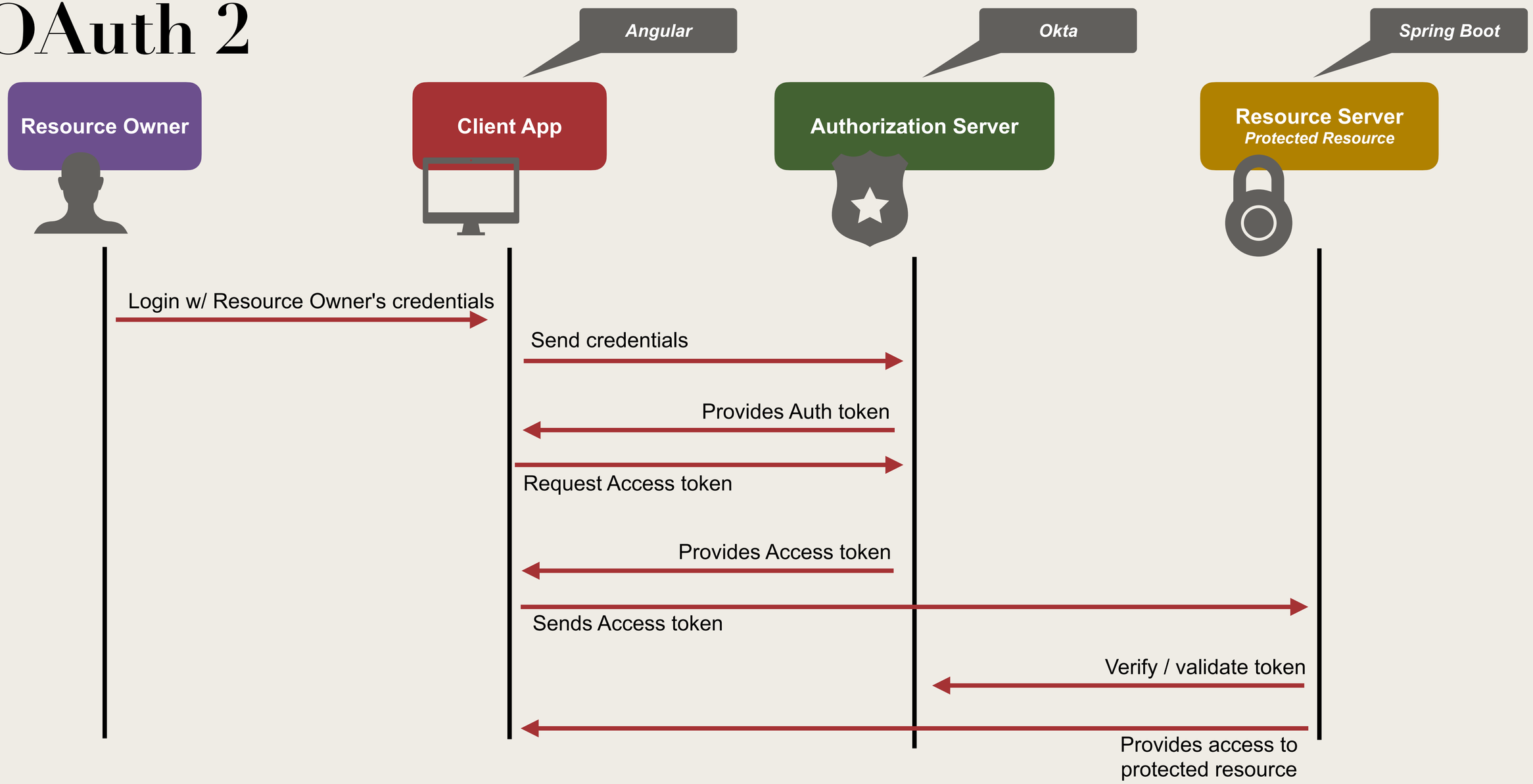
Resource Server

- The "Resource Server" is the app that is hosting our protected resources
  - In our case, it is our Spring Boot app
- The "Resource Server" manages security using access tokens (JWT)
- The access tokens are validated with the "Authorization Server" (Okta)

OAuth 2



OAuth 2



Client sending access token

- Client sends the access token as an HTTP request header

Step 4: Protect endpoints in Spring Security configuration class

File: SecurityConfiguration.java

```
package com.luv2code.ecommerce.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws E

        // protect endpoint /api/orders
        http.authorizeRequests()
                .antMatchers("/api/orders/**")
                .authenticated()
                .and()
                .oauth2ResourceServer()
                .jwt();

        // add CORS filter
        http.cors();

    }
}
```

Protect the endpoint ... only accessible to authenticated users

Configures OAuth2 Resource Server support

Enables JWT-encoded bearer token support