

# OAuth2 Authentication with Spring Boot and Spring Security with H2 database

Github link: [hacker123shiva/oauth2-project: OAuth2 Authentication with Spring Boot and Spring Security with H2 database \(github.com\)](https://github.com/hacker123shiva/oauth2-project)

LinkedIn: <https://www.linkedin.com/in/shivasrivastava1/>

Java Dev Community: <https://www.linkedin.com/groups/14530255/>

## Introduction

In this tutorial, we will walk you through how to integrate OAuth2 authentication into a Spring Boot application using Spring Security. We will configure GitHub as an OAuth2 provider, securing API endpoints and allowing authenticated users to perform CRUD operations on a student entity.

```
OAuth-2-Authentication/
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── com.telusko
│   │   │   │   ├── OAuth2AuthenticationApplication.java
│   │   │   │   ├── config
│   │   │   │   │   ├── SecurityConfig.java
│   │   │   │   ├── controller
│   │   │   │   │   ├── StudentController.java
│   │   │   │   ├── dao
│   │   │   │   │   ├── StudentRepository.java
│   │   │   │   ├── entity
│   │   │   │   │   ├── Student.java
│   │   │   │   ├── exception
│   │   │   │   │   ├── ErrorResponse.java
│   │   │   │   │   ├── GlobalExceptionHandler.java
│   │   │   │   │   ├── StudentNotFoundException.java
│   │   │   │   ├── service
│   │   │   │   │   ├── StudentService.java
│   │   ├── resources
│   │   │   ├── static
│   │   │   ├── templates
│   │   │   └── application.properties
│   └── test
```

## 1. OAuth2 Setup with GitHub

To set up OAuth2 with GitHub:

### 1. Create a GitHub OAuth App:

- Go to [GitHub Developer Settings](#).
- Under "OAuth Apps", click "New OAuth App".

Register a new OAuth application

Application name \*

Something users will recognize and trust.

Homepage URL \*

The full URL to your application homepage.

Application description

This is displayed to all users of your application.

Authorization callback URL \*

Your application's callback URL. Read our [OAuth documentation](#) for more information.

☐ Enable Device Flow

Allow this OAuth App to authorize users via the Device Flow.

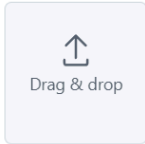
Read the [Device Flow documentation](#) for more information.

Register application

[Cancel](#)

- Provide details:
  - **Application name:** Your app's name.
  - **Homepage URL:** <http://localhost:8080> (for local development).

#### Application logo



Upload new logo

You can also drag and drop a picture from your computer.

#### Application name \*

outh2-first-project

Something users will recognize and trust.

#### Homepage URL \*

http://localhost:8080

The full URL to your application homepage.

#### Application description

This is for trial purpose

This is displayed to all users of your application.

#### Authorization callback URL \*

http://localhost:8080/login/oauth2/code/github

Your application's callback URL. Read our [OAuth documentation](#) for more information.

#### ☐ Enable Device Flow

Allow this OAuth App to authorize users via the Device Flow.

Read the [Device Flow documentation](#) for more information.

### ■ Authorization callback URL:

<http://localhost:8080/login/oauth2/code/github> (this must match your `redirect-uri` in `application.properties`).

You can list your application in the [GitHub Marketplace](#) so that other users can discover it.

List this application in the Marketplace

1 user

Revoke all user tokens

#### Client ID

0v231iKvoQpS9FZ9d5w8

#### Client secrets

Generate a new client secret



Client secret

\*\*\*\*\*332c-f0f8

Added yesterday by hacker123shiva

Last used within the last week

You cannot delete the only client secret. Generate a new client secret first.

Delete

## 2. Get Client ID and Secret:

- After registering, GitHub will provide you with a **Client ID** and **Client Secret**.
- Copy these and add them to `application.properties` as shown above.

## 3. Spring Security OAuth2 Configuration: Spring Boot automatically configures OAuth2 login when `spring-boot-starter-oauth2-client` is included as a dependency.

The flow is:

- User is redirected to GitHub for authentication.
- After successful authentication, GitHub redirects back to the `redirect-uri` with an authorization code.
- Spring exchanges this code for an access token and retrieves the user's details.

## 2 Setting Up the Project

The first step is to create a Spring Boot project with the necessary dependencies.

### Maven Dependencies

```
<dependencies>
  <!-- Spring Data JPA for database access -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <!-- Spring Security OAuth2 Client -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
  </dependency>

  <!-- Spring Security Core -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>

  <!-- Spring Web for creating REST APIs -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <!-- H2 Database (In-memory) -->
```

```

    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>
  </dependencies>

```

## pom.xml

```

<?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>3.2.10</version>
    <relativePath/> <!-- Lookup parent from repository -->
  </parent>
  <groupId>com.telusko</groupId>
  <artifactId>shiva</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Outh-2-Authentication</name>
  <description>This is feature of restclient</description>
  <url/>
  <licenses>
    <license/>
  </licenses>
  <developers>
    <developer/>
  </developers>
  <scm>
    <connection/>
    <developerConnection/>
    <tag/>
    <url/>
  </scm>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>

```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</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-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</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>
```

```

</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
<groupId>org.projectlombok</groupId>
                                <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

## 2. Configuring OAuth2 with GitHub

### Application Properties

In the `application.properties` file, add the following OAuth2 configurations to enable GitHub login:

```

# GitHub OAuth2 configuration
spring.security.oauth2.client.registration.github.client-id=your-client-id
spring.security.oauth2.client.registration.github.client-secret=your-client-secret
spring.security.oauth2.client.registration.github.scope=user
spring.security.oauth2.client.registration.github.redirect-uri=http://localhost:8080/login/oauth2/code/github
spring.security.oauth2.client.registration.github.authorization-grant-type=authorization_code

```

### Explanation:

- `client-id`: The client ID obtained from GitHub when you register your application.

- **client-secret**: The secret key provided by GitHub.
- **scope**: Defines what data the OAuth2 provider will share with your app (in this case, user information).
- **redirect-uri**: The URL where GitHub will redirect users after they authenticate. Ensure this matches the URI registered in your GitHub OAuth app.

## application.properties

```
# H2 Database Configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
# H2 Console Configuration (for in-memory database access via browser)
spring.h2.console.enabled=true

# OAuth2 Google Configuration
#spring.security.oauth2.client.registration.google.client-id=1234567890-abc123def456.apps.googleusercontent.com
#spring.security.oauth2.client.registration.google.client-secret=GOCSPX-abc123def456
#spring.security.oauth2.client.registration.google.scope=profile, email
#spring.security.oauth2.client.registration.google.redirect-uri=http://localhost:8080/Login/oauth2/code/google
#spring.security.oauth2.client.registration.google.authorization-grant-type=authorization_code

# OAuth2 GitHub Configuration
spring.security.oauth2.client.registration.github.client-id=0v23liKvoQpS9F19d8w8
spring.security.oauth2.client.registration.github.client-secret=de8214c9ca66f999ad784036afa24301332cf0f8
spring.security.oauth2.client.registration.github.scope=user
spring.security.oauth2.client.registration.github.redirect-uri=http://localhost:8080/Login/oauth2/code/github
spring.security.oauth2.client.registration.github.authorization-grant-type=authorization_code

# JPA Configuration for H2 Database
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```



### 3. Security Configuration

Create the `SecurityConfig` class to secure the API and configure OAuth2 login.

```
package com.telusko.config;
import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.context.annotation.Configuration;
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .authorizeHttpRequests(authz -> authz
                .requestMatchers("/h2-console/**").permitAll() // Allow
access to H2 console without authentication
                .anyRequest().authenticated() // Secure all
other endpoints
            )
            .oauth2Login() // Enable OAuth2 Login
            .and()
            .csrf().disable() // Disable CSRF protection for H2 console
access
            .headers().frameOptions().disable(); // Allow H2 console to be
displayed
        return http.build();
    }
}
```

#### Explanation:

- `authorizeHttpRequests()`: Configures which endpoints are secured. The H2 console is publicly accessible, but all other API endpoints require authentication.
- `oauth2Login()`: Enables OAuth2 login via the configured provider (GitHub in this case).
- `csrf().disable()`: Disables CSRF protection, often necessary when using H2 for testing.
- `frameOptions().disable()`: Allows H2 console to display properly in the browser.

## 4. Creating the Student Entity and Repository

Define the `Student` entity and the `StudentRepository` to interact with the H2 database.

### Student Entity

#### Student

```
package com.telusko.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.SequenceGenerator;
import jakarta.persistence.Table;
import jakarta.persistence.Transient;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
@Entity
@Table(name = "StudentTable")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "student_seq")
    @SequenceGenerator(name = "student_seq", sequenceName = "student_sequence",
allocationSize = 1, initialValue = 2115800000)
    private int id;
    private String name;
    private String email;
    private int age;
}
```

### Repository

```
public interface StudentRepository extends JpaRepository<Student, Integer>
{
}
```

## 5. Building the Student API and Service layer

Now, let's create a controller that exposes the API endpoints for managing student records. This controller will only allow authenticated users to perform operations like creating, reading, updating, and deleting students.

```
package com.telusko.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.security.oauth2.core.oidc.user.OidcUser;
import org.springframework.web.bind.annotation.*;
import com.telusko.entity.Student;
import com.telusko.service.StudentService;
import java.util.List;

@RestController
@RequestMapping("/api/students")
public class StudentController {
    @Autowired
    private StudentService studentService;

    // Get all students
    @GetMapping
    public List<Student> getAllStudents() {
        return studentService.getAllStudents();
    }

    // Get student by ID
    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable Integer id) {
        return studentService.getStudentById(id);
    }

    // Save a new student
    @PostMapping
    public Student saveStudent(@RequestBody Student student,
        @AuthenticationPrincipal OidcUser principal) {
        // Ensure only authenticated users can create students
        System.out.println("Authenticated user: " + principal.getEmail());
        return studentService.saveStudent(student);
    }

    // Update a student
    @PutMapping("/{id}")
    public Student updateStudent(@PathVariable Integer id, @RequestBody Student
        updatedStudent) {
        Student existingStudent = studentService.getStudentById(id);
        if (existingStudent != null) {
            existingStudent.setName(updatedStudent.getName());
            existingStudent.setAge(updatedStudent.getAge());
        }
    }
}
```

```

        return studentService.saveStudent(existingStudent);
    }
    return null;
}
// Delete a student
@DeleteMapping("/{id}")
public void deleteStudent(@PathVariable Integer id) {
    studentService.deleteStudent(id);
}
}

```

### Explanation:

- The `@AuthenticationPrincipal` annotation is used to access the authenticated user's details (from GitHub in this case).
- The authenticated user's email is printed and tied to the operations they perform.

### service layer:

```

package com.telusko.service;
import com.telusko.dao.StudentRepository;
import com.telusko.entity.Student;
import com.telusko.exception.StudentNotFoundException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class StudentService {
    @Autowired
    private StudentRepository studentRepository;
    public List<Student> getAllStudents() {
        return studentRepository.findAll();
    }
    public Student getStudentById(int id) {
        return studentRepository.findById(id)
            .orElseThrow(() -> new StudentNotFoundException("Student with ID " +
id + " not found."));
    }
    public Student saveStudent(Student student) {
        return studentRepository.save(student);
    }
    public void deleteStudent(int id) {
        if (!studentRepository.existsById(id)) {
            throw new StudentNotFoundException("Student with ID " + id + " not

```

```

found.");
    }
    studentRepository.deleteById(id);
}
}

```

## 6. Handling Exception

### ErrorResponse Entity

```

package com.telusko.exception;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorResponse {
    private String message;
    private int status;
}

```

### GlobalExceptionHandler class:

```

package com.telusko.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionHandler {

```

```

    @ExceptionHandler(StudentNotFoundException.class)
    public ResponseEntity<ErrorResponse>
    handleStudentNotFoundException(StudentNotFoundException ex) {
        ErrorResponse errorResponse = new ErrorResponse(ex.getMessage(),
        HttpStatus.NOT_FOUND.value());
        return
        ResponseEntity.status(HttpStatus.NOT_FOUND).body(errorResponse);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleGenericException(Exception
    ex) {
        ErrorResponse errorResponse = new ErrorResponse("An unexpected
        error occurred: " + ex.getMessage(),
        HttpStatus.INTERNAL_SERVER_ERROR.value());
        return
        ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(errorResponse)
        ;
    }
}

```

## StudentNotFoundException

```

package com.telusko.exception;
public class StudentNotFoundException extends RuntimeException {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public StudentNotFoundException(String message) {
        super(message);
    }
}

```

## 7. Testing with Postman

Type

OAuth 2.0

The authorization data will be automatically generated when you send the request. Learn more about [OAuth 2.0](#) authorization.

Add authorization data to

Request Headers

Current Token

This token is only available to you. Sync the token to let collaborators on this request use it.

Token

GitHub OAuth2 Token

Header Prefix ⓘ

e.g. Bearer

Auto-refresh Token

Your expired token will be auto-refreshed before sending a request.

Share Token

This will allow anyone with access to this request to view and use it.

Configure New Token

Token Name

GitHub OAuth2 Token

POST

http://localhost:8080/api/students

Send

Params

Auth

Headers (11)

Body

Pre-req.

Tests

Settings

Cookies

Token Name

GitHub OAuth2 Token

Grant type

Authorization Code

Callback URL ⓘ

http://localhost:8080/login/oauth...

☐ Authorize using browser

Auth URL ⓘ

https://github.com/login/oauth/a...

Access Token URL ⓘ

https://github.com/login/oauth/a...

Client ID ⓘ

Ov23ilKvoQpS9FZ9d5w8

Client Secret ⓘ

de8214c9ca66f999ad78403f..

Scope ⓘ

user

State ⓘ

State

Client Authentication ⓘ

Send client credentials in body

## Auth Request ⓘ


	Key	Value
	Create parameter	Value

## Token Request ⓘ

	Key	Value	Send In
	Create parameter	Value	

## Refresh Request ⓘ

	Key	Value	Send In
	Create parameter	Value	

 Clear cookies ⓘ

Get New Access Token

## Get new access token



## Authentication complete

This dialog box will automatically close in 3...

Proceed

Key

Value

Send In



MANAGE ACCESS TOKENS

All Tokens

Delete

▼

GitHub OAuth2 Token

GitHub OAuth2 Token

GitHub OAuth2 Token

GitHub OAuth2 Token

GitHub OAuth2 Token

GitHub OAuth2 Token

GitHub OAuth2 Token

GitHub OAuth2 Token

GitHub OAuth2 Token

Token Details

Token Name

GitHub OAuth2 Token

Access Token

gho\_I24qNG9Klrh9vAKDN36D9qNTn4mXSB0a6Kt0

Token Type

bearer

scope

user

access\_token\_url

https://github.com/login/oauth/access\_token

client\_id

Ov23liKvoQpS9FZ9d5w8

client\_secret

de8214c9ca66f999ad784036afa24301332cf0f8

timestamp

1727278062019

Use Token

Type

OAuth 2.0

▼

The authorization data will be automatically generated when you send the request. [Learn more about OAuth 2.0 authorization.](#)

Add authorization data to

Request Headers

▼

Current Token

This token is only available to you. Sync the token to let collaborators on this request use it.

Token

GitHub OAuth2 Token

▼

gho\_I24qNG9Klrh9vAKDN36D9c ...

Header Prefix ⓘ

Bearer

Auto-refresh Token

Your expired token will be auto-refreshed before sending a request.

☐

Share Token

☐

Once you've implemented the OAuth2 configuration, you'll need to test the authentication and API access using Postman.

## Steps:

### 1. Get Access Token:

- Open Postman and click "Authorization."

- Choose "OAuth 2.0" and click "Get New Access Token."
- Use the client ID and secret, set the authorization grant type to **Authorization Code**, and provide the redirect URI.
- Once authenticated via GitHub, Postman will generate an access token.

## 2. Access the API:

- Use the **GET** and **POST** endpoints with the **Bearer** token (OAuth access token) to interact with the Student API.
- Set the token in the "Authorization" tab as a "Bearer Token."

## 8. Through browser

<http://localhost:8080/h2-console>

The screenshot displays the H2 database console web interface. At the top, there's a navigation bar with a language dropdown set to 'English' and links for 'Preferences', 'Tools', and 'Help'. Below this is the 'Login' section, which includes a 'Saved Settings' dropdown menu currently showing 'Generic H2 (Embedded)'. Underneath, there's a 'Setting Name' field with the same text and buttons for 'Save' and 'Remove'. The main configuration area contains fields for 'Driver Class' (org.h2.Driver), 'JDBC URL' (jdbc:h2:mem:testdb), 'User Name' (sa), and 'Password'. At the bottom of this section are 'Connect' and 'Test Connection' buttons.

The bottom half of the interface features a database structure tree on the left, showing the 'jdbc:h2:mem:testdb' database with its schema, tables, sequences, and users. The main area on the right is for executing SQL statements, with buttons for 'Run', 'Run Selected', 'Auto complete', and 'Clear'. Below the SQL statement input area, there's a section titled 'Important Commands' which lists several shortcuts: a question mark icon for 'Displays this Help Page', a magnifying glass for 'Shows the Command History', 'Ctrl+Enter' for 'Executes the current SQL statement', 'Shift+Enter' for 'Executes the SQL statement defined by the text selection', and 'Ctrl+Space' for 'Auto complete'.

## Insert some data into student table:


jdbc:h2:mem:testdb  
+ STUDENT\_TABLE  
+ INFORMATION\_SCHEMA  
+ Sequences  
+ Users  
i H2 2.2.224 (2023-09-17)

Run Run Selected Auto complete Clear SQL statement:

```
INSERT INTO student_table (id, email, name, age)
VALUES (126, 'arjun@gmail.com', 'arjun', 23);
```

```
INSERT INTO student_table (id, email, name, age)
VALUES (126, 'arjun@gmail.com', 'arjun', 23);
Update count: 1
```

<http://localhost:8080/api/students>

→  github.com/login?client\_id=Ov23liKvoQpS9FZ9d5w8&return\_to=%2Flogin%2Foauth%2Fauthorize%3Fclient\_id%3DOv23liKvoQpS9FZ9d5w8%26redirect\_uri%3Dhttp%253A%25...



Sign in to GitHub  
to continue to outh2-first-project

Username or email address

Password

[Forgot password?](#)

Sign in

[Sign in with a passkey](#)

New to GitHub? [Create an account](#)

github.com/login/oauth/authorize?client\_id=Ov23liKvoQpS9FZ9d5w8&redirect\_uri=http%3A%2F%2Flocalhost%3A8080%2Flogin%2Foauth2%



You are being redirected to the authorized application.  
If your browser does not redirect you back, please visit [this setup page](#) to continue.

localhost:8080/api/students?continue

```
[
  {
    "id": 123,
    "name": "shiva",
    "email": "shiva@gmail.com",
    "age": 23
  },
  {
    "id": 126,
    "name": "arjun",
    "email": "arjun@gmail.com",
    "age": 23
  }
]
```

## 9. Conclusion

By the end of this guide, you will have a secure API that leverages OAuth2 authentication using GitHub, with full access control over API operations. The Student entity is managed through a REST API, and OAuth2 ensures only authenticated users can create or update data.