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)]
LinkedId: https://www.linkedin.com/in/shivasrivastava1/
Java Dev Community: https://www.linkedin.com/groups/14530255/

Introduction

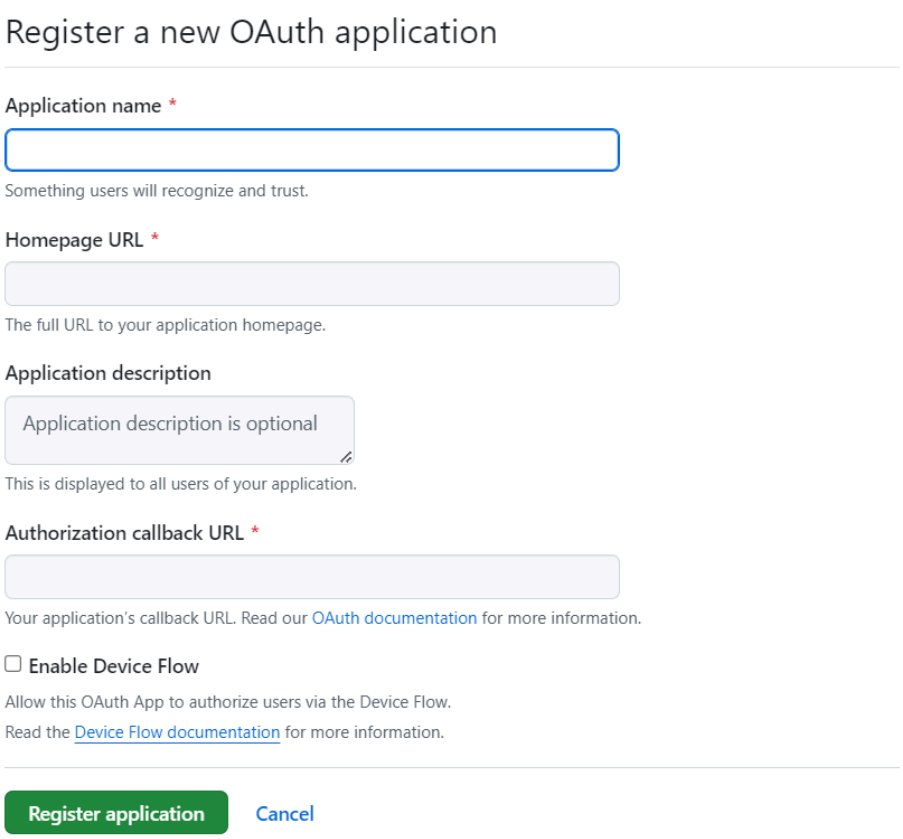
In this tutorial, we will walk you through how to integrate OAuth2 authentication into a Spring Boot application using Spring Security. We will configure GitHub as an OAuth2 provider, securing API endpoints and allowing authenticated users to perform CRUD operations on a student entity.

```
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".



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

Application logo

Drag & drop

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

Ov23liKvoQpS9FZ9d5w8

Client secrets

Generate a new client secret

Client secret

*****332cf0f8

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://local
host: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-abc123def45
6.apps.googleusercontent.com
#spring.security.oauth2.client.registration.google.client-secret=GOCSPX-abc123def45
6
#spring.security.oauth2.client.registration.google.scope=profile, email
#spring.security.oauth2.client.registration.google.redirect-uri=http://localhost:80
80/login/oauth2/code/google
#spring.security.oauth2.client.registration.google.authorization-grant-type=authori
zation_code

# OAuth2 GitHub Configuration
spring.security.oauth2.client.registration.github.client-id=Ov23liKvoQpS9F19d8w8
spring.security.oauth2.client.registration.github.client-secret=de8214c9ca66f999ad7
84036afa24301332cf0f8
spring.security.oauth2.client.registration.github.scope=user
spring.security.oauth2.client.registration.github.redirect-uri=http://localhost:808
0/login/oauth2/code/github
spring.security.oauth2.client.registration.github.authorization-grant-type=authoriz
ation_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:

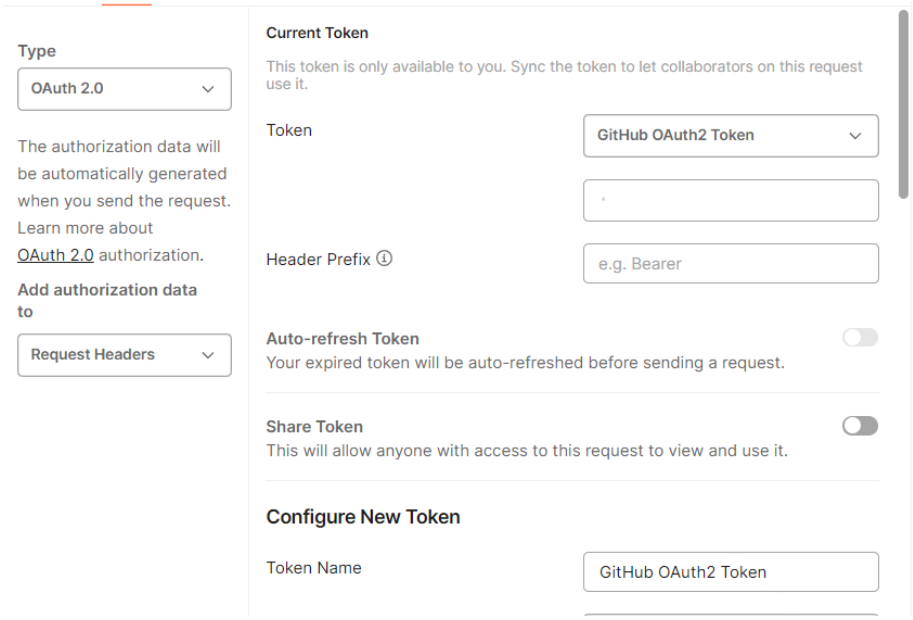
```
package com.telusko.exception;

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 {
```

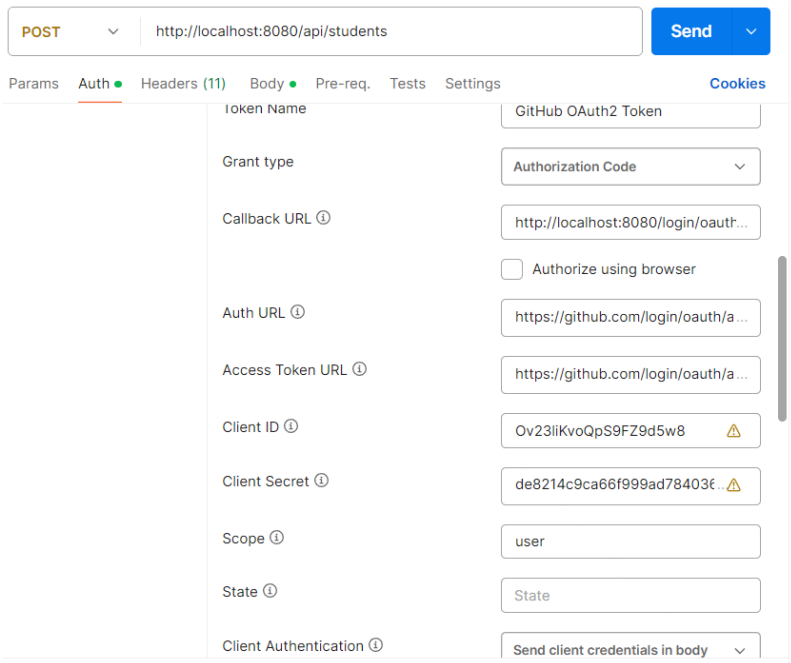
```
    @ExceptionHandler(StudentNotFoundException.class)
    public ResponseEntity<ErrorResponse>
handleStudentNotFoundException(StudentNotFoundException ex) {
        ErrorResponse errorResponse = new ErrorResponse(ex.getMessage(),
HttpStatus.NOT_FOUND.value());
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(errorResponse);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

    @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





Params Auth Headers (11) Body Pre-req. Tests Settings Cookies

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

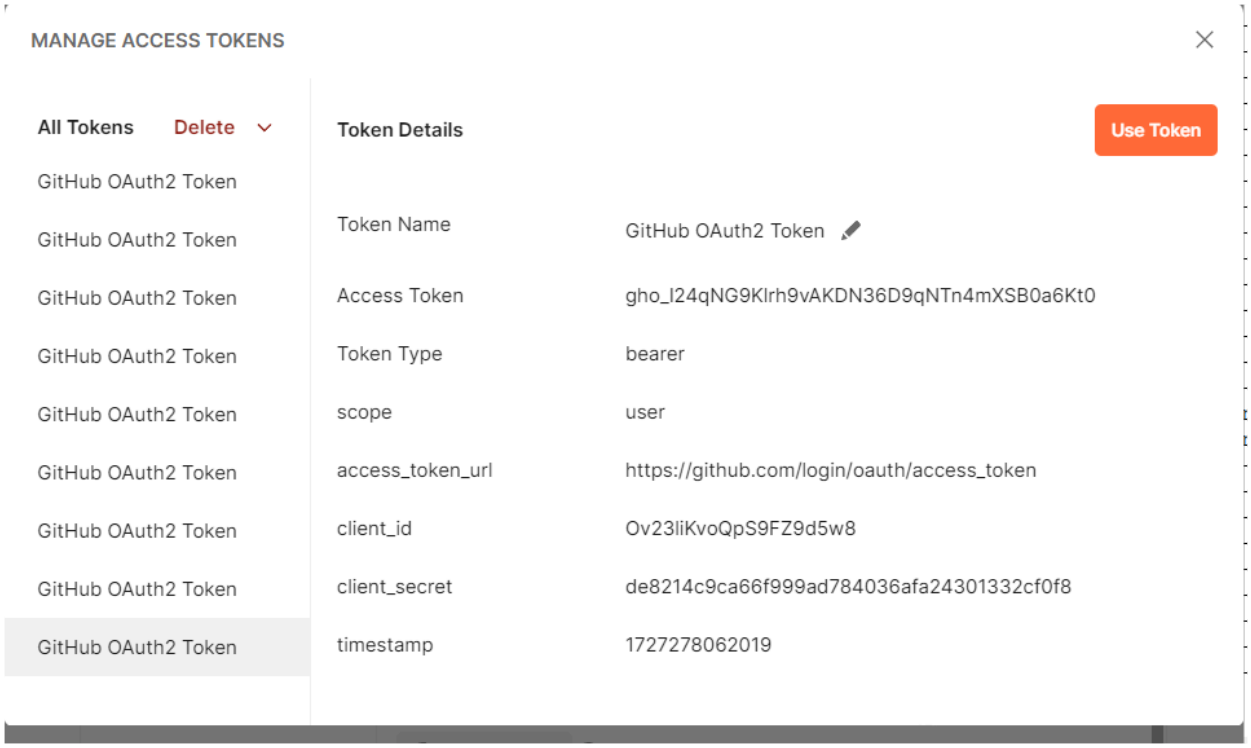
Get new access token

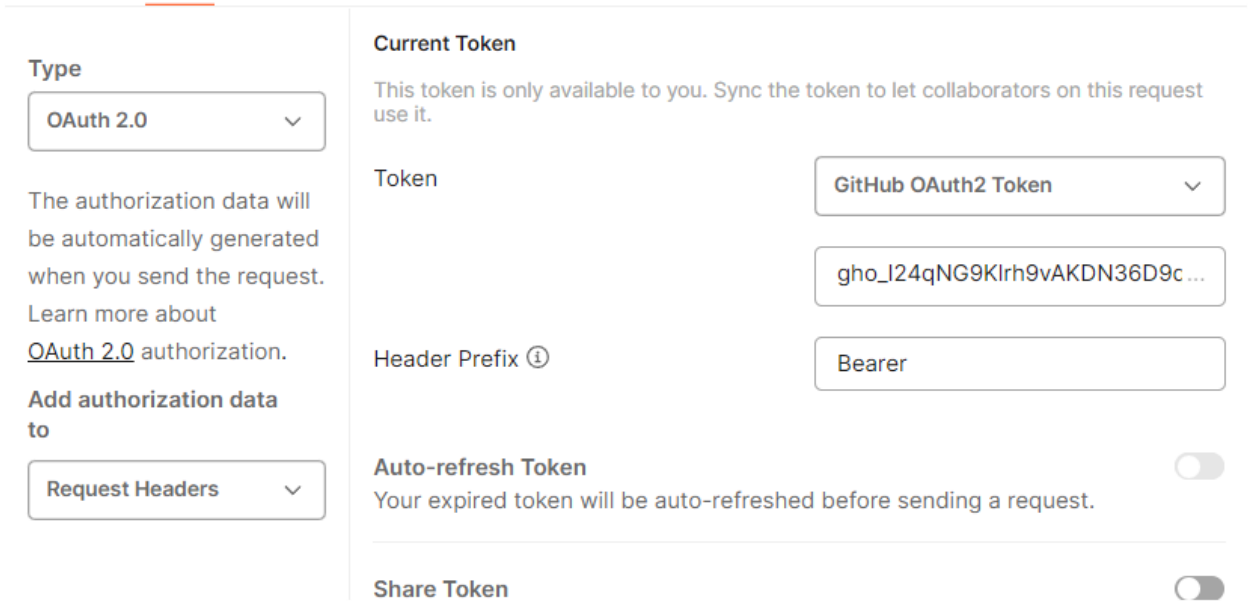
Authentication complete

This dialog box will automatically close in 3...

Proceed

Key Value Send In





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

Steps:

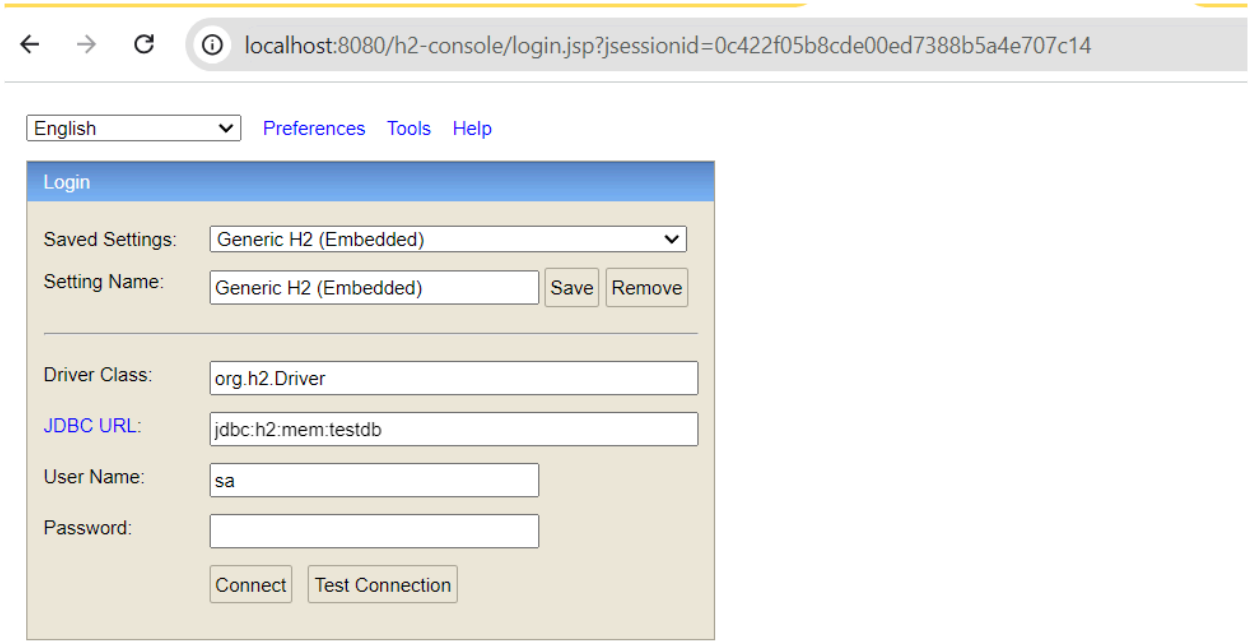
1. **Get Access Token**:
    - Open Postman and click "Authorization."

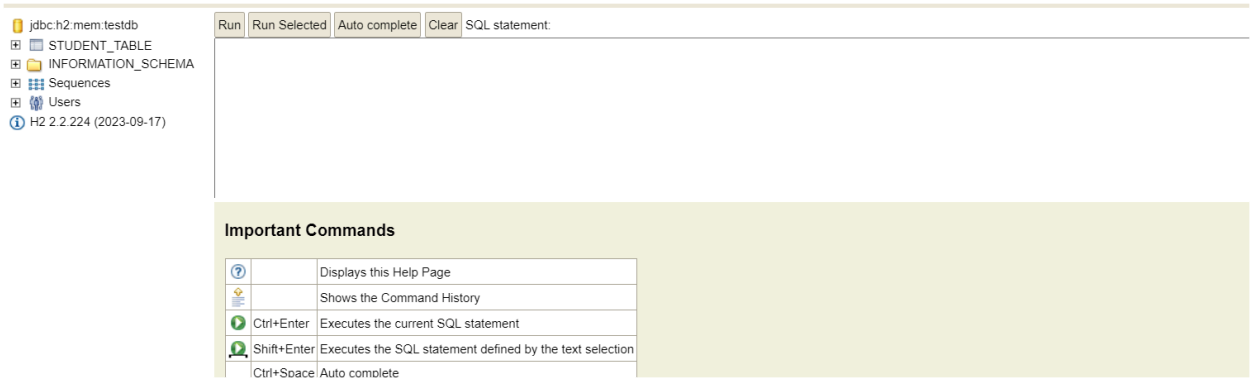
- Choose "OAuth 2.0" and click "Get New Access Token."
- Use the client ID and secret, set the authorization grant type to `Authorization Code`, and provide the redirect URI.
- Once authenticated via GitHub, Postman will generate an access token.

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."

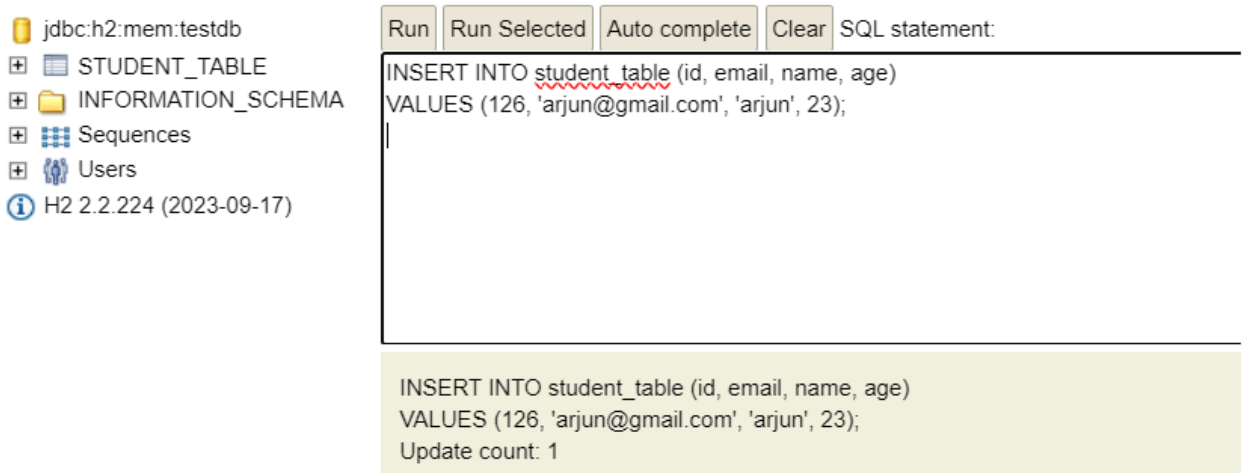
8. Through browser

http://localhost:8080/h2-console

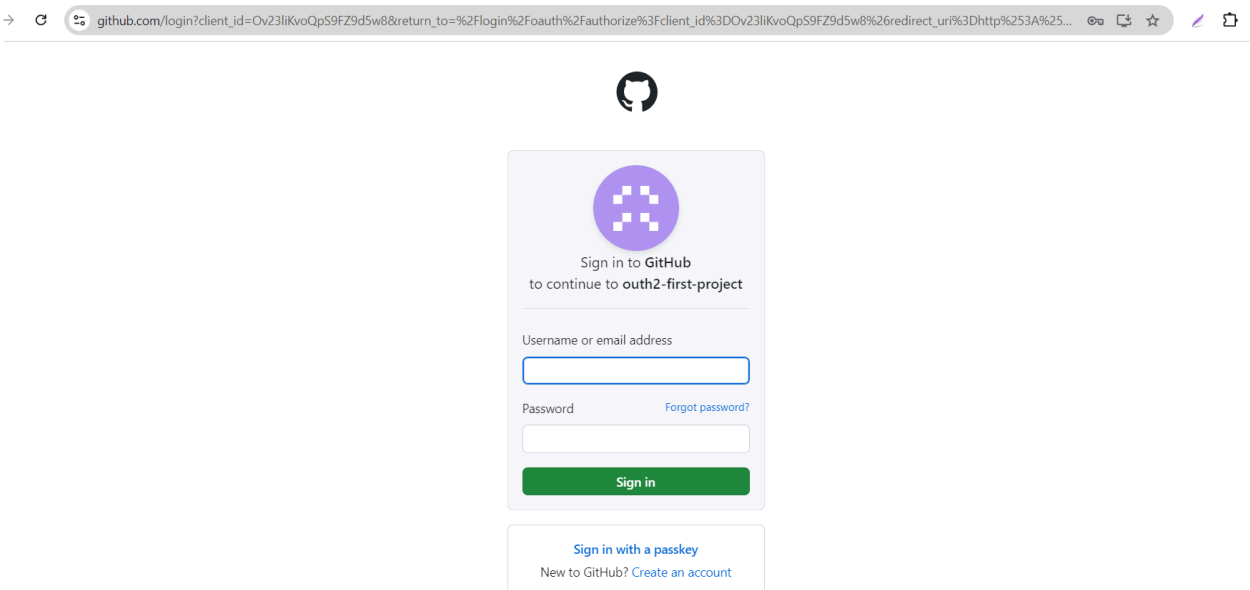


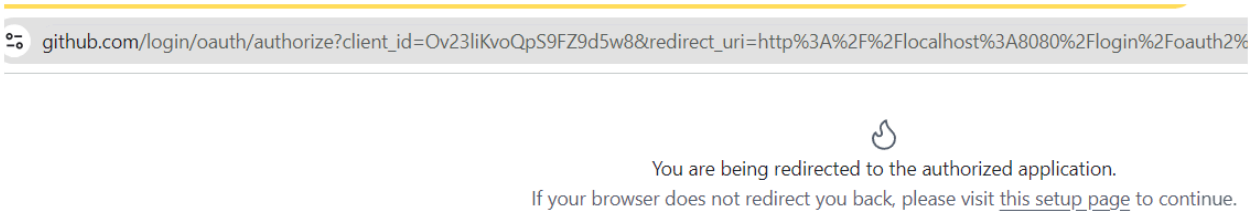


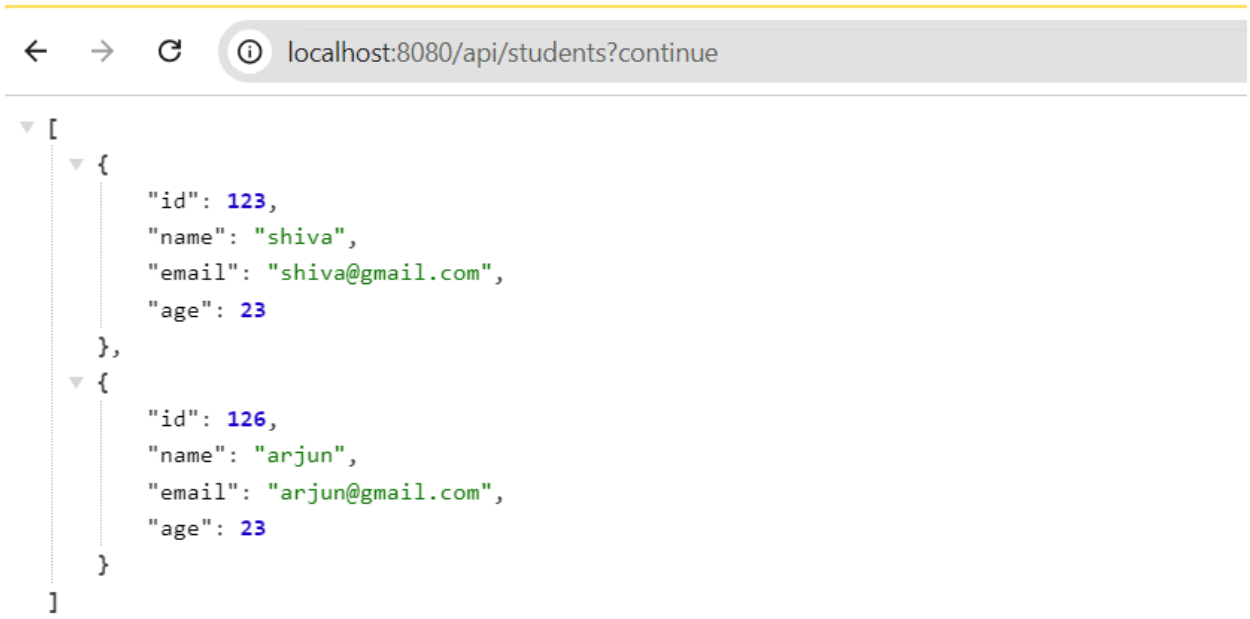
Insert some data into student table:



http://locahost:8080/api/students







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.