**Spring Boot и OAuth2**

**https://spring.io/guides/tutorials/spring-boot-oauth2/#\_social\_login\_simple**

В этом руководстве показано, как создать пример приложения, выполняющего различные действия с «социальным входом» с использованием [OAuth 2.0](https://tools.ietf.org/html/rfc6749) и [Spring Boot](https://projects.spring.io/spring-boot/) .

Он начинается с простого единого входа с одним провайдером и работает до клиента с выбором поставщиков аутентификации: [GitHub](https://github.com/settings/developers) или [Google](https://developers.google.com/identity/protocols/OpenIDConnect) .

Все образцы представляют собой одностраничные приложения, использующие Spring Boot и Spring Security на внутренней стороне. Все они также используют простой [jQuery](https://jquery.org/) на переднем конце. Но изменения, необходимые для преобразования в другую среду JavaScript или использования рендеринга на стороне сервера, будут минимальными.

Все примеры реализованы с использованием встроенной поддержки OAuth 2.0 в [Spring Boot](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/" \l "boot-features-security-oauth2) .

Есть несколько примеров, дополняющих друг друга, добавляя новые функции на каждом этапе:

* [**simple**](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_simple) : очень простое статическое приложение с простой домашней страницей и безусловным входом через свойства конфигурации Spring Boot OAuth 2.0 (если вы посетите домашнюю страницу, вы будете автоматически перенаправлены на GitHub).
* [**click**](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_click) : добавляет явную ссылку, которую пользователь должен щелкнуть для входа в систему.
* [**logout**](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_logout) : добавляет ссылку для выхода из системы для авторизованных пользователей.
* [**two-provider**](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_two_providers) : добавляет второго провайдера входа, чтобы пользователь мог выбрать на домашней странице, какой из них использовать.
* [**custom-error**](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_custom_error) : добавляет сообщение об ошибке для неаутентифицированных пользователей и настраиваемую аутентификацию на основе API GitHub.

|  |  |
| --- | --- |
|  | Изменения, необходимые для перехода от одного приложения к другому в релейной диаграмме функций, можно отслеживать в [исходном коде](https://github.com/spring-guides/tut-spring-boot-oauth2) . Каждая версия приложения представляет собой отдельный каталог, чтобы вы могли сравнить их различия. |

Каждое приложение можно импортировать в IDE. Вы можете запустить main метод, SocialApplication чтобы запустить приложение. Все они создают домашнюю страницу по [адресу http: // localhost: 8080](http://localhost:8080/) (и все требуют, чтобы у вас была хотя бы учетная запись GitHub и Google, если вы хотите войти в систему и просмотреть контент).

Вы также можете запускать все приложения в командной строке, используя mvn spring-boot:run или создав файл jar и запустив его с помощью mvn package и java -jar target/\*.jar(согласно [документации Spring Boot](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-first-application-run) и другой [доступной документации](https://spring.io/guides/gs/spring-boot/) ). Нет необходимости устанавливать Maven, если вы используете [оболочку](https://github.com/takari/maven-wrapper) на верхнем уровне, например

$ cd simple

$ ../mvnw package

$ java -jar target/\*.jar

|  |  |
| --- | --- |
|  | Все приложения работают, localhost:8080потому что они будут использовать клиентов OAuth 2.0, зарегистрированных в GitHub и Google для этого адреса. Чтобы запускать их на другом хосте или порту, вам необходимо зарегистрировать свои приложения таким образом. Если вы используете значения по умолчанию, нет опасности утечки ваших учетных данных за пределы localhost. Но будьте осторожны с тем, что вы публикуете в Интернете, и не помещайте свои собственные регистрации приложений в общедоступную систему контроля версий. |

Единый вход с помощью GitHub

В этом разделе вы создадите минимальное приложение, использующее GitHub для аутентификации. Это будет довольно просто, если воспользоваться функциями автоконфигурации в Spring Boot.

Создание нового проекта

Во-первых, вам нужно создать приложение Spring Boot, что можно сделать несколькими способами. Самый простой - перейти на [https://start.spring.io](https://start.spring.io/) и создать пустой проект (выбрав в качестве отправной точки зависимость «Интернет»). Аналогично, сделайте это в командной строке:

$ mkdir ui && cd ui

$ curl https://start.spring.io/starter.tgz -d style=web -d name=simple | tar -xzvf -

Затем вы можете импортировать этот проект в свою любимую среду IDE (по умолчанию это обычный Java-проект Maven) или просто работать с файлами и mvnв командной строке.

**Домашняя страница (если есть желание изменить внешний вид по умолчанию)**

В вашем новом проекте создайте index.html в папке src/main/resources/static. Вам следует добавить несколько таблиц стилей и ссылок JavaScript, чтобы результат выглядел так:

**index.html**

<!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>

Ничего из этого не требуется для демонстрации возможностей входа в систему OAuth 2.0, но было бы неплохо иметь приятный пользовательский интерфейс в конце, поэтому вы также можете начать с некоторых базовых вещей на домашней странице.

Если вы запустите приложение и загрузите домашнюю страницу, вы заметите, что таблицы стилей не загружены. Итак, вам нужно добавить их, добавив jQuery и Twitter Bootstrap:

**pom.xml**

<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>

Последняя зависимость - это "локатор" веб-страниц, который предоставляется сайтом веб-страниц в виде библиотеки. Spring может использовать локатор для поиска статических ресурсов в веб-файлах без необходимости знать точные версии (отсюда и /webjars/\*\*ссылки без версий в файле index.html). Локатор webjar активирован по умолчанию в приложении Spring Boot, если вы не отключите автоконфигурацию MVC.

После внесения этих изменений у вас должна получиться красивая домашняя страница для вашего приложения.

**Защита приложения с помощью GitHub и Spring Security**

Чтобы сделать приложение безопасным, вы можете:

**1. Добавить Spring Security в качестве зависимости.**

Поскольку вы хотите выполнить "социальный" вход (делегировать GitHub), вам следует включить стартер клиента Spring Security OAuth 2.0:

**pom.xml**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-oauth2-client</artifactId>

</dependency>

Добавив это, он по умолчанию защитит ваше приложение с помощью OAuth 2.0.

Затем вам необходимо настроить приложение для использования GitHub в качестве поставщика аутентификации. Для этого сделайте следующее:

* [Добавить новое приложение GitHub](https://spring.io/guides/tutorials/spring-boot-oauth2/#github-register-application)
* [Настроить application.yml](https://spring.io/guides/tutorials/spring-boot-oauth2/#github-application-config)
* [Загрузите приложение](https://spring.io/guides/tutorials/spring-boot-oauth2/#github-boot-application)

**2. Добавить новое приложение GitHub**

Чтобы использовать систему аутентификации GitHub OAuth 2.0 для входа, вы должны сначала [добавить новое приложение GitHub](https://github.com/settings/developers).

Выберите «Новое приложение OAuth», после чего откроется страница «Зарегистрировать новое приложение OAuth». Введите название и описание приложения. Затем введите домашнюю страницу своего приложения , в данном случае это [http: // localhost: 8080](http://localhost:8080/) . Наконец, укажите URL-адрес обратного вызова авторизации как <http://localhost:8080/login/oauth2/code/github>и нажмите « *Зарегистрировать приложение»*.

URI перенаправления OAuth - это путь в приложении, на который пользовательский агент конечного пользователя перенаправляется обратно после того, как они прошли аутентификацию с помощью GitHub и предоставили доступ к приложению на странице *авторизации приложения*.

|  |  |
| --- | --- |
|  | Перенаправление по умолчанию URI шаблона это {baseUrl}/login/oauth2/code/{registrationId}. ***RegistrationID*** представляет собой уникальный идентификатор для ClientRegistration. |

**3. Настроить application.yml**

Затем, чтобы сделать ссылку на GitHub, добавьте в свой application.yml:

application.yml

spring:

security:

oauth2:

client:

registration:

github:

clientId: github-client-id

clientSecret: github-client-secret

# ...

Просто используйте учетные данные OAuth 2.0, которые вы только что создали с помощью GitHub, заменив github-client-id их идентификатором и github-client-secret клиента секретом клиента.

**4. Загрузите приложение**

После этого изменения вы можете снова запустить приложение и посетить домашнюю страницу по адресу [http: // localhost: 8080](http://localhost:8080/) . Теперь вместо домашней страницы вы должны быть перенаправлены на вход через GitHub. Если вы сделаете это и примете все запрашиваемые авторизации, вы будете перенаправлены обратно в локальное приложение, и будет видна домашняя страница.

Если вы останетесь в системе GitHub, вам не придется повторно проходить аутентификацию с помощью этого локального приложения, даже если вы откроете его в новом браузере без файлов cookie и кэшированных данных. (Вот что означает единый вход.)

|  |  |
| --- | --- |
|  | Если вы работаете в этом разделе с примером приложения, не забудьте очистить кеш браузера от файлов cookie и учетных данных HTTP Basic. Лучший способ сделать это для одного сервера - открыть новое личное окно. |

Предоставлять доступ к этому образцу безопасно, поскольку только приложение, работающее локально, может использовать токены, а объем запрашиваемых им ограничен.

Помните, что вы одобряете, когда входите в подобные Приложения: они могут попросить разрешения сделать больше, чем вам удобно (например, они могут попросить разрешения на изменение ваших личных данных, что вряд ли будет в ваших интересах).

Что сейчас произошло?

1. Приложение, которое вы только что написали, в терминах OAuth 2.0, является *клиентским приложением* , и оно использует [предоставление кода авторизации](https://tools.ietf.org/html/rfc6749#section-4) для получения токена доступа от GitHub (как сервера Авторизации).
2. Приложение использует токен доступа, чтобы запросить у GitHub (как сервера Ресурсов) некоторые личные данные (только то, что вы ему разрешили), включая ваш логин и ваше имя. На этом этапе GitHub действует как сервер Ресурсов.
3. GitHub (как сервер Ресурсов) декодирует отправляемый Приложением токен и проверяет, дает ли он Приложению разрешение на доступ к данным Пользователя. Если этот процесс успешен, отправляет Приложению запрашиваемую информацию.
4. Приложение вставляет данные Пользователя в контекст Spring Security, чтобы вы прошли аутентификацию.

Если вы посмотрите в инструменты браузера (F12 в Chrome или Firefox) и проследите за сетевым трафиком для всех переходов, вы увидите перенаправления назад и вперед с GitHub, и, наконец, вы вернетесь на домашнюю страницу с новым Set-Cookieзаголовком. Этот файл cookie ( JSESSIONIDпо умолчанию) является токеном для ваших данных аутентификации для приложений Spring (или любых других сервлетов).

Итак, у нас есть безопасное приложение, в том смысле, что для просмотра любого контента пользователь должен пройти аутентификацию у внешнего поставщика (GitHub).

Мы бы не хотели использовать это для веб-сайта интернет-банкинга. Но для базовых целей идентификации и разделения контента между разными пользователями вашего сайта это отличная отправная точка. Вот почему такой вид аутентификации очень популярен в наши дни.

В следующем разделе мы собираемся добавить в приложение несколько основных функций. Мы также сделаем более очевидным для пользователей, что происходит, когда они получат начальное перенаправление на GitHub.

Добавить страницу приветствия

В этом разделе вы измените только что созданное [простое](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_simple) приложение, добавив явную ссылку для входа в GitHub. Вместо немедленного перенаправления новая ссылка будет видна на домашней странице, и пользователь может выбрать, войти в систему или остаться без аутентификации. Только после того, как пользователь щелкнет ссылку, отобразится защищенный контент.

Условный контент на главной странице

Чтобы отображать контент при условии, что пользователь аутентифицирован, у вас есть возможность отрисовки на стороне сервера или на стороне клиента.

Здесь вы измените клиентскую сторону с помощью [JQuery](https://jquery.org/) , хотя, если вы предпочитаете использовать что-то еще, не должно быть очень сложно перевести клиентский код.

Чтобы начать работу с динамическим контентом, вам нужно отметить пару HTML-элементов следующим образом:

**index.html**

<div class="container unauthenticated">

With **GitHub:** <a href="/oauth2/authorization/github">click here</a>

</div>

<div class="container authenticated" style="display:none">

Logged in as: <span id="user"></span>

</div>

По умолчанию первый <div>будет отображаться, а второй - нет.

Обратите внимание также на пустое поле <span>с idатрибутом.

Далее вы добавите конечную точку на стороне сервера, которая будет возвращать данные вошедшего в систему пользователя в виде JSON.

Но сначала добавьте следующий код JavaScript, который попадет в эту конечную точку. На основе ответа конечной точки этот JavaScript заполнит <span>тег именем пользователя и <div>соответствующим образом переключит :

**index.html**

<script type="text/javascript">

$.get("/user", function(data) {

$("#user").html(data.name);

$(".unauthenticated").hide()

$(".authenticated").show()

});

</script>

Обратите внимание, что этот JavaScript ожидает вызова конечной точки на стороне сервера /user.

/userEndpoint

Теперь вы добавите только что упомянутую конечную точку на стороне сервера, назвав ее /user. Он отправит обратно текущего вошедшего в систему пользователя, что мы можем довольно легко сделать в нашем основном классе:

**SocialApplication.java**

@SpringBootApplication

@RestController

public class SocialApplication {

@GetMapping("/user")

public Map<String, Object> user(@AuthenticationPrincipal OAuth2User principal) {

return Collections.singletonMap("name", principal.getAttribute("name"));

}

public static void main(String[] args) {

SpringApplication.run(SocialApplication.class, args);

}

}

Обратите внимание на использование @RestController, @GetMapping и OAuth2User вставляют в методе обработчика.

|  |  |
| --- | --- |
|  | Не рекомендуется возвращать целое OAuth2Userв конечной точке, поскольку оно может содержать информацию, которую вы не хотели бы раскрывать клиенту браузера. |

Сделать домашнюю страницу общедоступной

Вам нужно сделать одно последнее изменение, расширив класс **SocialApplication.java**.

Это приложение теперь будет работать нормально и аутентифицироваться, как и раньше, но оно все равно будет перенаправлять перед отображением страницы. Чтобы сделать ссылку видимой, нам также нужно отключить безопасность на домашней странице, расширив WebSecurityConfigurerAdapter:

**SocialApplication.java (**теперь у нас в этом классе будут и ***main*** и ***controller*** и ***блок безопасности*):**

@SpringBootApplication

@RestController

public class SocialApplication extends WebSecurityConfigurerAdapter {

// ...

@Override

protected void configure(HttpSecurity http) throws Exception {

// @formatter:off

http

.authorizeRequests(a -> a

.antMatchers("/", "/error", "/webjars/\*\*").permitAll()

.anyRequest().authenticated()

)

.exceptionHandling(e -> e

.authenticationEntryPoint(new

HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED))

)

.oauth2Login();

// @formatter:on

}

}

Spring Boot придает особое значение WebSecurityConfigurerAdapterклассу, аннотированному @SpringBootApplication: он использует его для настройки цепочки фильтров безопасности, которая содержит процессор аутентификации OAuth 2.0.

Приведенная выше конфигурация указывает на белый список разрешенных конечных точек, при этом для каждой другой конечной точки требуется аутентификация.

Вы разрешаете вход на:

* / - поскольку это страница, которую вы только что сделали динамической, и часть ее содержимого видна неаутентифицированным пользователям,
* /error - поскольку это конечная точка Spring Boot для отображения ошибок,
* /webjars/\*\* - так как вы хотите, чтобы ваш JavaScript запускался для всех посетителей, аутентифицированных или нет.

Однако в этой конфигурации вы не увидите /user. Т.к. все остальные страницы, включая /user, остаются в безопасности из-за  .anyRequest().authenticated()в конце.

Наконец, поскольку мы взаимодействуем с серверной частью через Ajax, мы хотим настроить конечные точки для ответа с кодом 401 вместо стандартного поведения перенаправления на страницу входа. Конфигурация authenticationEntryPoint позволяет добиться этого за нас.

После внесения этих изменений приложение готово, и если вы запустите его и посетите домашнюю страницу, вы должны увидеть красиво оформленную HTML-ссылку «Войти через GitHub». Ссылка приведет вас не напрямую к GitHub, а по локальному пути, который обрабатывает аутентификацию (и отправляет перенаправление на GitHub). После аутентификации вы будете перенаправлены обратно в локальное приложение, где теперь отображается ваше имя (при условии, что вы настроили свои разрешения в GitHub, чтобы разрешить доступ к этим данным).

Добавить кнопку выхода

В этом разделе мы модифицируем созданное нами приложение для [кликов](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_click) , добавляя кнопку, позволяющую пользователю выйти из приложения. Это кажется простой функцией, но для ее реализации требуется немного внимания, поэтому стоит потратить некоторое время на обсуждение того, как именно это сделать. Большинство изменений связано с тем, что мы трансформируем приложение из ресурса только для чтения в ресурс для чтения и записи (для выхода из системы требуется изменение состояния), поэтому такие же изменения потребуются в любом реалистичном приложении, которое не было Это не просто статический контент.

Изменения на стороне клиента

На клиенте нам просто нужно предоставить кнопку выхода и некоторый JavaScript для обратного вызова на сервер, чтобы запросить отмену аутентификации. Во-первых, в разделе пользовательского интерфейса «аутентифицированный» мы добавляем кнопку:

**index.html**

<div class="container authenticated">

Logged in as: <span id="user"></span>

<div>

<button onClick="logout()" class="btn btn-primary">Logout</button>

</div>

</div>

а затем мы предоставляем logout()функцию, на которую он ссылается в JavaScript:

**index.html**

var logout = function () {

$ .post ("/ logout", function () {

$ ("# пользователь"). html ('');

$ (". неавторизованный"). show ();

$ (". аутентифицирован"). hide ();

})

вернуть истину;

}

logout()Функция делает POST к , /logoutа затем очищает содержимое динамического. Теперь мы можем переключиться на серверную часть, чтобы реализовать эту конечную точку.

Добавление конечной точки выхода

Spring Security имеет встроенную поддержку /logoutконечной точки, которая будет делать правильные вещи для нас (очистить сеанс и аннулировать cookie). Чтобы настроить конечную точку, мы просто расширяем существующий configure()метод в нашем WebSecurityConfigurerAdapter:

**SocialApplication.java**

@Override

protected void configure(HttpSecurity http) throws Exception {

// @formatter:off

http

// ... existing code here

.logout(l -> l

.logoutSuccessUrl("/").permitAll()

)

// ... existing code here

// @formatter:on

}

Добавление CSRF.

Конечная точка /logout требует от нас POST к нему, а также для защиты пользователя от Cross Site Request Подделки (CSRF), он требует маркер для включения в запросе. Значение токена связано с текущим сеансом, который обеспечивает защиту, поэтому нам нужен способ получить эти данные в нашем приложении JavaScript.

Многие фреймворки JavaScript имеют встроенную поддержку CSRF (например, в Angular они называют это XSRF), но часто это реализуется немного иначе, чем нестандартное поведение Spring Security. Например, в Angular внешний интерфейс хотел бы, чтобы сервер отправил ему файл cookie с именем «XSRF-TOKEN», и если он это увидит, он отправит значение обратно в виде заголовка с именем «X-XSRF-TOKEN». Мы можем реализовать то же поведение с нашим простым клиентом jQuery, и тогда изменения на стороне сервера будут работать с другими реализациями внешнего интерфейса без изменений или с очень небольшими изменениями. Чтобы научить Spring Security об этом, нам нужно добавить фильтр, который создает cookie.

В нем WebSecurityConfigurerAdapterмы делаем следующее:

**SocialApplication.java**

@Override

protected void configure(HttpSecurity http) throws Exception {

// @formatter:off

http

// ... existing code here

.csrf(c -> c

.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())

)

// ... existing code here

// @formatter:on

}

Добавление CSRF в клиент

Поскольку в этом примере мы не используем фреймворк более высокого уровня, вам необходимо явно добавить токен CSRF, который вы только что сделали доступным в качестве файла cookie из серверной части. Чтобы сделать код немного проще, включите js-cookieбиблиотеку:

pom.xml

<dependency>

<groupId>org.webjars</groupId>

<artifactId>js-cookie</artifactId>

<version>2.1.0</version>

</dependency>

А затем вы можете ссылаться на него в своем HTML:

index.html

<script type="text/javascript" src="/webjars/js-cookie/js.cookie.js"></script>

Наконец, вы можете использовать Cookiesудобные методы в XHR:

index.html

$.ajaxSetup({

beforeSend : function(xhr, settings) {

if (settings.type == 'POST' || settings.type == 'PUT'

|| settings.type == 'DELETE') {

if (!(/^http:.\*/.test(settings.url) || /^https:.\*/

.test(settings.url))) {

// Only send the token to relative URLs i.e. locally.

xhr.setRequestHeader("X-XSRF-TOKEN",

Cookies.get('XSRF-TOKEN'));

}

}

}

});

Готов к работе!

После внесения этих изменений мы готовы запустить приложение и опробовать новую кнопку выхода. Запустите приложение и загрузите домашнюю страницу в новом окне браузера. Щелкните ссылку «Войти», чтобы перейти на GitHub (если вы уже вошли в систему, вы можете не заметить перенаправление). Нажмите кнопку «Выход», чтобы отменить текущий сеанс и вернуть приложение в состояние без аутентификации. Если вам интересно, вы сможете увидеть новые файлы cookie и заголовки в запросах, которыми браузер обменивается с локальным сервером.

Помните, что теперь конечная точка выхода из системы работает с клиентом браузера, тогда все остальные HTTP-запросы (POST, PUT, DELETE и т. Д.) Также будут работать. Так что это должна быть хорошая платформа для приложения с более реалистичными функциями.

Войти через GitHub

В этом разделе вы измените созданное вами приложение для [выхода](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_logout) из [системы](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_logout) , добавив страницу с наклейками, чтобы конечный пользователь мог выбирать между несколькими наборами учетных данных.

Давайте добавим Google в качестве второго варианта для конечного пользователя.

Начальная настройка

Чтобы использовать систему аутентификации Google OAuth 2.0 для входа в систему, вы должны настроить проект в консоли Google API для получения учетных данных OAuth 2.0.

|  |  |
| --- | --- |
|  | [Реализация Google OAuth 2.0](https://developers.google.com/identity/protocols/OpenIDConnect) для аутентификации соответствует спецификации [OpenID Connect 1.0](https://openid.net/connect/) и имеет [сертификат OpenID](https://openid.net/certification/) . |

Следуйте инструкциям на странице [OpenID Connect](https://developers.google.com/identity/protocols/OpenIDConnect) , начиная с раздела «Настройка OAuth 2.0».

После выполнения инструкций «Получить учетные данные OAuth 2.0» у вас должен быть новый клиент OAuth с учетными данными, состоящими из идентификатора клиента и секретного кода клиента.

Установка URI перенаправления

Кроме того, вам нужно будет указать URI перенаправления, как вы это делали ранее для GitHub.

В подразделе «Установить URI перенаправления» убедитесь, что в поле **URI авторизованного перенаправления** установлено значение <http://localhost:8080/login/oauth2/code/google>.

Добавление регистрации клиента

Затем вам нужно настроить клиент, чтобы он указывал на Google. Поскольку Spring Security построен с учетом нескольких клиентов, вы можете добавить наши учетные данные Google вместе с теми, которые вы создали для GitHub:

**application.yml**

spring:

security:

oauth2:

client:

registration:

**github:**

clientId: github-client-id

clientSecret: github-client-secret

**google:**

client-id: google-client-id

client-secret: google-client-secret

Как видите, Google - еще один провайдер, для которого Spring Security предоставляет встроенную поддержку.

Добавление ссылки для входа

В клиенте изменение тривиальное - можно просто добавить еще одну ссылку:

**index.html**

<div class="container unauthenticated">

<div>

With **GitHub:** <a href="/oauth2/authorization/github">click here</a>

</div>

<div>

With **Google:** <a href="/oauth2/authorization/google">click here</a>

</div>

</div>

|  |  |
| --- | --- |
|  | Окончательный путь в URL-адресе должен соответствовать идентификатору регистрации клиента в application.yml. |
|  | Spring Security поставляется со страницей выбора поставщика по умолчанию, на которую можно перейти, указав /loginвместо /oauth2/authorization/{registrationId}. |

Как добавить базу данных локальных пользователей

Многим приложениям необходимо локально хранить данные о своих пользователях, даже если аутентификация делегирована внешнему провайдеру. Мы не показываем здесь код, но это легко сделать в два этапа.

1. Выберите серверную часть для своей базы данных и настройте несколько репозиториев (например, с использованием Spring Data) для настраиваемого Userобъекта, который соответствует вашим потребностям и может быть заполнен полностью или частично с помощью внешней аутентификации.
2. Внедрите и предоставьте возможность OAuth2UserServiceдля вызова сервера авторизации, а также вашей базы данных. Ваша реализация может делегировать реализацию реализации по умолчанию, которая сделает тяжелую работу по вызову сервера авторизации. Ваша реализация должна возвращать что-то, что расширяет ваш настраиваемый Userобъект и реализует OAuth2User.

Подсказка: добавьте в Userобъект поле для ссылки на уникальный идентификатор во внешнем провайдере (не имя пользователя, а что-то уникальное для учетной записи во внешнем провайдере).

Добавление страницы ошибки для неаутентифицированных пользователей

В этом разделе вы измените приложение с [двумя поставщиками,](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_two_providers) которое вы создали ранее, чтобы дать некоторую обратную связь пользователям, которые не могут пройти аутентификацию. В то же время вы расширите логику аутентификации, включив в нее правило, которое разрешает пользователям, только если они принадлежат к определенной организации GitHub. «Организация» - это концепция домена GitHub, но аналогичные правила могут быть разработаны для других провайдеров. Например, с помощью Google вы можете захотеть аутентифицировать пользователей только из определенного домена.

Переход на GitHub

В примере с [двумя поставщиками](https://spring.io/guides/tutorials/spring-boot-oauth2/#_social_login_two_providers) в качестве поставщика OAuth 2.0 используется GitHub:

application.yml

spring:

security:

oauth2:

client:

registration:

github:

client-id: bd1c0a783ccdd1c9b9e4

client-secret: 1a9030fbca47a5b2c28e92f19050bb77824b5ad1

# ...

Обнаружение сбоя аутентификации в клиенте

На клиенте вы можете оставить отзыв для пользователя, который не смог пройти аутентификацию. Чтобы облегчить это, вы можете добавить div, в который вы в конечном итоге добавите информативное сообщение.

index.html

<div class = "container text-dangerous error"> </div>

Затем добавьте вызов к /errorконечной точке, заполнив <div>результат:

index.html

$ .get ("/ error", function (data) {

if (data) {

$ (". ошибка"). html (данные);

} else {

$ (". ошибка"). html ('');

}

});

Функция ошибок проверяет с бэкэндом, есть ли какие-либо ошибки для отображения

Добавление сообщения об ошибке

Чтобы поддержать получение сообщения об ошибке, вам необходимо перехватить его при сбое аутентификации. Для этого вы можете настроить AuthenticationFailureHandler:

protected void configure(HttpSecurity http) throws Exception {

// @formatter:off

http

// ... existing configuration

.oauth2Login(o -> o

.failureHandler((request, response, exception) -> {

request.getSession().setAttribute("error.message", exception.getMessage());

handler.onAuthenticationFailure(request, response, exception);

})

);

}

Вышеупомянутое сохранит сообщение об ошибке в сеансе всякий раз, когда аутентификация не удастся.

Затем вы можете добавить простой /errorконтроллер, например этот:

SocialApplication.java

@GetMapping("/error")

public String error() {

String message = (String) request.getSession().getAttribute("error.message");

request.getSession().removeAttribute("error.message");

return message;

}

|  |  |
| --- | --- |
|  | Это заменит /errorстраницу по умолчанию в приложении, что подходит для нашего случая, но может быть недостаточно сложным для ваших нужд. |

Генерация 401 на сервере

Ответ 401 уже будет поступать от Spring Security, если пользователь не может или не хочет входить в систему с помощью GitHub, поэтому приложение уже работает, если вы не можете пройти аутентификацию (например, отклонив предоставление токена).

Чтобы немного оживить ситуацию, вы можете расширить правило аутентификации, чтобы отклонять пользователей, которые не входят в правильную организацию.

Вы можете использовать GitHub API, чтобы узнать больше о пользователе, поэтому вам просто нужно подключить его к правой части процесса аутентификации.

К счастью, для такого простого случая использования Spring Boot предоставил простую точку расширения: если вы объявите @Beanтип типа OAuth2UserService, он будет использоваться для идентификации участника-пользователя. Вы можете использовать этот хук, чтобы утверждать, что пользователь находится в правильной организации, и генерировать исключение, если нет:

SocialApplication.java

@Bean

public OAuth2UserService<OAuth2UserRequest, OAuth2User> oauth2UserService(WebClient rest) {

DefaultOAuth2UserService delegate = new DefaultOAuth2UserService();

return request -> {

OAuth2User user = delegate.loadUser(request);

if (!"github".equals(request.getClientRegistration().getRegistrationId())) {

return user;

}

OAuth2AuthorizedClient client = new OAuth2AuthorizedClient

(request.getClientRegistration(), user.getName(), request.getAccessToken());

String url = user.getAttribute("organizations\_url");

List<Map<String, Object>> orgs = rest

.get().uri(url)

.attributes(oauth2AuthorizedClient(client))

.retrieve()

.bodyToMono(List.class)

.block();

if (orgs.stream().anyMatch(org -> "spring-projects".equals(org.get("login")))) {

return user;

}

throw new OAuth2AuthenticationException(new OAuth2Error("invalid\_token", "Not in Spring Team", ""));

};

}

Обратите внимание, что этот код зависит от WebClientэкземпляра для доступа к API GitHub от имени аутентифицированного пользователя. Сделав это, он перебирает организации в поисках той, которая соответствует "spring -projects" (это организация, которая используется для хранения проектов с открытым исходным кодом Spring). Вы можете заменить здесь свое собственное значение, если хотите иметь возможность успешно пройти аутентификацию и не состоите в команде Spring Engineering. Если совпадений нет, он выдает сообщение OAuth2AuthenticationException, которое принимает Spring Security и превращает в ответ 401.

Он WebClientтакже должен быть создан как bean-компонент, но это тривиально, потому что все его ингредиенты могут быть автоматически добавлены в силу использования spring-boot-starter-oauth2-client:

@Bean

public WebClient rest(ClientRegistrationRepository clients, OAuth2AuthorizedClientRepository authz) {

ServletOAuth2AuthorizedClientExchangeFilterFunction oauth2 =

new ServletOAuth2AuthorizedClientExchangeFilterFunction(clients, authz);

return WebClient.builder()

.filter(oauth2).build();

}

|  |  |
| --- | --- |
|  | Очевидно, что приведенный выше код можно обобщить для других правил аутентификации, некоторые из которых применимы к GitHub, а некоторые - к другим поставщикам OAuth 2.0. Все, что вам нужно, - это WebClientнекоторые знания API поставщика. |