УДК 004

Защита REST API с помощью Spring Security

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье будут рассмотрены возможности создания запросов REST API и защита их с использованием Spring Security. Работа будет происходить в среде разработки IntelijIdea.
Ключевые слова: Spring Security, Spring, Spring boot

Securing REST APIs with Spring Security

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article will explore the possibilities of making REST API requests and securing them using Spring Security. The work will take place in the IntelijIdea development environment.
Keywords: Spring Security, Spring, Spring boot

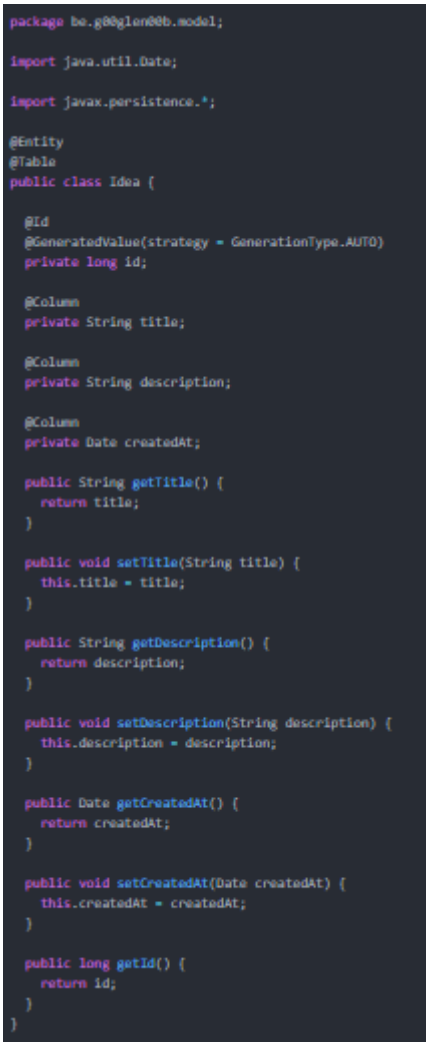
Чтобы начать проект, можно воспользоваться онлайн-сервисом Spring Initializer для настройки проекта. Для проекта будем использовать простую настройку с использованием Spring Data JPA, Spring Web и Spring Security.

Цель статьи – написать Rest API запросы и защитить их с помощью Spring Security.

А.А.Шейн, Д.Г.Залевский, С.В. Автайкин, С.В.Карташев, С.А.Скороход разработали и описали действия программы, предназначенной для автоматического создания набора классов для представления объектов модели Decode в виде нативных объектов языка Java [1]. Н.Н. Глибовец проанализировала в своей работе особенности агентных технологий и перспективы их использования для разработки сложных

многопользовательских программных систем [2]. В своей работе М.К.Ермаков, С.П.Вартанов рассмотрели вопросы проведения анализа программ интерпретируемых языков программирования [3]. С.В. Мельников провел обзор на работу с отладочным интерфейсом Java и методом модификации функциональности приложения, не изменяющий его бинарные файлы [4]. Так же А.А. Птицын, Н.Л. Подколодный, Д.А.Григорович, С.В. Лаврюшев разработали молекулярно-биологический сервер и на его базе создали ряд информационно-вычислительных систем, для изучения регуляции экспрессии генов с использованием новейших технологий Java [5].

Начнем с модели под названием Idea, имеющей такие свойства, как заголовок, описание и дату, когда она была создана (рис.1).



```
package be.g00glen00b.model;

import java.util.Date;

import javax.persistence.*;

@Entity
@Table
public class Idea {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;

  @Column
  private String title;

  @Column
  private String description;

  @Column
  private Date createdAt;

  public String getTitle() {
    return title;
  }

  public void setTitle(String title) {
    this.title = title;
  }

  public String getDescription() {
    return description;
  }

  public void setDescription(String description) {
    this.description = description;
  }

  public Date getCreatedAt() {
    return createdAt;
  }

  public void setCreatedAt(Date createdAt) {
    this.createdAt = createdAt;
  }

  public long getId() {
    return id;
  }
}
```

Рисунок 1 – Создание модели

Здесь в модели нет установщика для id свойства. Идентификатор должен быть неизменным, поэтому его следует устанавливать только при создании, а не при изменении объекта. Для создания объекта с идентификатором можно создать конструктор для инициализации этого поля или использовать шаблон.

Следующим шагом является репозиторий, который довольно прост, создадим интерфейс, который расширяется от JpaRepository (рис.2).

```
@Repository
public interface IdeaRepository extends JpaRepository<Idea, Long> {

}
```

Рисунок 2 – Создание репозитория

И последнее, но не менее важное: необходимо изменить файл application.properties для автоматического создания таблиц в базе данных в памяти на основе сущностей. Это значительно упрощает разработку, поэтому не нужно создавать новую схему вручную (рис.3).

```
spring.jpa.generate-ddl=true
```

Рисунок 3 – Изменение свойств

Далее пропустим часть создания службы и использования правильных DTO здесь и поместим всю логику в контроллер REST, но правильно было бы расширить это и использовать службы, которые предоставляют некоторые DTO вместо сущностей (рис.4).

```java
@RestController
@RequestMapping("/api/ideas")
public class IdeaController {
  @Autowired
  private IdeaRepository repository;

  @RequestMapping(method = RequestMethod.GET)
  public List<Idea> findAll() {
    return repository.findAll();
  }

  @RequestMapping(method = RequestMethod.POST)
  public Idea add(@RequestBody Idea idea) {
    Idea model = new Idea();
    model.setCreatedAt(new Date());
    model.setTitle(idea.getTitle());
    model.setDescription(idea.getDescription());
    return repository.saveAndFlush(model);
  }

  @RequestMapping(value = "/{id}", method = RequestMethod.GET)
  public Idea findOne(@PathVariable long id) {
    return repository.findOne(id);
  }

  @RequestMapping(value = "/{id}", method = RequestMethod.PUT)
  public Idea update(@PathVariable long id, @RequestBody Idea idea) {
    Idea model = repository.findOne(id);
    if (model != null) {
      model.setTitle(idea.getTitle());
      model.setDescription(idea.getDescription());
      return repository.saveAndFlush(model);
    }
    return null;
  }

  @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
  public void delete(@PathVariable long id) {
    repository.delete(id);
  }
}
```

Рисунок 4 – Заполненый контроллер

Здесь не так много обработок ошибок, и на самом деле не рекомендуется помещать все эти данные для входа в контроллер.

С Spring Web MVC довольно легко настроить REST API с помощью @RestController аннотации, которая является сокращением для обычного контроллера «@Controller», но где все возвращенные объекты автоматически отображаются как тело ответа, поэтому можно оставить @ResponseBody аннотацию.

Есть несколько методов для получения всех записей / одной записи, для добавления новых записей или для обновления / удаления существующих записей.

Собственно, это весь код, который нужен для защиты приложения. Если запустить само приложение и перейдете по адресу «http://localhost:8080/api/ideas», будет предложено ввести имя пользователя / пароль. Самое интересное в Spring Boot заключается в том, что для пользователя настроено множество вещей, например, с «spring-security-starter-security», проект уже настроен с некоторыми мерами безопасности.

Взглянув на справочное руководство, можно увидеть, что уже существует пользователь по умолчанию, созданный с именем пользователя «user» и случайным паролем. Чтобы увидеть пароль, нам нужно изменить некоторые уровни ведения журнала, поэтому сделаем это, отредактировав «application.properties» и добавив следующую строку (рис.5).

```
logging.level.org.springframework.boot.autoconfigure.security=INFO
```

Рисунок 5 – Изменение свойств

Если правильно войти в систему и добавить новую запись с помощью запроса POST, то увидим, что она работает нормально (рис.6).

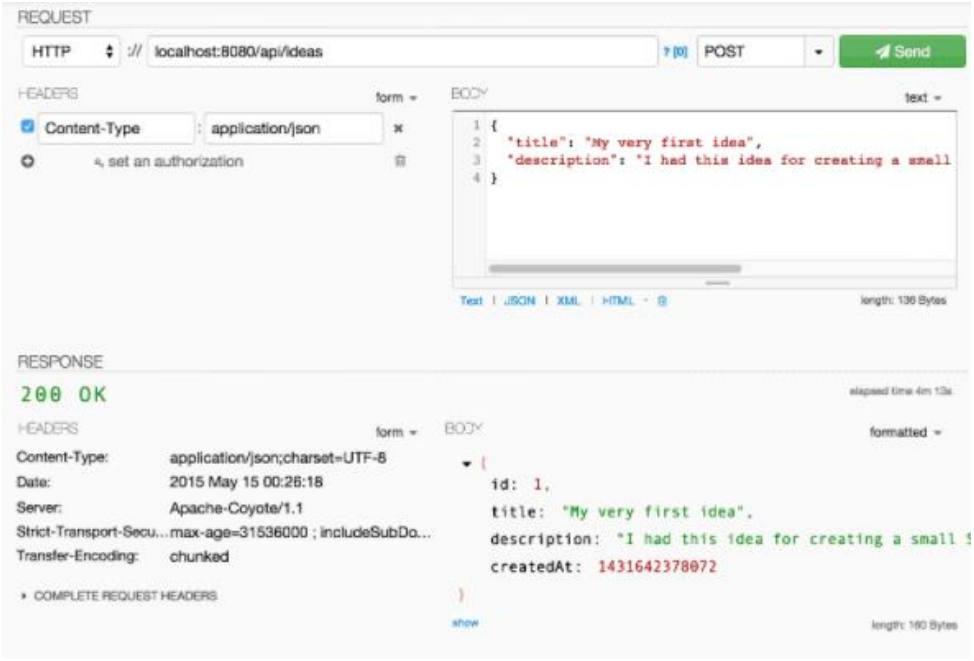


Рисунок 6 – Ответ запроса

Настройки по умолчанию работают. Это, безусловно, лучше, чем возиться со Spring Security, пока, наконец, не получите работающий базовый пример, теперь есть базовый пример, и нужно немного его настроить.

Прежде всего, необходимо установить пароль по умолчанию на пароль вместо UUID, который генерируется каждый раз (рис.7).

```
security.user.name=user
security.user.password=password
```

Рисунок 7 – Изменение свойств

Можно перезапустить приложение сейчас и попробовать его, теперь можно войти в систему, используя только комбинацию пользователь / пароль.

Следующая часть немного сложнее. Вместо защиты всех действий попробуем сохранить запросы GET для всех, в то время как добавление, обновление или удаление записей должно быть разрешено только для пользователя, который вошел в систему.

Для этого нужно написать новый класс, который расширяется от WebSecurityConfigurerAdapter. Затем можно переопределить

«configure(HttpSecurityhttp)» метод, позволяя работать со сборщиком для настройки безопасности для каждого пути (рис.8).

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
      .csrf().disable()
      .authorizeRequests()
        .antMatchers(HttpMethod.POST, "/api/**").authenticated()
        .antMatchers(HttpMethod.PUT, "/api/**").authenticated()
        .antMatchers(HttpMethod.DELETE, "/api/**").authenticated()
        .anyRequest().permitAll()
        .and()
      .httpBasic().and()
      .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
  }
}
```

Рисунок 8 – Создание класса

Настраиваем все запросы POST/PUT/DELETE для всего, что находится ниже «/api/**», чтобы они были разрешены только для людей, которые прошли проверку подлинности. Все остальные запросы разрешены всем.

Запускаем приложение и попробуем отправить POST-запрос(рис.9).

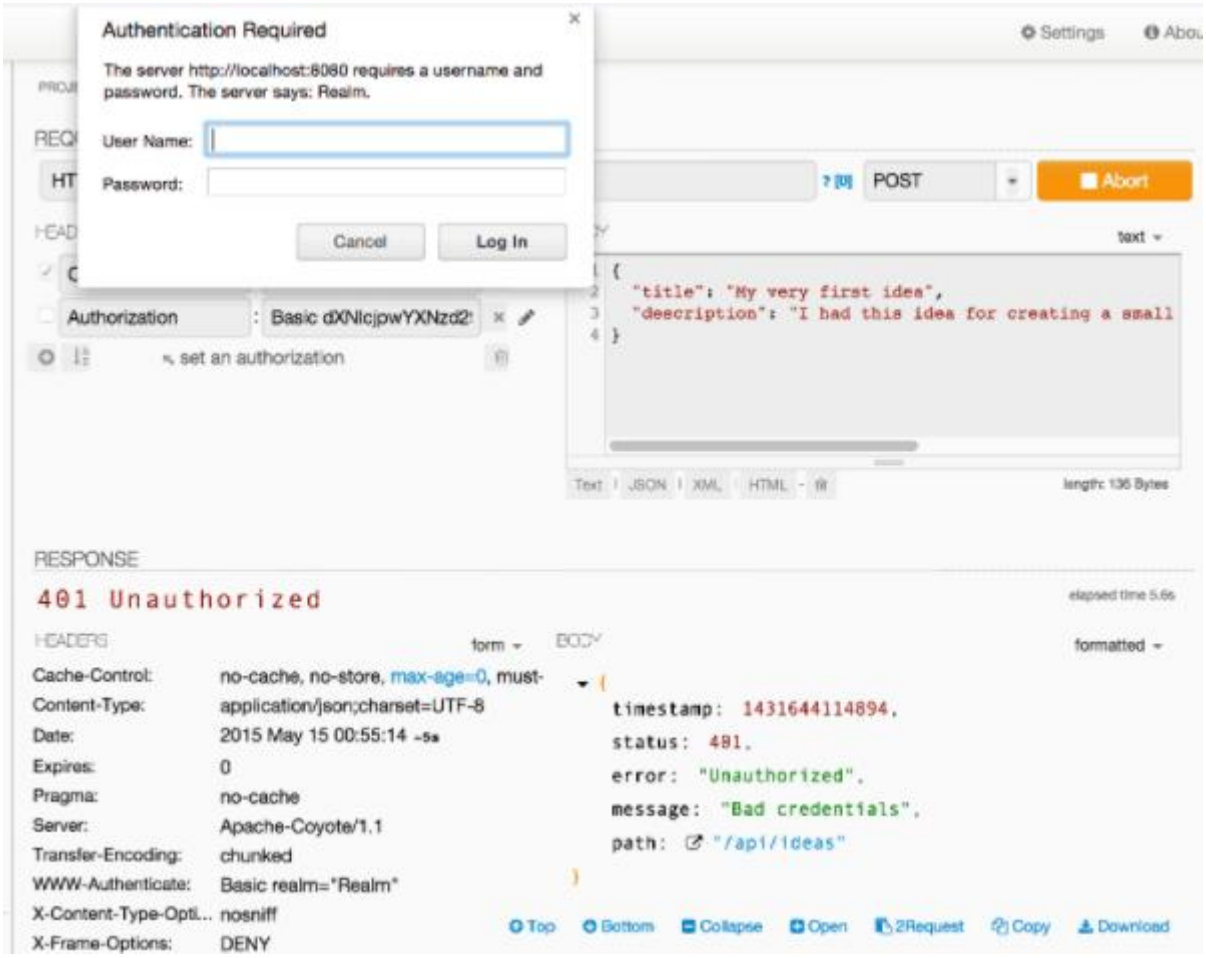


Рисунок 9 – Отправка запроса

Это должно дать еще одно окно авторизации, потому что добавление разрешено только для аутентифицированных пользователей.

В данной статье был рассмотрен пример создания REST API запросов и защита их с использованием Spring Security.

Библиографический список

1. Шейн А.А., Залевский Д.Г., Автайкин С.В., Карташев С.В., Скороход С.А. Генератор исходного кода на языке java по описанию бортовых компонентов decode (decode java generator 0.2) // Вестник волжского университета им. в.н. татищева. 2019. №3. С. 26-32.
2. Глибовец Н.Н. Использование jade (java agent development environment) для разработки компьютерных систем поддержки дистанционного обучения агентного типа // Заметки по информатике и математике. 2019. №10. С. 15-20.
3. Ермаков М.К., Вартанов С.П. Подход к проведению динамического анализа java-программ методом модификации виртуальной машины java // Научные труды Винницкого национального технологического университета. 2018. №6. С. 10-17.
4. Мельников С.В. Обзор и применение отладочного интерфейса java (jdi)

для обратимой модификации программных продуктов // Современные проблемы науки и образования. 2018. №8. С. 8-19.

5. Птицын А.А., Подколодный Н.Л., Григорович Д.А., Лаврюшев С.В. Создание молекулярно-биологического сервера www с использованием новейших технологий java // Заметки по информатике и математике. 2020. №1. С. 11-20.