GeekBrains

Разработка интернет-магазина на Spring Framework

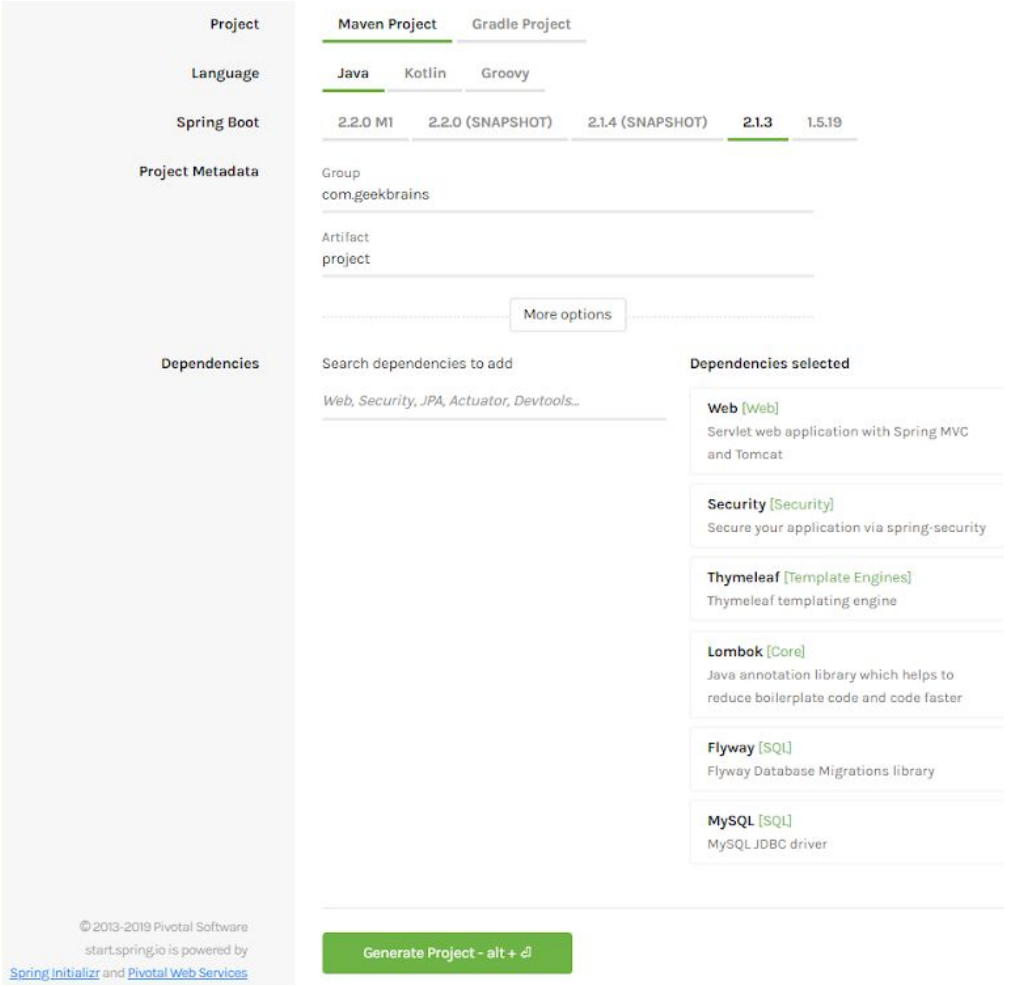
Собираем базовое приложение

Подключаем Spring Boot, Security Data, Thymeleaf.

Оглавление

Построение базового проекта

Заходим на сайт Spring Initializr и формируем проект Spring Boot. Подключаем следующие модули: Web, Security, Thymeleaf, Lombok, Flyway, MySQL.



Когда проект сгенерирован и открыт в студии, добавляем в **pom.xml** еще несколько зависимостей и настройку **flyway**. Чтобы с ней не возникло трудностей, ниже приведен весь **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
http://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>2.1.3.RELEASE</version>
      <relativePath/>
   </parent>
```

```
    <groupId>com.geekbrains</groupId>
    <artifactId>spring-boot-project</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <name>project</name>
    <description>Demo project for Spring Boot</description>

    <properties>
       <java.version>1.8</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-thymeleaf</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-web</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-security</artifactId>
       </dependency>
       <dependency>
          <groupId>org.thymeleaf.extras</groupId>
          <artifactId>thymeleaf-extras-springsecurity5</artifactId>
          <version>3.0.3.RELEASE</version>
       </dependency>
       <dependency>
          <groupId>mysql</groupId>
          <artifactId>mysql-connector-java</artifactId>
          <version>8.0.13</version>
       </dependency>
       <dependency>
          <groupId>org.flywaydb</groupId>
          <artifactId>flyway-core</artifactId>
          <version>5.2.1</version>
       </dependency>
       <dependency>
          <groupId>org.projectlombok</groupId>
          <artifactId>lombok</artifactId>
          <version>1.18.4</version>
          <scope>provided</scope>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-test</artifactId>
          <scope>test</scope>
```

```
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.flywaydb</groupId>
            <artifactId>flyway-maven-plugin</artifactId>
            <version>5.2.1</version>
            <configuration>
<url>jdbc:mysql://localhost:3306/geek_db?useSSL=false&amp;useUnicode=true&amp;cha
racterEncoding=UTF-8&amp;serverTimezone=UTC</url>
               <user>geek</user>
               <password>geek</password>
               <schemas>
                  <schema>geek_db</schema>
               </schemas>
            </configuration>
         </plugin>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
      </plugins>
   </build>
</project>
```

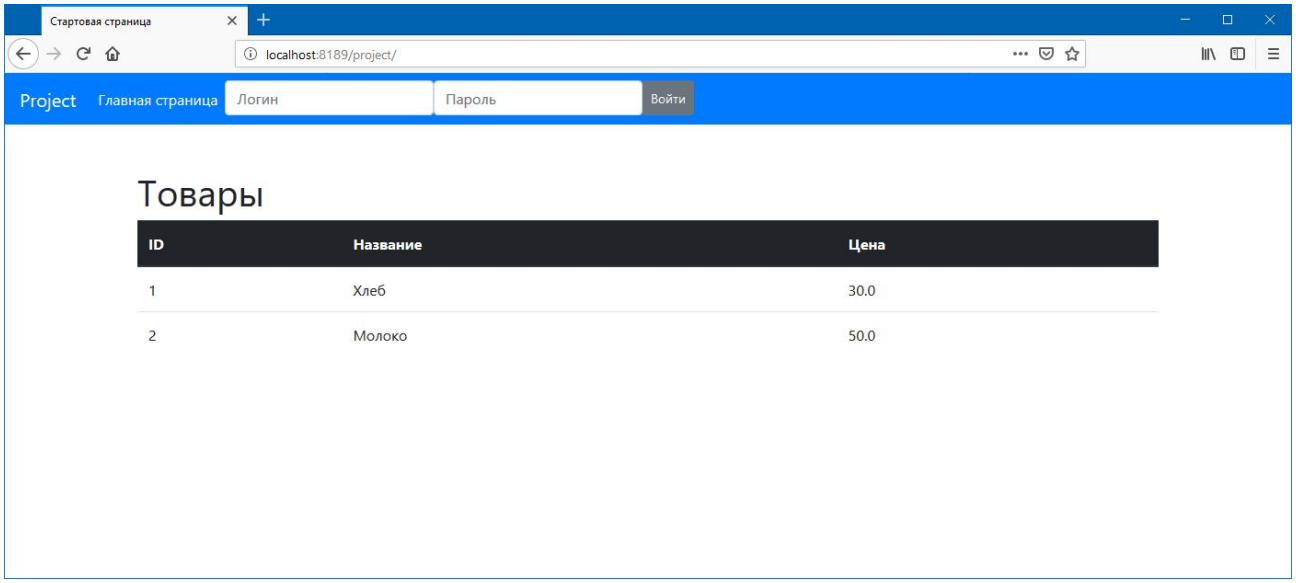
Для чего нужны все эти модули:

- **Web** — основной модуль для разработки веб-приложения;
- **Thymeleaf** — шаблонизатор;
- **Security** — безопасность приложения;
- **Lombok** — автогенерация геттеров, сеттеров и конструкторов;
- **Thymeleaf-extras-springsecurity5** — дополнительный набор тегов для Thymeleaf для интеграции с модулем Spring Security;
- **MySQL Connector** — драйвер для работы с MySQL;
- **Flyway Core** — система миграции БД.

Посмотрим, как сделать базовый проект для будущего магазина. Для этого понадобится:

- база данных,
- подключение к БД,
- возможность выполнять аутентификацию и регистрацию пользователей,
- выводить список товаров на страницу нашего веб-приложения.

Если сделать просто набросок, то получим:



Файл настроек приложения **application.properties**:

```
server.port=8189
server.servlet.context-path=/project

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=none

spring.thymeleaf.encoding=UTF-8

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/geek_db?allowPublicKeyRetrieval=true&useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.username=geek
spring.datasource.password=geek

spring.jackson.serialization.indent_output=true
```

Чтобы связать приложение с БД, мы уже подключили **MySQL Connector**, **Spring Data** и **Flyway**.

Для включения безопасности в приложении создаем бин **SecurityWebApplicationInitializer**:

```
@Component
public class SecurityWebApplicationInitializer extends
AbstractSecurityWebApplicationInitializer {
}
```

Сами настройки безопасности могут выглядеть так:

```
package com.geekbrains.springbootproject.config;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    private UserService userService;

    @Autowired
    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
{
        auth.authenticationProvider(authenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .anyRequest().permitAll()
                .antMatchers("/admin/**").hasRole("ADMIN")
                .and()
                .formLogin()
                .loginPage("/login")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
                .and()
                .logout()
                .logoutSuccessUrl("/")
                .permitAll();
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
        auth.setUserDetailsService(userService);
        auth.setPasswordEncoder(passwordEncoder());
        return auth;
    }
}
```

В качестве защиты пароля мы используем алгоритм **BCrypt**. Пускаем всех пользователей на все страницы будущего магазина, кроме админки.

Для аутентификации пользователей создаем интерфейс **UserService** с наследованием от интерфейса **UserServiceDetail**.

```
public interface UserService extends UserDetailsService {
    User findByUserName(String username);
    boolean save(SystemUser systemUser);
}
```

И прописываем его реализацию:

```
@Service
public class UserServiceImpl implements UserService {
    private UserRepository userRepository;
    private RoleRepository roleRepository;
    private BCryptPasswordEncoder passwordEncoder;

    @Autowired
    public void setUserRepository(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Autowired
    public void setRoleRepository(RoleRepository roleRepository) {
        this.roleRepository = roleRepository;
    }

    @Autowired
    public void setPasswordEncoder(BCryptPasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    @Transactional
    public User findByUserName(String username) {
        return userRepository.findOneByUserName(username);
    }

    @Override
    @Transactional
    public boolean save(SystemUser systemUser) {
        User user = new User();
        if (findByUserName(systemUser.getUserName()) != null) {
            return false;
        }
        user.setUserName(systemUser.getUserName());
        user.setPassword(passwordEncoder.encode(systemUser.getPassword()));
        user.setFirstName(systemUser.getFirstName());
```

```
        user.setLastName(systemUser.getLastName());
        user.setEmail(systemUser.getEmail());

user.setRoles(Arrays.asList(roleRepository.findOneByName("ROLE_CLIENT")));
        userRepository.save(user);
        return true;
    }

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String userName) throws
UsernameNotFoundException {
        User user = userRepository.findOneByUserName(userName);
        if (user == null) {
            throw new UsernameNotFoundException("Invalid username or password");
        }
        return new
org.springframework.security.core.userdetails.User(user.getUserName(),
user.getPassword(),
                mapRolesToAuthorities(user.getRoles()));
    }

    private Collection<? extends GrantedAuthority>
mapRolesToAuthorities(Collection<Role> roles) {
        return roles.stream().map(role -> new
SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());
    }
}
```

Класс **SystemUser** используется для заполнения формы регистрации на сайте, после чего преобразуется к **User** и сохраняется в БД.

```
@Data
@NoArgsConstructor
public class SystemUser {
    private String userName;
    private String password;
    private String matchingPassword;
    private String firstName;
    private String lastName;
    private String email;
}
```

А вот класс **User**:

```
@Entity
@Data
@Table(name = "users")
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id")
   private Long id;

   @Column(name = "username")
   private String userName;

   @Column(name = "password")
   private String password;

   @Column(name = "first_name")
   private String firstName;

   @Column(name = "last_name")
   private String lastName;

   @Column(name = "email")
   private String email;

   @ManyToMany(fetch = FetchType.LAZY)
   @JoinTable(name = "users_roles",
   joinColumns = @JoinColumn(name = "user_id"),
   inverseJoinColumns = @JoinColumn(name = "role_id"))
   private Collection<Role> roles;

   public User() {
   }

   public User(String userName, String password, String firstName, String
lastName, String email) {
      this.userName = userName;
      this.password = password;
      this.firstName = firstName;
      this.lastName = lastName;
      this.email = email;
   }

   public User(String userName, String password, String firstName, String
lastName, String email,
         Collection<Role> roles) {
      this.userName = userName;
      this.password = password;
      this.firstName = firstName;
      this.lastName = lastName;
      this.email = email;
```

```
        this.roles = roles;
    }

    @Override
    public String toString() {
        return "User{" + "id=" + id + ", userName='" + userName + '\'' + ",
password='" + "*********" + '\''
                + ", firstName='" + firstName + '\'' + ", lastName='" + lastName +
'\'' + ", email='" + email + '\''
                + ", roles=" + roles + '}';
    }
}
```

Практическое задание

1. Продумать структуру базы данных для интернет-магазина. В качестве домашнего задания сдаете SQL скрипт, и краткое обоснование выбранных вами сущностей (почему они выглядят именно так).

Дополнительные материалы

1. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

Используемая литература

Для подготовки данного методического пособия литература не использовалась.