***Spring Security*** – модуль *Spring*, использующийся для обеспечения безопасности приложения.

Для его подключения нужно добавить в проект стартер ***spring-boot-starter-security***. Сразу после этого, при доступе к любой странице будет требоваться аутентификация на простой странице входа. По умолчанию имя пользователя – *user*, а пароль будет сгенерирован и выведен в консоль приложения. Никаких особых ролей или полномочий не предусматривается.

Для ручной настройки безопасности необходимо создать класс и добавить аннотацию ***@Configuration***.

Для включения шифрования передаваемого пароля нужно создать бин, возвращающий одну из реализаций ***PasswordEncoder***:

* *BCryptPasswordEncoder* – использовать шифрование *bcrypt*.
* *NoOpPasswordEncoder* – не применять шифрование.
* *Pbkdf2PasswordEncoder* – использовать шифрование *PBKDF2*.
* *SCryptPasswordEncoder* – использовать шифрование *Scrypt*.
* *StandardPasswordEncoder* – использовать шифрование *SHA-256*.

В результате, пароли будут храниться в БД в зашифрованном виде.

*@Configuration*

*public class SecurityConfig {*

*@Bean // включение шифрования паролей*

*public PasswordEncoder passwordEncoder() {*

*return new BCryptPasswordEncoder(); } }*

Для настройки хранилища учетных записей пользователей для их аутентификации нужно в конфигурации создать бин, возвращающий одну из реализаций ***UserDetailsService***:

* хранилище УЗ в памяти
* хранилище УЗ *JDBC*

Также можно использовать сторонние сервисы *LDAP* или *OAuth2*.

Создадим реализацию хранения УЗ в памяти и инициализируем её с двумя пользователями.

*import org.springframework.security.core.userdetails.User; // специальный класс User  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;*

*@Configuration*

*public class SecurityConfig {*

*@Bean // реализация хранения УЗ в памяти*

*public UserDetailsService userDetailsService(PasswordEncoder encoder) {*

*List<UserDetails> usersList = new ArrayList<>();*

*usersList.add( new User("buzz", encoder.encode("password"),*

*Arrays.asList(new SimpleGrantedAuthority("ROLE\_USER"))));*

*usersList.add( new User("woody", encoder.encode("password"),*

*Arrays.asList(new SimpleGrantedAuthority("ROLE\_USER"))));*

*return new InMemoryUserDetailsManager(usersList); } }*

Для создания реализации хранения УЗ в памяти, создадим свой класс *User*, имплементирующий *UserDetails*, и репозиторий для него.

*@Entity*

*@Data*

*@NoArgsConstructor (access=AccessLevel.PRIVATE, force=true)*

*@RequiredArgsConstructor*

*public class User implements UserDetails {*

*private static final long serialVersionUID = 1L;*

*@Id*

*@GeneratedValue(strategy=GenerationType.AUTO)*

*private Long id;*

*private final String username;*

*private final String password;*

*private final String phoneNumber;*

*@Override // наделим правами с ролью USER*

*public Collection<? Extends GrantedAuthority> getAuthorities() {*

*return Arrays.asList(new SimpleGrantedAuthority(“ROLE\_USER”)); }*

*@Override // реализация-заглушка*

*public Boolean isAccountNonExpired() {*

*return true; }*

*@Override // реализация-заглушка*

*public Boolean isAccountNonLocked() {*

*return true; }*

*@Override // реализация-заглушка*

*public Boolean isCredentialsNonExpired() {*

*return true; }*

*@Override // реализация-заглушка*

*public Boolean isEnabled() {*

*return true; } }*

*public interface UserRepository extends CrudRepository<User, Long> {*

*User findByUsername(String username); }*

Стр.154