2 – Автентикација

Избработив веб апликација во Јава со Spring Boot во која имам имплементирано 2-факторска автентикација при логирање на корисници, без користење на Spring Security.

Во моделот имам 2 ентитети кои : **User** и **ConfirmationToken**. Овие ентитети се чуваат и во **PostgreSQL** база на податоци.

Класата **User** има 4 атрибути:

**String username** – која воедно е и примарен клуч во базата на податоци  
**String email** – важен атрибут корисен за подоцнежното 2-факторско логирање  
**String password**  
**boolean isConfirmed** – променлива која ни кажува дали корисникот при регистрација го верификувал својот емаил.

Објектите од класата **ConfirmationToken** претставува токен кој се генерира за секој обид за регистрација на нов корисник. Има 6 атрибути:

**Long id** – примарен клуч за базата на податоци  
**String token** – вредноста на токенот  
**LocalDateTime createdAt** – време на креирање на токенот  
**LocalDateTime expiresAt** – време на истекување на токенот  
**LocalDateTime confirmedAt** – времето кога е потврден емаилот на корисникот  
**User user** – релација кон корисникот за кого е генериран токенот

За пристап до базата на податоци се користат 2 Repository интерфејси **UserRepository** и **ConfirmationTokenRepository** соодветно. Наследуваат од интерфејсот **JpaRepository<Object, ID>.**

Вo **UserRepository** се користат методите:

* ***Optional<User> findByUsername(String username);***
* ***Optional<User> findByEmail(String email);***

Во ConfirmationTokenRepository се користат методите:

* ***Optional<ConfirmationToken> findByToken(String token);***
* ***List<ConfirmationToken> findAllByUser(User user);***
* ***void updateConfirmedAt(String token, LocalDateTime now);***

Во сервисната логика освен **UserService** и **ConfirmationTokenService** имам и дополнителни 3 интерфејси:

**EmailService** – сервис користен за праќање на токенот за потврда на емајлот при регистрација со помош на методот ***void sendConfirmationCode(String to, String email)***и праќање на кодот за 2-факторска автентикација со методот ***void send2FACode(String email, String code)****.*

**SessionService** – со овој сервис се менаџираат сесиите. Наместо во база се чуваат сесиите во конкурентната мапа **ConcurrentHashMap<String, String> sessions**. Тука се креирани 3 методи:

* ***void storeSession(String sessionToken, String username)*** – зачувување на сесијата во мапата
* ***void removeSession(String sessionToken)*** *–* бришење на сесијата од мапата
* ***boolean isSessionValid(String sessionToken)*** – проверка за валидност на сесијата

**AuthService** – сервис за автентикација на корисникот, односно тука имаме еден метод:

* ***String generate2FACode()*** *–* кој се користи за генерирање на код кој се користи за автентикација при логирање. Се користи објект од класата **SecureRandom** за генерирњае на 6 цифрен број.

Веб слојот е составен од 4 контролери и 1 филтер.

**LoginController** – менаџира со логин активностите на корисниците. Има 3 методи:

* ***String verifyLogin(@RequestParam String username, @RequestParam String password, Model model, HttpServletRequest request)***– ги верифицира корисничкото име и лозинката што корисникот ја внесува при логирање на апликацијата. Тоа се прави со помош на **UserService**.При успешно верифицирање барањето го редиректираме на методот за автентицирање.
* ***String getAuthenticatePage(HttpServletRequest request)*** – Овој метод ја прикажува html страната на која треба да се внесе кодот за автентикација, а во исто време и испраќа мејл со **EmailService** со токенот.
* ***String authenticateUser(@RequestParam String code, HttpServletRequest request, HttpServletResponse response, Model model)*** – Со овој метод се обработува POST барањето откако ќе се внесе некој код. При успешна автентикација се креира колаче во кое се чува сесијата на корисникот и го редиректира корисникот кон главниот контролер.

**LogoutController** – има 1 метод:

* ***String logout(@CookieValue(name = "SESSIONID", required = false) String sessionToken, HttpServletResponse response)*** – ја инвалидира сесијата, односно, го брише токенот од мапата и редиректира на логин страната.

**RegisterController** – има 5 методи:

* ***String getRegisterPage(@RequestParam(required = false) String error, Model model)***– ја прикажува страната за регистрација.
* ***String validate(@RequestParam String username, @RequestParam String email, @RequestParam String password, @RequestParam String repeatedPassword)*** *–* ги валидира податоците кои се внесени во формата за регистрација. При успешна регистрација барањето се форвардира кон следниот метод подоле, а при неуспешно се покажува соодветната порака преку соодветни исклучо
* ***String sendConfirmationEmail(@RequestParam String username, @RequestParam String email, @RequestParam String password, HttpSession session)*** – метод со кој се праќа емајл на корисникот за потврда на валидност на неговиот емајл. Тука се користи и PasswordEncoder.*hashPassword*(password) за хеширање на лозинката пред да се зачува корисникот во базата на податоци. По праќањето на мејл барањето се редиректира кон следниот метод.
* ***String getVerificationPage(@RequestParam(required = false) String error, Model model)*** – Овој метод ја прикажува html страната за внесување на регистрацискиот токен.
* ***String verifyCode(@RequestParam String code, HttpSession session)*** – со овој метод се проверува дали за внесениот код постои таков токен во базата на податоци и дали припаѓа на истиот корисник кој се обидува да се регистрира (имаат исто корисничко име). При успешно регистрирање барањето се редиректира на логин со цел корисникот да може да се најави, при неуспешна потврда на токенот се враќа истата страна со исхендлана грешка.

**MainController** – има 1 метод:

* ***String getHomePage(HttpServletRequest request, Model model)*** *–* прикажува home page заедно со корисничкото име на корисникот кој е во сесија.

**SessionFilter** – тука се имплементирани методите од интерфејсот **Filter**:

* **void init(FilterConfig filterConfig)**
* **void doFilter (ServletRequest request, ServletResponse response, FilterChain filterChain)** – во овој метод се проверува доколку корисникот не е на некоја од слободните страни како регистрација или логин, дали го има новото колаче во кое се чува сесијата за корисникот доколку го има го пушта да влезе на останатите страни од апликацијата, доколку го нема го пренасочува на логин страната.
* **void destroy().**

За хеширање на лозинките ја користам библиотеката: **org.mindrot.jbcrypt** од која ја користам класата **BCrypt** во која со помош на статичкиот метод *hashpw(plainPassword, BCrypt.gensalt())* ја хеширам лозинката. За проверка за валидност на лозинката при логин се користи статичкиот метод *checkpw(plainPassword, hashedPassword).*