



UINF/PAZ1c
epizóda 9

Heslá
Prihlasovanie
HTTP Basic Auth

Používatelia poriadne: bezpečné heslá

Bezpečnosť hesiel:

1. Bezpečné heslá
2. Bezpečné uloženie hesiel
3. Neposielať heslá po sieti nešifrovane
4. Nelogovať heslá
(ani nič iné z entít okrem ID-čiek)

This is getting ridiculous...

Enter a new password:

The password must:

- have at least 8 characters
- have no more than 8 characters
- have both upper and lower case characters
- have no more than 0 non-alphanumeric characters
- have at least 3 letters
- have at least 2 digits
- not contain any 3 consecutive characters of your user ID or user name
- not contain more than 4 numerical characters
- have no more than 2 pairs of repeating characters
- not be an old password
- allow old passwords after 1200 days
- maximum password age (required to change every 90 days)

New password:

Confirm:

[routing.cz](#)

Change passwords

Rýchlosť cracknutia – brute force

Kvalita hesla	10 miliónov/s bežný procesor	7 miliárd/s grafické karty, botnety
8 alfanumerických znakov	252 dní	22 sekúnd
8 ľubovoľných znakov	17 rokov	9,25 dní
11 ľubovoľných znakov	14 miliónov rokov	20 rokov
13 alfanumerických znakov	232 miliárd rokov	906 rokov
12 ľubovoľných znakov	484 miliárd rokov	1,8 miliónov rokov

Na overenie kvality
hesla existujú
knižnice, napr.
Zxcvbn4j
nbvcxz



Sila hesla

Treba donútiť užívateľov používať silné heslá

Umelé pravidlá

Niečo rozumné

- Aspoň 10 znakov: veľké písmeno, malé písmeno, číslo, špeciálny znak
- Aspoň 15 znakov: veľké písmeno, malé písmeno, číslo
- Aspoň 20 znakov: veľké písmeno, malé písmeno
- Ak dĺžka viac ako 25 znakov, tak možno kontrolujte či to nie je opakujúca sa sekvencia typu "aaaaaaaaa..."



Sila hesla

Treba donútiť užívateľov používať silné heslá

- Špecializované knižnice na kontrolu kvality
 - <https://github.com/GoSimpleLLC/nbvcxz>
 - **Pozor**, nepoužívajte defaultné nastavenia
 - **Pozor**, predpokladajú angličtinu



Hešovanie

Hash je funkcia $y = h(x)$, ktorá pre každý vstup x vráti y fixnej dĺžky

- X je typicky hocičo, napr. pole bajtov ľubovoľnej dĺžky
- Y je buď int, alebo string/pole bajtov fixnej dĺžky
 - MD5(x) ... 4 bajty
 - SHA-1(x) ... 20 bajtov
 - SHA-256(x) ... 32 bajtov

Hešovanie

Vlastnosti $h(x)$

- Ak $x1 == x2$, tak $h(x1) == h(x2)$
- Ak $x1 != x2$, tak **typicky** $h(x1) != h(x2)$
- **Kolízia** je, ak $x1 != x2$ a $h(x1) == h(x2)$
- Výpočet $y = h(x)$ chceme rýchly
- Inverzný výpočet $x = h'(y)$ **chceme brutálne pomalý**
a samozrejme s takmer 100% pravdepodobnosťou dostaneme iba kolíziu

Čím je menšia šanca kolízie a čím je inverzný výpočet pomalší, tým je hešovacia funkcia "kvalitnejšia"

V jave má každý objekt metódu `hashCode`, ktorá je základom pre `HashMap`.

`hashCode` nie je "kvalitná" funkcia, no je veľmi rýchla.

Hešovanie



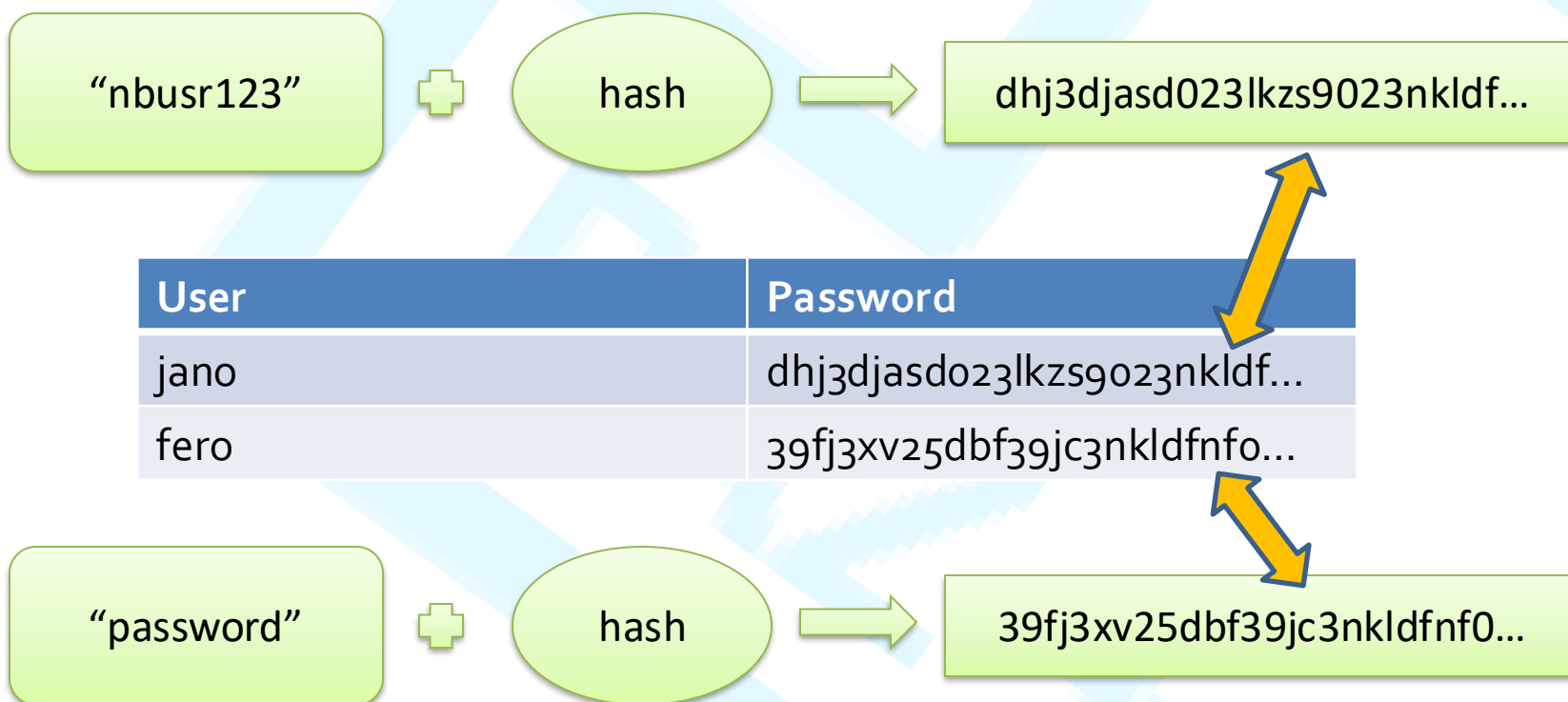
Čím je menšia šanca kolízie a čím je "inverzný" výpočet pomalší, tým je hešovacia funkcia "kvalitnejšia"

V Jave má každý objekt metódu hashCode, ktorá je základom pre HashMap.

- hashCode nie je "kvalitná" funkcia, no je veľmi rýchla.

Bezpečné uloženie

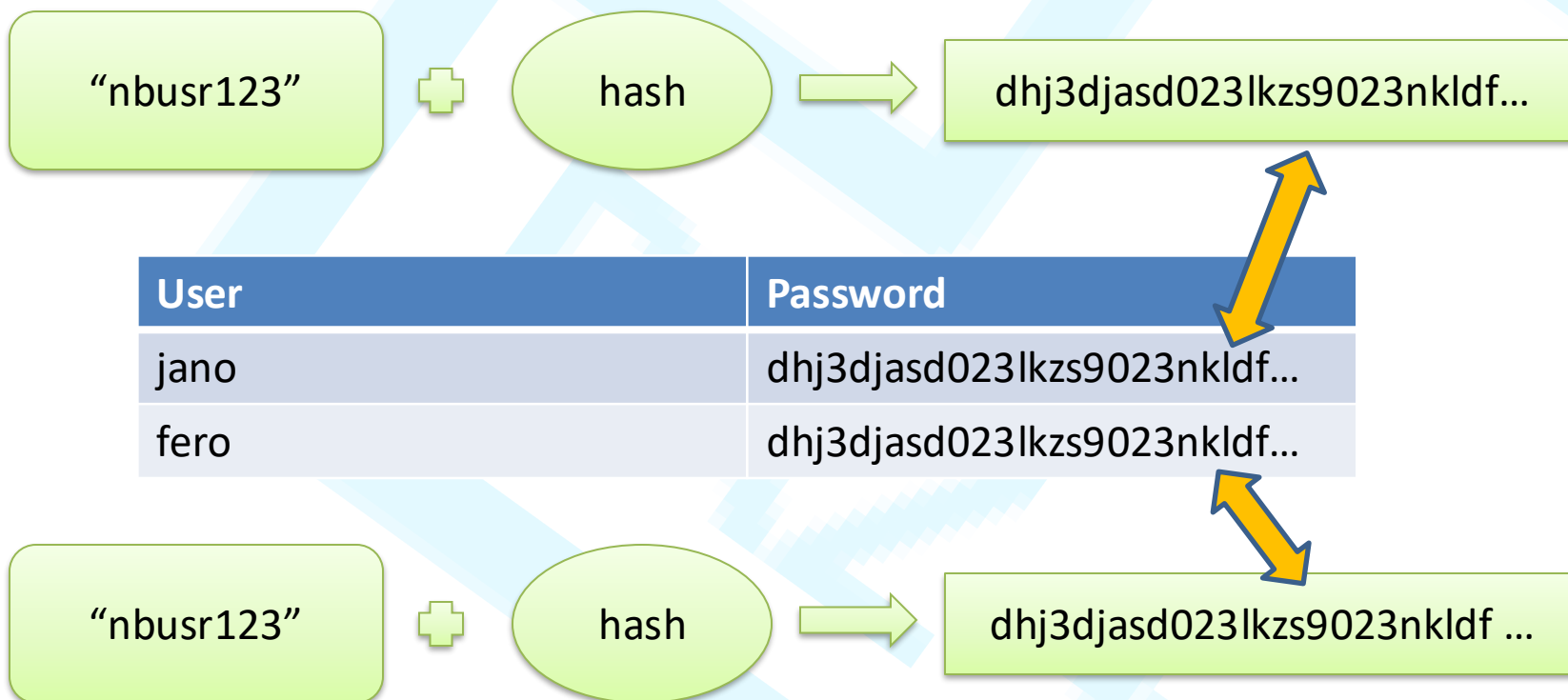
- **NIKDY!** neukladáme heslá v pôvodnom tvare
- Vždy iba kryptografický hash (SHA-256, SHA-384, ...)
 - Špec. hashe na heslá: **bcrypt**, scrypt, PBKDF2, **Argon2**



Bezpečné uloženie

ALE!

- Rovnaké heslá = rovnaký hash
- Rainbow tabuľky – mapa z hash-ov na heslá
 - využitie napr. : odpočúvanie signálu GSM



Bezpečné uloženie

- heslo rozšírime o náhodné znaky = soľ , ktoré si zapamätáme
- posolením prinútime útočníka rátať hash-e



User	salt	Password
jano	gkt478eru9sdfjdfu9orsdf39	894df0234kl sdf90sdlkfj9...
fero	sdfjklasdfjl45636r4ufjsd90	psdfj9034nmjs0v2340sd...



Generovanie soli

- Nechceme problémy pri práci s DB
 - žiadne divoké znaky

1 znak =
 2^5 bitov

```
new BigInteger(130, new SecureRandom()).toString(32)
```

l4oampdfglgmqmmss8qrrd146v

26 znakov = $130 / 5$

```
UUID.randomUUID().toString();
```

67bd4666-8e2b-4b3e-a986-6ac53c3d1ee3

32 hexa znakov

Hashujeme (java.security.MessageDigest)

```
/*pripravíme si reťazec na hasovanie*/
```

```
String stringForHash = password + salt;
```

```
/*vezmeme si hashovací algoritmus SHA-256*/
```

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
```

```
/*vložíme bajtovú reprezentáciu reťazca do SHA-256*/
```

```
md.update(stringForHash.getBytes());
```

```
/*vyberieme výsledný hash ako pole bajtov (vždy dĺžky 32)*/
```

```
byte[] data = md.digest();
```

```
[-115,24,-44,-67,240,12,97,-8,211,...]
```

Hash na String (Hex je zo spring-security-core)

- new String(hash)

#ÉR¿âEÚ°èóì|'šæÃX"8?ZòŽpOE\$"ž

- divoké binárne dáta, zbytočné komplikácie s JDBCTemplate
- každý bajt na dva hexaznaky
 - dokopy 64 znakov

```
String hashString = new BigInteger(1,hash).toString(16);
```

```
String hashString = new String(Hex.encode(hash));;
```

```
AD23C952BFE245DABAE8F3EC7C919A1017E6C358A838177F5AF28EDE8C24229E
```

Pomalé hashovanie – pomaly ďalej zájdeš

Tento šibal vytiahne 0-100 km/h...
za jedno vypitie kávy,
dole kopcom



Pomalé hashovanie – pomaly ďalej zájdeš

- Štandardné hashovacie funkcie (rodina SHA)
 - Navrhnuté na čo najvyššiu rýchlosť a malé pamäťové nároky
 - Veľmi efektívne implementácie pre GPU, FPGA, ASIC
 - Útočník s prvotriednym HW sa môže pokúsiť o backtrack pri slabých heslách

Pomalé hashovanie – pomaly ďalej zájdeš

- Pomalé hashovacie funkcie (Bcrypt, Argon2)
 - Užívateľom nevadí ak kontrola hesla bude trvať trebárs 500ms
 - Efektívne znemožnia backtrack
 - Ťažko akcelerovateľné na GPU/FPGA/ASIC (najmä Argon2)



Pomalé hashovanie cez BCrypt

(org.springframework.security.crypto.bcrypt)

- Importujeme cez maven knižnicu spring-security-core

```
/*vytvoríme sol'*/
```

```
String salt = BCrypt.gensalt();
```

```
/*vytvoríme reťazec alg.sol'.hash*/
```

```
String pwHash = BCrypt.hashpw(heslo, salt);
```

\$2a\$10\$V1bQOnfxjf76Rmu7yTrKiu

alg: 7 znakov
sol': 22 znakov

\$2a\$10\$V1bQOnfxjf76Rmu7yTrKiuVX9Wj4L/IKadTNPZtbtbkgytpQo5tFO

```
/*overíme heslo*/
```

```
boolean ok = BCrypt.checkpw(heslo, pwHash);
```

alg: 7 znakov
sol': 22 znakov
hash: 31 znakov

Hashovanie v databáze?

SHA-512

```
UPDATE user SET heslo=  
SHA2("heslo67bd4666-8e2b-4b3e-a986-6ac53c3d1ee3",512) WHERE  
id=1;
```

- uložíme hash – výsledok je ok, ale:
- príkazy sa v databáze niekedy logujú
- kto získa prístup k logom, môže vidieť pôvodné heslo

Súhrn

- Každú po sieti dostupnú aplikáciu sa niekto snaží hacknúť
- Ochrana hesiel:

Používajte	To vás ochráni proti
Osolené hashe	rainbow tabuľkám
Dlhé heslá	slovníkovým útokom
Pomalé hashovacie funkcie	backtracku

- BCrypt algoritmus už počíta so všetkým
- Spoiler na predmet Projekt 1
 - Delegujte autentifikáciu na niekoho iného cez OpenID Connect (Keycloak, Azure, Google)

Zabezpečenie aplikácie

- Každú po sieti dostupnú aplikáciu sa niekto snaží hacknúť
- Ochrana hesiel:

Používajte	To vás ochráni proti
Osolené hashe	rainbow tabuľkám
Dlhé heslá	slovníkovým útokom
Pomalé hashovacie funkcie	backtracku

- BCrypt algoritmus už počíta so všetkým
- Spoiler na predmet Projekt 1
 - Delegujte autentifikáciu na niekoho iného cez OpenID Connect (Keycloak, Azure, Google)

Full-stack aplikácia

- Pre našu appku Entrance máme
 - Desktop GUI – JavaFX
 - REST API – Spring Boot
- Príklad: Lucka chce zobrazit' všetky osoby v Entrance
 - 1. Lucka chce v UI vykonať akciu
 - JavaFX pošle GET /users požiadavku na Spring Boot
 - Spring Boot pošle zoznam osôb



1



Spring Boot

HTTP Basic Auth

- Základná forma autentifikácie
- Každá HTTP požiadavka má hlavičku
 - Authorization: Basic `base64Encode(username:password)`
- Každé volanie stále posiela meno+heslo
- REST server musí ukladať heslo
 - Napr. v DB ako osolený heš

HTTP Basic Auth

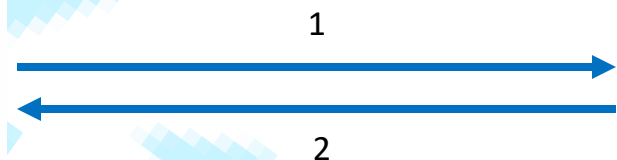
Príklad: Lucka chce zobrazíť všetky osoby v Entrance

1. Prihlási sa s menom a heslom v JavaFX

- POST /users/login
- SpringBoot povie OK
- JavaFX si do RestDao tried uloží správne meno a heslo

2. Lucka ide zobrazíť všetky osoby v Entrance

- JavaFX pošle GET /users požiadavku na Spring Boot
- Pribalí hlavičku Authorization: Basic s menom a heslom
- Spring Boot pošle zoznam osôb



Spring Boot

Dáme užívateľom schopnosť prihlásiť sa

Na prihlásenie užívateľ potrebuje 2 stĺpce

- Hash hesla – 60 znakov pre BCrypt
- **UNIQUE** prirodzený identifikátor
 - Typicky je to email
 - Alebo username
 - Pre jednoduchosť teraz iba nastavíme name na UNIQUE



Dáme užívateľom schopnosť prihlásiť sa

Na prihlásenie užívateľ potrebuje 2 stĺpce

- Hash hesla – 60 znakov pre BCrypt
- **UNIQUE** prirodzený identifikátor

- POZOR: Heslo ani jeho hash nedávame do entity ako inšt. premenné. Nechceme, aby lietali kade tade v kontroléroch, dao, ...

Riziko nechceného úniku.

```
name VARCHAR(10)  
active TINYINT(1)  
password_hash VARCHAR(60)
```

Indexes

REST Server + Spring Security

pom.xml

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

Niekde si vytvoríme takúto triedu – **Factory** pre Spring Framework

```
@Configuration  
public class SecurityConfig {  
}
```

SecurityConfig - UserDetailsService

Musíme najprv Spring Security naučiť ako získať z našej DB **hash hesla** a či je user **aktívny** na základe **mena** užívateľa

```
private record UserLogin(  
    boolean active, String passwordHash  
) {}  
  
@Bean Spring toto berie ako Factory metódu  
UserDetailsService userDetailsService() {  
    return username -> {  
        ...  
    }  
}
```

SecurityConfig - UserDetailsService

```
var query = "SELECT active, password_hash FROM users WHERE name = ?";
UserLogin userLogin;
try {
    userLogin =
Factory.INSTANCE.getJdbcOperations().queryForObject(query,
    (rs, row) -> new UserLogin(
        rs.getBoolean("active"),
        rs.getString("password_hash")
    ),
    username
);
} catch (EmptyResultDataAccessException e) {
    throw new UsernameNotFoundException("User not found", e);
}

...
```



Ak počet vrátených riadkov nie je 1, tak login považujeme za zamietnutý

SecurityConfig - UserDetailsService

```
import org.springframework.security.core.userdetails.User;
```

```
if (userLogin == null || userLogin.passwordHash() == null) {  
    throw new UsernameNotFoundException("User not found");  
}  
  
return User.withUsername(username)  
    .password(userLogin.passwordHash())  
    .disabled(!userLogin.active())  
    .authorities(List.of())  
    .build();
```

SecurityConfig - PasswordEncoder

Spring potrebuje ešte ako chceme heslá hashovať.

```
@Bean
PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

SecurityConfig - PasswordEncoder

Spring potrebuje ešte ako chceme heslá hashovať.

```
@Bean
PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

POZOR: Ak raz použijete konkrétny hešovací algoritmus s konkrétnym nastavením, tak s tým budete žiť roky.

Akkoľvek zmena algoritmu hešovania znemožní prihlásenie všetkým užívateľom.

SecurityConfig - AuthenticationProvider

UserDetailsService + PasswordEncoder = Authentication Provider

```
@Bean
AuthenticationProvider authenticationProvider(
    UserDetailsService userService,
    PasswordEncoder passwordEncoder
) {

    var provider = new DaoAuthenticationProvider(userService);
    provider.setPasswordEncoder(passwordEncoder);
    return provider;
}
```

SecurityConfig – SecurityFilterChain

@Bean

```
SecurityFilterChain entranceSecurityFilterChain(HttpSecurity http,  
AuthenticationProvider entranceAuthenticationProvider) throws Exception {  
    http  
        .csrf(csrf -> csrf.disable())  
        .sessionManagement(session -> session  
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))  
        .authenticationProvider(entranceAuthenticationProvider)  
        .authorizeHttpRequests(authorize -> authorize  
            .anyRequest().authenticated())  
        .httpBasic(httpBasic -> httpBasic.realmName("Entrance"));  
  
    return http.build();  
}
```

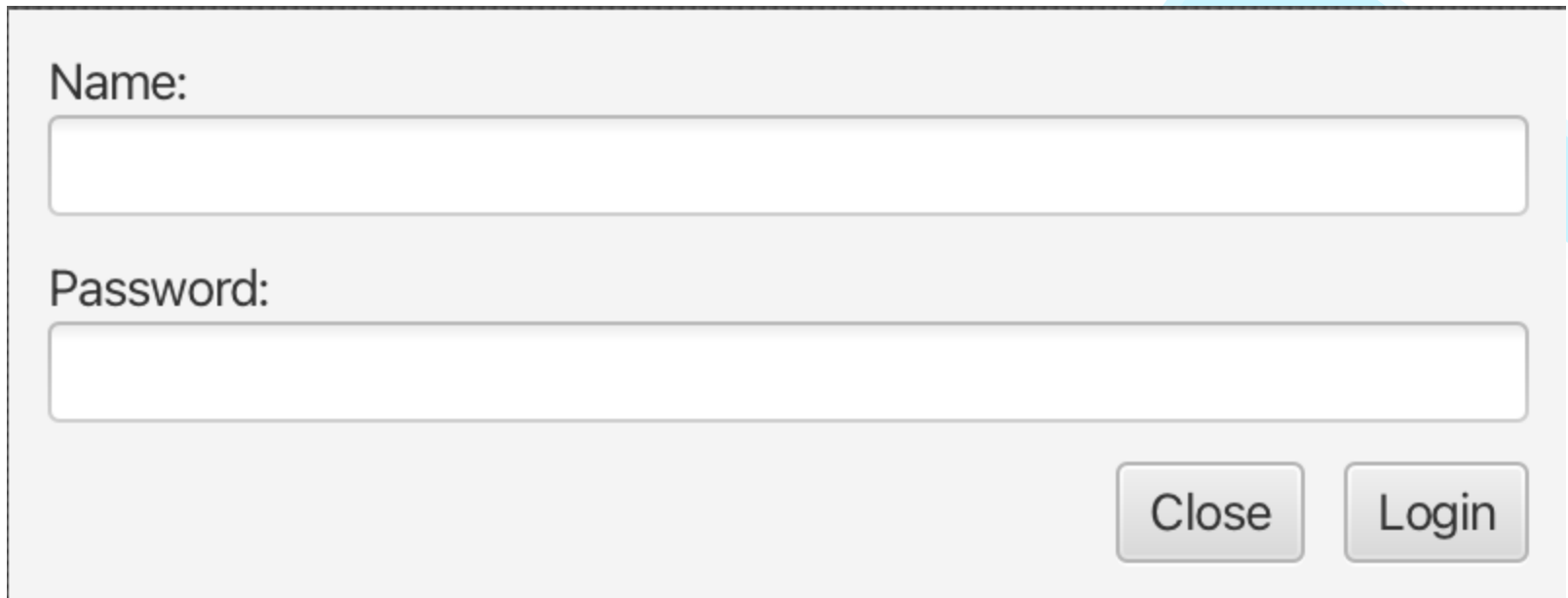
SecurityConfig – SecurityFilterChain

Povieme Springu: pomocou **httpBasic** vytiahni name+heslo z HTTP požiadavky a daj ich do **authenticationProvider**-a. Ten vytiahne hash z DB a skontroluje či heslo z hlavičky sedí s hashom. **AuthorizeHttpRequests** to zapne pre všetky naše RestController-y

@Bean

```
SecurityFilterChain entranceSecurityFilterChain(HttpSecurity http,
AuthenticationProvider entranceAuthenticationProvider) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .sessionManagement(session -> session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authenticationProvider(entranceAuthenticationProvider)
        .authorizeHttpRequests(authorize -> authorize
            .anyRequest().authenticated()
        ).httpBasic(httpBasic -> httpBasic.realmName("Entrance"));
    return http.build();
}
```

Login okno



A light gray login window with a thin black border. It contains two text input fields: one for 'Name' and one for 'Password'. At the bottom right, there are two buttons: 'Close' and 'Login'. The window is set against a background with a horizontal dotted line and abstract blue geometric shapes.

Name:

Password:

Close Login

Login kontrolér

```
@FXML
void loginButtonOnAction(ActionEvent event) {
    ...
    try {
        login(name, password);
        this.authenticated = true;
        closeButton.getScene().getWindow().hide();
    } catch (UnauthorizedException e) {
        errorLabel.setText("Invalid username or password");
    } catch (RuntimeException e) {
        errorLabel.setText("Could not connect to the entrance server");
    }
}
```

Login kontrolér

```
public void login(String name, String password) throws
UnauthorizedException {
    var previousToken = Factory.INSTANCE.getAuthToken();
    Factory.INSTANCE.setCredentials(name, password);
    try {
        var response =
Factory.INSTANCE.getEntranceHttpClient().POSTRequest("login", null,
Void.class);
        if (response.statusCode() < 200 || response.statusCode() >=
300) {
            Factory.INSTANCE.setAuthToken(previousToken);
            throw new UnauthorizedException();
        }
    } catch (RuntimeException e) {
        Factory.INSTANCE.setAuthToken(previousToken);
        throw e;
    }
}
```

Meno a heslo si budeme držat' vo Factory

```
@Getter
@Setter
private String authToken = null;

public void setCredentials(String name, String password) {
    this.authToken = name == null || password == null
        ? null
        : EntranceHttpClient.encodeBasicAuth(name,password);
}
```

Úprava HTTP klienta

Pomocná metóda na enkódovanie HTTP Basic Auth Authorization header do Base64 formátu

```
public static String encodeBasicAuth(String name, String password) {  
    var credentials = name + ":" + password;  
    var encodedCredentials = Base64.getEncoder()  
        .encodeToString(credentials.getBytes(StandardCharsets.UTF_8));  
    return "Basic " + encodedCredentials;  
}
```

Úprava HTTP klienta

Pripojenie HTTP Authorization headera do požiadavky

```
var builder = HttpRequest.newBuilder(this.baseUri.resolve(path)).GET();

if (Factory.INSTANCE.getAuthToken() != null) {
    builder.header("Authorization", Factory.INSTANCE.getAuthToken());
}

var request = builder.build();
```

Hlavný kontrolér



Ak používame REST DAO, tak otvoríme Login okno

```
void initialize()
```

```
if (Factory.INSTANCE.getDaoType() == Factory.DaoType.REST) {  
    this.showLoginWindow();  
}
```

Hlavný kontrolér

Otvorenie modálneho okna a vytiahneme si z neho výsledok (zavoláme getter inštančnej premennej z login kontroléra)

```
private void showLoginWindow() {  
    var loader = new FXMLLoader(getClass().getResource("Login.fxml"));  
  
    Parent pane;  
    try {  
        pane = loader.load();  
    } catch (IOException e) {  
        throw new RuntimeException("Error loading Login.fxml", e);  
    }  
  
    var controller = (LoginController) loader.getController();  
  
    var loginStage = new Stage();  
    loginStage.initModality(Modality.APPLICATION_MODAL);  
    loginStage.setTitle("Entrance login");  
    loginStage.setScene(new Scene(pane));  
    loginStage.showAndWait();  
  
    if (!controller.isAuthenticated()) {  
        this.nameTextField.getScene().getWindow().hide();  
    }  
}
```

Spúšťame

Potrebuje najprv nejakému užívateľovi nastaviť heslo ručne. Alebo si urobme nový main s takouto mini utilitkou na vloženie usera s heslom do našej DB.

```
var input = new BufferedReader(new InputStreamReader(System.in, StandardCharsets.UTF_8));

System.out.print("Username: ");
var username = input.readLine();
System.out.print("Password: ");
var password = input.readLine();

if (username == null || username.isBlank()) {
    throw new IllegalArgumentException("Username cannot be blank");
}
if (password == null || password.isBlank()) {
    throw new IllegalArgumentException("Password cannot be blank");
}

var passwordHash = new BCryptPasswordEncoder().encode(password);
Factory.INSTANCE.getJdbcOperations().update(
    "INSERT INTO users (chip_id, name, active, password_hash) VALUES (?, ?, ?, ?)",
    "",
    username,
    true,
    passwordHash
);

System.out.println("User created: " + username);
```

Otázky?

