

LOG!

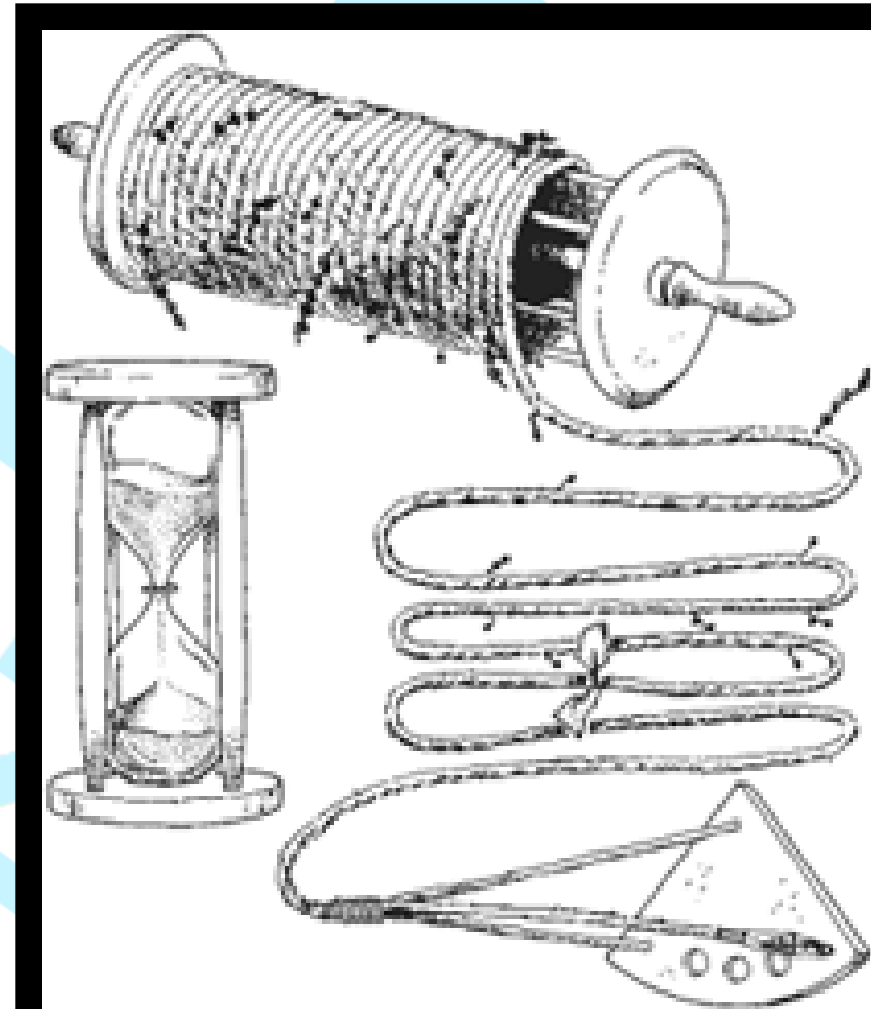


UINF/PAZ1c
epizóda 7

logovanie
a heslá

Logovanie

- Čo sa deje v systéme?
- Sledovanie medzikrokov algoritmu
- Čo sa stalo predtým, keď systém spadol?



System.out a System.err

- Štandardný a chybový výstup procesu
- Výpis iba na konzolu

```
Vyrátať som prvý krok  
x = 2.6621687e02  
Teraz rátam SELECT * FROM user;  
Vyrábam používateľa s id=1  
Vyrábam používateľa s id=2  
Vyrábam používateľa s id=3  
Neviem otvoriť súbor users.txt
```

Logovanie



java.util.logging

log4j

logback

slf4j

Dotiahnutie závislostí do pom.xml

```
<dependency>  
  <groupId>ch.qos.logback</groupId>  
  <artifactId>logback-classic</artifactId>  
  <version>VERZIA</version>  
  <scope>runtime</scope>  
</dependency>
```

```
<dependency>  
  <groupId>org.slf4j</groupId>  
  <artifactId>slf4j-api</artifactId>  
  <version>VERZIA</version>  
  <scope>runtime</scope>  
</dependency>
```

Logback je závislý od slf4j takže maven nám dotiahne aj slf4j-api-x.x.x.jar, takže treba aj tú (od logback 1.3+)

slf4j - použitie

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Arrays {
    private static final Logger logger
        = LoggerFactory.getLogger(Arrays.class);
}
```

Úroveň podrobnosti logov

- fatal
- error
- warning
- info
- debug
- trace

```
try {  
    ...  
} catch (PripojenieKDatabaseException e) {  
    logger.error("Databaza nie je dostupna.", e);  
}
```

Čo logovať

- Do **info** logujte začiatky "hlavných" udalostí (a prípadne aj úspešné konce)
 - užívateľ niečo urobil, spustenie automatickej úlohy
 - V čo najvyššej vrstve (controller)
- Logujte neúspešné konce "hlavných" udalostí
 - Odchytené výnimky
 - Zlyhaná validácia (niekde sme očakávali *true*, ale dostali sme *false*)
 - **error**; ak to neviete/nechcete riešiť (problém s DB spojením)
 - **warning**; ak to viete/chcete riešiť (súbor neexistuje - použijem default hodnoty; problém s DB spojením - počkám 5s a skúsím znova, alebo skúsím záložný DB server)
 - **info**; ak nastáva pomerne často (užívateľ nenájdený, nesprávne heslo)
 - **error** v čo najvyššej vrstve (controller), **warning/info** čo najbližšie k zdroju výnimky

Čo logovať

- Do **debug** logujte volania komplexnejších metód
 - Volanie DB/vzdialeného API, otvorenie/zápis súboru, spustenie iného skriptu/programu, volanie funkcie z C knižnice, ...
 - Situačne: komplexnejšie výpočty (vyhľadávanie v grafe, transformácia dátových štruktúr)
 - Typicky, ale nie nutne, čo najbližšie k zdroju (DAO, business)
- **Trace** používajte iba situačne
 - Napr. chyba nastáva iba na vzdialenom počítači, kde nemáte debugger
 - Dočasne dodáte logger.trace pre každý riadok kódu v problémovej metóde a spustíte tam upravenú verziu

Aká je úroveň logovania – logback.xml

- vytvoríme súbor v src/main/resources

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSXXX", UTC} [%thread] %-5level
%logger{5} -- %msg -- %kvp%n</pattern>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

alebo TRACE, DEBUG, INFO, ERROR, OFF

Logovanie v triede

- Ak chceme pre konkrétnu triedu logovať s iným levelom, rozšírime logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss,SSSXXX", UTC} [%thread] %-5level %logger{5}
-- %msg -- %kvp%n</pattern>
    </encoder>
  </appender>

  <root level="WARN">
    <appender-ref ref="STDOUT" />
  </root>
  <logger name="sk.gursky.entrance.UserEditController" level="TRACE" />
</configuration>
```

Logovanie s parametrami

- Klasický spôsob

```
logger.info("Validating user access with chip id={}" to a card reader with id={}", chipId, cardReaderId);
```

```
... INFO s.u.i.p.EntranceServiceImpl - Validating user with chip id=47656 to a card reader with id=1701
```

- Štruktúrované logovanie

- Ľahšie sa parsuje a strojovo spracováva

Možno priamo vypisovať ako JSON, logfmt

```
logger.atInfo().  
    .addKeyValue("op", "validateAccess")  
    .addKeyValue("chipId", chipId)  
    .addKeyValue("cardReaderId", cardReader.getId())  
    .log("Validating user access to card reader");
```

```
... INFO s.u.i.p.EntranceServiceImpl -- Validating user access to a card reader -- op="validateUserAccess" chipId="47656"  
cardReaderId="1701"
```

Čo má log obsahovať

- Presný čas,
 - RFC3339/ISO8601 formát
- Typ logu (INFO, ERROR, ...)
- Identifikátor operácie
 - typicky ID vlákna
 - vlastné umelé ID ak používame aplikačný framework/knižnicu na báze korutín
- Približné miesto logu (trieda, názov zdrojového súboru, ...)
- Stručná správa čo sa robilo (načítaváme/vyrábame niečo)
- Parametre
 - Typicky ID entity/entít
 - Akú operáciu sme robili – napr. názov metódy
 - **NIKDY nelogujte mená osôb, emaily, heslá, ani iné osobné údaje**

```
logger.atInfo().
```

```
.addKeyValue("op", "validateUserAccess")  
.addKeyValue("chipId", chipId)  
.addKeyValue("cardReaderId", cardReader.getId())  
.log("Validating user");
```

```
2023-10-29T17:56:40,827Z INFO [thread-123]s.u.i.p.EntranceServiceImpl -- Validating user – op="validateUserAccess" chipId="47656"  
cardReaderId="1701"
```

Koľko logovať

- Správne logovanie je umenie
- Príliš málo logov
 - Neviete čo sa presne stalo v prípade problému
- Príliš veľa logov
 - Aplikácia papká veľa CPU, RAM, disku, siete
 - Pomalá aplikácia
 - Rýchlo vybíja baterku (mobilné appky)
 - Nestíhate logy agregovať (ELK, Loki, Splunk)
 - ...



Logujeme kde-kade, kde-čo, rôzne

- kam posielat' logy
 - na konzolu (**ConsoleAppender**)
 - do lokálnych súborov (FileAppender, **RollingFileAppender**)
 - do tabuľky v databáze (DBAppender)
 - na vzdialený server cez ssh (SSLSocketAppender)
 - na mail (SMTPAppender)
 - do systémových logov (SyslogAppender)
 - logovanie pre každého klienta zvlášť (SiftingAppender)
- rôzny level logovania pre rôzne balíčky
- rôzny level do rôznych appenderov
- a mnoho ďalšieho na <http://logback.qos.ch/manual/>

Používatelia poriadne: bezpečné heslá

Bezpečnosť hesiel:

1. Bezpečné heslá
2. Bezpečné uloženie hesiel
3. Neposielat' heslá po sieti nešifrovane
4. Nelogovat' 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:

[Change passwords](#)

routing.cz

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

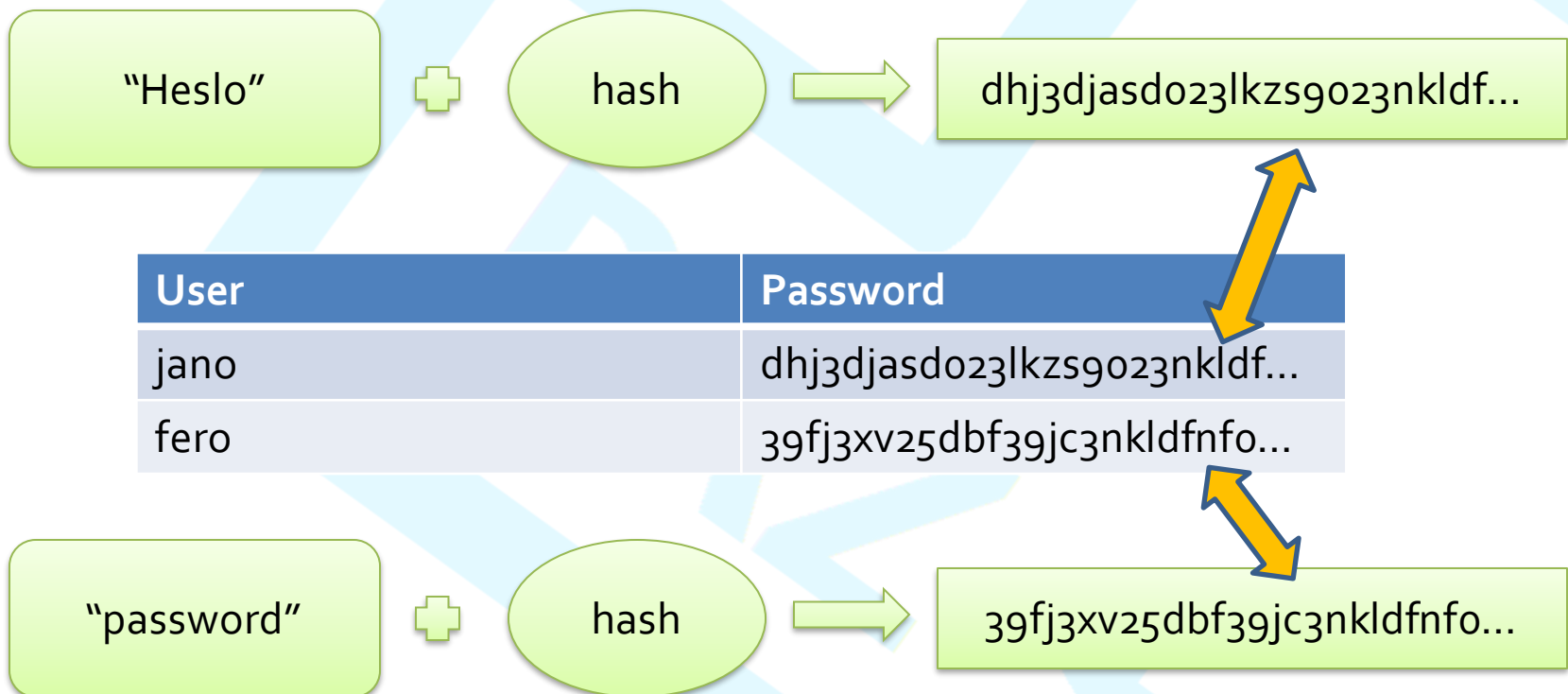
Treba donútiť užívateľov používať silné heslá

- Umelé pravidlá
 - Niečo rozumné
 - Aspoň 10 znakov, z toho 1 číslo, veľké písmeno, malé písmeno, špeciálny znak
 - Ale ak dĺžka viac ako 16 znakov, tak nekontrolujte nič
- Špecializované knižnice na kontrolu kvality
 - <https://github.com/GoSimpleLLC/nbvcxz>
 - Pozor, nepoužívajte defaultné nastavenia
 - Tie predpokladajú angličtinu



Bezpečné uloženie

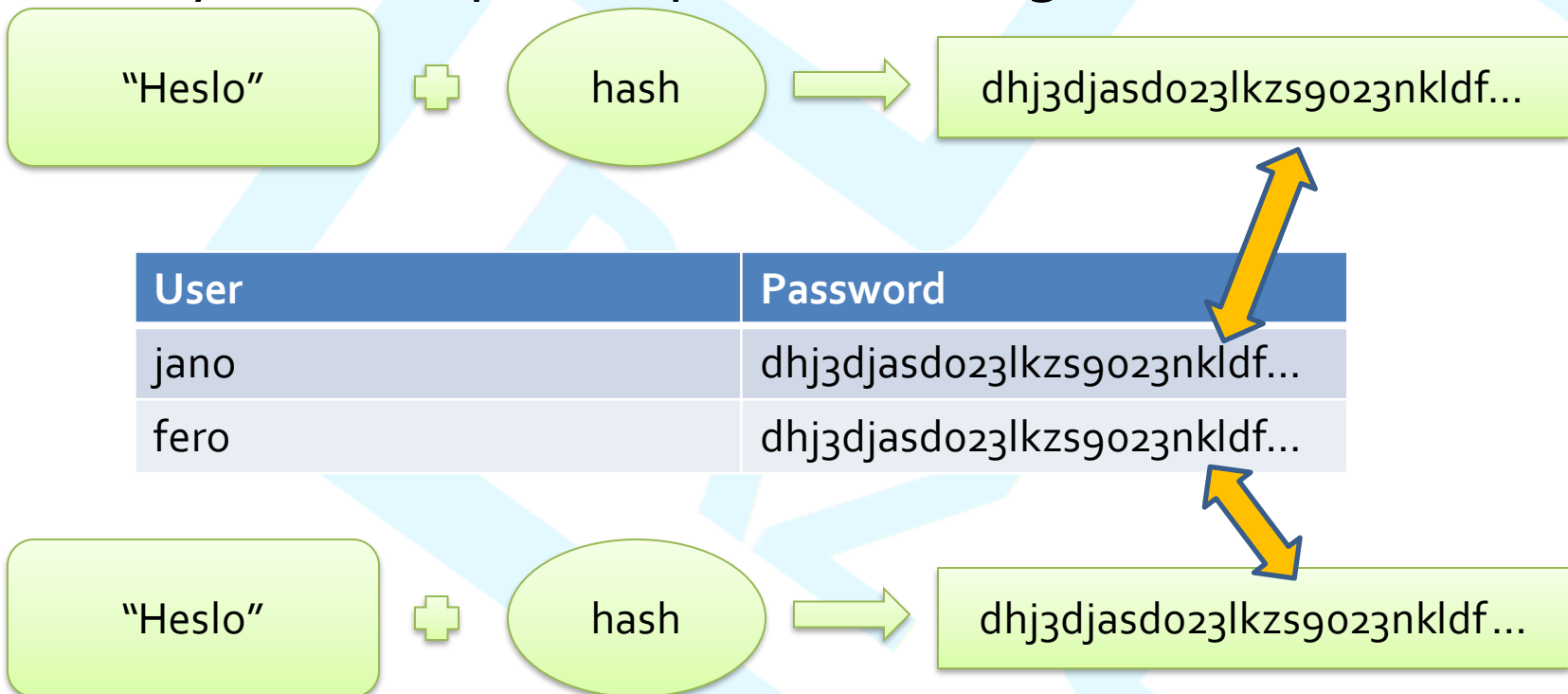
- **NIKDY!** neukladáme heslá v pôvodnom tvare
- Vždy iba kryptografický hash (SHA-256, SHA-384, ...)
 - Špec. hashe na heslá: **bcrypt**, scrypt, PBKDF2, Argon2



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 prinútíme útočníka rátať hash-e



| User | salt | Password |
|------|----------------------------|-----------------------------|
| jano | gkt478eru9sdfjdfu9orsdf39 | 894df0234kl sdf9osdlkfj9... |
| fero | sdfjklasdfjl45636r4ufjsd9o | psdfj9034nmjsov234osd... |



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-ag86-6ac53c3d1ee3

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"8ZòŽPŒ\$"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 cez BCrypt

(org.springframework.security.crypto.bcrypt)

- Importujeme cez maven knižnicu `spring-security-core`

```
/*vytvoríme sol'*/
```

```
$2a$10$V1bQOnfxjf76Rmu7yTrKiu
```

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

alg: 7 znakov
sol': 22 znakov

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

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

```
$2a$10$V1bQOnfxjf76Rmu7yTrKiuVX9Wj4L/IKadTNPZtbtnkgytpQo5tFO
```

```
/*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 tabuľkami |
| Dlhé heslá | Slovníkovým útokom |
| Pomalé hashovacie funkcie | Backtracku |

- Spoiler
 - Delegujte autentifikáciu na niekoho iného cez OpenID Connect (Keycloak, Azure, Google)

Otázky?

