



Универзитет „Св. Кирил и Методиј“ во Скопје  
**ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО**

Финален проект по предметот:  
**Безбедност на компјутерски системи**

**Тема:**  
**Безбедна веб – апликација за едноставна е-банка**

Стефан Андонов, 151020  
[stefan.andonov@students.finki.ukim.mk](mailto:stefan.andonov@students.finki.ukim.mk)

Јануари 2019г.  
Скопје, Република Македонија

## **Содржина**

<b>Вовед.....</b>	<b>2</b>
<b>Користени технологии.....</b>	<b>2</b>
<b>Детален опис на апликацијата .....</b>	<b>3</b>
<b>Безбедносни аспекти на апликацијата .....</b>	<b>8</b>

## Вовед

Целта на овој проект е да се изгради **безбедна** апликација која што симулира едноставна е-банка во која што имаме два типа на корисници (users): клиенти (clients) и вработени службени лица (employees). Во апликацијата се овозможени следните акции:

- Вработеното лице да додава нов клиент во системот
- Вработеното лице спроведува уплатница за финансиски средства од едно кон друго лице
- Секој корисник-клиент има опција да врши трансакции, односно да уплаќа пари на друго лице, за што е потребна двофакторска автентикација
- Секој корисник-клиент има опција да ги прегледува сите трансакции кои се однесуваат на него (прилив и одлив на финансиски средства)

Под безбедна апликација се подразбира апликација која што користи Htpps протокол наместо Http, користи SSL сертификати за енкриптирано испраќање на податоците на релација клиент-сервер, како и за автентикација на клиентот и на серверот еден пред друг. Дополнително апликацијата користи и автентикација со лозинки (и тоа силни лозинки кои не е лесно да се пробијат со напад на мапирање), како и двофакторска автентикација со помош на листа од токени кои што корисникот ги поседува (налик на апликацијата која што ја користи во моментот Комерцијална Банка АД Скопје).

Проектот е прикачен на [github.com](https://github.com).

## Користени технологии

Апликацијата беше дизајнирана со Spring Boot (back-end), како и со Thymeleaf (за front-end) и е дигната на Tomcat сервер.

За back-end се користеше слоевита MVC (Model, View, Controller) архитектура со програмскиот јазик Java. Беа користени многу библиотеки меѓу кои најважни за проектот се:

- `import org.springframework.security.*;`
- `import javax.persistence.*;`
- `import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;`

Првата библиотека се користеше за генерирање на сигурносни функции за автентикација/авторизација, втората се користеше за креирање на податочно репо и извршување queries врз него, додека пак третата е специфична библиотека и ја користев за хеширање и валидирање на лозинките на корисниците.

Сите податоци се чуваат во MySQL база, чии што табели можеме многу лесно да ја провериме за да видиме дали апликацијата функционира.

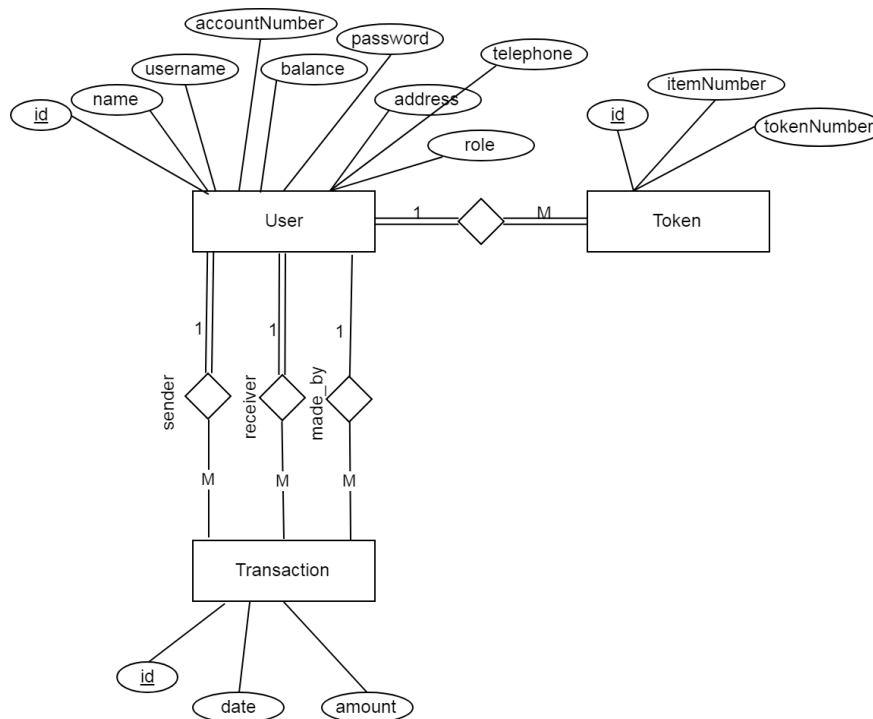
Слоевитата архитектура MVC во нашиот случај е составена од:

- модели (Класи за ентитетите во базите User,Token,Transaction)
- сервиси (една класа UsersService која што служи за бизнис логиката на апликацијата и комуницира со web слојот како и со базата на податоци)
- репозиториум (три класи што означуваат табели во базата за секој од трите модели)
- веб слој (неколку класи кои што се справуваат со сите можни Get,Post https requests).

Поради тоа што приказот на информации е во обичен формат, односно имаме форми на кои што внесуваме информации, копчиња за спроведување на определени акции, табели за приказ на некои информации, се користи чист HTML (без премногу дизајн и CSS), со цел да се согледа само функционалноста на серверот. Овие HTML страници се прикажуваат преку Thymeleaf, односно тоа значи дека сите страници кои што треба да ги прикажуваме ги чуваме во самиот Spring Boot проект во folder templates, и преку контролерите враќаме стринг што означува која html страница треба да се отвори.

### Детален опис на апликацијата

Како што веќе беше наведено во воведот, креирам едноставна апликација за е-банка (налик на некои од постоечките во Македонија), која што ќе се базира на следната база на податоци:



За секој корисник се чуваат следните информации:

- Id (вештачки клуч од тип Long)
- Name (назив на корисникот од тип String)
- Username (уникатен String)
- accountNumber (String што мора да биде уникатен, со должина 15 и сите карактери да се бројки)
- balance (Double вредност)
- password (хеширана вредност за password, String)
- address (String)
- telephone (String)
- role (енумерација од тип Role, што означува дали корисникот е обичен клиент или банкарско лице).

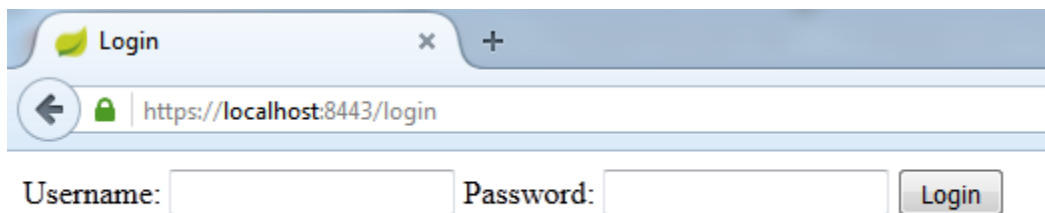
За секој токен се чуваат информации за:

- Id (вештачки клуч)
- itemNumber (цел број во рангот од 1-40 што го означува редниот број на токенот)
- tokenNumber (long број со 6 цифри, се гарантира уникатноста на 40те токени по клиент!)
- User (се чува id на клиентот на којшто токенот му припаѓа, следствено од 1-M релацијата во базата)

За секоја трансакција се чуваат информации за:

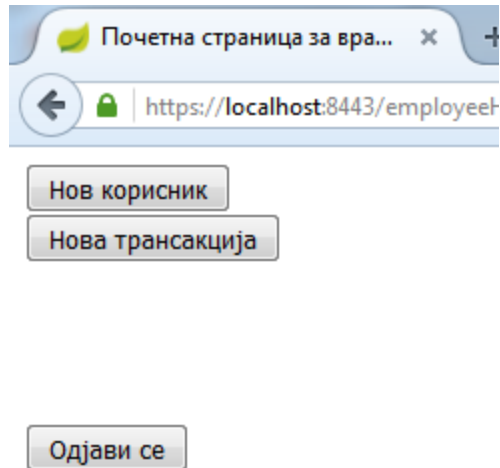
- Id (вештачки клуч)
- Date (датум)
- Amount (Double број – износ)
- Sender (id на корисникот којшто праќа пари во овој случај)
- Receiver (id на корисникот којшто прима пари во овој случај)
- Employee (id на вработениот којшто ја спроведува трансакцијата во овој случај)

При отварање на формата, на почетната страна потребно е корисникот да се логира:



Username:  Password:

При најавување на админ/службено лице вработено во банката се добива почетната страна за вработени која изгледа овака:



Со клик на опцијата „Нов корисник“ се отвара нов прозор во којшто вработенот креира корисничка сметка (account) за нов клиент во системот.

## Внесување на нов клиент во системот

Назив на клиентот

Стефан Андонов

Username

stefan.andonov123

Број на сметка

2000000000000000

Адреса

Гито бр.130/4 Неготино

Телефонски број

076434284

Почетена состојба на сметката:

15255

Внеси клиент

Внеси

Доколку сите податоци се во ред, односно, имаме уникатни username и број на сметка во банката, ќе се изврши оваа наредба и ќе се запише во базата нов корисник. За новиот корисник се генерира силна лозинка од 10 рандом карактери, како и 40 токени кои што истиот може да ги користи за да извршува трансакции спрема другите корисници

## Информации за нов корисник

Username: stefan.andonov123 Password: dmHhSLZBiR Tokens: 1: 1590608 2: 1797924 3: 1452084 4: 1053777 5: 1217042 6: 1053326 7: 1175011 8: 1141429 9: 1210836 10: 1084924 11: 1360033 12: 1884993 13: 1688450 14: 1429892 15: 1235494 16: 1341754 17: 1310867 18: 1138572 19: 1419513 20: 1656217 21: 1219620 22: 1329748 23: 1209787 24: 1855138 25: 1511105 26: 1200515 27: 1533923 28: 1248895 29: 1461308 30: 1019116 31: 1243363 32: 1291641 33: 1022048 34: 1162076 35: 1135882 36: 1309043 37: 1291934 38: 1005283 39: 1586083 40: 1314339

[Назад](#)

Овие информации банкарот ги принта за клиентот и се враќа назад. Доколку проба повторно да ги отвори истите информации (со кликање на копчето назад во browser), ќе ја добие страницата за грешка.

Сега ќе пробаме да извршиме трансакција (offline според уплатница) која ја спроведува вработениот според писмена уплатница. За истото вработениот на почетната страна треба да кликне на копчето: „Нова трансакција“ и го добива следниот преглед:

## Креирање на нова трансакција

Сметка на налогодавач:

2000000000000000

Сметка на примач:

2000000000000011

Износ:

1000

Изврши трансакција

По извршувањето на трансакцијата се враќаме на почетната страна на вработениот. Доколку се отвори табелата од базата на податоци може да забележеме дека трансакцијата навистина е извршена:

632 1000 2019-02-11 10:02:03 4 504 547

Дополнително и балансот е намален кај корисникот, односно апликацијата функционира како што е предвидено.

Сега ќе се најавиме како клиент на е-банката, односно со корисничкото име *stefan.andonov123* и ќе видиме дали апликацијата функционира и од страна на клиентите на банката. Почетната страна при најавување на клиентите е следната:

## Почетна страна на клиент

stefan.andonov123

Промена на лозинка  
Нова трансакција  
Извештај за трансакции

Одјави се

Најпрво ќе ја истестираме опцијата за промена на лозинка, бидејќи силната лозинка која што ја имаме изгенерирано рандом е тешка за помнење.

## Промена на лозинка на корисникот:

Нова лозинка:

.....

Смени лозинка

По промената, повторно се враќаме на почетна страна за клиентот и ќе ја провериме опцијата за извршување на трансакции online.

## Направи нова трансакција:

Корисник: stefan.andonov123

Сметка на примач:

2000000000000011

Износ

1560

Повторно внесете ја лозинката:

.....

17-иот токен

Реден број на токенот

17

1310867

Изврши



По извршувањето на трансакцијата, автоматски се префрламе на трета опција која што корисникот ја има а тоа е преглед на своите трансакции и извештај за балансот (моменталната состојба) на парички средства:

## Извештај за состојба и извршени трансакции за клиент:

stefan.andonov123

Моментално салдо: 10495.0

	Испраќач/Примач	Износ	Датум
Stefan Andonov	-1200.0	2019-02-11T10:55:59+01:00[Europe/Belgrade]	
Stefan Andonov	-1000.0	2019-02-11T10:56:46+01:00[Europe/Belgrade]	
Stefan Andonov	-1000.0	2019-02-11T11:01:07+01:00[Europe/Belgrade]	
Stefan Andonov	-1000.0	2019-02-11T11:02:03+01:00[Europe/Belgrade]	
Stefan Andonov	1000.0	2019-02-11T11:07:53+01:00[Europe/Belgrade]	
????? ?????????	-1560.0	2019-02-11T11:18:32+01:00[Europe/Belgrade]	

[Назад](#)

На крај се одјавуваме од системот. Доколку пробаме да се вратиме или да отвориме нешто друго откако ќе се одјавиме ја добиваме следната страна:

## Грешка

Настана грешка. Причината е една од следните:

- Не сте автентифицирани (најавени) за пристап на страната
- Немате дозвола за пристап на оваа страна.
- Имате внесено грешка информации.

## Безбедносни аспекти на апликацијата

### ➤ HTTPS конекција со серверот

При креирање на апликацијата се воспоставува HTTPS конекција наместо обичната HTTP конекција, како серверски сертификат се користи мојот потпишан студентски сертификат, се конфигурира и безбедносна порта на која што се извршува апликацијата (8443). Овие елементи се конфигурирани во два фајла и тоа:

- Класата ServersConnectorConfig која се наоѓа во фајлот config

```

private Connector createSslConnector() {
    Connector connector = new Connector( protocol: "org.apache.coyote.http11.Http11NioProtocol");
    Http11NioProtocol protocol = (Http11NioProtocol) connector.getProtocolHandler();
    try {
        connector.setScheme("https");
        connector.setSecure(true);
        connector.setPort(sslPort);

        protocol.setSSLEnabled(true);
        protocol.setKeystoreFile(keyStore);
        protocol.setKeystoreType(keyStoreType);
        protocol.setKeyAlias(keyAlias);
        protocol.setKeystorePass(keyStorePassword);

        return connector;
    } catch (Exception ex) {
        throw new IllegalStateException("can't access keystore: [" + "keystore"
            + "] or truststore: [" + "keystore" + "]", ex);
    }
}

@Bean
public TomcatServletWebServerFactory servletContainer() {
    TomcatServletWebServerFactory tomcat =
        new TomcatServletWebServerFactory() {

            @Override
            protected void postProcessContext(Context context) {
                SecurityConstraint securityConstraint = new SecurityConstraint();
                securityConstraint.setUserConstraint(SECURITY_USER_CONSTRAINT);
                SecurityCollection collection = new SecurityCollection();
                collection.addPattern(REDIRECT_PATTERN);
                securityConstraint.addCollection(collection);
                context.addConstraint(securityConstraint);
            }

        };
    tomcat.addAdditionalTomcatConnectors(createSslConnector());
    return tomcat;
}

```

- application.properties

```

app.ssl.port=8443
app.ssl.key-store=classpath:keystore/151020.pkcs12
app.ssl.key-store.password=password
app.ssl.key-store.type=PKCS12
#app.ssl.key-store.alias=keystore

```

#### ➤ Автентикација при логирање

Во базата на податоци се чува хеширана вредност од лозинките на клиентот, додека пак реалната вредност се заборува и не се чува никаде. При најавување корисникот треба да ја внесе својата лозинка. На ваквата лозинка и се пресметува

хеш вредност според истите параметри како и претходно, и хешираната вредност се споредува со таа што се чува во базата. Доколку е поклопуваат тие две вредности му се дозволува пристап на корисникот, во спротивно најавувањето не е успешно.

Во сесија се чуваат две информации employeeUsername и clientUsername. Откако ќе се најави некој корисник, во зависност од неговата улога (role), неговиот username се чува како сесиски атрибут.

При одјавување на соодветните сесиските атрибути им се става вредност Null.

➤ Двофакторска автентикација при сензитивни операции

Кога станува збор за сензитивни информации како на пример, праќање на парични средства од еден корисник на друг корисник, потребно е да бидеме што е можно посигурни дека корисникот што праќа пари е навистина тој што се претставува. Затоа се користи two factor-authentication со тоа што на корисникот кога сака да врши трансакција му се бара да внесе:

- Повторно лозинка
- Еден рандом избран токен (од 1-40) – (автентикација со нешто што поседува)

Доколку се потврдат овие две вредности со вистинските кои се чуваат во базата, тогаш трансакцијата се извршува.

➤ Авторизација на корисниците

Во системот имаме два типа на корисници и тоа клиенти и вработени (службени лица/админи). Мораме да внимаваме дали корисникот има пристап до одреден сајт, информација, ресурс итн.

Пример, корисник којшто е вработен не смее да пристапи на ниту една од страните кои почнуваат со /clientHome, како и корисник којшто е клиент не смее да пристапи на страница која почнува со /employeeHome.

Ова се постигнува со постојана проверка на Role атрибутот којшто секој корисник го има. Доколку улогата не е соодветна се фрла исклучок NotSufficientPermissionException и се појавува страницата со грешка (/error).