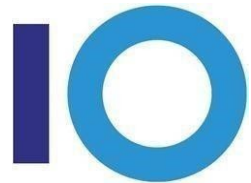




УНИВЕРЗИТЕТ „СВ. КИРИЛ И МЕТОДИЈ“ - СКОПЈЕ  
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО



Дипломски труд

Тема:

Веб апликација за трговија со криптовалути



**BLOCK TRADER**

Ментор:  
проф. д-р Ристе Стојанов

Изработил:  
Габриел Димитриевски,  
СИИС, 181225

Скопје, 2022

# Содржина

1. Вовед.....	3
2. Користени Технологии.....	4
3. Архитектура.....	5
База на податоци.....	5
Backend.....	7
a) Model package.....	7
b) Repository package.....	9
c) Service package.....	11
d) Controller package.....	15
e) dto.jsonparser package.....	19
f) Config package.....	19
g) Exceptions package.....	21
Frontend.....	22
a) Поглед за регистрација на нови корисници - /register или „/“.....	22
b) Поглед за најава на веќе постоечки корисници - /login.....	22
c) Поглед на почетна страница – „/home”.....	23
d) Поглед за додавање и подигнување на средства од апликацијата - „/balance”.....	23
e) Поглед за прикажување на поседувани криптовалути од најавениот корисник - „/myCrypto”.....	24
f) Поглед за прикажување на сите направени трансакции од најавениот корисник – „/myTransactions”.....	24
g) Поглед за прикажување на сите направени трансакции од сите корисници во апликацијата - „/allTransactions”.....	24
4. Кориснички сценарија.....	27
5. Заклучок.....	33
6. Референци.....	34

# 1. Вовед

Парични средства се користат на дневна основа веќе илјадници години наназад и бидејќи живееме во свет кој што има глобална економија, се јавува потребата за глобални дигитални валути.

Криптовалута претставува виртуелна или дигитална валута која што работи на блоковски вериги.

Моменталниот финансиски систем е централизиран и е раководен од големите банки и владите низ светот. Со текот на развојот и напредокот на светот доаѓаме до потреба од децентрализиран систем кој што ќе се разликува од веќе постоечните модерни начини на плаќање. Една одлична аналогија на децентрализиран систем е следниов пример. Доколку имаме банкнота од сто денари и директно ја дадеме на некој друг, рака на рака, во овој момент немаме зависност од некој друг ентитет како на пример банка. Со помош на дигиталните валути и блоковските вериги доаѓаме до истото сценарио.

Системите на криптовалутите овозможуваат трансакциите да бидат завршени брзо, безбедно, дозволуваат пристап до овие системите 24 часа на ден, седум дена неделно. Исто така, нема додатни такси кога се прават интернационални плаќања и секој може да ги користи само со креирање на корисничка сметка.

Голем дел од популацијата на глобални рамки иако имаат паметен телефон, немаат пристап до никаков систем за финансии. Моќта на криптовалутите е тоа што само преку паметен телефон и интернет врска да биде дел од глобалната економија.

Криптовалутите се веќе неколку години актуелна тема на разговор помеѓу луѓето, инвестирањето во нив, дали е вредно, како функционира, кој го контролира, дали постојат некои шеми и тактики со кои што може да се дојде до профитирање и така натака.

Поради тоа што системите на блоковските вериги не се директно контролирани од некоја индивидуа, група на луѓе, владите или централни авторитети, доаѓа и до една од најголемите негативни страни, илегални активности кои што се вршат преку истите.

После сите претходно наведени причини, како и темата самата по себе ми е доволно фасцинантна, и така дојдов до идеја за градење на еден ваков систем кој што ќе овозможи кориснички интерфејс преку кој можат да прават купопродажби на криптовалути само преку неколку едноставни кликови.

## 2. Користени Технологии

За развојот на оваа апликација користев повеќе технологии кои што се моментално многу актуелни и се користат секојдневно во многу ИТ компании.

За основните функционалности на апликацијата, односно за бекенд делот користев Spring Boot во комбинација со програмскиот јазик Јава додека пак погледите на сите кориснички интерфејси се изработени во React JS, JavaScript библиотека.

Пред да почнам со имплементација на фронтенд делот сакав да ги тестирам сите API's достапни од контролерите, за таа цел користев Postman. Пред да бидат тестирани API повиците веќе имав имплементирано JWT<sup>2</sup>[1] автентикација, и поради тоа во Postman исто така требаше да биде ставен и токен при праќање на барањата.

Станува збор за PostgreSQL база на податоци, и за самото моделирање ја користев алатката TerraER, бесплатна алатка со отворен изворен код.

Како што напредуваше развојот на кодот, сите локални промени ги ставав на GitHub во мојот репозиториум за овој проект[2].

За стилизирање на кодот и уредни погледи на екрани со различни големини користев Bootstrap и CSS<sup>3</sup>.

Зависностите кои што доаѓаат од надворешни библиотеки се дефинираат во pom.xml фајлот кој што доаѓа со Maven.

За генерирање на PDF<sup>4</sup> документ кој што служи како потврда за успешна исплата на средства е користена библиотеката lowagie.

Земањето на вредностите на криптовалути се земаат од надворешен API, за таа цел користам RestTemplate.

---

1 Application programming interface

2 JSON Web Token

3 Cascading Style Sheets

4 Portable Document Format

## 3. Архитектура

### База на податоци

Релациониот дијаграм на базата на податоци е направен во два дела, првиот дел е составен од табели кои што се меѓусебно поврзани и имаат силна зависност меѓу нив, додека пак другиот дел е составен од една самостојна табела која има цел да ги чува вредностите на криптовалути.

Првиот дел од базата е составен од четири табели. Првата табела е „Available\_app\_crypto“ и содржи три колони, уникатен идентификатор кој е примарен клуч за таа табела, име на криптовалутата и соодветна вредност на поседување од истата. Оваа табела ги чува сите криптовалути кои што се достапни во апликацијата за купување од страна на корисниците, значи доколку некој корисник е сака да купи некоја валута, се направат проверки во оваа табела дали апликацијата веќе ја поседува таа валута за да може да ја продаде. Оваа табела има релација 1-1 со втората табела „Transaction“ составена од редици кои што се потребни за да се зачуваат сите успешни трансакции кои што се направени во апликацијата. Содржи уникатен идентификатор кој се смета за примарен клуч на оваа табела, параметар за да се види дали корисникот продава или купува валута, име на валутата, време на извршување на трансакцијата, вредност на валутата во USD и вредност во крипто според моменталната цена соодветно за секоја валута.

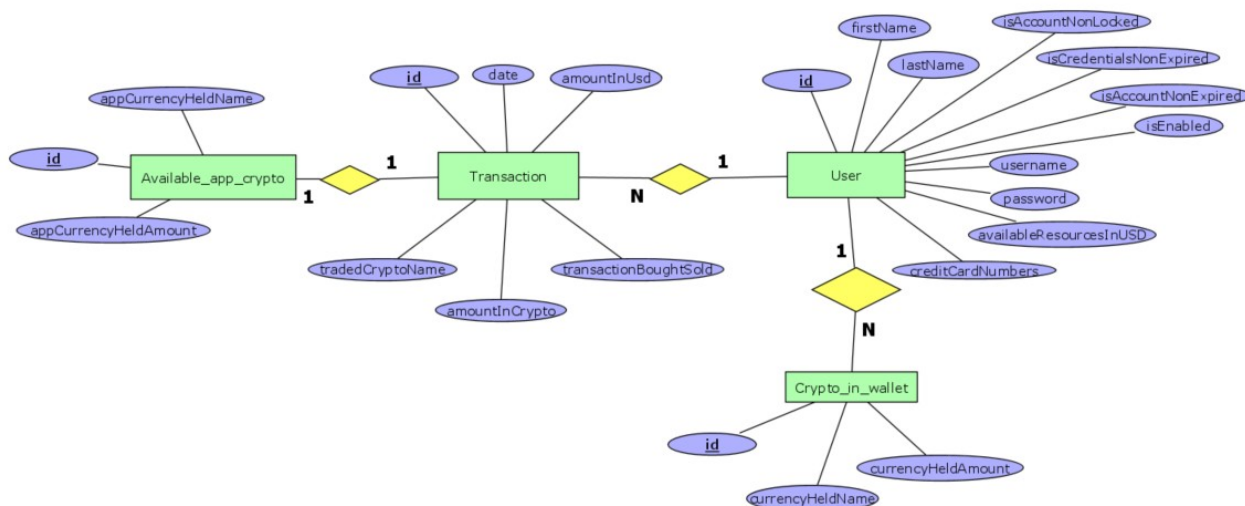
Според логиката дека еден корисник може да направи повеќе трансакции, и од друга страна една трансакција е уникатна и може да биде направена само од една корисничка сметка, доаѓаме до релација 1-N помеѓу табелата „Transaction“ и „User“.

Табелата за корисници содржи најмногу редици. Повторно, имаме уникатен идентификатор има улога на примарен клуч, име и презиме на корисникот, корисничко име, лозинка која што е хаширана, достапни расположливи ресурси во USD со кои што може да купува криптовалути од апликацијата, кредитна картичка се со цел кога ќе посака корисникот да повлече пари да се пратат на соодветна сметка, како и четири параметри кои што се важни од аспект на безбедноста, односно проверки дали корисничката сметка е валидна.

Доаѓаме и до последната табела од првиот дел „Crypto\_in\_wallet“ која што има за цел да ги чува сите криптовалути кои што ги поседуваат корисниците на апликацијата, без разлика дали тие имаат административни или обични привилегии. Од аспект на

редиви во табелата, ги имаме следниве: идентификатор кој е примарен клуч, име на валутата која што корисникот ја поседува и во која количина ја има истата.

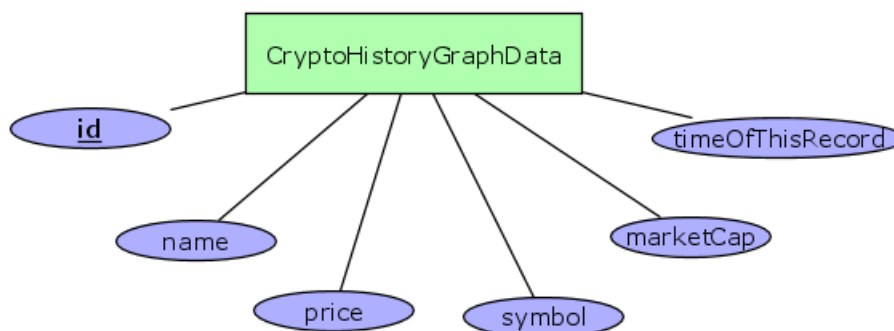
Притоа, како што можеме да забележиме од сликата подолу, табелата „User“ има релација 1-N со табелата „Crypto\_in\_wallet“, што значи дека корисниците имаат можност да прават трговија и имаат во сопственост повеќе криптовалути во ист момент.



Слика 1. База на податоци – прв дел

Во вториот дел од базата на податоци е една табела „CryptoHistoryGraphData“ која содржи податоци за сто криптовалути кои доаѓаат од надворешен извор. Оваа табела има за цел прибирање на податоци кои што се потребни за приказ на графикон со историја на криптовалути кој е прикажан на корисничкиот интерфејс. Табелата е составена од шест редици, идентификатор кој е примарен клуч, целосно име на валутите, време кога колоните се додадени во базата, цената која што ја имаат во тој момент, симболите позади кои што стојат криптовалути и пазарната вредност на валутите.

Релациониот дијаграм на оваа табела може да го видиме на сликата подолу.



Слика 2. База на податоци – втор дел

## Backend

Ова е дел од апликацијата кој што не се гледа директно на корисничкиот интерфејс но има клучна улога за нејзината функционалност.

Апликацијата е направена во слоевата архитектура, притоа се состои од седум пакети во кои е организиран целиот бекенд код.

Во составот на архитектурата се основите четири пакети: модел, репозиториум, сервисна логика и контролери или веб слој, и три останати пакети кои што носат со себе посебни логики. Ваквите поделби се направени за подобра организација на кодот.

### a) Model package

Како што кажува самото име на пакетот, тука се сите модели кои што беа опишани во секцијата за базата на податоци и целата логика директно поврзана со нив.

Во секој модел ги имам вклучено следниве Spring Boot анотации, @Getter, @Setter, @AllArgsConstructor – конструктор кој што ги содржи сите атрибути, @NoArgsConstructor – конструктор кој не содржи ниту еден атрибут, @Entity кој кажува дека овие класи претставуваат табели во база, како и @Table анотација со која што експлицитно е кажано името на класите како ќе бидат мапирани во базата на податоци. До сите овие полиња може да се додаваат вредности преку Set функции кои што се овозможени преку анотацијата @Setter, како и нивно пристапување со помош на Get функции кои што се овозможени преку анотацијата @Getter.

Уште една заедничка работа која што ја споделуваат сите класи е над идентификаторот кој што е примарен клуч за секоја од нив соодветно има две анотации, @Id со која што јасно се кажува дека овој атрибут е примарен клуч во

секоја од табелите соодветно и `@GeneratedValue` кој што доделува уникатна вредност на идентификаторот секогаш кога треба да се креира нова редица во базата.

Внатре во пакетот `Model` имам пакет „enumeration” составен од една енумерација „`Role`” која што го имплементира интерфејсот „`GrantedAuthority`” и има две полиња кои што помагаат за распределба на административни и обични привилегии на корисниците.

Притоа, исто така креиран е и пакет за пренос на податоци помеѓу бекенд и фронтенд под името `dto`. Овој пакет има пет класи кои што содржат по еден или неколку атрибути што ги примаат контролерите за да се воспостави успешно препраќање на податоците.

#### i. `User` класа

Оваа класа е мапирана во базата на податоци под името „`users`”.

Покрај веќе споменатите и опишани работи, оваа табела има и неколку свои уникатни својства.

За сите корисници кои што ќе бидат додавани во базата, колоната „`username`” мора да биде уникатена, во спротивно ќе биде прикажана грешка дека веќе постои корисник со такво кориснично име.

Класата „`User`” има еден атрибут од енумерација `Role`.

Притоа, во оваа класа имам и листа на трансакции кои што корисниците ги имаат направено. Ова е овозможено со чување на листа од објекти кои што доаѓа од класата `Transaction` и мапирање со анотацијата `@OneToMany`.

Исто така, во оваа класа имам и уште едно `@OneToMany` мапирање за пристапување до сите криптовалути кои што корисниците ги поседуваат. Ова е имплементарано преку чување на листа од објекти од класата `CryptoInWallet`.

#### ii. `AvailableAppCrypto` класа

Како што самото име кажува, оваа класа се состои од три атрибути кои што ги содржат сите криптовалути кои се достапни на апликацијата за продавање. Оваа класа е мапирана во базата на податоци под името „`available_app_crypto`”.



### iii. Transaction класа

Во класата за трансакции, покрај веќе опишаните редици од базата на податоци, имам имплементирано и два дополнителни објекти за апликацијата да може да функционира на посакуваниот начин.

Еден од објектите е од класата User, кој што го зачувува корисникот кој што ја направил оваа трансакција. Притоа, овој објект е мапиран со анотацијата @ManyToOne, и дополнително има и анотација @JsonIgnore.

Другиот објект кој што е дел од оваа класа е инстанца од класата AvailableAppCrypto кој што е соодветно мапиран со анотацијата @OneToOne и ги чува податоците за криптовалути со која што се прави трговија во дадената трансакција.

### iv. CryptoInWallet класа

Класата CryptoInWallet ги содржи сите атрибути од базата на податоци и плус дополнително еден @ManyToOne мапиран објект од класата User кој што ја означува состојбата на криптовалути кои што корисниците ги поседуваат. Исто така, има само еден дефиниран конструктор со два параметри, име на криптовалута и во колкава вредност таа валута е поседувана од даден корисник.

### v. CryptoHistoryGraphData класа

За разлика од другите Јава класи, станува збор за класа која што е независна од останатите во овој архитектурен слој. Од аспект на атрибути, ги содржи веќе опишаните во релациониот дијаграм соодветно со веќе споменатите анотации генерално за сите класи и анотациите за примарниот клуч.

## ***b) Repository package***

Дел од архитектурниот слој кој што е задолжен за конекција на Spring Boot апликацијата со базата на податоци. Составен е од пет интерфејси каде што секој од интерфејсите е задолжен за еден ентитет во базата. Притоа, секој од нив прави екстензија на интерфејсот JpaRepository кој што овозможува веќе некои основни функции за пребарување низ базата.

Именувањето на сите интерфејси од овој слој е името на класата од претходниот слој со наставка „Repository”.

#### i. UserRepository

Покрај веќе дефинираните функции кои што доаѓаат од JPA, имам додадено уште една функција која што пребарува корисник според нековото корисничко име кое што е препратено како параметар во функцијата.

#### ii. AvailableAppCryptoRepository

Исто така и овој репозиториум има само една дефинирана функција од моја страна која што враќа објект од класата AvailableAppCrypto. Замислата на оваа функција е да врати објект кој што е пребаран според името на криптовалутата која е пратена како параметар.

#### iii. TransactionRepository

За разлика од претходно, тука имаме две функции кои што враќаат листа од трансакции, објекти од класата Transaction.

Едната од функциите има за задача да ги врати сите трансакции кои што се направени во апликацијата во опаѓачки редослед, или со други зборови, таа трансакција која што ќе биде последно направена да биде прва прикажана.

Другата функција ги враќа сите направени трансакции за даден корисник кој го прима како параметар на функцијата и дополнително истите ги сортира според датумот на запушување во базата по опаѓачки редослед.

#### iv. CryptoInWalletRepository

Овој репозиториум не содржи дополнителни функции кои што се имплементирани, тие што се веќе предефинирани од JPA се доволни за потребните на оваа апликација.

#### v. CryptoHistoryGraphDataRepository

Репозиториум составен од две дефинирани функции.

Едната функција враќа листа од објекти од класата CryptoHistoryGraphData сортирани според опаѓачки редослед. Оваа функција служи за враќање на JSON<sup>5</sup> објект до фронтенд со податоци кои што се потребни за да се прикаже соодветниот градикон.

---

5 JavaScript Object Notation

Втората функција е враќање на симболите од сите сто криптовалути со кои што работи оваа апликација. Ова е направено со користење на `nativeQuery`, со `@Query` анотација која содржи PostgreSQL прашање.

### **c) Service package**

Бизнис логиката, односно сврзувањето на сите претходно веќе споменати работи и создавање на една нова целина се прави во сервисниот архитектурен дел. Апликацијата содржи четири интерфејси кои што имаат веќе дефинирани имиња на функции со нивни параметри кои што соодветните класи кои наследуваат од истите треба да ги имплементираат. Притоа, за подобра организација на кодот имам дефинирано пакет “implementation” каде што се вршат овие спојувања на класите со интерфејсите.

#### **i. UserServiceImplementation**

Класа која што е од клучна улога и содржи најмногу бизнис логика во апликацијата.

Најгоре во класата се дефинираат приватни и финални променливи од претходниот архитектурен слој, репозиториум. Овој концепт на вклучување на претходен слој во следниот или во слој од исто ниво се нарекува `Dependency Injection`. Јас го применувам овој концепт во оваа класа со креирање на конструктор за иницијализација на сите објектите од претходните слоеви. Во овој случај тие објекти се од класите: `UserRepository`, `PasswordEncoder`, `CryptoInWalletRepository`, `AvailableAppCryptoRepository`, `TransactionRepository`.

Оваа класа е составена од осум функции.

#### **1. register**

Оваа функција креира и запишува нов дојден корисник на апликацијата. Прима параметри кои што се дел од класата `User`, соодветно прави проверки дали полињата `username` и `password` се правилно пополнети, прави проверка дали `password` и `repeatPassword` имаат исти вредности, ја хашира лозинката и го зачувува корисникот доколку сите претходно споменати проверки успешно поминат, во спротивно се фрлаат соодветни исклучоци.

## 2. LoadUserByUsername

Оваа функција е доста едноставна, прави повик до функцијата за пронаоѓање на корисник според корисничко име од репозиториумот `userRepository`.

## 3. AddUSDMoneyInWallet

Ова е функција која што прима параметар т.е депозит кој што го прави корисникот за да додаде нови расположливи средства со кои може да прави понатамошна трговија. Го зема моментално најавениот корисник преку класата `SecurityContextHolder`, ги зема веќе постоечките расположливи средства, доколку корисникот ги има, и само го додава депозитот.

## 4. buyCrypto

Една од функциите која што има најголема комплексност.

Прима два параметри, името на валутата која што корисникот сака да го купи и во која вредност. Вредноста на валутата која што сака корисникот да ја купи е во USD.

Прво се прави проверка дали апликацијата ја има достапна за продажба таа криптовалута во дадената вредност, во спротивно се фрла исклучок `NotEnoughAppResourcesException` со порака „Sorry, we don't have enough from this currency!“. Следна проверка која што се прави е дали корисникот има доволно расположливи средства за тргување со тоа што се зема моментално најавениот корисник и се проверува неговата состојба. Доколку расположливите средства се помали од побараните средства за купување се фрла исклучок `NotEnoughUserResourcesException` со порака „Sorry, you don't have enough from this currency!“.

За да се направи точна пресметка по колкава е моменталната вредност на валутата која што сака корисникот да ја купи се прави повик до функцијата `getCurrencyRealTimePrice`, објаснета подолу.

Наредниот дел од логиката на оваа функција е да се провери дали корисникот ја има веќе поседувано оваа валута, доколку ја има се додаваат купените средства на веќе постоечките на сметката на корисникот, но доколку првпат ја купува оваа валута се додава нова колона во базата на податоци.

Следно што се случува е намалување на сметката на расположливи средства кај корисникот и се намалува расположливата количина за продажба на валутата на страна на апликацијата.

Се креира нова трансакција која што врши евиденција дека е направено купување на валутата и на крајот се прават записи во база во сите ентитети кои што имаат директно влијание во пресметките на вредностите.

## 5. SellCrypto

Исто како и претходната функција, и оваа функција прима два параметри, името на валутата која што сака корисникот да ја продаде и количината во која што сака да ја продаде.

Се прави проверка дали корисникот ја има поседувано оваа валута и количината која што сака да ја продаде. Доколку не, се фрла исклучок `NotEnoughUserResourcesException` со порака „You don't have enough from this cryptocurrency to sell!“

Следна акција која што се врши е додавање на расположливи средства на страна на корисникот во USD како и додавање на продадената криптовалута на страна на апликацијата.

Исто така, и тука се креира нова трансакција дека е направено продавање на криптовалутата од страна на најавениот корисник и се зачувуваат во база сите овие промени на вредностите.

## 6. withdrawAmount

Оваа функција прима параметри кои ја означуваат вредноста на пари која што сака корисникот да ја повлече од апликацијата како и самиот корисник.

Се зема корисник и се прави проверка дали вредноста која што сака да ја земе е поголема од вредноста на расположливи средства кои што ги има, доколку тоа е случајот се фрла исклучок `NotEnoughUserResourcesException` со порака

"Sorry, you don't have *amountToWithdraw* available resources", каде што *amountToWithdraw* динамички ја носи вредноста која што корисникот се обидува да ја земе.

Доколку има расположливи средства, се одземаат од веќе постоечките и се евидентираат промените во база.

## 7. GetLoggedUserAvailableResource

Преку класата `SecurityContextHolder` повторно го земам моментално најавениот корисник во апликацијата и се повлекуваат податоците за корисникот од

базата на податоци, се земаат расположливите средства кои што корисникот го поседува и се враќаат назад до функцијата од каде што се повикува.

#### 8. `getCurrentRealTimePrice`

Оваа функција прима параметар име на криптовалута и ја враќа моменталната вредност на валутата по која што се пребарува. Логиката позади оваа функција е подетално објаснета во делот за `CronJobServiceImpl`.

#### ii. `TransactionServiceImpl`

Во оваа класа може да најдеме повторна имплементација на концептот Dependency Injection од претходниот архитектурен слој со иницијализирање на Beans од интерфејсите `TransactionRepository` и `UserRepository` преку конструктор. Тука има имплементација на две функции кои враќаат листа на објекти од класата `Transaction`. Едната функција ги враќа сите направени трансакции во апликацијата под услов моментално најавениот корисник да е администратор, во спротивно враќа празна листа. Втората функција е задолжена за враќање на сите трансакции кои што се направени од најавениот корисник сортирани во опаѓачки редослед.

#### iii. `CryptoHistoryGraphDataServiceImplementation`

Станува збор за класа која што е задолжена за чување на историја на криптовалутите. Прави повик на неколку функции од соодветниот репозиториум и има една функција која што е дополнително имплементирана. Функција која што е се повикува од класата `CronJobServiceImpl` и е задолжена за правење на записи во ентитетот `crypto_history_graph_data`.

#### iv. `PDFGeneratorServiceImpl`

Намената на оваа класа е да генерира PDF кој што ќе означува потврда дека е направено плаќање од страна на апликацијата до корисникот кој што сака да повлече пари. Притоа, прави проверка дали корисникот има доволно пари за да повлече. Оваа проверка се прави со повик на функцијата `withdrawAmount` до имплементацијата на `UserService` интерфејсот која што беше претходно објаснета.

#### v. CronJobServiceImpl

Овозможување на повик до надворешен сервер кој што враќа податоци за сто криптовалути преку RestTemplate[3]. Во header делот од барањето е ставено името според документацијата од CoinMarketCap[4], "X-CMC\_PRO\_API\_KEY" соодветно со вредноста на мојот генериран клуч на нивната платформа "d547fd3a-f9a6-4fdd-9dd0-71774b4cdcd5". Тука е дефиниран Cron Job преку анотацијата @Scheduled над функцијата со кој што се извршува на секои пет минути „ 0 \*/5 \* ? \* \* ” и прави повик до серверот.

За извршувањето на функции на определено време исто така треба е ставена анотација @EnableScheduling[5] во класата CryptotradingApplication.

Дефинирано е URI до кое што се прават повиците „https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest“ и се прави барање до серверот. Откако ќе се вратат податоците со помош на ObjectMapper соодветно се прави мапирања на вредностите со класата APIResponseCryptocurrencies која што е објаснета малку подоле.

Исто така, овие вредности се зачувуваат во ентитетот кој што е задолжен за на историја crypto\_history\_graph\_data.

#### d) Controller package

Пакетот каде што се сместени контролерите е одговорен за обработка на дојдовните REST API барања, правење на промени на бекенд делот или земање на податоци од истиот и враќање на истите преку JSON.

Генерално кажано, пред да се дефинираат класите на сите контролери во апликацијата стојат некои неопходни анотации за функционирање на истите на посакуваното начин.

Првата анотација е @RestController[6] која и дава до знаење на Spring Boot апликацијата дека станува збор за класа која што е контролер.

Повеќето од контролерите имаат предефинирана рута која што мораме да ја напишеме во за да пристапиме до ресурсите од контролерите во оваа апликација. Тоа го правам со доделување на нова анотација @RequestMapping("/api").

За пристапување на апликацијата од надворешни рутирања кои што не се директно дел од Spring Boot, ја користам анотацијата @CrossOrigin(value="\*"), каде што ѕвездата означува дека овој ресурс може да се пристапи од сите надворешни рути.

Кај анотациите `@CrossOrigin` и `@RequestMapping` има единствено мала разлика во контролерите за најава и регистрација на нови корисници, и тие различности се опфатени во нивните соодветни секции подолу во објаснувањата.

#### i. `CryptoApiController`

Повторно тука има појавување и имплементација на концептот `Dependency Injection` од претходниот архитектурен слој од класата `CryptoHistoryGraphDataService`. Во овој контролер има имплементација на две функции.

Првата функција е земање на ресурси од серверот со доделување на анотацијата `@GetMapping` на рутата `„/crypto“` кој што враќа листа од објекти од класата `CryptoHistoryGraphData`. Оваа листа на криптовалути која што се обработува се користи за приказ на податоци на графиконот на корисничкиот интерфејс. Притоа прима еден параметар преку анотацијата `@RequestParam` кој што е симболот за кој корисникот избрал за да добие графикон од таа криптовалута. Се прави повик до функција од класата `CryptoHistoryGraphDataService` во сервисната логика која што ја враќа листата на објектите кои се пуштаат низ `Java Stream` кој прави филитритање според симболот на криптовалути која корисникот ја избрал и се прави лимитирање на тринаесет објекти кои што се праќаат назад. Ограничувањето на трианесет објекти е поради прикажувањето на графиконот на еден час, односно на секои пет минути по еден нов објект.

Второто мапирање е исто така `@GetMapping` на рутата `„/cryptoSymbolName“` која што прави повик до класата `CryptoHistoryGraphDataService` и ги враќа назад сите уникатни симболи на сто криптовалути. Понатака ќе видиме дека има листа на корисничкиот интерфејс кој што може корисниците да селектираат симбол кој што се препраќа до рутата `„/crypto“` која што е објаснета погоре.

#### ii. `LoginRestController`

Станува збор за единствен контролер кој што има ограничување за пристапување на ресурси од надворешни ентитети. Внатре во анотацијата `@CrossOrigin` има дефинирани рути за пристап само од повици кои што доаѓаат од `"http://localhost:3000"` и `"http://localhost:3001"`.

Преку конструктор се прави иницијализација на објект кој што доаѓа од класата `JWTAuthenticationFilter`.



@RequestMapping анотацијата е дефинирана на основна рута „/api/login“ и притоа единствената функција во оваа класа која што е со @PostMapping анотација нема предефинирана рута, но цел да најави корисник доколку ги исполнува исловите од функциите до кои што се прават повици дефинирани во JWTAuthenticationFilter класата.

### iii. PDFController

После дефинирањето на класата може да се види дека има објект од класата PDFGeneratorServiceImpl. Со пристапување на рутата „/generatePDF“ која што е мапирана со @GetMapping. Соодветно се сетираат headerKey и headerValue параметри кои што се потребни за генерирање на PDF кој што го прави функцијата export од класата PDFGeneratorServiceImpl. Притоа, функцијата за рутата „/generatePDF“ прима три параметри HttpServletResponse, сумата која што корисникот сака да ја повлече од апликацијата и најавениот корисник. [7]

### iv. RegisterController

Основната рута за пристапување на ресурси од овој контролер е „/register“. Се прави Dependency Injection од интерфејсот UserService и има една функција @PostMapping која што нема дефинирана рута, односно се пристапува со POST повик до основната рута.

Прима @RequestBody објект од класата RegisterUserDto кој што содржи податоци потребни за регистрација на нов корисник. Се прави повик до функција register преку објект од UserService каде што се регистрира нов корисник и се запишуваат податоците во база. Логиката за регистрација е објаснета во делот на config package каде што се сите работи поврзани со безбедност на апликацијата.

### v. TransactionController

Овој контролер има две функции кои што имаат прилично слични задачи. Двете функции имаат анотација @GetMapping. Првата функција која што е мапирана на рутата „/transactions“ ги враќа сите трансакции кои што се направени во апликацијата, притоа за корисникот да може да ги добие назад овие трансакции мора да има административни привилегии.

Втората функција е земање на сите трансакции кои што ги направил најавениот корисник. Рутата на оваа функција е „/loggedUserTransactions“.

Двете функции прават повици до класата `TransactionService` од која што имаме иницијализирано објект во контролерот со помош на `Dependency Injection`.

#### vi. `UserController`

Како што можевме да видиме, најмногу бизнис логика имаме во делот од `UserServiceImpl` класата. Па според тоа, тука имаме и најмногу функции кои што се овозможени за пристапување.

На почетокот на класата имаме `Dependency Injection` од два архитектурни слоеви до пивисоко ниво, `UserService` и `UserRepository`.

Рутите кои што се мапирани со `@PostMapping` во овој контролер се `„/addMoney“`, `„/buyCrypto“` и `„/sellCrypto“`. Сите три параметри примаат по еден различен `Data Transfer Object` преку анотација `@RequestBody`. Внатре во овие објекти ги имаме вредностите кои што се користат за соодветни промени на промени во базата на податоци.

`DepositCashDto` е класа која што е примена од функцијата за рутата `„/addMoney“` и соодветно си прави додавање на расположливи средства на сметката на корисникот. Оваа функција прави повик до функцијата `addUSDMoneyInWallet` од интерфејсот `UserService`.

`BuyCryptoDto` е класа задолжена за прифаќање на вредности за која криптовалута и во која количина корисникот сака да ја купи и притоа прави повик до функцијата `buyCrypto` од `UserService` интерфејсот. Рутата за купување на криптовалута е мапирана на `„/buyCrypto“`.

Последната функција од `@PostMapping` мапирањата во оваа класа прима објект од класата `SellCryptoDto` која што зема вредност за која валута и во која количина корисникот сака да ја продаде и притоа прави повик до функцијата `sellCrypto` од `UserService` интерфејсот.

Следно доаѓаат функциите од оваа класа кои што имаат мапирање `@GetMapping`. Овие функции не примаат ниту еден параметар.

Функцијата со рута `„/getLoggedUserCryptocurrencies“` ги враќа сите криптовалути кои што најавениот корисник ги поседува. Овој резултат се добива преку правење на повик до функцијата `getCryptoInWallet` од `UserRepository` интерфејсот.

Рутата `„/loggedUser“` исто така прави повик до функција од интерфејсот `UserRepository`. Со повикот до `findByUsername` го добиваме моментално најавениот корисник и истиот се враќа назад до корисничкиот интерфејс.

Како последна рута од оваа класа е „/availableResources“ која што ја враќа вредноста на расположливи ресурси од најавениот корисник до корисничкиот интерфејс овозможена преку повик на функција од UserService интерфејсот.

#### ***e) dto.jsonparser package***

Пакет кој што е задолжен за парсирање и соодветно преземање на податоци од JSON објектите кои што доаѓаат при повик на рутата „https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest“ која што се користи за добивање на информации за последните цени за сто криптовалути во USD.

Притоа, организацијата на класите во овој пакет е според информациите кои доаѓаат од JSON-от објектот кој што е преземен од споменатото API и се земаат само информации кои што се потребни за развојот на оваа апликација.

Кога се користи ObjectMapper имињата на JSON-от мора да бидат исти како и имињата на променливите во Java класите и од таа причина неколку пати се појавува анотацијата @JsonProperty(“”) која што во наводниците ја означува вредноста на променливата дефинирана во JSON објектот, и ни овозможува да имаме поинакви имиња во Java класите за подобра организација на кодот.

Позади класата APIResponseCryptocurrencies се крие целата структура на податоци кои што ги имаме во JSON објектот. Внатре е дефинирана листа на објекти од класата Cryptocurrency која што содржи идентификатор, име на криптовалутата, симбол и објект од класата Quote. Класата Quote има само еден објект од класата USD која што содржи податоци за цената и пазарната вредност на криптовалутата.

#### ***f) Config package***

Станува збор за пакет кој што е задолжен за безбедноста на апликацијата. Цела сервисна логика поврзана со најава и регистрација на нови како и веќе постоечки корисници е дел од класите кои што се наоѓаат низ овој пакет.

Притоа, внатре во пакетот за конфигурација се наоѓа уште еден пакет каде што се се предефинирани филтри за JWT автентикација и авторизација.

#### i. WebSecurityConfig class

Оваа класа содржи декларирање на две променливи од класите PasswordEncoder и класата CustomUsernamePasswordAuthenticationProvider како и нивна иницијализација преку конструктор.

Има два метоци кои што се исто именувани под називот configure.

Првата метода прима објект од класата HttpSecurity и во неа се дефинираат кои рути се сметаат за основни и пристапливи од сите корисници во апликацијата, за кои од нив е потребно корисниците да имаат административни привилегии како и бришење на Cookies кога корисникот сака да се одјави од апликацијата.

Другата функција прима објект од класата AuthenticationManagetBuilder и ја повикува функцијата authenticationProvider која зема објект од класата CustomUsernamePasswordAuthenticationProvider.

#### ii. JWTWebSecurityConfig

Логиката на имплементација е слична како и во WebSecurityConfig класата, но тука се користат дефинираните филтри за автентикација и авторизација на корисниците. Друга разлика е тоа што е воведена нова логика која не ја чува состојбата во сесија. Кога има комуникација од надворешни сервиси до Spring Boot апликацијата мора секогаш да се препраќа JWT токенот за да се автентичира корисникот на страна на серверот и со тоа му се дава до знаење на серверот дека станува збор за валидно HTTP<sup>6</sup> барање.

JWT токенот мора секогаш да биде енкодиран кога се препраќа како апликацијата не би имала безбедности слабости.

#### iii. JWTAuthConstants

За подобра организација на кодот креирана е класа која што содржи финално дефинирани променливи кои што нема да се менуваат и се потребни за JWT автентикацијата.

#### iv. CustomUsernamePasswordAuthenticationProvider

Класа составена од две функции.

Првата функција е supports која што само проверува дали параметарот кој што е примен е од класата UsernamePasswordAuthenticationToken.

---

6 Hypertext Transfer Protocol

Класата `authenticate` има поголем дел од работите поврзани со логиката на оваа класа. Овој метод се повикува кога се прави POST барање на „/login“ рутата. Откако корисникот ќе внесе информации за корисничкото име и лозинката во формата на корисничкиот интерфејс, податоците стигаат како параметар кој што ја прима оваа функција. Се прават проверки дали овие полиња се правилно пополнети, дали хашираните вредности на лозинките од корисникот се исти, лозинката која што доаѓа од форма од корисничкиот интерфејс со таа што се наоѓа во базата на податоци. Доколку сите проверки поминат успешно, корисникот ќе добие генериран JWT токен кој што ќе го користи извесно време додека прави акции на апликацијата, во спротивно ќе се фрли исклучок со порака „Invalid User Credentials“.

Апликацијата при секое барање треба да користи JWT токен за комуникацијата да биде успешна. Преку форма од кориснички интерфејс се праќа POST барање до серверот. Серверот доколку открие дека корисникот е валиден, односно неговите кренцијали се точни тогаш креира за него JWT токен со користење на таен код. Генерираниот токен се враќа назад до прелистувачот кој што ја запишува оваа информација во соодветно колаче и при секое ново барање до серверот оваа информација мора да биде испратена и служи за автентикација на корисникот во Authorization Header.

Форматот на Authorization Header е следниов: *Authorization: Bearer <token>*.

Серверот ги чита информациите од Authorization Header, го гледа JWT потписот и доколку се валидни продолжува со процесирање на барањето од корисникот.

### **g) Exceptions package**

За прикажување на поточни грешки, креирани се шест различни типови на исклучоци кои ми беа потребни во текот на развојот на апликацијата. Сите класи наследуваат од класата `Exception` и праќаат различен вид на порака во конструкторот.

Типовите на исклучоци се следниве:

- i. `InvalidCryptocurrencySearchException`
- ii. `InvalidUserCredentialsException`
- iii. `InvalidUserPasswordsException`
- iv. `NotEnoughAppResourcesException`
- v. `NotEnoughUserResourcesException`
- vi. `UserAlreadyExistsException`

## Frontend

За градење на кориснички интерфејси користев React, JavaScript билбиотеката.

Кодот кој што е искуцан за потребите на апликацијата се наоѓа во „/src“ пакетот кој што содржи внатре низа од пакети за подобра организација на кодот.

Основната фронтенд рута на апликацијата е „http://localhost:3000/“.

Апликацијата е составена од повеќе погледи:

### ***a) Поглед за регистрација на нови корисници - /register или „/“***

При самото пристапување на апликацијата, овој поглед е основен, односно прв кој што секој корисник ќе го види.

По реднерирање на овој поглед може да се види форма на која што новите корисници кои што сакаат да ја користат оваа апликација можат да креираат кориснички сметки. Притоа, на оваа форма потребно е да внесат податоци за нивното корисничко име, лозинка, да ја повторат лозинката, нивното име и презиме, податоци за кредитна картичка и на крајот дали корисникот ќе има обични или административни привилегии. Доколку некое од овие полиња во формата не е правилно пополнето или не е воопшто пополнето ќе се појави порака за грешка при обидот за креирање на сметката.

Исто така, најдоле, после формата има линк до погледот за најава доколку корисниците веќе имаат своја корисничка сметка можат да кликнат на тој линк кој што ќе направи редиректирање на рутата „/login“.

### ***b) Поглед за најава на веќе постојечки корисници - /login***

Доколку корисниците веќе имаат креденцијали за најава на апликацијата истите можат да ги искористат на оваа форма. Се што треба да направат корисниците е да пополнат формата која е составена од две полиња, корисничко име и лозинка.

Доколку некое од полињата не е правилно пополнето или не е воопшто пополнето ќе се прикаже грешка на екранот на корисникот, во спротивно, доколку се е како што треба ќе има редирекција на рутата „/home“.

### ***с) Поглед на почетната страница – „/home”***

На левата страна на екранот може да се види графикон со историски цени на основна криптовалута Bitcoin во USD во последниот час на интервали од секои пет минути.

Веднаш над графиконот има листа на сто криптовалути со кои што корисниците можат да прават трговија. Во секој момент можат да изберат нова криптовалута од листата и ќе биде рендериран графикон на историја за соодветната валута.

Вредностите на у-оската се претставуваат динамички за попрегледен приказ на графиконот.

Под графиконот може да се видат податоци за селектираната криптовалута, нејзиниот симбол и пазарната вредност. Притоа, при секое ново селектирање на нова криптовалута на графиконот, овие податоци веднаш се ажурираат.

Десно од графиконот може да се види секција на која што можат да се прават трансакции. Овој дел е составен од две форми кои што служат за купување и продавање на криптовалути. Двете форми се составени од име на криптовалутата со која што корисниците сакаат да направат купување или продавање, како и вредноста по која сакаат да биде извршена трговијата во USD. Доколку корисниците немаат доволно средства или пак некоја друга проверка од бекенд не дозволи да се заврши трансакцијата, ќе биде соодветно прикажана грешка на корисничкиот интерфејс.

### ***д) Поглед за додавање и подигнување на средства од апликацијата - „/balance”***

Преку овој поглед корисниците можат да прават додавање на расположливи средства на своите сметки како и подигнување на пари од апликацијата. Ова може да го направат преку две различни форми кои што имаат само едно поле, сума во USD за која што сакаат да се направи соодветната акција.

Доколку корисниците не задоволуваат некоја проверка од бекенд за да се заврши подигнувањето на посакуваната сума или додавање на нови расположливи средства за понатамошна трговија, ќе биде соодветно прикажана грешка на корисничкиот интерфејс.

Притоа, доколку корисниците подигнуваат пари од апликацијата ќе им биде генерирана фактура која што служи како потврда дека парите се иплатени. Оваа акција се прави преку приказ на PDF документ.

#### ***e) Поглед за прикажување на поседувани криптовалути од најавениот корисник - „/myCrypto“***

Евиденција на исходите на успешните купувања на криптовалути, како и одземање од веќе постоечките при предажба на истите можат да се следат преку овој поглед.

Најгоре на екранот може да се види кој корисник е најавен и колку расположливи средства поседува. Веднаш под ова може да се види табела со две колони, име на криптовалута и сумата која што корисникот ја има соодветно во криптовалутата.

#### ***f) Поглед за прикажување на сите направени трансакции од најавениот корисник – „/myTransactions“***

Сите купувања и продавања на криптовалути од најавениот корисник може да се видат на овој поглед кој што преку табеларен приказ дава податоци за истите. Табелата е составена од пет колони, името на криптовалутата која што била дел од трговијата, сумата во крипто, сумата од валутата во USD, вид на трансакцијата дали е купена или продадена валутата и времето на комплетирање на таа трансакција. Доколку најавениот корисник сеуште нема направено ниту една трансакција, на екранот е прикажана порака „Sorry, no transactions to show!“.

#### ***g) Поглед за прикажување на сите направени трансакции од сите корисници во апликацијата - „/allTransactions“***

Овој поглед е дозволен за пристап само за корисници со административни привилегии во апликацијата. Ги прикажува сите трансакции кои што се направени во апликацијата од сите корисници. Исто така и тука податоците се прикажани преку табеларен приказ кој што ги содржи истите колони како и кај погледот „/myTransactions“ со плус додатна колона на почетокот на табелата која што го означува бројот на трансакцијата.

Доколку нема направено трансакции на апликацијата или пак доколку корисникот нема административни привилегии за пристап ќе биде прикажана порака на екранот „Sorry, currently you don't have permissions for this action or there are no transactions!“.

Сите оснати можни рути кои што не се претходно споменати прават редиректирање на “/PageNotFound” кој што прикажува дека таа рута не постои и корисникот може преку линк да пристапи назад до „/register“.



Горе десно во навигацијата може да се види и копче „Log out” кој што прави редиректирање до страната „/login” на која ист или друг корисник може да се најави на страната. Притоа, со самото рендерирање на „/login” погледот се брише и JWT токенот на најавениот корисник од прелистувачот.

Во компонентата App.js се дефинирани сите претходно споменати рутирања кои што се достапни во апликацијата со имплементација на функции од библиотеката react-router-dom. Притоа, тука се назначени кои компоненти ќе бидат рендерирани при пристапување на некоја од рутите.

Притоа, за имплементација на овие погледи, кодот е организиран во повеќе пакети:

a) components package

Во овој пакет од апликацијата се наоѓа најголем дел од логиката во фронтенд апликацијата. Дефинирани се дванаесет компоненти кои што се распоредени да се рендерираат само по потреба при пристапување на соодветен поглед на корисничкиот интерфејс.

Компонентите кои што се дефинирани се следниве: AddMoneyInWallet, AllTransactions, Balance, BuyCrypto, CashChanges, Graph, HomePage, MyCrypto, MyTransactions, NavBarm NavBarElements, PageNotFound, SellCrypto и WithdrawMoney. Притоа, сите имаат екстензија .js соодветно после името.

За да се подобри изгледот на екранот на корисниците се користи Bootstrap[8] кој што ја прави апликацијата да изгледа добро на екрани со различни големини.

b) custom-axios package

Креирање на нова инстанца за пристапување на рути до бекенд делот преку библиотеката „axios”. Во оваа инстанца се дефираат базично URL со додавање на неопходни карактеристики на барањата кои стигаат до бекенд како JWT токен, контрола за пристап и вид на одговор кој што се чека.

c) images package

Тука се наоѓа сликата која што е лого на апликацијата.

#### d) login package

Овој пакет содржи една компонента која што е формата за логирање на нов корисник.

Кога се пристапува компонентата одма се брише JWT токенот долку има веќе поставено во складиштето на прелистувачот.

Користи соодветни состојби за корисничко име и лозинка, и при клик на копчето за најава истите податоци се препраќаат до бекенд преку пакетот за репозиториум.

#### e) register package

Формата за регистрација си има свој пакет во кој што се дефинирани состојби од полињата во формата. Кога корисникот точно ќе ги пополни полињата во формата се праќа соодветно HTTP барање до бекенд.

Исто така, тука е дефинирана рутата за навигација до „/login“ формата доколку корисникот има веќе корисничка сметка, или пак успешно е завршено барањето за регистрирање на нов корисник.

#### f) repository package

Овој пакет има една компонента која што ја користи веќе предефинираната рута од axios од пакетот „custom-axios“. Притоа, тука се дефинирани сите параметри, вид на барања и рути до кои што можеме да пристапиме кога праќаме податоци од фронтенд до бекенд и обратно.

Тука може да се видат сите параметри кои што се препреќаат од формите од фронтенд и имињата под кои што тие се дефинирани за да бидат прифатени од бекенд делот.

#### g) style package

За додавање на стилови на апликацијата, покрај користењето на Bootstrap, додаден е и style.css фајл.

## 4.Кориснички сценарија

Визуелно прикажување на сите сценарија кои што се очекувани и достапни на корисничкиот интерфејс се покриени во оваа секција.

### Welcome To Block Trader



#### Register Form

**Username**

**Password**

**Repeat Password**

**Firstname**

**Lastname**

**Credit Card**

**Role**

---

Have account? [Sign in here](#)

Слика 3. Пополнета форма за регистрација на нов корисник

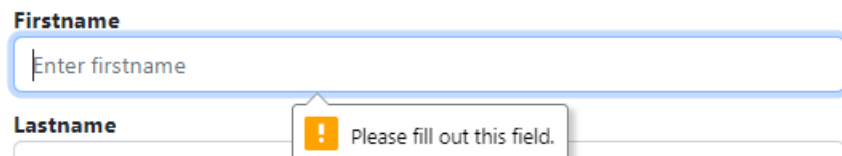
**Role**

Bad Request - Check the credentials, otherwise the username is already taken!

---

Have account? [Sign in here](#)

Слика 4. Порака при погрешно пополнување на полињата на формата за регистрација



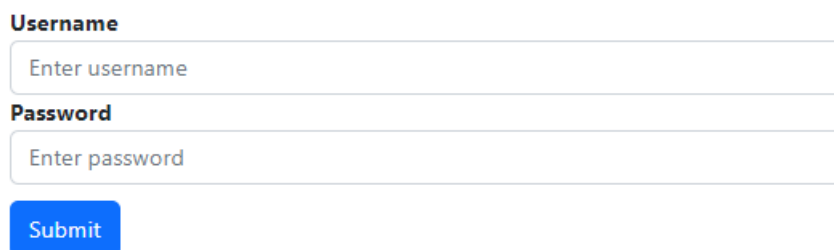
A registration form with two input fields. The first field is labeled 'Firstname' and contains the placeholder text 'Enter firstname'. The second field is labeled 'Lastname' and is empty. A yellow tooltip with an exclamation mark icon and the text 'Please fill out this field.' points to the 'Lastname' field.

**Firstname**  
Enter firstname

**Lastname**  
Please fill out this field.

Слика 5. Валидација на полињата за регистрација на нов корисник

## Login Form



A login form with two input fields. The first field is labeled 'Username' and contains the placeholder text 'Enter username'. The second field is labeled 'Password' and contains the placeholder text 'Enter password'. Below the fields is a blue 'Submit' button.

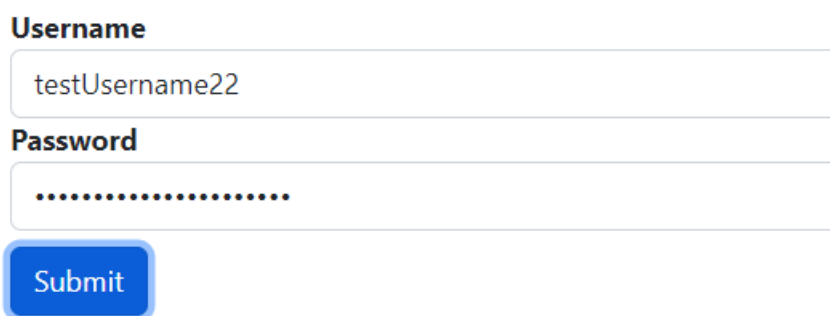
**Username**  
Enter username

**Password**  
Enter password

Submit

Слика 6. Форма за најава на корисници на апликацијата

## Login Form



A login form with two input fields. The first field is labeled 'Username' and contains the text 'testUsername22'. The second field is labeled 'Password' and contains a series of dots. Below the fields is a blue 'Submit' button.

**Username**  
testUsername22

**Password**  
.....

Submit

- Invalid User Credentials!

Слика 7. Порака при погрешни пополнување на полињата на формата за најава



Слика 8. Почетна страница



Слика 9. Селектирање на друга криптовалута за приказ на графиконот на историја

[My Transactions](#)
[All Transactions](#)
[Log out](#)

### Make Transaction:

Cryptocurrency Name:

Amount(USD):

[Buy Crypto](#)

Something went wrong, the transaction is not completed:  
Internal Server Error


Cryptocurrency Name:

Amount(USD):

[Sell Crypto](#)

Слика 10. Порука при неуспешна трансакција при купување на криптовалута

[←](#)
[→](#)
[↻](#)


[Withdraw/Deposit](#)
[My Cryptocurrencies](#)

### Deposit Available Resources:

Amount(USD):


[Deposit Cash](#)

### Withdraw Money:

Amount(USD):

[Withdraw Money](#)

Слика 11. Поглед за додавање и повлекување на расположливи средства

← → ↻ localhost:3000/myCrypto	
<div>  <div>Withdraw/Deposit</div> <div>My Cryptocurrencies</div> </div>	
<p>Current logged user - <b>test</b></p> <p>Users Available Resources (USD)- <b>5035 \$</b></p> <p>Cryptocurrencies owned by the given user:</p>	
Currency Name	Currency Amount USD
Ethereum	173
Bitcoin	39198

Слика 12. Поглед за прикажување на сите криптовалути поседувани од моментално најавен корисник

← → ↺

localhost:3000/myTransactions

BLOCK TRADER

Withdraw/Deposit

My Cryptocurrencies

My Transactions

All Transactions

Log out

Traded Crypto	Crypto Amount	Crypto USD	Transaction Type	Date
Bitcoin	0.3642488072729707	7000	Bought	2022-09-08T14:42:49.817934
Bitcoin	0.0861209417634373	2000	Bought	2022-08-06T15:24:01.493496
Bitcoin	0.008022725702336244	190	Bought	2022-07-31T10:37:04.322806
Bitcoin	0.008022725702336244	190	Sold	2022-07-31T10:36:26.487332
Bitcoin	0.0008453464698310848	20	Bought	2022-07-31T10:19:20.111614
Bitcoin	0.0005045753416305411	12	Sold	2022-07-29T19:22:07.154584
Bitcoin	0.0008409589027175685	20	Bought	2022-07-29T19:22:01.276768
Bitcoin	0.41935293570073157	10000	Bought	2022-07-28T22:19:09.750384
Bitcoin	0.00008396693400006089	2	Sold	2022-07-28T20:21:22.538572

Слика 13. Поглед за прикажување на сите направени трансакции од моментално најавен корисник

←

→

↺

📍 localhost:3000/allTransactions

Block Trader

Withdraw/Deposit

My Cryptocurrencies

My Transactions

All Transactions

Log out

Transaction Number	Traded Crypto	Crypto Amount	Crypto USD	Transaction Type	Date
78	Bitcoin	0.3642488072729707	7000	Bought	2022-09-08T14:42:49.817934
77	Bitcoin	0.018685721125332923	400	Sold	2022-08-26T09:28:00.525026
76	Bitcoin	0.0023365797454824497	50	Bought	2022-08-26T09:27:18.195924
75	Bitcoin	0.046731594909648996	1000	Bought	2022-08-26T09:26:47.981297
74	Bitcoin	0.004631067388179892	100	Bought	2022-08-25T22:43:31.940045
73	Bitcoin	0.01389591426099122	300	Sold	2022-08-25T22:40:41.418903
72	Bitcoin	0.02779182852198244	600	Bought	2022-08-25T22:39:55.939211
71	Bitcoin	0.2207417296249863	4800	Bought	2022-08-25T16:45:09.562476
70	Bitcoin	0.004602065429577305	100	Sold	2022-08-25T16:40:26.675422
69	Bitcoin	0.013807671837406173	300	Bought	2022-08-25T16:39:07.230996
68	Bitcoin	0.00013830441009762305	3	Bought	2022-08-25T13:29:59.95042

Слика 14. Поглед за прикажување на сите направени трансакции од сите корисници

## INVOICE PAYMENT CONFIRMATION

BLOCK TRADER  
59 West 46th Street  
New York City, NY 10036.  
+585 992 019932  
payments@blocktrader.com

**Paid to:**  
Gabriel Dimitrievski  
Known as: "gabrieldim"  
+389 78 783 294

By submitting this invoice, we confirm that requested amount of 1000 USD has been paid to the user's account: 9999333311112222

According to our policy for the use of this service, if there are any irregularities, they will be accepted only in the next three working days, otherwise it is considered that the funds have successfully reached the user's account.

Thank you for using our services. We hope you had a great experience and an even better trade.

Date this message was sent: 2022-09-11

Слика 15. Пример генериран PDF документ кој служи како потврда за исплатена фактура



## 5.Заклучок

Во рамките на овој дипломски труд успешно беше изработена веб апликација за трговија со криптовалути. Оваа апликација им овозможува на корисниците да купуваат и продаваат сто криптовалути кои што се земаат од надворешно API.

Исто така, сметам дека ваков тип на апликација има голем потенцијал за надградба и успех во иднина од причина што според се изгледа дека со текот на времето дигиталните валути стануваат се повеќе прифатливи од многу држави на глобални рамки.

Оваа дипломска работа претставува заклучен труд на моето досегашно образование и соодветно ги прикажува моите знаења и вештини во повеќе области: моделирање на бази на податоци, развој на веб апликации, дизајнирање на кориснички интерфејси, менаџирање на проекти преку верзии на изворен код итн.

## 6. Референци

- [1] JWT - <https://www.javainuse.com/spring/boot-jwt>
- [2] GitHub Project Repository - <https://github.com/gabrieldim/Crypto-Trading-Graduate-Thesis>
- [3] RestTemplate - <https://www.baeldung.com/rest-template>
- [4] CoinMarketCap - <https://coinmarketcap.com/api/documentation/v1/>
- [5] Spring Boot Scheduling - <https://www.baeldung.com/spring-scheduled-tasks>
- [6] @RestController - <https://www.baeldung.com/spring-controller-vs-restcontroller>
- [7] PDF Generator Tutorial - <https://www.youtube.com/watch?v=S7udzd3xjGQ&t=33s>
- [8] Bootstrap Grid System - <https://getbootstrap.com/docs/4.0/layout/grid/>