1.Вовед

Парични средства се користат на дневна основа илијадници 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 и 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 немаат парични граници до кои што можат да се прават трансакции, имаме пристап до овие системи 24часа 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 кој што има паметен телефон и интернет врска да биде дел од глобалната економија.

Криптовалутите се веќе неколку години актуелна тема на разговор помеѓу луѓето, како 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 еден ваков огромен систем не е директно контролиран 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 овие блоковски вериги.

После сите претходно наведени работи оваа темата самата по себе ми е доволно фасцинантна, и така дојдов до идеја за градење на еден ваков систем со кој што ќе

овозможи на корисници да прават купопродажби на криптовалути само преку неколку едноставни кликови на кориснички интерфејс.

2.Користени Технологии

За развојот на оваа апликација користев повеќе технологии кои што се моментално многу актуелни и се користат на секојдневно во многу ИТ компании.
За основните фукнционалности на апликацијата т.е за бекенд делот користев Spring Boot во комбинација со програмскиот јазик Јава.

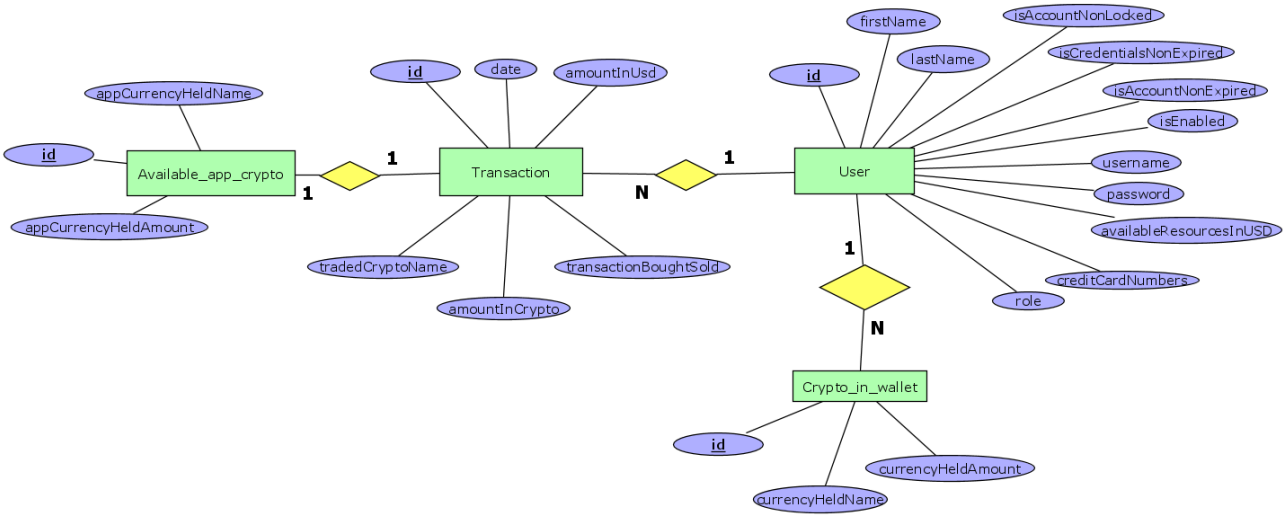
-spring -react -postman -maven -git л - resttemplate -pdf -postgres -dbeaver -bootstrap

3.Архитектура

- <u>База на податоци</u>

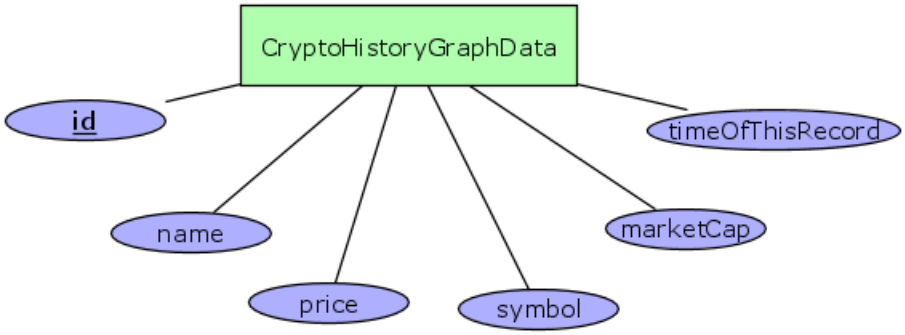
Релациониот дијаграм на базата на податоци е во два дела, првиот дел е составен од табели кои што се меѓусебно поврзани и имаат силна зависност меѓу нив, додека пак другиот дел е составен од една самостојна база која има цел да ги чува вредностите на криптовалутите.
Првиот дел од базата е составен од четири табели. Првата табела „Available_app_crypto“ е составена од три параметри, уникатен идентификатор кој е примарен клуч за таа табела, има на криптовалутата и соодветна вредност на поседување на истата. Оваа табела ги чува сите криптовалути кои што се достапни на страна на 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 еден наспрема еден релација со втората табела „Transaction“. Оваа табела содржи параметри 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 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 параметри, повторно имаме уникатен идентификатор има улога на примарен клуч, име и презиме на корисникот, корисничко име, лозинка која што е хаширана, достапни ресурси во USD со кои што може да купува криптовалути од апликацијата, кредитна картичка се со цел кога ќе посака да повлече пари да се пратат на соодветна сметка, параметар кој означува дали моментално логираниот корисник е администратор или пак обичен корисник, како и четири параметри кои што се важни од апспект 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 параметри 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 валути кој е прикажан на корисничкиот интерфејс.
Табелата е составена од шест параметри, идентификатор кој е примарен клуч, целосно име на валутите, време кога колоните се додадени во базата, цената која што ја имаат во тој момент, симболот позади кој што стојат валутите и вкупната вредност на сите криптовалути кои се ископани.
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 слоевита архитектура на бекенд делот, притоа се состои од седум пакети во кои е организиран целиот бекенд код.
Во составот на пакетите се основите четири пакети: модел, репозиториум, сервисна логика и контролери или веб слој, и три останати кои што се прават посебни логики за потребите на апликациајта и се организирани во посебни пакети.

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 логига 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 кажуваме за името на класите како ќе биодат 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 фунцкии кои што доаѓаат од анотацијат @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” кој има 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 шест класи кои што содржат по еден или неколку атрибути што ги примаат контролерите за да се воспостави успешно препраќање на податоците.

i. User класа

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

За сите редици за колоната „username” кој што ќе бидат додавани во базата, овој параметар мора да биде уникатен, 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.

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

ii. AvaliableAppCrypto 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 параметри од базата на податоци, имам имлпементирано и два дополнителни објект за апликацијата да фунционира на посакуваниот начин.

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

Другиот објект кој што е дел од оваа класа е инстанца од класата AvailableAppCrypto кој што е соодветно мапиран со @OneToOne анотација и ги чува податоците за криптовалутата 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 објект од класата User кој што означува состојбата криптовалутите кои што корисниците ги поседуваат. Исто така, има само еден мануелно дефиниран конструктор со име на криптовалути и во колкава вредност таа валута е поседувана од даден корисник.

v. CryptoHistoryGraphData класа

За разлика од останатите 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 кои што означуваат табели и анотациите за идентификатор.

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

интерфејс претставува 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 наставна „Repository”.

i. UserRepository

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

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 фунцика е да го врати објектот, но да прими параметар кој што означува име на криптовалута и да пребарува според истиот.

iii. TransactionRepository

За разлика од претходно, тука имаме две фунцкии 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 фунцкија ги враќа сите направени трансакции за даден корисник кој го прима како параметар на фунцкијата 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

Составено од две дефинирани функции.

Листа од објекти од класата CryptoHistoryGraphData сортирани според опаѓачки редпослед. Оваа функција служи за враќање на JSON објект до фронтенд со податоци кои што се потребни за да се прикаже соодветниот градикон.
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 фунцкии 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 кодот 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 фунцкија креира и запишува нов дојден корисник на апликациајта. 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 фунција која што прима параметар кој што означува депозит кој што го прави корисникот за да додаде нови расположливи средства со кои што може да прави понатамошна трговија. Го зема моментално логираниот корисник преку класата SecurityContextHolder, ги зема веќе постоечките разположливи 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 валута 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 логираниот 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 подоле.

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, 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, се намалува расположливата вредност 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 на 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 проодаде. Доколку не, се фрла исклучок NotEnoughUserResourcesException со порака „You don't have enough from this cryptocurrency to sell!“

Се зачувува моменталната цела на валутата која што корисникот сака да ја продаде.

Следна акција која што се врши е додавање на расположливи средства на страна на корисникот во USD.

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

6. withdrawAmount

Оваа функција прима само еден параметар кој што ја носи вредноста на пари која што сака корисникот да ја повлече од апликацијата.

Се зема логираниот 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 логираниот корисник во апликациајта, се земаат расположливите средства кои што корисникот го поседува и се враќаат назад до функцијата од каде што се повикува.

8. getCurrentRealTimePrice

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

ii. TransactionServiceImplementation

Во оваа класа може да најдеме повторна имплементација на концептот Dependency Injection од претходниот архитектурен слој со иницијалицирање 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 логираниот корисник да е администратор, во спротивно враќа празна листа. Втората функција е задолжена за враќање на сите трансакции кои што се направени од логираниот 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 фунцкии 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 запаиси 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 UserService интерфејсот кој што беше претходно објаснет.

v. CronJobServiceImpl

Овозможување на повик до надворешен сервер кој што враќа податоци за сто криптовалути преку RestTemplate. Во header делот од барањето е ставено името според документацијата од CoinMarketCap, “X-CMC_PRO_API_KEY” соодветно со вредноста на мојот генериран клуч на нивната платформа "d547fd3a-f9a6-4fdd-9dd0-71774b4cdcd5".

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 JavaScript Object Notation познат како JSON.

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

Првата анотација е @RestController 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 вторите две анотации има единствено мала разлика во контролерите за логирање 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.

Овој контролер има имплементирано 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 анотациајта @RequestParam кој што е симболот кој што корисникот го избрал за да добие графикон од таа криптовалута. Се прави повик до фунцкијата од класата од сервисната логика CryptoHistoryGraphDataService која што ја враќа листата на објектите кои потоа се пуштаат низ Java Stream кој прави филитритање според симболот на криптовалутата која корисникот ја избрал и се прави лимитирање на тринаесет објекти кои што се праќаат назат. Ограничувањето на трианесет објекти е поради приказот 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 го рутата „/crypto" која што е објаснета погоре.

ii. LoginRestController

Станува збор за единствен контролер кој што има ограничување за пристапување на реруси 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 контролер 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 притоа има за цел да логира корисник доколку ги исполнува условите 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 @PostMapping се соодветно се сетираат headerKey и headerValue параметри кои што се потребни за генерирање на PDF кој што го прави функциајта export од класата PDFGeneratorServiceImpl. Притоа, функцијата за рутата „/generatePDF“ прима два параметри HttpServletResponse и сумата која што корисникот сака да ја повлече од апликацијата.

iv. RegisterController

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

Прима @RequestBody објект од класата RegisterUserDto кој што прави 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. Методата 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 и двете 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 логираниот корисник. Рутата на оваа функција е „/loggedUserTransactions”.

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

vi. UserController

Како што можевме да видиме, најмногу бизнис логика имаме во делот од UserServiceImplementation класата. Па според тоа, тука имаме и најмногу фунцкии 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 со мапирање @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 сумите кои што се за правење на промени во базата.

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 фунцкија прави повик до функцијата addUSDMoneyInWallet од интерфејсот UserService.

BuyCryptoDto е класа задолжена за прифаќање на вредност за која валута и во која количина корисникот сака да ја купи и притоа прави повик до функцијата buyCrypto од UserService интерфејсот.

Последната фунцкија 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 кои што од оваа класа кои што имаат мапирање @GetMapping кои што не примаат ниту еден параметар.

Функцијата со рута „/getLoggedUserCryptocurrencies" ги враќа сите криптовалути кои што логираниот 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 логираниот корисник и истиот се враќа назад до корисничкиот интерфејс.

Како последна рута од оваа класа е „/availableResources" која што ја враќа вредноста на достапни ресурси 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 како информациите доаѓаат од JSON-от. Од JSON објектот 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(“”) со некоја вредност во наводниците која што го означува името како дадена променлива дефинирана во JSON објектот, а ова на нас 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 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 логирање 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 CustumUsernamePasswordAuthenticationProvider 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 CustumUsernamePasswordAuthenticationProvider.

ii. JWTWebSecurityConfig

Логиката на имплементација е слична како и во WebSecurityConfig класата, но тука се користат филтри 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 секој пат да се препраќа JWT токенот за да се автентицира корисникот на страна на серверот и како да му се даде до знаење на серверот дека станува збор за валидно HTTP 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 конечно дефинирани променливи кои што нема да се менуваат и се потребни за JWT автентикацијата.

iv. CustumUsernamePasswordAuthenticationProvider

Композирана класа од две фунции.
Првата функција е supports која што само проверува дали параметарот кој што е пример е од класата UsernamePasswordAuthenticationToken.
Класата authenticate со 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 се наоѓаат во параметарот кој што ја прима оваа функција. Се прават проверки дали овие полиња се правилно пополнети, дали хаширaните 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.

Апликацијата при секое барање треба да користи JWT токен за комуникациајта да биде успешна.
Преку форма од кориснички интерфејс се праќа POST барање до серверот. Серверот доколку открие дека корисникот е валиден, односно неговите креденцијали 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 мора да биде оваа информација испратена која што 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

-slika od frontend arhitektura, I idenje na view po view da se opishe shto se deshava.

4.Кориснички сценарија

-kako e napravena aplikacijata(da se pokazhe so sliki – toa so go pravevme na demoto)

5.Заклучок

deka spored kraken sum idel za da sfatam kako idat procsite.

Iako ima turbulencii, sepak e mnogu atraktivna tema.

6. Референци