Спецификација на Софтверски Барања (SRS)

Проект: CryptoInfo

Тим:

- Димитар Арсов
- Филип Гавриловски
- Андреј Ристиќ

1. Вовед

1.1 Цел на документот

Целта на овој документ е детално да ги дефинира барањата за развој на веб апликацијата **CryptoInfo**. Овој документ служи како водич за развојниот тим и засегнатите страни, опишувајќи ги функционалните и нефункционалните карактеристики, архитектурата на системот, корисничките сценарија и очекуваните резултати.

1.2 Опсег на проектот

CryptoInfo е веб апликација дизајнирана за длабинска анализа на пазарот на криптовалути. Системот ќе овозможи собирање, обработка, складирање и визуелизација на историски податоци за 1000 највредни криптовалути во моментот.
Главниот фокус е надминување на ограничувањата на бесплатните јавни API сервиси преку имплементација на сопствен ETL (Extract, Transform, Load) процес користејќи податоци од Yahoo Finance. Апликацијата ќе обезбеди историски податоци во опсег од најмалку 10 години (каде што е можно), овозможувајќи техничка анализа и следење на трендови.

1.3 Дефиниции и кратенки

- **OHLCV:** Open, High, Low, Close, Volume (Основни параметри за цената на акциите/криптовалутите).
- **ETL:** Extract, Transform, Load (Процес на извлекување, трансформација и вчитување податоци).
- **Scraping:** Автоматско преземање податоци од веб-страници.
- **SRS:** Software Requirements Specification.
- **Frontend:** Кориснички интерфејс на апликацијата (на пр., веб-страница или мобилна апликација) со кој директно комуницира крајниот корисник (во овој случај, React.js апликацијата).
- **Endpoint:** Специфична URL адреса на која може да се пристапи преку API (на пр., `/api/v1/history/eth`). Тоа е точката на комуникација каде што Backend сервисот (Spring Boot) прима барања и испраќа податоци назад до Frontend-от.
- **API** (Application Programming Interface): збир на правила, протоколи и алатки кои овозможуваат различни софтверски апликации да комуницираат меѓусебе. API-то служи како посредник (интерфејс) помеѓу два софтверски системи.

2. Описен Наратив на Системот

Во денешниот свет на дигитални финансии, пристапот до квалитетни историски податоци е често скап или ограничен. Инвеститорите и аналитичарите кои не сакаат да плаќаат високи месечни претплати за премиум сервиси (како платени верзии на CoinGecko) се соочуваат со проблем: бесплатните алатки нудат само кратка историја (на пр. 1 година) или имаат строги ограничувања на бројот на повици.

CryptoInfo го решава овој проблем преку робустен систем кој симулира работа на професионален агрегатор на податоци. Системот работи во три главни фази:

1. **Фаза на прибирање:** Пајтон скрипти периодично ги собираат (scrape) податоците за топ 1000 валути според пазарна капитализација од Yahoo Finance. За секоја од овие валути, системот користи yfinance библиотеката за да ги повлече деталните историски податоци. Овие податоци се чистат, се проверуваат за конзистентност и се складираат во релациона база на податоци (PostgreSQL).

2. **Фаза на Обработка и Сервисирање (Backend Service)**
   - **Алатка: Spring Boot** (Java).
   - **Процес:** Оваа фаза служи како централен систем и API слој на апликацијата.
   - **Акција:** Spring Boot апликацијата:
     - Создава **RESTful API ендпоинти** за пристап до податоците.
     - Комуницира со **PostgreSQL базата на податоци** за ефикасно пребарување и филтрирање на историските податоци (на пример, по симбол, датумски опсег, или временска рамка).
     - Ја обработува и структурира суровата историја на податоци во формат погоден за брзо прикажување на графикони (пр. JSON објекти за свеќници).
     - Ги имплементира сите **бизнис логики** и **безбедносни механизми** на системот.
3. **Фаза на презентација (Frontend):** Корисникот пристапува до модерна веб апликација (React.js). Таму, тој може да пребарува криптовалути, да гледа интерактивни графикони кои прикажуваат историја од 10 години, и да врши техничка анализа користејќи ги податоците што системот претходно ги собрал и структурирал.

Ова овозможува демократизација на финансиските податоци, нудејќи им на студентите, хоби-трејдерите и истражувачите алатка која е бесплатна, а моќна.

3. Кориснички Персони

За подобро разбирање на барањата, дефиниравме три клучни кориснички персони:

Персона 1: Марко - Дневен Трговец (Day Trader)

- **Опис:** 28 години, работи како фриленс програмер и активно тргува со криптовалути во слободно време.
- **Цели:** Сака брзо да ги види дневните промени (High/Low) и волуменот на тргување за популарните валути.
- **Проблеми:** Фрустриран е што голем дел од бесплатните сајтови му бараат пари за да види историја постара од 12 месеци за да ги спореди моменталните трендови со претходните циклуси (на пр. 2017 или 2021 година).
- **Потреба од системот:** Прецизни графикони и можност за филтрирање на валути со

голем волумен.

Персона 2: Ана - Финансиски Аналитичар (Long-term Investor)

- **Опис:** 35 години, работи во инвестициски фонд, но истражува крипто за лично портфолио.
- **Цели:** Бара долгорочни трендови. Не ја интересира цената од минута во минута, туку дневната цена на затворање (Close) во последните 5-10 години.
- **Проблеми:** Податоците од различни извори често се неконзистентни или недостасуваат денови.
- **Потреба од системот:** Сигурност дека податоците се целосни (без дупки во датумите) и можност за прегледност на топ 1000 листата за да открие нови потенцијални проекти.

Персона 3: Иван - Data Science студент

- **Опис:** 21 години, студент на ФИНКИ.
- **Цели:** Сака да прави предвидувања на цени користејќи машинско учење.
- **Проблеми:** Не може да најде чист dataset кој е бесплатен и ажуриран дневно.
- **Потреба од системот:** Пристап до чисти OHLCV податоци кои се добро структурирани во базата, за да може да ги анализира трендовите.

4. Кориснички Сценарија

А. Сценарија за комуникација со корисникот

Сценарио: Преглед на историски циклуси

Актери: Марко (Корисник)

1. Марко ја отвора веб апликацијата CryptoInfo.
2. На почетната страна ја гледа табелата со топ 1000 валути.
3. Во пребарувачот внесува "Ethereum".
4. Системот ја прикажува страната за Ethereum со интерактивен графикон.
5. Марко избира временски период "Max" (максимален).

6. Апликацијата моментално ги вчитува податоците од 2015 година до денес.
7. Марко зумира на периодот од 2017 година за да го спореди тогашниот пораст со денешниот.

Б. Системски/административни сценарија

Сценарио: Ажурирање на базата (Системско сценарио)

Актери: Системски Администратор / Автоматски скрипти

1. Поминало 24 часа од последното ажурирање.
2. Автоматскиот "Scheduler" ја активира Python скриптата.
3. **Филтер 1:** Скриптата прави *scraping* на Yahoo Finance и ја зема новата листа на Топ 1000 валути.
4. **Филтер 2:** Системот детектира кои валути се нови, а кои веќе постојат.
5. За постоечките валути, системот го бара последниот датум во базата (на пр. вчера) и преку yfinance ги презема податоците само за денешниот ден.
6. **Филтер 3:** Податоците се чистат и се запишуваат во PostgreSQL.
7. Системот генерира лог дека процесот е успешно завршен.

5. Функциски барања (Functional Requirements - FR)

Овие барања опишуваат што системот мора да прави.

Група 1: Прибирање и обработка на податоци

- **FR-01 (Scraping на листа):** Системот мора да биде способен да ја преземе листата на моментално активни топ 1000 криптовалути според пазарна капитализација од Yahoo Finance.
- **FR-02 (Детално преземање):** За секоја идентификувана валута, системот мора да преземе историски податоци користејќи ја библиотеката yfinance.
    - Податоците мора да вклучуваат: Date, Open, High, Low, Close, Volume.

- **FR-03 (Инкрементално ажурирање):** Системот мора да проверува кој е последниот зачуван датум за дадена валута во базата и да ги преземе само податоците што недостасуваат (delta load), наместо да ја презема целата историја секој пат.
- **FR-04 (Филтрирање на невалидни податоци):** Системот мора да ги отфрли валутите кои немаат доволно историски податоци или чии податоци се корумпирани (null вредности за некоја колона).

Група 2: Функционални барања за Spring Boot API сервис (Backend Service)

- **FR-05 (API за пребарување на валути):** Системот мора да обезбеди RESTful API endpoint кој му овозможува на frontend-от да пребарува и да ја прикаже листата на **активно поддржани** криптовалути во базата (вклучувајќи ги симболот и името).
- **FR-06 (API за повлекување историја):** Системот мора да обезбеди RESTful API endpoint за повлекување на историските податоци за специфична валута. Ендпоинтот мора да поддржува филтрирање по временски опсег (на пр. од 2017-01-01 до денес) и интервал (на пр. дневно, неделно).
- **FR-07 (API за најнови податоци):** Системот мора да обезбеди API endpoint кој ги враќа најновите податоци (последен зачуван датум) за дадена валута, со цел да се прикаже моменталната цена.

Група 3: Складирање (Database)

- **FR-08 (Складирање):** Системот мора да ги чува податоците во релациона база (PostgreSQL).
- **FR-09 (Релации):** Мора да постои врска помеѓу табелата со валути (шифра, име) и табелата со историски податоци (информации по датум).

Група 4: Кориснички интерфејс и визуелизација (Frontend)

- **FR-10 (Листа на валути):** Корисникот мора да може да види пагинирана табела со 1000 криптовалути, која ги прикажува основните информации (име, симбол, моментална цена).
- **FR-11 (Пребарување):** Корисникот мора да има можност да пребарува

криптовалута по нејзиното име или симбол (на пр. BTC, Bitcoin).

- **FR-12 (Визуелизација):** При избор на валута, системот мора да прикаже графикон (Line chart или Candlestick chart) кој ги визуелизира OHLC податоците.
- **FR-13 (Временски филтри):** Корисникот мора да може да го менува временскиот опсег на графиконот (1 недела, 1 месец, 1 година, 5 години, Max).
- **FR-14 (Филтрирање на табела):** Корисникот треба да може да ја сортира главната табела според пазарна капитализација или име.

6. Нефункциски Барања (Non-Functional Requirements - NFR)

Овие барања го опишуваат квалитетот на системот и ограничувањата.

Перформанси

- **NFR-01 (Време на одзив):** Вчитувањето на графиконот за избрана валута не смее да трае подолго од 2 секунди при стандардна интернет конекција.
- **NFR-02 (Ефикасност на ETL):** Дневното ажурирање на базата за сите 1000 валути треба да се изврши во "off-peak" време и да не трае подолго од 1 час.

Доверливост и Точност

- **NFR-03 (Интегритет на податоци):** Податоците мора да бидат прецизни (floating point precision) бидејќи се работи за финансиски податоци.
- **NFR-04 (Отпорност на грешки):** Доколку Yahoo Finance API е недостапен привремено, системот не смее да "падне", туку треба да го логира проблемот и да се обиде повторно подоцна.

Одржливост и Скалабилност

- **NFR-05 (Модуларност):** Архитектурата мора да биде Pipe and Filter, што ќе овозможи лесно додавање на нови филтри (на пр. филтер за технички индикатори како Moving Average) без промена на целиот систем.
- **NFR-06 (Историски капацитет):** Базата на податоци мора да биде дизајнирана да поддржи милиони записи (1000 валути * 365 дена * 10 години = ~3.6 милиони редови) без драстично намалување на перформансите.

Корисничко искуство (UX)

- **NFR-07 (Интуитивност):** Интерфејсот треба да биде чист и разбирлив, без потреба од упатство за употреба.

7. Архитектура на Системот

Системот е дизајниран како Layered Architecture со посебен Data Pipeline.

7.1 Data Pipeline (Python - Pipe and Filter)

Ова е заднински процес кој ја полни базата.

- Филтер 1:
    - *Акција:* Scraping и HTML парсирање.
    - *Излез:* Листа на тикери (симболи) за Топ 1000 валути, филтрирани од шум.
- Филтер 2:
    - *Акција:* Проверка на updated_at во базата. Паралелно повикување на yfinance кај досега невидени валути.
    - *Логика:* При иницијално полнење -> земи сè. При последователни полнења -> идентификувај нови валути и зачувај ги нивните историски податоци во помошна датотека.
    - *Излез:* Листа на симболи со нивните датуми на ажурирање.
- Филтер 3:
    - *Акција:* Преземање на missing data (историски податоци за постоечки валути од датумот на ажурирање до денес) и додавање во постоечката помошна датотека.
    - *Излез:* Листа на симболи со нови датуми на ажурирање.
- Филтер 4:
    - *Акција:* Запишување на симболите од проследената листа на симболи и нивните историски податоци од помошната датотека во база на податоци.

7.2 Application Layer (Spring Boot & React)

Ова е делот со кој комуницира корисникот.

1. **Database Layer (PostgreSQL)**: Централно складиште каде Pipeline-от запишува, а Spring Boot чита.
2. **Backend Layer (Spring Boot)**:
    - Служи како "Presentation Logic" за податоците.
    - Ги мапира редовите од базата во JSON објекти.
    - Овозможува REST endpoints за Frontend-от.
3. **Presentation Layer (React)**:
    - Single Page Application (SPA).
    - Чита директно од Spring Boot API-то.
    - Користи библиотеки за исцртување на графикони врз основа на примените JSON податоци.