Дизајн и архитектура на софтвер
Домашна работа 1

Целта на овој проект е развој на веб апликација за анализа на крипто берзата, која ќе овозможи преглед, споредба и анализа на историски податоци за највредните 1000 активни криптовалути. Апликацијата ќе работи со податоци на дневно ниво за период од најмалку последните 10 години, со фокус на цена, волумен, пазарна капитализација, ликвидност и други релевантни индикатори.

Во првата фаза (првата домашна задача) фокусот е на обработка и трансформација на податоци користејќи ја **pipe-and-filter архитектурата**, односно креирање на цевка од независни филтри кои последователно ги преземаат и трансформираат податоците од надворешни извори во структуриран формат погоден за понатамошна анализа и прикажување.

Технологии што ќе се користат се:

- **Python** како главен програмски јазик поради неговата едноставност и богатиот екосистем за обработка на податоци, автоматизација и работа со API.
- **Requests** за преземање податоци од крипто API извори.
- (по потреба) **Spring Boot** за понатамошно развивање на веб слојот и REST API
- **SQLite** како лесна, вградена релациона база на податоци.
- **Pandas** за трансформација, чистење на податоци ако е потребно.

За преземање на податоците ќе се истражат повеќе извори: крипто берзи како Binance, Coinbase, Kraken, но и сервиси како CoinGecko, CoinMarketCap, CryptoCompare, кои нудат јавни и бесплатно достапни API за пазарни податоци. Иако податоците потекнуваат од меѓународни берзи, за потребите на проектот ќе се користи страната CoinGecko како главен извор, затоа што:

- обезбедува јавно достапно API без потреба од API клуч за основни податоци,
- овозможува добивање листа на криптовалути како „топ 1000“,
- нуди историски податоци на дневно ниво за повеќегодишки периоди.

Податоците што ќе се обработуваат ќе вклучуваат дневни **OHLCV** вредности (Open, High, Low, Close, Volume), тековна цена, 24-часовен волумен, пазарна капитализација и евентуални дополнителни индикатори. Овие податоци ќе поминуваат низ низа филтри:

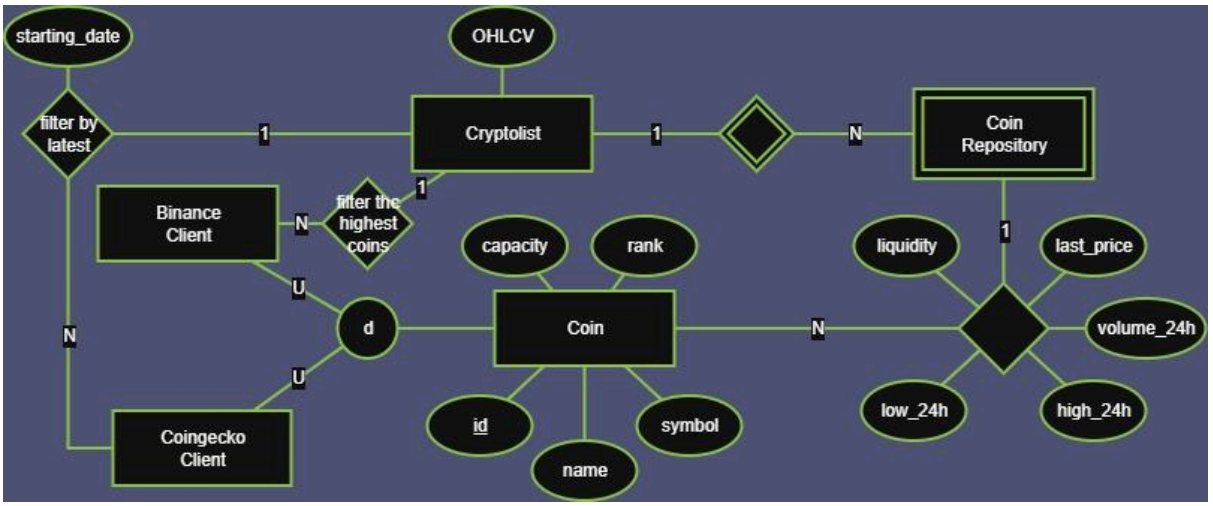
- филтри за чистење и валидација на листата на валути,
- филтри за проверка на последниот датум за кој има податоци во базата,
- филтри за преземање на податоците што недостасуваат и нивно нормализирање во конзистентен формат.

Складирањето на податоците ќе се врши во **SQLite база**, во специјално дизајнирани табели за криптовалути и нивните историски записи. На тој начин, во следните фази од проектот ќе може да се изгради интерфејс за визуелизација (графици, табели, филтри) и анализа на истите.

Очекуваните резултати и придобивки од проектот се:

- автоматизирана и скалабилна инфраструктура за собирање и ажурирање на крипто податоци,
- конзистентна и нормализирана база со историски податоци поголеми од 10 години за голем број криптовалути,
- можност за изградба на аналитички и визуелизациски функционалности врз истиот dataset,
- употреба на **pipe-and-filter архитектура** за реални податоци.

Дијаграм за како е поврзана апликацијата



Функциски барања

1. **Автоматско преземање на листа на топ 1000 криптовалути**
   Системот треба автоматски да ја презема и ажурира листата на 1000 активни криптовалути од избраниот извор на податоци.

2. **Филтрирање на невалидни или неподобни криптовалути**
   Системот треба да идентификува и исклучи коини со ниска ликвидност, дупликати по симбол или ID и потенцијално неупотребливи валути.

3. **Преземање на дневни историски податоци (OHLCV)**
   За секоја валута во листата, системот треба да презема дневни OHLCV податоци и дополнителни 24-часовни метрики (последна цена, 24h волумен, 24h high/low)

за период од најмалку 10 години, или максимумот што го поддржува изворот.

4. **Проверка на последен зачуван датум во базата**
   Пред преземање нови податоци, системот треба да провери до кој датум веќе постојат податоци за дадена криптовалута во базата и да пресмета период за кој недостасуваат записи.

5. **Надополнување на податоци**
   Кога ќе се преземаат нови податоци, системот треба да ги ажурира записите во базата така што ќе се избегнат дупликати и ќе се задржи целосна историја за секоја валута.

6. **Pipe-and-filter обработка**
   Податоците треба да поминуваат низ неколку филтри односно компоненти, каде што излезот на еден филтер ќе биде влез за следниот (тоа значи, ако имаме филтер за избор на валути треба да оди во филтер за проверка на последен датум, па следно е филтер за симнување на недостасувачките записи и на крај филтер за нормализација и складирање).

7. **Мерење на времето на извршување**
   Системот треба да мери колку време е потребно за целосно полнење на празна база со податоците за избраната крипто берза и да ги зачува тие мерења за споредба и оптимизација.

Нефункциски барања

1. **Перформанси и ефикасност**
   Ќе се потрудиме системот да ги минимизира бројот и времетраењето на повиците кон надворешните API сервиси и да го оптимизира протокот на податоци.

2. **Скалабилност и проширливост**
   Архитектурата треба да биде дизајнирана така што подоцна може да се додадат нови извори на податоци или нови филтри без да дојде до промени во постоечкиот код, всушност со ова правиме примена на Interceptor pattern-от.

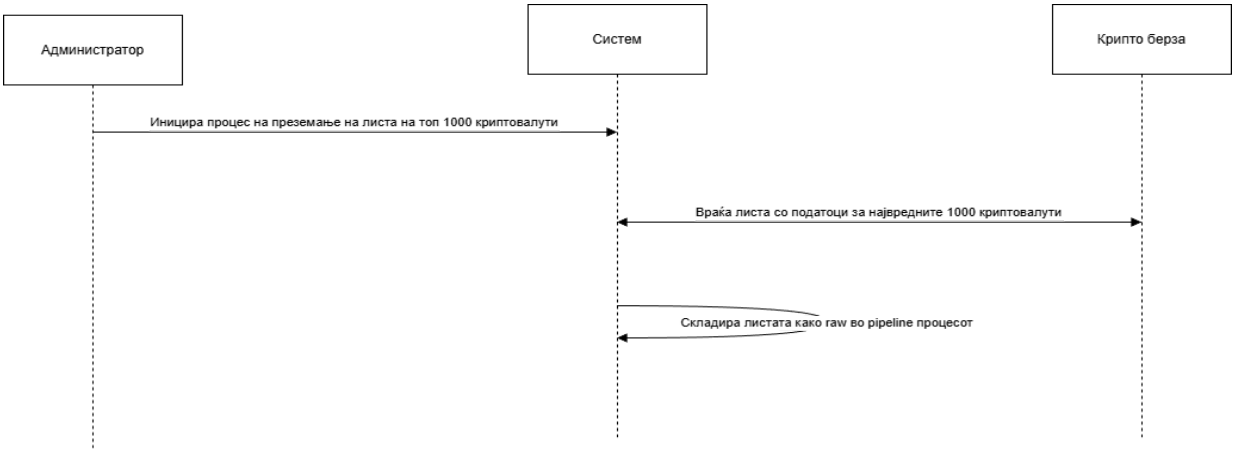
3. **Толеранција на грешки**
   При појава на грешка од API (HTTP грешки, timeout, невалиден одговор) или прекин на интернет конекција, системот не смее „да се сруши“, туку треба да:
   - пријави грешка во лог,
   - проба повторно,
   - безбедно да го прекине процесот, задржувајќи ги веќе обработените податоци.

4. **Прифатлив формат и интегритет на податоците**
   Податоците што се внесуваат во базата треба да бидат во формат кој се прифаќа односно тоа значи да имаат стандардизирани датуми, исти мерни единици, ист формат на симболи како и да се почитуваат интегритетни ограничувања (примарни клучеви не смеат да бидат null, уникатни записи по датум и валута).

5. **Одржливост и читливост на кодот**
   Кодот ќе биде структуриран по јасни архитектурни слоеви во Python пакети. Ова овозможува лесно одржување, повторна употребливост и едноставно проширување на проектот во следните фази.

Кориснички сценарија

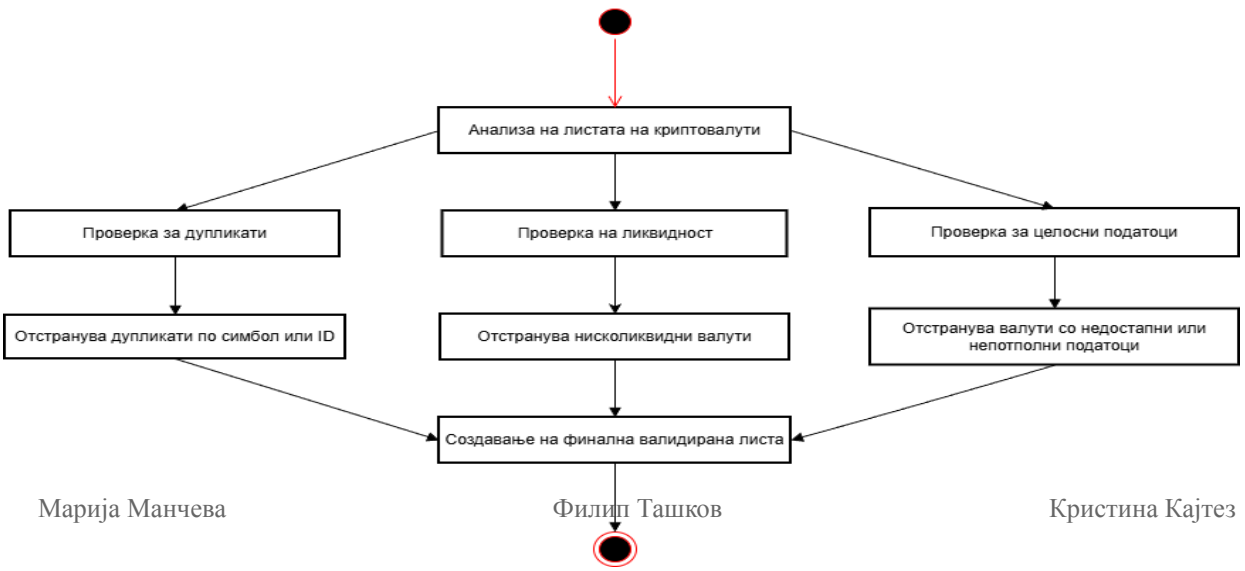
1.



Сценарио: Преземање на листа на топ 1000 криптовалути.

1. Администраторот го иницира процесот за преземање на листа од највредните 1000 криптовалути.
2. Системот испраќа барање до крипто берзата за највредните 1000 криптовалути.
3. Крипто берзата враќа листа со податоци за највредните 1000 криптовалути.
4. Системот ја складира листата како raw во pipeline процесот.

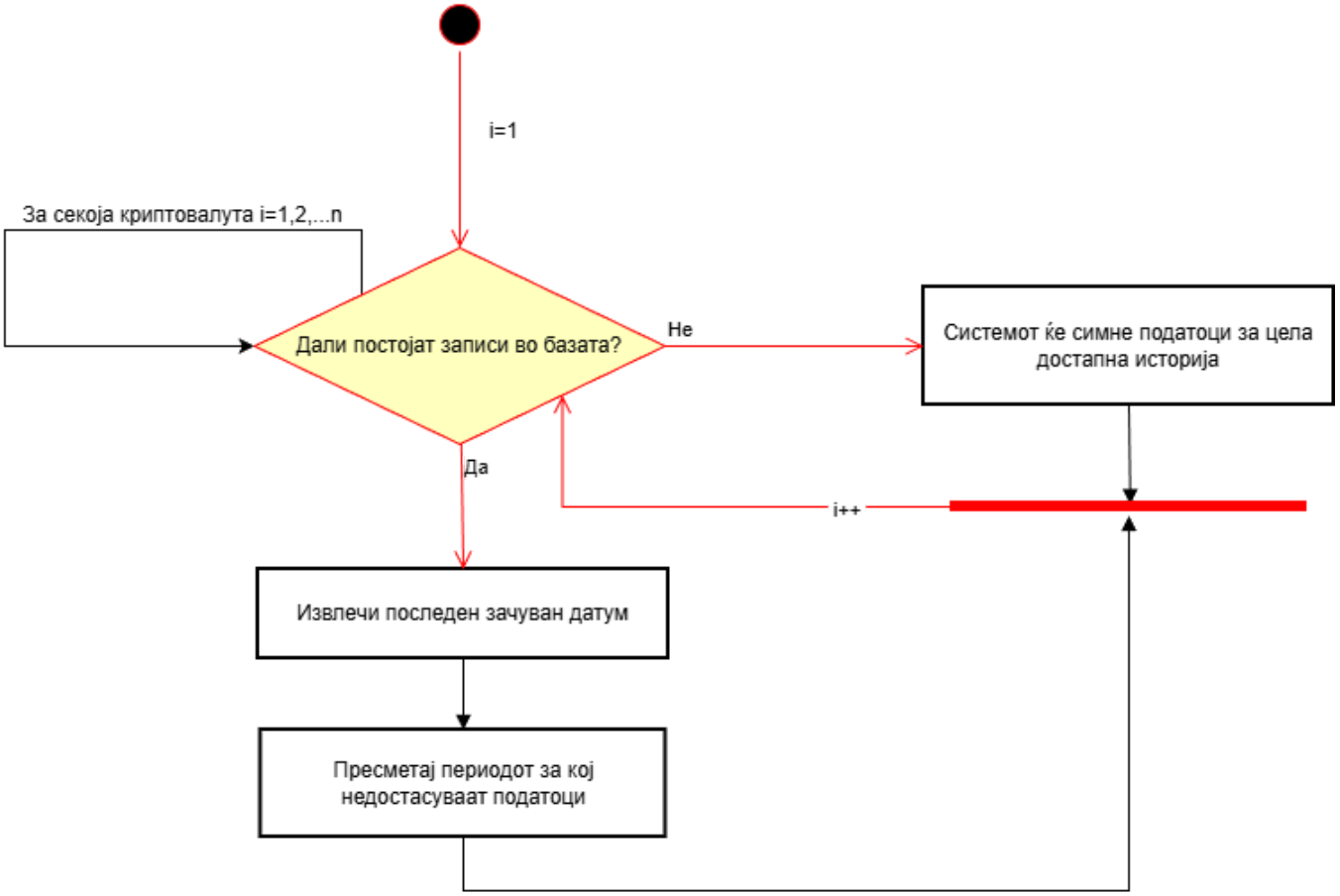
2.



Сценарио: Филтрирање на невалидни криптовалути.

1. Системот ја анализира листата на криптовалути.
2. Ги отстранува валутите со ниска ликвидност.
3. Ги елиминира дупликатите по симбол или ID.
4. Ги отфрла криптовалутите со недостапни или непотполни податоци.
5. Создава финална валидирана листа.

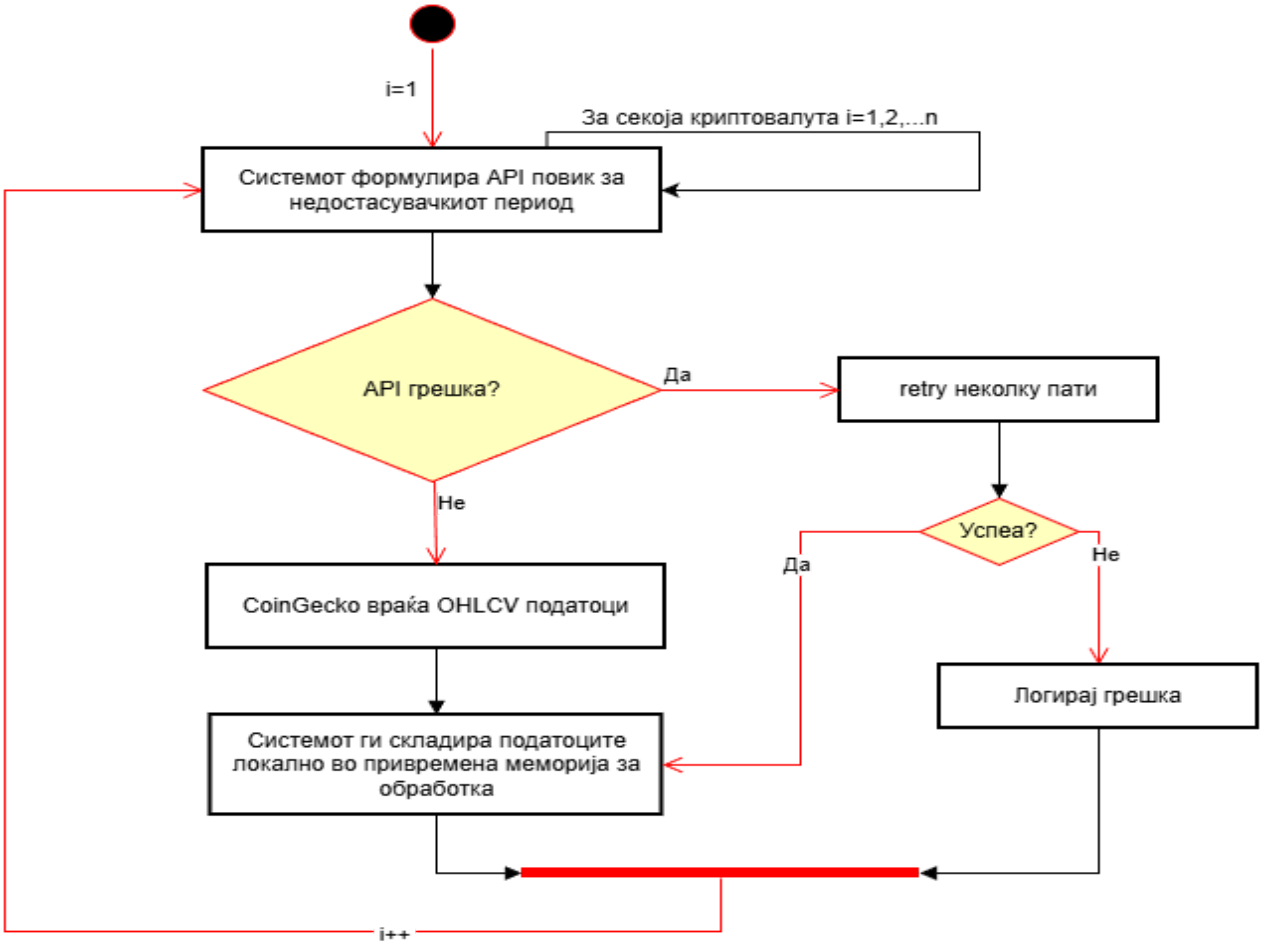
3.



Сценарио: Проверка на последен зачуван датум во базата.

1. Системот за секоја валута проверува дали постојат записи во базата.
2. Случај постојат:
   2.1. Го извлекува последниот зачуван датум.
   2.2. Го пресметува периодот за кој недостасуваат податоци.
3. Случај не постојат:
   3.1. Системот ќе симне податоци за цела достапна историја.
4. Продолжува со проверка за следната криптовалута.

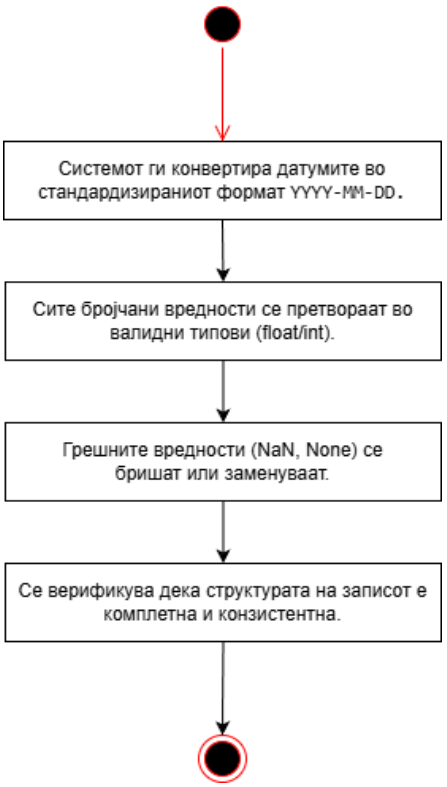
4.



Сценарио: Преземање дневни OHLCV податоци.

1. Системот формулира API повик за недостасувачкиот период.
2. Провери дали има API грешка:
3. Случај нема:
   3.1. CoinGecko враќа OHLCV податоци.
   3.2. Системот ги складира податоците локално во привремена меморија за обработка.
4. Случај има:
   4.1. Прави retry неколку пати.
   4.2. Ако успеа retry → системот ги складира податоците локално во привремена меморија за обработка.
   4.3. Ако не успеа retry → логира грешка.
5. Продолжува со проверка за следната криптовалута.

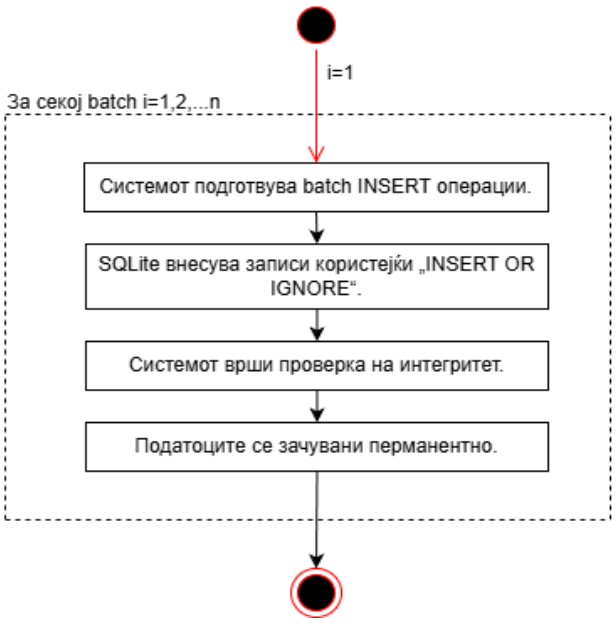
5.



Сценарио: Нормализација и чистење на податоци.

1. Системот ги конвертира датумите во стандардизиран формат YYYY-MM-DD.
2. Сите бројчани вредности се претвораат во валидни типови (float/int).
3. Грешните вредности (NaN, None) се бришат или заменуваат.
4. Се верификува дека структурата на записот е комплетна и конзистентна.

6.



Сценарио: Складирање на податоци во SQLite.

1. Системот подготвува batch INSERT операции.
2. SQLite внесува записи користејќи „INSERT OR IGNORE“.
3. Системот врши проверка на интегритет.
4. Податоците се зачувани перманентно.

7.

Сценарио: Мерење време на извршување.

1. Системот го стартува тајмерот пред обработка.
2. За секој филтер:
   2.1. Филтерот во pipeline го логира своето време.
   2.2. Времето на филтерот се додава во total_vreme.
3. По завршување системот пресметува вкупно време (total_vreme).
4. Времето се зачувува во датотека или база за анализа.

8.

Сценарио: Преглед на податоци (за подоцнежна фаза).

1. Корисникот ја отвора апликацијата.
2. Системот ги извлекува податоците од базата.
3. Податоците се прикажуваат преку UI (графици, табели).