*Дизаjн и архитектура на софтвер*

*Домашна работа 2*

***1. Архитектурен дизаjн***

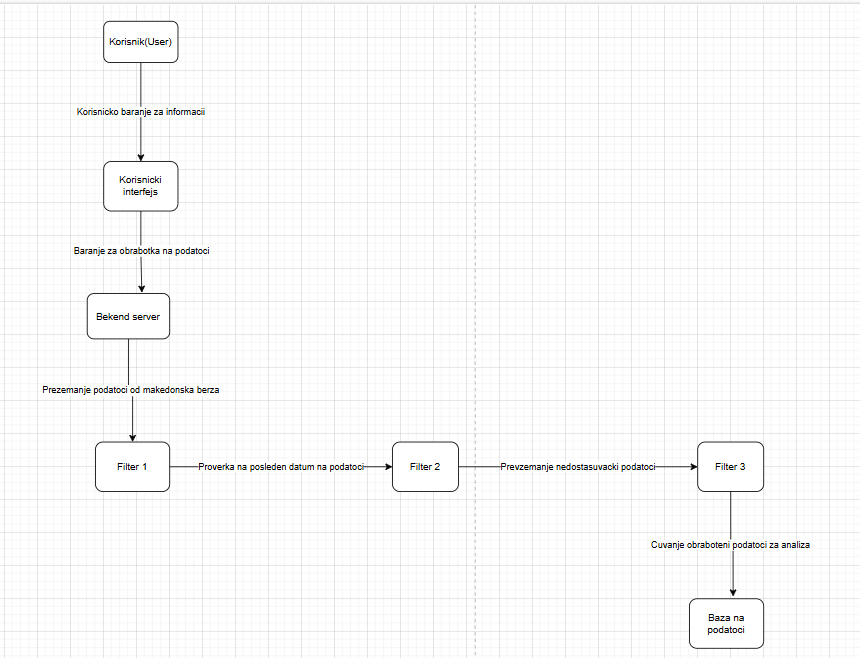
* **Концептуална архитектура**

Концептуалната архитектура го претставува високо-нивовското функционирање на апликацијата. Нашата апликација користи Pipe-and-Filter архитектура каде секој филтер обработува дел од податоците и ги проследува на следниот.

1. **Корисник (Korisnik/User):**
   * Корисникот започнува со барање за информации.
   * Ова е прв контакт со апликацијата и може да се изврши преку графички кориснички интерфејс.
2. **Кориснички интерфејс:**
   * Интерфејсот е слој кој го посредува барањето од корисникот до бекенд серверот.
   * Одговорен е за прикажување на резултатите или грешките кои произлегуваат од обработката.
3. **Бекенд сервер:**
   * Се грижи за обработка на барањата од корисничкиот интерфејс.
   * Овој дел иницира комуникација со базите на податоци и филтрите.
   * Презема податоци од македонската берза користејќи соодветен API или веб-скрепирање.
4. **Филтри:**
   * **Filter 1:**
     + Првичен чекор за проверка на податоците.
     + Ги идентификува и проверува последните достапни датуми за податоците.
   * **Filter 2:**
     + Презема податоци кои недостасуваат (ако има податоци што не се ажурирани или не се вклучени).
   * **Filter 3:**
     + Финална обработка на податоците.
     + Се подготвуваат за складирање во базата на податоци и анализа.
5. **Складирање:**
   * Обработените податоци се чуваат во база на податоци за идно користење.
   * Оваа база на податоци е централно складиште кое се користи за анализа или генерирање извештаи.

**Главни процеси:**

* **Проток на информации:**
  + Корисничкото барање се пренесува од интерфејсот до бекенд серверот.
  + Податоците се преземаат, филтрираат и обработуваат низ трите филтри.
  + Конечните податоци се складираат за анализа.
* **Филтрирање и проверка:**
  + Податоците се проверуваат за интегритет и комплетност.
  + Недостасувачките податоци се преземаат и верификуваат.
* **Анализа и чување:**
  + Сите релевантни податоци се обработуваат и складираат во формат погоден за понатамошна анализа.



* **Извршна архитектура**

Извршната архитектура се фокусира на реализацијата и интеграцијата на компонентите во системот. Таа го опишува начинот на кој фронтенд, бекенд и базата на податоци комуницираат за да се обезбеди функционирањето на апликацијата. Во нашиот пример, тоа вклучува поврзување на GUI, REST API и серверот за обработка на податоци. Секој дел работи независно, но сите се поврзани за да ја постигнат целта на системот.

#### **1. GUI (Graphical User Interface) – React.js или HTML/CSS**

* **Форма:**
  + **GUI** (Графички кориснички интерфејс) е визуелниот дел на апликацијата кој им овозможува на корисниците да комуницираат со системот.
  + Овој дел може да се изгради преку различни технологии како **React.js**, **HTML/CSS**, **Vue.js** или слични алатки за креирање на динамични и интерактивни веб страници.
* **Функција:**
  + **GUI** е одговорен за поставување на визуелни елементи како што се формули, копчиња, навигација и графики.
  + Овде корисниците ќе внесуваат податоци, ќе вршат пребарувања и ќе комуницираат со апликацијата.
  + GUI обезбедува интеракција со системот преку **UI** и ги визуелизира резултатите добиени од **Backend** преку **REST API**.
  + Тоа е првата точка на контакт за корисниците, која обезбедува лесен начин за користење на функционалностите на апликацијата.

#### **2. UI (User Interface) – Конектор помеѓу корисникот и REST API**

* **Форма:**
  + **UI** претставува **посредник** помеѓу корисникот и серверската логика, и ја овозможува интеракцијата преку визуелни компоненти на веб страницата.
  + UI е одговорен за динамично прикажување на податоци и управување со корисничките барања.
* **Функција:**
  + UI прима внесени податоци од корисниците преку GUI (форми, копчиња) и ги испраќа овие барања до **REST API**.
  + Ова претставува поврзаност која **предава податоци** од корисничкиот интерфејс до бекендот.
  + UI овозможува прикажување на резултати, и овие податоци се претставуваат на корисниците во форма на табели, графици, статистики или текст.

#### **3. REST API – Логичка поврзаност помеѓу фронтенд и бекенд**

* **Форма:**
  + **REST API** претставува **интерфејс** кој овозможува комуникација помеѓу различни компоненти на апликацијата, односно **фронтенд** и **бекенд**.
  + **REST** (Representational State Transfer) е архитектурен стил кој овозможува комуникација преку HTTP барања (GET, POST, PUT, DELETE) во формат JSON или XML.
* **Функција:**
  + **REST API** ги проследува барањата од UI до бекенд серверот, и ја пренесува логиката за обработка на податоците.
  + API-то ја овозможува **комуникацијата** помеѓу клиентската страна (фронтенд) и серверската страна (бекенд).
  + Тоа е посредник кој го координира протокот на податоци и сигурно ја управува размената помеѓу компонентите на апликацијата.

#### **4. Backend (Spring Boot/Node.js) – Обработка на податоци и логика на апликацијата**

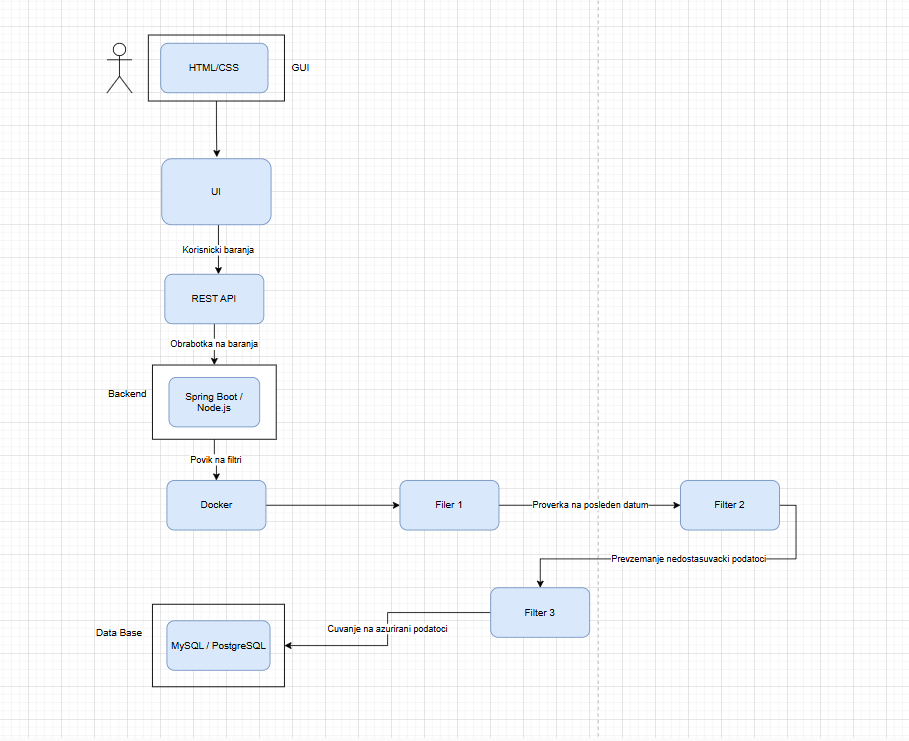
* **Форма:**
  + **Backend** е серверска компонента која ја обработува логиката на апликацијата и управува со комуникацијата со базата на податоци.
  + **Spring Boot** (Java) и **Node.js** (JavaScript) се два примери на серверски технологии кои се користат за развој на бекенд апликации.
* **Функција:**
  + Бекендот го прима барањето преку **REST API**, го обработува барањето, и ако е потребно, комуницира со **базата на податоци**.
  + Овој дел ја управува логиката на апликацијата, како што се валидирање на податоци, обработка на кориснички барања и генерација на резултати.
  + Бекендот исто така извршува операции, како што се пресметки, обработка на бизнис логика, или сл.

#### **5. Docker-контејнери – Изолирани контејнери за извршување на операции (филтри)**

* **Форма:**
  + **Docker** е алатка која овозможува **изолација на сервисите** преку контејнери.
  + Контејнерите се независни окружења каде може да се извршуваат различни операции, како што се обработка на податоци или специфични услуги, без да се влијае на останатите делови на апликацијата.
* **Функција:**
  + **Docker** го олеснува поставувањето и извршувањето на компоненти во изолирани окружења.
  + Кога зборуваме за филтри, тие можат да бидат поставени во Docker контејнери и да работат независно, обезбедувајќи изолација и контрола над ресурсите.
  + Изолирањето преку Docker овозможува поефикасно тестирање, обнова на компоненти и скалабилност на апликацијата.

#### **6. Database (MySQL/PostgreSQL) – Централна база која складира податоци за понатамошна анализа**

* **Форма:**
  + **Базата на податоци** е компонентата која се користи за складирање на сите податоци кои се потребни за понатамошна обработка и анализа.
  + **MySQL** и **PostgreSQL** се релациски бази на податоци кои обезбедуваат структурирани начини за складирање и управување со податоците.
* **Функција:**
  + Базата на податоци обезбедува **централно складиште за податоци**.
  + Тука се чуваат податоци кои се користат од страна на апликацијата: информација за корисниците, податоци за извештаите, историја на податоци, итн.
  + Базата на податоци може да биде поврзана со **бекендот** кој ја користи за чување, измена или вчитување на податоци во однос на потребите на апликацијата.



* **Имплементациска архитектура**

**Имплементациска архитектура** за нашиот пример се однесува на начинот на кој се имплементираат и организираат различните компоненти (модули) во софтверот. Во оваа архитектура, фокусот е на реализацијата и поврзувањето на различни делови од системот за да се обезбеди функционирање на апликацијата.

#### **Прв модул:** data\_pipeline.py

* **Опис**: Овој модул ги повикува функциите од сите филтри (filter1, filter2, filter3). Тој е главниот иницијатор за процесот и ја започнува целата пипелина на податоци.

#### **Втор модул:** filters.py

* **Опис**: Содржи имплементација на функциите filter1, filter2, и filter3. Овој модул е одговорен за обработката на податоците според правилата дефинирани во овие филтри.

#### **Трет модул:** data\_processing.py

* **Опис**: Овој модул се грижи за обработка и структура на податоците што ги добива од филтрите, и подготвува податоците за понатамошна анализа или складирање.

