Увод

Както знаем в съвременното интернет пространство имаме огромен набор от различни инструменти предлагащи ни различни решения.

Последните години един такъв инструмент започна да става изкуствения интелект. Неговата възможност да извършва различни типове операции като се започне от запълване на таблични данни и стигнем до генериране на текст, изображения и видео е практически неограничен. Всичко зависи от големината, качеството на данните и ресурса готов да заделим за да създадем такъв тип модели. С развитието на хардуера и софтуера все по-лесно обикновените хора биха имали възможност да създадат различни езикови модели, извършващи определена дейност. Например за текстова обработка.

Проблема идва в това, че на този етап все още технологията е твърде скъпа.

Поради тази причина реших да разработя приложение даващо възможност да запълним една много специфична нисша а именно prediction на данни. Идеята на приложението е всеки човек, независимо колко на брой данни има да може относително лесно и бързо да получи „предсказване“ как в бъдеще тези данни биха се променяли на база старите. За целта основната работа на алгоритъма ще се състои в следните неща: Прочитане на клиентските данни, preprocessing, за да се докарат до формат, който може да бъде разчетен от модела, expanding на данните, за да се добавят релациите между дните (за основа алгоритъма ще ползва autoregression + moving average + difference [Arima]), трениране на дървовиден тип boosting модел (xgboost) и изкарване на метрики и графики.

Чрез този алгоритъм целта е всеки клиент да има възможност напълно безплатно да получи prediction на своите данни.

Съществуващи подобни решения

1. Реално конкуренция като продукт, който да прави същото нещо няма. Има подобни решения, които съществуват ако хората са клиенти на дадена облачна услуга като google cloud, aws, azure и др.
2. Cloud Providers (google cloud, aws, azure).

Cloud Providers предлагат отскоро услуги, чрез които хората могат да натренират модели точно спрямо техните данни, независимо от формата им. Например гугъл предоставят две възможности:

Чрез техния vertex AI сервиз хората имат възможност да подадат файловете си и да следват стъпките, след които ще трябва да deploy-нат моделите си и да правят рикуести.

Другия вариант е да запазят данните в Big Query и чрез техния подобен сервиз на база тях да се натренира модел, който да прави prediction.

Основните проблеми с cloud providers са следните:

А. Висока цена – В момента в зависимост от данните, които има клиента и тяхната големина, цената може прогресивно да расте. Първоначалната тренировката и деплоймънта са стотици долари, а пък резултата в повечето случаи е меко казано лош, тъй като за тренировката се ползват основно automl модели, чието представяне единствено гледа оптимизацията на определена метрика, без да показва добри резултати дори след много часове тренировка. А изобщо за да се стигне до задоволителни резултати, часовете тренировка трябва съответно да са в повечето случаи > 10 часа. Като се има предвид че към днешна дата тренировката на всеки час е около 20 долара за 10 часа тренировка имаме 200 долара само за да се получи модел. Вече в зависимост от неговата големина, трябва да се деплойне на определена машина, а най слабата машина на месечна база харчи 70 долара, която в повечето случаи изобщо не е достатъчна.

За другите облаци pricing модела е различен, но горе долу цените се приближават до тези цифри.

Б. Техническа подготовка:

Повечето случаи хората, които искат да получат prediction на данните си нямат достатъчна техническа подготовка, за да използват тези облачни услуги.

Процеса по създаване, трениране и деплойване на модел въпреки UI улесненията изисква някаква минимална експертиза в сферата на AI, за да има възможност клиента да разбира стъпките и да взима правилни решения за да получи максимално реален резултат. Проблема е че повечето клиенти нямат такъв тип експертиза и оттам идва и затруднението.

В. Като положително единствено би могло да се счита факта, че данните, независимо колко разнообразни и сложни като формат са имат възможност да бъдат подадени и на база тях да се изкарат резултати.

3. Пренатренирани модели и интернет код.

Има много пренатренирани модели, които биха могли да се използват. Палтформа, която предлага такъв тип модели е HuggingFace, където потребител има възможност да използва готови модели и да ги промени за своя цел.

Има и други статии и материали, които предлагат такъв тип решения.

Основния проблем тук е именно по-голямата нужда от експертиза в AI, защото целия процес е обвързан с кодене, а над 95% от клиентите нямат никакъв опит и този процес за тях е практически невъзможен.

1. Точно заради горепосочените причини приложението, което ще бъде разработено (Predictor) е толкова решаващо. Чрез него единственото нещо, което ще се изисква от клиента е именно да се регистрира в приложението, да сложи един ексел файл, който трябва да има определена структура (дата, поле за предсказване) и да бъде напълнен с данни за даден период. След това заявката за prediction ще бъде изпратена и човека ще има възможност след нейното одобряване да получи резултати.

Цели и характеристика

1. Цели
   1. Целта на приложението е да помогне на потребителите, които имат събрани данни през годините да се възползват от пълно описание и предсказание на неговите данни.
   2. Всеки човек ще има възможност да се възползва от прииложението без да се изисква от него някаква експертиза в машинното обучение.
   3. Друга цел е приложението да бъде достъпно за хората през телефон, лаптоп или други устройства, които имат браузър, инсталиран.
2. Характеристика
   1. Технологична характеристика
      1. Аутентикация чрез JWT токен: Приложението използва JWT токен за аутентикация на потребителите, което гарантира сигурност на данните и предотвратява неоторизиран достъп до функционалностите на приложението.
      2. Приложението е изградено с много модерни технологии, които гарантират бързина, стабилност и сигурност. За бекенд частта се използва Spring Boot, с Java 17 като основен език за програмиране. Този избор осигурява лесна интеграция и разширение на системата. Spring boot още предоставя лесна за поддържане и същевременно много богата на функционалности архитектура за Уеб програмиране. Фронтенд частта на приложението е разработена с Angular - популярен JavaScript фреймуърк, който предоставя бързина и отзивчивост на потребителския интерфейс. Освен това дава на програмистите възможността за бърза и лесна интеграция с други архитектури, както и бързовременно оправяне на проблеми и поддръжка.
      3. Към angular се използва известната angular materials библиотека, която ни позволява прилагането на materials design концепцията, както и по-добра, красива и бърза имплементация на желаните функционалности.
      4. За съхранение на данните се използва PostgreSQL база данни. PostgreSQL е мощна релационна база данни, която осигурява сигурност и надеждност при управлението на информацията.
      5. Тъй като приложението ще предвижда цени, за да се предостави максимално висока точност се използват известни технологии и статистически модели за обработка на данни. За основен модел се използва широко известния Xgboost. Това е дървовиден буустинг статистистически модел, който на базата на подадените параметри, конструира дърво чрез ентропия и други средства, за да може да предскаже бъдещите данни. Това прави модела изключително точен за данни, които е научил. Специфично за него е че е подходящ за информация, която върви отностително синхронно и няма много аномалии. Други библиотеки за обработка на данни, плотване на графики и показване на метрики са sklearn, pandas, numpy, matplotlib и други.

Защо тези технологии

При разработката на приложението за предсказване на резултат на база минали данни, избрах Spring Boot с Java за бекенд, Angular за фронтенд и PostgreSQL за база данни. Тези технологии бяха избрани след анализ на изискванията и характеристиките на проекта и предлагат редица предимства, които спомагат за постигане на успешно и надеждно решение на съответния проблем.

1. Spring boot с java за бекенд

* Удобство и бързина на разработка: Spring Boot предоставя много полезни инструменти и библиотеки, които значително улесняват разработката на бекенд частта на приложението. Използването на Java осигурява стабилност и сигурност, които са от съществено значение за управлението на потребителски данни и сигурност на приложението.
* Интеграция и гъвкавост: Spring Boot е силно интегриран с други инструменти и позволява бързо и лесно свързване с различни технологии и външни услуги. Това предоставя гъвкавост и възможност за разширение на функционалностите на приложението, което е от особено значение за бъдещото развитие на приложението и неговото разширяване.

1. Angular за фронтенд

* Отзивчив интерфейс: Angular предоставя мощни инструменти за създаване на отзивчив и интерактивен потребителски интерфейс. Това е от голямо значение за добро потребителско изживяване и удовлетворение от работата с приложението.
* Модулна архитектура: Angular позволява създаването на модулна архитектура, която улеснява разработката и поддръжката на кода. Това прави приложението по-структурирано и лесно мащабируемо, което е от значение за постигане на добра поддръжка и разширяемост на приложението.

1. PostgreSql за база данни

* Надеждност и скалируемост: PostgreSQL е мощна релационна база данни, която предоставя надеждност и устойчивост на данните. Това е от съществено значение за сигурното съхранение и обработка на данните на потребителите.
* Поддръжка на големи обеми данни: PostgreSQL може да се справя с обработка на големи обеми данни и е подходяща за системи с интензивно натоварване. Това е от значение за приложението, което трябва да обработва различни заявки и изисква постоянна работа с потребители.

Общо взето, изборът на Spring Boot с Java, Angular и PostgreSQL беше направен с цел да се осигури сигурно и лесно управляемо приложение, което предоставя бързина, отзивчивост и добро потребителско изживяване за всички потребители, търсещи бизнеси и записвайки се за часове.

1. JupyterLab + Python за ИИ

* Изборът на JupyterLab + Python идва поради няколко причини:
  + Огромното community и support, което се предоставя за този тип технологии
  + Огромен набор от библиотеки за модели с подходящи и богати интерфейси. Такива библиотеки са например pytorch, tensorflow, sklearn, xgboost и много други.
  + Огромен набор от библиотеки за data обработки като pandas, numpy и други.
  + Наложен е като стандарт за задачки, свързани с машинно обучение и изкуствен интелект
* Средата предоставя лесна и интуитивна визуализация
* Поддържа интеграция с документация и коментари
* Лесно стартиране и възпроизвеждане на резултати от тестове и експерименти

Архитектури

1. MVC

Този архитектурен модел се използва в много сфери на разработката на софтуер, включително уеб приложения, настолни приложения и мобилни приложения.

По-долу може да се види подробно описание на компонентите и работата на MVC, както и ползите от използването му:

1. Модел (Model): Моделът представлява данните и бизнес логиката на приложението. Той съдържа структурирани данни, които приложението обработва и манипулира. Моделът не зависи от нищо друго в архитектурата и не знае за съществуването на изгледа или контролера. Основната цел на модела е да осигури интегритет и актуалност на данните, както и да извършва бизнес логиката на приложението.
2. Изглед (View): Изгледът отговаря за представянето на данните на потребителите. Той се занимава със структурата и изгледа на информацията, която потребителите виждат, и с която взаимодействат. Изгледът съдържа логиката за това как да се покажат данните, но не съдържа логиката, свързана със самата обработка на данните. Изгледът реагира на промени в модела и извежда актуалната информация на потребителите.
3. Контролер (Controller): Контролерът служи като посредник между модела и изгледа. Той приема вход от потребителите чрез изгледа, обработва го и взаимодейства с модела за извършване на необходимата логика. Контролерът реагира на събития като например натискане на бутони, взаимодейства с модела, актуализира изгледа и изпраща обратно резултатите към изгледа.
4. Ползи от MVC
5. Разделяне на отговорности: MVC разделя приложението на четири основни структури с различни отговорности: управление на данните, представяне на данните, бизнес логика и управление на потребителския интерфейс. Това прави приложението по-структурирано и лесно за поддръжка.
6. Повторно използване на код: Поради разделянето на отговорностите, често може да се използва същият модел или изглед за различни части от приложението, което увеличава повторното използване на код.
7. Гъвкавост и мащабируемост: Тoзи архитектурeн модел позволява лесно да се добавят нови функционалности или да се променят съществуващите без да се нарушава цялостната структура на приложението.
8. Тестваемост: Моделът и контролерът могат да бъдат лесно тествани автономно, без зависимост от изгледа. Това улеснява написването на модулни и надеждни тестове.
9. Изглед на данните: Изгледът може да бъде лесно адаптиран за различни устройства и платформи. Например, един и същи модел и контролер може да се използва с различни изгледи за уеб, мобилни устройства и десктоп приложения.
10. Лесно управление на дизайна: Разделянето на изгледа от бизнес логиката прави по-лесно за дизайнерите да работят с изгледа без да се налага да разбират подробностите на бизнес логиката.
11. Дизайн:

Material Design е дизайн концепция, разработена от Google, която се използва за създаване на съвременни и интуитивни потребителски интерфейси за уеб и мобилни приложения. Тази концепция на дизайн комбинира красота, функционалност и удобство за потребителите.

По-долу е предоставено подробно описание на Material Design и неговите основни принципи:

1. Материален интерфейс (Material Design): Основният принцип на Material Design е изграждането на интерфейса чрез представяне на обекти като комбинация от "материали". Тези материали се движат и реагират на входните действия на потребителите като реални материали в тримерния свят.
2. Тримерен дизайн (3D Design): Material Design използва тримерни ефекти, за да придаде дълбочина и реалистичност на интерфейсите. Този тримерен дизайн се постига чрез ефекти като подчертаване, сянка и анимации и други.
3. Движение и анимации: Анимациите са важна част от Material Design и се използват, за да подчертаят движението и действията на потребителите. Те правят преходите между различните екрани и действия по-приятни и ефективни.
4. Цветова палитра и теми: Material Design предоставя широка цветова палитра и теми, които могат да бъдат използвани за създаване на съвременни и привлекателни интерфейси. Цветовата палитра включва основни цветове, акцентни цветове и тонове. Разделят се на два вида – такива които са предварително предефинирани или къстомизирани според нуждите**.**
5. Икони и типография: Дизайнът на иконите и типографията в Material Design е ясен и четлив. Иконите са минималистични и улесняват разпознаването на действията, а типографията предоставя четърто и удобство при четене.
6. Следене на движението (Motion Mapping): Този принцип показва как се движат и реагират различни елементи на интерфейса на потребителски действия. Това подпомага потребителите да разбират какво се случва в интерфейса при техните действия.
7. Интерактивност и отзивчивост: Material Design се фокусира върху интуитивността и отзивчивостта на интерфейсите. Потребителите трябва да могат лесно да взаимодействат с интерфейса и да получават бързи обратни сигнали за своите действия.
8. Унифицирано изживяване (Unified Experience): Material Design се стреми да предостави унифицирано и съвместимо изживяване на различни устройства и платформи, включително уеб, мобилни устройства и десктоп приложения.
9. Предефинирани компоненти (Material Components): Material Design предоставя богат набор от готови компоненти и библиотеки, които улесняват изграждането на интерфейси в съответствие с дизайн концепцията. Тези компоненти са налични за различни платформи и приложения.

Ползите от използването на Material Design включват подобрена употребимост, привлекателен дизайн, по-добра интуитивност и по-добро съответствие със стандартите на дизайна на Google. Тази концепция на дизайн е широко приета и се използва в много софтуерни продукти и уебсайтове, което прави потребителските изживявания по-добри и консистентни.

A screenshot of a mobile application

Description automatically generated

Изисквания

1. Функционални изисквания
   * Регистрация и логин
     1. Приложението ще предоставя заглавна страница, която ще води към 2 екрана – логин или регистрация
     2. Логин страницата ще съдържа две инпут полета – 1 поле за username и едно за парола
     3. Логин страницата ще има бутон, с който да може потребител да отиде да се регистрира
     4. Компонента за регистрация ще се състои от няколко инпут полета – потребителско име, парола и потвърди парола
     5. След регистрация потребителя ще бъде пренасочен да към логин страницата
   * Роли:
     1. Приложението ще поддържа две роли – обикновен потребител и администратор
     2. Обикновения потребител ще има възможност да създава заявки за prediction и да преглежда статуса им, както и след завършване да види резултатите
     3. Обикновения потребител ще може да променя и да трие заявки в статус pending и approved
     4. Администратора ще има възможност да одобрява или отказва заявките на различни потребители
   * Заявки
     1. След логин обикновения потребител ще бъде пренасочван към основния екран, през който ще има възможност да преглежда своите заявки, статуса им.
     2. Ще има възможност след като заявката е завършена да отива в overview екран, който да показва резултатите от неговата заявка.
     3. Резултатите са prediction на база csv файл, който е подал в заявката.
     4. Резултата включва графика, в която по x оста ще бъдат показани „подадените“ и „бъдещите“ дати. По y оста ще се показва движението на алгоритъма, т.е. как той се е справил на старите данни и какво предсказва за бъдещите.
     5. В overview екрана освен графиката ще се виждат и основни метрики като RMSE(Root mean squared error), RMAE (Root mean absolute error) и други. Така потребителя ще има възможност да проследи дали данните му са били „годни“ за prediction
   * Създаване на заявки
     1. В основния екран на обикновения потребител ще има бутон, който ще ни пренасочва към екрана за създаване на заявки.
     2. В екрана Create request ще има няколко полета – име на заявката, описание на данните, ъплоуд бутон за добавяне на екселски файл (с валидация и текст, който описва желаната структура на данните), телефон и бутон submit, който запазва заявката
     3. След ъплоуд на файл на потребителя ще се показва дали файла е с изискуемата структура
     4. Освен това ще се показва таблица на файла с нейните първи 8 записа
   * Статус на заявките
     1. Непосредствено след създаване на заявка тя е със статус Pending.
     2. Админа може да преглежда и да сменя статусите на заявките към APPROVED или DECLINED
     3. След смяна на статуса на APPROVED, ръчно се включва сървъра за prediction, който взима всички заявки със статус APPROVED и минава през процеса за prediction.
     4. След завършване на процеса за prediction данните се записват в таблица на име prediction\_response, която сочи към съответната заявка.
2. Нефункционални изисквания
   * Имплементацията на Prediction алгоритъма да се осъществи с jupyterlab + python
   * Приложението ще бъде уеб базирано
   * Приложението ще може да се достъпва през chrome, firefox и internet explorer
   * Приложението да поддържа телефонен изглед
   * Системата трябва да бъде налична 24 часа 7 дни в седмицата
   * Системата да бъде лесно поддържана и разширяването й откъм нови функционалности да става максимално бързо.
   * За дизайна да се използва material design концепцията
   * Системното секюрити трябва да бъде осъществено използвайки refresh и access JWT токен
   * С до 4 клика трябва да се достига до всяка една функционалност на приложението
3. Бизнес изисквания
   * Да се връща отговор на всяка заявка до 1 ден
   * Да се позволяват до 3 заявки на клиент максимално

Екранна визуализация

1. Визуализация на основните екрани
   1. Изглед на създаване на заявка за предсказване на температурата по зададени стари данни без подаден файл:

A screenshot of a computer

Description automatically generated

* 1. Изглед на създаване на заявка за предсказване на температурата по зададени стари данни с подаден файл:

A screenshot of a computer

Description automatically generated

* 1. Изглед на потвърдена и отказана заявка от администратора:

A screenshot of a computer

Description automatically generated

* 1. Изглед на по-горния екран от потребителска гледна точка:

A screenshot of a computer

Description automatically generated

* 1. Изглед на изтриване на заявка:

A screenshot of a computer

Description automatically generated

* 1. Изглед на промяна на заявката с непопълнен телефон:

A screenshot of a computer screen

Description automatically generated

* 1. Изглед на footer при hover на него:

A screenshot of a computer

Description automatically generated

* 1. Изглед от изпълнен prediction за температури (Периода е за 6 месеца – може да се променя):

A screenshot of a computer

Description automatically generated

* 1. Страница за влизане в системата

A white rectangular object with black text

Description automatically generated

1. Работен поток:
   1. Основен потребител:
      1. Основния поток на информация и данни се изпълнява в приложението по следния начин: Първо след регистрация клиента бива пренасочен в основния екран за неговата роля. Оттам той има възможност да създаде заявка или да прегледа негови създадени заявки.

Оттам след създаване на заявка, тя е в статус Pending. Потребителя трябва да изчака администратора да одобри или да откаже заявката.

При одобрение на заявката се изчаква притежателя на кода за prediction да стартира сървъра.

Самия сървър след стартиране захваща от същата база данни всички заявки със статус approved, изчита техните файлове и тренира по един Prediction модел на файл. След това прави prediction за период, който притежателя на сървъра определя (1,2,3... месеца). След завършване на Prediction-а резултатите се записват в базата и сървъра се стопира, като превключва всички заявки в статус DONE. След приключване потребителя има възможност да види генерираните от сървъра prediction-и като натисне иконата изобразяваща око.

Use case diagram

A diagram of a company

Description automatically generated

Посочената по-горе снимка е диаграма на потребителските случаи за потребител с роля „Обикнове потребител“ и такъв с роля „Администратор“

Бъдещи насоки за развитие

1. Системата е разработена и имплементирана така че да има възможност за бърза, лесна и евтина поддръжка и развитие. Технологиите са последно поколение и има доста добра и развита общност, което допринася за нейното бъдещо развитие.
2. Това, което се планира е системата да бъде хоствана на облак като AWS или GCP като това ще намали драстично разходите по нейната поддръжка и ще я направи лесно скалируема.
3. Откъм функционалности ще бъдат добавени възможности алгоритъма да работи с повече фийчъри ако някой prediction зависи и от други странични характеристики освен ARIMA ( autoregression, moving average, difference )
4. Ще се добавят повече метрики, с които потребителя да сверява точността на своя prediction.
5. Ще се създаде възможността потребителя ръчно да задава някои от характеристиките на алгоритъма за предсказване (напр: Дали да бъде по-генерализиращ към данните или по overfit-ващ).

Имплементация с примери

1. Както е споменато в предишни точки секюритито включва два филтъра – ауторизационен и аутентикационен, които чрез вградената секюрити конфигурация на спринг се интегрират в системата. Аутентикационния филтър имплементира UsernamePasswordAuthenticationFilter, а Ауторизационния – OncePerRequestFilter, защото това са основните интерфейси, които спринг секюрити конфигурация изисква.

A screenshot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

1. Този код конфигурира сигурността на приложението, използващо Spring Security, като настройва няколко ключови компонента за автентикация и авторизация. Ето по-подробно описание на основните му части:
   1. Анотации и дефиниция на класа: `@Configuration`: Маркира класа като конфигурационен клас за Spring. `@EnableWebSecurity`: Активира уеб сигурността на Spring Security.

`@EnableMethodSecurity`: Активира сигурността на методите в приложението.

* 1. Методи и конфигурация:

Дефинира бийн за `AuthenticationManager`, който се използва за управление на автентикацията.

* 1. filterChain: Конфигурира основното секюрити на HTTP заявките чрез `SecurityFilterChain`, чрез следните конфигурации – CORS (Cross-Origin Resource Sharing), CSRF (Cross-Site Request Forgery): Дезактивира CSRF защитата.

Настройва сесиите да бъдат `STATELESS`, което означава, че приложението няма да създава сесии за потребителите.

Дефинира правилата за достъп до различни URL-и. Публичните URL-и са достъпни за всички, а всички останали изискват автентикация.

Настройва обработката на exceptions при автентикация и отказан достъп.

Добавя два филтъра, които имат custom имплементация -

`CustomAuthenticationFilter`: Обработва заявките за вход (логин).

CustomAuthorizationFilter`: Филтър, който се изпълнява преди UsernamePasswordAuthenticationFilter и проверява правата на издадените JWT токени.

1. Контролери

A screenshot of a computer program

Description automatically generated

* 1. На снимката са показани три контролера, имащи за цел да осъществяват основната комуникация с angular frontend.
  2. BaseControllerClass е базовия клас, който всеки долен контролер имплементира. Там се поставят всички основни глобални конфигурации на останалите контролери, за да се избегне дублиране на код и контекст.
  3. Останалите два контролера наследяват базовия и неговите конфигурации като имплементират свои методи за достъп до фронтенда. Ето пример за листването на всички predictionRequests на даден потребител:

A screenshot of a computer program

Description automatically generated

1. DTO

A blue and grey background with white text

Description automatically generated

* 1. Това са обекти, които всеки един контролер генерира и връща на фронтенда. Целта им е да не се закарва до потребителя информация, която е скрита за него като например ид-та на обектите, които се запазват в базата данни, сензитивна информация за текущия потребител и други.

1. Сервизи:

A screenshot of a computer program

Description automatically generated

* 1. Сервизът съдържа основната бизнес логика за обработката и запазването на данни.
  2. Аналогично и със същата цел съществува базов сервиз, който бива наследен от всеки сървиз клас.
  3. Съществуват интерфейси за всеки клас с цел да достигат до контролера само функции и методи, чиято цел е да генерират финалната бизнес логика, за да може всеки слой от приложението да е максимално изолиран и да върши точно конкретна задача. Това дава на приложението лесна конфигурируемост и позволява по-бърза и лесна модификация и разширение.

Пример за интерфейс:

A screen shot of a computer program

Description automatically generated

Пример за имплементационен файл:

A screenshot of a computer program

Description automatically generated

Използвана литература

1. <https://www.codejava.net/frameworks/spring-boot/spring-security-jwt-authentication-tutorial> - за JWT аутентикацията
2. <https://www.baeldung.com>
3. <https://www.tutorialspoint.com/>
4. <https://swagger.io/docs/> - В последствие беше премахнато от приложението
5. <https://www.postgresql.org/docs/>
6. <https://www.w3schools.com/>
7. <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
8. <https://stackoverflow.com/>
9. <https://material.angular.io/>
10. <https://angular.io/docs>
11. <https://materialtheme.arcsine.dev/> - за цветовете и шрифта на приложението