Обозначения и сокращения

P&L – бюджет доходов и расходов

NPV – чистый приведенный доход

IRR – внутренняя норма доходности

DPI – дисконтированный индекс доходности

CI/CD - непрерывная интеграция и непрерывное развертывание

MVC – архитектуный паттерн «модель – представление - контроллер»

ООП – объектно-ориентированное программирование

HTML – язык гипертекстовой разметки

SSR (англ. server-side rendering) – серверный рендеринг – генерация HTML-кода всей страницы на сервере в ответ на запрос.

SaaS - программное обеспечение как сервис.

PaaS - платформа как сервис.

IaaS - инфраструктура как сервис модели и/или их комбинация.

1. Финансовая модель

Для построения модели заданы начальные макроэкономические параметры, оказывающие влияние на проект:

* темп инфляции
* налоговые ставки
* курсы валют
* демографические показатели
* занятость населения
* уровень цен
* ставки по кредитам

Делается ряд предположений для расчета основных финансовых показателей модели и оценки инвестиционной привлекательности:

* старт проекта
* разгон проекта
* прогноз спроса
* прогноз предложения
* портрет клиента
* стоимость услуг
* затраты

На основании оценки потребности в ресурсов строится бюджет расходов на весь горизонт планирования, составляющий три года. План по выручке строится на основании маркетингового плана продаж.

Для оценки показателей инвестиционной привлекательности анализируется состава и стоимость капитала. Ставка дисконтирования к расчету принимается на основе WACC. Рассчитываются показатели (по сути все они участвуют в расчете WACC):

* WACC (ставка дисконтирования)
* безрисковая ставка
* премия за риск
* доходность рыночного портфеля
* бета актива фактическая (средняя для аналогов)

Модель предусматривает различные схемы финансирования проекта. Объем финансирования рассчитывается исходя из оценки необходимых инвестиций и капитальных затрат для запуска проекта и объема финансирования операционной деятельности.

На основе рассчитанного бюджета доходов и расходов строятся денежные потоки по операционной, инвестиционной и финансовой деятельности. Рассчитывается свободный дисконтированный денежный поток и показатели инвестиционной эффективности (NPV, IRR, DPI). Бюджет согласуется с метриками юнит-экономики.

* 1. Бюджет доходов и расходов

* 1. Аргументация прогнозных вводных данных основных статей бюджета доходов и расходов (P&L)
     1. Расходы проекта. Расчет потребности в ресурсах проекта.
        1. Трудовые ресурсы.

На ранней стадии проекта самая большая статья расходов и самый труднодоступный ресурс. Было составлено штатное расписание в максимальной комплектации для анализа трудовых ресурсов и расчета ФОТ, что позволило понять как потребность в специалистах, так и рассчитать примерный бюджет на команду разработки и поддержки и получить бенчмарк для дальнейшего  выбора, балансировки и композиции этапа программной разработки (фриланс, аутсорсинг, внутренние ресурсы команды проекта, комбинирование и пр.)

Продукт разрабатывается собственными силами и средствами команды проекта. Технологический стек проекта представлен open source решениями.

* + - 1. Информационные и интеллектуальные ресурсы.

Использование облачных решений позволит значительно сократить затраты на аппаратно-техническую составляющую проекта (практически исключить материальные ресурсы) и упростить этап развертывания и тестирования сервиса проекта. В расчет принят бюджет в основе которого предварительная оценка вычислительных мощностей и средняя стоимость ежемесячной аренды серверов с рассчитанными показателями (SaaS - программное обеспечение как сервис, PaaS - платформа как сервис, IaaS - инфраструктура как сервис модели и/или их комбинация) на облачных сервисах РФ (Яндекс, Cloud.ru, Selectel, Immers.cloud.ru - требование проекта - нахождение  дата центров на территории РФ). По мере роста числа пользователей сервиса при успешной реализации проекта доступно масштабирование мощностей, как горизонтально, так и вертикально, и, соответственно, бюджет будет подлежать корректировке.

Также в бюджет заложены, в случае возникновения потребности, затраты на покупку лицензий ПО (интегрированные среды разработки, редакторы, расширение  версий для коммерческой разработки open source решений (github,  docker, серверы БД, проприетарные и платные библиотеки для  кодовой базы проекта и машинного обучения, прочие платные API). Также зарезервированы средства для оплаты  расширенных возможностей работы с источниками исторических и оперативных данных для машинного обучения (MOEX -  Московская биржа) и подписки на необходимые медиа-ресурсы.

Маркетинг, реклама и продвижение. Сделана предварительная оценка расходов на продвижение сервиса с учетом расценок за услуги интеграций на тематических каналах с определенным количеством аудитории, затраты на Яндекс Директ, партнерские программы. Также предполагается часть средств направить на создание и поддержание бренда (youtube & telegram каналы, smm). Следующим этапом необходимо четко проработать маркетинговую программу и каналы привлечения и атрибуции.

* + - 1. Административные расходы.

Заложены средства для оплаты банковского, бухгалтерского и юридического обслуживания (аутсорсинг).

* + - 1. Финансовые ресурсы и налоги.

Финансовые ресурсы и налоги. Рассматривается возможность получение гранта для финансирования разработки и чистого оборотного капитала на  старте проекта. В противном случае будет применяться схема софинансирования и кредитного финансирования.

С учетом специфики проекта (ИТ - высокомаржинальный продукт) предполагается воспользоваться упрощенной системой налогообложения по схеме 6 процентов (6 % из выручки).

* 1. Основные финансовые показатели

1. Разработка веб-приложения.
   1. Архитектура

Приложение представляет собой клиент-серверную архитектуру и построено на принципах шаблона (паттерна) проектирования MVC, а именно на идеи разделения логики и данных приложения на отдельные компоненты. Использование MVC-подобных подходов на стыке парадигмы ООП и основных принципов разработки программного обеспечения де-факто являются стандартами в современной разработке веб-приложений, обеспечивая гибкую и адаптивную разработку, поддерживаемость, тестируемость, читаемость и чистоту кода. В своих канонических трудах Мартин Фаулер и Роберт Мартин исчерпывающе описывают практическую значимость принципов ООП и важность «чистой» архитектуры, а в знаменитой и широко известной в профессиональных кругах работе «банды четырех» обоснованы и описаны практические примеры использования архитектурных паттернов.

* 1. Технологический стек
     1. Языки и парадигмы программирования

Язык программирования для написания серверной части проекта - python, а конкретно версии 3.11. Для создания интерфейсов в проекте используется язык JavaScript последей версии.

В основу выбора языков легли следующие соображения и потребности проекта:

**Серверная часть:**

* **Простота и скорость разработки**  
  Python известен своей простотой и читаемостью кода. Это позволяет быстро разрабатывать и поддерживать приложения, что критически важно для небольших команд и проектов с ограниченными ресурсами.  
  **Привязка к проекту**: Проект на старте, для разработки MVP не требует сложной логики или расчетов, поэтому Python идеально подходит для быстрой реализации функционала. Также следует отметить, что микросервис для машинного обучения также написан на python, хотя этот факт и не является определяющим, однако позволяет достичь максимальной совместимости, синхронизации и взаимодействия сервисов друг с другом.
* **Асинхронность (FastAPI + asyncio)**  
  Python с библиотекой asyncio и фреймворком FastAPI обеспечивает высокую производительность для обработки множества одновременных запросов.  
  **Привязка к проекту**: Проект, его коммерческий успех и популярность в будущем обеспечивает перспективу большого количества запросов, и асинхронность Python позволяет эффективно справляться с такой нагрузкой.
* **Богатая экосистема библиотек**  
  Python имеет огромное количество библиотек для работы с данными (SQLAlchemy), аутентификации (OAuth2, JWT), кеширования (Redis) и фоновых задач (Celery).  
  **Привязка к проекту**: Все эти библиотеки уже используются в вашем проекте, что упрощает разработку и интеграцию.
* **Поддержка JSON и RESTful API**  
  FastAPI автоматически генерирует документацию OpenAPI и поддерживает работу с JSON "из коробки".  
  **Привязка к проекту**: Ваш проект использует RESTful API и JSON, что делает Python идеальным выбором.
* **Сообщество и документация**  
  Python имеет огромное сообщество и отличную документацию, что упрощает поиск решений и обучение.  
  **Привязка к проекту**: Для небольшой команды это критически важно, так как снижает затраты на поддержку и развитие проекта.

**Почему не Go?**

* **Сложность разработки**  
  Go требует большего внимания к деталям (например, управление памятью, строгая типизация), что замедляет разработку.  
  **Привязка к проекту**: Проект не требует сложной логики, поэтому простота Python предпочтительнее.
* **Меньше библиотек для стека** **проекта**  
  Go имеет меньшую экосистему библиотек для работы с базами данных, аутентификации и фоновых задач по сравнению с Python.  
  **Привязка к проекту**: Вам уже нужны Redis, Celery, SQLAlchemy, и Python предоставляет готовые решения.
* **Асинхронность**  
  Хотя Go поддерживает корутины, Python с asyncio и FastAPI предоставляет более удобный и понятный интерфейс для асинхронной разработки.  
  **Привязка к проекту**: Асинхронность Python проще в реализации и поддержке.

**Почему не Java?**

* **Сложность и объем кода**  
  Java требует больше boilerplate-кода, что замедляет разработку.  
  **Привязка к проекту**: Ваш проект небольшой, и Python позволяет быстрее достичь результата.
* **Меньшая гибкость**  
  Java менее гибкий язык, особенно в сравнении с Python.  
  **Привязка к проекту**: Python позволяет быстрее адаптироваться к изменениям в требованиях.
* **Производительность**  
  Хотя Java быстрее Python, для проекта (отсутствие сложных расчетов) это не критично.  
  **Привязка к проекту**: Асинхронность Python с запасом компенсирует разницу в производительности.

**Клиентская часть:**

**JavaScript**

* **Простота и скорость разработки**  
  JavaScript не требует дополнительной настройки и компиляции, что ускоряет разработку.  
  **Привязка к проекту**: Ваш проект небольшой, и JavaScript позволяет быстрее реализовать функционал.
* **Широкая поддержка и сообщество**  
  JavaScript имеет огромное сообщество и множество библиотек (React, Ant Design).  
  **Привязка к проекту**: Это упрощает поиск решений и интеграцию с вашим стеком.
* **Отсутствие сложной логики**  
  Ваш frontend не требует сложной типизации или строгой структуры, поэтому TypeScript может быть избыточным.  
  **Привязка к проекту**: JavaScript достаточно для реализации интерфейса.

**Почему не TypeScript?**

* **Избыточность**  
  TypeScript добавляет строгую типизацию, что полезно для больших проектов, но избыточно для небольших.  
  **Привязка к проекту**: Проект не требует сложной типизации, поэтому JavaScript предпочтительнее.
* **Дополнительные затраты**  
  TypeScript требует настройки и компиляции, что увеличивает время разработки.  
  **Привязка к проекту**: Для небольшой команды это может быть неоправданно.
* **Кривая обучения**  
  TypeScript требует дополнительных знаний, что может замедлить разработку.  
  **Привязка к проекту**: JavaScript проще для быстрого старта.

**Итог**

* **Backend: Python**  
  Выбран благодаря простоте, асинхронности, богатой экосистеме и поддержке RESTful API. Go и Java избыточны для нашего проекта.
* **Frontend: JavaScript**  
  Выбран благодаря простоте, скорости разработки и отсутствию необходимости в строгой типизации. TypeScript не требуется для небольшого проекта.
  + 1. Архитектурный стиль и форматы данных

REST (Representational State Transfer) - это архитектурный стиль, который определяет набор принципов для построения распределенных систем в сети. REST основан на принципах, используемых в вебе, таких как клиент-серверная архитектура, отсутствие состояния (stateless), кэширование, единообразие интерфейса и многослойная структура.

REST позволяет создавать гибкие, масштабируемые и легко поддерживаемые веб-сервисы, которые могут быть использованы различными клиентами, включая веб-приложения, мобильные приложения и другие сервисы. REST позволяет разделить клиентскую логику и серверную логику, что упрощает разработку и поддержку приложений.

**Почему RESTful и JSON, а не SOAP/XML?**

1. **RESTful API**

* **Простота и гибкость**: REST базируется на стандартных HTTP-методах (GET, POST, PUT, DELETE), что упрощает разработку и интеграцию.
* **Масштабируемость**: RESTful архитектура идеальна для распределенных систем, так как не требует сохранения состояния (stateless), что упрощает балансировку нагрузки.
* **Совместимость с веб-стандартами**: REST легко интегрируется с современными инструментами (Swagger/OpenAPI для документации).
* **Привязка к проекту**: Ваше приложение — клиент-серверное с большим количеством запросов. REST обеспечивает предсказуемость и легкость масштабирования.

1. **JSON**

* **Легковесность**: JSON имеет меньший объем данных по сравнению с XML, что ускоряет передачу и парсинг.
* **Совместимость с JavaScript**: JSON нативно поддерживается JavaScript, что критически важно для frontend-части.
* **Читаемость**: Структура JSON проста для восприятия, что упрощает отладку и разработку.
* **Привязка к проекту**: Ваш frontend на React.js легко работает с JSON, а сервер на FastAPI автоматически сериализует данные в JSON.

1. **Почему не SOAP/XML?**

* **Сложность**: SOAP требует строгой схемы (WSDL) и сложной XML-структуры, что замедляет разработку.
* **Низкая производительность**: XML многословен, а парсинг требует больше ресурсов, что недопустимо для высоконагруженных систем.
* **Устаревание**: REST/JSON стал отраслевым стандартом для веб-сервисов, тогда как SOAP/XML чаще используется в корпоративных legacy-системах.

В целом, REST является широко используемым и популярным подходом для построения веб-сервисов и API в современной разработке программного обеспечения.

* + 1. Фреймворки и библиотеки
  1. Безопасность
     1. SQL инъекции
     2. Шифрование трафика

Любое действие в Интернете — это обмен данными. Каждый вбитый в поисковую строку запрос проходит путь от пользователя к серверу и обратно. акая коммуникация возможна благодаря работе протокола HTTP. Протокол HTTP не шифрует данные. Следовательно, их без труда может перехватить третья сторона, личная информация (пароль, номер банковской карты, реквизиты, паспортные данные) может быть украдена злоумышленниками.

В современном мире защита данных имеет принципиальное значение. Поэтому внедрили HTTPS, который расшифровывается как протокол безопасного соединения. Принципом работы защищенного протокола HTTPS является обмен ключами шифрования. На запрос браузера сервер отдает ответ, что соединение возможно через ключ шифрования. Браузер запрашивает SSL-сертификат и проверяет его в Удостоверяющем центре. Если сертификат валидный, то браузер и сервер обмениваются ключами шифрования и только после этого отрисовывается сама страница сайта через безопасное соединение. Так происходит каждую сессию, то есть каждый раз при обмене запросами и ответами. Таким способом и обеспечивается сохранность данных и конфиденциальность при обмене информацией.

Чтобы сайт стал работать по протоколу безопасного соединения HТТPS, нужен SSL-сертификат. Это виртуальный документ, который содержит данные об организации, её владельце и подтверждает их существование. Позволяет узнать сервер и подтвердить безопасность сайта.

Использование сертификата безопасности для сайта гарантирует:

* Подлинность ресурса, к которому обращается пользователь. Это повышает у посетителей уровень доверия.
* Целостность передаваемой информации. При транспортировке от сервера к браузеру данные не изменятся и не потеряются.
* Конфиденциальность. 256-разрядное шифрование исключает доступ злоумышленников к информации.

Что дает SSL-сертификат для сайта кроме защиты данных? [SSL-сертификат](https://www.reg.ru/ssl-certificate/) помогает в SEO-продвижении проекта — позволяет занять более высокую позицию в поисковой выдаче. Поисковые системы (Google, Яндекс и пр.) дорожат доверием аудитории и выше ранжируют сайты, которые работают через безопасное соединение.

* + 1. Аутентификация
    2. Подбор пароля. Алгоритм хэширования
    3. Контроль соединений
  1. Разработка серверной части (backend)
     1. Окружение
        1. Среда разработки
        2. Переменные окружения
        3. Редактор кода
        4. Зависимости
     2. Структура приложения
        1. Слой представления
        2. Сервисный слой
        3. Слой доступа к данным
     3. База данных
        1. Объектно-реляционное отображение (ORM)
        2. Модели
        3. Схемы
        4. Связи
        5. Подключение и транзакции
  2. Разработка клиентской части (frontend)
     1. Среда и инструменты разработчика
     2. Зависимости
     3. Структура и компоненты
     4. SSR и SEO-оптимизация
  3. Разработка сервиса для работы с данными
     1. Интеграция с информационным сервером Московской биржи
     2. Хранение обработка больших данных
  4. Документация
  5. Обработка исключений
  6. Логирование
  7. Тестирование
  8. Развертывание (deploy)
  9. Контейнеризация и оркестрация
  10. Непрерывная интеграция и непрерывное развертывание (CI/CD)

1. Список использованных источников и литературы.

**Архитектура, парадигма и принципы программирования**

* Роберт Мартин. Чистая архитектура. Искусство разработки программного обеспечения. - СПб.: Питер, 2024. – 352с.: ил. – (Серия «библиотека программиста»)
* Роберт Мартин. Чистый код. Создание, анализ и рефакторинг. СПб.: Питерб 2025. – 464 с.: ил.
* Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Паттерны объектно-ориентированного программирования. СПб.: Питер, 2024. – 448 с.: ил. – (Серия «библиотека программиста»)
* Фаулер Мартин. Шаблоны корпоративных приложений. Исправленное издание. : пер. с англ. – СПб. : ООО «Диалектика» 2020. – 544 с.: ил. Парал. тит. англ.
* Мартин Фаулер, Кент Бек, Джон Брант, Уильям Апдайк, Дон Робертс. Рефакторинг. Улучшение проекта существующего кода. : Пер. с англ. СПб.: ООО «Диалектика», 2022. – 448 с.: ил. Парал. тит. англ.
* Мартин Фаулер, Кент Бек. Рефакторинг кода на JavaScript: улучшение проекта существующего кода, 2-е издание.: Пер. с англ. СПб.: ООО «Диалектика», 2020. – 464 с.: ил. Парал. тит. англ.

**Основной фреймворк для разработки API**

* "FastAPI: Modern Python Web Development" (Б. Любанович),
* Lubanovic B. FastAPI. – " O'Reilly Media, Inc.", 2023.
* "Building Python Web APIs with FastAPI" (Адесина)
* Adeshina A. A. Building Python Web APIs with FastAPI: A fast-paced guide to building high-performance, robust web APIs with very little boilerplate code. – Packt Publishing Ltd, 2022.
* Voron F. Building Data Science Applications with FastAPI: Develop, manage, and deploy efficient machine learning applications with Python. – Packt Publishing Ltd, 2023.
* Официальная документация FastAPI: <https://fastapi.tiangolo.com/>

**Асинхронность**

* Официальная документация asyncio: <https://docs.python.org/3/library/asyncio.html>
* Фаулер М. Asyncio и конкурентное программирование на Python / пер.
* с англ. А.А. Слинкина. – М.: ДМК Пресс, 2022. – 398 с.
* Async IO in Python: A Complete Walkthrough – Real Python [Электрон-
* ный ресурс]. – URL: https://realpython.com/async-io-python/ (дата обращения:
* 12.04.2023).

**ORM и работа с данными**

* Официальная документация Pydantic: <https://pydantic-docs.helpmanual.io/>
* SQLAlchemy: "Essential SQLAlchemy" (Майерс, Копленд)
* Myers J., Copeland R., Copeland R. D. Essential SQLAlchemy. – " O'Reilly Media, Inc.", 2015.
* Официальная документация SQLAlchemy: <https://docs.sqlalchemy.org/>
* Официальная документация Alembic: <https://alembic.sqlalchemy.org/>
* Официальная документация Postgres: <https://www.postgresql.org/docs/>

**Развертывание**

* Docker: "Docker in Action" (Николофф)
* Официальная документация Docker: <https://docs.docker.com/>
* Nginx: "Nginx HTTP Server" (Неделку),
* Sharma R. NGINX high performance. – Packt Publishing, 2015.
* Fjordvald M. B., Nedelcu C. Nginx HTTP Server: Harness the power of Nginx to make the most of your infrastructure and serve pages faster than ever before. – Packt Publishing Ltd, 2018.
* Официальная документация nginx: <https://nginx.org/en/docs/>

**Безопасность**

* Что такое SQL-инъекция? Определение и описание: <https://www.kaspersky.ru/resource-center/definitions/sql-injection>
* Что такое SSL сертификат и для чего он нужен: <https://help.reg.ru/support/ssl-sertifikaty/obshchaya-informatsiya-po-ssl/chto-takoye-ssl-sertifikat-i-dlya-chego-on-nuzhen#0>

**Бенчмарки**

* FastAPI: <https://fastapi.tiangolo.com/benchmarks/>
* TechEmpower: <https://www.techempower.com/benchmarks/>

**Статьи**

* "Evaluating Web Frameworks" (2025), <https://www.researchgate.net/publication/389129746>
* Stack Overflow:
* Хабр:
* Real Python: <https://realpython.com/fastapi-python-web-apis/>

**Практика**

* GeekBrains – курс java разработчик: <https://gb.ru/>
* GeekBrains – курс разработчик (общий)
* GeekBrains – курс frontend разработка на REACT.js:
* Школа Артем Шумейко – курс по backend разработке <https://artemshumeiko.ru>
* Result University: <https://result.school/>

1. Приложения.