# Введение

Идея: Создание клиентоориентированного приложения для отслеживания потребляемой пищи. Приложение должно самостоятельно следить за потреблением килокалорий, микро- и макро-нутриентов, а также направлять пользователя в выборе продуктов питания.

Цель: Создать клиентоориентированное приложение для отслеживания потребляемой пищи, килокалорий, микро- и макро-нутриентов.

# Анализ рынка

Рассмотрим потенциальных конкурентов сервиса:

1. **FatSecret**

*Возможности приложения*:

Приложение позволяет записывать продукты и блюда, которые вы съедаете на завтрак, обед, ужин и перекус.

Отслеживает норму употребления калорий,

Формирует диаграмму БЖУ (белки, жиры и углеводы).

Можно установить цель по калориям и БЖУ, а также получить РСК, введя данные от физической активности и состоянии тела пользователя.

Также в FatSecret есть большая коллекция рецептов, блюда из которых можно добавлять в свой дневник по порциям, но не по граммам.

Приложение распознаёт еду по фотографии, приложение предлагает разные варианты его калорийности.

Также можно быстро добавлять в дневник продукты, сканируя штрих-код на упаковке.

Есть возможность отслеживать упражнения и сон, однако эта система имеет определенные недостатки.

*Платные возможности*:

Отслеживание потребления воды

Доступ к готовым планам питания, создание собственных планов питания

Дополнительные приемы пищи

Запоминание регулярно употребляемых продуктов

1. **MyFitnessPal**

*Возможности приложения*:

Основная цель— пищевой дневник. Приложение считает калории и соотношение белков, жиров и углеводов.

Приёма пищи: завтрак, обед, ужин, перекус. Блюдо можно выбрать из поиска или создать самому.

Отслеживать, сколько воды вы выпили в течение дня и сколько калорий сожгли, прогуливаясь или занимаясь спортом.

При первом входе понадобится выбрать цель («Похудеть», «Сохранить текущий вес» или «Набрать вес»), указать образ жизни («Не очень подвижный», «Малоподвижный», «Подвижный» или «Очень подвижный»), пол, возраст и страну, рост, текущий и желаемый вес, еженедельную цель по весу. Приложение само предложит вам количество калорий, которые нужно съедать в течение дня, — эту цифру можно изменить.

Можно сканировать штрих-код продукта и автоматически добавлять его пищевую ценность в ваш дневник (если продукт есть в базе приложения).

По геолокации можно выбрать заведение, в котором вы заказали блюдо и, если его меню есть в базе приложения, калории также подгрузятся автоматически.

*Платные возможности*:

Настраивать сводку пищевой ценности

Анализировать блюда по макроэлементам

Загружать в приложение CSV-файлы с данными о вашем прогрессе, приёмах пищи и упражнениях

Смотреть все отчёты за предыдущее время, устраивать цели на день.

1. **Yazio**

*Возможности приложения*:

Приложение отслеживает общую калорийность питания за день, баланс белков, жиров и углеводов и сожжённые калории (Yazio может брать информацию о пройденных шагах из других приложений, также можно добавлять активности вручную, например «Бег», «Уборка, работа по дому» и другие).

Позволяет контролировать количество выпитой за день воды — можно установить цель и придерживаться её.

Перед началом использования нужно пройти короткую регистрацию и указать цель («Потеря веса», «Нарастить мышцы», «Поддержание веса»), исходный и желаемый вес, пол, возраст и рост, уровень активности. Приложение само подскажет, сколько калорий вам нужно будет съедать каждый день, — эти данные можно корректировать вручную.

Из удобных фишек: можно сканировать штрих-коды продуктов, чтобы быстрее добавлять их в дневник (если штрих-код отсутствует в базе приложения, продукт можно загрузить вручную и поделиться им с другими пользователями).

Также еду, которую вы часто употребляете, можно добавлять в «Избранное».

*Платные возможности*:

Можно получить доступ к базе рецептов

Создавать собственный план питания (или загружать те, что есть в приложении) и добавлять диеты.

1. **Grow food**

Сервис ежедневной доставки готовой еды, который полностью берет на себя заботу о подсчете КБЖУ, составлении рациона, приготовлении пищи и доставке. Приложение позволяет подписаться на приготовление и доставку еды.

*Возможности приложения*:

Выбор цели пользования приложением (Экономия времени, похудение, быть в форме)

После выбора цели, можно определить тип питания из предложенных, количество дней доставки, дату первой доставки.

Пользователь может просмотреть предложенные рационы питания.

А также редактировать его, добавлять и заменять приемы пищи.

*Платные возможности*:

Доставка еды, можно оформить семейный тариф.

**Итог:**

Общие свойства конкурентов:

* Отслеживание съеденной за сутки еды (приемы пищи разделены на завтрак, обед, ужин, перекус)
* Добавление дневных активностей
* Сканировать штрих-коды продуктов
* Выбор цели пользования приложением
* Анализ питания пользователя (СНК, БЖУ)
* Есть база рецептов, их можно создавать самому
* Отслеживание потребления воды
* Запоминание регулярно употребляемых продуктов
* Приложение предлагает готовые рационы питания
* Составлении своего плана питания

Уникальные фичи конкурентов:

* Приложение позволяет подписаться на приготовление и доставку еды
* Загружать в приложение CSV-файлы с данными о вашем прогрессе, приёмах пищи и упражнениях
* Можно выбрать заведение, в котором вы заказали блюдо и, если его меню есть в базе приложения, калории также подгрузятся автоматически
* Создание дополнительных приемов пищи
* Распознавание еды по фотографии

Стратегия разработки**:**

Таким образом, можно сформировать стратегию разработки сервиса:

Рынок полон подобных сервисов, благодаря чему можно определить базовый набор функций, которые должен выполнять сервис. Однако из-за высокой конкуренции помимо базового функционала, приложение должно предлагать уникальные особенности, которые позволят выделить наш продукт на фоне конкурентов. Например, использование трендовых, современных технологий. Одной из них сегодня является *Искусственный Интеллект*, которая позволяет решать различные задачи. Найти применение этой технологии можно где угодно, и это будет значительным плюсом к репутации приложения, поэтому проект должен использовать ее.

Непосредственно стратегия:

1. Анализ целевой аудитории, выявление требований клиентов, а также составление требований исходя из ситуации на рынке.
2. Составление списка функций, которые должно выполнять приложение.
3. Составление диаграмм.
4. Разработка полной архитектуры системы.
5. Непосредственно разработка системы
6. Тестирование осуществляется параллельно разработке и после нее. Для финального тестирования, будет определены группа для закрытого тестирования.
7. Поддержка продукта.

# Требования к продукту:

Исходя из ситуации на рынке, можно сформировать какой функционал должно предлагать наше приложение. Базовый набор функций приложения:

* Отслеживание съеденной за сутки еды. Употребленная еда привязывается ко времени, что позволяет пользователю есть в любое время суток и делать сколько угодно приемов пищи. Также благодаря такому обилию информации, мы можем подробнее анализировать рацион пользователя. Например, ближе к концу суток, стоит исключить потребление продуктов с высоким ГИ.
* Продукты могут быть привязаны к конкретному производителю.
* Анализ КБЖУ и микронутриентов дневного рациона
* Сканировать штрих-коды продуктов для добавления в дневной рацион
* Выбор цели пользования приложением
* Анализ питания пользователя (СНК, БЖУ, ГИ продуктов)
* Есть база рецептов, их можно создавать самому
* Возможность добавлять составные блюда (Как рецепты, но содержит только список ингредиентов)
* Добавление продуктов/рецептов в избранное
* Доступ к базе планов питания
* Возможность составить свой план питания

Уникальные фичи приложения, которые позволят ему выделиться на фоне конкурентов:

* Приложение позволяет совместить доставку еды из популярных сетей ресторанов и магазинов с отслеживанием дневного рациона.
* Составление индивидуального рациона Искусственным Интеллектом
* Выявление недостатков рациона Искусственным Интеллектом, выдача пользователю советов по питанию.
* Отслеживание ГИ употребленных продуктов

## Целевая аудитория:

Определим целевую аудиторию проекта по нескольким критериям:

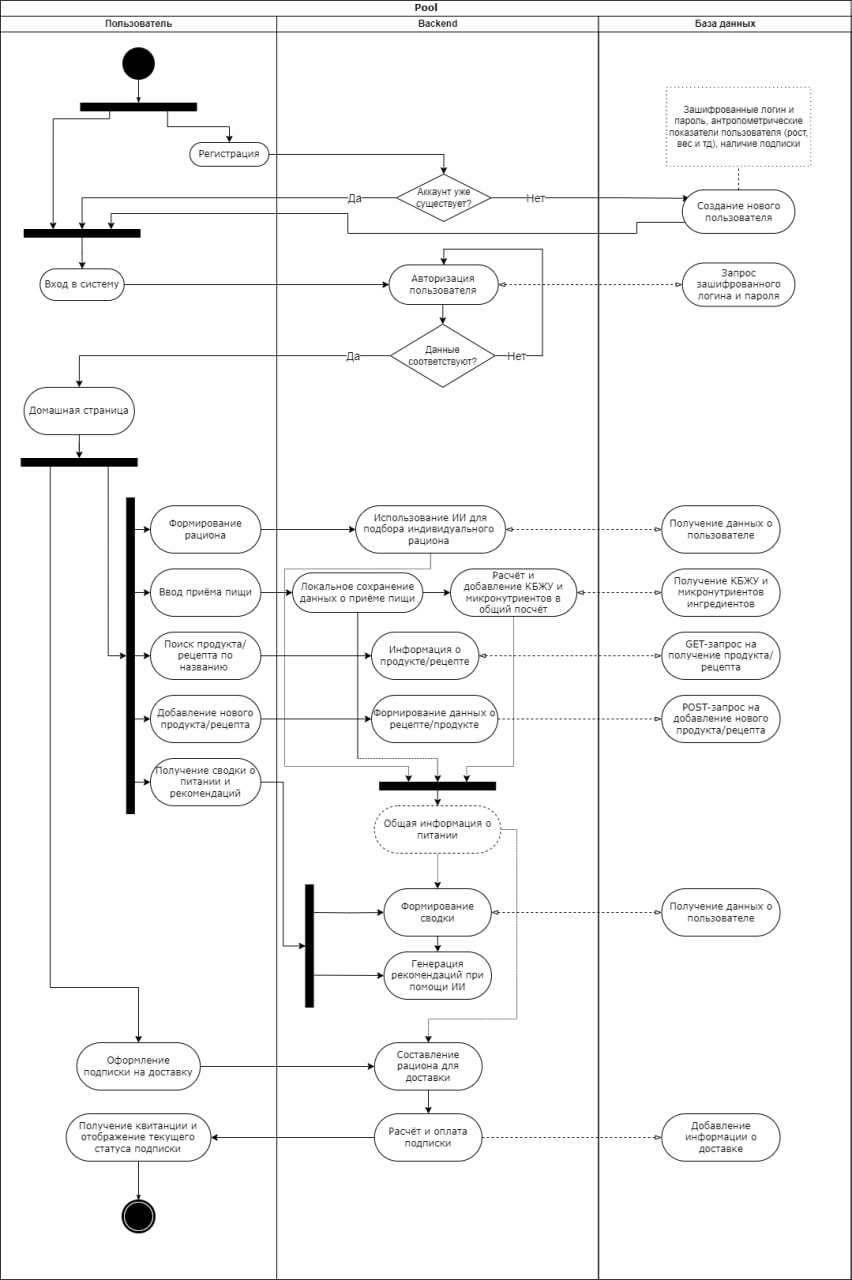
1. Возраст: от 16 до 45 лет, однако может быть и выше;
2. Нацелен как на мужчин, так и на женщин;
3. Для людей на похудении, наборе веса, контролирующих свой вес, а также занимающихся фитнесом;
4. Сфера деятельности не имеет значения;
5. Среди аудитории могут быть школьники, студенты, трудоустроенные и самозанятые;
6. Уровень дохода не имеет значения, т.к. основные функции будут бесплатными, а отключение рекламы и дополнительный функционал будет подключаться по доступной цене;
7. Также приложение можно распространять по любым регионам, а в будущем и странам.

## USER STORY

1. Как пользователь, я хочу иметь возможность отмечать и отслеживать все приемы пищи за день, чтобы видеть полную картину своего рациона.
2. Как пользователь, я хочу, чтобы приложение предлагало адаптированный план питания, учитывая мои предпочтения и ограничения.
3. Как пользователь, я хочу получать уведомления о нехватке определенных нутриентов в моем рационе.
4. Как пользователь, я хочу иметь возможность сохранять любимые рецепты и добавлять их в свой рацион.
5. Как пользователь, я хочу, чтобы приложение предоставляло анализ изменений в моем весе и композиции тела.
6. Как пользователь, я хочу получать рецепты и рекомендации для приготовления блюд, соответствующих моим диетическим ограничениям.
7. Как пользователь, я хочу иметь возможность оформить подписку на доставку еды через какой-либо сервис внутри приложения.
8. Как пользователь, я хочу иметь возможность устанавливать цели и отслеживать свой прогресс в их достижении.
9. Как пользователь, я хочу, чтобы приложение предоставляло информацию о сезонности продуктов и их полезных свойствах.
10. Как пользователь, я хочу иметь возможность экспортировать отчеты о своем питании для обсуждения с врачом или диетологом.

# Проектирование

**Activity Diagram**



## Бэкэнд

Для архитектуры бэкенда были выбраны Python, FastAPI, PostgreSQL, SQLModel и SQLAlchemy

### Стек

1. FastAPI: FastAPI — это современный, быстрый (и высокопроизводительный) веб-фреймворк для создания API на Python 3.7+, использующий стандартные типы Python для валидации, что упрощает его использование с SQLModel и SQLAlchemy. Он прост в использовании и его легко изучить.

2. PostgreSQL: PostgreSQL — это мощная объектно-реляционная система баз данных с открытым исходным кодом. Она обладает высокой расширяемостью и обеспечивает поддержку различных типов данных, индексации и сложных запросов.

3. SQLModel: SQLModel — это библиотека для взаимодействия с базами данных SQL из кода на языке Python с использованием объектов. По сути, она является оберткой поверх Pydantic и SQLAlchemy, что упрощает работу с ними.

4. SQLAlchemy: SQLAlchemy — это ORM для работы с реляционными базами данных в Python. Она предоставляет возможность работать с запросами к базам данных на уровне объектов в Python (валидацию данных в которых берёт на себя SQLModel)

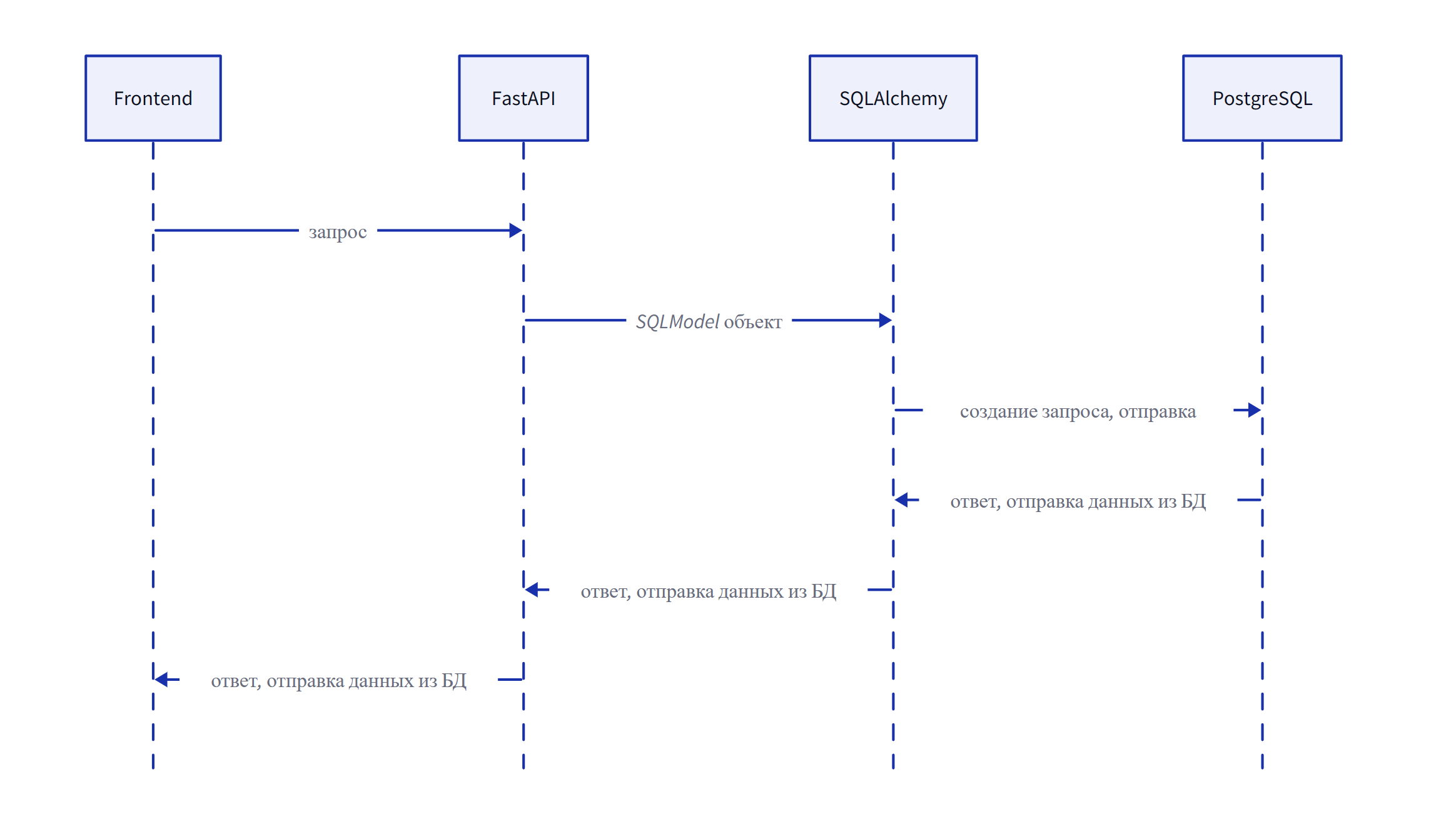
### Диаграмма последовательностей

Frontend -> FastAPI: запрос

FastAPI -> SQLAlchemy: SQLModel объект

SQLAlchemy -> PostgreSQL: создание запроса, отправка

PostgreSQL -> SQLAlchemy -> FastAPI -> Frontend: ответ, отправка данных из БД



### Примерная структура бэкэнда

project/

│

├── app/

│ ├── main.py

│ ├── db.py

│ ├── models/

│ │ ├── \_\_init\_\_.py

│ │ └── models.py

│ ├── schemas/

│ │ ├── \_\_init\_\_.py

│ │ └── schemas.py

│ └── crud/

│ ├── \_\_init\_\_.py

│ └── crud\_operations.py

│

├── alembic/

│ ├── versions/

│ │ └── ...

│ └── env.py

│

├── tests/

│ └── ...

│

├── requirements.txt

└── main.py

Что за что отвечает:

app/: основной каталог

main.py: инициализирует приложение FastAPI и включает маршрутизаторы

db.py: конфигурация и настройка базы данных с использованием SQLAlchemy

models/: модели баз данных.

models.py: модели SQLAlchemy для таблиц базы данных

schemas/: модели Pydantic для проверки запросов и ответов

schemas.py: модели Pydantic для проверки запросов и ответов

crud/: CRUD-операции (Create, Read, Update, Delete) для взаимодействия с базой данных

crud\_operations.py: CRUD-операции для базы данных

alembic/: скрипты миграции Alembic для управления изменениями схемы базы данных

tests/: тестовые файлы для приложения

requirements.txt: все Python-зависимости, необходимые для проекта

main.py: точка входа в приложение

Пример кода

python

main.py

from fastapi import FastAPI

from .db import engine

from .models import Base

from .routers import router

app = FastAPI()

Create tables on startup

Base.metadata.create\_all(bind=engine)

app.include\_router(router)

python

db.py

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

SQLALCHEMY\_DATABASE\_URL = "postgresql://user:password@postgresserver/db"

engine = create\_engine(SQLALCHEMY\_DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

### Плюсы и минусы

Плюсы:

1. FastAPI — это высокопроизводительный веб-фреймворк, обеспечивающий производительность на уровне NodeJS и GO. SQLModel и SQLAlchemy обеспечивают эффективный и высокопроизводительный доступ к базам данных.
2. Удобный дизайн API, автоматическая проверка данных и функции подготовки документации позволяют быстро и эффективно создавать API с помощью FastAPI
3. Модульность: Использование SQLModel и SQLAlchemy обеспечивает модульность (можно изменить объект SQLModel и SQLAlchemy продолжит с ним нормально работать), что облегчает сопровождение и масштабирование приложения
4. “Питоничность”: Использование типов Python и моделей Pydantic для проверки запросов и ответов делает код более “питоничным” и более простым в написании и сопровождении

Минусы:

1) FastAPI требует изучения, особенно для тех, кто не знаком с аннотациями типов Python и асинхронным программированием. Асинхронное программирование сложнее в написании и сопровождении, чем синхронное

2) Недостаточная безопасность: FastAPI не обеспечивает безопасность напрямую, а полагается на пакет fastapi.security

Источники:

https://www.educative.io/answers/what-is-fastapi

https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison

https://www.imaginarycloud.com/blog/flask-vs-fastapi/

https://christophergs.com/tutorials/ultimate-fastapi-tutorial-pt-8-project-structure-api-versioning/

https://testdriven.io/blog/fastapi-sqlmodel/

https://fastapi.tiangolo.com/tutorial/sql-databases/

https://sqlmodel.tiangolo.com/tutorial/fastapi/update/

https://github.com/tiangolo/full-stack-fastapi-postgresql

## RESTful API

1. /product (GET) Поиск продуктов по названию;
   1. Тело запроса:

* Название продукта
  1. Тело ответа:
* Список из продуктов, подходящих по названию

{

[

id: int,

name: string,

]

}

1. /product/id (GET) Получение информации о продукте;
   1. Тело запроса:

* ID продукта
  1. Тело ответа:
* Информация о продукте

1. /recipe (GET) Поиск продуктов (рецептов) по названию;
   1. Тело запроса:

* Название рецепта
  1. Тело ответа:
* Список из продуктов, подходящих по названию

{[

id: Int,

name: String,

]}

1. /recipe/id (GET) Получение информации о рецепте;
   1. Тело запроса:

* ID рецепта
  1. Тело ответа:
* Информация о рецепте

1. /cpfc (GET) Расчёт КБЖУ и микронутриентов блюда по ингредиентам;

Тело запроса:

* Состав блюда
  1. Тело ответа:
* Информация о блюде

1. /product (POST) Добавление нового продукта в общую базу данных;
   1. Тело запроса:

* Пищевая ценность продукта
  1. Тело ответа:
* ID продукта

1. /receipe (POST) Добавление нового рецепта в общую базу данных;
   1. Тело запроса:

* Информация о рецепте
  1. Тело ответа:
* ID рецепта

1. /product/id (DELETE) Удаление продукта из общей базы данных;
   1. Тело запроса:

* ID продукта
  1. Тело ответа:
* Результат (String)

1. /receipe/id (DELETE) Удаление рецепта из общей базы данных;
   1. Тело запроса:

* ID рецепта
  1. Тело ответа:
* Результат (String)

1. /dci (GET) Расчёт СНК (Суточная норма калорий);
   1. Тело запроса:

* Информация о человеке
  1. Тело ответа:
* ccal: Int

1. /rda (GET) Расчёт РСК (Рекомендуемое суточное количество);
   1. Тело запроса:

* RDA request body
  1. Тело ответа:
* Diet goal

1. /analize (GET) - Анализ дневного/недельного/месячного рациона;
   1. Тело запроса:

{

diet: Diet

goal: diet goal

}

* 1. Тело ответа:

{

ccal\_control: Int

carbo\_control: Float

protein\_control: Float

fat\_control: Float

micro\_nutrients: [Float]

}

1. /diet (GET) - Формирование рациона;
   1. Тело запроса:

* Diet goal
  1. Тело ответа:
* Diet

1. /deliver (POST) - Оформление подписки на доставку;
   1. Тело запроса:

{

diet: Diet

budget: Float

deliver\_time: BigInt

}

* 1. Тело ответа:
* Результат (String)

1. /deliver (GET) - Проверить подписку на доставку;
   1. Тело запроса:

* ID пользователя
  1. Тело ответа:

{

diet: Diet

budget: Float

deliver\_time: BigInt

}

1. /deliver (DELETE) – Отменить подписку на доставку;
   1. Тело запроса:

* ID пользователя
  1. Тело ответа:
* Результат (String)

1. /deliver (PUT) – Отредактировать доставку
   1. Тело запроса:

* ID пользователя
* Diet
  1. Тело ответа:
* Результат (String)

JSONы

* Пищевая ценность еды:

{

ccal: Int,

carbo: Float,

protein: Float,

fat: Float,

micro\_nutrients: [Float]

}

* Состав еды:

{

Ingredients {[

product\_id: Int

portion: Int

]}

}

* Информация о продукте

{

Id: Int

nutritional\_value

}

* Информация о блюде

{

Id: Int

ingredients(состав)

nutritional\_value(пищевая ценность)

}

* Информация о рецепте

{

Id: Int

ingredients(состав)

nutritional\_value(пищевая ценность)

instruction {[

description: String

]}

cook\_time: Int

prepare\_time: Int

}

* Информация о человеке

{

id: Int

weight: Float

height: Int

activity\_level: Float

}

* RDA request body

{

human: информация о человеке

goal: diet goal

}

* Diet goal

{

days: Int

weight\_delta: float

ccal\_delta: Int

}

* Diet

{

food: [

{

time: BigInt

food: Информация о рецепте/

Информация о блюде/

Информация о продукте

portion: Int

}

]

}

# Тестирование

## REST API

• GET /product/id – нужно добавить тело ответа для случая, если по данному id не найден продукт

• GET /recipe/id – нужно добавить тело ответа для случая, если по данному id не найден рецепт

• GET /cpfc - нужно добавить тело ответа для случая, если по данному id не будет найден ингредиент

• POST /recipe, POST /product, DELETE /product/id, DELETE /recipe/id – в теле запроса нет id, по которому можно будет определить, от кого исходит запрос. Любой человек в сети сможет менять данные в ЬД.

• POST /product – в теле запроса должна быть информация о продукте.

• DELETE /reciepe/id, DELETE /product/id – нужно продумать, какими правами нужно обладать, чтобы удалить продукт или рецепт в общую базу данных

• JSON “Информация о блюде” не нужен, уже есть “Информация о продукте” и “Информация о рецепте”

• GET /diet - предусмотреть тело ответа для невыполнимого случая, когда пользователь хочет похудеть на 20 кг за 1 день (например)