

Государственное учреждение образования
«Белорусский государственный университет информатики и
радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ОТЧЕТ
по лабораторной работе № 3
«Исследование архитектурного решения»

Студент:

М.О. Прозоров
Н.А. Стук
Д.В. Миленкевич

Преподаватель:

О.М. Внук

Минск 2026

1. ЦЕЛЬ

Целью данной лабораторной работы будет проектирование и анализ архитектуры приложения.

2. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

2.1 Проектирование архитектуры

ТИП ПРИЛОЖЕНИЯ

В ходе выбора темы проекта было рассмотрено множество типов приложений. Применительно к нашему приложению CookHelper было принято решение разрабатывать Веб-приложение. Приложения этого типа, как правило, поддерживают сценарии с постоянным подключением и различные браузеры, выполняющиеся в разнообразнейших операционных системах и на разных платформах.

Помимо этого, главным преимуществом именно Веб-приложения является широкая доступность и поддержка на различных платформах. Минусы тоже есть: сетевое подключение, сложность обеспечения пользовательского интерфейса.

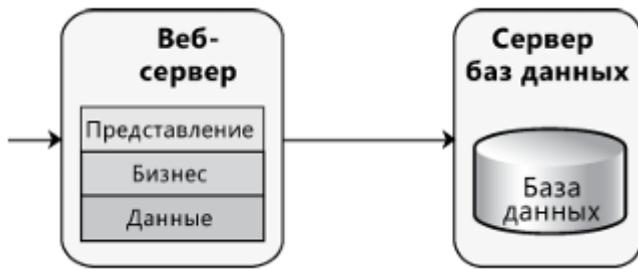
Веб-приложение – приложение, которое может использоваться пользователями через Веб-браузер или специализированный агент пользователя. Браузер создает HTTP-запросы к определенным URL, которые сопоставляются с ресурсами на Веб-сервере. Сервер формирует визуальное представление HTML-страниц, которое может быть отображено браузером, и возвращает их клиенту. Ядро Веб-приложения – это его логика на стороне сервера. Такое приложение может состоять из множества отдельных слоев.

ВЫБОР СТРАТЕГИИ РАЗВЕРТЫВАНИЯ

При развертывании Веб-приложения необходимо учесть то, как размещение слоев и компонентов будет влиять на производительность, масштабируемость и безопасность приложения. Возможно, придется пойти на некоторые компромиссы. В зависимости от требований и ограничений инфраструктуры используйте либо распределенное, либо нераспределенное развертывание. Нераспределенное развертывание, как правило, приводит к повышению производительности за счет снижения числа вызовов через физические границы. Однако распределенное развертывание позволит

обеспечить лучшую масштабируемость и реализовать защиту каждого слоя в отдельности.

Применительно к нашему приложению используем **нераспределенное развертывание**. В сценарии нераспределенного развертывания все логически разделенные слои Веб-приложения, кроме базы данных, физически располагаются на одном Веб-сервере.



Использование нераспределенного развертывания обусловлено в первую очередь тем, что наше Веб-приложение чувствительно к производительности. Использование локальных вызовов между слоями не будет иметь для приложения столь пагубного влияния на производительность.

ВЫБОР ТЕХНОЛОГИИ

Для разработки основной логики выбрана микросервисная архитектура на базе фреймворка FastAPI и языка программирования Python. Выбор обусловлен следующими характеристиками:

- высокая производительность и асинхронность;
- строгая типизация и автоматическая валидация данных;
- использование OpenAPI;
- наличие и поддержка механизмов кэширования.

Строгое разделение frontend и backend с помощью использования RestAPI. FastAPI поощряет архитектуру без сохранения состояния сеанса на сервере, что позволяет применять горизонтальное масштабирование.

Для слоя доступа к данным будет использована база данных PostgreSQL, что позволит хранить данные и выступать их источником наряду с данными, получаемыми из внешних API.

ПОКАЗАТЕЛИ КАЧЕСТВА

Показатели качества – это общие факторы, оказывающие влияние на поведение во время выполнения, дизайн системы и взаимодействие с

пользователем. Они представляют функциональные области, потенциально влияющие на все приложение со всеми его слоями и уровнями. Некоторые из этих показателей касаются общего дизайна системы, тогда как другие относятся только ко времени выполнения, времени проектирования или связаны только с вопросами взаимодействия с пользователем.

Степень, с которой приложение реализует желаемое сочетание показателей качества, таких как удобство и простота использования, производительность, надежность и безопасность, свидетельствует об успешности дизайна и общем качестве программного приложения. При выполнении любого из требований показателей качества во время проектирования приложений важно учесть, какое влияние это может оказать на другие требования. Необходимо проанализировать соотношение выгод и потерь для совокупности множества показателей качества.

В рамках нашего проекта будет сделан упор на следующие показатели:

1) Качество дизайна:

Концептуальная целостность – определяет согласованность и связность дизайна в целом. Сюда относится то, как спроектированы компоненты или модули, а также такие факторы, как стиль написания кода и именования переменных.

Удобство и простота обслуживания – способность системы изменяться. Это касается изменения компонентов, сервисов, функций и интерфейсов при добавлении или изменении функциональности, исправление ошибок и реализации новых требований.

2) Качество времени выполнения:

Производительность – показатель, характеризующий скорость, с которой система выполняет любое действие за промежуток времени. Производительность измеряется в показателях задержки или пропускной способности. Задержка – это время, необходимое для ответа на любое событие. Пропускная способность – это число событий, имеющих место в заданный промежуток времени.

Масштабируемость – это способность системы справляться с увеличением нагрузки без влияния на ее производительность или способность легко расширяться. Существует два метода улучшения масштабируемости: вертикальное масштабирование (scale up) и горизонтальное масштабирование (scale out). Для обеспечения вертикального масштабирования в систему добавляется больше ресурсов, а для обеспечения горизонтального масштабирования в ферме, выполняющей приложение и распределяющей нагрузку, используется большее количество компьютеров.

Безопасность – это способность системы предотвращать злонамеренные или случайные действия, не предусмотренные при проектировании, или не допускать разглашение или утрату данных. Безопасная система должна защищать ресурсы и предотвращать несанкционированные изменения данных.

3) Качества системы:

Тестируемость – это мера того, насколько просто создать критерий проверки для системы и ее компонентов и выполнить эти тесты, чтобы определить, отвечает ли система данному критерию. Хорошая тестируемость означает большую вероятность того, что сбои в системе могут быть своевременно и эффективно изолированы.

4) Качества взаимодействия с пользователем:

Удобство и простота использования определяет, насколько приложение соответствует требованиям пользователя и потребителя с точки зрения понятности, простоты локализации и глобализации, удобства доступа для пользователей с физическими недостатками и обеспечения хорошего взаимодействия с пользователем в общем.

СКВОЗНАЯ ФУНКЦИОНАЛЬНОСТЬ

В большинстве проектируемых приложений имеется общая функциональность, которую нельзя отнести к конкретному слою или уровню. Обычно это такие операции как аутентификация, авторизация, кэширование, связь, управление исключениями, протоколирование и инструментирование, а также валидация. Эти функции называют сквозной функциональностью (crosscutting concerns), потому что они оказывают влияние на все приложение и, по возможности, должны реализовываться централизованно. Применительно к нашему проекту будем использовать следующие функции:

Аутентификация – безопасность от атак с подделкой пакетов, перехватов сеансов и других типов атак, обеспечивает надежность приложения.

Авторизация – защита от разглашений сведений, повреждений и подделки данных.

Кэширование – улучшение производительности и времени отклика приложения за счет оптимизации поиска используемых данных, сокращая количество обращений к сети и предотвращение ненужной или повторной обработки.

Сетевое взаимодействие – обеспечивает взаимодействие компонентов разных слоев и уровней.

Управление конфигурацией – повышение безопасности и надежности работы приложения.

Управление исключениями – позволяет корректно обрабатывать ошибки и предотвращать их распространение, не нарушая работу приложения и не вызывая глобальных сбоев.

Протоколирование – фиксирует ключевые события, ошибки, запросы и технические параметры работы системы, обеспечивая возможность мониторинга, диагностики и последующего анализа поведения приложения.

СТРУКТУРНАЯ СХЕМА ПРИЛОЖЕНИЯ

Типовая структура Веб-приложения представлена на рисунке 1.

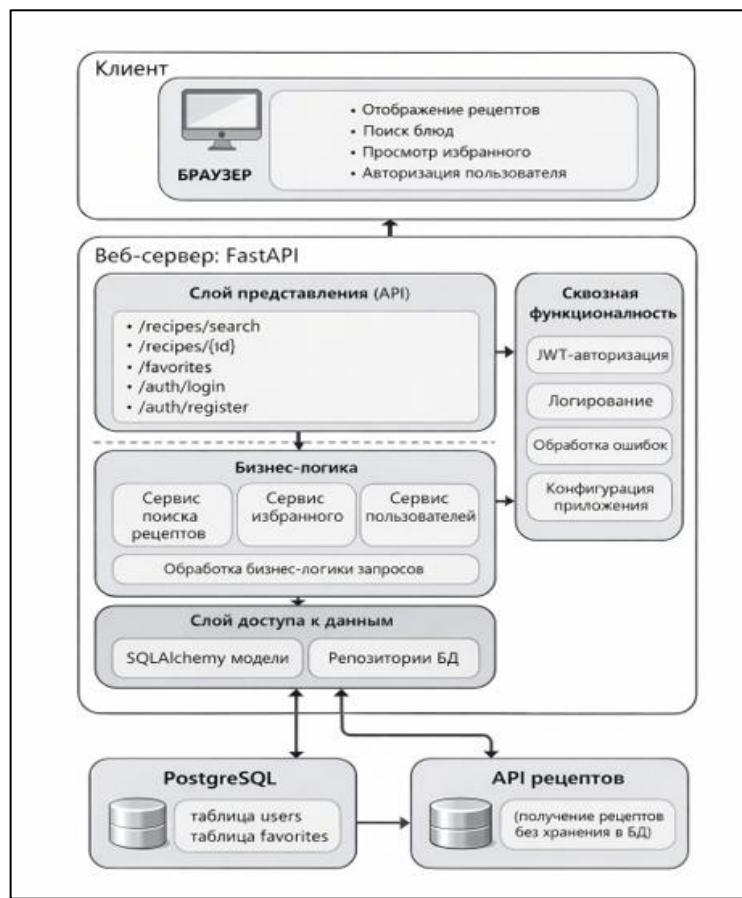


Рисунок 1 – Типовая архитектура Веб-приложения
(Архитектура To Be)

Архитектура представляет нам модель клиент-сервер. Клиенту, он же пользователь, через браузер могут быть доступны следующие функции: отображение рецептов, поиск блюд, просмотр избранного, авторизация пользователя.

Веб-сервер будет реализован с помощью «фрэймворка» FastAPI, преимущества которого были описаны ранее подразделе о выборе технологии для разработки. В рамках данной типовой структуры можно выделить следующие слои:

Слой представления. Данный слой содержит ориентированную на пользователя функциональность, которая отвечает за реализацию взаимодействия пользователя с системой, и, как правило, включает в себя компоненты, обеспечивающие общую связь с основной бизнес-логикой, инкапсулированной в бизнес-слое. Иными словами, это «фронтенд» на стороне сервера. Он включает в себя следующие конечные точки, обеспечивающие доступ к функциональности приложения:

- /recipes/search – поиск рецептов;
- /recipes/{id} – получении информации о рецепте;
- /favorites – работа с избранных;
- /auth/login – авторизация;
- /auth/register – регистрация.

Бизнес-слой. Этот слой реализует основную функциональность системы и инкапсулирует связанную с ней бизнес-логику. Бизнес-слой включает в себя следующую логику:

- сервис поиска рецептов;
- сервис управления избранным;
- сервис работы с пользователем;
- обработка бизнес-правил и валидация.

Слой доступа к данным. Этот слой обеспечивает доступ к данным, хранящимся в рамках системы, и данным, предоставляемым другими сетевыми системами. Доступ может осуществляться через сервисы. Слой данных предоставляет универсальные интерфейсы, которые могут использоваться компонентами бизнес-слоя. Он включает в себя репозитории и SQLAlchemy (описание таблиц и сущностей)

По поводу **сквозной функциональности** и ее значимости было сказано выше. Более подробно она включает в себя

Под **источниками данных и сервисами** можно понимать все компоненты, откуда наше приложение получает информацию: базы данных (PostgreSQL), внешние API, сторонние сервисы.

2.2 Анализ архитектуры

Ниже на рисунке 2-4 представлена диаграмма классов разрабатываемого приложения.

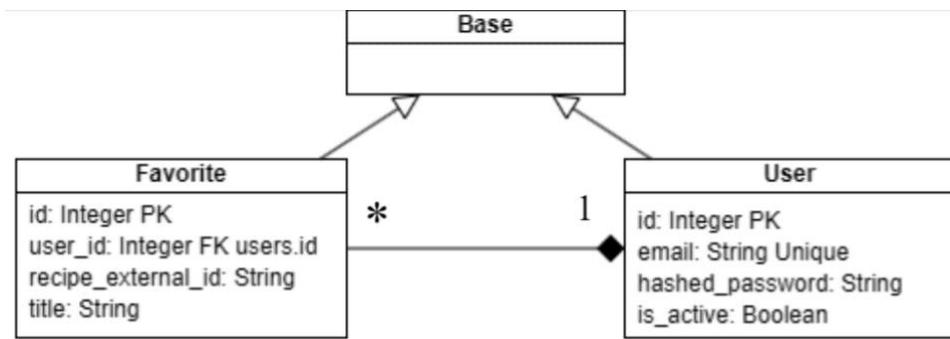


Рисунок 2 – Диаграмма классов веб-приложения. Слой данных

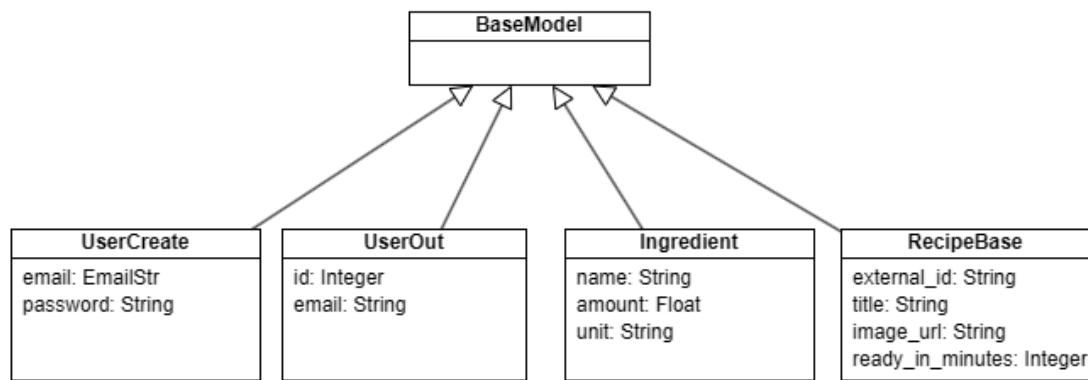


Рисунок 3 – Диаграмма классов веб-приложения. Слой валидации

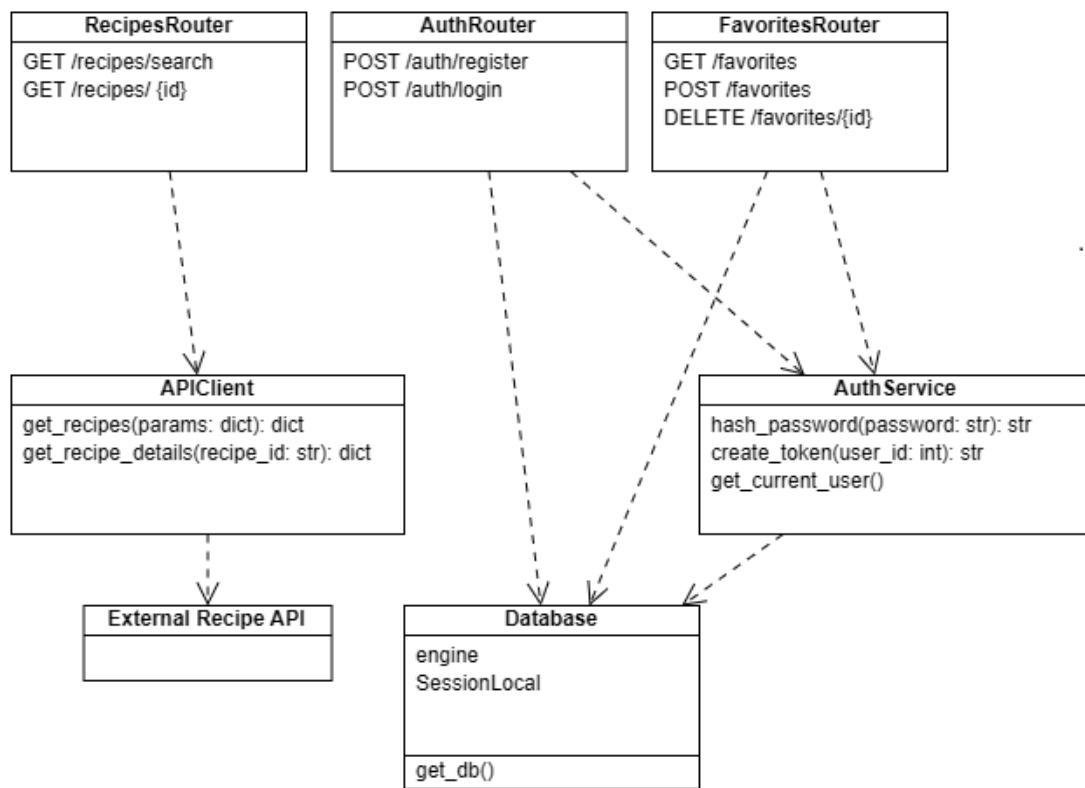


Рисунок 4 – Диаграмма классов веб-приложения. Слой логики и связи

2.3 Сравнение и рефакторинг

1) Архитектура To Be представляет собой целостное представление системы как единого рабочего механизма. В ней чётко выделены слои: слой представления, бизнес-слой, сквозная функциональность и слой доступа к данным. Такая структура делает архитектуру понятной и читаемой для разработчика, поскольку она отображает общую логику работы приложения, не углубляясь в детали реализации. Архитектура To Be показывает, как взаимодействуют клиент, сервер, база данных и внешние сервисы, формируя полную картину системы.

Архитектура As Is, напротив, фокусируется на детализации внутренней реализации FastAPI-приложения. Она описывает модели, схемы, сервисы и роутеры, но не демонстрирует систему как единое целое. Границы слоёв в As Is размыты: сервисы одновременно содержат бизнес-логику, обращаются к базе данных и взаимодействуют с внешними API. В результате система выглядит как набор отдельных проработанных блоков, а не как структурированная архитектура.

2) Архитектура As Is отражает текущее состояние кода и ориентирована на разработчика backend-части. Её цель — показать, как устроены модели, схемы, сервисы и роутеры, какие зависимости существуют между ними. Поэтому она детализирована, но не охватывает систему целиком. Такой подход естественен для этапа разработки, когда внимание сосредоточено на конкретных компонентах и их реализации.

Архитектура To Be, напротив, описывает систему на уровне высокоуровневого проектирования. Она показывает, как взаимодействуют клиент, сервер, бизнес-логика, база данных и внешние сервисы. В To Be-архитектуре выделены слои и зоны ответственности, что позволяет увидеть систему как единый механизм. Это важно для анализа, масштабирования и дальнейшего развития проекта.

3) Для перехода от начальной архитектуры As Is к уже полноценной структуре веб-приложения необходимо выполнить следующие шаги:

- четкое разделение слоев;
- введение слоя репозиториев;
- выделение сквозной функциональности;
- формализация работы с API;
- масштабирование.

3. ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были выполнены следующие задачи:

- ознакомились с документацией от Microsoft по проектированию различных приложений.

- на основе полученных теоретических сведений разработали и построили архитектуры As Is и To Be. Определили тип приложения, технологию, стратегию развертывания, показатели качества, обозначили пути реализации сквозной архитектуры.

- сравнили и проанализировали полученный диаграммы.

Весь ход выполнения лабораторной работы был отражен в отчете.