Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Жизненный цикл разработки программного обеспечения

ОТЧЕТ
по лабораторной работе № 3
«Исследование архитектурного решения»

Студенты: П.Е. Казаченко
А.Г. Слинько
И.Д. Закоркин
К.С. Пигулевский

Преподаватель: Д.А. Жалейко

МИНСК 2025

ВВЕДЕНИЕ

В данной лабораторной работе проводится исследование архитектурного решения для разрабатываемой системы. Архитектура программного обеспечения играет важную роль в создании надежных, масштабируемых и поддерживаемых приложений, определяя структуру системы, взаимодействие её компонентов и обеспечивая выполнение функциональных и нефункциональных требований. В рамках работы будут рассмотрены как теоретические аспекты проектирования архитектуры, так и практические шаги по анализу и улучшению существующего решения.

Работа разделена на три основные части. В первой части основное внимание уделяется проектированию архитектуры системы на высоком уровне абстракции. Будут изучены теоретические основы, описанные в «Руководстве Microsoft по моделированию приложений», а также определены ключевые аспекты, такие как тип приложения, стратегия развёртывания, выбор технологий, показатели качества и пути реализации сквозной функциональности. Результатом этой части станет структурная схема приложения, представленная в виде функциональных блоков или диаграмм UML, которая отражает архитектуру «To Be».

Во второй части работы проводится анализ существующей архитектуры на основе реального кода, используемого в системе. С помощью автоматизированных средств обратной инженерии будут сгенерированы диаграммы классов, отражающие текущее состояние системы. Это позволит получить представление об архитектуре «As Is» и выявить её особенности.

Третья часть работы посвящена сравнению архитектур «As Is» и «To Be». На основе выявленных различий будут предложены пути улучшения архитектуры, учитывающие принципы проектирования, архитектурные стили и шаблоны. Это позволит не только проанализировать текущее состояние системы, но и наметить направления для её дальнейшего развития и оптимизации.

1 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ

1.1 Определения типа приложения

Выбор соответствующего типа приложения – ключевой момент процесса проектирования приложения. Этот выбор определяется конкретными требованиями и ограничениями среды. От многих приложений требуется поддержка множества типов клиентов и возможность использования более одного базового архетипа.

Для реализации онлайн-форума было выбрано веб-приложение. Этот выбор обусловлен следующими факторами:

- Доступность через интернет – пользователи могут получить доступ к форуму с любого устройства без необходимости установки дополнительного ПО.
- Независимость от платформы – веб-интерфейс позволяет взаимодействовать с приложением независимо от операционной системы пользователя.
- Простая модель развертывания – обновления и исправления могут внедряться централизованно на сервере, без необходимости обновления клиентских приложений.
- Минимальные требования к ресурсам клиента – пользователю нужен только браузер, что снижает нагрузку на клиентские устройства.
- Масштабируемость – веб-приложение легко адаптируется к увеличению числа пользователей за счет серверных решений и облачных технологий.

Таким образом, веб-приложение является оптимальным выбором для разработки онлайн-форума, обеспечивая удобство развертывания, доступность и независимость от клиентской платформы. Реализовывать форум иначе, чем в виде веб-приложения, было бы нерациональным решением, так как его основная цель – обеспечить взаимодействие пользователей через интернет. Другие типы приложений не обеспечивают такого удобства и доступности для широкой аудитории.

1.2 Выбор стратегии развёртывания

Для развертывания онлайн-форума было выбрано 3-уровневое развертывание, поскольку оно обеспечивает баланс между производительностью, удобством масштабирования и безопасностью. В данной архитектуре:

- Клиент, в нашем случае браузер, взаимодействует с веб-сервером.
- Сервер приложений обрабатывает запросы, выполняет бизнес-логику и обращается к базе данных.
- Сервер базы данных хранит и обрабатывает информацию,

обеспечивая целостность данных.

Преимущества 3-уровневого развертывания для форума:

- Разделение ответственности – клиентский, серверный и базовый уровни четко разграничены, что повышает удобство поддержки и разработки.
- Гибкость и масштабируемость – сервер приложений и базу данных можно масштабировать независимо друг от друга.
- Безопасность – можно настроить межсетевые экраны между уровнями, ограничивая доступ к данным и снижая риск атак.

Такой подход является стандартным для веб-приложений и позволяет эффективно управлять нагрузкой и безопасностью.

При увеличении требований к безопасности и производительности возможен переход на 4-уровневое развертывание, в котором веб-сервер и сервер приложений разделены физически. Это полезно, если требуется изолировать бизнес-логику в отдельной защищенной среде или вынести ресурсоемкие процессы на отдельные серверы.

На текущем этапе 3-уровневое развертывание является оптимальным решением для форума, обеспечивая надежность, масштабируемость и простоту развертывания.

1.3 Обоснование выбора технологии

Ключевым фактором при выборе технологий является соответствие требованиям проекта, топологии развертывания и архитектурным принципам. Учитывались производительность, удобство развертывания, безопасность и масштабируемость. Для серверной части выбрана FastAPI, так как этот фреймворк обеспечивает высокую скорость обработки запросов, поддержку асинхронности и удобство разработки. В качестве базы данных используется PostgreSQL, поскольку она обладает высокой надежностью, поддерживает сложные запросы и легко масштабируется.

Для клиентской части выбран Vue.js, так как это современная библиотека для создания пользовательских интерфейсов, которая обеспечивает высокую производительность, модульность и удобство разработки. Vue.js позволяет создавать динамичные и отзывчивые интерфейсы, что особенно важно для веб-форума, где пользователи ожидают быстрого и интуитивно понятного взаимодействия.

Эти технологии обеспечивают баланс между производительностью и гибкостью, что особенно важно для веб-форума, работающего с большим количеством пользователей и текстовых данных. FastAPI позволяет эффективно обрабатывать запросы, PostgreSQL – надежно хранить и управлять данными, а Vue.js – создавать современный и удобный интерфейс. Выбранный стек технологий соответствует требованиям к веб-приложению и стратегии развертывания, обеспечивая надежность, безопасность и высокую производительность.

1.4 Показатели качества

При проектировании онлайн-форума ключевыми показателями качества являются производительность, безопасность, масштабируемость, удобство и простота использования, надежность и возможность повторного использования.

Производительность обеспечивается FastAPI, который благодаря асинхронности обрабатывает множество запросов без задержек, и Vue.js, оптимизирующим рендеринг интерфейса.

Безопасность критична для защиты пользовательских данных. FastAPI предоставляет инструменты для интеграции систем защиты, таких как аутентификация и авторизация.

Масштабируемость позволяет системе адаптироваться к росту нагрузки. Асинхронный подход FastAPI и модульность Vue.js упрощают добавление новых функций и масштабирование.

Удобство и простота использования достигается за счет Vue.js, который позволяет создавать динамичные и интуитивно понятные интерфейсы, улучшая взаимодействие с пользователем.

Надежность обеспечивает стабильную работу форума благодаря отказоустойчивости, резервному копированию и мониторингу системы.

Возможность повторного использования определяет пригодность компонентов и подсистем для применения в других приложениях и сценариях. Поскольку проект является открытым исходным кодом, его можно легко адаптировать под любую тематику, что снижает дублирование компонентов и сокращает время на реализацию новых решений.

Эти показатели помогают создать эффективный, удобный и гибкий онлайн-форум с современным интерфейсом.

1.5 Решение о путях реализации сквозной функциональности

Решение о путях реализации сквозной функциональности является важным этапом проектирования архитектуры приложения. Сквозная функциональность охватывает аспекты, которые не привязаны к конкретному функционалу, но критичны для обеспечения надежности, безопасности и производительности системы.

Механизм протоколирования будет реализован с использованием централизованного подхода, где каждый слой системы будет вести журнал событий в общем хранилище. Это позволит сопоставлять данные из разных слоев для анализа и отладки.

Аутентификация и авторизация будут реализованы с использованием JWT. Это позволит передавать аутентифицированные удостоверения между слоями системы и обеспечивать доступ к ресурсам на основе ролей пользователей.

Инфраструктура управления исключениями будет организована таким образом, чтобы исключения перехватывались на границах слоев и обрабатывались централизованно. Это предотвратит утечку конфиденциальной информации и обеспечит единый подход к обработке ошибок.

Связь между слоями будет осуществляться через REST API с использованием HTTP/HTTPS протоколов. Это обеспечит минимальное количество сетевых вызовов и защиту передаваемых данных. Для повышения производительности будут использоваться асинхронные запросы.

Кэширование будет применяться для уменьшения нагрузки на базу данных и ускорения обработки запросов, особенно для часто запрашиваемых данных, таких как списки тем или сообщений.

Таким образом, реализация сквозной функциональности обеспечит высокую производительность, безопасность и удобство поддержки системы, что особенно важно для онлайн-форума с большим количеством пользователей и данных.

1.6 Структурная схема приложения

Структурная схема приложения в виде функциональных блоков представлена на рисунке 1.1.

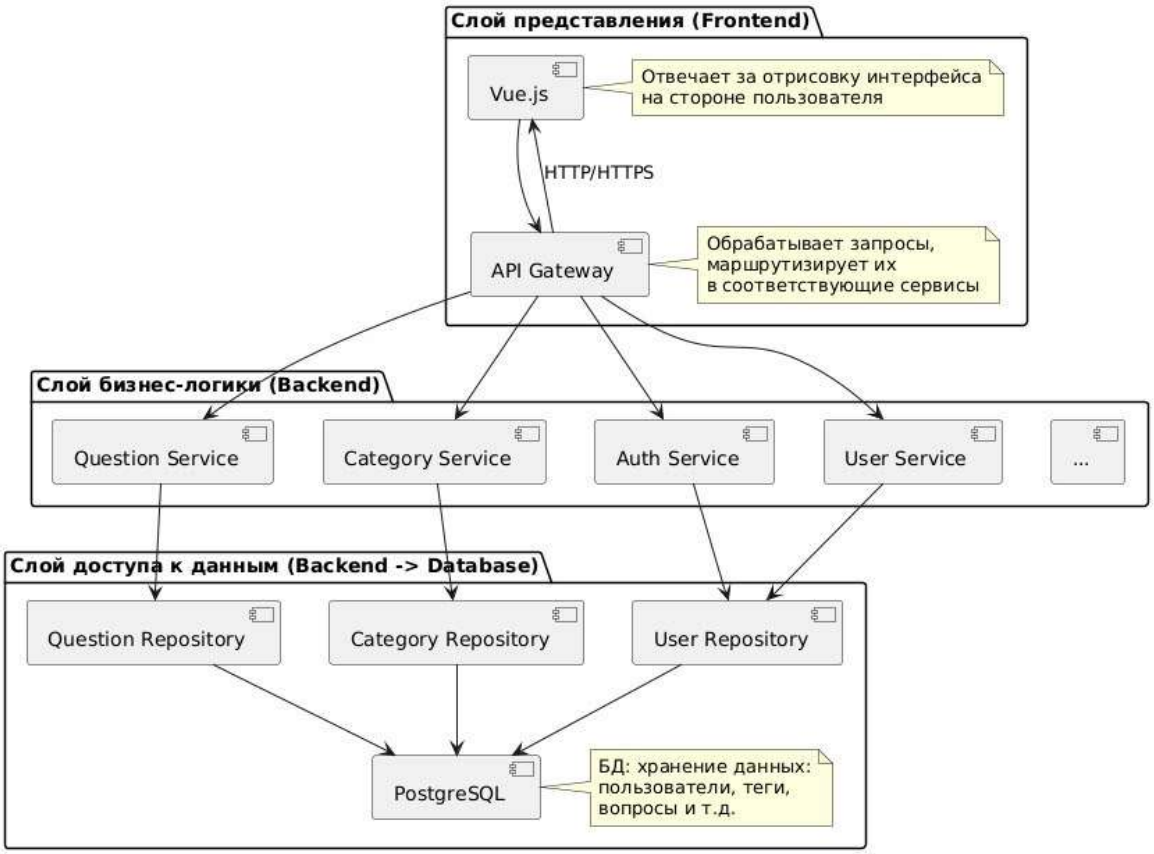

Рисунок 1.1 – Схема приложения

2 АНАЛИЗ АРХИТЕКТУРЫ

На данном этапе проведён анализ существующей архитектурысистемы, сформированной в ходе первого Sprint Review. С использованием инструмента обратной инженерии была сгенерирована диаграмма классов, отображающая структуру ключевых компонентов. Эта диаграмма классов представлена на рисунке 2.1.

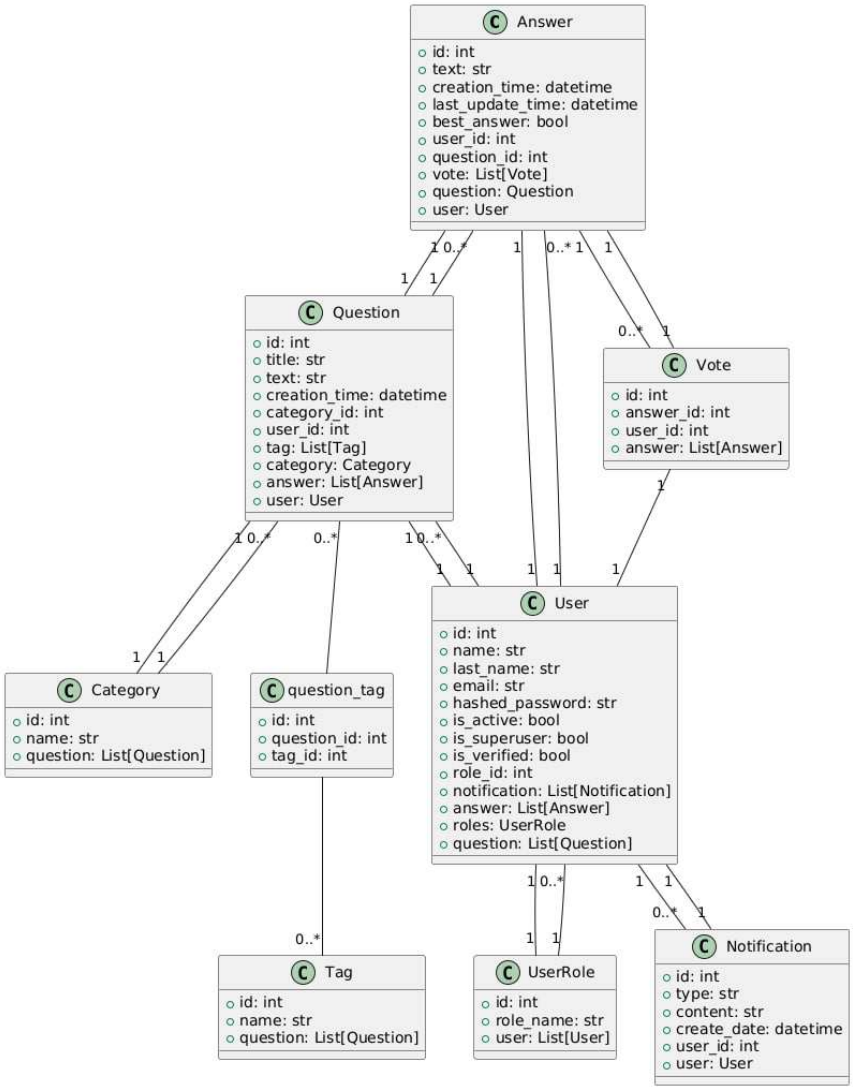

Рисунок 2.1 – Диаграмма классов

Связи между классами отражают их отношения, выявленные в исходном коде. Диаграмма демонстрирует основные модели системы:

User – отвечает за данные пользователей, включая их учетные данные, роли и связанные сущности, такие как вопросы, ответы и уведомления.

UserRole – определяет роли пользователей.

Question – представляет вопросы, создаваемые пользователями, и связан с ответами, тегами и категориями.

Answer – содержит ответы на вопросы, созданные пользователями, и связан с вопросами, пользователями и голосами.

Category – определяет категории, к которым могут относиться вопросы.

Tag – представляет теги, которые могут быть присвоены вопросам для их классификации.

Notification – отвечает за уведомления, отправляемые пользователям.

Vote – обеспечивает функциональность голосования за ответы.

question_tag – связывает вопросы и теги в отношении многие-ко-многим.

3 СРАВНЕНИЕ И РЕФАКТОРИНГ

3.1 Сравнение «As is» и «To be»

As is (текущая архитектура)

- Разделение на сервисы уже есть: Auth Service, User Service, Question Service, Category Service.
- Компоненты напрямую взаимодействуют между собой, API Gateway отсутствует.
- Взаимодействие между сервисами, вероятно, основано на REST API.
- Подключение к базе данных осуществляется напрямую из сервисов через репозитории.

To be (целевая архитектура)

- Введен API Gateway как единая точка входа для фронтенда и маршрутизации запросов к сервисам.
- Четкое разделение на Frontend (Vue.js), Backend (микросервисы) и Database.
- Схема хранения данных остается прежней, но добавлены уровни абстракции (репозитории).

3.2 Выделенные отличия и их причины

Отличие	Причина
Добавлен API Gateway	Централизованная маршрутизация, защита, балансировка нагрузки
Упрощено взаимодействие между сервисами	API Gateway скрывает внутреннюю структуру, фронтенд взаимодействует с одним входом
Возможное использование асинхронного взаимодействия (Kafka, RabbitMQ)	Уменьшение связности между сервисами, повышение отказоустойчивости
Разделение логики на уровни (Frontend, Backend, Database)	Четкое разграничение зон ответственности

3.3 Пути улучшения архитектуры

1. Оптимизация API Gateway
   - Добавить кеширование для популярных запросов.
   - Внедрить механизм rate limiting для защиты от перегрузки.
2. Декомпозиция сервисов
   - Разделить Question Service на Question и Answer Service для

уменьшения нагрузки на один сервис.

3. Внедрение событийной модели
   - Использовать Kafka или RabbitMQ для событий (например, создание уведомлений при ответе на вопрос).
4. Аутентификация через OAuth2
   - Использовать Keycloak или Auth0 для улучшенной безопасности и масштабируемости.
5. Горизонтальное масштабирование базы данных
   - Рассмотреть шардирование и репликацию PostgreSQL.

Эти изменения сделают систему более гибкой, отказоустойчивой и удобной в эксплуатации.