|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_Программное обеспечение ЭВМ и информационные технологии» (ИУ7)\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***Разработка системы хранения заметок***

Студент **\_\_\_\_ИУ7-22М\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_Андреев А.А.\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ Ступников А.А. \_\_**

(Подпись, дата) (И.О.Фамилия)

*2024 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. В. Рудаков

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине Распределенные системы обработки информации

Студент группы ИУ7-22м

Андреев Александр Алексеевич

(Фамилия, имя, отчество)

Тема курсового проекта Разработка системы хранения заметок

Направленность КП: Учебный

Источник тематики: кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

***Задание:***

Разработать прототип системы хранения заметок на веб-интерфейсе с микросервисной архитектурой. Реализовать сервисы: пользовательский интерфейс, авторизации, заметок, категоризации, команд, статистики (через Kafka). Добавить Identity Provider с OpenID Connect и Authorization Flow. Развернуть Single Page Application (React, Angular, Vue) с CSS. Настроить авторизацию, ролевую модель и валидацию JWT токенов через JWKs. Развернуть Managed Kubernetes Cluster с Ingress Controller.

***Оформление курсового проекта:***

Расчетно-пояснительная записка на 25-30 листах формата А4. Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую часть, конструкторскую часть, технологическую часть, заключение, список литературы, приложения.

На защиту проекта должна быть представлена презентация, состоящая из 10-15 слайдов.

Дата выдачи задания « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

**Руководитель курсового проекта** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ступников А.А.

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Андреев А.А.

(Подпись, дата) (И.О.Фамилия)

**Определения, обозначения и сокращения**

API - application programming interface

HTML - HyperText Markup Language

ПО - программное обеспечение

БД - база данных

СУБД – система управления базами данных

OSI - Open Systems Interconnection

CRUD – Create Read Update Delete

**Оглавление**

[Введение 10](#_Toc135948555)

[1. Аналитический раздел 12](#_Toc135948556)

[1.1. Описание системы 12](#_Toc135948557)

[1.2. Существующие аналоги 12](#_Toc135948558)

[1.2.1. Confluence 12](#_Toc135948559)

[1.2.2. Яндекс.Заметки 13](#_Toc135948560)

[1.2.3. Notion 13](#_Toc135948561)

[1.2.4. Evernote 13](#_Toc135948562)

[1.3. Требования к системе с точки зрения пользователя 13](#_Toc135948563)

[1.4. Требования к программной реализации 15](#_Toc135948564)

[1.5. Топология системы 15](#_Toc135948565)

[1.6. Функциональные требования к сервисам 16](#_Toc135948566)

[1.6.1. Фронтенд (пользовательский интерфейс) 16](#_Toc135948567)

[1.6.2. Сервис агрегации запросов (gateway сервис) 17](#_Toc135948568)

[1.6.3. Сервис авторизации 18](#_Toc135948569)

[1.6.4. Сервис заметок 18](#_Toc135948570)

[1.6.5. Сервис пространств страниц 18](#_Toc135948571)

[1.6.6. Сервис категорий 19](#_Toc135948572)

[1.6.7. Сервис статистики 19](#_Toc135948573)

[2. Конструкторский раздел 19](#_Toc135948574)

[2.1. Ролевая модель 19](#_Toc135948575)

[2.2. Архитектура системы 20](#_Toc135948576)

[2.2.1. Клиент-серверная архитектура системы 21](#_Toc135948577)

[2.2.2. Архитектура клиента 22](#_Toc135948578)

[2.2.3. Микросервисная архитектура сервера 22](#_Toc135948579)

[2.2.3.1. Протоколы связи между сервисами 23](#_Toc135948580)

[2.2.3.2. Адресация запросов между сервисами 24](#_Toc135948581)

[2.2.4. Сервис пространства страниц 25](#_Toc135948582)

[2.2.5. Сервис заметок 26](#_Toc135948583)

[2.2.6. Сервис категорий 27](#_Toc135948584)

[2.2.7. Сервис авторизации 27](#_Toc135948585)

[2.2.8. Сервис статистики 28](#_Toc135948586)

[2.2.9. Сервис агрегации запросов (gateway cервис) 29](#_Toc135948587)

[2.2.10. Сервис фронтенда 30](#_Toc135948588)

[2.3. Инструменты разработки 30](#_Toc135948589)

[2.4. Инструменты развертывания, обслуживания 31](#_Toc135948590)

[2.4.1. Docker 31](#_Toc135948593)

[2.4.2. Kubernates 31](#_Toc135948594)

[2.4.3. Helm 31](#_Toc135948595)

[2.4.4. Apache Kafka 32](#_Toc135948596)

[2.4.5. Протокол авторизации 32](#_Toc135948597)

[2.5. Концептуальный дизайн 33](#_Toc135948598)

[2.6. Диаграммы прецедентов 34](#_Toc135948599)

[2.7. Представление структуры классов 36](#_Toc135948600)

[2.8. Диаграмма последовательности действий 41](#_Toc135948601)

[2.9. Диаграмма поток данных 42](#_Toc135948602)

[3. Технологический раздел 46](#_Toc135948603)

[3.1. Схема базы данных 46](#_Toc135948604)

[3.2. Реализация серверной части 47](#_Toc135948605)

[3.3. Реализация клиентской части 47](#_Toc135948606)

[3.4. Сборка и развертывание системы 48](#_Toc135948607)

[3.5. Тестирование системы 48](#_Toc135948608)

[Заключение 49](#_Toc135948609)

[Список литературы 50](#_Toc135948610)

# **Введение**

В последнее время все большую популярность приобретают системы хранения документации, различных описаний проводимых исследований, заметок, ссылок на необходимые для работы ресурсов. Приложения, которые содержат такие сервисы, позволяют поддерживать в актуальном состоянии необходимую информацию, а также позволяют быстро найти нужный ресурс.

Целью данной курсовой работы является создание веб-приложения для хранения заметок, которое будет обладать конкурентными преимуществами в своей нише, а также соответствовать современным требованиям к разработке ПО, включая масштабируемость и гибкость архитектуры, а также использование современных библиотек и инструментов разработки.

В рамках работы над курсовым проектом производилось изучение существующих систем хранения заметок, велся анализ их пригодности к применению, плюсов и минусов относительно выполняемых задач. Была произведена разработка архитектуры приложения, с учетом достоинств и недостатков тех или иных аспектов проектирования. После этого под разработанную архитектуру были подобранны технологии для реализации поставленной задачи. Для работоспособности системы была подобрана современная СУБД и разработана база данных с учетом всех требований архитектуры и поставленной задачи. Были разработаны реализации клиента и сервера для выполнения поставленной задачи.

В качестве клиента разработан веб-сайт, рассчитанный на удобство и интуитивность использования, написанный с использование современного фреймворка, с учетом необходимости расширения функционала в будущем.

В качестве сервера была разработана и настроена группа скоординированных между собой программ каждая из которых выполняет свою определенную цель (микросевисов), развернутых в кластере Kubernetes.

После завершения разработки было произведено комплексное тестирование с использованием современных подходов различных частей системы на работоспособность и стабильность.

# 

1. Аналитический раздел

В данном разделе представлено описание системы и ее компонентов, а также аналогов. Приведены требования к системе.

* 1. Описание системы

Система представляет собой веб-сайт, доступ к которому пользователи осуществляют с персональных компьютеров, реализованный в виде веб-приложения. Главная цель системы для хранения заметок – предоставить доступ к корпоративной информации, к которой относятся: официальные документы, шаблоны заявлений, документация к проектам, пользовательские страницы c информацией для других сотрудников, заметки, а также ссылки на сторонние ресурсы. Пользователи системы должны иметь возможность зарегистрироваться, войти в аккаунт и выйти из сессии, просматривать, создавать, редактировать заметки, хранить информацию и ранжировать ее по категориям. Система должна позволять создавать пространство страниц конкретного отдела, доступ к которому имеют только сотрудники данного отдела и администраторы, пространство страниц для персонального использования, доступ к которому имеют приглашенные создателем пользователи, а также создавать приватные страницы для персонального использования. Пользователь может также просмотреть и редактирвоать информацию о себе.

* 1. Существующие аналоги

Существуют различные аналоги систем для хранения заметок. Некоторые предназначены для корпоративной работы, другие для персонального использования. К самым известным системам можно отнести: «Confluence», «Яндекс.Заметки», «Notion», «Evernote».

1. Confluence

Confluence – система, разработанная австралийской компанией Atlassian. Она предназначена для внутреннего использования организациями с целью создания единой базы знаний. Распространяется под проприетарной лицензией, бесплатна для некоммерческих организаций и открытых проектов.

* + 1. Яндекс.Заметки

Яндекс.Заметки — сервис для создания заметок в облаке. При помощи Яндекс.Заметок пользователи могут хранить идеи, задачи, списки покупок и другие заметки. Добавлять и просматривать заметки можно в браузере или с мобильного приложения. Заметки автоматически синхронизируется на устройствах пользователя. Сервис существует как отдельный инструмент и как часть [Яндекс.Диска](https://startpack.ru/application/yandex-drive" \o "Описание облака Яндекс.Диск" \t "_blank).

* + 1. Notion

Notion — приложение, которое предоставляет такие компоненты, как базы данных, доски канбан, вики, календари и напоминания. Пользователи могут подключать эти компоненты для создания собственных систем управления знаниями, ведения заметок, управления данными, управления проектами и другими. Эти компоненты и системы могут использоваться индивидуально или совместно с другими.

* + 1. Evernote

Evernote — веб-сервис и набор программного обеспечения для создания и хранения заметок. В качестве заметки может выступать фрагмент форматированного текста, веб-страница целиком, фотография, аудиофайл или рукописная запись. Заметки могут также содержать вложения с файлами другого типа. Заметки можно сортировать по блокнотам, присваивать им метки, редактировать и экспортировать.

* 1. Требования к системе с точки зрения пользователя

Система должна обеспечивать реализацию следующих функций:

* + 1. Регистрация и авторизация пользователей с валидацией вводимых значений с помощью интерфейса системы;
    2. Аутентификация пользователей;
    3. Возможность выйти из сессии;
    4. Разделение пользователей на роли:
    - Пользователь,
    - Администратор;
    1. Предоставление **Пользователю** следующих функций:
* Получение ранжированного списка хранимых документов и заметок;
* Поиск заметок по названию, дате, категории, отделу и названию проекта;
* Просмотр официальных документов, шаблонов заявлений, документаций к проектам, пользовательских страницы c информацией для других сотрудников, ссылок на сторонние ресурсы, заметок;
* Создание, редактирование и удаление персональных пространств страниц;
* Добавление и удаление пользователей из персонального пространства страниц;
* Создание, редактирование и удаление официальных документов, шаблонов заявлений, документаций к проектам, пользовательских страницы c информацией для других сотрудников, ссылок на сторонние ресурсы, заметок;
* Добавление и удаление категорий для заметок;
* Изменение информации аккаунта;
  + 1. Предоставление **Администратору** следующих функций:

(+ все функции пользователя)

* Управление корпоративным пространством страниц;
* Распределение ролей;
* Функционал для добавления, удаления и настройки узлов;
* Возможность просмотра статистики.
  1. Требования к программной реализации
     1. Система должна состоять из микросервисов, каждый из которых отвечает за свою область логики приложения;
     2. Взаимодействие между сервисами осуществляется по протоколу HTTP;
     3. Данные сервисов хранятся в базе данных. Каждый сервис имеет свою базу данных и не имеет доступа к базам данных других сервисов;
     4. При недоступности систем портала должна осуществляться деградация функционала или выдача пользователю сообщения об ошибке;
     5. Необходимо предусмотреть авторизацию пользователей через интерфейс веб-приложения;
     6. Для запросов, обновляющих данные в нескольких узлах системы, нужно предусмотреть откат всей операции в случае возникновения ошибки.
  2. Топология системы

Разрабатываемая система состоит из фронтенда (пользовательский интерфейс) и 6 подсистем:

* сервис агрегации запросов (gateway сервис);
* сервис авторизации;
* сервис пространства страниц;
* сервис заметок;
* сервис категорий;
* сервис статистики.

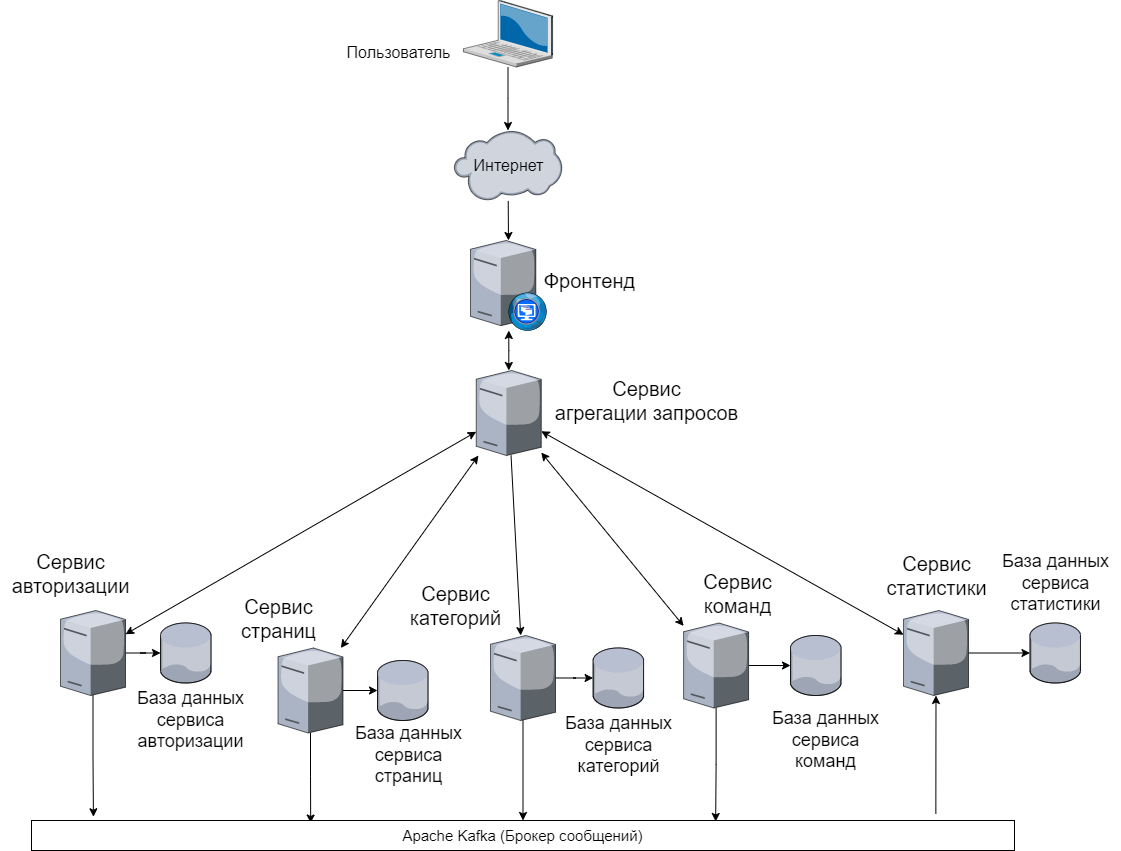


Рисунок 1 – логическая топология системы.

* 1. Функциональные требования к сервисам

Здесь перечислены функциональные требования к каждому из сервисов.

* + 1. Фронтенд (пользовательский интерфейс)

Сервис фронтенда (пользовательского интерфейса) представляет собой сервер, отдающий пользователю файлы веб-приложения. После получения и запуска веб-приложения пользователь работает с системой через него.

Веб-приложение (фронтенд) должно отвечать следующим требованиям:

1. Фронтенд принимает и обрабатывает команды пользователя, передаваемые через элементы интерфейса;
2. Фронтенд разрабатывается в формате толстого клиента и самостоятельно формирует веб-страницы на основе данных, полученных от gateway сервиса; рекомендуется использовать фреймворк Vue для разработки;
3. При необходимости получения данных от сервисов фронтенд отправляет на gateway сервис запросы по протоколу HTTP и получает ответ;
4. Обмен данными по протоколу HTTP осуществляется в формате JSON;
5. Фронтенд самостоятельно выполняет часть операций обработки данных, а именно их фильтрацию, сортировку и преобразования в пригодный для отображения в интерфейсе вид;
6. Фронтенд может кешировать данные, чтобы предотвратить лишние обращения к серверу.
   * 1. Сервис агрегации запросов (gateway сервис)

Сервис агрегации запросов (gateway сервис) является единственным доступным для взаимодействия с фронтендом. Он выполняет роль прослойки между фронтендом и остальной системой. Он должен удовлетворять следующим требованиям:

* + 1. Сервис должен осуществлять взаимодействие с фронтендом и другими сервисами по протоколу HTTP, принимать и передавать данные в формате JSON;
    2. Сервис должен предоставлять методы API системы, каждый из которых (кроме регистрации и входа в систему) должен быть защищен JWT авторизацией;
    3. Сервис должен разделять методы, доступные только пользователям и только администраторам;
    4. Сервис должен реализовывать логику деградации функциональности при недоступности сервисов, необходимых для совершения операции;
    5. Сервис должен выполнять откат сложных операций при возникновении в них ошибок;
    6. Сервис должен получать данные от других сервисов и выполнять их предобработку, если операция этого требует.
    7. Сервис авторизации

Сервис авторизации хранит данные пользователей и сессий, а также осуществляет авторизацию и проверку токенов пользователей. Он должен удовлетворять следующим требованиям:

1. Сервис должен осуществлять взаимодействие с gateway сервисом по протоколу HTTP, передавать и принимать данные в формате JSON;
2. Сервис должен проводить авторизацию пользователей, управлять сессией пользователя;
3. Сервис должен отдавать информацию о пользователях;
4. Сервис использует JWT авторизацию с одним токеном, токен доступа имеет максимальное время жизни 30 минут;
5. Сервис должен давать возможность регистрации новых пользователей.
   * 1. Сервис заметок

Сервис заметок хранит информацию о заметках. Он должен удовлетворять следующим требованиям:

1. Сервис должен осуществлять взаимодействие с gateway сервисом по протоколу HTTP, передавать и принимать данные в формате JSON;
2. Сервис должен отдавать список всех заметок и информацию о конкретной заметке;
3. Сервис должен хранить и реализовывать работу с заметками и их структурой, получение, создание, редактирование, удаление страниц, перенос страницы в другую директорию.
   * 1. Сервис пространств страниц

Сервис пространства страниц хранит информацию о пространствах страниц. Он должен удовлетворять следующим требованиям:

1. Сервис должен осуществлять взаимодействие с gateway сервисом по протоколу HTTP, передавать и принимать данные в формате JSON;
2. Сервис должен отдавать список всех пространств страниц и информацию о конкретном пространстве страниц;
3. Сервис должен позволять выполнять следующие действия: создание и удаление команды, добавление и удаление участника из команды.
   * 1. Сервис категорий

Сервис категорий хранит информацию о категориях. Он должен удовлетворять следующим требованиям:

1. Сервис должен осуществлять взаимодействие с gateway сервисом по протоколу HTTP, передавать и принимать данные в формате JSON;
2. Сервис должен отдавать список всех категорий и информацию о конкретной категории;
   * 1. Сервис статистики

Сервис статистики собирает информацию о происходящих в системе событиях. Каждый сервис при совершении операции отправляет сообщение на этот сервис. Он должен удовлетворять следующим требованиям:

1. Сервис должен осуществлять взаимодействие с gateway сервисом по протоколу HTTP, передавать и принимать данные в формате JSON;
2. Сервис должен принимать записи о проводимых на сервисах операциях и сохранять их;
3. Сервис должен отдавать статистику о работе системы.
4. Конструкторский раздел

В данном разделе описывается архитектура системы. Приведено описание ролевой модели и каждого компонента системы.

* 1. Ролевая модель

В системе присутствуют две роли пользователей:

* User – обычный пользователь системы. Может создавать, редактировать, удалять пользовательские пространства страниц, заметки и категории, а также добавлять и удалять пользователей в своем пространстве страниц;
* Admin – администратор. Может просматривать статистику по операциям, производимым сервисами бекенда. Может создавать, редактировать, удалять корпоративные пространства страниц, а также добавлять и удалять пользователей в корпоративном пространстве страниц. Может создавать, редактировать, удалять пользовательские пространства страниц, заметки и категории, а также добавлять и удалять пользователей в своем пространстве страниц;
  1. Архитектура системы

В данной системе каждая функциональная область реализована посредством собственного микросервиса. Этот подход позволяет бороться со сложностью современных систем. На рисунке 2 представлена архитектура системы, которая показывает размещение элементов системы на физических носителях и способах их взаимодействия, то есть, указаны протоколы, по которым происходит информационный обмен.

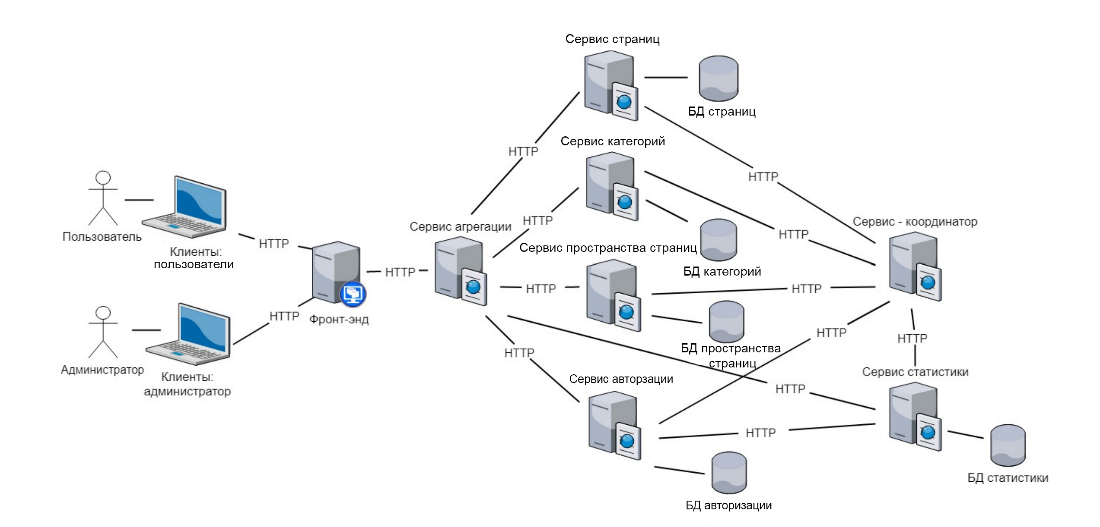


Рисунок 2 – архитектура разрабатываемой системы.

* + 1. Клиент-серверная архитектура системы

В клиент-сервисной архитектуре функционал приложения и вычислительная нагрузка разделяется на 2 части: клиентскую и серверную.

Клиент — это часть ПО, отвечающая за взаимодействие с конкретным пользователем.

Сервер — это часть ПО, отвечающая за обслуживание клиентов. Она занимается выполнением запросов от клиентов, а также при необходимости хранит данные необходимые для их выполнения. К одному серверу может быть подключено множество клиентов.

Как сервер, так и клиент могут быть как монолитными программами, так и группой программ, нацеленных на выполнение своей задачи или физическими устройствами. Чаще всего все тяжелые операции выполняются на одном мощном сервере, а на стороне клиента остаются лишь легкие операции, так же иногда в целях безопасности все операции переносятся на сервер оставляя клиенту лишь возможность запрашивать результат.

Данная архитектура имеет преимущества и недостатки.

Преимущества:

* Экономия на ресурсах клиентских машин, за счет выполнения операций на сервере.
* Увеличение безопасности за счет ограничения доступа к данным клиентам.
* Возможность использовать ресурсы сервера на различных программных платформах и на значительном удалении от сервера.

Недостатки:

* При выходе из строя сервера, клиенты тоже выходят из строя.
* Необходимость в обслуживании сети.
* Повышенная сложность разработки относительно обычного локального приложения.
  + 1. Архитектура клиента

Изменения в клиенте будут происходить по инициации сервера, необходимо описать мутации. Мутации — это функции, вызывающиеся при вызове определенных событий и изменяющие состояние компонентов отрисовки.

В данном случае используется store.js, который принимает команды с сервера и распределяет их по окнам. Далее соответствующие окнам компоненты мутации изменяют состояния компонентов отрисовки за которые они отвечают.

Также в соответствии с командой с сервера производится смену состояния через компонент state, это позволяет менять окна между собой в независимости от их состояний и не внося изменения в их мутации.

* + 1. Микросервисная архитектура сервера

Микросервисная архитектура — вариант [сервис-ориентированной архитектуры](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B8%D1%81-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов, получивший распространение в середине 2010-х годов в связи с развитием практик [гибкой разработки](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D0%BA%D0%B0%D1%8F_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) и [DevOps](https://ru.wikipedia.org/wiki/DevOps" \o "DevOps).

Плюсы микросервесной архитектуры:

* Модули можно относительно легко подменять, что упрощает разработку и развертывание нового функционала
* Отказ одного из модулей не выводит из строя всю систему
* Модули можно писать на разных языках программирования, они могут работать в разных средах и их всё еще будет легко соединять между собой посредством API
* Легко перераспределять ресурсы между разными сервисами, и производить масштабирование

Минусы микосервисной архитектуры:

* Усложнение разработки системы в целом
* Увеличение потребления ресурсов относительно монолитной архитектуры

Исходя из плюсов и минусов микросервисной и монолитной архитектур выходит, что для прототипирования больше подходит монолитная архитектура, а для систем, которые планируется поддерживать и развивать, на расстоянии, более выгодно использование микросервисной архитектуры за счет уменьшения связности между модулями и масштабируемости.

* + - 1. Протоколы связи между сервисами

Существует множество протоколов для связи между сервисами, но при разработке веб-приложений в основном используется два самых популярных HTTP и TCP.

* + 1. Протокол HTTP

Протокол HTTP (HyperText Transfer Protocol) – протокол прикладного уровня модели передачи данных OSI, изначально разработанный для передачи текстовых данных, но в настоящее время используется для передачи произвольных данных. Основан на TCP/IP соединении.

Наибольшее распространение получил во Всемирной паутине. Сам является основой для многих других протоколов.

HTTP предполагает существование:

* Клиентов (потребителей) – устройств, которые инициируют соединение и отправляют запрос на сервер.
* Серверов (поставщиков) – устройств, которые ожидают соединения для получения запроса, производят требуемые операции и высылают обратно ответ с результатом.

Основным объектом в протоколе является ресурс, на который ссылается URI (Uniform Resource Identifier) в запросе от клиента. Это могут быть как файлы, лежащие на сервере, так и абстрактные результаты вычислений. Особенностью протокола является возможность указать формат представления ресурса с помощью параметров. Именно благодаря этому свойству устройства могут обмениваться двоичными данными по протоколу несмотря на то, что он рассчитан на текстовый формат представления.

Большинство протоколов предусматривают однократное соединение с последующим обменом данными. Протокол HTTP же создает новую сессию на каждый запрос или во второй версии группу запросов. Данное решение с одной стороны дает широкие возможности по распределению нагрузки между серверами и увеличивает количество одновременно обрабатываемых клиентов, но с другой оказывает негативное влияние на скорость конкретного запроса.

* + 1. TCP

Kafka использует двоичный протокол поверх TCP. Протокол определяет все API-интерфейсы как пары сообщений "запрос-ответ". Все сообщения ограничены по размеру и состоят из следующих примитивных типов.

Клиент инициирует сокетное соединение, а затем записывает последовательность сообщений запроса и считывает обратно соответствующее ответное сообщение. При подключении или разъединении рукопожатие не требуется. Протокол TCP работает лучше, если поддерживать постоянные соединения, используемые для многих запросов, чтобы компенсировать стоимость подтверждения TCP, но помимо этого штрафа подключение обходится довольно дешево.

* + - 1. Адресация запросов между сервисами

Для решения задачи адресации запросов между пользователем и сервисами и сервисов между собой и частичной балансировки нагрузки необходимо использовать сервис прокладку который будет отвечать за перенаправление запроса на конечные адреса и порты. Так как это часто возникающая проблема на рынке существует множество готовых решений, в связи с чем нет целесообразности в реализации собственной имплементации. Для решения поставленных задач идеально подходит HTTP-сервер Nginx.

Nginx - HTTP-сервер и обратный прокси-сервер, почтовый прокси-сервер, а также TCP/UDP прокси-сервер общего назначения. Является отечественной разработкой и согласно исследованию компании, специализирующейся на информационной безопасности Неткрафт (с английского Netcraft) более 21% самых нагруженных сайтов в апреле 2022 года, обслуживались данным прокси сервером.

Преимущества:

* Высокая скорость работы относительно конкурентов
* Малое потребление ресурсов
* Хорошая масштабируемость

Недостатки:

* Меньше возможностей в сравнении с конкурентами
* Меньшее количество сторонних модулей чем у конкурентов

Данное ПО используется для адресации и балансировки HTTP сервисов, а также адресации динамически выделяемых узлов с соединением WebSocket, из-за необходимости следить за состоянием и управлять узлами балансировкой нагрузки узлов занимаются их диспетчеры.

* + 1. Сервис пространства страниц

Сервис пространства страниц хранит информацию о пространствах страниц. Сервис способен отдавать список пространства страниц, а также отдавать информацию о конкретном пространстве страниц. Таблица 1 содержит методы API сервиса.

Таблица . API сервиса пространства страниц

|  |  |
| --- | --- |
| URL | Описание |
| GET /namespaces | Получение списка пространств страниц.  Query параметры:   * page * size |
| GET /namespaces/:id | Получение пространства страниц по id. |
| POST /namespaces | Создание пространства страниц.  Тело запроса:   * name |
| DELETE /namespaces/:id | Удаление пространства страниц по id. |
| GET /namespaces/:id/users/:id | Получение пользователя в пространстве страниц. |
| GET /namespaces/:id/users/me | Проверка, что текущий пользователь принадлежит данному пространству страниц. |
| POST /namespaces/:id/users/:id | Добавление пользователя в пространство страниц. |
| DELETE /namespaces/:id/users/:id | Удаление пользователя из пространства страниц. |

Сервис связан с базой данных, в которой присутствует таблица namespace, хранящая информацию о пространствах страниц. Таблица 2 содержит ее спецификацию.

Таблица 2. Спецификация таблицы namespace

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Описание атрибута |
| id | UUID | Идентификатор пространства страниц |
| name | string | Название пространства страниц |

* + 1. Сервис заметок

Сервис заметок хранит информацию о заметках. Сервис способен отдавать список заметок, а также отдавать информацию о конкретной заметке. Таблица 3 содержит методы API сервиса.

Таблица 3. API сервиса заметок

|  |  |
| --- | --- |
| URL | Описание |
| GET /notes | Получение списка заметок для пространства страниц.  Query параметры:   * namespace\_id * page * size |
| GET /notes/:id | Получение заметки по ее id. |
| POST /notes | Создание заметки.  Тело запроса:   * title * text * namespace\_id |
| DELETE /notes/:id | Удаление заметки по ее id. |

Сервис связан с базой данных, в которой присутствует таблица note, хранящая информацию о заметках. Таблица 4 содержит ее спецификацию.

Таблица 4. Спецификация таблицы note

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Описание атрибута |
| id | UUID | Идентификатор заметки |
| title | string | Название заметки |
| text | string | Текст заметки |
| created\_date | datetime | Дата создания заметки |
| author\_id | UUID | Идентификатор автора заметки |
| namespace\_id | UUID | Идентификатор пространства страниц, в котором находится заметка |

* + 1. Сервис категорий

Сервис категорий хранит информацию о категориях. Сервис способен отдавать список категорий, а также отдавать информацию о конкретной категории. Таблица 5 содержит методы API сервиса.

Таблица 5. API сервиса заметок

|  |  |
| --- | --- |
| URL | Описание |
| GET /categories | Получение списка категорий.  Query параметры:   * namespace\_id * page * size |
| GET /categories/:id | Получение категории по id. |
| POST /categories | Создание категории.  Тело запроса:   * name * namespace\_id |
| DELETE /categories/:id | Удаление категории по id. |
| GET /notes/:id/categories | Получение категорий для заметки.  Query запросы:   * page * size |
| POST /notes/:id/categories/:id | Добавление категории для заметки. |

Сервис связан с базой данных, в которой присутствует таблица category, хранящая информацию о категориях. Таблица 6 содержит ее спецификацию.

Таблица 6. Спецификация таблицы category

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Описание атрибута |
| id | UUID | Идентификатор категории |
| name | string | Название категории |
| namespace\_id | UUID | Идентификатор пространства страниц |

* + 1. Сервис авторизации

Сервис авторизации хранит информацию о пользователях и сессиях, позволяет создавать и редактировать пользователей, создавать и удалять сессии, проводить валидацию jwt токенов. Jwt токены генерируются при создании сессии. Таблица 7 содержит методы API сервиса.

Таблица 7. API сервиса заметок

|  |  |
| --- | --- |
| URL | Описание |
| GET /users | Получение списка пользователей.  Query параметры:   * page * size |
| GET /users/:id | Получение пользователя по id. |
| GET /users/me | Получение текущего пользователя. |
| POST /users | Создание пользователя.  Тело запроса:   * username * first\_name * second\_name * email * password * is\_superuser |
| DELETE /users/:id | Удаление пользователя по id. |
| POST /token | Получение кода для авторизации  Query параметры:   * client\_id * username * password |
| POST /access\_token | Получение токена доступа  Query параметры:   * code |

Сервис связан с базой данных, в которой присутствует таблица user, хранящая информацию о пользователях. Таблица 8 содержит ее спецификацию.

Таблица 8. Спецификация таблицы user

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Описание атрибута |
| id | UUID | Идентификатор пользователя |
| username | string | Логин пользователя |
| first\_name | string | Имя пользователя |
| last\_name | string | Фамилия пользователя |
| email | string | Электронная почта пользователя |
| hashes\_pwd | string | Пароль пользователя |
| role | string | Роль пользователя |

* + 1. Сервис статистики

Сервис статистики хранит записи о происходящих в системе действиях, которые он получает от других сервисов бекенда. Он предоставляет возможность добавить запись и получить все записи для конкретного сервиса в диапазоне дат. Таблица 9 содержит методы API сервиса.

Таблица 9. API сервиса заметок

|  |  |
| --- | --- |
| URL | Описание |
| GET /statistics | Получение списка пользователей.  Query параметры:   * service * start\_date * finish\_date * page * size |

Сервис связан с базой данных, в которой присутствует таблица statistic, хранящая информацию о действиях сервисов. Таблица 10 содержит ее спецификацию.

Таблица 10. Спецификация таблицы user

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Описание атрибута |
| id | int | Id записи, первичный ключ |
| service | varchar(32) | Название сервиса |
| description | varchar(512) | Описание записи |
| timestamp | Timestamp with time zone | Дата создания записи |

* + 1. Сервис агрегации запросов (gateway cервис)

Gateway сервис предоставляет внешнее API системы и выступает в роли агрегатора запросов. Сервис принимает запросы от клиентского приложения по протоколу HTTP и отправляет ответы на них. При необходимости сервис осуществляет взаимодействие с другими сервисами бекенда. Сервис осуществляет валидацию данных запросов и инициирует проверки авторизации и прав доступа. Проверка авторизации осуществляется для всех запросов, кроме запросов регистрации и авторизации пользователя.

Все операции, требующие авторизации, предполагают наличие заголовка Authorization со значением формата “Bearer <jwt-token>”, по которому проверяется факт авторизации и из которого выводится производящий операцию пользователь.

Сервис предусматривает действия при недоступности требуемых для успешного выполнения операции сервисов. Для гарантии записи статистики и отката транзакций используются очереди задач. Для обеспечения деградации функциональности при недоступности некритичных источников данных используется подход “Circuit breaker”.

* + 1. Сервис фронтенда

Сервис фронтенда хранит статические файлы клиентского приложения и отдает их пользователю при использовании сайта. Веб-приложение имеет формат SPA и само рендерит страницы, поэтому этот сервис только отдает статические файлы приложения.

* 1. Инструменты разработки

1. **Vue JS**

Vue это реактивный java script фреймворк для создания пользовательских интерфейсов. Одним из отличительных качеств данного фреймворка является возможность постепенного внедрения за счет его модульности. Это возможно за счет того, что его ядро содержит решает только задачи уровня представления, что упрощает его интеграцию с другими библиотеками и существующими проектами.

Реактивность фреймворка означает что представление в модели Model-View-Controller изменяется по мере изменения модели. В Vue разработчик описывает модели, а потом фреймворк сам изменяет представление на основе изменения модели. Это реализовано с помощью шаблона наблюдатель.

В Vue существуют экземпляры и компоненты. Экземпляры привязаны к HTML узлу, к одному html документу может быть подключено произвольное количество экземпляров. Компоненты напротив не привязаны к конкретным узлам, они содержат свой шаблон html и используются для расширения функционала экземпляров. Изменяемость HTML-шаблону придается 2 способами: вставками и директивами.

Вставки представляют собой java script объекты, вставленные в шаблон посредством оборачивания в двойные фигурные скобки. При изменении вставленного объекта будет маниться и его представление.

Директивы представляют собой HTML-атрибуты, начинающиеся с префикса «v\_». Директивы позволяют задавать условное поведение при отрисовке html-тега. Из основных: отрисовка по условию, перехват событий, отрисовка коллекции.

* 1. Инструменты развертывания, обслуживания

Для упрощения процесса развертывания использовались следующие технологии.

1. 1. 1. Docker

Система “контейнеризации”, которая позволяет запускать приложения в отдельных средах для решения проблем зависимостей.

Пользователь может упаковать приложение в контейнер и запускать далее на любой ЭВМ с установленной системой docker, так же имеется возможность оперирования группами контейнеров для разворачивания сложных систем. Для создания контейнера необходимо описывать специальный DockerFile. Для организации группы контейнеров в единую систему необходимо описывать docker-compose.yml файл.

В данной работе каждый отдельный сервис запакован в отдельный контейнер для простоты развертывания новых версий.

* + 1. Kubernates

Kubernetes — открытое программное обеспечение для оркестровки контейнеризированных приложений — автоматизации их развёртывания, масштабирования и координации в условиях кластера. Поддерживает основные технологии контейнеризации, включая Docker, rkt, также возможна поддержка технологий аппаратной виртуализации.

Даная технология позволяет управлять состоянием, количеством экземпляров, трафиком и выделяемыми ресурсами множества контейнеров. Используется для упрощения развертывания и поддержки систем состоящих из множества контейнеров.

* + 1. Helm

Helm — это пакетный менеджер, облегчающий задачи настройки, установки и обновления приложений в Kubernetes.

В Helm система описывается в виде чарта — совокупности yaml-файлов с шаблонами конфигураций, а также нескольких служебных файлов. После написания чарт может быть упакован в архив и распространен с помощью мезанизма Helm-репозиториев.

Данный инструмент в разы упрощает конфигурирование приложений, а также развертывание новых версий без остановки системы.

* + 1. Apache Kafka

Apache Kafka — распределённый программный брокер сообщений с открытым исходным кодом, разрабатываемый в рамках фонда Apache на языках Java и Scala.

Отвечает за обмен сообщениями между сервисами и сохранение логов, для решения возникающих в процессе эксплуатации системы багов.

* + 1. Протокол авторизации

В качестве системы авторизации выбран перспективный протокол OpenID connect. Схема работы представлена на рисунке 3:

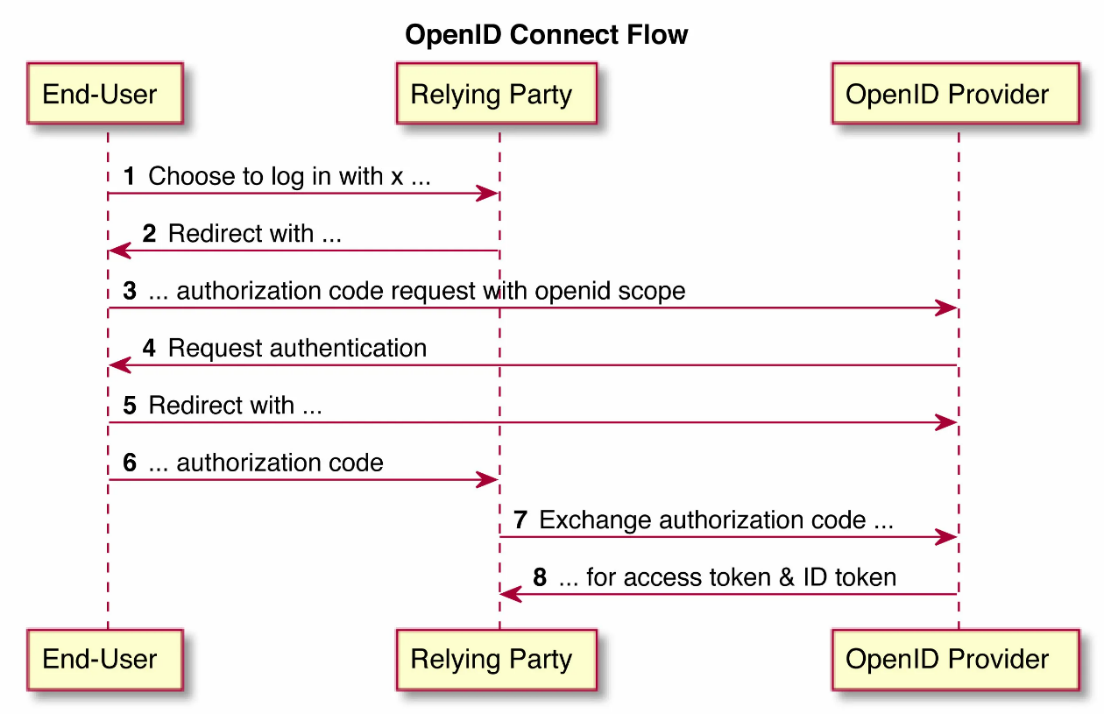


Рисунок 3 — схема работы OpenID.

Протокол основан на Oauth2.0.

Выделяются 4 сущности:

* End-user – клиентская часть системы.
* Relying Party – серверная часть системы.
* OpenID Provider - сервис аутентификации.
* Resource Server – не представлен на схеме, является конечным хранилищем запрашиваемого ресурса.

Данный протокол позволяет организовывать систему авторизации в системе на основе множества внешних систем авторизации, что позволяет ускорить регистрацию и авторизацию для части новых пользователей, а также упрощает смену сервиса авторизации.

* 1. Концептуальный дизайн

Концептуальный дизайн информационного портала содержит наиболее общие схемы описания функционала приложения с точки зрения пользователей. Одной из таких схем является IDEF0-модель и графические модели, входящие в нее. На рисунке 4 отображена контекстная диаграмма верхнего уровня, которая обеспечивает наиболее общее или абстрактное описание работы информационного портала. Данный вид диаграммы позволяет формализовать описание запросов пользователя и ответов системы на данные запросы, отобразив систему в виде “черного” ящика.

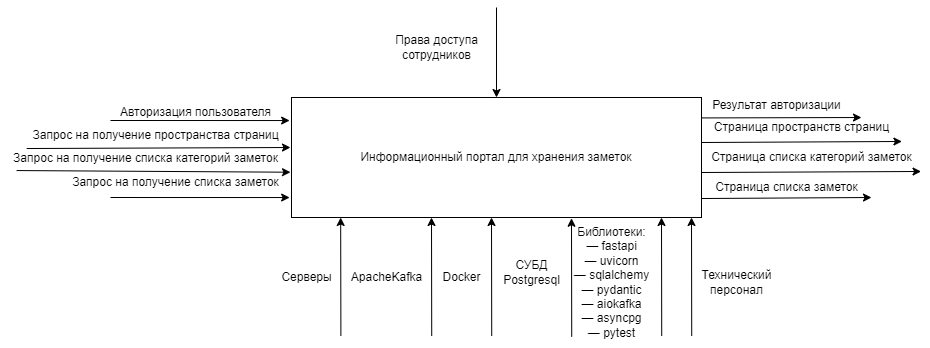


Рисунок 4 — концептуальная модель системы в нотации IDEF0.

Для уточнения деталей работы информационного портала применяется декомпозиция функций, отображенных на диаграмме верхнего уровня, при помощи создания дочерних диаграмм. В качестве примера на рисунке 5 изображена дочерняя диаграмма, которая определяет последовательность выполнения операций в информационном портале при обработке запроса пользователя на получение списка страниц с точки зрения администратора системы.

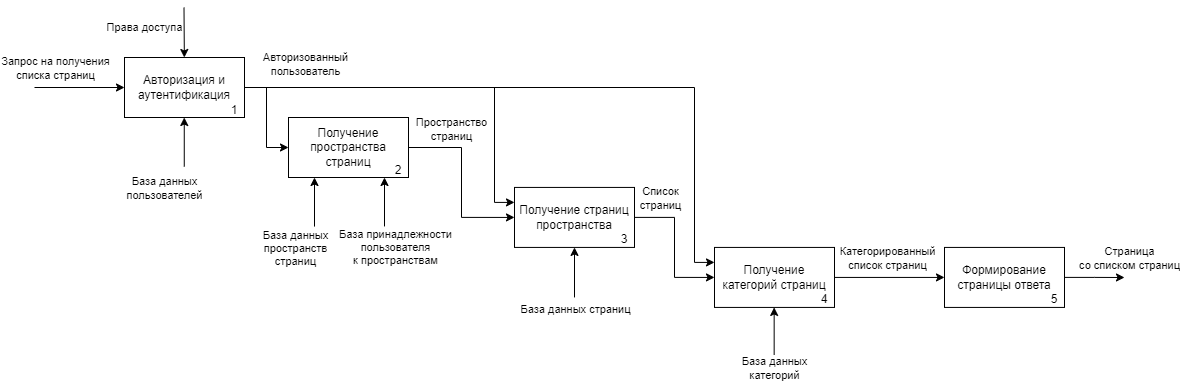


Рисунок 5 — концептуальная модель запроса на получение списка страниц.

На рисунке 6 представлена концептуальная модель запроса на создание заметки с точки зрения пользователя.

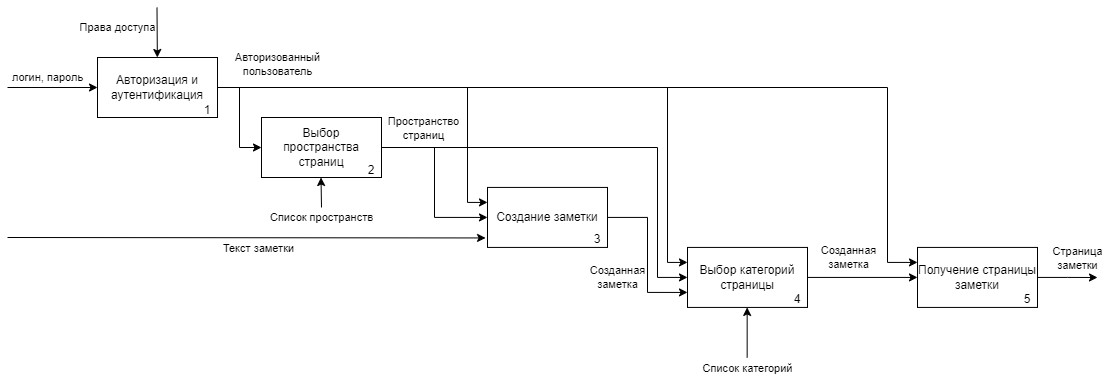


Рисунок 6 — IDEF0 диаграмма создания заметки с точки зрения пользователя.

* 1. Диаграммы прецедентов

Графически сценарии функционирования системы можно представить при помощи диаграмм прецедентов. Они позволяют схематично отобразить типичные сценарии взаимодействия между клиентами и приложением. В системе выделены 2 основных роли: пользователь и администратор, диаграммы прецедентов для этих ролей изображены на рисунках 7 и 8.

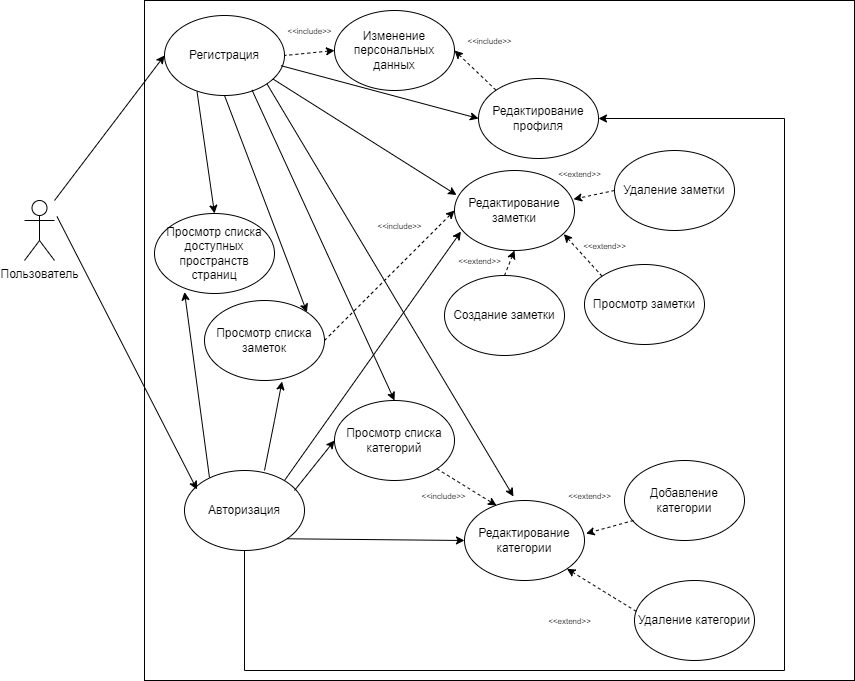


Рисунок 7 — диаграмма прецедентов с точки зрения пользователя.

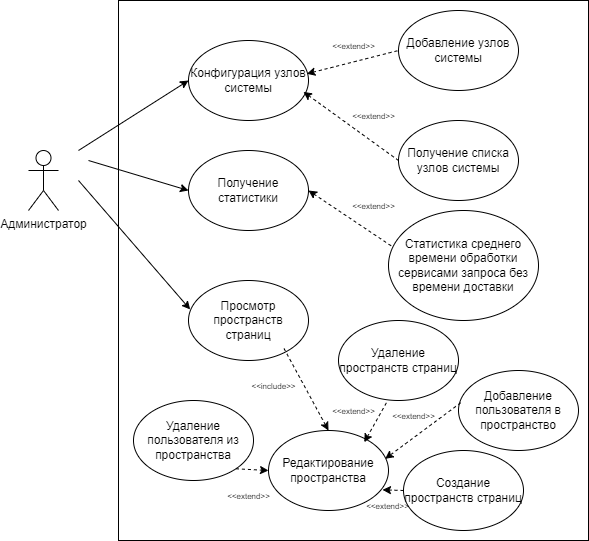


Рисунок 8 — диаграмма прецедентов с точки зрения администратора.

* 1. Представление структуры классов

На рисунке 9 представлена диаграмма классов для разработки сервиса страниц.

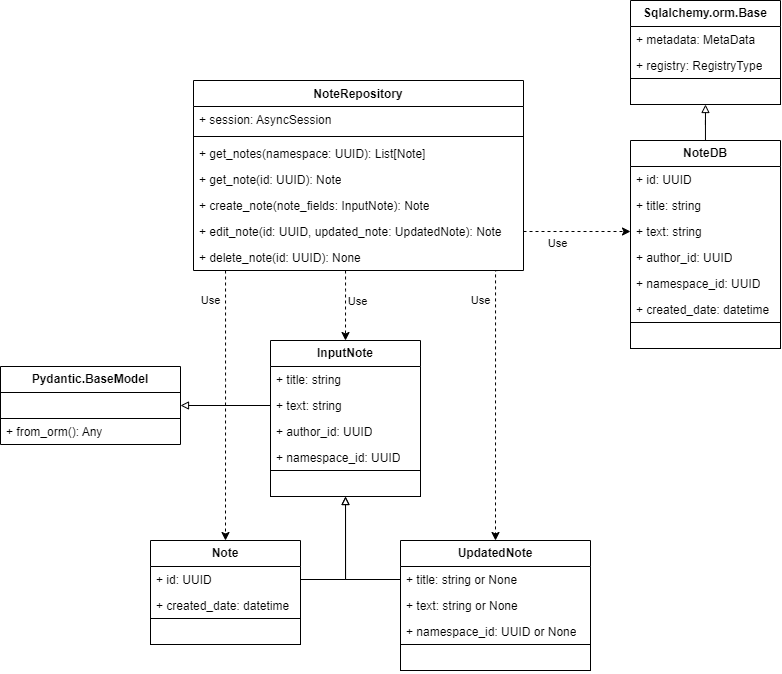


Рисунок 9 — диаграмма классов сервиса страниц.

Функциональные требования, предъявляемые к сервису страниц, реализуются при помощи методов контроллера NoteRepository. В таблице 11 приведено описание каждого метода данного контроллера.

Таблица 1. Спецификация класса NoteRepository

|  |  |
| --- | --- |
| Метод | Описание |
| Get\_notes | Получение списка заметок |
| Get\_note | Получение заметки |
| Create\_note | Создание заметки |
| Edite\_note | Редактирование заметки |
| Delete\_note | Удаление заметки |

На рисунке 10 представлена диаграмма классов для разработки сервиса авторизации.

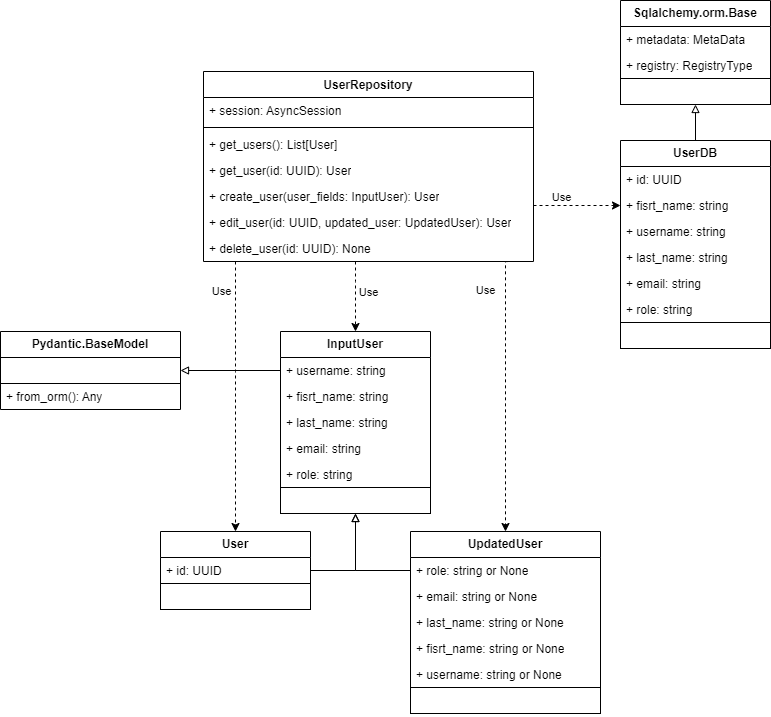


Рисунок 10 — диаграмма классов сервиса авторизации.

Функциональные требования, предъявляемые к сервису авторизации, реализуются при помощи методов контроллера UserRepository. В таблице 12 приведено описание каждого метода данного контроллера.

Таблица 2. Спецификация класса UserRepository

|  |  |
| --- | --- |
| Метод | Описание |
| Get\_users | Получение списка пользователей |
| Get\_user | Получение пользователя |
| Create\_user | Создание пользователя |
| Edite\_user | Редактирование пользователя |
| Delete\_user | Удаление пользователя |

На рисунке 11 представлена диаграмма классов для разработки сервиса пространства страниц.

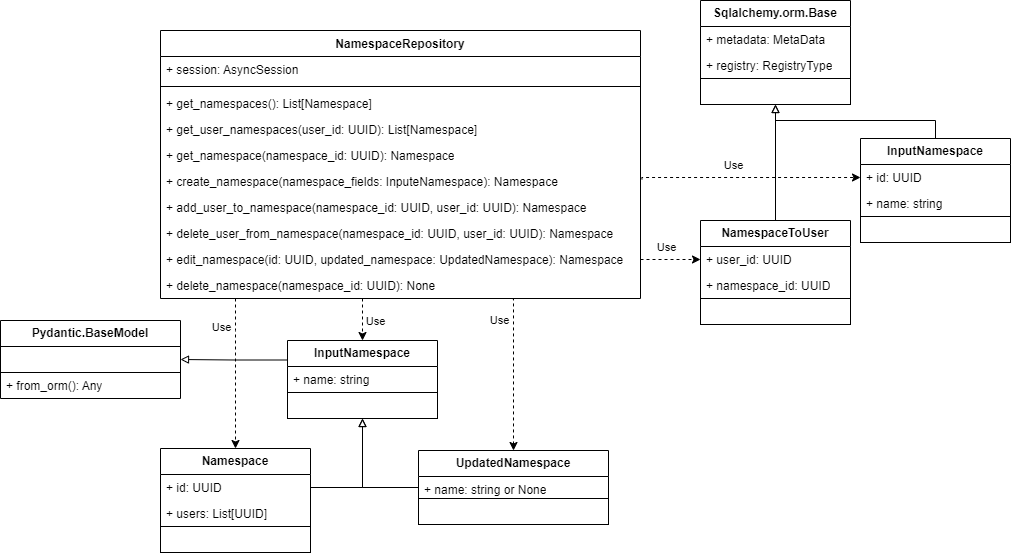


Рисунок 11 — диаграмма классов сервиса пространства страниц.

Функциональные требования, предъявляемые к сервису пространства страниц, реализуются при помощи методов контроллера NamespaceRepository. В таблице 13 приведено описание каждого метода данного контроллера.

Таблица 3. Спецификация класса NamespaceRepository

|  |  |
| --- | --- |
| Метод | Описание |
| Get\_namespaces | Получение списка пространств страниц |
| Get\_user\_namespaces | Получение пользовательского пространства страниц |
| Get\_namespace | Получение пространства страниц |
| Create\_namespace | Создание пространства страниц |
| Add\_user\_to\_namespace | Добавление пользователя в пространство страниц |
| Delete\_user\_to\_namespace | Удалить пользователя из пространства страниц |
| Edite\_namespace | Редактирование пространства страниц |
| Delete\_namespace | Удаление пространство страниц |

На рисунке 12 представлена диаграмма классов для разработки сервиса категорий.

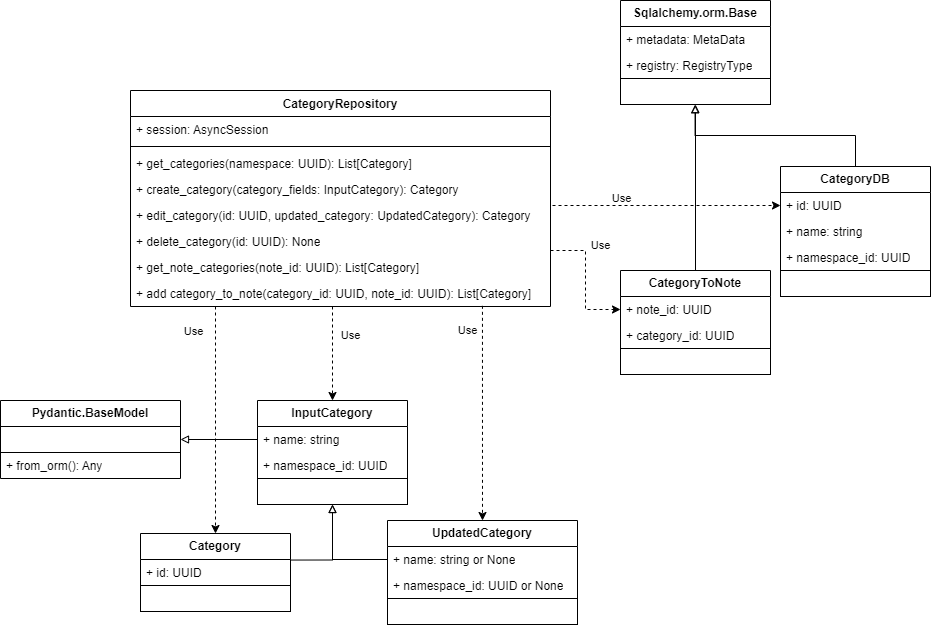


Рисунок 12 — диаграмма классов сервиса категорий.

Функциональные требования, предъявляемые к сервису категорий, реализуются при помощи методов контроллера CategoryRepository. В таблице 14 приведено описание каждого метода данного контроллера.

Таблица 4. Спецификация класса CategoryRepository

|  |  |
| --- | --- |
| Метод | Описание |
| Get\_categories | Получение списка категорий |
| Create\_category | Получение категории |
| Edit\_category | Редактирование категории |
| Delete\_category | Удаление категории |
| Get\_note\_categories | Получение категории заметки |
| Add\_category\_to\_note | Добавить категорию в заметку |

* 1. Диаграмма последовательности действий

Для описания поведения компонентов информационного портала на единой оси времени используются диаграммы последовательности действий, при помощи которых можно описать последовательность действий для каждого прецедента, необходимую для достижения цели. Например, на рисунке 13 изображен процесс регистрации пользователя.

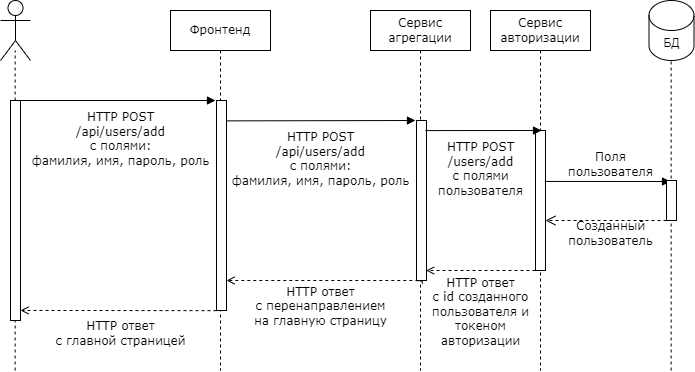


Рисунок 13 — диаграмма последовательностей действий при регистрации пользователя.

На рисунке 14 представлена диаграмма последовательностей при создании заметки.

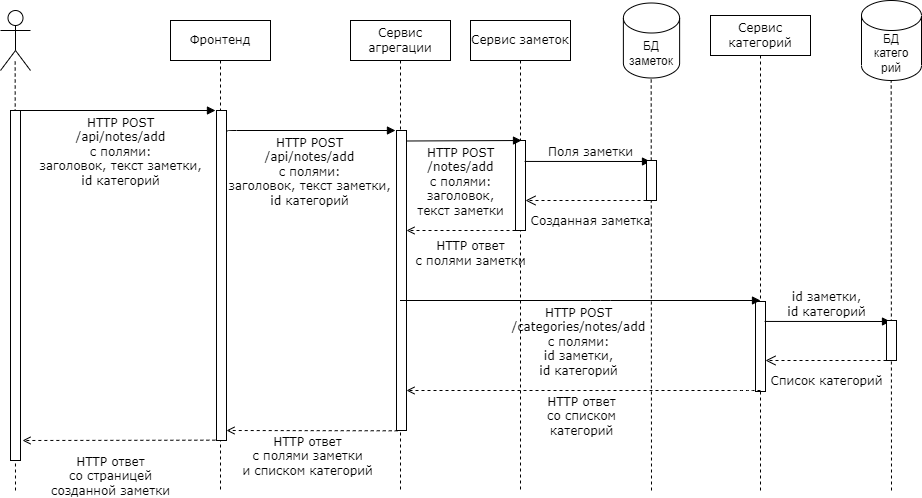


Рисунок 14 — диаграмма последовательности действий для создания заметки.

* 1. Диаграмма поток данных

Рассматриваемый информационный портал предполагает распределенное хранение данных. Все данные информационного портала предполагают хранение в единой базе данных, хранилищами данных являются таблицы. Диаграмма потоков данных, представленная на рисунке 15, отображает модель информационного портала с точки зрения хранения, передачи и обработки данных во время обработки запроса пользователя на получение пространств страниц.

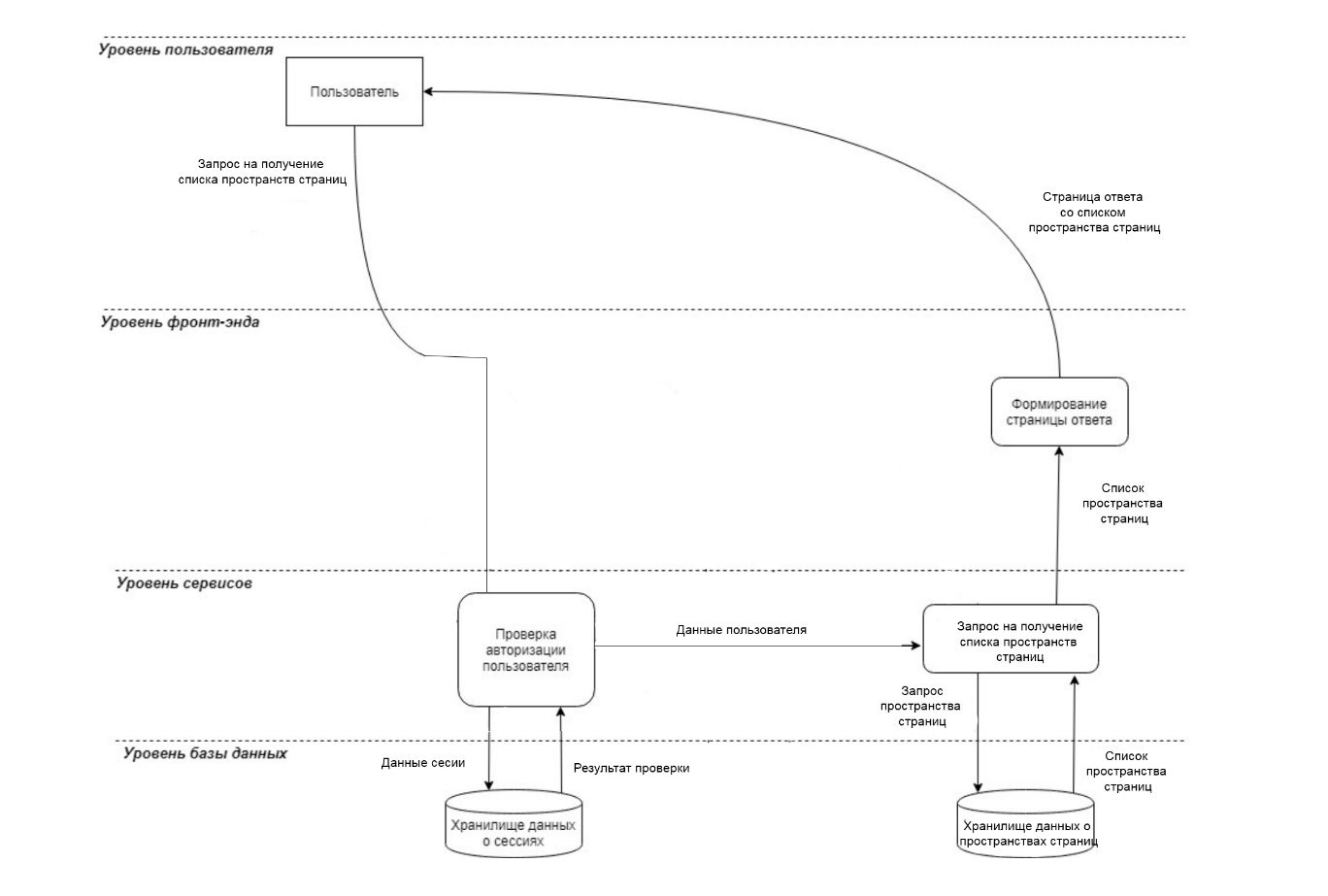


Рисунок 15 — диаграмма потоков данных при запросе пользователем пространств страниц

Диаграмма потоков данных, представленная на рисунке 16, отображает модель информационного портала с точки зрения хранения, передачи и обработки данных во время обработки запроса пользователя на получение списка страниц.

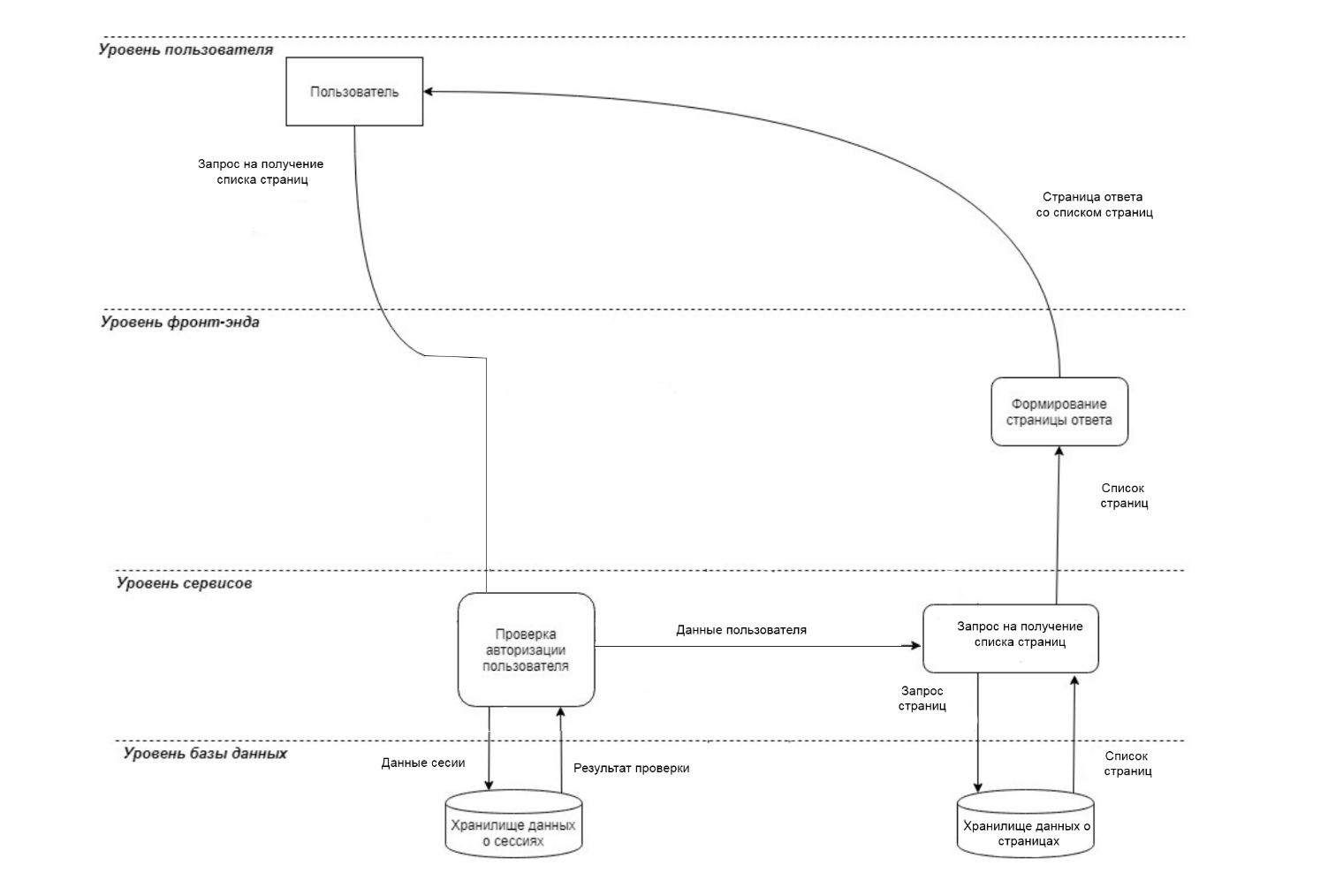


Рисунок 16 — диаграмма потоков данных при запросе пользователем списка страниц

Диаграмма потоков данных, представленная на рисунке 17, отображает модель информационного портала с точки зрения хранения, передачи и обработки данных во время обработки запроса пользователя на получение списка категорий.

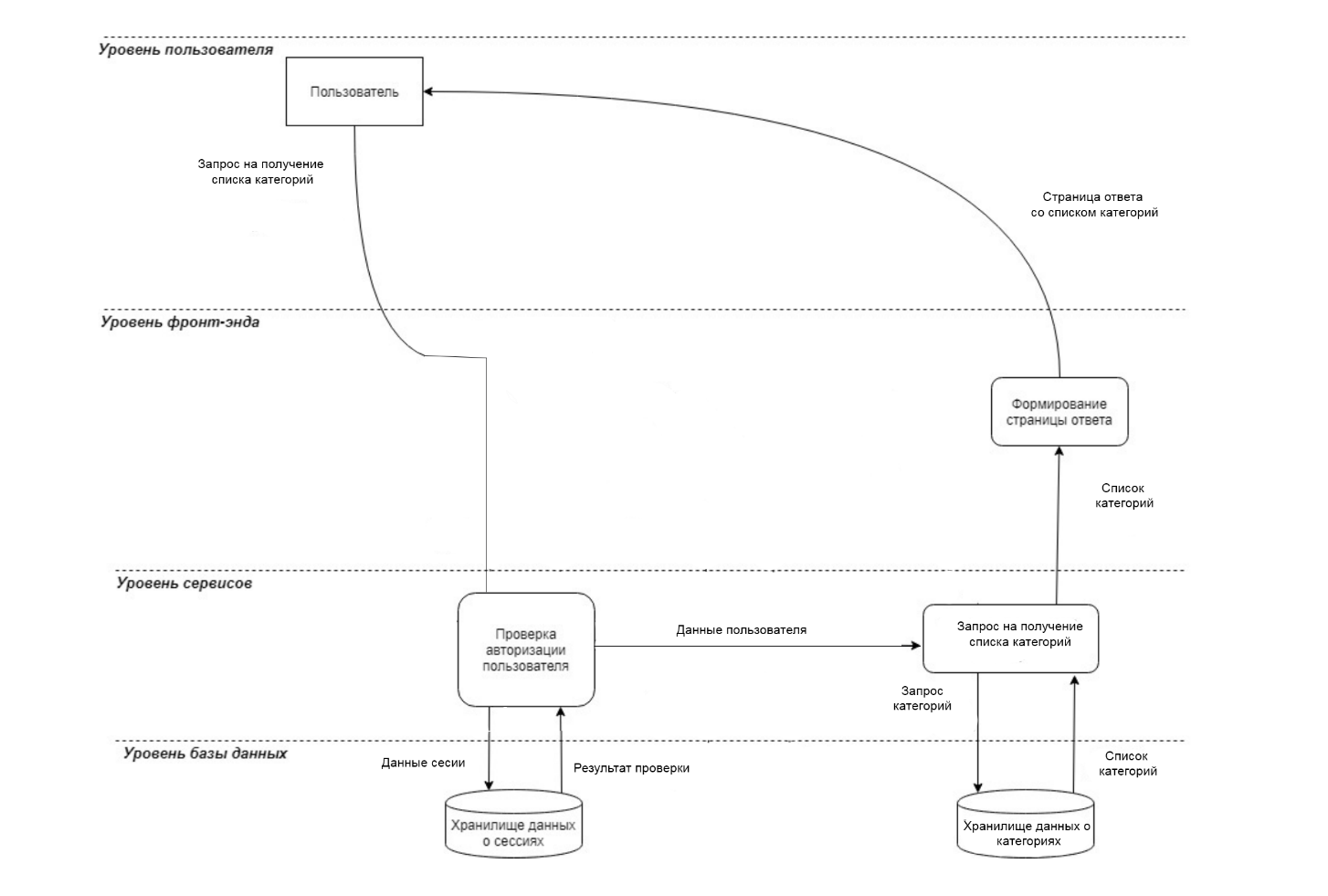


Рисунок 17 — диаграмма потоков данных при запросе пользователем списка категорий

1. Технологический раздел

В этом разделе приведены описания структур данных и описана реализация системы.

* 1. Схема базы данных

В качестве СУБД решено использовать PostgreSQL. PostgreSQL – это объектно-реляционная СУБД с открытым исходным кодоном. Её преимуществами является:

* Поддержка баз данных не ограниченного размера – PostgreSQL не налагает ограничений на размер базы данных. Так же в этой СУБД достаточно высокие максимальные объемы таблицы и поля: 32 терабайта и 1 гигабайт соответственно;
* Надежные механизмы транзакций. Надежность PostgreSQL обеспечивается соответствием принципам ACID (атомарность, изолированность, непротиворечивость, сохранность данных), многоверсионностью, а также наличием Write Ahead Logging (WAL) - общепринятого механизма протоколирования всех существующих транзакций;
* Расширяемость. СУБД обладает поддержкой С-совместимых модулей расширения, что упрощает разработку расширений и в купе с большой базой пользователей рожает огромное количество разнообразных модулей;
* Открытый исходный код. Он позволяет скомпилировать СУБД под необходимое устройство, а также позволяет применять её после проверки в том числе и в решениях для государственного сектора.
* Полная SQL-совместимость. PostgreSQL поддерживает схемы, подзапросы, внешние связки, правила, курсоры, наследование таблиц, триггеры и множество других полезных в разработке БД функций;
* Поддержка многочисленных типов данных. СУБД PostgreSQL поддерживает численные (целые, денежные, с фиксированной/плавающей точкой), булевые, символьные, составные, сетевые типы данных, а также перечисление, типы «дата/время», геометрические примитивы, массивы, XML- и JSON-данные. Плюс можно создавать свои типы данных.

Рисунок 18 показывает общую схему базы данных системы. Каждый сервис имеет свою базу данных, поэтому отношения между большинством таблиц фактически не контролируются на уровне СУБД. Спецификации таблиц приведены в разделах 2.2.4. – 2.2.8.

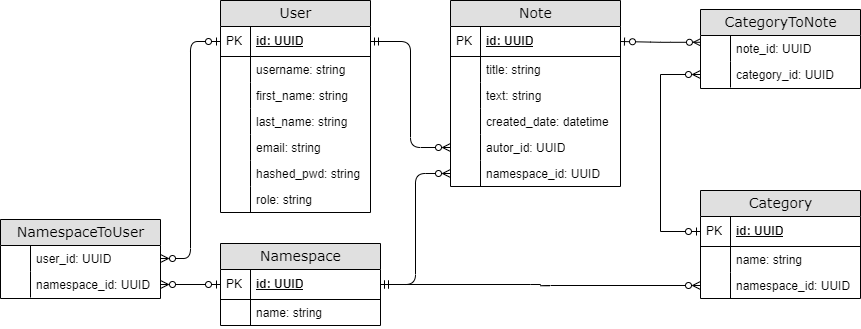


Рисунок 18 — схема базы данных системы.

* 1. Реализация серверной части

Взаимодействие между сервисами осуществляется по протоколу HTTP. В качестве СУБД используется postgresql, взаимодействие с ней осуществляется посредством typeorm. Сервисы авторизации, заметок, простарнства страниц, категорий и статистики имеют связь с базами данных. Каждый из этих сервисов имеет доступ только к своей базе данных.

Проверка авторизации инициируется gateway сервисом и осуществляется по его запросам на сервисе сессий. Вся работа с паролями и jwt токенами осуществляется также на сервисе авторизации. Остальные сервисы авторизацию не проводят. К общедоступным (не защищенным авторизацией) методам относятся методы регистрации и авторизации. Остальные методы требуют наличия заголовка авторизации.

Сервис gateway осуществляет валидацию данных входящих запросов и в случае неудачи возвращает ошибку. Остальные сервисы валидацию не проводят.

Сервис gateway имеет средства обеспечения функциональности системы в случае отказа ее узлов. Если недоступен источник некритичных данных, используется значение-заглушка. В случае недоступности критичных для операции сервисов клиенту возвращается ошибка 503 и в случае операций с транзакциями выполняется их откат.

* 1. Реализация клиентской части

Фронтенд приложение реализовано с использованием фрейморка Vue. Приложение выполнено по принципу SPA, то есть страница сайта загружается только один раз, после чего перерисовывается в соответствии с действиями пользователя. Интерфейс приложения сверстан с использованием библиотеки bootstrap, а также собственной CSS разметки.

Приложение имеет экраны, доступные только авторизованным пользователям и только не авторизованным пользователям. Экраны списка пространства страниц, списка заметок и списка категорий доступны только авторизированным пользователям. Экраны авторизации и регистрации доступны только неавторизованным пользователям. Экран со статистикой работы системы доступен только авторизованным пользователям с ролью Admin.

* 1. Сборка и развертывание системы

Процесс сборки и развертывания системы осуществляется с помощью CI/CD. После прохождения этапов сборки и тестирования сервисы приложения собираются в отдельные докер-контейнеры. Собранные контейнеры публикуются на Docker Hub.

Репозиторий с проектом содержит также файл docker compose, позволяющий развернуть систему локально.

* 1. Тестирование системы

Тестирование системы производилось методом черного ящика. Тестировались сервисы как по отдельности, так и вместе. Прорабатывались различные сценарии, в том числе:

* Регистрация пользователя;
* Авторизация пользователя;
* Просмотр списка пространства страниц;
* Покупка списка заметок и конкретной заметки;
* Просмотр списка категорий и конкретной категории;
* Просмотр статистики;
* Выход из аккаунта.

Для тестирования отказоустойчивости отключались некоторые сервисы и проверялось поведение системы.

В CI/CD также присутствует этап автоматического тестирования. Он проверяет систему на отсутствие ошибок при инстанцировании классов. Более подробное автоматическое тестирование было решено не проводить ввиду отсутствия временных ресурсов на разработку тестов.

Заключение

По результатам работы была спроектирован и разработана система для хранения заметок. Система состоит из 6 сервисов: фронтенда, сервиса пространства страниц, сервиса заметок, сервиса категорий, сервиса авторизации, сервиса статистики.

Сервисы заметок, категорий, авторизации, статистики, пространства страниц имеют доступ к связанным с ними базам данных, но не имеют доступа к базам данных других сервисов. Запросы пользователей требуют авторизацию для всех случаев, кроме запросов регистрации и создания сессии, а все ошибки обрабатываются. В случае недоступности некритичного функционала осуществляется деградация функциональности. Все действия на сервисах логируются на сервисе статистики. Все сервисы собираются и разворачиваются через CI/CD.

# Список литературы

|  |  |
| --- | --- |
| [1] | «Docker Hub» [В Интернете]. URL: https://hub.docker.com/. [Дата обращения: 20.05.2023]. |