Серверная часть системы хранения и обработки датасетов включает в себя несколько независимых сервисов, которые в совокупности обеспечивают функциональность приложения, выполняя функции по взаимодействию с фронтендом, базами данных, RAG-системой и другими компонентами. Такая архитектура называется микросервисной.

Микросервисная архитектура – это подход к разработке программного обеспечения, при котором приложение разбивается на небольшие, независимые сервисы, каждый из которых выполняет одну бизнес-функцию и взаимодействует с другими через API (чаще всего HTTP/REST, gRPC или messaging-брокеры).

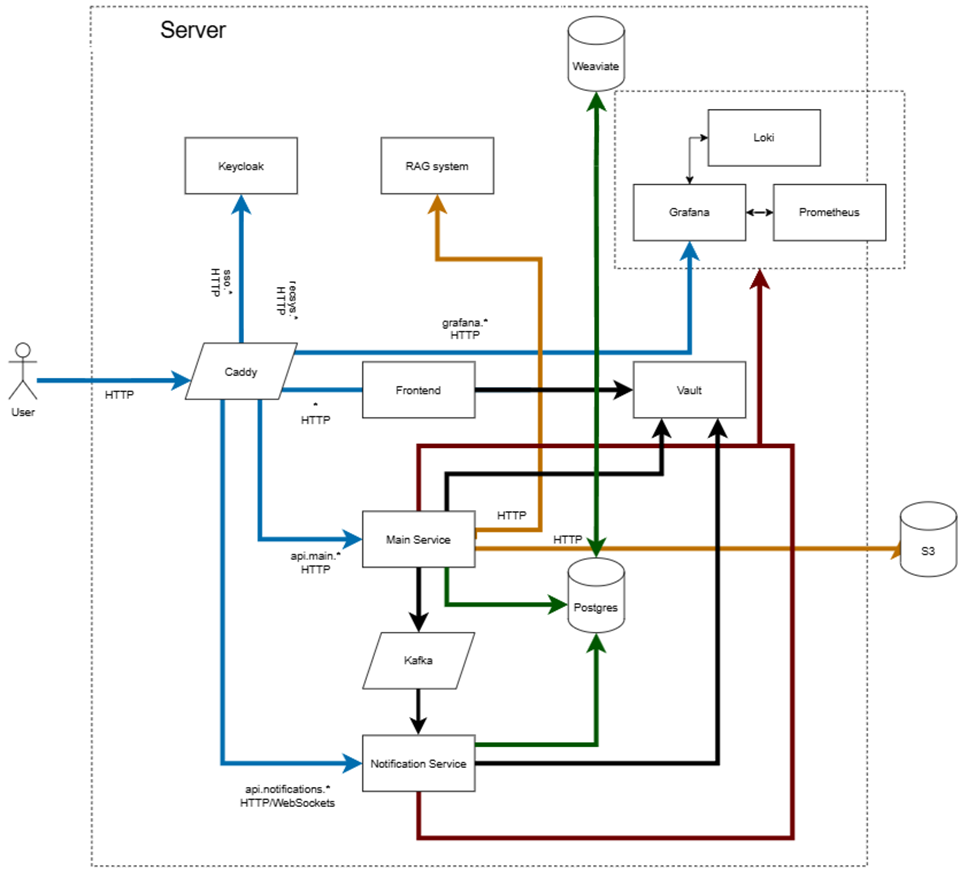


Рисунок 1 – Схема серверной архитектуры приложения

Среди основных характеристик микросервисной архитектуры можно выделить следующие:

* Независимость сервисов. Каждый микросервис разрабатывается, развертывается и масштабируется независимо, может использовать свою БД, язык программирования и фреймворк.
* Слабая связанность. Сервисы общаются через API, а не через общую базу данных, изменения в одном сервисе не должны ломать другие.
* Автономность. У каждого сервиса своя зона ответственности (например: аутентификация, платежи, логирование).
* Масштабируемость. Можно масштабировать только нужные сервисы под нагрузку.
* Отказоустойчивость. Если один сервис отказывает, это не должно приводить к отказу всей системы.

В состав разработанного приложения входят следующие сервисы, схема взаимодействия которых представлена на рисунке 1:

* Основной сервис (Main service). Обеспечивает загрузку и обработку датасетов, управление правами доступа и группами пользователей, менеджмент базы данных, поиск и получение датасетов пользователями;
* Сервис уведомлений (Notification service). Обеспечивает отправку и получение пользователями уведомлений о статусе загрузки датасетов и изменения принадлежности к организациям;
* Сервис по работе с RAG-системой (ML service). Обеспечивает непосредственное взаимодействие с нейросетью в процессе углублённого поиска и рекомендации датасетов;
* Очередь сообщений (Kafka). Реализует очередь задач для ML Service, а также очередь уведомлений для Notification Service;
* Объектное хранилище (S3). Обеспечивает хранение датасетов;
* Основная база данных (PostgreSQL). Обеспечивает хранение общих данных приложения;
* Система аутентификации (Authentik). Обеспечивает авторизацию пользователей;
* Система хранения конфиденциальных данных приложения (Vault);
* Системы сбора логов и метрик приложения (Grafana и Prometheus);
* Прокси-сервер (Caddy). Обеспечивает проксирование трафика на определённые сервисы.

Далее будет подробнее рассмотрена структура и функции каждого сервиса.

**1 Основной сервис**

Основной сервис реализует собой взаимодействие ключевых элементов приложения, а именно сервиса уведомлений, RAG-системы, баз данных и сборщиков логов метрик, между собой.

Сервис написан на языке программирования C# с использованием фреймворка ASP.NET Core.

ASP.NET Core — это современный кроссплатформенный фреймворк с открытым исходным кодом, предназначенный для разработки высокопроизводительных веб-приложений на платформе .NET. В отличие от классического ASP.NET, он обладает улучшенной архитектурой, модульностью и поддерживает работу на различных операционных системах, включая Windows, Linux и macOS.

Разрабатываемый Microsoft в сотрудничестве с открытым сообществом, ASP.NET Core обеспечивает высокую скорость работы, гибкость настройки и широкие возможности для создания как веб-API, так и полноценных веб-приложений. Благодаря встроенной поддержке контейнеризации (Docker) и облачных технологий, он отлично подходит для современных микросервисных и облачных решений.

Ключевыми преимуществами использования ASP.NET Core являются:

* Кроссплатформенность — запуск на разных ОС без изменений кода;
* Высокая производительность — оптимизирован для работы с большими нагрузками;
* Модульность — возможность включать только необходимые компоненты;
* Интеграция с современными технологиями — поддержка gRPC, SignalR, Blazor и других инструментов.
* Гибкость развертывания — возможность публикации как в облаке, так и на локальных серверах.

С помощью ASP.NET Core в рамках приложения реализована логика работы так называемых эндпоинтов – URL-адресов, по которым клиент (например, браузер или мобильное приложение) взаимодействует с сервером. Каждый эндпоинт соответствует определенному маршруту (пути) и HTTP-методу (GET, POST, PUT, DELETE и др.), выполняя конкретную функцию (например, получение данных, добавление записи в базу и т. д.).

Значительная часть созданных эндпоинтов реализует надстройку над базой данных, обеспечивающую создание и изменение хранящихся в ней сущностей.

Основные сущности, такие как профиль пользователя, датасет и его метаданные хранятся в основной базе данных приложения – PostgreSQL.

PostgreSQL – это свободная, открытая и высоконадежная система управления реляционными базами данных (СУБД). Она отличается расширяемостью, соответствием стандарту SQL и поддержкой современных функций, таких как JSON, геопространственные данные, полнотекстовый поиск и др.

Особенностями этой СУБД являются:

* Открытый исходный код – бесплатная лицензия (PostgreSQL License), возможность модификации и использования в коммерческих проектах.
* Кроссплатформенность – работает на Windows, Linux, macOS, FreeBSD и других ОС.
* Поддержка стандартов SQL – высокая совместимость с ANSI SQL, включая сложные запросы, оконные функции, CTE (Common Table Expressions).
* Расширяемость – поддержка пользовательских типов данных, функций, операторов и даже языков программирования (PL/pgSQL, PL/Python, PL/Java и др.).
* Производительность – оптимизированные запросы, индексы (B-tree, Hash, GiST, SP-GiST, GIN, BRIN), параллельная обработка данных.
* Надежность и отказоустойчивость – ACID-совместимость, репликация (master-slave, логическая репликация), точки восстановления (WAL).
* Поддержка NoSQL-функций – работа с JSON/JSONB, XML, HStore (ключ-значение).
* Геопространственные данные – расширение PostGIS для GIS-приложений.
* Безопасность – ролевая модель доступа, SSL-шифрование, аутентификация (LDAP, Kerberos, OAuth).

Схема базы данных, используемой в приложении, представлена на рисунке 2. Реализованы следующие таблицы:

* users, organizations, datasets, notifications – пользователи, организации, датасеты и уведомления соответственно;
* organization\_users, dataset\_users – пользователи, получившие доступ к конкретным организациям или датасетам;
* organization\_roles, dataset\_roles – роли (права), которые могут быть назначены пользователю в рамках конкретной организации или датасета;
* metadatas, tags, data\_language, data\_task, data\_format, data\_type – метаданные датасетов и система тегов, обеспечивающие поиск датасетов и их разделение на соответствующие категории;
* invites – приглашения пользователя по вступлению в организацию или получению доступа к датасету.

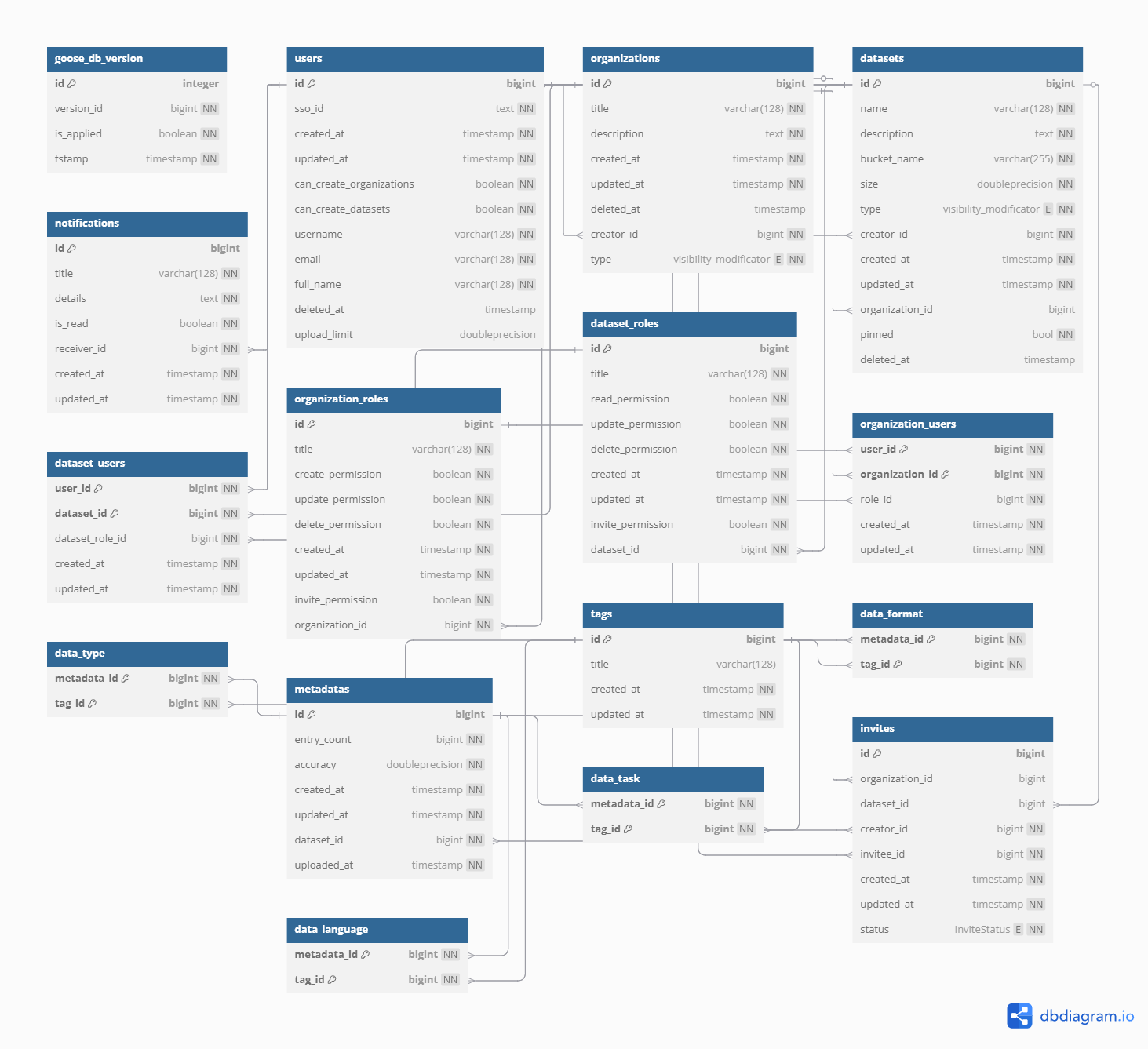


Рисунок 2 – Схема таблиц базы данных

Эндпоинты в ASP.NET Core реализованы с использованием таких технологий как MediatR и Entity Framework Core (EF Core) основывается на принципах CQRS (Command Query Responsibility Segregation) и чистой архитектуры, что обеспечивает разделение ответственности, тестируемость и масштабируемость приложения.

MediatR — это реализация паттерна Mediator (посредник) для .NET, который упрощает взаимодействие между компонентами приложения, уменьшая прямые зависимости между ними.

Entity Framework Core (EF Core) — это объектно-реляционное отображение (ORM) для .NET, которое позволяет работать с базой данных, используя объекты и LINQ, вместо прямых SQL-запросов. EF Core поддерживает различные СУБД (SQL Server, PostgreSQL, SQLite, MySQL и др.) и интегрируется с ASP.NET Core, обеспечивая удобный доступ к данным.

CQRS (Command Query Responsibility Segregation) — это архитектурный паттерн, который разделяет операции чтения (Queries) и записи (Commands) данных, что позволяет оптимизировать производительность, масштабируемость и поддерживаемость приложения. В сочетании с MediatR и Entity Framework Core он часто используется в современных ASP.NET Core-приложениях.

В соответствии с этими принципами, реализованные эндпоинты обрабатывают запросы следующим образом:

1. HTTP-запрос поступает в контроллер ASP.NET Core;
2. Контроллер делегирует обработку запроса MediatR, отправляя команду (для модификации данных) или запрос (для чтения данных);
3. MediatR направляет команду/запрос соответствующему обработчику (IRequestHandler);
4. Обработчик использует DbContext (EF Core) для взаимодействия с базой данных;
5. Результат возвращается через цепочку вызовов обратно в контроллер, который формирует HTTP-ответ;

Основными программными компонентами, реализующими эту схему, являются:

1. Контроллер (API Endpoint), отвечающий за прием HTTP-запросов и возврат ответов. Не содержит бизнес-логики, только валидацию входных данных, отправку команды/запроса через IMediator, обработку исключений (или их передачу в middleware). Использует атрибуты маршрутизации ([HttpGet], [HttpPost] и т. д.) и [ApiController];
2. Команды и запросы (CQRS) изменяют состояние системы (создание, обновление, удаление), реализуют интерфейс IRequest или IRequest<TResponse> и получают данные без изменения состояния соответственно;
3. Обработчики (Handlers) – реализуют IRequestHandler<TRequest, TResponse>, содержат основную бизнес-логику и взаимодействуют с базой данных через DbContext (EF Core), используют репозитории (дополнительный уровень абстракции) и AutoMapper для преобразования сущностей в DTO (Data Transfer Objects) для передачи данных, реализуют валидацию и логирование;
4. Репозиторий (Repository) – реализует непосредственно запросы к базе данных и возврат ответов.

Таким образом, жизненный цикл запроса представляется следующим образом:

1. HTTP-запрос → Контроллер → MediatR;
2. MediatR → Обработчик → EF Core → База данных;
3. База данных → EF Core → Обработчик → MediatR → Контроллер → HTTP-ответ.