ПРОЕКТ «Воспоминания»

Автор

Андрей Фесенко

Оглавление

[ПРОЕКТ «Воспоминания» 1](#_Toc200321096)

[Описание проекта. 1](#_Toc200321097)

[🧱 Архитектурная схема (уровень сервисов) 1](#_Toc200321098)

[🧩 Общая структура микросервисов (чистая архитектура) 2](#_Toc200321099)

[📋 Таблица микросервисов 3](#_Toc200321100)

[БД, сводная таблица всех схем, таблиц, их назначений 4](#_Toc200321101)

[Ключевые индексы и связи БД 6](#_Toc200321102)

[Контекст БД. 8](#_Toc200321103)

[Создание .NET-решения для микросервисного проекта «Воспоминания» с использованием чистой архитектуры. 9](#_Toc200321104)

[Разработка микросервиса IdentityService 10](#_Toc200321105)

[Разработка микросервиса ProfileService. 14](#_Toc200321106)

2025

# ПРОЕКТ «Воспоминания»

# Описание проекта.

Проект создан для сохранения фото, видео, текстового блога для своих потомков.

Каждый из нас проживает уникальную жизнь, наполненную важными событиями, эмоциями и мыслями. Часто мы не успеваем или не знаем, как передать это своим детям, внукам, друзьям. Проект «Воспоминания» предлагает решение — цифровой архив, доступ к которому можно открыть своим близким при жизни или после ухода. Пользователь загружает фото, видео, ведёт текстовой дневник, записывает послания. Он сам определяет, кто и когда сможет получить доступ к этим материалам — немедленно или только после того, как его уже не станет. Форма наследия в цифровую эпоху Мы создаём не просто сервис, а новую форму цифрового завещания — не юридического, а душевного. Проект «Воспоминания» помогает людям осознанно оставить след и быть рядом с близкими даже после ухода.

Как это работает- Основной пользователь создаёт аккаунт и наполняет его воспоминаниями: фотоальбомами, видеороликами, личными письмами, блогом, голосовыми посланиями. В настройках он указывает список близких людей, которым будет открыт доступ к архиву, а также условия: немедленно или после смерти. После ухода пользователя, доверенные лица вводят дату смерти, и система автоматически активирует доступ к контенту и рассылает уведомления по заранее подготовленным каналам — SMS, email, мессенджеры. Родные и друзья могут добавлять свои воспоминания и фото, которые становятся частью единого цифрового памятника.

Ключевые функции Цифровой дневник жизни — медиаархив, наполненный воспоминаниями. Наследование доступа — управление правами и моментом раскрытия информации. Пожелания и прощания — возможность оставить последнее слово своим близким. QR-ссылка на вечную память — для размещения на памятнике, надгробии или в мемориальной книге. Модерация контента — близкие подтверждают публикации, добавленные после смерти пользователя.

Примерный технологический стек - Backend: C# .NET 8 Web API Frontend: React + Tailwind / Blazor WASM БД: PostgreSQL / MongoDB Очереди: RabbitMQ / Azure Service Bus Хранилище: AWS S3 / Azure Blob Storage Идентификация: IdentityServer / Duende / Auth0 Контейнеризация: Docker + Kubernetes Логирование и мониторинг: ELK / Grafana + Prometheus / Serilog + Seq.

# 🧱 Архитектурная схема (уровень сервисов)

**1. API Gateway**

* Входная точка во все микросервисы
* Авторизация и маршрутизация
* Поддержка CORS, rate-limiting и логирования

**2. Identity Service**

* Регистрация/аутентификация
* Управление ролями: держатель аккаунта, близкий, гость
* Поддержка Duende IdentityServer / Auth0

**3. User Profile Service**

* Профиль основного пользователя
* Настройки доступа, список доверенных лиц
* Контакты для уведомлений, дата смерти и политика доступа

**4. Memory Archive Service**

* Работа с воспоминаниями: фото, видео, блог, голос
* Хранение метаданных, связка с S3 / Blob Storage
* Поддержка публичного и приватного контента

**5. Access Control Service**

* Обработка триггера "дата смерти"
* Активация доступа, смена ролей
* Проверка условий доступа

**6. Notification Service**

* Рассылка по Email/SMS/мессенджерам
* Шаблоны уведомлений
* Очереди на основе RabbitMQ или Azure Service Bus

**7. Moderation Service**

* Модерация внешнего контента (от других пользователей)
* Интерфейс для подтверждения публикаций близкими

**8. QR Code Service**

* Генерация и хранение QR ссылок
* Служит как публичная точка входа

**9. Audit & Logging Service**

* Слежение за действиями в системе
* Интеграция с Serilog, Seq, Grafana/Prometheus

# 🧩 Общая структура микросервисов (чистая архитектура)

Каждый сервис следует схеме:

css

КопироватьРедактировать

/src

/[ServiceName]

/Application ← use cases, interfaces, logic

/Domain ← сущности, value objects, enums

/Infrastructure ← реализация интерфейсов (репозитории, интеграции)

/API ← Web API (контроллеры)

/Persistence ← EF Core / Dapper / Mongo

/Shared ← базовые классы, контракты

🛰️ Взаимодействие между сервисами

Внутреннее общение: gRPC или HTTP REST

Асинхронная интеграция: RabbitMQ / Azure Service Bus

Общие события: UserDeathConfirmed, MemoryAdded, AccessGranted

🔐 Безопасность

JWT + Refresh токены

RBAC: роль определяет доступ к ресурсам

Аудит всех действий, особенно "дата смерти"

🧾 Пример базового списка микросервисов.

|  |  |  |
| --- | --- | --- |
| **Сервис** | **Технологии** | **Примечания** |
| API Gateway | YARP, Ocelot | Rate limiting, routing |
| Identity | Duende / Auth0 + .NET 8 | OAuth2/OpenID, MFA |
| User Profile | .NET 8 Web API + PostgreSQL | Настройки, список доверенных |
| Archive | .NET 8 + AWS S3 | Фото, видео, блоги, голос |
| Access Control | .NET 8 + Redis / PostgreSQL | Триггеры, смена ролей |
| Notifications | .NET 8 + RabbitMQ + Twilio | Email/SMS/messenger |
| Moderation | .NET 8 + MongoDB/PostgreSQL | Подтверждение публикаций |
| QR Code | .NET 8 + QRCoder | Ссылки на надгробия |
| Logging & Monitoring | Serilog + Grafana/Prometheus | Отдельный стек логов и метрик |

**🔄 Типы коммуникации**

|  |  |  |
| --- | --- | --- |
| **Тип** | **Протокол** | **Примеры использования** |
| Синхронный | HTTP/gRPC | Авторизация, CRUD-операции в админке |
| Асинхронный | RabbitMQ/SB | Триггеры смерти, уведомления, подтверждения |
| Публичный доступ | HTTPS (REST) | QR-код, публичный просмотр воспоминаний |

# 📋 Таблица микросервисов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название сервиса | Описание | Роль в системе | Технологии |
| 1 | API Gateway | Входная точка для клиентов. Маршрутизация, авторизация, логирование. | Централизация доступа к сервисам | YARP / Ocelot, .NET 8, Serilog |
| 2 | Identity Service | Регистрация, логин, управление токенами и ролями пользователей. | Управление пользователями и безопасностью | Duende IdentityServer / Auth0, .NET 8, PostgreSQL |
| 3 | User Profile Service | Профиль держателя аккаунта, список доверенных лиц, дата смерти. | Управление персональными данными и правилами доступа | .NET 8 Web API, PostgreSQL |
| 4 | Memory Archive Service | Хранение и управление медиа (фото, видео, блог, голосовые записи). | Основной контентный хаб | .NET 8, AWS S3 / Azure Blob, MongoDB/PostgreSQL |
| 5 | Moderation Service | Проверка и одобрение контента, добавленного другими после смерти. | Гарантия корректности и уважительности памяти | .NET 8, PostgreSQL, SignalR (по желанию), gRPC |
| 6 | Access Control Service | Обработка ввода даты смерти. Активация доступа и прав. | Триггер переключения состояний: «живой» → «посмертный доступ» | .NET 8, Redis / PostgreSQL, Event Bus |
| 7 | Notification Service | Рассылка сообщений по SMS, email, мессенджерам. | Связь с доверенными лицами, нотификация | .NET 8, RabbitMQ / Azure SB, Twilio / SendGrid |
| 8 | QR Code Service | Генерация QR-кода для доступа к воспоминаниям с надгробия и т.п. | Упрощение публичного доступа через печатные или физические носители | .NET 8, QRCoder, Blob Storage / CDN |
| 9 | Audit & Logging Service | Хранение и просмотр истории действий пользователей и сервисов. | Безопасность, отслеживание активности, отладка | Serilog + Seq / Grafana + Prometheus, Elasticsearch |
| 10 | Content Approval Service *(опционально)* | Автоматизация процесса подтверждения внешнего контента. | Интеграция UI для модерации от близких | Blazor WASM / React + gRPC, PostgreSQL / Redis |
| 11 | Public View Service *(опционально)* | Просмотр общедоступного архива воспоминаний по QR. | Чтение публичной части профиля после разрешения | .NET 8 Minimal API, CDN, MongoDB |
|  |  |  |  |  |

🧩 Связь между сервисами

* Взаимодействие: gRPC или HTTP REST
* Интеграционные события: RabbitMQ / Azure Service Bus
* Контракты: вынесены в отдельную общую сборку Memories.Contracts

🗂️ Общие практики

* Docker: каждый сервис контейнеризован.
* Kubernetes / Azure Container Apps: для деплоя и масштабирования.
* CI/CD: GitHub Actions / Azure DevOps.
* Observability: Prometheus + Grafana / ELK.

# БД, сводная таблица всех схем, таблиц, их назначений

**🔐 identity — Аутентификация и авторизация**

| **Таблица** | **Назначение** |
| --- | --- |
| Users | Основные учетные записи пользователей |
| Roles | Роли в системе: держатель аккаунта, близкие, гости |
| UserRoles | Привязка пользователей к ролям (many-to-many) |
| RefreshTokens | Хранение активных и истекших refresh-токенов |

**👤 profile — Личные данные и настройки**

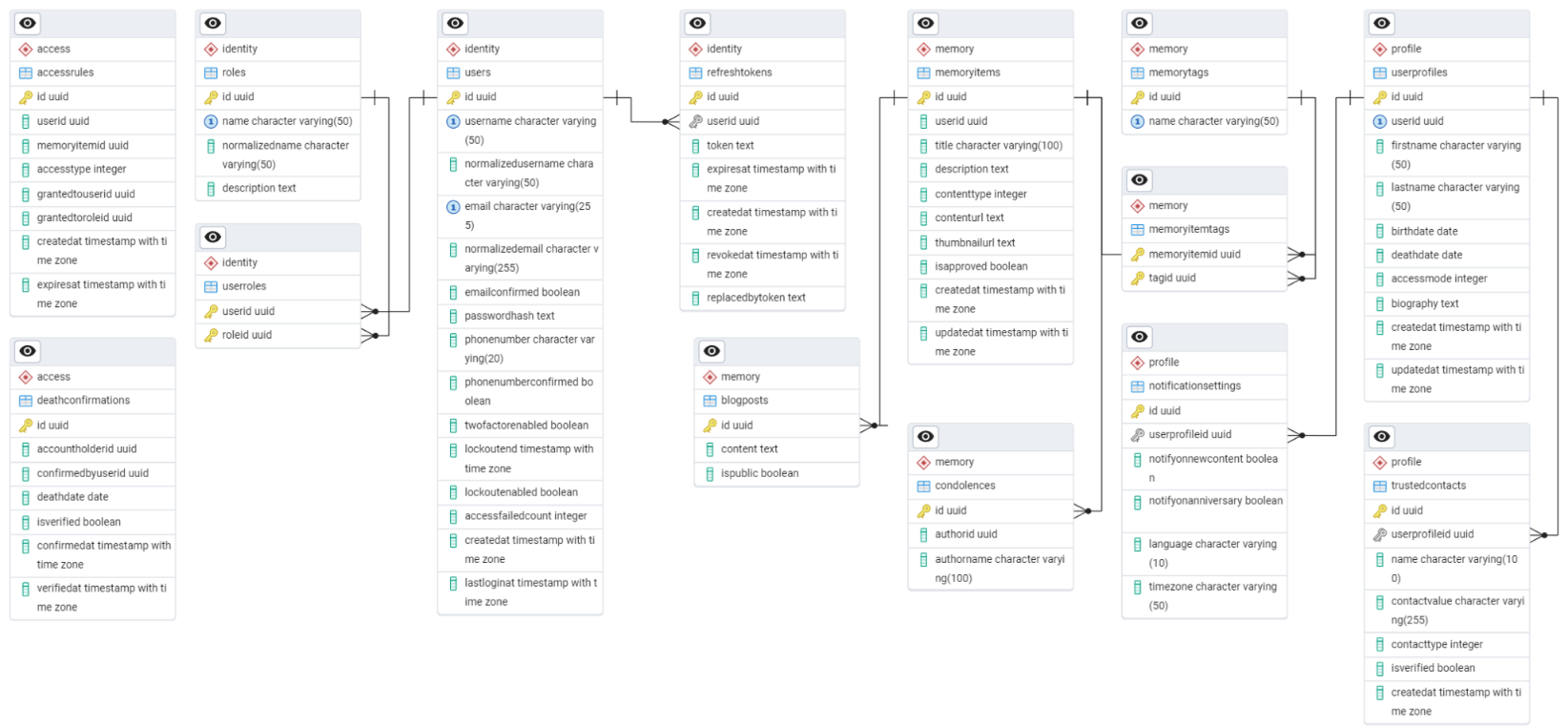
| **Таблица** | **Назначение** |
| --- | --- |
| UserProfiles | Расширенные профили с биографией, датой смерти, политикой доступа |
| TrustedContacts | Список доверенных лиц для уведомлений/подтверждений |
| NotificationSettings | Настройки нотификаций, локализация, часовой пояс |

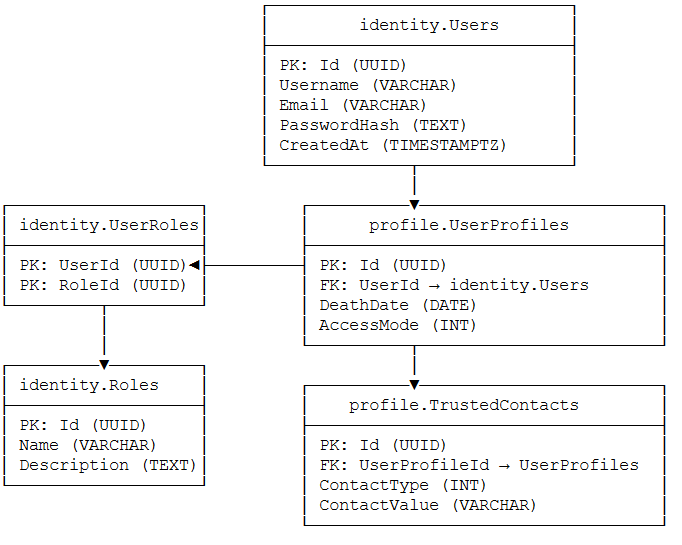
**🧠 memory — Воспоминания, контент, блог**

| **Таблица** | **Назначение** |
| --- | --- |
| MemoryItems | Базовая сущность воспоминания (фото, видео, текст, аудио) |
| MemoryTags | Теги для группировки и навигации |
| MemoryItemTags | Связь "многие ко многим" между воспоминаниями и тегами |
| BlogPosts | Текстовые записи блога (расширение MemoryItems) |
| Condolences | Соболезнования от других пользователей (автор и имя) |

**🔐 access — Управление доступом и событиями**

| **Таблица** | **Назначение** |
| --- | --- |
| AccessRules | Контроль прав доступа к контенту: пользователю или роли |
| DeathConfirmations | Подтверждение смерти (триггер раскрытия контента и уведомлений) |







# Ключевые индексы и связи БД

1. Основные индексы:

   ► identity.idx\_refresh\_tokens\_user\_id - Ускорение поиска токенов по пользователю

   ► profile.idx\_user\_profiles\_user\_id - Уникальный индекс связи с identity.Users

   ► memory.idx\_memory\_items\_user\_id - Быстрый доступ к контенту пользователя

   ► access.idx\_death\_confirmations\_account\_holder\_id - Поиск подтверждений смерти

2. Внешние ключи:

   ► profile.UserProfiles.UserId → identity.Users.Id (CASCADE)

   ► memory.MemoryItems.UserId → identity.Users.Id (RESTRICT)

   ► access.AccessRules.UserId → identity.Users.Id (CASCADE)

3. Триггеры:

   ► Автоматическое обновление updated\_at при изменениях

   ► Валидация ContentUrl при вставке в memory.MemoryItems

► Проверка DeathDate при подтверждении смерти

4. Специальные ограничения:

   ► CHECK (profile.UserProfiles.DeathDate > BirthDate)

   ► UNIQUE (memory.MemoryTags.Name)

   ► CHECK (access.AccessRules.GrantedToUserId

IS NOT NULL OR GrantedToRoleId IS NOT NULL)

🔗 Внешние ключи (FOREIGN KEYS)

Внешние ключи обеспечивают ссылочную целостность между связанными таблицами.

| **Таблица** | **FK поле** | **Ссылается на** | **Действие при удалении** |
| --- | --- | --- | --- |
| UserRoles | UserId | Users(Id) | ON DELETE CASCADE |
|  | RoleId | Roles(Id) | ON DELETE CASCADE |
| RefreshTokens | UserId | Users(Id) | ON DELETE CASCADE |
| UserProfiles | UserId | Users(Id) | ON DELETE CASCADE |
| TrustedContacts | UserProfileId | UserProfiles(Id) | ON DELETE CASCADE |
| NotificationSettings | UserProfileId | UserProfiles(Id) | ON DELETE CASCADE |
| MemoryItems | UserId | Users(Id) | ON DELETE CASCADE |
| MemoryItemTags | MemoryItemId, TagId | MemoryItems, MemoryTags | ON DELETE CASCADE |
| BlogPosts | Id | MemoryItems(Id) | ON DELETE CASCADE |
| Condolences | Id, AuthorId | MemoryItems, Users | ON DELETE CASCADE |
| AccessRules | UserId, MemoryItemId | Users, MemoryItems | ON DELETE CASCADE |
|  | GrantedToRoleId | Roles(Id) | ON DELETE CASCADE |
| DeathConfirmations | AccountHolderId, ConfirmedByUserId | Users(Id) | ON DELETE CASCADE |

**⚡ Индексы**

| **Таблица** | **Индексированные поля** | **Назначение** |
| --- | --- | --- |
| Users | UserName, Email | Быстрый поиск и уникальность |
| RefreshTokens | UserId, Token | Быстрый доступ и валидация токенов |
| UserProfiles | UserId | Связь с пользователем |
| TrustedContacts | UserProfileId | Поиск контактов по профилю |
| MemoryItems | UserId, CreatedAt, ContentType | Фильтрация и сортировка |
| MemoryItemTags | MemoryItemId, TagId | Соединения many-to-many |
| AccessRules | UserId, MemoryItemId | Проверка доступа |
| DeathConfirmations | AccountHolderId | Триггеры доступа после смерти |

**🔔 Триггеры (TRIGGERS)**

**📌 Триггер для UserProfiles.UpdatedAt**

Автообновление даты обновления:

CREATE OR REPLACE FUNCTION update\_updated\_at\_column()

RETURNS TRIGGER AS $$

BEGIN

NEW.UpdatedAt = NOW();

RETURN NEW;

END;

$$ language 'plpgsql';

CREATE TRIGGER trg\_user\_profiles\_updated\_at

BEFORE UPDATE ON profile.UserProfiles

FOR EACH ROW

EXECUTE PROCEDURE update\_updated\_at\_column();

**📌 Триггер на DeathConfirmations**

Для автоматической верификации, если подтверждено >= N доверенных контактов (если будет реализовано на уровне БД).

**✅ Ограничения (CONSTRAINTS)**

| **Ограничение** | **Тип** | **Описание** |
| --- | --- | --- |
| UNIQUE(UserName) | Уникальность | Один логин на пользователя |
| UNIQUE(Email) | Уникальность | Один email — один аккаунт |
| UNIQUE(UserId) в UserProfiles | 1:1 связь | Один профиль на одного пользователя |
| AccessMode, ContentType и т.д. | Enum-like | Использование INTEGER для перечислений |

**🧩 Специальные поля**

* CreatedAt, UpdatedAt — у всех сущностей: ✅ отлично для аудита
* CreatedBy, UpdatedBy — введены, можно реализовать soft-audit
* IsApproved, IsVerified — модерация и доверенность

# Контекст БД.

**✅ 1. Целевая модель данных**

«Что хранит система “Воспоминания”»

* Пользовательские данные (имя, даты жизни, роли)
* Контент: фото, видео, тексты, голосовые послания
* Настройки доступа: при жизни / после смерти
* Доверенные лица и уведомления
* Соболезнования и наследуемый архив

**🔄 2. Цепочка событий "После смерти"**

«Как работает база при подтверждении смерти»

* Ввод даты смерти → создание записи в DeathConfirmations
* Активация доступа → AccessRules
* Отправка уведомлений → NotificationService
* Разрешение соболезнований → Moderation

**🛡️ 3. Безопасность и контроль доступа**

* Шифрование паролей (hashing)
* Уникальные токены доступа (RefreshTokens)
* Управление ролями (RBAC)
* Верификация доверенных лиц
* Аудит действий (CreatedBy, UpdatedBy)

**📈 4. Масштабируемость и модульность**

* Используются схемы (identity, profile, memory, access) — можно масштабировать и изолировать по контекстам.
* Возможна миграция к NoSQL для медиа-метаданных.
* Микросервисный подход позволяет выделить каждую часть (например, ModerationService, NotificationService).

**📦 5. Преимущества структуры данных для бизнеса**

* 💡 Готовность к расширению (например, API для нотариусов или мемориальных агентств)
* 🧬 Гибкость политики доступа (наследуемый контент, публичные QR)
* 💬 Сбор обратной связи и историй (соболезнования — engagement)
* 💰 Возможность создания платных функций (например, расширенные медиатеги, хранение навсегда, премиум-уведомления)

# Создание .NET-решения для микросервисного проекта «Воспоминания» с использованием чистой архитектуры.

Пример структуры

Memories.sln

│

├── src/

│ ├── **IdentityService**/

│ │ ├── IdentityService.API # Веб-интерфейс

│ │ ├── IdentityService.Application # Use Cases

│ │ ├── IdentityService.Domain # Сущности, интерфейсы

│ │ └── IdentityService.Infrastructure # EF, Auth0, DB

**🧪 Используемые пакеты (для всех сервисов)**

| **Тип** | **Пакеты** |
| --- | --- |
| Web API | Microsoft.AspNetCore.OpenApi, Swashbuckle.AspNetCore |
| EF Core | Microsoft.EntityFrameworkCore, Npgsql.EntityFrameworkCore.PostgreSQL |
| Валидация | FluentValidation |
| IoC / Mediator | MediatR, MediatR.Extensions.Microsoft.DependencyInjection |
| Логирование | Serilog.AspNetCore, Serilog.Sinks.Console |
| Auth | Microsoft.AspNetCore.Authentication.JwtBearer |

# Разработка микросервиса IdentityService

**IdentityService** — это ключевой микросервис, отвечающий за управление пользователями, их аутентификацию, авторизацию, роли и безопасность во всей экосистеме проекта «Воспоминания».  
Этот сервис реализован по принципам чистой архитектуры (Clean Architecture) и использует современные инструменты .NET.

Используемые технологии

* C# .NET 8 — основной язык и платформа
* ASP.NET Core Web API — построение REST API
* Entity Framework Core (EF Core) + PostgreSQL — ORM для работы с БД
* MediatR — CQRS-паттерн, обработчики команд и запросов
* FluentValidation — валидация входящих данных
* Serilog — структурированное логирование и аудит
* Swagger/OpenAPI — автогенерация документации и тестирование API
* JWT — безопасная аутентификация с помощью токенов
* BCrypt — безопасное хеширование паролей
* Docker (опционально) — контейнеризация сервиса
* xUnit, Moq, FluentAssertions — модульное тестирование

Архитектура и структура проекта

Сервис построен на принципах чистой архитектуры (Clean Architecture) и разделения по слоям:

Основные слои

* Domain
  + Содержит бизнес-сущности (User, Role, RefreshToken и др.)
  + Не зависит ни от каких технологий (максимальная “чистота”)
* Application
  + CQRS-обработчики (Command/Query Handlers)
  + DTO, команды, интерфейсы сервисов и репозиториев
  + Валидация команд
* Infrastructure
  + Работа с внешними системами: реализация репозиториев (EF Core), сервисы отправки писем и др.
  + Реализация интерфейсов Application-слоя
* API
  + Контроллеры (REST endpoints)
  + Конфигурация приложения
  + DI (Dependency Injection), Swagger, авторизация

**Взаимодействие слоёв**

* API зависит от Application и Infrastructure
* Application зависит только от Domain
* Domain не зависит ни от кого

Описание всех подсервисов и компонентов

**Пользователи и роли**

* **User** — основная сущность пользователя (Id, Email, PasswordHash, роли, блокировки, т.д.)
* **Role** — сущность роли (например, AccountHolder, RegularUser, Admin)
* **UserRole** — связь многие-ко-многим между User и Role

**Аутентификация и авторизация**

* **JWT** — для выдачи access-токенов (и refresh-токенов)
* **RefreshToken** — обновление access-токена без повторного ввода пароля
* **Password hashing (BCrypt)** — безопасное хранение паролей

**Безопасность**

* **Lockout** — после 5 неудачных попыток входа пользователь блокируется на 15 минут
* **Email confirmation** — подтверждение email пользователя по ссылке
* **Reset password** — восстановление пароля по ссылке

**Аудит и логирование**

* **Serilog** — логирует действия (входы, смену пароля, удаление, админские действия)
* **IAuditService** — отдельный сервис для логирования действий в базе/файле

**CRUD операций**

* **Регистрация** — создание нового пользователя
* **Вход** — аутентификация, выдача access/refresh токена
* **Обновление** — смена email, пароля
* **Удаление** — удаление пользователя (админ)
* **Назначение ролей** — CRUD ролей, назначение/отзыв ролей
* **Админ-панель** — endpoints для просмотра/удаления пользователей, управления ролями

**Интеграция с внешними сервисами**

* **ProfileService** — для получения профиля и пользовательских claims
* **EmailSender** — (опционально) отправка писем для подтверждения/сброса пароля

Реализация слоев — по шагам

**Domain Layer (IdentityService.Domain)**

* **Сущности**:
  + User — все поля, связанные с учётной записью (Id, Email, PasswordHash, Roles, LockoutEnd и пр.)
  + Role — роли пользователей
  + RefreshToken, UserRole — дополнительные сущности
* **Особенности**:
  + Нет ссылок на EF Core, MediatR и т.п.
  + Все сущности — POCO-классы

**Application Layer (IdentityService.Application)**

* **Команды и запросы**:
  + LoginCommand, RegisterCommand, ChangePasswordCommand, DeleteUserCommand, AssignRoleCommand, и др.
  + DTO для входа/выхода данных (LoginResponse, RegisterResponse и пр.)
* **Обработчики (Handlers)**:
  + Для каждой команды — свой Handler, использующий DI-репозитории и сервисы
* **Интерфейсы сервисов и репозиториев**:
  + IUserRepository, IRoleRepository, IJwtTokenGenerator, IAuditService
* **Валидация**:
  + Классы с наследованием от AbstractValidator<T> (FluentValidation)

**Infrastructure Layer (IdentityService.Infrastructure)**

* **EF Core реализации**:
  + Репозитории: UserRepository, RoleRepository (работа с БД через DbContext)
  + MemoriesDbContext — описывает все таблицы, схемы, индексы, связи
* **Реализация сервисов**:
  + JwtTokenGenerator, RefreshTokenService, AuditService
  + HTTP-клиенты (например, для интеграции с ProfileService)

**API Layer (IdentityService.API)**

* **Контроллеры**:
  + AuthController, AdminController, RolesController и др.
  + REST endpoints: /api/auth/register, /api/auth/login, /api/admin/users, /api/roles, ...
* **Swagger/OpenAPI**:
  + Автоматическая документация всех endpoints
  + Тестирование прямо из браузера
* **DI конфигурация и мидлвары**:
  + Регистрация всех сервисов, репозиториев, обработчиков MediatR, FluentValidation
  + Настройка JWT-аутентификации и авторизации
* **Логирование и аудит**:
  + Включение Serilog, настройка логов в файл и консоль

Примеры работы основных функций

**Регистрация пользователя**

* POST /api/auth/register
* Валидация email/пароля
* Пароль хешируется
* Назначается роль по умолчанию
* Отправляется email для подтверждения

**Вход пользователя**

* POST /api/auth/login
* Проверка email/пароля (BCrypt)
* Если неудачно — увеличивается счетчик, возможна блокировка
* Если удачно — возвращается accessToken, refreshToken

**Получение профиля/ролей**

* GET /api/admin/users
* GET /api/roles
* Для админов: просмотр всех пользователей, их ролей

**Назначение/снятие ролей**

* POST /api/roles/assign
* POST /api/roles/create
* DELETE /api/roles/delete

**Аудит**

* Все админские действия, логины, смены пароля и др. — логируются с привязкой к userId/adminId и сохраняются в лог

В ходе работы изучили

 Принципы **чистой архитектуры** и разделения ответственности по слоям

 Использование **CQRS** и паттерна MediatR для построения handler-based API

 Современные практики безопасности: **JWT**, **refresh-токены**, **BCrypt**, блокировка аккаунта

 Работа с реляционной БД и **EF Core**: миграции, схемы, индексы, связи

 Реализация кастомных репозиториев и сервисов, а также интеграция с внешними сервисами

 Валидация команд через **FluentValidation**

 Использование **Swagger** для самодокументирования и тестирования API

 **Логирование** (Serilog) и построение аудит-трейлов для безопасности

 Принципы **Dependency Injection** (DI) и конфигурирования сервисов

 Основы построения REST API для production-ready решения

Доступные функции после реализации IdentityService

* Регистрация, подтверждение email, вход и смена пароля
* Получение/обновление информации о пользователе
* Аудит всех важных действий (входы, ошибки, изменения)
* Админ-панель для управления пользователями и ролями
* Назначение, создание и удаление ролей
* Защита аккаунта от брутфорса (lockout)
* Поддержка refresh-токенов и безопасного выхода (logout)
* Интеграция с другими микросервисами (ProfileService, Email)
* Документация API через Swagger

# Разработка микросервиса ProfileService.

**Минимальная архитектура ProfileService**

**Технологии:**

* ASP.NET Core Web API (.NET 8/9)
* EF Core (PostgreSQL)
* MediatR + Clean Architecture
* OpenAPI (Swagger)
* Логирование (Serilog)

**Основные сущности и требования:**

* **Profile (UserProfile)**
  + Id (GUID)
  + UserId (GUID, внешний ключ на пользователя в Identity)
  + FirstName, LastName, BirthDate, DeathDate
  + AccessMode (AfterDeath/Anytime)
  + Biography, CreatedAt, UpdatedAt
  + DeathConfirmed (bool)
* **TrustedContact** (список доверенных лиц)
  + Id, UserProfileId, Name, ContactValue, ContactType, IsVerified
* **NotificationSettings**
  + Id, UserProfileId, NotifyOnNewContent, NotifyOnAnniversary, Language, TimeZone

**Структура проекта**

ProfileService/

├── ProfileService.API/

├── ProfileService.Application/

├── ProfileService.Domain/

├── ProfileService.Infrastructure/

└── ProfileService.sln