Geekbrains

**Дипломный проект на тему:**

**Разработка универсального веб-приложения, реализующего алгоритм приоритизации ИТ-инцидентов**

Программа:

Разработчик C#. Специалист

Егоров Дмитрий Васильевич

Самара

2024

# Содержание

[Содержание 2](#_Toc162807511)

[Введение 3](#_Toc162807512)

[1. Основы разработки веб-приложений 6](#_Toc162807513)

[1.1. Rest API 7](#_Toc162807514)

[1.2. ASP.Net Core 10](#_Toc162807515)

[1.3. Базы данных 14](#_Toc162807516)

[1.4. Entity Framework 16](#_Toc162807517)

[1.5. Подходы работы с Базой Данных 17](#_Toc162807518)

[2. Техническое решение разрабатываемого приложения 18](#_Toc162807519)

[2.1. Создание сущностей 21](#_Toc162807520)

[2.2. Создание базы данных 22](#_Toc162807521)

[2.3. Создание Data Transfer Objects 23](#_Toc162807522)

[2.4. Создание контроллера 24](#_Toc162807523)

[2.5. Сборка проекта 24](#_Toc162807524)

[3. Демонстрация работы приложения 26](#_Toc162807525)

[Заключение 28](#_Toc162807526)

[Список используемой литературы и интернет-ресурсов 29](#_Toc162807527)

# Введение

Во всех крупных организациях мира важнейшую роль играет процесс ITSM (Управление ИТ-услугами). Поскольку организации все больше полагаются на технологии для обеспечения своей деятельности, ИТ-команды становятся важными деловыми партнерами. Библиотека инфраструктуры информационных технологий (ITIL), особенно ее последняя версия, ITIL 4, служит основой, которая дает ИТ-командам гибкость и стабильность, необходимые для успешной поддержки бизнеса.

ITIL (Information Technology Infrastructure Library — библиотека инфраструктуры информационных технологий) — это общепризнанный набор рекомендаций, призванный помочь организациям максимально эффективно использовать ИТ путем согласования ИТ-услуг с бизнес-стратегией.

ITSM включает все процессы и действия, связанные с проектированием, созданием, предоставлением и поддержкой ИТ-услуг.

Основная идея ITSM заключается в том, что ИТ нужно предоставлять как услугу. Типичный сценарий ITSM может включать запрос на новое оборудование, например ноутбук. Пользователь отправляет запрос через портал, заполняя заявку с указанием всей необходимой информации, и тем самым запускает воспроизводимый рабочий процесс. Заявка помещается в очередь ИТ-команды, где входящие запросы сортируются и рассматриваются в соответствии с их важностью.

Помимо запросов (будь то запросы на изменение, оборудование, доступы или обслуживание), ИТ-команды также занимаются решением поступающих инцидентов.

Важную роль в ITSM процессах играет правильно организованная служба Service Desk, которая выполняет функции первой линии технической поддержки, маршрутизации и координации инцидентов и запросов. Экспертный Service Desk может оказывать услуги отчетности, прикладного администрирования систем, управления доступами к информационным системам. Предоставлять услуги второй линии поддержки.

В Service Desk по телефону и в чате чаще всего обращаются внутренние сотрудники компании, для консультации по созданию заявок, уточнения их статуса и консультаций на первой линии поддержки. Исследуя обращения в техподдержку, выявлено, что почти каждое пятое обращение связано с уточнением статуса заявки. Мы проанализировали причины повторных обращений по заявкам и поняли, что многие пользователи не готовы ждать крайнего срока исполнения из-за объективных внешних обстоятельств. Или бывают случаи, когда крайний срок решения заявки истёк, но она ещё не решена.

В ИТ-службе Компании не был предусмотрен документ, регламентирующий действия сотрудника первой линии поддержки, при поступлении запросов на приоритизацию и ускорение сроков решения инцидентов. Поэтому подобные обращения обрабатывались бессистемно. Сотрудники субъективно принимали решение относительно обоснованности требования пользователя об ускорении решения запроса. Это могло приводить к тому, что более высокий приоритет получали ординарные запросы, а заявки от ключевых пользователей решались в порядке очереди.

Также этот подход мог приводить к созданию второй очереди из приоритизированных запросов в некоторых ИТ-командах.

В результате, в 2024 году мы решили внедрить автоматизированный алгоритм приоритизации инцидентов. Таким образом принятие решения по ускорению будет приниматься на уровне алгоритма и исключен человеческий фактор.

В этом алгоритме мы заложили всего два критерия, что позволит производить ускорение заявок действительно в исключительных случаях и не приведет к появлению второго уровня очереди из ускоренных запросов.

Первый критерий – истечение крайнего срока по заявке. Это действительно справедливая причина для рассмотрения инцидента или запроса в приоритете.

Второй – запрос на приоритизацию от ключевого сотрудника. Мы предложим Бизнесу управлять списком привилегированных сотрудников по ключевым направлениям. Это люди, которые будут получать эскалации от пользователей по заявкам в их Бизнес вертикалях и принимать решение о необходимости решения той или иной заявки ранее остальных. Получится, что менеджмент заказчика ИТ услуг будет самостоятельно оценивать важность ИТ заявки.

Именно автоматизация этого алгоритма и полное исключение из процесса человека позволит сократить нагрузку на персонал первой линии поддержки, а также благоприятно повлияют на моральное состояние персонала, ввиду того что запросы на ускорение часто связаны с негативной эмоциональной окраской.

Алгоритм будет выглядеть таким образом, как изображено на Рис.1.

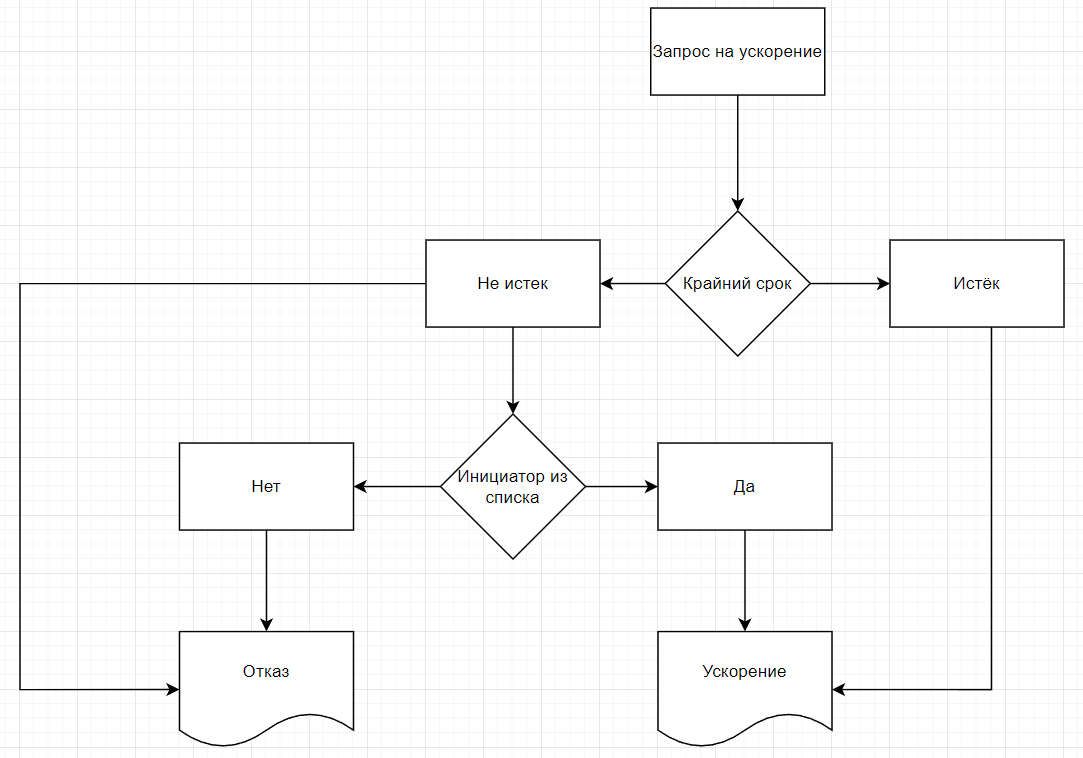


Рис.1.

В рамках данной работы будет спроектировано и создано web-приложение, написанное на языке C#. Это будет отдельный сервис (модуль), который можно будет подключить к любой ITSM-системе (система управления заявками) используя WebAPI по REST для проверки статуса инцидента (крайнего срока) и проставления признака приоритизации. С другой стороны, будет API для подключения к порталу самообслуживания, IVR (голосовой помощник), сервису самообслуживания по SMS, в чат-бот и другие каналы взаимодействия с пользователями. Через этот API будут приниматься запросы на ускорение и предоставляться результат обработки запроса инициатору.

Внутри приложения будут методы проверки запроса на соответствие критериям.

Приложение будет разработано на .Net 8 в MS Visual Studio Community 2022 с применением Entity Framework, ASP.Net. БД на MS SQL Lite.

В профессиональной деятельности являюсь руководителем команды разработки. Есть обширный опыт в анализе требований и проектировании информационных систем. Опыт разработки промышленных систем у меня отсутствует.

В рамках данной работы ставлю перед собой следующие задачи:

* Изучить литературу, касающуюся темы исследования
* Рассмотреть основные технологии разработки веб-приложений.
* Разработать приложение, которое можно будет в последствии встроить в промышленную систему
* Сделать Базу Данных для хранения данных и эмуляции работы ITSM-системы
* Продемонстрировать работу системы.

# Основы разработки веб-приложений

Реализация корпоративной информационной системы требует тщательной подготовки. Не надо забывать про важные факторы разработки корпоративных приложений, а именно:

1. Безопасность: Корпоративные веб-приложения часто содержат конфиденциальную информацию, поэтому необходимо обеспечить максимально возможную безопасность приложения и защитить его от взлома и других угроз.

2. Надежность: Приложения должны быть надежными и доступны всегда, когда это необходимо. Важно обеспечить высокую доступность приложения и решать проблемы быстро, если они возникают.

3. Масштабируемость: Корпоративные веб-приложения могут иметь большое количество пользователей и требовать большого количества ресурсов для работы. Важно обеспечить масштабируемость приложения, чтобы оно могло обрабатывать большой трафик и работать на больших объемах данных.

4. Интеграция: Корпоративные веб-приложения часто интегрируются с другими системами и приложениями внутри компании. Важно обеспечить простоту интеграции и возможность обмена данными с другими системами.

5. Модульность: Корпоративные веб-приложения могут иметь сложную логику и требовать большого количества функций. Важно разбить приложение на модули, чтобы упростить его разработку и обслуживание.

6. Отказоустойчивость: Корпоративные веб-приложения должны быть отказоустойчивыми, чтобы обеспечить непрерывную работу приложения даже в случае сбоев или неисправностей в системе.

7. Удобство использования: Корпоративные веб-приложения часто используются сотрудниками компании, поэтому важно обеспечить простоту использования и интуитивное пользовательское интерфейс.

Далее рассмотрим основные технологии, используемые при создании нашего приложения.

# Rest API

REST расшифровывается как REpresentational State Transfer. Это был термин, первоначально введен Роем Филдингом (Roy Fielding), который также был одним из создателей протокола HTTP. Отличительной особенностью сервисов REST является то, что они позволяют наилучшим образом использовать протокол HTTP.

**REST API** — это способ взаимодействия сайтов и веб-приложений с сервером. Его также называют RESTful.

Термин состоит из двух аббревиатур, которые расшифровываются следующим образом.

**API (Application Programming Interface)** — это код, который позволяет двум приложениям обмениваться данными с сервера. На русском языке его принято называть программным интерфейсом приложения.

**REST (Representational State Transfer)** — это способ создания API с помощью протокола HTTP. На русском его называют «передачей состояния представления».

Технологию REST API применяют везде, где пользователю сайта или веб-приложения нужно предоставить данные с сервера. Например, при нажатии иконки с видео на видеохостинге REST API проводит операции и запускает ролик с сервера в браузере. В настоящее время это самый распространенный способ организации API. Он вытеснил ранее популярные способы SOAP и WSDL.

У RESTful нет единого стандарта работы: его называют «архитектурным стилем» для операций по работе с сервером. Принципы REST API

У RESTful есть 7 принципов написания кода интерфейсов.

* **Отделение клиента от сервера (Client-Server).** Клиент — это пользовательский интерфейс сайта или приложения, например, поисковая строка видеохостинга. В REST API код запросов остается на стороне клиента, а код для доступа к данным — на стороне сервера. Это упрощает организацию API, позволяет легко переносить пользовательский интерфейс на другую платформу и дает возможность лучше масштабировать серверное хранение данных.
* **Отсутствие записи состояния клиента (Stateless).** Сервер не должен хранить информацию о состоянии (проведенных операций) клиента. Каждый запрос от клиента должен содержать только ту информацию, которая нужна для получения данных от сервера.
* **Кэшируемость (Casheable).** В данных запроса должно быть указано, нужно ли кэшировать данные (сохранять в специальном буфере для частых запросов). Если такое указание есть, клиент получит право обращаться к этому буферу при необходимости.
* **Единство интерфейса (Uniform Interface).** Все данные должны запрашиваться через один URL-адрес стандартными протоколами, например, HTTP. Это упрощает архитектуру сайта или приложения и делает взаимодействие с сервером понятнее.
* **Многоуровневость системы (Layered System).** В RESTful сервера могут располагаться на разных уровнях, при этом каждый сервер взаимодействует только с ближайшими уровнями и не связан запросами с другими.
* **Предоставление кода по запросу (Code on Demand).** Серверы могут отправлять клиенту код (например, скрипт для запуска видео). Так общий код приложения или сайта становится сложнее только при необходимости.
* **Начало от нуля (Starting with the Null Style).** Клиент знает только одну точку входа на сервер. Дальнейшие возможности по взаимодействию обеспечиваются сервером.

Несмотря на отсутствие стандартов, при создании REST API есть общепринятые лучшие практики, например:

* использование защищенного протокола HTTPS
* использование инструментов для разработки API Blueprint и Swagger
* применение приложения для тестирования Get Postman
* применение как можно большего количества HTTP-кодов
* архивирование больших блоков данных

В REST API есть 4 метода HTTP, которые используют для действий с объектами на серверах:

* **GET** (получение информации о данных или списка объектов)
* **DELETE** (удаление данных)
* **POST** (добавление или замена данных)
* **PUT** (регулярное обновление данных)

Такие запросы еще называют идентификаторами CRUD:

* create (создать),
* read (прочесть),
* update (обновить),
* delete (удалить).

Это стандартный набор действий для работы с данными.

В каждом HTTP-запросе есть заголовок, за которым следует описание объекта на сервере — это и есть его состояние.

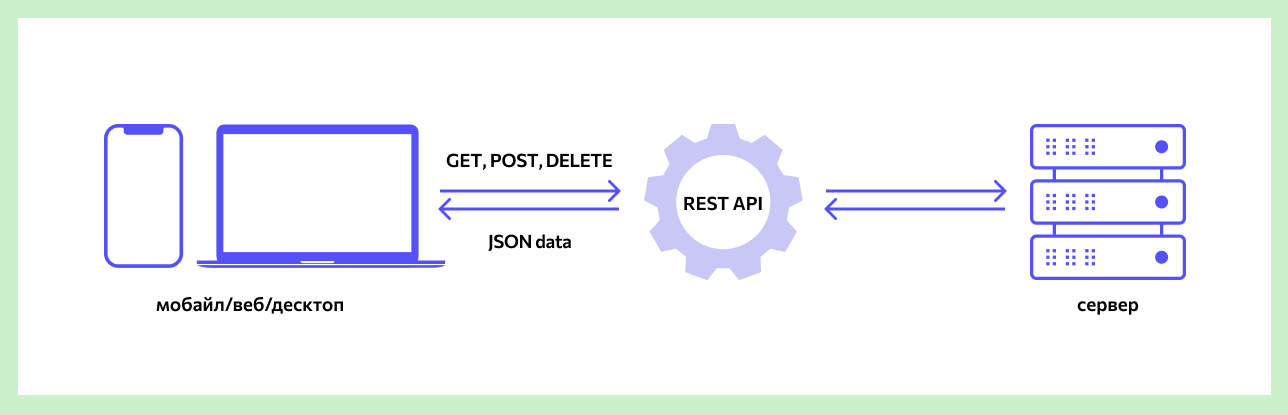


Рис. 2.

REST API рекомендуют использовать в следующих случаях:

* ограниченная пропускная способность соединения с сервером;
* есть необходимость кэшировать запросы;
* приложение или сайт будет значительно масштабироваться;
* приложение или сайт использует AJAX (метод фонового обмена данными с сервером).

REST API используют чаще альтернативных методов, например SOAP. Помимо сайтов и веб-приложений RESTful используют для облачных вычислений.

# ASP.Net Core

#### Основные понятия

ASP.NET Core представляет технологию для создания веб-приложений на платформе .NET, развиваемую компанией Microsoft. В качестве языков программирования для разработки приложений на ASP.NET Core используются C# и F#.

История ASP.NET фактически началась с выходом первой версии .NET в начале 2002 года и с тех пор ASP.NET и .NET развивались параллельно: выход новой версии .NET знаменовал выход новой версии ASP.NET, поскольку ASP.NET работает поверх .NET. В то же время изначально ASP.NET была нацелена на работу исключительно в Windows на веб-сервере IIS (хотя благодаря проекту Mono приложения на ASP.NET можно было запускать и на Linux).

Однако 2014 год ознаменовал большие перемены, фактически революцию в развитии платформы: компания Microsoft взяла курс на развитии ASP.NET как кроссплатформенной технологии, которая развивается как opensource-проект. Данное развитие платформы в дальнейшем получило название ASP.NET Core, собственно, как ее официально именует Microsoft до сих пор. Первый релиз обновленной платформы увидел свет в июне 2016 года. Теперь она стала работать не только на Windows, но и на MacOS и Linux. Она стала более легковесной, модульной, ее стало проще конфигурировать, в общем, она стала больше отвечать требованиям текущего времени.

Текущая версия ASP.NET Core вышла вместе с релизом .NET 8 в ноябре 2023 года.

ASP.NET Core теперь полностью является opensource-фреймворком. Все исходные файлы фреймворка доступны на github в репозитории <https://github.com/dotnet/aspnetcore/>.

#### Архитектура и модели разработки

Текущую архитектуру платформы ASP.NET Core можно выразить следующим образом:

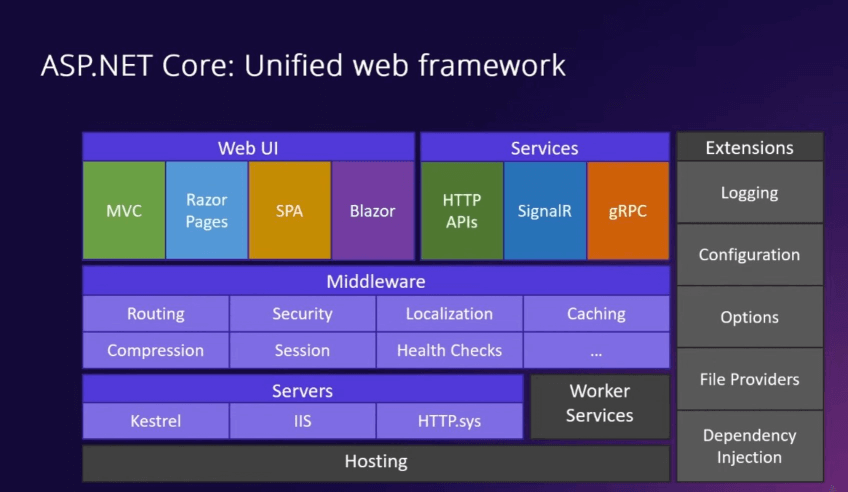


Рис.3.

На самом верхнем уровне располагаются различные модели взаимодействия с пользователем. Это технологии построения пользовательского интерфейса и обработки ввода пользователя, как MVC, Razor Pages, SPA (Single Page Application - одностраничные приложения с использованием Angular, React, Vue) и Balzor. Кроме того, это сервисы в виде встроенных HTTP API, библиотеки SignalR или сервисов GRPC.

Все эти технологии базируются и/или взаимодействуют с чистым ASP.NET Core, который представлен прежде всего различными встроенными компонентами middleware - компонентами, которые применяются для обработки запроса. Кроме того, технологии высшего уровня также взаимодействуют с различными расширениями, которые не являются непосредственной частью ASP.NET Core, как расширения для логгирования, конфигурации и т.д.

И на самом нижнем уровне приложение ASP.NET Core работает в рамках некоторого веб-сервера, например, Kestrel, IIS, библиотеки HTTP.sys.

Это вкратце архитектура платформы, теперь рассмотрим моделей разработки приложения ASP.NET Core:

* Прежде всего это **базовый ASP.NET Core**, который поддерживает все основные моменты, необходимые для работы современного веб-приложения: маршрутизация, конфигурация, логгирования, возможность работы с различными системами баз данных и т.д.. В ASP.NET Core 6 в фреймворк был добавлен так называемый **Minimal API** - минимизированная упрощенная модель, который еще упростила процесс разработки и написания кода приложения. Все остальные модели разработки работают поверх базового функционала ASP.NET Core
* **ASP.NET Core MVC** представляет в общем виде построения приложения вокруг трех основных компонентов - Model (модели), View (представления) и Controller (контроллеры), где модели отвечают за работу с данными, контроллеры представляют логику обработки запросов, а представления определяют визуальную составляющую.

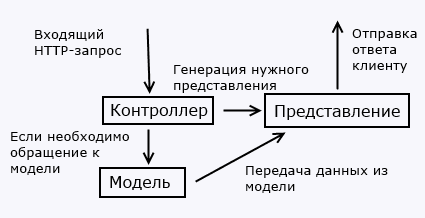


Рис. 4.

* **Razor Pages** представляет модель, при котором за обаботку запроса отвечают специальные сущности - страницы Razor Pages. Каждую отдельную такую сущность можно ассоциировать с отдельной веб-страницей.
* **ASP.NET Core Web API** представляет реализацию паттерна REST, при котором для каждого типа http-запроса (GET, POST, PUT, DELETE) предназначен отдельный ресурс. Подобные ресурсы определяются в виде методов контроллера Web API. Данная модель особенно подходит для одностраничных приложений, но не только.

**Blazor** представляет фреймворк, который позволяет создавать интерактивные приложения как на стороне сервера, так и на стороне клиента и позволяет задействовать на уровне браузера низкоуровневый код WebAssembly.

#### Особенности платформы

1. ASP.NET Core работает поверх платформы .NET и, таким образом, позволяет задействовать весь ее функционал.
2. В качестве языков разработки применяются языки программирования, поддерживаемые платформой .NET. Официально встроенная поддержка для проектов ASP.NET Core есть у языков C# и F#
3. ASP.NET Core представляет кроссплатформенный фреймворк, приложения на котором могут быть развернуты на всех основных популярных операционных системах: Windows, Mac OS, Linux. И таким образом, с помощью ASP.NET Core мы можем как создавать кроссплатформенные приложения на Windows, на Linux и Mac OS, так и запускать на этих ОС.
4. Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget.
5. Поддержка работы с большинством распространенных систем баз данных: MS SQL Server, MySQL, Postgres, MongoDB
6. ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.
7. Богатый инструментарий для разработки приложений. В качестве инструментария разработки мы можем использовать такую среду разработки с богатым функционалом, как **Visual Studio** от компании Microsoft.

# Базы данных

**База данных** — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

Одним из главных последствий широкого использования информационных технологий стало появление и все более активное практическое применение различных баз данных. Их объем и количество растут очень быстро.

Для лучшего понимания, что такое электронная база данных, целесообразно перечислить ее основные характеристики. К числу таковых относятся:

1. **Высокая производительность.** Выражается в получении запрашиваемых данных в течение нескольких секунд, то есть практически мгновенно.
2. **Удобство пополнения и обновления.** Аналогичные требования касаются возможности вносить корректировки в имеющийся массив информации.
3. **Автономная структура.** Любые изменения никак не затрагивают ни основу базы данных, ни другое программное обеспечение на компьютере или сервере.
4. **Соответствие стандарту.** Предусматривает возможность обновления программы СУБД без влияния на сохранность, структурированность и другие характеристики базы данных.
5. **Безопасность.** Обеспечивает конфиденциальность всех сведений, размещенных в БД, в соответствии с установленными правилами доступа.
6. **Интеграция с другими информационными системами.** Эффективность практического использования баз данных требует взаимодействия с другими БД и специализированным программным обеспечением.
7. **Одновременный доступ большому числу пользователей.** Ключевое отличие современной базы данных от обычной электронной таблицы. Выражается в возможности внесения изменений и работы с БД несколькими пользователями одновременно, причем с разных устройств.

Базы данных постоянно совершенствуются. В том числе – за счет появления новых разновидностей. Классификация существующих сегодня БД производится по нескольким параметрам. К числу наиболее часто применяемых относятся такие:

* Формат представления сведений – документальные (набор документов), фактографические (параметр – значение) и мультимедийные (контент сразу в нескольких форматах).
* Тип размещения информации – реляционные (взаимосвязанные таблицы) и нереляционные (сетевые или иерархические структуры).
* Способ хранения – локальные (на одном ПК или другом устройстве) и распределенные (на сервере с многопользовательским доступом или нескольких ПК).
* Назначение – информационные (для предоставления сведений) и операционные (для работы с данными).
* Доступность – открытые (без ограничений доступа) и с ограниченным доступом.

Для работы с Базами Данных используются СУБД — системы управления базами данных (англ. Database Management System, сокр. DBMS) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Самыми широко распространенными являются две СУБД – MySQL и Oracle. Достойную конкуренцию им составляют еще несколько программных продуктов, включая PostgreSQL, MongoDB, Redis и ряд других.

В корпоративном сегменте для стека Microsoft популярной СУБД является MS SQL Server. В работе мы буде использовать её бесплатную версию MS SQL Server Lite.

# Entity Framework

Entity Framework представляет ORM-технологию (object-relational mapping - отображения данных на реальные объекты) от компании Microsoft для доступа к данным. Entity Framework Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Как технология доступа к данным Entity Framework Core работает поверх платформы .NET и поэтому может использоваться на различных платформах стека .NET. Это и стандартные платформы типа Windows Forms, консольные приложения, WPF, UWP и ASP.NET Core. При этом кроссплатформенная природа EF Core позволяет задействовать ее не только на ОС Windows, но и на Linux и Mac OS X.

Поскольку Entity Framework Core работает на основе платформы .NET, то он развивается вместе с данной платформой. Текущая версия EF Core - 8.0 была выпущена в ноябре 2023 года вместе с .NET 8, и технология продолжает развиваться.

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер. По умолчанию на данный момент Microsoft предоставляет ряд встроенных провайдеров: для работы с MS SQL Server, для SQLite, для PostgreSQL. Также имеются провайдеры от сторонних поставщиков, например, для MySQL.

Стоит отметить, что Entity Framework Core развивается как opensourse-проект, все файлы которого можно найти в репозитории на github по адресу <https://github.com/dotnet/efcore>.

Также стоит отметить, что EF Core предоставляет универсальный API для работы с данными. И если, к примеру, мы решим сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данными, получает данные, добавляет их в БД и т.д., останется прежним.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность определяет набор данных, которые связаны с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их коллекциями.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст. Свойства необязательно представляют простые данные типа int или string, но могут также представлять и более комплексные типы данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework Core, как технологии ORM, является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А Entity Framework при выполнении запроса транслирует выражения LINQ в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

# Подходы работы с Базой Данных

У ORM есть несколько подходов для работы с Базой Данных.

Первый – **Code First**. Он подразумевает, что сначала пишется код на C#, а потом по этому коду создается база данных. Для этого подхода очень важно определить классы модели или entity, которая будет храниться в базе данных, описать ее в классах C# в виде модели, и написать класс контекста, который и будет работать с используемой базой данных. Подход Code First используется чаще всего программистами C#.

Второй подход — **Database-First** — подходит для тех, кто хорошо знает SQL, но в этом случае необязательно хорошо знать C#. Первым делом создается база данных, затем генерируется EDMX-модель базы данных. В этом XML в файле .edmx содержится информация о структуре базы, модель данных и маппинг их друг на друга. В Visual Studio есть графический дизайнер, с помощью которого можно работать с .edmx

**Model-First** – это третий подход ORM. Его часто используют архитекторы, так как при этом подходе можно не знать ни SQL, ни синтаксис C#. В этом случае сначала создается графическая модель EDMX, в это время в фоновом режиме создаются классы C# модели, а затем генерируется база данных на основе диаграммы EDMX.

Все таблицы базы данных определяются в Entity Framework в виде классов моделей или сущностей (entity), как правило, по принципу 1 таблица, например users, – 1 класс в NET, например, User. Такие пары называют условностями, и они определены в классе контекста данных как наборы DbSet и такой подход работает по умолчанию.

Хотя существуют такие механизмы, такие как Fluent API и аннотации данных, возможно переопределить эти условности или дополнительные правила конфигурации.

# Техническое решение разрабатываемого приложения

В разрабатываемом приложении будет реализовано 3 сервиса, унаследованные от соответствующих интерфейсов:

1. WebApi - сервис отвечающий за взаимодействие с ITSM системой. С его помощью мы будем получать крайний срок заявки, а также отправлять запрос на ускорение. Это небольшая библиотека, используемая для интеграций по REST с корпоративными системами. В данном проекте она не используется, так как с корпоративной ITSM системой не просто провести интеграцию из внешней сети. Поэтому в проекте работа с ITSM системой эмулируется работой с Базой Данных.

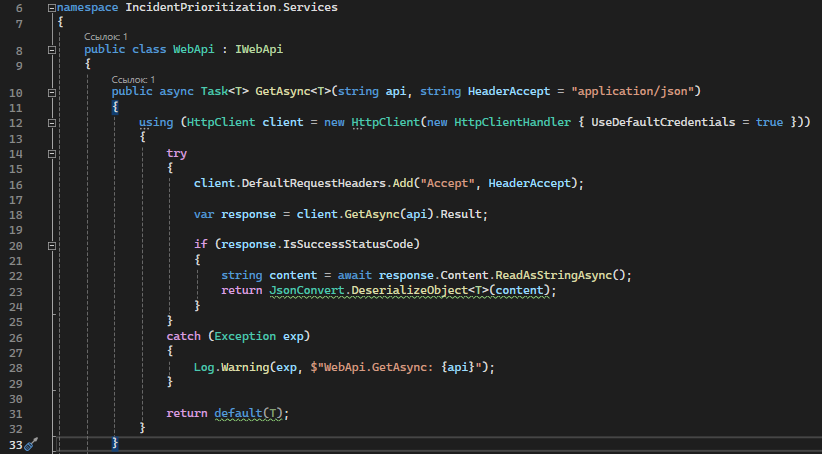


Рис.5. Пример GET запроса

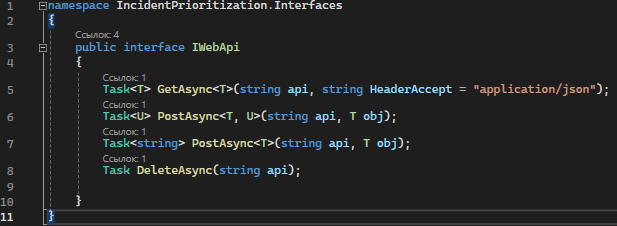


Рис.6.Интерфейс IWebApi

Интерфейс предполагает реализацию 4 методов. Один из POST методов использует JSON сериализацию.

1. DataService – сервис работающий с Базами Данных. В одной у нас содержится список привилегированных пользователей, которым доступна приоритизация инцидентов.

Другая база эмулирует БД ITSM системы.

Сервис проверяет наличие инициатора в списке и крайний срок по заявке.

Пишет в БД ITSM системы признак ускоренного инцидента и комментарий.

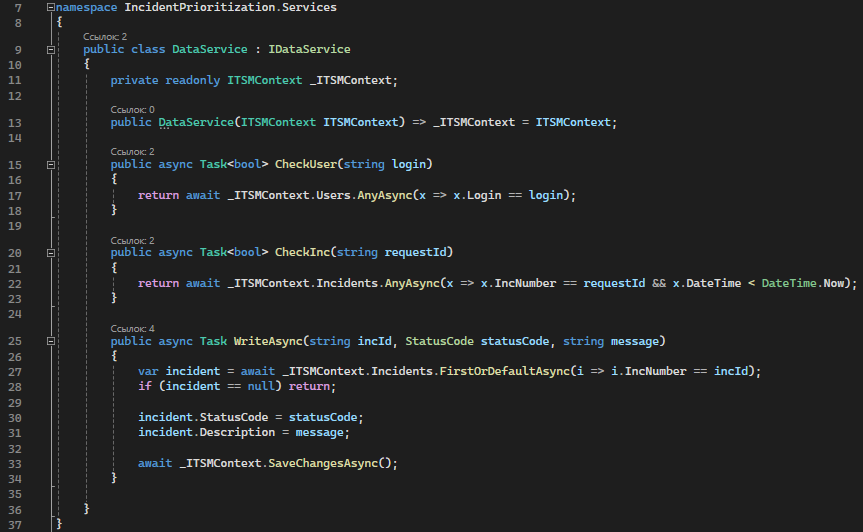


Рис.7.Сервис DataService.

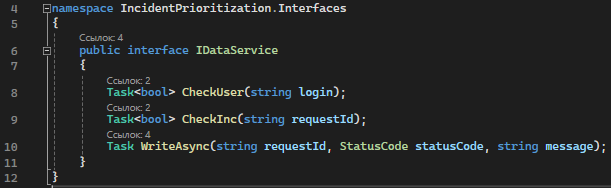


Рис. 8. Интерфейс IDataService

1. ITSMService – реализует логику проверок запроса на ускорение. И отдает ответы инициатору. В этом сервисе мы реализуем метод ITSMRequest интерфейса IITSMService.

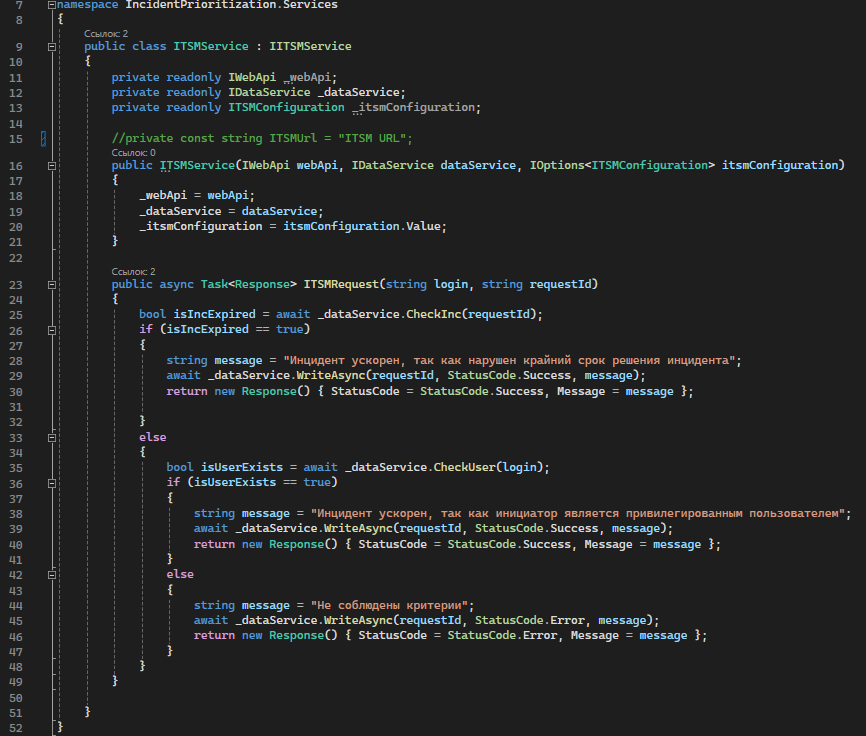


Рис.9. ITSMService

Как было указано ранее, в приложении предусмотрена интеграция с боевой ITSM системой, поэтому в этом сервисе есть вызов экземпляра класса ITSMConfiguration в который необходимо передать URL эндпоинта для подключения по REST. Сам URL указывается в appsettings.json.

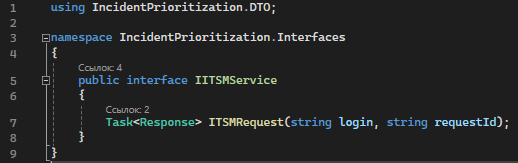


Рис.10. Интерфейс IITSMService

# Создание сущностей

В приложении мы будем работать с двумя сущностями.

Incidents – имеющая свойства:

* Id – PKey в БД,
* IncNumber – номер инцидента,
* DateTime – крайний срок исполнения,
* StatusCode – статус, показывающий ускорен инцидент или нет,
* Description – комментарий.

StatusCode реализован через энумератор, который был положен в отдельную папку Enums для организации красивой структуры файлов проекта.

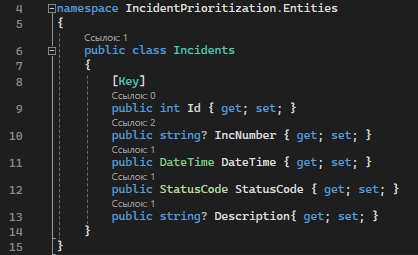


Рис. 11. Сущность Incidents

И сущность Users со свойствами:

Id – PKey,

Login – уникальный логин пользователя.

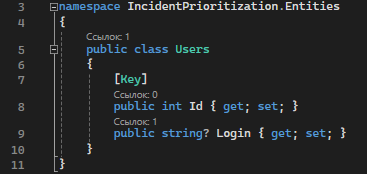


Рис. 12. Сущность Users

Теперь, имея сущности можно приступить к созданию Базы Данных.

# Создание базы данных

В проекте решено было использовать БД MS SQL Lite, так как в нашей компании есть необходимый сервер и в последствии будет проще внедрить приложение на работе.

Для работы с базой данных мы используем Entity Framework Core.

Чтобы создать миграцию, был создан класс ITSMContext.cs и используя зависимость от DBContext описаны наши будущие таблицы:

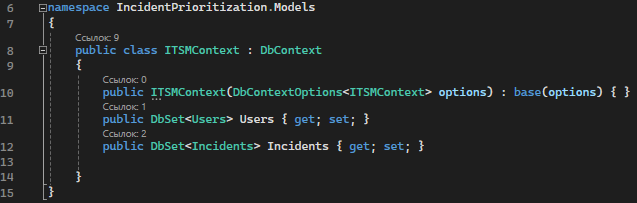


Рис.13. Создание контекста

После создания контекста, используя команду **dotnet ef migrations** создаем миграцию, а командой **dotnet ef database update** записываем её в Базу Данных. За время разработки приложения было сделано три миграции, так как не сразу было понимание об итоговом виде базы.

# Создание Data Transfer Objects

Перед созданием контроллеров необходимо реализовать DTO.

**Data Transfer Object (DTO)** — один из шаблонов проектирования, используется для передачи данных между подсистемами приложения.

Мы создадим объект Request в котором будет описан запрос пользователя, состоящий из логина инициатора и номером инцидента.

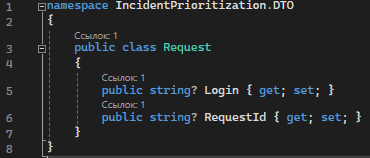


Рис. 14. DTO Request

И объект Response, в котором будем направлять пользователю результат обработки его запроса с указанием статуса и комментария.

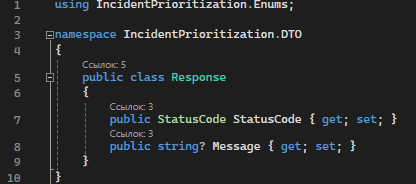


Рис.15. DTO Response

Теперь можно приступать к написанию контроллера.

# Создание контроллера

Разрабатываем контроллер для взаимодействия с пользователем. Подключаем зависимость от ControllerBase, и обращаемся к экземпляру нашего класса.

Создаем POST метод принимающий объект Request c логином и номером заявки и возвращающий результат отработки метода ITSMRequest.

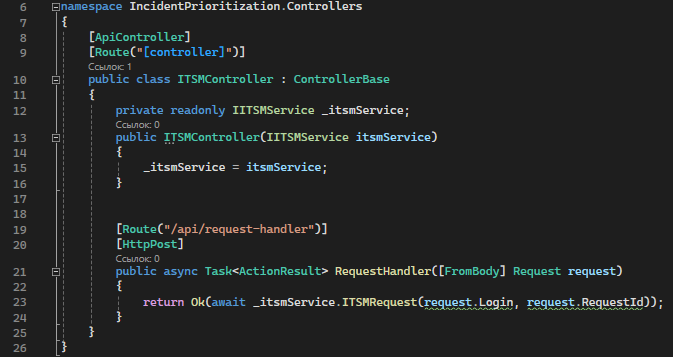


Рис. 16. ITSMController

# Сборка проекта

На текущем этапе мы создали все необходимые классы и теперь необходимо подготовить файл Program.cs.

Добавляем DBContext:



Рис.17.

и прописываем ConnectionString в appsettings:



Рис.18.

Используя CreateBuilder добавляем все наши сервисы:

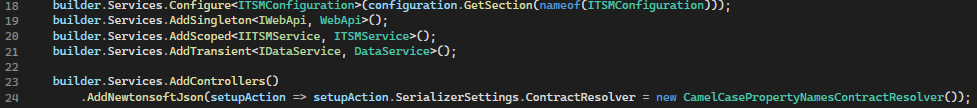


Рис.19.

Итоговый вид файла Programs.cs:

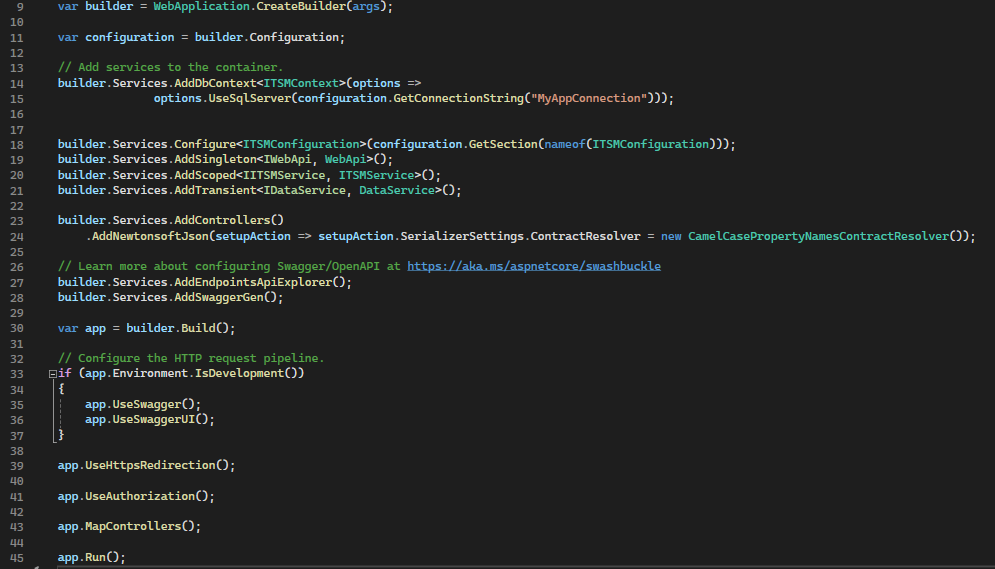


Рис.20.

# Демонстрация работы приложения

Для начала заполняем таблицу Incidents тестовыми данными.

Для проверки всех возможных кейсов мы используем 3 записи, из которых 1 с истекшим крайним сроком решения, а 2 нет:

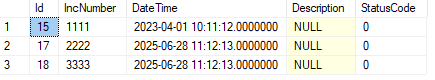


Рис.21.

Также заполним таблицу Users логинами привилегированных пользователей:

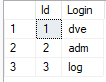


Рис.22.

Теперь запускаем наше приложение и через Swagger отправляем запрос с логином petr, которого нет в привилегированном списке и инцидентом номер 0000, которого у нас нет в БД инцидентов. Получаем ответ с кодом 200.

Видим StatusCode 1, что означает Error и message «Не соблюдены критерии».

В Базу Данных ничего не пишется, так как в базе нет заявки с номером 0000.



Рис.23.

Отправляем от этого же пользователя запрос с номером инцидента 1111 у которого истек крайний срок и получаем успешный код и соответствующий комментарий:

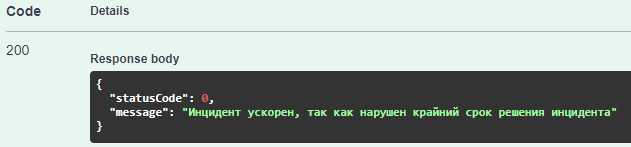


Рис.24.

Делаем запрос с номером 2222 и снова видим отказ в ускорении, так как крайний срок не истёк, а пользователя нет в привилегированном списке.

Меняем логин на adm, который есть в списке и делаем запрос с номером инцидента 3333 у которого не истек крайний срок и видим сообщение об успешном ускорении запроса:

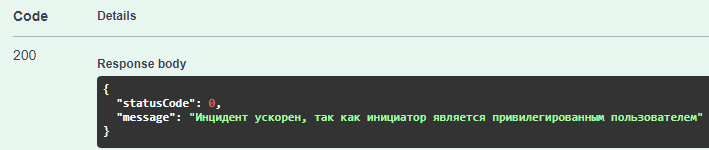


Рис.25.

Все результаты выполнения операций записаны в Базу Данных:

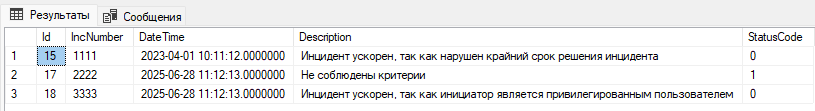


Рис.26.

# Заключение

Код проекта опубликован в открытом доступе на github: <https://github.com/dvegor/Diploma>

Исследуя предметную область, была обоснована потребность внутрикорпоративных пользователей в приоритизации инцидентов с истекшим крайним сроком или по ключевым Бизнес вертикалям.

В ходе работы над проектом были изучены основы разработки web-приложений, такие фреймворки как ASP.Net и Entity Framework.

Было разработано web-приложение и WebApi для подключения к действующей ITSM системе.

Используя EntityFramework мы реализовали подход CodeFirst и создали Базу данных с таблицей привилегированных пользователей и таблицей эмулирующей работу ITSM системы. И продемонстрировали работу приложения.

В программе следует доработать ответы системы, в случаях несоответствия критериев, чтобы пользователи имели достаточно информации для понимая происходящего.

Также стоит подключить логирование к корпоративной платформе наблюдаемости, проверить код на уязвимости используя автоматизированные системы проверки кода.

При необходимости изменения и расширения перечня критериев влияющих на принятие решения об ускорении, это можно легко делать добавляя новые методы в существующие сервисы.

В рамках данного проекта все поставленные цели исследования успешно достигнуты.

В дальнейшем данное приложение будет готовится к интеграции в корпоративную среду нашей компании.

# Список используемой литературы и интернет-ресурсов

* 1. Интернет-сайт: <https://www.atlassian.com>
  2. Интернет-сайт: <https://blog.skillfactory.ru>
  3. Интернет-сайт: <https://habr.com/>
  4. Интернет-сайт: <https://metanit.com>
  5. Интернет-сайт: <https://skillbox.ru>
  6. Интернет-сайт: <https://ru.wikipedia.org>
  7. Telegram-бот: <https://t.me/mts_vicuna_bot>