**Московский международный колледж цифровых технологий «ТОР»**

# ПРОЕКТ

## Система управления информационным стендом аэропорта

**Работу выполнили:**

Потехин Сергей Эмильевич

Бондаренко Сергей Алексеевич

Затонский Егор Олегович

**Группа РПО 9/2**

**Руководитель:**

Рослова Оксана Александровна

Октябрь 2025 Санкт-Петербург

Россия

# ОГЛАВЛЕНИЕ

## Цель проекта

Цель проекта — разработать базу данных и структуру для информационной системы аэропорта, обслуживающей отображение данных на информационном стенде (табло). Система хранит расписание рейсов (вылеты/прилёты), информацию об авиакомпаниях и воздушных судах, привязку к воротам/терминалам, текущие статусы рейсов (запланирован, регистрацию завершена, посадка, задержан, отправлен и т.д.) и историю изменений статусов. Система должна обеспечивать целостность данных, быструю выдачу информации для табло и возможность администрирования расписания и статусов.

## Задачи проекта

Отображение списка вылетов/прилётов (по времени, по статусу).

Хранение статусов рейса и истории изменений (для логов и аналитики).

Редактирование/добавление рейсов сотрудником (админ).

Привязка к самолёту, авиакомпании, гейту/терминалу.

Поддержка реального обновления Estimated/Actual times.

API / View, удобный для фронтенда табло.

## 1. Обзор проекта

Airport Management System — это консольное приложение, разработанное для управления информацией об аэропортах, самолетах, воротах и их статусах. Система предоставляет полный функционал CRUD (Create, Read, Update, Delete) для работы с базой данных SQL Server через микро-ORM Dapper.

**Вот 4 консольные команды для установки Dapper через NuGet:**

1. Через .NET CLI (рекомендуется для .NET Core/5+)

```
dotnet add package Dapper
```

2. Через .NET CLI с указанием конкретной версии

```
dotnet add package Dapper --version 2.1.35
```

3. Через Package Manager Console (Visual Studio)

```
Install-Package Dapper
```

4. Через NuGet CLI

```
nuget install Dapper
```

### 1.1. Ключевые технологии

| Категория | Технология |
|---|---|
| Backend | C# .NET 9.0 |
| ORM | Dapper 2.1.66 (микро-ORM) |
| Провайдер БД | Microsoft.Data. SqlClient 6.1.2 |
| База данных | SQL Server Express, T-SQL |
| Тестирование | xUnit 2.9.2, Moq 4.20.70 |
| Паттерны | Repository Pattern, Dependency Injection, Async/Await |

### 1.2. Структура проекта

Проект следует архитектуре, разделяющей ответственность на слои:

• Models/: Модели данных (POCO), соответствующие таблицам БД.

• Services/: Сервисы, инкапсулирующие логику работы с внешними ресурсами (DatabaseService).

• Repositories/: Репозитории, реализующие логику доступа к данным (Repository Pattern).

• Program.cs: Точка входа и логика консольного интерфейса.

## 2. Детальный анализ файлов и логики работы

### 2.1. Models (Модели данных)

Эти классы (POCO) используются для маппинга данных из SQL Server с помощью Dapper.

| Файл | Описание логики |
|---|---|
| Airplane.cs | Модель самолета. Содержит поля для всех атрибутов самолета и навигационные свойства Gate и Status. |
| Airport.cs | Модель аэропорта. Содержит основные данные об аэропорте и список Gates. |
| Gate.cs | Модель ворот. Содержит информацию о воротах и связанном Airport. |
| Status.cs | Модель статуса самолета. Содержит название и описание статуса. |

### 2.2. Services (Сервисы)

**DatabaseService.cs**

Этот класс является ядром подключения к SQL и инкапсулирует всю логику работы с микро-ORM Dapper. Он предоставляет универсальные асинхронные методы для выполнения SQL-запросов.

| Метод | Назначение |
|---|---|
| QueryAsync<T> | Выполнение SELECT-запросов, возвращающих список объектов. |
| QueryFirstOrDefaultAsync<T> | Выполнение SELECT-запросов, возвращающих один объект. |
| ExecuteAsync | Выполнение команд INSERT, UPDATE, DELETE. |
| ExecuteScalarAsync<T> | Выполнение команды и возврат скалярного значения (например, ID новой записи). |
| QueryAsync<T1, T2, ..., TReturn> | Multi-Mapping: для запросов с JOIN, маппинг нескольких сущностей за один проход. |

### 2.3. Repositories (Репозитории)

Реализуют Repository Pattern, инкапсулируя логику доступа к данным для каждой сущности. Зависят от DatabaseService.

| Файл | Логика работы |
|---|---|
| AirplaneRepository.cs | CRUD-операции для самолетов. Использует Multi-Mapping для получения связанных Gate, Airport и Status. |
| AirportRepository.cs | Получение списка активных аэропортов. |
| GateRepository.cs | Получение ворот, включая связанную информацию об аэропорте (Multi-Mapping). |
| StatusRepository.cs | Получение списка активных статусов. |

### 2.4. Program.cs (Точка входа)

Файл содержит логику инициализации, Dependency Injection (ручное создание сервисов и репозиториев), проверку подключения к БД и основной цикл консольного меню. Вся логика UI/UX (отображение таблиц, цветовое кодирование) реализована в этом файле.

## 3. База данных

Проект использует реляционную базу данных на основе MS SQL Server для хранения информации об аэропортах, воротах, статусах и самолетах. Схема базы данных разработана с учетом принципов нормализации и состоит из четырех основных таблиц.

### 3.1. Описание таблиц

**Таблица 1. Airport (Аэропорты)**

Таблица предназначена для хранения информации об аэропортах, к которым могут быть привязаны ворота.

| Поле | Тип данных | Ключ | Описание | Ограничение |
|---|---|---|---|---|
| **AirportID** | INT | PK | Уникальный идентификатор аэропорта. | NOT NULL |
| **AirportCode** | NVARCHAR(10) | | Код аэропорта (например, IATA). | NOT NULL |
| **AirportName** | NVARCHAR(200) | | Полное название аэропорта. | NOT NULL |
| **City** | NVARCHAR(100) | | Город, в котором расположен аэропорт. | NOT NULL |
| **Country** | NVARCHAR(100) | | Страна расположения. | NOT NULL |
| **CreatedDate** | DATETIME | | Дата создания записи. | NOT NULL |
| **IsActive** | BIT | | Флаг активности записи. | NOT NULL |

**Таблица 2. Status (Статусы)**

Таблица-справочник, содержащая предопределенные статусы, которые могут быть присвоены самолетам.

| Поле | Тип данных | Ключ | Описание |
|---|---|---|---|
| **StatusID** | INT | PK | Уникальный идентификатор статуса. |
| **StatusName** | NVARCHAR(50) | | Название статуса (например, 'In Flight', 'Repair'). |
| **StatusDescription** | NVARCHAR(200) | | Подробное описание статуса. |
| **IsActive** | BIT | | Флаг активности статуса. |
| **CreatedDate** | DATETIME | | Дата создания записи. |

**Таблица 3. Gate (Ворота)**

Таблица для хранения информации о конкретных воротах (гейтах) в аэропорту.

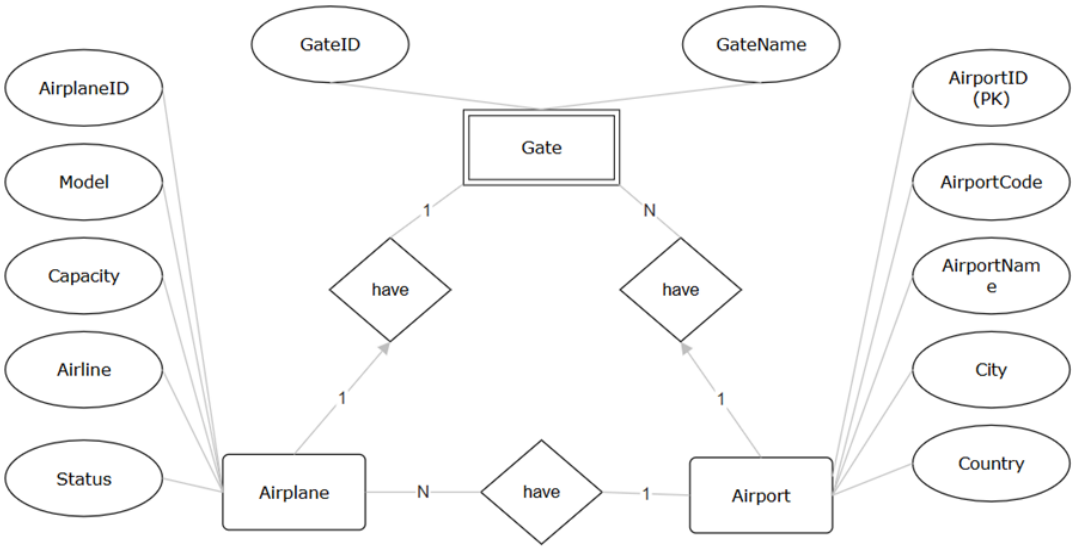
| Поле | Тип данных | Ключ | Описание | Ограничение |
|---|---|---|---|---|
| **GateID** | INT | PK | Уникальный идентификатор ворот. | NOT NULL |
| **GateName** | NVARCHAR(50) | | Название ворот (например, 'A1', 'B23'). | NOT NULL |
| **AirportID** | INT | FK | Внешний ключ к таблице Airport. | NOT NULL |
| **GateType** | NVARCHAR(50) | | Тип ворот (например, 'Domestic', 'International'). | NULL |
| **Capacity** | INT | | Максимальная вместимость (опционально). | NULL |
| **IsOperational** | BIT | | Флаг работоспособности ворот. | NOT NULL |
| **CreatedDate** | DATETIME | | Дата создания записи. | NOT NULL |

**Таблица 4. Airplane (Самолеты)**

Основная таблица, содержащая детальную информацию о каждом самолете.

| Поле | Тип данных | Ключ | Описание |
|---|---|---|---|
| **AirplaneID** | INT | PK | Уникальный идентификатор самолета. |
| **Model** | NVARCHAR(100) | | Модель самолета (например, 'Boeing 737'). |
| **Capacity** | INT | | Пассажирская вместимость. |
| **Airline** | NVARCHAR(100) | | Название авиакомпании. |
| **RegistrationNumber** | NVARCHAR(50) | | Регистрационный номер самолета. |
| **ManufactureDate** | DATE | | Дата производства. |
| **LastMaintenanceDate** | DATE | | Дата последнего обслуживания. |
| **NextMaintenanceDate** | DATE | | Дата следующего обслуживания. |
| **StatusID** | INT | FK | Внешний ключ к таблице Status. |
| **GateID** | INT | FK | Внешний ключ к таблице Gate. |
| **CreatedDate** | DATETIME | | Дата создания записи. |

### 3.2. ER-диаграмма




### 3.3. Нормализация до 3НФ

База данных проекта "Airport Management System" спроектирована с соблюдением требований Третьей Нормальной Формы (3НФ).

1НФ: Все таблицы содержат атомарные значения.

2НФ: Все неключевые атрибуты зависят от полного ключа.

3НФ: Устранены транзитивные зависимости.

## 4. Интерфейс программы

Вставьте сюда скриншоты консольного интерфейса и описание каждого экрана.

?? СПИСОК АЭРОПОРТОВ

| ID | Код | Название | Город | Страна |
|---|---|---|---|---|
| 4 | CDG | Шарль де Голль | Париж | Франция |
| 3 | DME | Домодедово | Москва | Россия |
| 2 | LED | Пулково | Санкт-Петербург | Россия |
| 5 | LHR | Хитроу | Лондон | Великобритания |
| 1 | SVO | Шереметьево | Москва | Россия |

?? Всего аэропортов: 5

?? СТАТУСЫ САМОЛЕТОВ

| ID | Название | Описание |
|---|---|---|
| 1 | Active | Самолет активен и готов к полетам |
| 2 | inactive | Самолет неактивен и не используется |
| 3 | repair | Самолет находится на ремонте |
| 4 | in flight | Самолет находится в полете |
| 5 | Boarding | Происходит посадка пассажиров |

?? Всего статусов: 5

?? СПИСОК САМОЛЕТОВ

| ID | Модель | Авиакомпания | Рег. номер | Вмест | Статус | Ворота | Аэропорт |
|---|---|---|---|---|---|---|---|
| 1 | dfg | Аэрофлот | RA-12345 | 189 | Неизвестно | C1 | SVO |
| 2 | Airbus A320 | Аэрофлот | RA-67890 | 180 | Boarding | B2 | SVO |
| 3 | Boeing 777-300ER | Аэрофлот | RA-11111 | 402 | repair | A2 | LED |
| 4 | Sukhoi Superj... | Аэрофлот | RA-22222 | 98 | in flight | B1 | SVO |
| 5 | Boeing 737-800 | S7 Airlines | RA-33333 | 189 | repair | A1 | LED |
| 6 | Airbus A321 | Аэрофлот | RA-44444 | 220 | Active | B2 | SVO |
| 7 | Boeing 747-8 | Аэрофлот | RA-55555 | 467 | inactive | A1 | SVO |
| 14 | superjet 100 | Yamal | 89 | 100 | Active | A1 | SVO |
| 15 | zus | airline | 123-784 | 123 | Active | Не назнач. | - |

?? Всего самолетов: 9

## 5. Юнит-тестирование

Проект включает 6 юнит-тестов для AirplaneRepository с использованием xUnit и Moq.

• xUnit: Фреймворк для запуска тестов.

• Moq: используется для создания имитаций (Mock) DatabaseService, что позволяет тестировать репозиторий в изоляции, без реального подключения к базе данных.

Тесты покрывают позитивные сценарии (успешное создание/обновление) и негативные сценарии (обработка некорректных данных и несуществующих ID).