Практическая работа: Система управления персоналом и рабочими сменами

Технологический стек: Avalonia (MVVM), .NET Web API, Pomelo.EntityFrameworkCore.MySql, JWT

Программа предназначена для управления персоналом и рабочими сменами в небольшой компании. Она позволяет авторизованным пользователям (администраторам, менеджерам) добавлять, редактировать и удалять сотрудников, а также назначать, изменять и отслеживать их рабочие смены через графический интерфейс, взаимодействуя с защищённым веб-API и базой данных MySQL.

📌 Общая схема работы системы

```
[Пользователь]
    → (Avalonia UI) → HttpClient → [Web API]
        → (JWT-валидация) → [EF Core]
            → [MySQL: Employee, Credential, Shift]
```

1. Запуск Avalonia-приложения → окно авторизации
2. Успешная авторизация → сохранение JWT в отдельном классе
3. Переход в главное окно → все запросы к API содержат заголовок `Authorization: Bearer {token}`
4. При 401-ошибке → автоматический возврат к окну авторизации
5. Данные между компонентами передаются через DTO (без прямого использования EF-моделей)

🗄️ Структура базы данных (MySQL)

Таблица `Employee`

Поле	Тип	Описание
Id	INT (PK)	Уникальный идентификатор
FirstName	VARCHAR(50)	Имя
LastName	VARCHAR(50)	Фамилия
Position	VARCHAR(100)	Должность
HireDate	DATETIME	Дата приёма на работу
IsActive	BOOLEAN	Активен ли сотрудник

Таблица `Credential`

Поле	Тип	Описание
Id	INT (PK)	Уникальный идентификатор
EmployeeId	INT (FK)	Связь с Employee.Id (каскадное удаление)
Username	VARCHAR(50)	Уникальное имя для входа (индекс)
PasswordHash	VARCHAR(255)	Хэш пароля (bcrypt)
Role	VARCHAR(20)	Роль: Admin, Manager, Employee

Таблица `Shift`

Поле	Тип	Описание
Id	INT (PK)	Уникальный идентификатор
EmployeeId	INT (FK)	Связь с Employee.Id

Поле	Тип	Описание
StartDateTime	DATETIME	Начало смены
EndDateTime	DATETIME	Окончание смены
Description	VARCHAR(255)	Описание смены (опционально)

> ⚠️ ***Важно для студентов***:
>
> - *Настроить каскадное удаление: при удалении Employee удаляются связанные Credential и Shift*
> - *Добавить индексы:* `Credential.Username`, `Shift.EmployeeId`, `Shift.StartDateTime`
> - *Хэшировать пароли через* `BCrypt.Net-Next` *(пример внизу документа)*

🌐 Web API: Полный список контроллеров и методов

🔒 `AuthController` (без атрибута `[Authorize]`)

Метод	Маршрут	Описание
POST	`/api/auth/login`	Принимает `{ username, password }`. Проверяет учётные данные, возвращает `{ token: "jwt...", expiresIn: 3600 }`. При ошибке — 401.
GET	`/api/auth/profile`	Возвращает данные текущего пользователя: `{ employee: {id, firstName, lastName, position}, role }`.

👥 `EmployeesController` (`[Authorize]`)

Метод	Маршрут	Описание
GET	`/api/employees`	Список всех сотрудников (без паролей, хэшей). Возвращает массив `EmployeeDto`.
GET	`/api/employees/{id}`	Данные сотрудника по ID. 404 если не найден.
POST	`/api/employees`	Создаёт сотрудника **и** учётные данные. Принимает `CreateEmployeeDto` (включая `username`, `password`, `role`). Проверяет уникальность username. Возвращает 201 + созданный объект.
PUT	`/api/employees/{id}`	Обновляет данные сотрудника (без учётных данных). Принимает `UpdateEmployeeDto`.
DELETE	`/api/employees/{id}`	Удаляет сотрудника (каскадно удаляются учётные данные и смены). Возвращает 204.

🕒 `ShiftsController` (`[Authorize]`)

Метод	Маршрут	Описание
GET	`/api/shifts`	Все смены. Возвращает `ShiftDto` с вложенным `employee: {id, firstName, lastName}`.
GET	`/api/shifts/{id}`	Смена по ID с данными сотрудника.

Метод	Маршрут	Описание
GET	`/api/shifts/employee/{id}`	Смены конкретного сотрудника (за последние 30 дней).
POST	`/api/shifts`	Создаёт смену. Валидация: `EndDateTime > StartDateTime`, проверка существования `EmployeeId`.
PUT	`/api/shifts/{id}`	Обновляет смену.
DELETE	`/api/shifts/{id}`	Удаляет смену.

🫂 Дополнительные требования к API:

- JWT настройка: секрет из `appsettings.json`, lifetime = 1 час, issuer/audience указаны . Пример внизу документа. !ВАЖНО - В реальных проектах jwt-секрет не хранится в appsettings, и не заливается в git. Вместо этого для хранения используются менеджеры операционной системы и запросы к ним.
- Глобальный обработчик исключений (возвращает JSON с `error` вместо стека)
- Валидация моделей через `[ApiController]` + FluentValidation (опционально, но рекомендуется)
- Swagger/OpenAPI для тестирования (пакет Swashbuckle.AspNetCore)

💻 Avalonia UI (MVVM)

📂 Структура проекта

```
Views/
  LoginWindow.axaml
  MainWindow.axaml
  EmployeesView.axaml
  ShiftsView.axaml
ViewModels/
  LoginViewModel.cs
  MainViewModel.cs
  EmployeesViewModel.cs
  ShiftsViewModel.cs
  EmployeeEditorViewModel.cs (диалог)
  ShiftEditorViewModel.cs (диалог)
Models/
  DTOs: LoginRequest, TokenResponse, EmployeeDto, ShiftDto...
Services/
  ApiService.cs (HttpClient + обработка токена)
```

🖼️ Окна и функционал

1. **LoginWindow**
   - Поля: Username, Password (PasswordBox)
   - Кнопка «Войти» → вызов `ApiService.LoginAsync()`
   - Обработка ошибок: вывод сообщения при 401/сетевых ошибках
   - Чекбокс «Запомнить сессию» (сохранение токена в `SecureStorage`)
2. **MainWindow**
   - TabControl: «Сотрудники», «Смены»
   - Статус-бар: отображение текущего пользователя и роли
   - Кнопка «Выйти» → очистка токена → переход к LoginWindow
3. **EmployeesView**
   - DataGrid со списком сотрудников (колонки: ФИО, Должность, Статус)
   - Кнопки: «Добавить», «Редактировать», «Удалить»
   - При удалении — подтверждение через диалог

- Поиск по ФИО (реактивный фильтр)

4. **ShiftsView**
   - DataGrid: Дата, Время начала/окончания, Сотрудник, Описание
   - ComboBox для фильтрации по сотруднику
   - Календарь (опционально, через Avalonia.Controls.Calendar)
   - Кнопки управления сменами
5. **Диалоговые окна**
   - EmployeeEditor: поля для ФИО, должности, даты приёма, статуса + (только при создании) логин/пароль/роль
   - ShiftEditor: выбор сотрудника (ComboBox), даты/время через DatePicker/TimePicker, описание

⚙️ Ключевые сервисы

- **ApiService**:
  - Автоматическая подстановка `Authorization` header для защищённых запросов
  - Обработка 401 → событие `OnUnauthorized`, подписка в `MainViewModel`
  - Методы: `GetEmployeesAsync()`, `CreateShiftAsync(ShiftDto)`, и т.д.
- **AuthService**:
  - `SaveTokenAsync(string)`, `GetTokenAsync()`, `ClearTokenAsync()`
  - Проверка валидности токена при старте приложения

🔄 Последовательность действий для студента

1. **База данных**

- Создать БД в MySQL (название — по выбору)
- Написать SQL-скрипт создания таблиц с индексами и FK
- Проверить вручную вставку тестовых данных

2. **Web API**
   - Настроить проект: зависимости (Pomelo, JWT), `appsettings.json`
   - Реализовать модели EF Core, DbContext
   - Написать контроллеры с полной валидацией и обработкой ошибок
   - Протестировать в Postman/Swagger:
     - Регистрация сотрудника → вход → получение списка → создание смены

3. **Avalonia-приложение**
   - Настроить MVVM (CommunityToolkit.Mvvm рекомендуется)
   - Реализовать сервисы работы с API и аутентификацией
   - Создать окна и ViewModel с привязками
   - Добавить обработку ошибок и пользовательские уведомления
   - Протестировать полный сценарий: запуск → вход → управление данными

🔐 Пример: чтение секрета JWT

Добавьте значение в `appsettings.json`

```
{
  "Jwt": {
    "Secret": "your-very-long-and-secure-secret-key-here",
    "Issuer": "MyCompany",
    "Audience": "MyApp",
    "ExpireMinutes": 60
  }
}
```

Создайте класс:

```
public class JwtOptions
{
    public const string SectionName = "Jwt";
    public string Secret { get; set; } = string.Empty;
    public string Issuer { get; set; } = string.Empty;
    public string Audience { get; set; } = string.Empty;
    public int ExpireMinutes { get; set; } = 60;
}
```

Зарегистрируйте в `Program.cs`:

```
builder.Services.Configure<JwtOptions>(
    builder.Configuration.GetSection(JwtOptions.SectionName)
);
```

Инжектируйте в контроллер или сервис:

```
public class AuthController : ControllerBase
{
    private readonly JwtOptions _jwtOptions;

    public AuthController(IOptions<JwtOptions> jwtOptions)
    {
        _jwtOptions = jwtOptions.Value;
    }

    // Теперь используйте _jwtOptions.Secret
}
```

🧪 Хэширование пароля

```
using BCrypt.Net;

// Исходный пароль от пользователя
string plainPassword = "mySecret123";

// Генерация хэша (автоматически добавляет соль)
string passwordHash = BCrypt.HashPassword(plainPassword);
```

```
// Пример результата: "$2a$12$XyZ... (60 символов)"
```

🔍 Проверка пароля

```
 string plainPassword = "userInputPassword";
string storedHash = /* полученный из БД */;

bool isPasswordValid = BCrypt.Verify(plainPassword, storedHas

if (isPasswordValid)
{
    // Пароль верный — разрешить вход
}
else
{
    // Неверный пароль
}
```