**Development Plan**

**Prioritized Task List (Using MoSCoW Method)**

**Must Have:**

1. Database with Two Tables:
   * Employee Table: Fields for Employee ID, Full Name, Position, and Email.
   * Issue Table: Fields for Issue ID, Title, Description, Status, Priority, Assigned Employee, and Due Date.
2. CRUD Operations: Fully functional Create, Read, Update, and Delete for both tables via the API.
3. Foreign Key Relationship: AssignedTo in the Issue table linked to EmployeeID in the Employee table.
4. API with Entity Framework: Includes migrations and controllers for database interactions.
5. Angular SPA: Fully functional Single Page Application enabling CRUD operations for both tables.

**Should Have:**

1. Status Management: Preset issue statuses (e.g., Open, In Progress, Closed).
2. Priority Levels: Option to assign priority (e.g., High, Medium, Low).
3. Responsive UI: User-friendly, responsive design for the SPA.

**Could Have:**

1. Search/Filter Features: Ability to search and filter issues based on status, priority, or assigned employee.
2. Activity Log: Track changes made to issues (e.g., status updates).

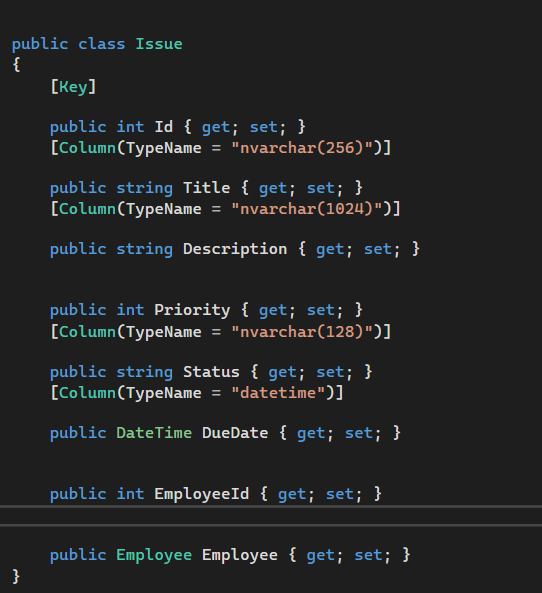
**Won’t Have:**

1. Advanced Analytics: Data visualization or reporting tools are out of scope.
2. Multi-user Role Management: Features like admin or user role access will not be included in this phase.

**GANTT Chart**

|  |  |
| --- | --- |
| **Week** | **Task** |
| 1 | Research, finalize requirements, and set up the development environment. |
| 2 | Design and implement the database schema. Create API using ASP.Net Core with Entity Framework. |
| 3 | Develop CRUD controllers for Employee and Issue tables. |
| 4 | Design Angular SPA and integrate it with the API for basic CRUD operations. |
| 5 | Add status and priority features, test the application, and refine the UI. |
| 6 | Create documentation, record demo video, and prepare for submission. |

**Database Schema**



*Issue Table*

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

*Employee Table*

**System Architecture**

**Design Patterns Applied**

**Repository Pattern**

**Purpose:** The repository pattern is used to abstract data access logic, making the application more testable and reducing coupling between the data layer and the business logic.

**Implementation:**

* A Repository interface is created for handling database operations.
* Specific repositories (e.g., EmployeeRepository, IssueRepository) implement this interface.
* The services in the business layer interact with repositories instead of directly querying the database.

Before Applying Pattern:

Employee Controllers:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*GET(LIST)*

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

*GET*

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

*PUT*

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

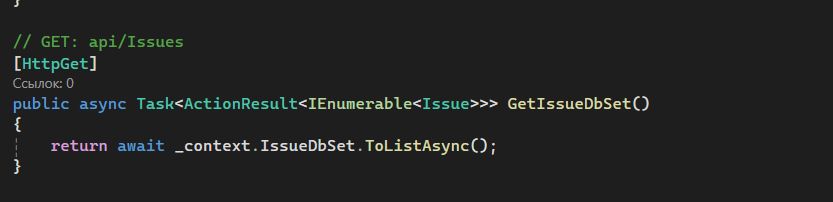
*POST*

Изображение выглядит как текст, снимок экрана, программное обеспечение

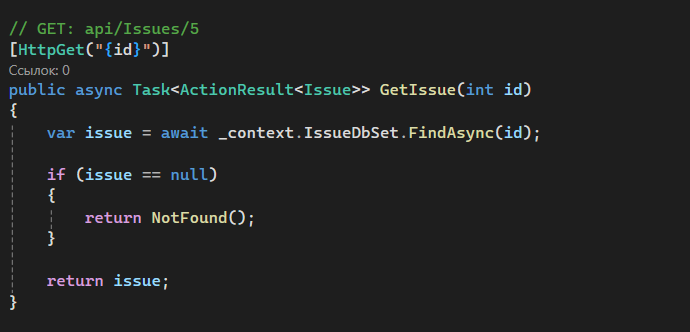
Автоматически созданное описание

*DELETE*

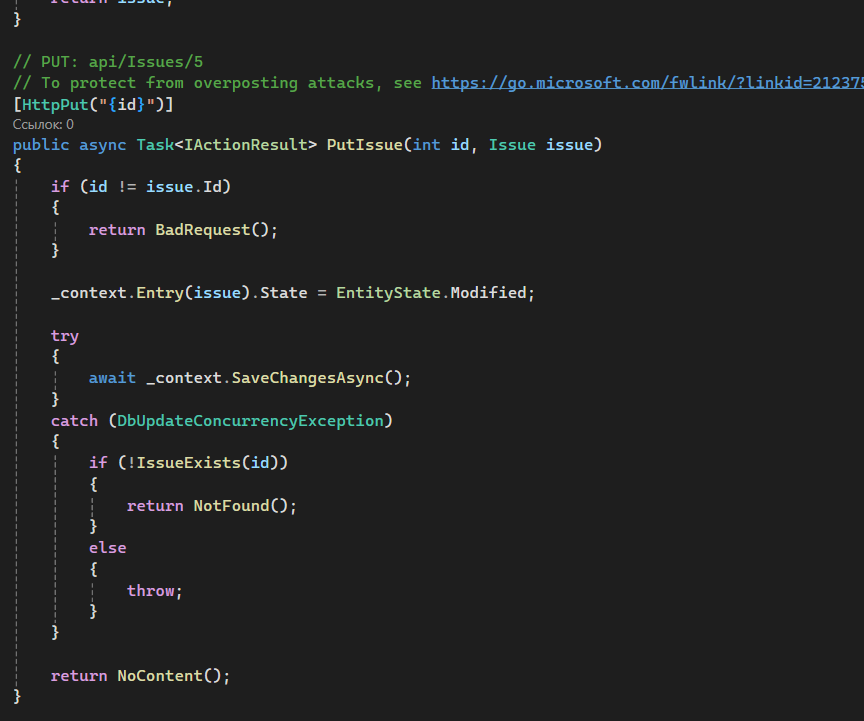
Issue Controllers:



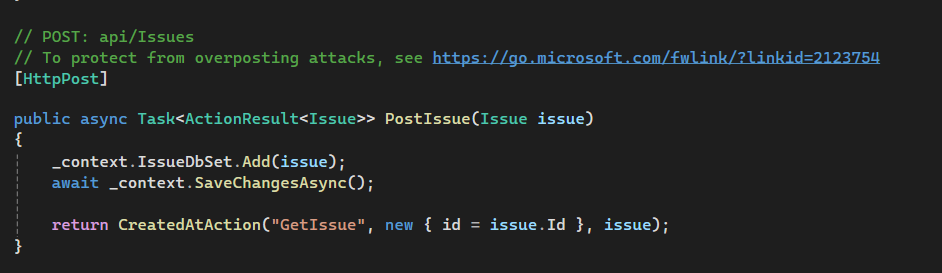
*GET(LIST)*



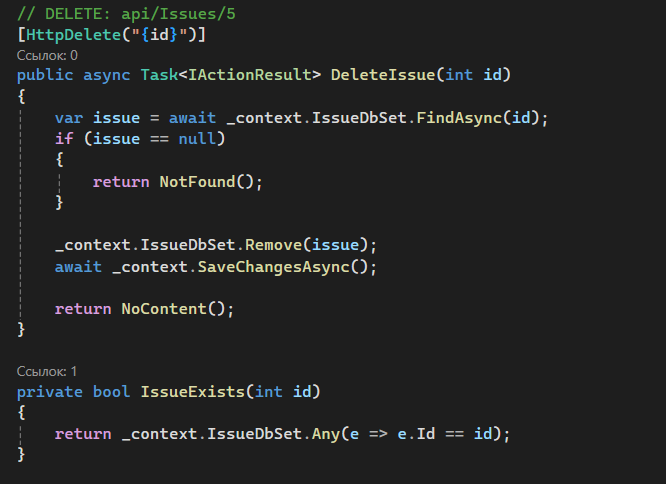
*GET*



*PUT*



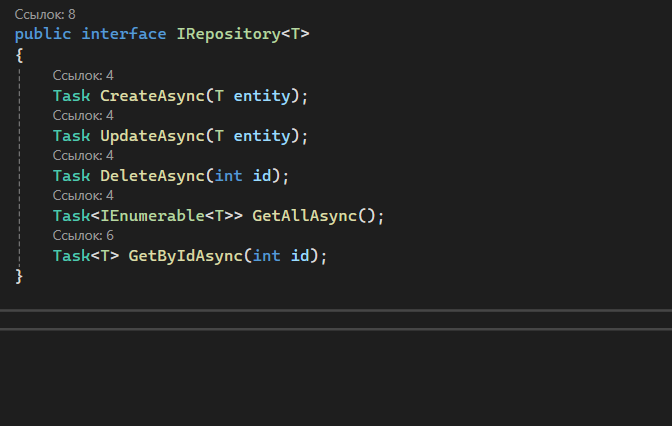
*POST*



*DELETE*

After Applying Pattern:

Repositories:



*IRepository Interface*

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

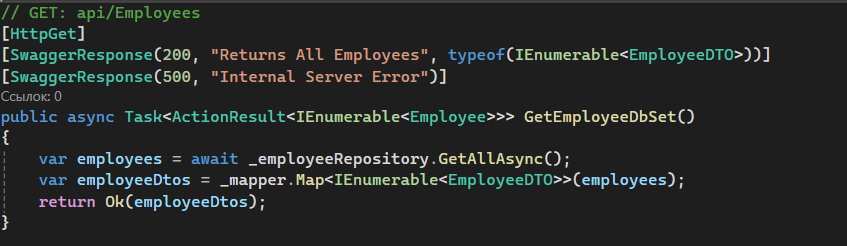
*Employee Repository*

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

*Issue Repository*

Employee Controllers:



*GET(LIST)*

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*GET*

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

*PUT*

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

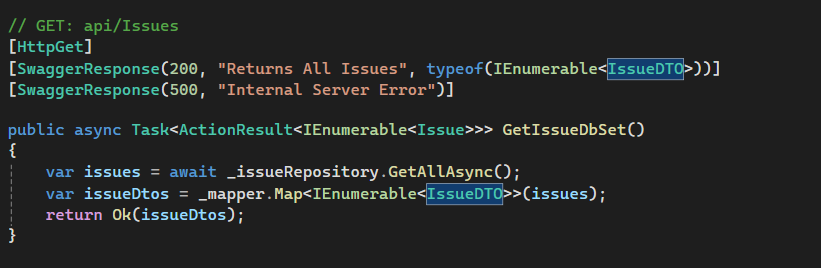
*POST*

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*DELETE*

Issue Controllers:



*GET(LIST)*

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

*GET*

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

*PUT*

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

*POST*

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

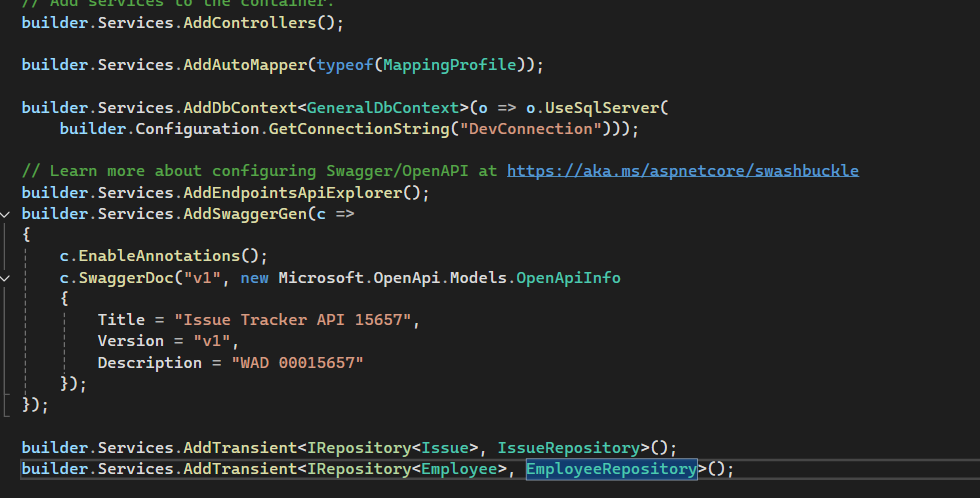
*DELETE*

Dependencies Injection:

**Purpose:** Dependency injection is used to provide dependencies to classes, reducing hardcoded dependencies and improving testability.

**Implementation:**

* Services and repositories are registered in the Program.cs file.
* The DI container resolves dependencies automatically.



Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

**API**

Employees

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

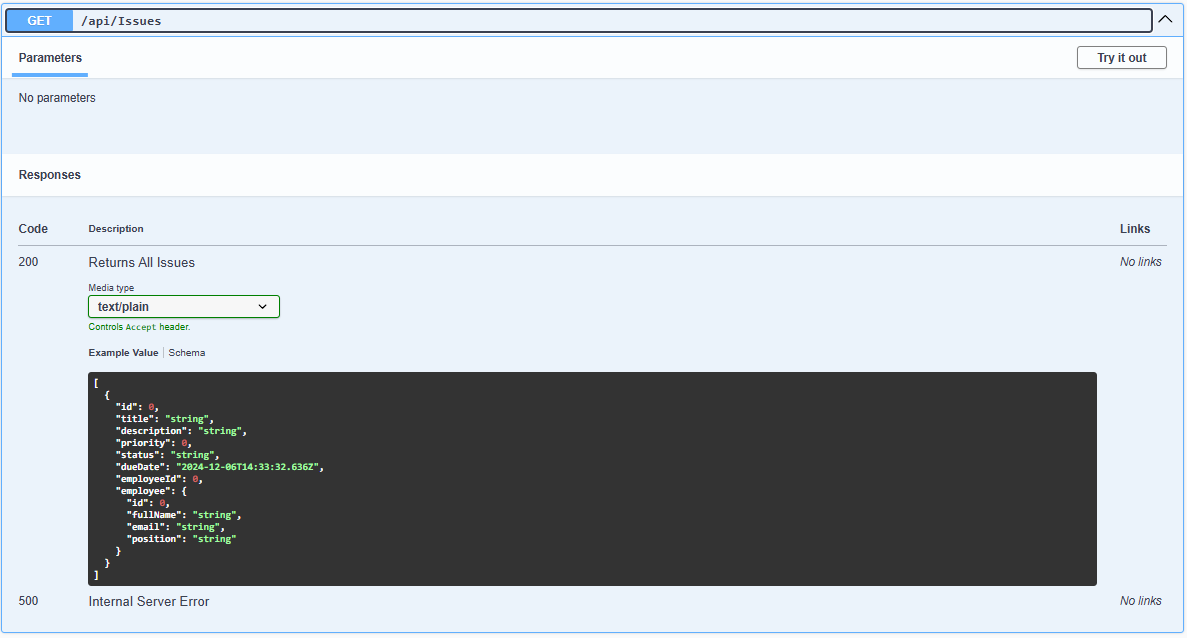
Изображение выглядит как снимок экрана, текст, программное обеспечение, веб-страница

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**Issues**



Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

**SPA**

**Github**

<https://github.com/wiut00015657/wad_00015657>

**Demo**

<https://youtu.be/JexWVJv39QQ>

**Libraries and Packages**

Angular Framework:

Google. (2024) Angular Documentation. Available at: https://angular.io/docs.

RxJS (Reactive Extensions for JavaScript):

Microsoft. (2024) RxJS Documentation. Available at: https://rxjs.dev.

ngx-toastr:

Maxim Ivanov. (2024) ngx-toastr Documentation. Available at: https://github.com/scttcper/ngx-toastr.

Luxon (Library for Date and Time):

Isaac Cambron. (2024) Luxon Documentation. Available at: https://moment.github.io/luxon.

Bootstrap:

Bootstrap. (2024) Bootstrap Documentation. Available at: https://getbootstrap.com.

Bootstrap Icons:

Bootstrap. (2024) Bootstrap Icons Documentation. Available at: https://icons.getbootstrap.com.

TypeScript:

Microsoft. (2024) TypeScript Documentation. Available at: https://www.typescriptlang.org/docs.

Angular Forms:

Google. (2024) Angular Forms Module Documentation. Available at: https://angular.io/guide/forms-overview.

Angular Router:

Google. (2024) Angular Router Documentation. Available at: https://angular.io/guide/router.

JSON Pipe and Async Pipe (Angular Common Module):

Google. (2024) Angular Pipes Documentation. Available at: https://angular.io/guide/pipes.

.NET Core Framework:

Microsoft. (2024) ASP.NET Core Documentation. Available at: https://learn.microsoft.com/en-us/aspnet/core.

Entity Framework Core:

Microsoft. (2024) Entity Framework Core Documentation. Available at: https://learn.microsoft.com/en-us/ef/core.

SQL Server:

Microsoft. (2024) SQL Server Documentation. Available at: https://learn.microsoft.com/en-us/sql.

AutoMapper:

Jimmy Bogard. (2024) AutoMapper Documentation. Available at: https://automapper.org.