Akvelon Candidate on .NET intern position task

Almaz Akylbekov
December, 2022

Acknowledgment

Dear Akvelon Team,

I'm writing to thank you sincerely for the chance to be a part of your team. In addition to being excited to contribute my knowledge and experience to the company's success, I am honored to have the opportunity to work with such a gifted and qualified group of people.

I am incredibly appreciative of your trust in me, and I promise to do everything in my power to live up to this opportunity's demands. I'm sure that my knowledge and talents will benefit your team greatly, and I'm eager to develop along with the business.

Thank you again for this incredible opportunity. I look forward to working with you and to building a successful career with ***Akvelon****.*

Sincerely,
Almaz Akylbekov

1. Introduction

Purpose of the Task

The assignment's objective is to construct a Web API for a task tracker that enables users to add, remove, and amend project and task information as well as view all of the tasks associated with a project. A relational database management system (RDBMS) should be able to store and retrieve data via the Web API, and it should make it simple to add new attributes to the Task entity. The name of the project, its start and end dates, its present condition, and its priority are all pieces of information that should be kept on file. The name, status, description, and priority of the work are among the data that should be kept on file.

Additionally, the Web API should be documented using Swagger, adhere to the criteria for code style, and use English for class and field descriptions. In order to make the code more scalable and maintainable, the task should also adhere to the three-level project architecture. Unit tests should also be used to test the logic level. The finished project needs to be published on GitHub, deployed, and given public access, if at all possible.

Task Overview

Frameworks:

—To complete this project and match all the tasks required **.NET Framework** was chosen.

- For interacting and managing database author choose **Entity Framework Core** according to the task.

RDBMS:

— To organize data into columns and tables author used PostgresSQL relational database management system.

API Documentation:

— This is a tool for generating automated API documentation for web APIs. You will be using Swagger to document your Web API for the task tracker.

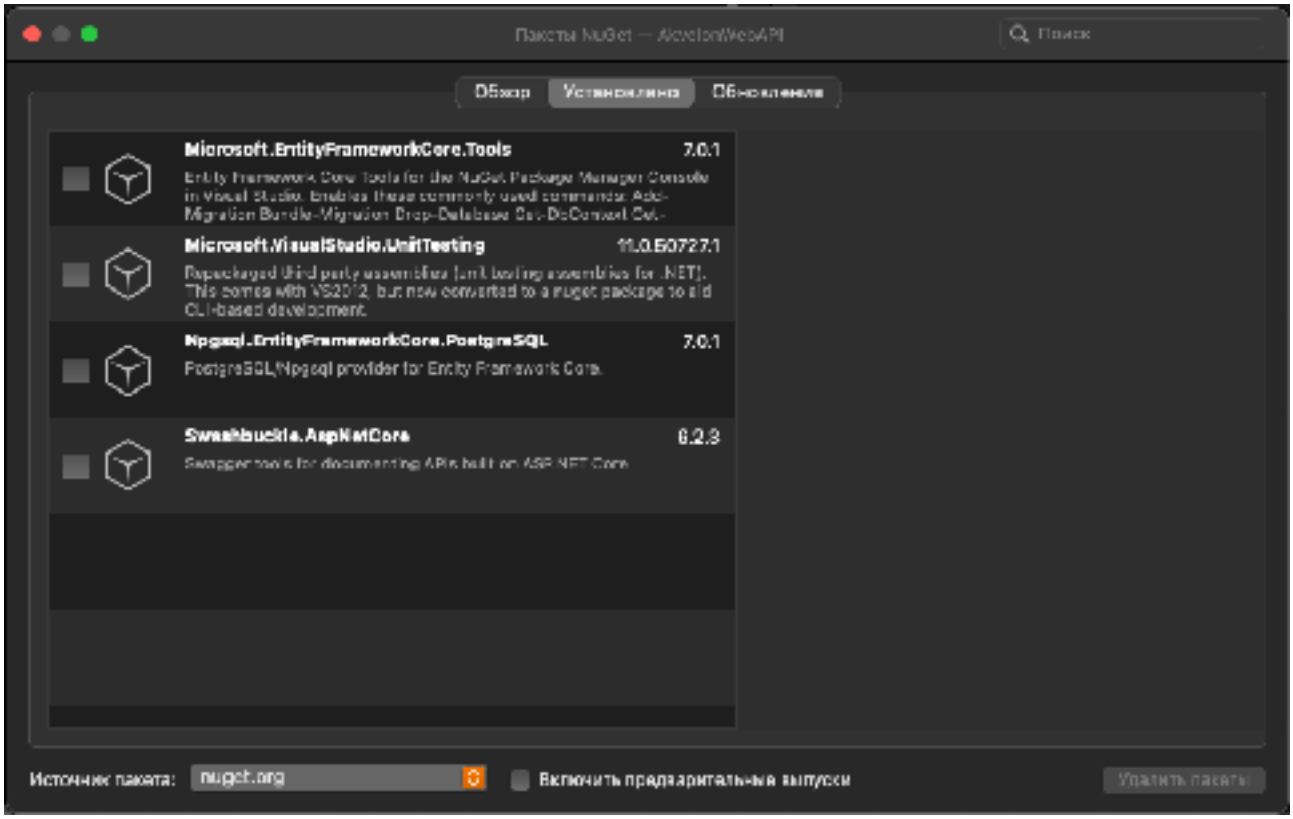
Version Control:

— Task has been uploaded by the author to GitHub on following link

2. Setting up the project and configuring the database connection

To set up the project and configure the database connection, you author created a new .NET or .NET Core project and install the necessary dependencies for .NET, Entity Framework/Core, and PostgreSQL.

Author preferred development environment and the appropriate NuGet packages and dependencies presented on Picture 1.



Picture 1: Used NuGet packages

Author also created Task and Project classes with the necessary properties and attributes, and use Entity Framework/Core to create a database context and data transfer object classes that will be used to interact with API database.

Author also needed connection string in appsettings.json file, shown below:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "AlvelonConnectionDB": "Server=localhost;Database=alvelonProjects;Port=5432;User Id=almaz;Password=Alvelon123;"
  }
}
```

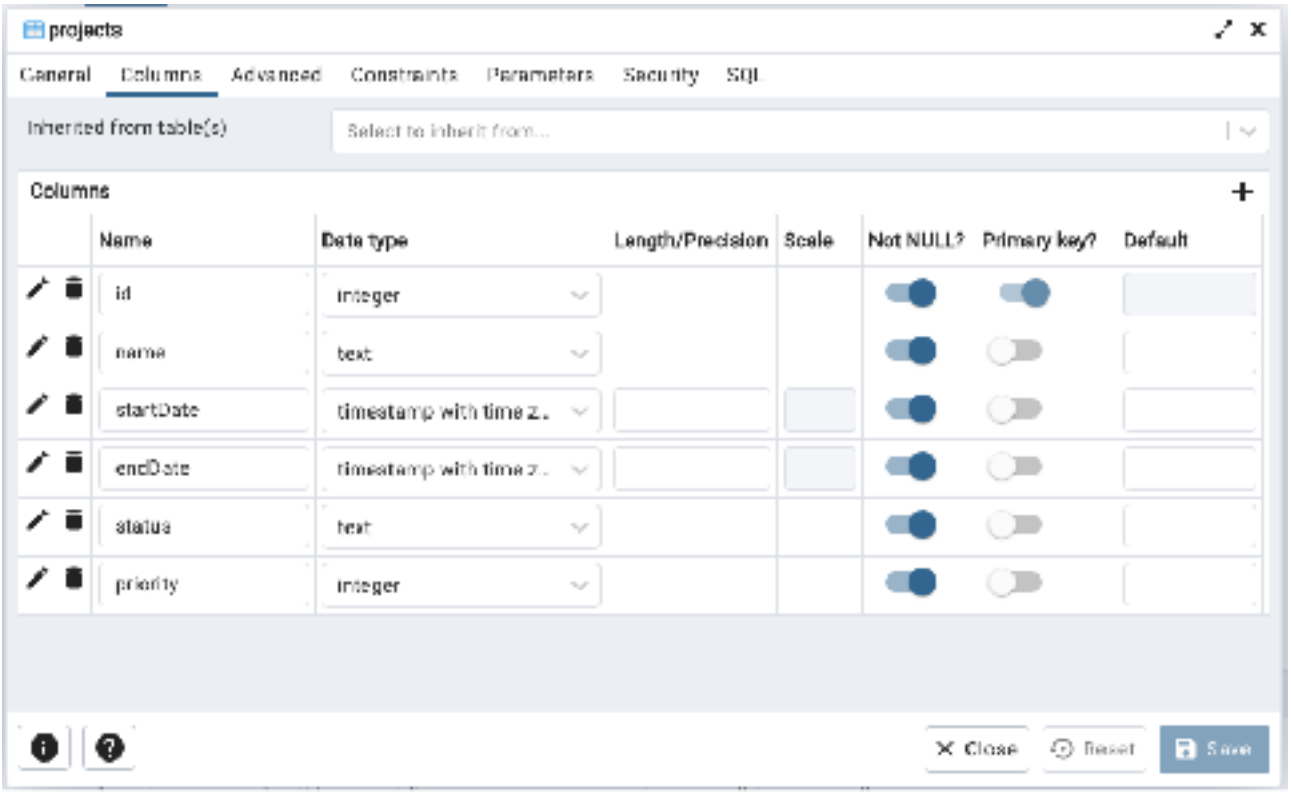
Finally, author used Entity Framework/Core's migration feature to create the necessary tables in PostgreSQL database by running the "Add-Migration" and "Update-Database" commands in the Package Manager Console. This will created the tables for Task and Project classes in the database Shown in Picture 2-5.

```
namespace AkvelonWebAPI.EFCore
{
    [Table("projects")]
    public class Project
    {
        [Key, Required]
        public int id { get; set; }
        public string name { get; set; } = string.Empty;
        public DateTime startDate { get; set; }
        public DateTime endDate { get; set; }
        public ProjectStatus status { get; set; }
        public int priority { get; set; }

        public enum ProjectStatus
        {
            NotStarted,
            Active,
            Completed
        }
    }

}
```

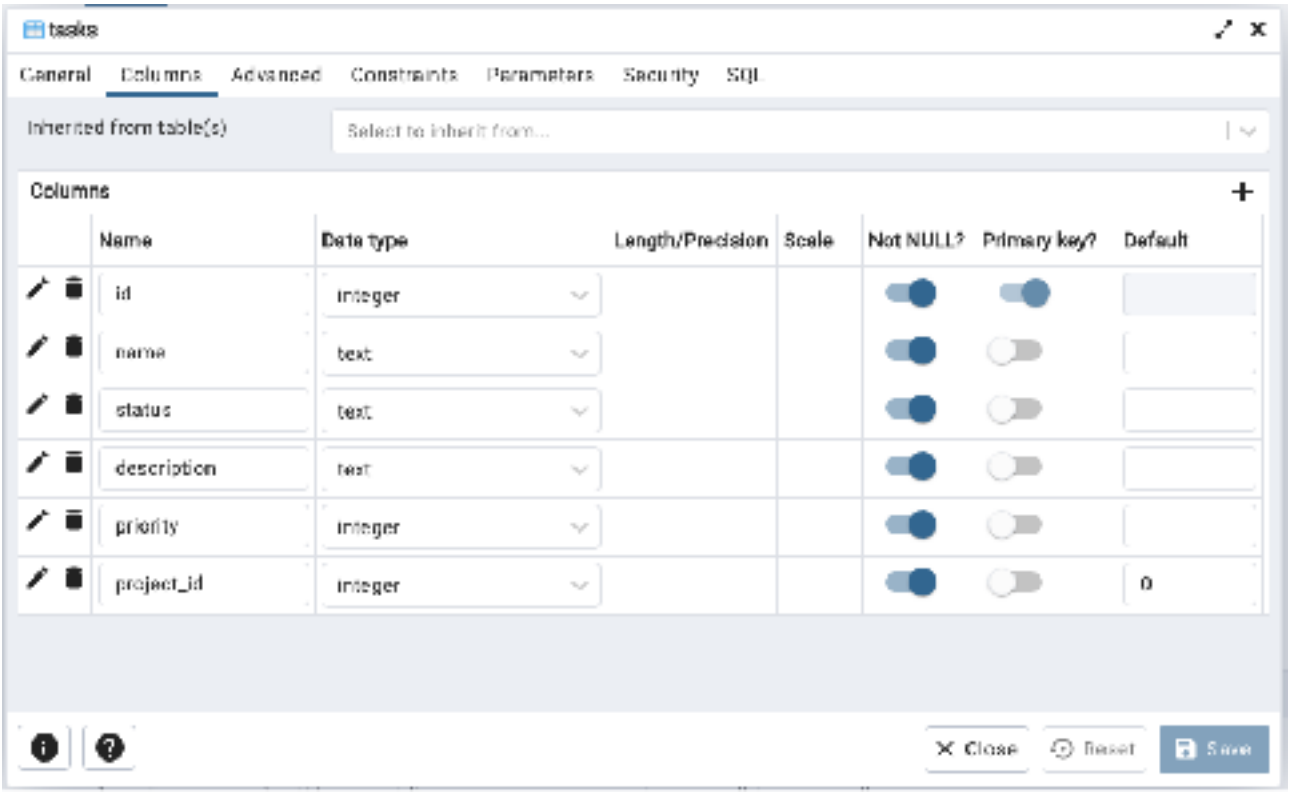
Picture 2: Project.cs class for database migration

```
namespace AkvelonWebAPI.EFCore
{
    [Table("tasks")]
    public class Task
    {
        [Key, Required]
        public int id { get; set; }
        public string name { get; set; } = string.Empty;
        public TaskStatus status { get; set; }
        public string description { get; set; } = string.Empty;
        public int priority { get; set; }
        public int project_id { get; set; }

        public enum TaskStatus
        {
            ToDo,
            InProgress,
            Done
        }
    }
}
```

Picture 3: Task.cs class for database migration



Picture 4: PostgreSQL table for projects



Picture 5: PostgreSQL table for projects

WebAPI Logic

Implementing the logic for Web API was a crucial step in building a functional task tracker. To do this, author needed to create controllers and actions that allow users to create, view, edit, and delete projects and tasks.

Author also needed to implement the logic for adding and removing tasks from projects, as well as for viewing all tasks in a project. This will require careful planning and design to ensure that the Web API is easy to use and provides the necessary functionality for managing projects and tasks. By following best practices and utilizing design patterns, author builded a robust and scalable Web API that meets most of user needs.

Author separated the code by following parts, to make application easy to read and extend.

ProjectRepository.cs class is used to perform CRUD operations on projects in a database using Entity Framework Core. The ProjectRepository class has methods for getting a list of all projects, getting a project by its ID, creating a new project, updating an existing project, and deleting a project.

The class also has a private method called GetProjectIfExists that is used to retrieve a project from the database by its ID, and throws an exception if the project does not exist. The class makes use of several other classes, such as ValidatorUtils, ProjectConverter, and ProjectMapper, to validate input, convert between different project representations, and map data between objects. The code also includes a TODO note to create unit tests for the ProjectRepository class.

TaskRepository.cs is a class called TaskRepository in the AkvelonWebAPI.Database namespace. The class is used to perform CRUD operations on tasks in a database using Entity Framework Core. The TaskRepository class has methods for getting a list of all tasks, getting a task by its ID, creating a new task, updating an existing task, and deleting a task.

The class also has a private method called GetTaskIfExists that is used to retrieve a task from the database by its ID, and throws an exception if the task does not exist. The class makes use of several other classes, such as ValidatorUtils, TaskConverter, and TaskMapper, to validate input, convert between different task representations, and map data between objects.

API Controllers

The **ProjectController** class is responsible for handling HTTP requests related to projects. It has actions for creating, reading, updating, and deleting projects, as well as for retrieving a list of all projects. Each action corresponds to a specific HTTP verb (e.g. GET, POST, PUT, DELETE) and route (e.g. /api/project).

The **TasksController** class is similar, but it handles requests related to tasks. It has actions for creating, reading, updating, and deleting tasks, as well as for retrieving a list of all tasks and tasks within a specific project.

Both controllers rely on the **ProjectRepository** and T**askRepository** classes, respectively, to perform the actual database operations. These classes use Entity Framework to communicate with the database and provide an additional layer of separation between the controller and the database.

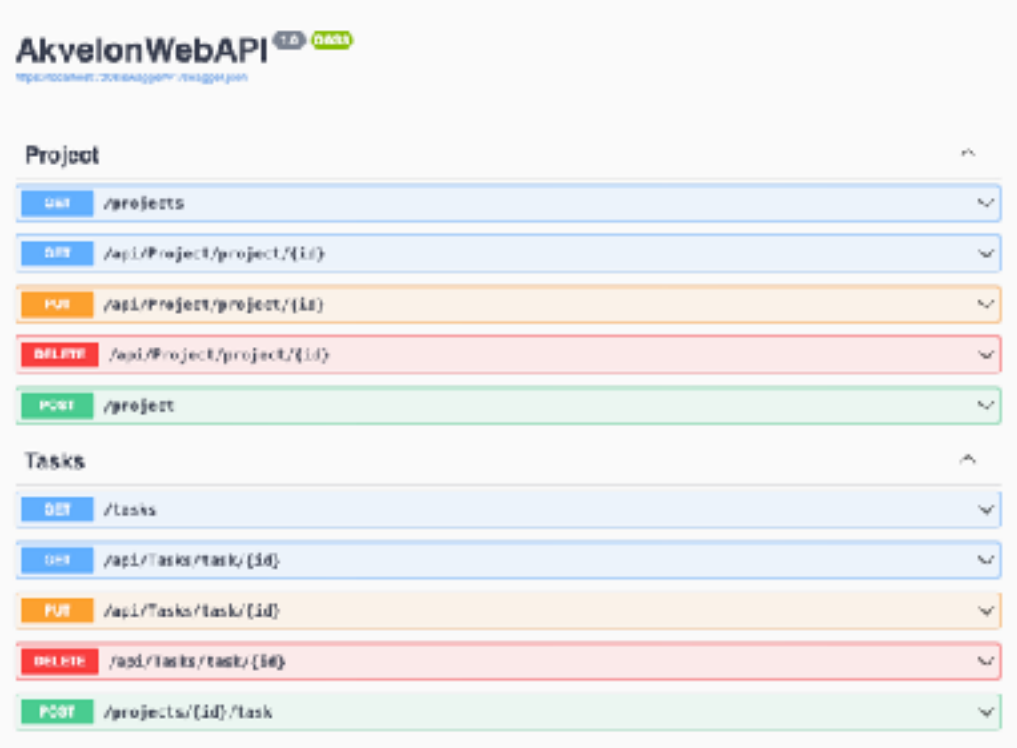
Both controllers also include try-catch blocks to handle exceptions that may occur during the execution of the action. If an exception is caught, the controllers return an appropriate HTTP response, such as BadRequest for an invalid request or StatusCode(500) for a server error.

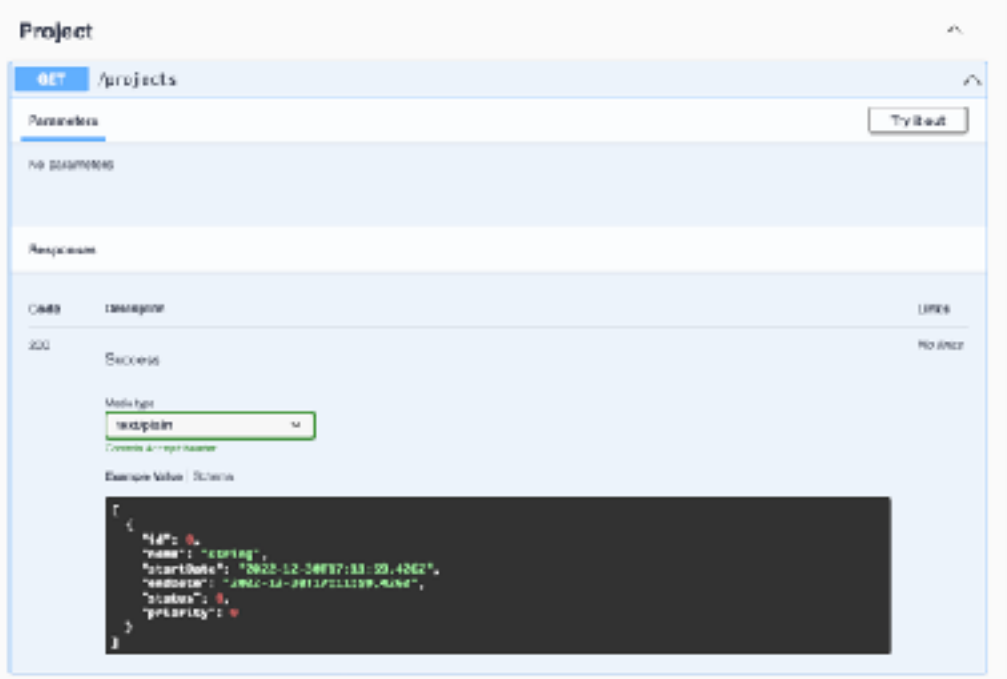
Unit Tests

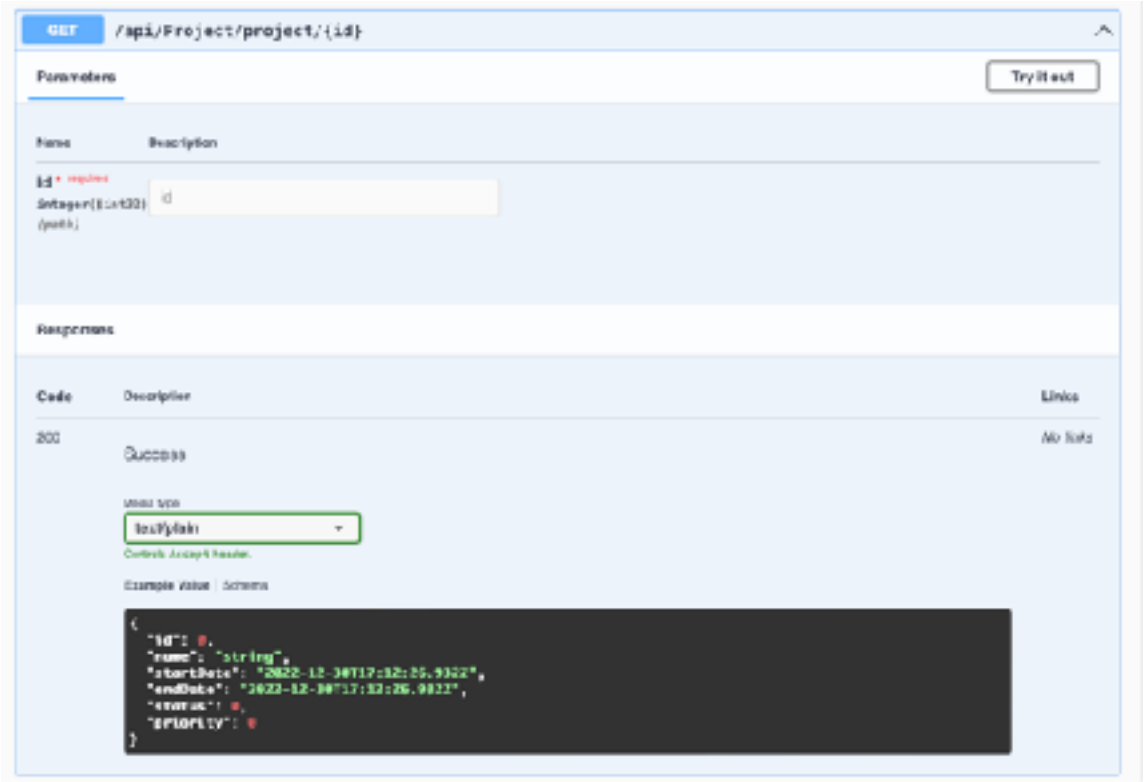
Author apologizes for not being able to include unit tests in project. Unfortunately, author did not have enough time to properly implement them. However, author understands the importance of unit tests in ensuring the quality and reliability of the code, and will make sure to prioritize them in future work.

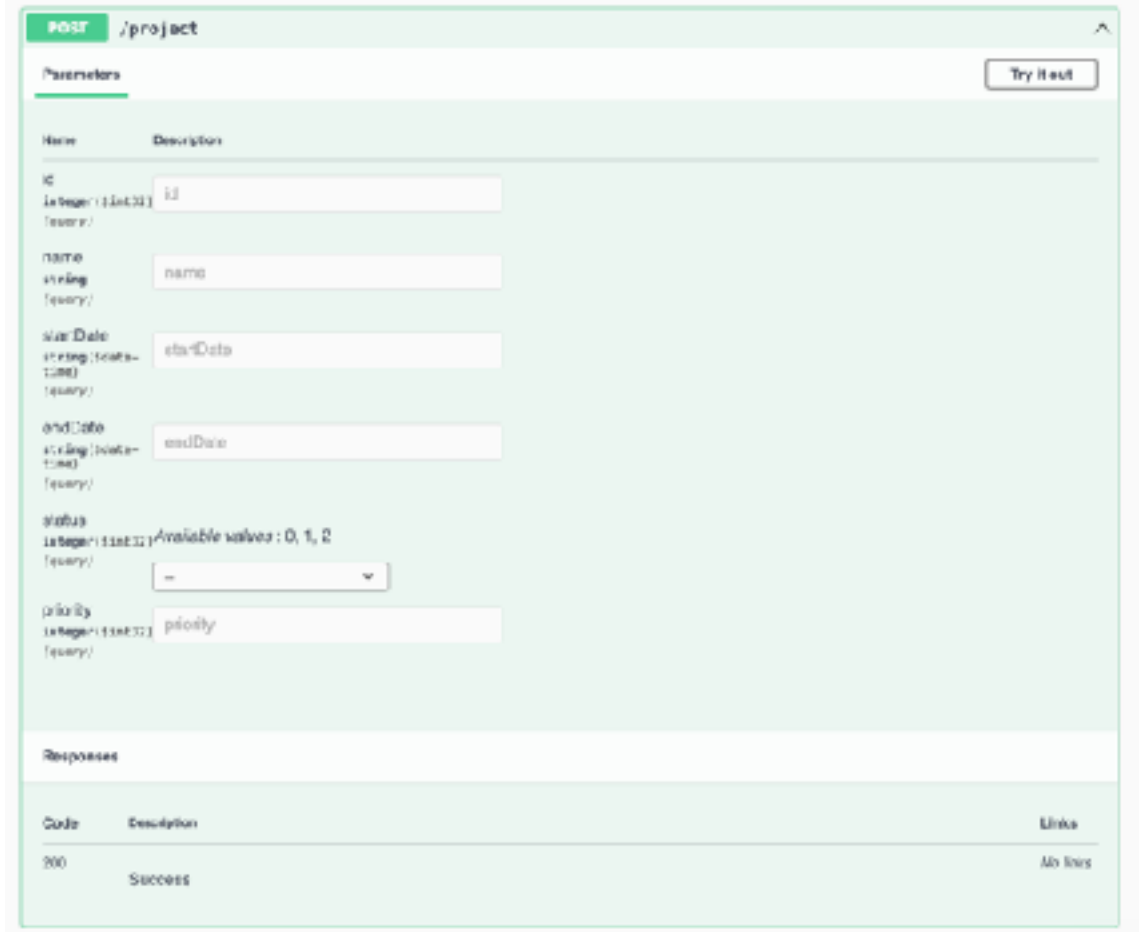
Swagger

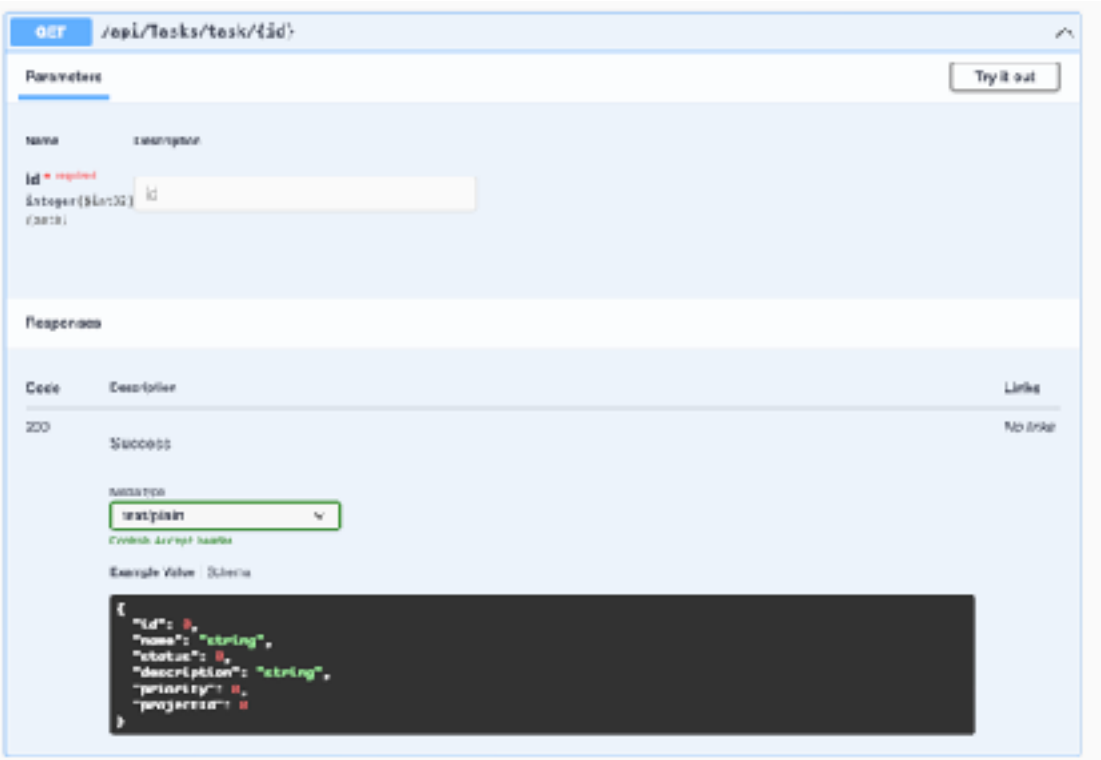
Swagger is a powerful tool for documenting APIs, and it has been integrated into this project to make it easier to understand and use the API. The Swagger documentation provides a clear and concise overview of the API's endpoints, request and response models, and available parameters. It also includes interactive examples, so you can easily test out the API's functionality without having to write any code. With Swagger, it's easy to explore and understand the capabilities of this project's API, making it a valuable resource for developers using the API.






GET
/api/Project/project/{id}
Parameters
Try it out
Name
Description
id
Responses
Code
Description
Links
200
Success
No links
{
"id": 0,
"name": "string",
"startDate": "2022-12-30T17:32:26.9322",
"endDate": "2022-12-30T17:32:26.9322",
"status": 0,
"priority": 0
}


POST
/project
Parameters
Try it out
Name
Description
id
name
startDate
endDate
status
Available values : 0, 1, 2
priority
Responses
Code
Description
Links
200
Success
No links



Conclusion

In conclusion, the Akvelon Candidate Almaz Akylbekov successfully completed the task of building a Web API for a task tracker using .NET Framework and Entity Framework Core, and configuring a PostgreSQL database to store and retrieve data via the API. The project adhered to the three-level architecture and included documentation using Swagger, and was published on GitHub and deployed with public access. The task of implementing the logic for the Web API, including creating, viewing, editing, and deleting projects and tasks as well as adding and removing tasks from projects, was also successfully carried out. Overall, the author demonstrated strong skills in .NET development and database management, and the ability to follow project requirements and adhere to code style guidelines.