**High Level Project Description:**

**Task-Manager Project**

Topics in Information Systems and Programming Languages - Mini Project

Table Of Contents:

1 …………………………………… How to run this project

2 …………………………………… General design notes

2 …………………………………… A diagram of modules in the project and their connections:

2 …………………………………… Entities

3 …………………………………… System Design and Client Communication:

4 …………………………………… Data corruption:

4 …………………………………… Errors

4 …………………………………… Concurrency

**How to run this project:  
1. Clone the git repository:**

<https://github.com/nogazax/miniproj-zederman.git>

**2. Install the following packages**

- Microsoft SQL Server

- Microsoft.AspNetCore

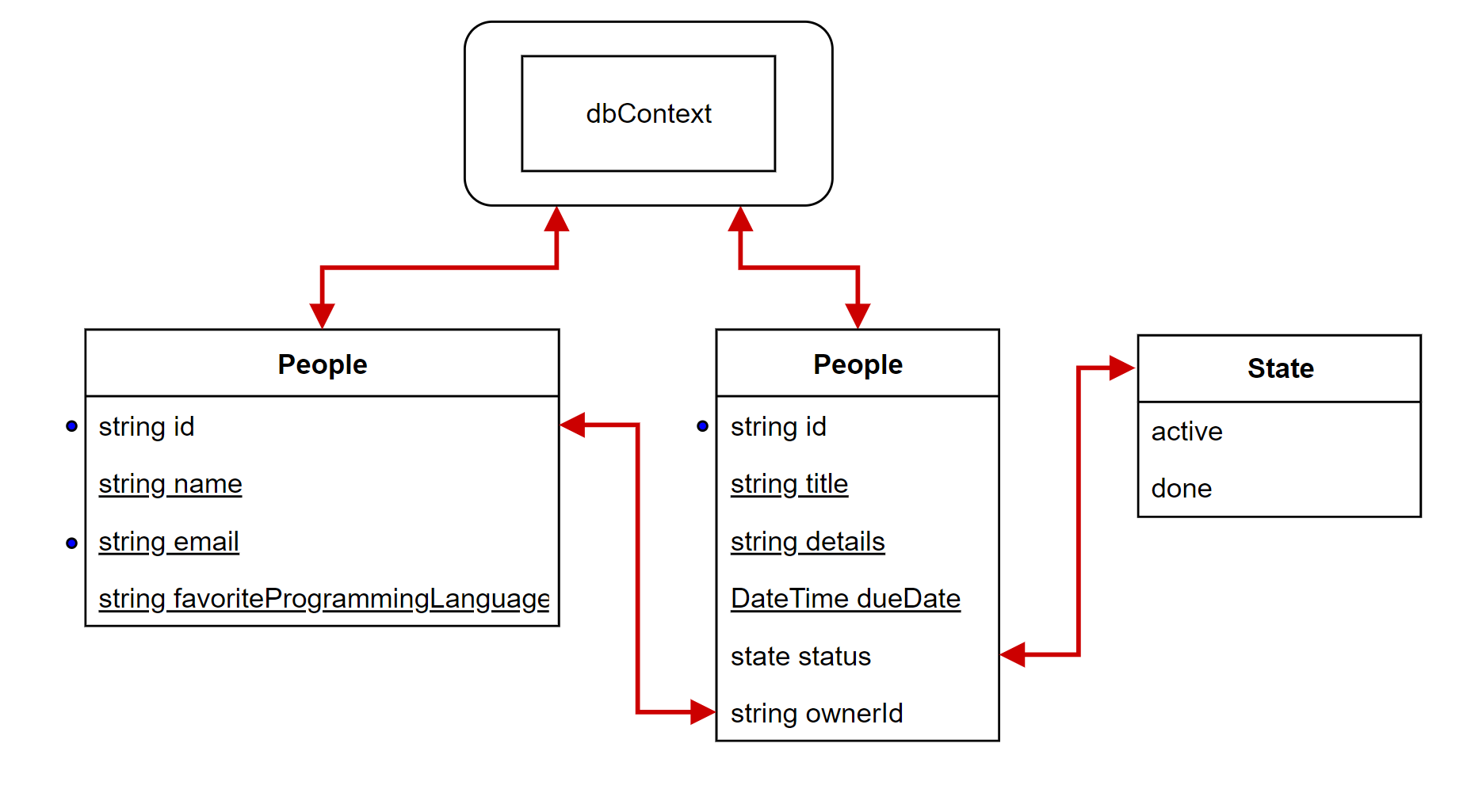
- Microsoft.AspNet

-Microsoft.EntityFrameworkCore

**General Design Notes:**

The server was implemented with abstraction of Data – Model – API in a way that keeps the pattern of designs like MVC, and ORM to use and save different objects in the database.

**A diagram of modules in the project and their connections:**



-Fields with underscore are mandatory when a client uses a POST endpoint.

-Fields with blue dot are unique fields (therefore it would be impossible to add another field from the same model with the same value).

-We chose this model because the framework we are working with is using the stracture of MVC- ‘Model-View-Controller’.

-The classes above are abstractions of the database tables we are working with (ORM).

**Entities:**

We are using the appropriate data-structure for each object in the system:

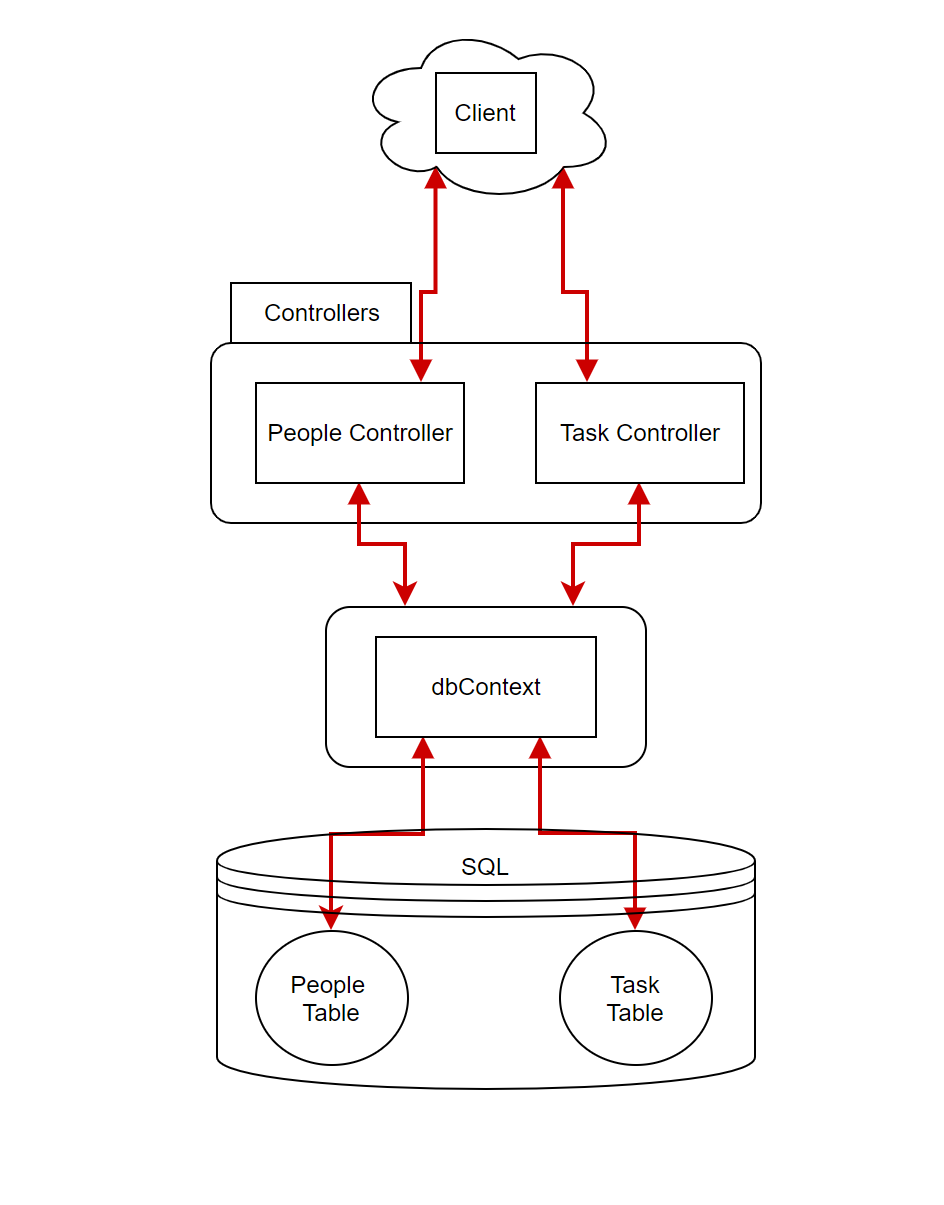
-Person: When a new person object is created, the system will automatically generate a unique-id for him.

-Tasks: When a new task object is created, the system will automatically generate a unique-id and make sure that the supplied due-date is converted into Time object.

-Status: An enumerable data structure. In a case where an illegal status is provided, the system will notify the user.

**System Design and Client Communication:**

The system structure and client communication can be described using the following diagram:

When a client sends a message to the server, it reaches the relevant controller, and redirected according to the request path (i.e post, get, put).

The controller, using the database-context communicates with the database in order to get or update the existing information as requested, and sends back a response to the client.

**Data corruption:**

The .NET framework is built in a way that it gives to direct access to the database, but by using the db-context of the project, which acts as a separation layer and keeping the data – integrity (throws relevant exceptions for errors like duplicate keys or unauthorized access to non-existing entity).

In addition, we implemented manual checks for any input coming from the user to make sure that the requests will not damage the database.

**Errors:**

For any request containing illegal values according to the API, we made sure to send the appropriate response error.

In addition to

**Concurrency**:

The server is expected to handle multiple requests simultaneously. The advantage of the framework we chose (.Net) is that the concurrency is handled by the SQL server, and the parallel connections are being handled by the controllers.