SE 4458 – Final Project(Blood Bank)

Yarkın Ataç – 19070006020

Design

Back-End

Initially, I created an Web API using .NET 7 and employed EF Core and Ocelot(but I don't have credits for azure because I can not create a apigateway)for managing database operations. After installing necessary libraries, I chose a layered architecture to structure the codebase because with the research I did and chatgpt, I thought it would make more sense. The division into layers aims to enhance code readability, reduce complexity. The application layers are: API Gateway, Web API, Business Logic Layer and Data layer.

Data Layer

Examining the data layer reveals four directories: models, models.dto, repositories, and resources.

- The Models directory contains the assets I designed to retrieve data from the database, pass data to the database, and manage my database connection.
- In the ModelDTO directory, store Data Transfer Objects (DTOs) and classes pertinent to the application.
- The repositories folder hosts my Repository Base class, a generic structure crafted through the repository pattern. The reason I did this was because this method was used in the video I watched for my graduation project and it seemed easier.

Business Logic Layer

In this layer, files and business logic specific to the relationships and models with the database are defined..

API

In this layer, I configured JWT to integrate the services to be used in the APIs and then create role-based and tokens. Next, the necessary controllers for my APIs were created and the appropriate endpoints were created.

API Gateway

My main purpose here was to create a gateway by sending requests to these endpoints with Ocelot after distributing my APIs to web services, but I could not create them because my Azure credits ran out, but I had the necessary additions for configuration in my source code.

Assumptions

Backend

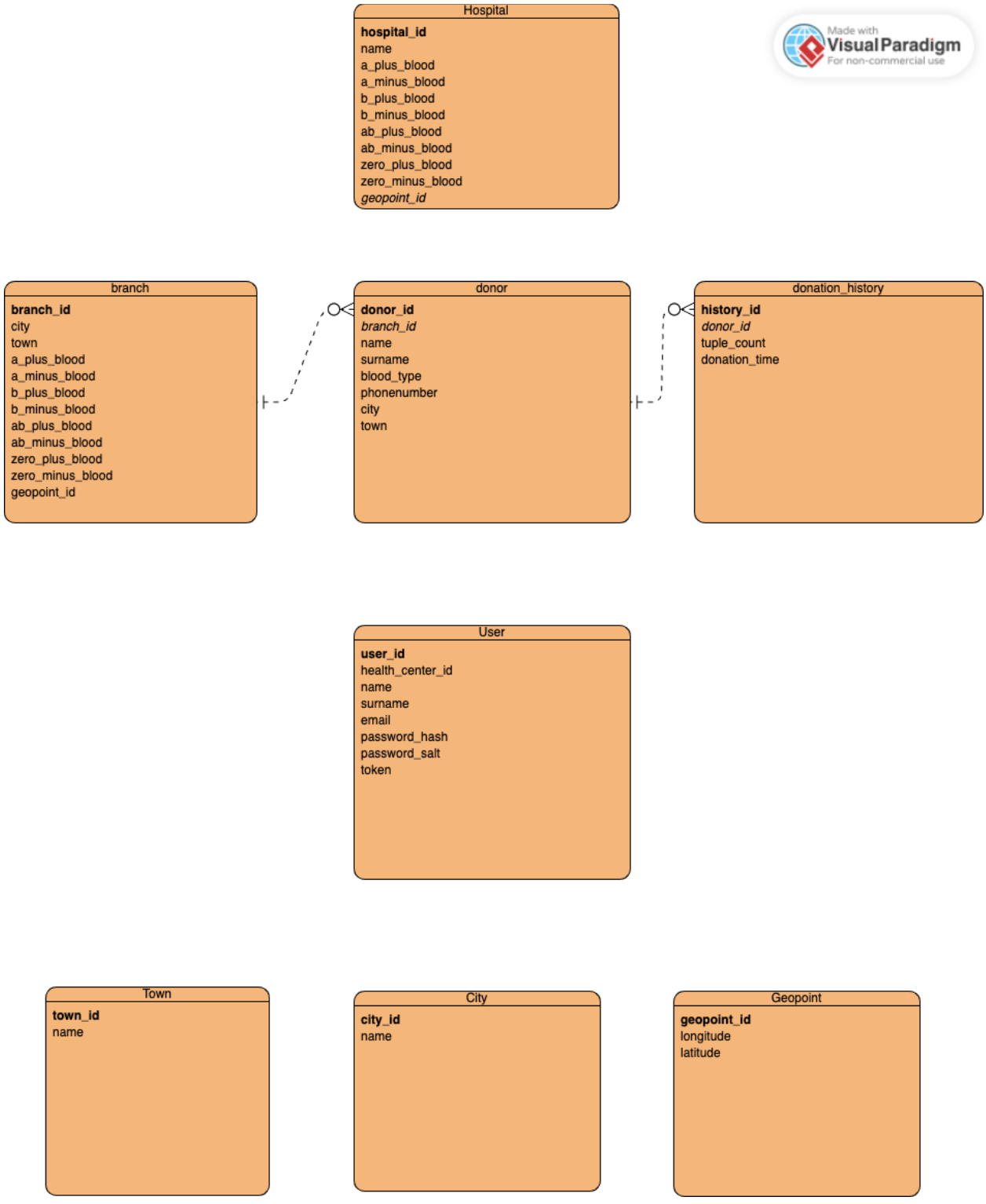
In designing the database, I consolidated separable yet related components under a unified name within a single database. After transforming the project components into tables, I set up the necessary environments for my project's development. As part of the solution, I created additional projects including Data, Business Logic, WebAPI, and API Gateway. Subsequently, I established a connection with Azure. This was done through a friend's Azure account because my own account was suspended after exhausting the credit limit. As I cannot deploy my APIs, I am unable to utilize Azure App Services.

Having worked on certain concepts in the midterm project facilitated the implementation of similar aspects in the final project. One of the challenges was accurately defining the relationships between tables when consolidating them.

Furthermore, in establishing specific logic, such as when making a blood request, I needed to first verify the specified city and district. Then I had to search within a certain range, like 50 kilometers, and if no blood was available, I was required to initiate blood requests. Challenges also included verifying the sufficiency of the counterpart's blood and managing the changes in blood count following a transfusion.

I also set up a Blob service on Azure and configured a CDN service within it. The images I uploaded are stored there.

ER Diagram



In my design approach, I began with the user database, establishing a table for users that encapsulates basic information like first and last name, as well as email. I treated branches and hospitals as distinct entities due to their differing roles: branches are able to enlist donors and collect blood, whereas hospitals

are in the position to request blood. Therefore, I allocated specific roles to each and included a 'health_center_id' column to delineate the affiliation of the healthcare members—branch IDs for branch members and hospital IDs for those linked to a hospital.

Within the donor database, attributes such as city and state were incorporated to denote the branch locations. To manage the inflow of blood from various donors, I introduced columns for tracking the quantity of each blood type received. Additionally, geopoint data was integrated to enable hospitals to solicit blood from branches located within a 50-kilometer range. A separate table was crafted to monitor the volume of blood donated by each individual.

For hospitals, I developed a blood bank table to display the inventory of available blood.

Lastly, I set up essential location tables in the location database, ensuring all pertinent geographic details—cities, states, counties, and other relevant geographic data—were systematically recorded for the system's use.

Issues

Azure

Because my Azure credit ran out and my account was suspended, I could not deploy the WebAPIs I created, and therefore I could not create an APIGateway and use these APIs with the help of Ocelot. I learned about ocelot as a result of my research and with the recommendation of chatgpt.

Back-End

Occasionally, issues arose with data entries due to inadvertent use of incorrect capitalization or misspelling of terms such as 'city' and 'town'. At times, uploading photos did not proceed as smoothly as I intended.

A notable hurdle we faced was the substantial credit consumption on Azure. After depleting my allotted credits, I unintentionally began using a friend's account. To complicate matters, the expenses escalated following certain activities within Azure API Management. Given that I'm unable to set up Azure API management myself, I'm not fully aware of the potential challenges that might come with using the gateway.

Links

Github: https://github.com/yarkinatac/Se4458Final

CDN azure link = https://se4458finalyarkin.blob.core.windows.net