Author: Georgescu Vlad
Group: 30444

Documentation
Distributed Systems, Assignment 1
Request-Reply Paradigm
Online Medication Platform

Content

1. Conceptual Architecture.....pag 2
2. Database Design.....pag 2
3. UML Deployment Diagram.....pag 3
4. Build and Execution Considerations.....pag 4
5. Bibliography.....pag 5

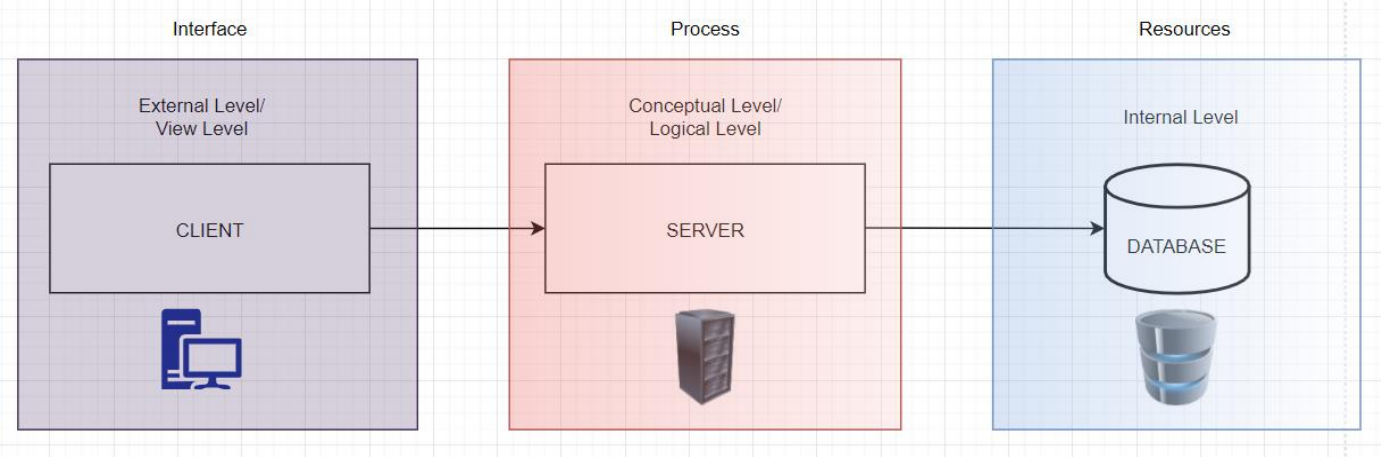
Author: Georgescu Vlad
Group: 30444

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA ROMANIA

1. Conceptual Architecture

The system is based on a 3 Tier architecture, containing Client, Server and Database. Client is at the view level and it makes requests to the Server, which is at the logical level. The server manipulates the request and makes the necessary adaptation and sends the request forward to the Database, accordingly.

Below, can be seen the diagram of the conceptual architecture and how does the data flow between the architecture components:



2. Database Design

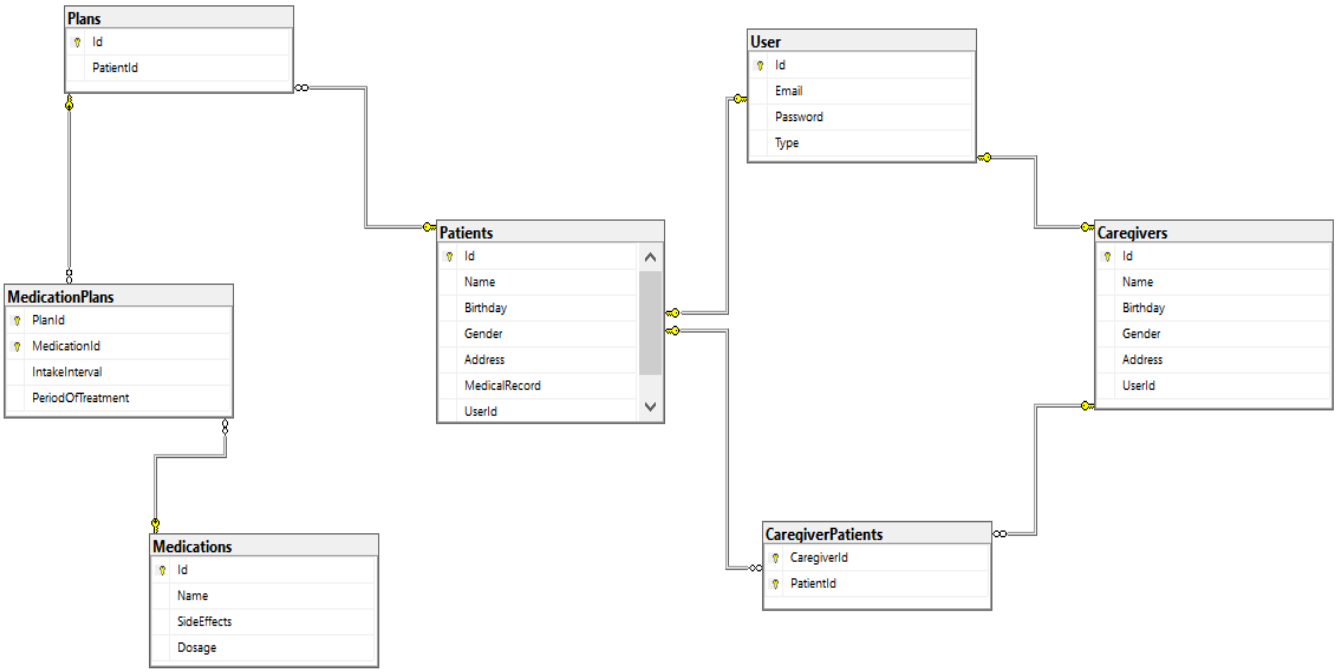
Regarding the Database, Sql Server was used. The database consists in seven tables: User, Caregivers, Patients, CaregiverPatients, Plans, MedicationPlans and Medications. User table contains information about user accounts such as email and password and also the user type. Patients contain the details regarding to patients and the foreign key to the user table, Caregivers contain the details regarding to the caregiver and the foreign key to the user table. CaregiverPatients is the many to many table between the Caregivers and the Patients and contains the ids of the linked caregivers and patients. Plans contain the id of the plan and the foreign key to the patients table. Medications contain the details about medications and MedicationPlans is the many to many linking table between Plans and Medications and contains the ids of associated plans and medications.

There are the following relationships between the Database tables:

- Between User and Caregivers and between User and Patients there is an one-to-one relationship because each caregiver and each patient have only one user account
- Between Caregivers and Patients there is a many-to-many relationship because a caregiver can cure many patients, and a patient can have more than one caregiver that take care of
- Between Patients and Plans there is an one-to-many relationship because a patient can have multiple medication plans
- Between Plans and Medications there is a many-to-many relationship because a plan can consist in many medications, and a medication can be part in many medication plans.

Author: Georgescu Vlad
Group: 30444

The SQL diagram of the database can be seen below:



3. UML Deployment Diagram

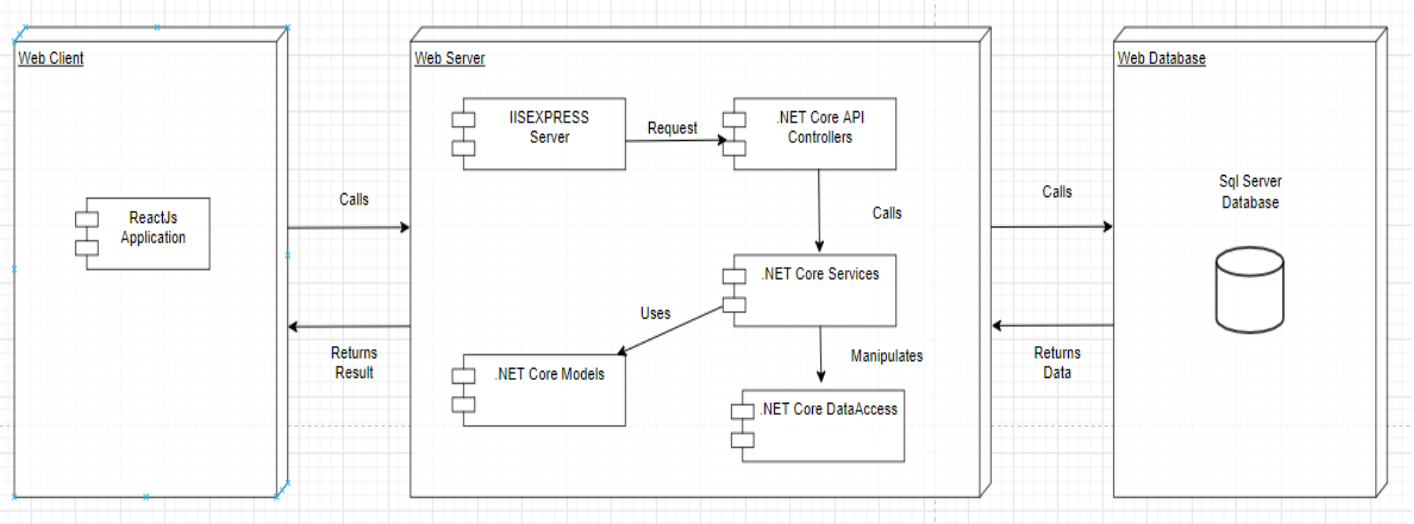
The system is based on a three tier architecture, as mentioned before. The Client application is a ReactJs application on typescript template and using React Hooks. Client makes requests to the Server in order to obtain the data that has to be displayed.

The Server application is a .NET Core application running on IISEXPRESS server. Server receives the request from the client. Controllers are responsible for getting the request and they adapt the requests and send them further to the services. Services are responsible for the business level and they make use of the data models in order to manipulate the data access for getting the required data. Data access is responsible to interact with the database and to make the right calls to get the needed data, or to store the needed data.

The database responds to the call with the asked data and this data is returned back to the client via controllers. Then, client displays the received data or uses it accordingly in order to perform its specific tasks.

Author: Georgescu Vlad
Group: 30444

Below can be seen the UML Deployment Diagram:



4. Build and Execution Considerations

Regarding build and execution consideration, there has to be noted that the application was written in .NET Core 3.1 for the backend part. Frontend part was written in ReactJs on Typescript template using React Hooks, which is the most modern way of writing React code nowadays, and for the database, Sql Server was used.

In order to use the application, Visual Studio 2019 is needed for backend, Visual Studio Code or any editor for frontend and SSMS 18 for Sql Server database. To link the backend to the database, a migration and an update to the database has to be made to DataAccess from Package manager console in Visual Studio. Backend has also a function for seeding, and from the beginning there exists a few rows of data in the database. Also, the proper connection string to the database mst be set in appsettings.json .To link the frontend application to the backend application, the proper backend link has to be inserted in the hosts.js in frontend. In order to run the applications, the server must be started from Visual Studio, and the client must be started from Visual Studio Code using the command npm start and the browser will start with the application.

The application was also deployed on Heroku cloud using Docker. For the backend part, a1-backend-georgescu-vlad application was build on Heroku using dockerfile and docker build –t appname . command, followed by heroku container:push –a appname web and heroku container:release –a appname web command. Backend had to be deployed using Docker since Heroku does not offer support for .Net. Same commands were also used to build the frontend application a1-frontend-georgescu-vlad, with according dockerfile and nginx.conf files. The database was deployed on Azure Sql Database cloud using the student subscription because again, Heroku does not offer free support to Sql Server. It was migrated from SSMS to Azure using some simple steps. But this has also a drawback, because the calls to the database are slow sometimes because of the weak servers provided by Azure for free subscriptions.

Author: Georgescu Vlad
Group: 30444

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA ROMANIA

5. Bibliography

https://jwt.io/introduction/
https://docs.microsoft.com/en-us/ef/core/modeling/relationships?tabs=fluent-api%2Cfluent-api-simple-key%2Csimple-key
https://docs.microsoft.com/en-us/ef/core/saving/transactions
https://dev.to/alrobilliard/deploying-net-core-to-heroku-1lfe
https://gitlab.com/ds_20201/react-demo/-/tree/docker_production
https://www.progress.com/documentation/sitefinity-cms/deploy-the-database-to-azure-sql
https://react-bootstrap.github.io/components/alerts/
https://react-icons.github.io/react-icons/