TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Andrei Gužovski 185818IADB

Artjom Pahhomov 186042IADB

FOOD DELIVERY PLATFORM

Building Distributed Systems

Course project thesis

Supervisor: Andres Käver

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Authors: Andrei Gužovski 185818IADB

Artjom Pahhomov 186042IADB

09.08.2020

Abstract

The goal of following course project thesis is to perform an analysis and implementation of a food delivery platform and as a result develop application for food stores, restarurants, cafes delivery systems which is well-designed back-end and front-end wise.

Web based food delviery platform solutions have become a mandatory part of a modern food preparing service-based small and larger businesses for promoting and increasing sales as well as organizing infrastructure. Currently, there are several solutions on the market, like Wolt or Bolt food, quality and properties of each can vary greatly. However, it appeared that there is no existing food delivery web platform which would be user-friendly both to user and restaurant as well as have all functionality and features needed by the common user.

The thesis will propose a solution that attempts to create a platform different from existing solutions, based on the reseach of existing solutions as well as on the market analysis..

In the first part of the work, the focus will be on gathering food delivery platform funtionality and features requirements for both customer and restaurant to use the platfrom. After that work proceeds with researching existing platforms, comparing, analyzing and finally selecting needed fameworks, libraries and architectures needed for implementation.

In the second part of the work, the main focus will be on documenting the implementation and testing process. Finally, all testing results will be evaluated,analyzed and a conclusion will be given.

As a result of this course project thesis, the MVP of a food delivery platform will be developed, tested and compared to existing web store solutions. After that work proceeds with researching

Introduction

The main goal of the thesis is to perform an analysis and implementation of a web based food delivery platform that will be powered by web-technologies. As a result, an MVP of an web based camping store for a modern small business will be created, which could be used as a basis for further development. The developed web store will have a set of unique properties, which are not present in other solutions: very lightweight back-end and front-end solution resulting in friendly user interface, REST based services for further development and improving. To achieve this goal, the analysis of requirements, goals, as well as analysis of different solutions, architectures, implementations, tools, and libraries will be performed. Gathered knowledge should provide an optimal solution for fulfilling the goal.

1. Analysis

1.1. Requirements provided by the supervisior

1) Minimal 10 functional entities - not including trivial m: m in-between tables, identity tables and language string / translation tables.
2) Layered clean architecture with:
    - Domain
    - DAL - using Entity Framework
    - BLL
    - Rest/Web
3) DTO mapping between every layer;
4) Swagger with XML docs;
5) API versioning support (and public versioned DTO-s);
6) Identity support, with RESTful API implementations for registering and login-in (JWT creation);
7) Base projects split into separate solution - and base packages hosted in Nuget.org
8) App (back-end and client) hosted in docker somewhere (azure for example);
9) Support for i18n;
10) Fully functional client app with usable UI/UX;
11) API endpoints correct security.

Food delivery platform project is perfectly suitable for this requirements since listed requirements are mostly basic for a modern well-designed web platform and further development tools choice is not restricted with anything but the skills of a developer.

2. Development methods analysis

In this part of the documentation, the author intends to go through certain technical aspects of relational databases. The subject of this paragraph is designing and implementing a SQL database for auditable/reversible data changes. This technology is also known as soft-update and soft-delete.

2.1. Soft-update and soft-delete

2.1.1. Single table

Designing a soft-delete and soft-update solution for a single table is a relatively easy problem. The author came up with a solution to using composite primary key –identification number and deletion time. This solution helps with binding all the related data together (the identification number does not change, only deletion time is changed) without the violation of the primary key's uniqueness. This solution, however, has its cons, because now the author is fated to always use some date as deletion time. Reason being the fact that the primary key or its parts cannot be nullable. The order of operations is the following:

- Soft-update:
    1. insert a copy of a record, but with the current time as deletion time;
    2. cascade update the record with the new information and last edit time.
- Soft-delete:
    1. set the deletion time to the current time.

2.1.2. One-to-many relationship

In order to solve the problem of updating the primary key and breaking all the possible joins, the author decided to use a composite primary key. The primary keys of both tables consist of identification number (also known as ID) and an additional field that contains the deletion time. This approach is also used in the next paragraph. Furthermore, when creating tables, the foreign key constraint of many-side was marked to cascade update. This means the referencing rows are updated in the child table when the referenced row isupdated in the parent table which has a primary key. So, taking into consideration all the aspects, the author came up with the following order of operations:

- Soft-update one-side:

1. insert a copy of a record, but with the current time as deletion time;
2. add a copy of all the dependent records in the many-table, also updating deletion time of them and foreign key (deletion time–part of a composite primary key of one-table);
3. cascade update the record with the new information and last edit time.

- Soft-delete one-side:
  1. cascade update the record with the new deletion time;
  2. manually update the dependent records with the new deletion time.

Although, when soft-deleting data from the many-side, it is possible to save the state of data at the one-side, author decided not to implement this because, in author's point of view, data at the many-side can change too often, and it could have caused table at the one-side of the relation to containing a lot of basically same records. It is still possible to return data as it was in each moment of time because one-side does not contain the time of creation/deletion of the many-side. So, soft-update and soft-delete on the many-side look the same as described in paragraph 1.1

2.1.3. One-to-zero/onerelationship

This relationship turned out to be the hardest to implement the soft-delete and soft-update on, nevertheless, it was not impossible. A solution that the author came up with was to add the unique constraint to zero/one-side which consists of three columns: identification number of the parent, deletion time of the parent (being the foreign key of a parent) and deletion time of the child. The latter helps by pointing at the presence of a valid child, so the author gets rid of multiple problems. For example, if a new parent table already has a deleted child, then a new child can still be added because although the combination of deletion time and ID of the parent is not unique, the deletion time of the child adds the needed uniqueness. If the valid child is already present at the table, one cannot add more children because of the unique constraint.

- Soft-update one-side:
  1. insert a copy of a record, but with the current time as deletion time;

2. inserta copy of all the dependent records in the zero/one-table, also updating deletion time of them and foreign key;
3. cascade update the record with the new information and last edit time.

- Soft-delete one-side:
  1. cascade update the record with the new deletion time;
  2. update the dependent records with the new deletion time.
- Soft-update zero/one-side:
  1. insert a copy ofarecord, but with the current time as deletion time;
  2. update the record with the new information and last edit time.
- Soft-delete zero/one-side:
  1. set the deletion time to current time.

In this case, the author also has decided in favor of not saving the state on one-side every time the zero/one-side is changed. Accessing the state of the database in each moment can still be done because the one-side does not contain information about records on zero/one-side.

2.2. Repository pattern

According to Microsoft official .NET documentation, repositories are classes or components that encapsulate the logic required to access data sources. They centralize common data access functionality, providing better maintainability and decoupling the infrastructure or technology used to access databases from the domain model layer.In other words,repositoriesactasalink between business logic layer and data access layer. Theytypically contain CRUD(create, read, update and delete)methods,such as update, addentity, get all entities, and so on.Repository classes often come withrepository interfaces. Thisis very convenient in case of, for example, switching to another data source, or changing certain business logic–repositories help to get rid of duplicate code.This project has only one generic repositoryinterface called IBaseRepository<TEntity>–this interfacecontains general methods needed for business logic layer.Repository pattern basically comes together with Unit of Work pattern. A Unit of Work keeps track of everything what is doneduring a business transaction that can affect the database.This patternallowstosimplifythework with the repositories, because it gives a certain confidence that all the repositories are using the same database context. In thisprojectthere areactually two different Unit of Work interfaces: IBaseUnitOfWork–higher level, saves all the changes to the data source, IAppUnitOfWork–interface that containsmore specific functionality, e.g.repositories needed for this project.There is also a class BaseUnitOfWorkwhich contains factory method for creating repositories.Both the Unit of Workand repositoryinterface implementationsare used in MVC controllers in ASP.NET MVC Application.In author's opinion,this pattern hasproven to be very convenient since theamount of code duplicationhas decreased drastically, and now it is quite easy to changethe waydata is saved or handled, because it is being managedonly in oneplace.There are similar pattern in computer software engineering, such as Data Access Object. DAOis a pattern that provides an abstract interface to some type of database or other persistence mechanism.Both repository and DAO are actually abstraction layers–they store data and they abstract the access to it. But nevertheless, there are some significant differences in the implementation. DAO is an abstraction of data persistence. Repository is an abstraction of a collection of objects.Repository is a higher level concept dealing directly with business/domain objects, while DAO is a lower level, closer to the database/storage dealing only with data–it is often said that DAO is table-centric.The fact that repository and DAO are on the different levelsalso means that Repository pattern could be implemented using DAO's, but the same cannot be said in the opposite.Furthermore, DAO uses Read and Writeconceptwhile repository uses Read Onlyconcept.Therefore, it was decided that project will be built using repositories as link between business logic layer and data access layer, because the Repository pattern suitsproject needs much better.

2.3. Domain → DAL → BLL → API pattern and mapping

During the course the supervisior of this project gave a full demo of making a project using C#, ASP.NET and EF from scratch. There were four layers which all were responsible for their own tasks:

- Domain – layer which describes all the structures used for realization of database for food delivery platform. Based on this layer EF would easily generate a database providing the structures are correctly linked between each other.

- DAL – Data Access Layer – layer which is used for realization of repository pattern and as it is described in its name – for access data from database and creating repositories for better usability.
- BLL – Business Logic Layer – layer which is responsible for all the logic needed for the work of the platform as well as for providing data for Razor page controllers and REST API controllers.
- API – Application Programming Interface – layer which is needed for developing a well designed system which can be further developed easily.

All the layers were designed in the way to work with each other and to provide easy modifying for them as well as re-usability of the layers for other projects. To provide such behavior it was crucial to create automatic factory methods for DAL Repositories and BLL Services as well as automatic mapping of Data Transfer Objects between every layer.

In this chapter, the comparison and choice of databases, platforms, and frameworks willbe performed. The choice will be conducted according to the analysis.

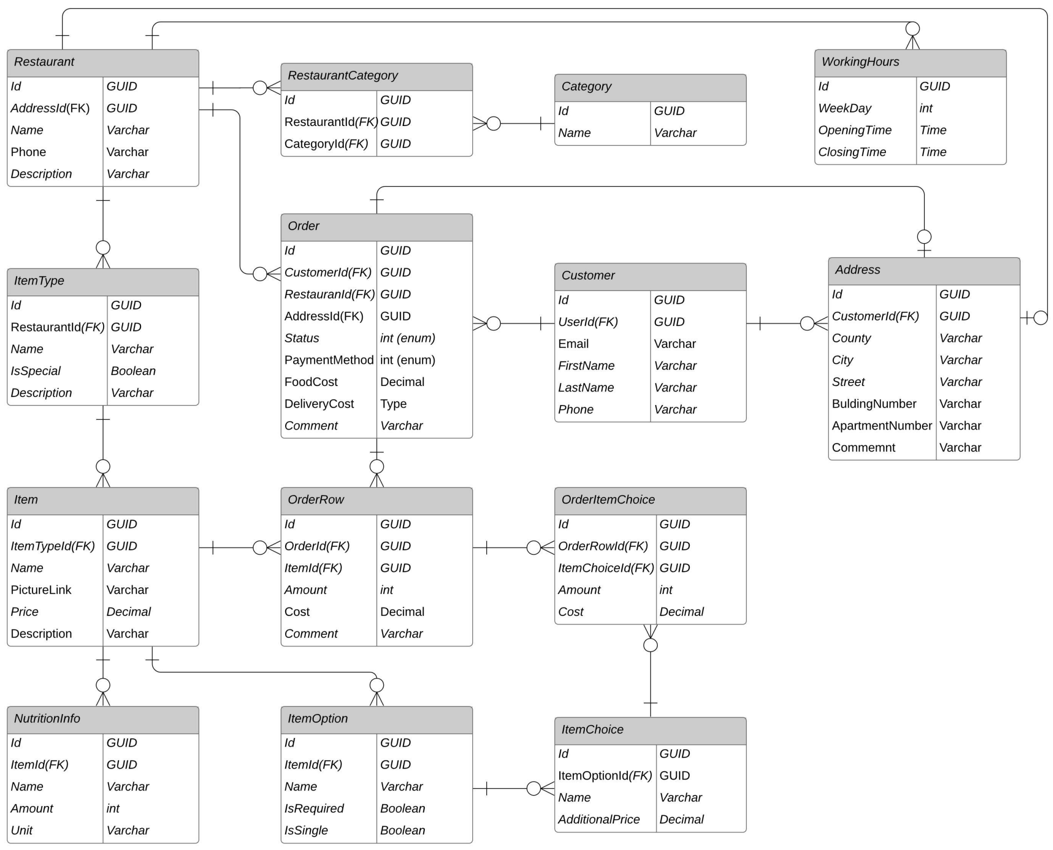
2. Implementation

2.1. Databases

It is important to use a correct database for particular problem. Based on the problems, requirements and possibilities Microsoft SQL(further will be referenced as MSSQL) database was chosen to be used for implementing restaurant system, user system and etc.

The main reasons for using MSSQL were:

1. A server provided by teacher for easy migrating, droping and updating database;
2. MSSQL better compatibility with Entity Framework tools as for example support for automated Guid creation.



The first version of entity-relational model is given below.

2.2. Back-end platform and framework

When choosing the back-end framework we must take into account following important properties:

- The platform must have a large pool of developers;
- The platform must have strong support of MSSQL databases;
- The backend framework must simplify and enhance development velocity;
- The more functionality framework (Authorization and authentication, validating of JWT tokens, etc) provides the better it is.

One of the most popular back-end frameworks in Estonia is Java Spring which suitable for all parameters listed. However as a matter of fact Building Distributed Systems and ASP.NET Web applications courses of TalTech are based on developing skills in using C#, ASP.NET(as well as MVC) and Entity Framework, so the choice of author for back-end was predefined which is not at any terms bad since both of them have similarities.

2.3. Front-end platform and framework

Frontend Framework must comply with following terms:

- Great amount of already premade plugins, components – for rapid application development
- Good real time support
- Typescript and JavaScript support

One of the choices for front-end was predefined by the supervisor – Aurelia is a good front-end framework in terms of compatibility with ASP.NET tools, however it lacks some of the tools for improving the customer interface, so it was decided to use it for restaurant side client interface.

Some of the most popular front-end frameworks like ReactJS, VueJS are widely used and are suitable for all parameters listed. The author chose the VueJS since he became more familiar with during Javascript course.

Conclusion

As a result of Building Distibuted Systems course a food delivery platfrom MVP was developed. The back-end based on C#, ASP.NET and EF and VueJS and Aurelia front-end collaboration was a good decision which turned into a good looking web application with almost seamless user interface. All requirements for the project were fulfilled.

References

1. https://ducmanhphan.github.io/2019-04-28-Repository-pattern/

2. https://stackoverflow.com/questions/8550124/what-is-the-difference-between-dao-and-repository-patterns

3. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

4. https://martinfowler.com/eaaCatalog/unitOfWork.html

5. https://deviq.com/repository-pattern/

Priit Raspel - Andmebaaside alused - https://enos.itcollege.ee/~priit/1.%20Andmebaasid/1.%20Loengumaterjalid/

Appendix 1 – Platform source code

The source code of a developed platform is available at the address

https://gitlab.cs.ttu.ee/arpahh/icd0009-2019s/-/tree/master/