RAW DATA

Portfolio 2 Report

DADIE Marie-Danielle
ŽIVKO Marko

Introduction

This subproject will be one out of three subprojects we have to finish throughout this semester. Final goal for the project is to create a solution which will be a web application that will help users in searching their favorite titles. The application will not just support searching the titles but a much more great functionalities.

In the subproject 1 we designed and created two independent data models combined subsequently into a single database: Movie Data Model which is responsible for storing all the data about titles, and the Framework Model that supports everything regarding the users like: registration of the users, search history, ratings and bookmarks made by the user.

In this part of the project portfolio we have to add a restful web service interface and extend the functionalities previously implemented.

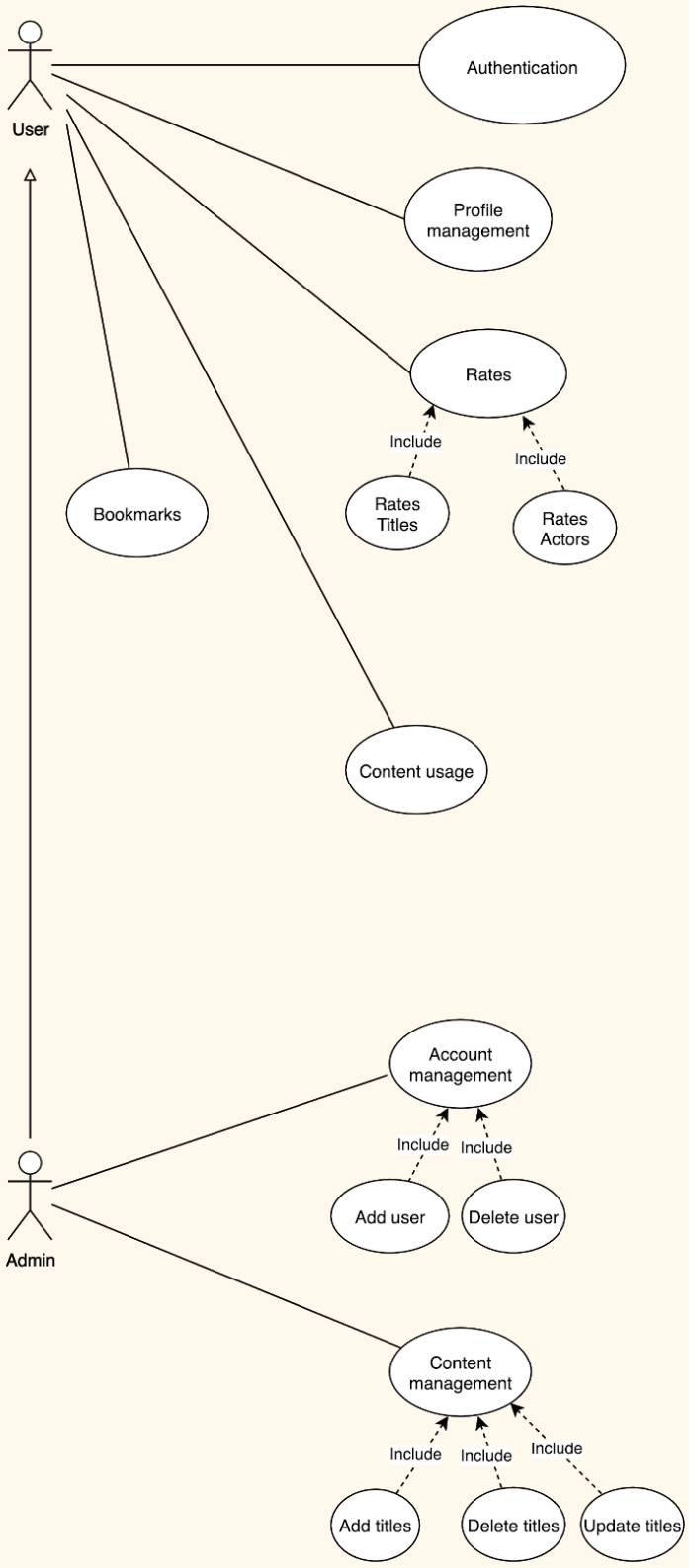
Application design

Architecture design

We started this subproject by revising the design of the portfolio 1. The first point was to supplement the sketch previously made for our E-R diagram. Hitherto, we moulded the entities and relationships taking heed of given tables structure and the real world constraints. The next step here was to define the use cases that the system must support and then extract the overview of these needs to implement the web service. We specified the role played by the actors and represented how they interact with the interface. Then we blueprinted the class diagram. All these UML models helped us convert our conceptual ideas to ready-to-use functionalities.

The class diagram was used to describe the structure of the system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

We used this diagram to show the dependencies within layers and between layers.


User
Authentication
Profile management
Rates
Include
Include
Rates Titles
Rates Actors
Bookmarks
Content usage
Admin
Account management
Include
Include
Add user
Delete user
Content management
Include
Include
Include
Add titles
Delete titles
Update titles

Tools and methods used

At the start of the project team discussed which potential software will be used for both planning and development of the project. We agreed to use the following. Slack is our main platform for communication since most of us have lots of experience using it. Although Slack is a great solution for remote communication we agreed to meet every Tuesday and Thursday to work on the project and of course sometimes during the weekend if there were potential issues. Also video communication with GoogleHangouts and Skype was always a great option. Google Drive and OneDrive are used for sharing the files and working simultaneously on the files. Trello Board is used for assigning and creating tasks for each group member. Also since Slack offers a great functionality to integrate other software into the group channel it was very easy to keep track of all that the team was working on. Although we used multiple applications we could see everything in one place and be notified by each change on our Slack Channel. DrawIO is the tool that we use for creating visual representation of our database like E-R diagrams and Relational Models. GitHub is our tool for version control since all of us use it regularly and it offers a private repository functionality for free. Navicat is a client used for querying and developing our database solution with PostgreSQL. Visual Studio and .NET are used to build our application coded in C#.

Data Access Layer

We used the data access layer to access data stored in the database. In order to map the DB relational model into the object-oriented model, we used the structure of the entities and their relationships:

- Each entity was represented by a class and each property was set as an attribute of the class.

- Relationships between classes were performed by using the entity framework. Considering a many-to-many relationship, we used a class to represent the relationship. This class contains attributes corresponding to the primary keys of the two classes that create that relationship. We also have to include a collection in each of the entities as a reference of the relationship class.
- To configure many to many relationships we followed the rules we mentioned before. For one to one relationships and one to many relationships the process was similar and we had to follow the logic and check if we needed to include collection or simple objects.
- We noticed that for some entities we did not need double navigation property. The entity for this example can be Genre. The reason was because when we included the collection in the class the collection represented the actual attribute of this newly created class.
- We introduced context class to query and save data to the database. Also, we used it to configure domain classes that were previously created. Here we also did all the work regarding the mappings to the database.
- Data service was created in order to define the interaction inside the whole system.

Some of the implemented services:

- **GetUser(int id):**
  - returns a specified user by the given id
- **IList<User> GetUsers()**
  - returns the list of all users
- **IList<SearchHistory> GetSearchHistoryForUser(int id)**
  - return the search history for specified user
- **UserRole CheckUserRole(int userid)**
  - checks the user role
- **IList<UserRates> GetUserRatings(int id)**
  - returns the list of all user ratings
- **UserRates GetUserSpecificRating(int id, string title)**
  - returns rating for specific user and specific title that he/she rated
- **TitleRating GetTitleRating(string title)**
  - return a rating for specific title
- **IList<TitleGenre> GetTitleGenres (string title)**
  - returns a list of genres for specific title
- **IList<Genre> GetAllGenres()**
  - return the list of all available genres in the database

- **IList<TitleBookmark> GetTitleBookmarkForUser(int id)**
  - return a list of bookmarked titles for specific user
- **IList<Personalities> GetPersonalitiesForUser(int id)**
  - return a list of saved personalities for certain user
- **IList<TitlePrincipal> GetTitlePrincipals(string title)**
  - list all the principals for specific title
- **IList<KnownFor> GetKnownTitleForPersons(string person)**
  - list the known titles for specific actor or actress
- **IList<TitleGenre> GetTitleByGenre(int idgenre)**
  - list all the titles for specific genre
- **IList<Episode> GetEpisodesBySeason(string serieid, int season)**
  - list all episodes for specific series and season in that series

Web Service Layer

Web service layer represents the interface to the backend. We tried to expose many methods that can be used in the fronted. We had to create new data models using data transfer objects to remove unnecessary information.

In this part we also introduced the simple authentication for the user. Some web services require the user to be identified in order to get certain information. For example users can create personalized bookmarks for titles and each user is able to check only his/her bookmarks. Also, users have to be identified to check the list of the titles or do many more things.

For requests that generate a lot of data and return multiple objects we introduced paging. Users can specify how many objects per page to display and also how many pages to create. Users are able to go back and forth in between the pages itself which are very useful for navigation.

We tried to respect rules for creating REST routes and methods.

Portfolio1 - changes

Table address had to be changed because of the mistake we created in the first part. The mistake we created was that when we tried to represent one to many relationship we mistakenly put iduser as foreign key in the Address and the correct solution was to put idaddress in the User table.

Because of this change we had to update functions that used foreign keys in these tables.

Another important change we did was regarding representation of series and episodes. We completely removed both the series and movies tables. This caused us to have duplicate rows because postgres inheritance feature has some limitations. We solved this by creating a recursive relationship between title basics and episodes. Episodes is a title but a title can also be an episode of another title.

Testing

We created a test project inside our solution but unfortunately we did not have enough time to create it properly. But, all the services have been tested in the Program.cs when DataLayer is called in the console application.

Each WebService path and methods were tested with Postman.

Conclusion

The Portfolio2 project was very extensive with lots of features that had to be implemented. Since there were only two of us we did not have enough time to implement all the services that we wanted. Some of the functionalities that we created in the database and the data service were not used in the web service just because of lack of time.

Hashing of the user password will be implemented in the last part of the project because we are aware of security issues that can happen.

Link

GitHub repo: https://github.com/madadadie/Portfolio2.git