UNIVERSITY OF STAVANGER, NORWAY

DAT240 Project Report (HansFriBerPedHaal): Image Guessing Game

Lea Madelen Hals Hansen, Berit Holen Friisø, Andreas Pedersen, Marius Hauge Håland, Stig Bergene

November 30, 2023

Abstract

Abstract of the game

This report gives a brief overview of how we chose to solve the Image Guessing Game using ASP.NET. The Image Guessing Game is a game with a purpose (GWAP). The Image Guessing Game consists of a guesser and a proposer, where the proposer gives the guesser fragments of an image and the guesser will guess what the image label is based on shown fragments. The goal of the game is to give the correct answer as fast as possible, meaning fewest possible guesses. The features of the game are: singleplayer and twoplayer; respectively with a random oracle and with a proposer. You can also upload your own images to play with.

The game also supports a leaderboard that tracks high scores and fosters a competitive environment. Additionally, players can review their past games, reflecting on their progress and strategies.

When designing the Image Guessing Game we tried to have scalability in mind. The application's architecture is intended to support easy integration of additional features, such as replacing the Oracle's implementation with advanced AI models. This is achieved through using a blend of technologies that adhere to modern software architecture principles, including Domain-Driven Design (DDD) and Command Query Responsibility Segregation (CQRS) using pipelines. We have also used other principles like "Separation of Concerns", incorporating business logic that is clearly separated from the infrastructure logic and UI logic, and writing loosely coupled code.

CONTENTS

1 Introduction

1.1 Organizing the Work

We mainly divided the work based on what people were interested in working with. We worked a lot together using pair-programming with live-sharing our screens for specific tasks that we thought were too complex to solve for one person, and better to solve as a pair. In the beginning it wasn't that easy to split the tasks into smaller tasks, and we weren't sure how to start, but as the code evolved, tasks were easier to split up and assigning tasks became easier.

1.1.1 Tracking the Work

We mainly tracked our work by communicating with each other face to face, as we mostly sat together and worked with the project which made it easy to communicate on what we were working on. Other than that we frequently used issues on GitHub for all the tasks that needed to be done. This made it easy to either assign a task to yourself or assign them to any other teammate. Issues made it easy to see what had been done, and what needed to be done.

1.1.2 Communication

Communication in our group went smoothly. Our group communicated thoroughly on what each of us were working with. We had a Messenger, Discord and Snapchat group where we communicated on when we were meeting up, what we were working with and shared progresses if some of us were working at home.

1.2 Evaluation During the Project

We did a continuous stream of evaluation during the project. Before incorporating new features, we discussed our plan for the feature and how we believed it would be best to implement it, taking into account all the features we already had. Following the implementation of a feature, we reassessed whether it was the optimal approach and made adjustments if deemed necessary.

Near the end of the project we took a final evaluation of the project and decided we needed to change a part of our architecture. This change will be discussed in subsection 3.3.1-Services.

1.3 Docker

We are using Docker to minimize the effort to set up and run the application. A detailed guide to starting the application can be found by viewing the README file in the Github repository. We encountered a RAM problem with using the docker with all the images since there were so many, so we had to remove some images so it would run without taking up all the RAM.

2 Design and Architecture

Person in Charge; all

We dedicated a considerable amount of time at the beginning of the project on creating the structure of the whole application; deciding what should be the domain models and how the domains would communicate with each other. This was mainly to make it easier for us all to have a good understanding on how the game should be set up. Looking back at it, doing this made it is easier for us to work simultaneously.

To maintain clear separation of functionality, we divided our domain into four distinct parts. The image domain is separated from the game domain, since storing and serving images is not related to the game logic. The user domain is also separated from the rest, which makes it easy to change or replace. Communicating between the domains happens via pipelines, services, DTOs, events, and handlers, as shown in the block diagram 2.1.

We chose to make a flow chart, as shown in figure 2.2, because it clearly shows the flow of the game from the moment you press the play button.

2.1 Block Diagram

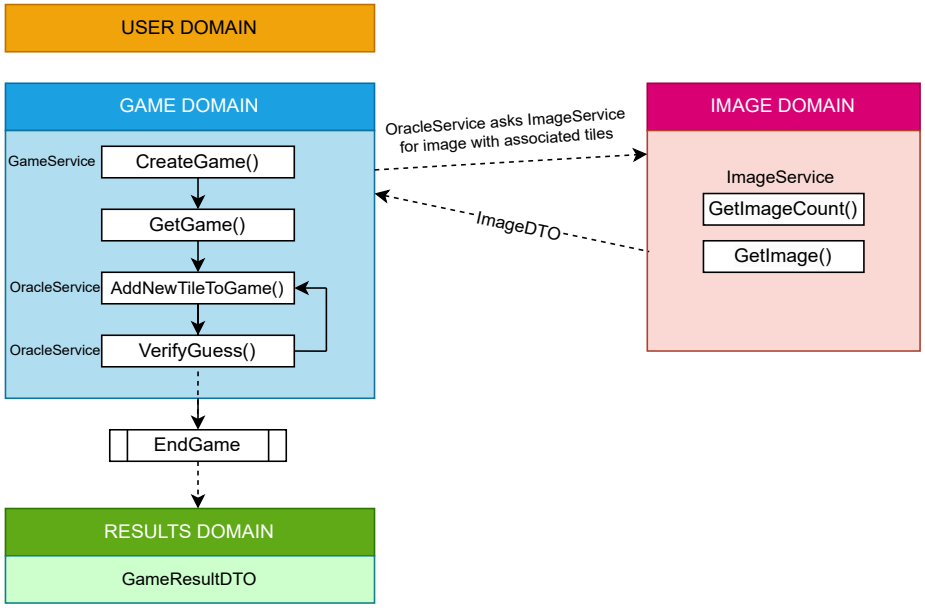

Figure 2.1: Block Diagram

2.2 Flow Chart

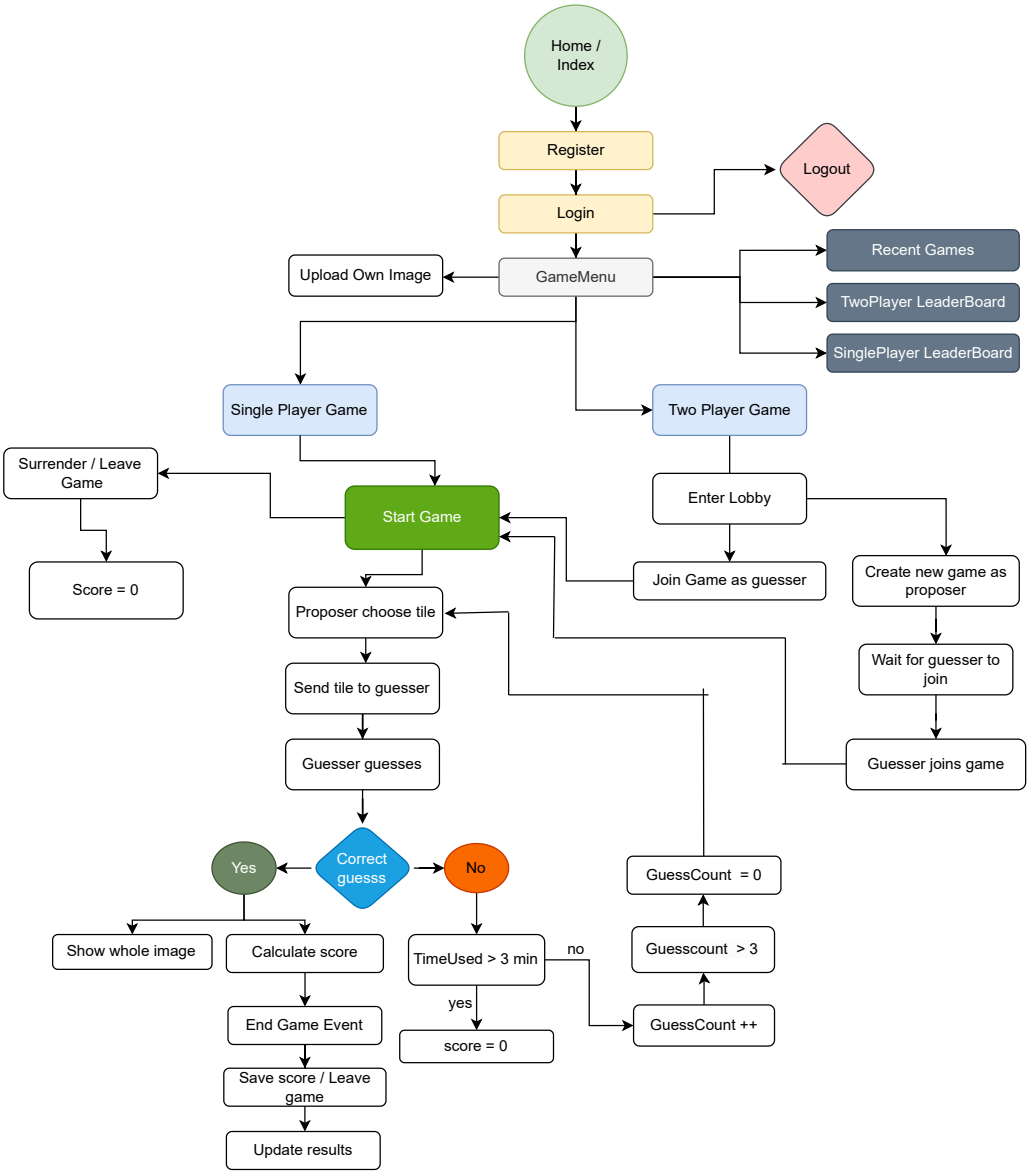

Figure 2.2: Flow chart

3 Backend

Person in Charge; Marius, Andreas, Berit, Lea

3.1 Frameworks and Packages

3.1.1 ASP.NET Core

We decided to do this project using ASP.NET Core because everyone in the group were already familiar with it, as we all have experience with C# and ASP.NET from the labs, but we also think ASP.NET Core has a nice framework for building web apps.

3.1.2 EF Core (Database)

For the database, we opted to use the SQLite database engine together with Entity Framework Core as we were familiar with them from the labs.

3.1.3 MediatR

We used the mediator pattern for promoting a maintainable and scalable code, through loosely-coupled design and easy implementation of cross-cutting concerns.

3.1.4 SignalR

We used SignalR to add real-time web functionality to the game application, so a two-player game comes alive with real-time interactions between the two players. We decided to implement this to create a better user experience among two players. Each game forms its own group, and players connect to this group. This setup allows for multiple games simultaneously, as well as the possibility to expand the application with multiplayer in the future.

3.1.5 Other

We used .NET Imagesharp for image processing, which we found valuable for tasks like our "Upload your own image" feature. We used Xunit to create unit tests.

3.2 Testing

Testing has not been implemented to the degree we wanted it to be, due to lack of time. Towards the end we realized the importance of beginning with testing early on. Reason being that we found small bugs in our code while testing, that we might have not discovered without testing. Regardless of running out of time at the end, we believe we ended up with testing the features that we thought were the most important to test. We ended up with having both unit

tests, integration tests and and an end-to-end test. That being said, if we had more time, we would have created more tests.

3.3 Changes during the project

3.3.1 Services

At the start of the project, when deciding on the architecture, we struggled a bit with the domain model as we were unsure of how many services we should implement. We first designed an application that had both a GameService for game-related functioning, and an OracleService for oracle-related tasks. We initially ended up only implementing the GameService. However, near the deadline we realized that this was not scalable, taking into consideration that it was supposed to be easy to replace with an improved AI model. Therefore, we decided to implement an OracleService and an ImageService, in addition to the GameService, as shown in figure 2.1.

3.4 Experiences

We found it challenging naming classes and files in parts of our code. Given our domain model is not large, names like add/get - tile, game and image were used frequently, as they hold a short and fitting description of what we were working with. This posed a challenge when working with libraries like ImageSharp, as our domain included a class named "Image", coincidentally clashing with the built-in "Image" class used by ImageSharp.

Currently, Proposer and Guesser are distinct attributes in the Game class. Given more time, a more scalable approach would involve changing this structure to a list of players. This modification would have made it easier to incorporate multiplayer as well.

We opted to save the fragments in the database as Base64. This has several advantages, such as it is compatible with any database that can store string. In addition, it is simple and easy to transfer between different systems. However, storing them as Base64 also has its disadvantages. The storage size is greatly increased, as well as the search and indexing time. For future projects we would probably research other possibilities.

During the development of the project we gained more experience with writing readable code, and more knowledge about coding principles. Due to the limited time available we did not manage to apply this to the entire project. An example of this can however be observed in the pages file SinglePlayerGuesser.cs, where we utilize functions to make the code more readable.

4 Frontend

Person in Charge; Lea, Andreas, Berit, Stig

Initially, we were not going to prioritize spending much time on the frontend, but as we progressed we dedicated some time to it, as it made it easier for us to test our features.

Towards the end of the project, we made the choice to move the internal JavaScript and CSS code to external files to improve code readability and make maintenance easier.

4.1 Frameworks and Languages

4.1.1 JavaScript

We opted for JavaScript as this is a language we are familiar with. JavaScript was mainly used to make alerts, create interactive elements in the DOM, create a smoother page-flow and overall a better user experience.

4.1.2 Bootstrap and CSS

We used Bootstrap as it is already included in ASP.NET, but also because it is easy, fast and very responsive to work with. For the parts of the application where we wanted a nicer layout, or some specific styling, we used CSS.

4.2 Experiences

Our frontend is based on CRUD principles, using HTTP POST and GET methods to retrieve data from the backend. We decided to use HTTP methods primarily because it is the approach we are most familiar with from other subjects in our degree. If we had more time we would also have followed the REST API principle, together with CRUD principle, as this would have created a smoother and more user-friendly application.

We struggled a bit with implementing the user defined sequence for choosing a better tile order for an image for single player games. We managed to implement it at last, but it is not optimized the way we would like it to be. The user improved tile order is now saved in a list so you can use it for future games, but if you get the same image twice, it wont replace the already added tile to the list, it just adds it again, which is not fully optimized.

We also had a "problem" with our use of Razor Pages and the use of POST method where we had to retrieve the game in every HTTP Post method to be able to continuously get/have the game. If we used for example Vue Router, we would not have had that issue, but we figured out that we did not have enough experience with Vue to use it in the application, so we went with using razor pages.

5 Summary and Contributions

5.1 Summary of the project

As a group, we believe that we managed to complete the project satisfactorily, despite there being some significant changes towards the end. We understand now that projects inevitably come with changes and challenges, regardless of how well-prepared you are.

5.2 What we have learned

This project has been a great learning experience for all of us. We learned a lot academically. For instance, most of us have never built a project of larger scale from scratch before, and we gained a lot of knowledge and understanding working with Domain Driven Design. We also learned the importance of creating clear and descriptive class-, variable-, and filenames, etc, from the beginning, as changing them may lead to a considerable amount of time spent on unnecessary work.

Further, we also gained valuable team working skills. Breaking down the work into smaller tasks, and working together when needed, made the workload manageable. Above all, we learned the importance of clear and effective communication.

5.3 Contributions

Task	Responsible Person(s)
User Domain	Lea
Game Domain	Marius, Andreas
Image Domain	Marius
Results Domain	Berit, Marius, Stig
Services	Marius, Lea
Single Player	Marius, Berit, Andreas
Two Player	Marius, Berit
Upload Images	Berit
Load Images to Database	Marius, Andreas
SignalR	Berit
Timer for Game	Andreas
Leaderboard	Andreas, Berit, Marius, Stig
Tests	Berit, Lea, Marius