UNIVERSITY OF STAVANGER, NORWAY

DAT240 Project Report (Group F): Guessing Game

Vidar André Bø, Liv Selle Underhaug,
Genette Klyve Våge & Mina Cecilie Wøien

November 26, 2021

Introduction

This report gives a brief overview on how we chose to solve the Image guessing game programming project with ASP.NET and Vue3. That is, the groups interpretation of the project, how we assessed the scope of the work, the decisions that were made, our workflow and what was learned.

In short, the Guessing Game is a game with a proposer and a guesser. The proposer will give the guesser parts of a image and the guesser will guess what the parts displays. The goal is to give the correct answer as fast as possible, meaning fewest possible guesses. The features of the game are: singleplayer, twoplayer, multiplayer with a proposer and with an oracle.

First we introduce architectural and technical decisions that were made and how we chose to implement the game. This is visualized with two diagrams. One flowchart that explains the game logic, the flow of the game when you enter as a player. The second, a block diagram, that displays the pipelines within the individual domains and how the different domains are connected through events and handlers.

The second chapter introduces the backend API part of the game. Which framework was used and why. It also describes our assessment of our work, the changes that was made and why.

The third chapter presents the frontend aspects of our game and the last part of the report gives insight in the project process. How the project was organized with Github, the collaboration and communication, and what obstacles we had to overcome.

CONTENTS

1 Architecture

1.1 Architectural decisions

The application is divided into a client and an API. This is a structure that is known to the group and made it is easier to work simultaneously.

To avoid large objects and to have distinct separation of concerns, meaning to put functionality were it belongs, the game consists of four main domains. The image domain is separated from the game domain, since serving and storing images is not related to game logic. The same goes for the auth domain, when separated from the rest, it is easy to replace, and it is not needed as a part of the game.

The game domain quickly became to large, and thus it was decided to divide it into two parts. Pregame, logic related to matchmaking, and game, logic related to actual playing. The only part connecting the different domains are the events and handlers displayed in the block diagram.

The flow chart, as introduced, show the flow of the game from the moment you enter the game from the user interface.

1.2 Block Diagram

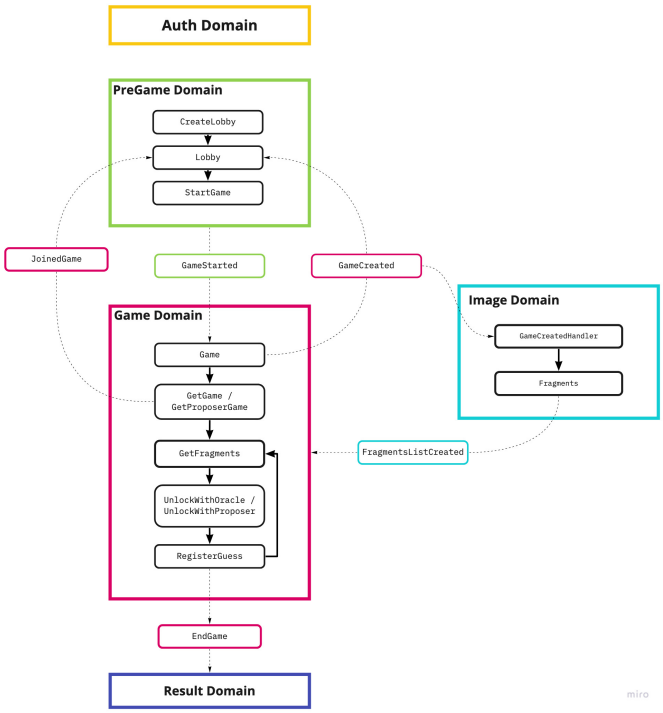

Figure 1.1: Overview.

1.3 FLOW CHART

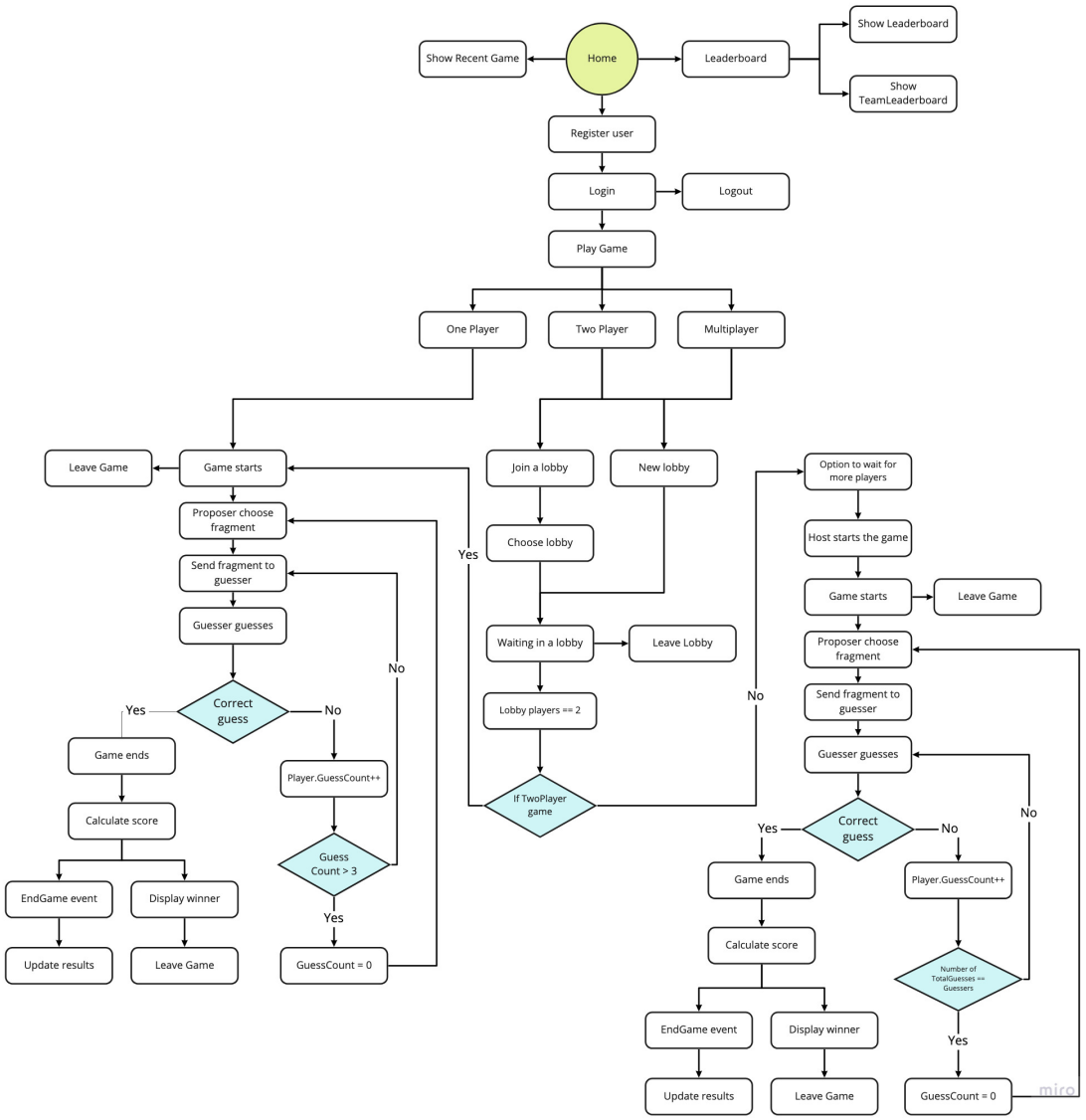

Figure 1.2: FlowChart

2 BACK-END

Person in Charge; Mina, Liv & Vidar

2.1 FRAMEWORKS AND PACKAGES

2.1.1 EF CORE

EF Core is used as the object relational mapper in the API. It is used because it simplifies the interaction between the application and the database. The group is familiar with it, and therefore the obvious choice.

2.1.2 MEDIATR

MediatR was used to obtain domain driven design. As it works as a "message broker" it separates the domains and make them independent. The domains has no access on each other's code and it is decoupled. When transferring data between domains, one creates an event in the current domain and it is picked up by a handler in another domain. It also separates the controllers from the domains and the database.

2.1.3 OTHER

The back-end API is an ASP.NET Core web-api
Xunit is used as the testing framwork for the API.
ImageSharp is used to read pixeldata from an imagefile.

2.2 TESTING

Testing has not been implemented to the degree that we initially wanted. As the code evolved and changed for the nth time, we struggled with making durable tests. Essentially we have implemented tests for our pipelines, ie. integration tests.

The test runs individually, but not as a whole, because of how they are setup. If we had had more time, we would made tests based on the arrange, act and assert-principle. Arrange the environment and the parameters needed, act - do the action that gives expected results and assert - validate the expected result.

The group has little experience with making tests, and when the project began a concrete decision to write test along with the code was not made. In hindsight we might have spent less time on debugging if we had implemented it properly from the start.

2.3 Changes during the project

As described in architectural decisions we aimed to divide the image and game domain. The initial plan was that image would live by its own. We needed access control for the images. In a game, we had to know which fragments that were unlocked. Therefore this information is stored per game and images themselves are stored in the image domain.

Another change that was made to improve reliability, was to let the server find out which game the user was a part of, rather than having the client ask for a specific game. This made it so that we could refresh the page and be sent right into the game again. This got implemented for the lobby as well.

Closing in on the end of the project, it was decided to drop some features in the leaderboard due to time constraints.

2.4 Experiences

Creating the game domain were the most time consuming part of the project. It was decided at the early stages of development that we were going to implement single player, two player and multiplayer. These features were developed simultaneously, which made it quite time consuming and complex. The logic for guessing was revised several times to make it work with single player, two player and multiplayer.

3 FRONT-END

Person in Charge; Genette & Liv

3.1 FRAMEWORKS

3.1.1 TYPESCRIPT

The group decided to use Typescript to write the front-end application. Typescript was something the group had to get acquainted with, but it was chosen to allow for easier detection of bugs. The benefit of using TS in this project was to have more structure in the transition from the JSON-object received from the API to the objects in the front-end application. When fetching data from the API, it is always specified which format the response will have. The front-end and back-end data types can be easily compared when writing the code, so that bugs when accessing non-existing fields is almost eliminated.

3.1.2 VUE3

It was decided to use Vue3 as the front-end framework. This decision was made in the start of the project because of all group members experience with this JS framework. With this framework we have easily implemented functionality and utilizing the easy ability to update the DOM. The front-end is a SPA, that allows us to update the data without refreshing the whole page. We used Vue mainly to create a good user experience.

3.1.3 BOOTSTRAP

For the CSS Framework, Bootstrap was used. With Bootstrap we could easily integrate a better user experience without spending to much time.

3.2 CLIENT SIDE REFRESHES

The front-end application uses polling on set interval to get updates from the API. When the front-end sees a change in the game-status, for example that an image fragment has been unlocked, the application will call the route for fetching images. The main reason for using polling instead of WebSockets was to reduce complexity and time consumption. The overhead that comes with polling the API was deemed acceptable due to the low performance requirements of the application.

4 Summary

4.1 Summary of the project

To utilize every group member's best abilities, it was decided to split up the work mainly in front-end and back-end. Four out of five group members knew each other from the start. We came to a quick agreement that working together at school, were the best way to collaborate. The fifth member wanted to work from home. It was agreed upon that as long as the communication were explicit and good, that was okay.

Unfortunately this did not work at all, as the communication were lacking and he was difficult to reach. It became a challenge for the whole group as work were progressing and tasks that were promised to be finished within a certain time, were not completed. Mid-project the fifth student decided to drop the course. As a result of the poor communication and loosing one participant, the remaining members had to rewrite a large amount of the work the participant left behind.

The workload of the back-end part of the game became significantly larger compared to the front-end than we first assumed. We decided to reallocated more resources to work with the back-end. In hindsight we think that the distribution of work could have been better. Some of the team-members worked with all parts of the code, and therefore got a good understanding of what needed to be done. For the others, it was easy to fall behind. We learned that we could have allocated the work differently and made every member take part in both front-end and back-end coding, so that the workload would not become so unevenly distributed. In addition, workflow could have been better if every member of the team took part in both front-end and back-end. There were several times that the workflow stopped due to bottlenecks.

To summarize the project process, we all feel that the workload has been quite heavy. On average we individually spent more than 30 hours per week during this four week project. A lot of the time working, have been spent together, all four of us, or two and two. As a result we have communicated a lot. Some frustrations, helping each other, making decisions and solving problems with talking about solutions.

4.2 Organizing the work

GitHub and a Discord Server were the main platforms to organize the work. The discord server were created first thing after the group was published so that a meeting at school could be planned. We used discord for planning meet-ups and asking short questions in channels based on topics. Discord was the main platform to ask questions and get help except from talking at school.

GitHub was used for the general project workflow. A project called "GuessingGame" was made and was equipped with four columns to keep track of the issues. At our first meeting we brainstormed initial tasks based on the project description. There were created issues continuous throughout the project as we came up with tasks needed to be done or bugs needed

to be solved. Every issue got assigned to a group member, and were labeled with tags related to its task.

To organize our code, we made use of branches in git. The goal was to create a new branch for every issues. To make sure that the master were always building, the pull requests (linked to its issues) had to be approved by another person in the group. To look and approve others' code made sure we stayed updated on the code and the progress.

After a large merge conflict the group agreed to make some changes on how large the issues should be. The issues were split into several and smaller issues. The goal was that a person should manage to finish an issue within at least a day. If that was not possible, then issues had to be split into even smaller issues. With this type of workflow the pull request became smaller and the branch lifetime shorter. In practise this led to affordable pull requests that were easier to merge in to master and with less conflicts.

4.2.1 What have we learned?

This is the first time any of the participants in the group have created a large project like this big project from scratch. It resulted in a longer startup period than what was desired. The upside with building a project start to finish, is the knowledge and understanding you gain from working with domain driven design. Working in a group with individual skill-sets and learning how to work together and communicate is a good experience to bring into the future as well.