Project Report

Team #27

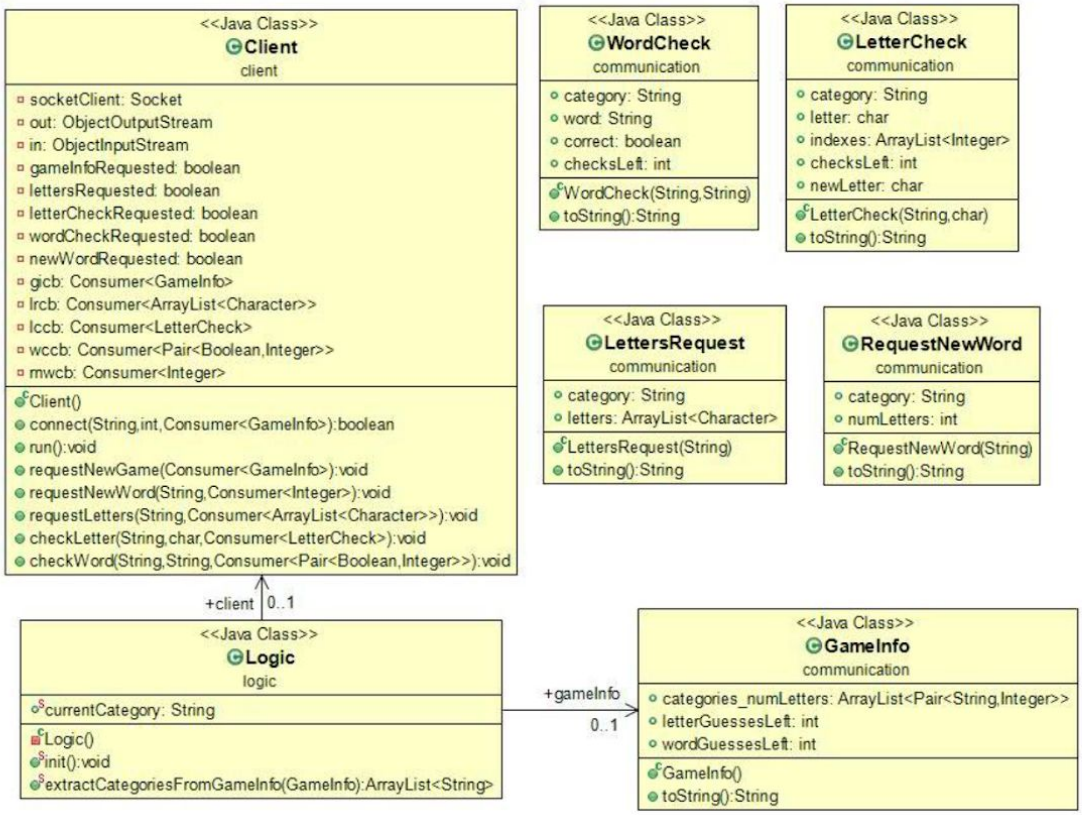
Zain Sadiq

Viktor Kirillov

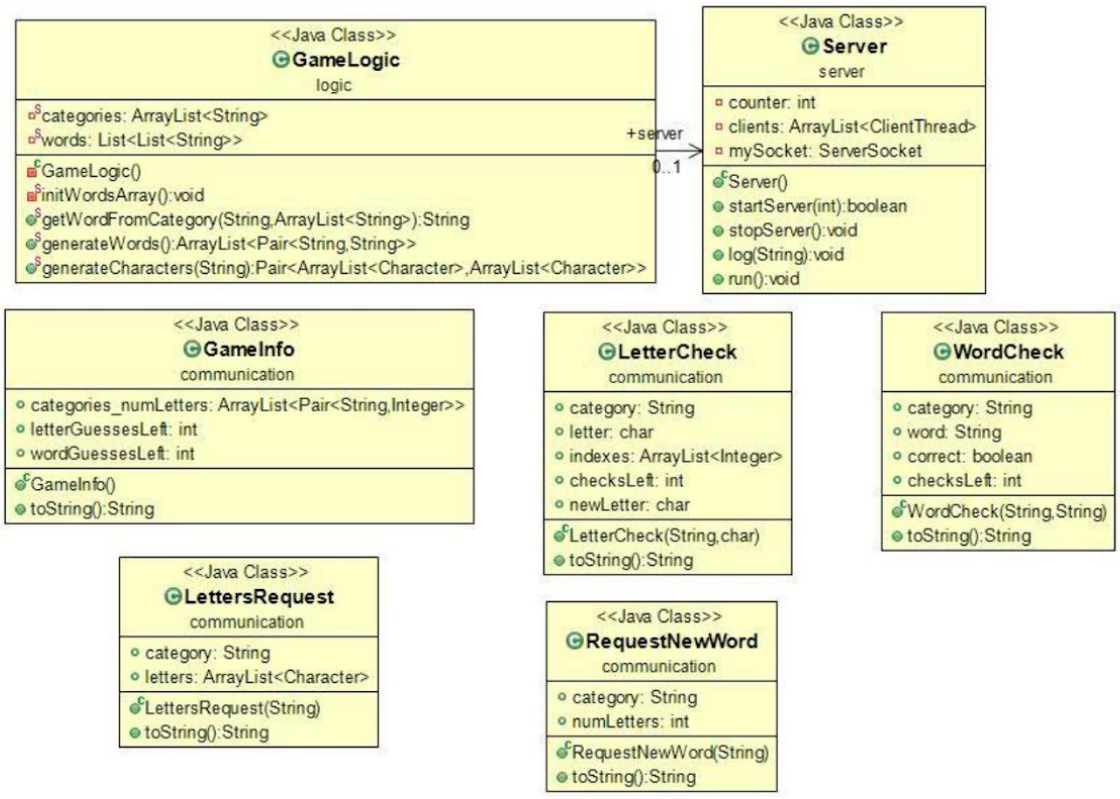
Mykyta Parovyi

Volodymyr Vakhniuk

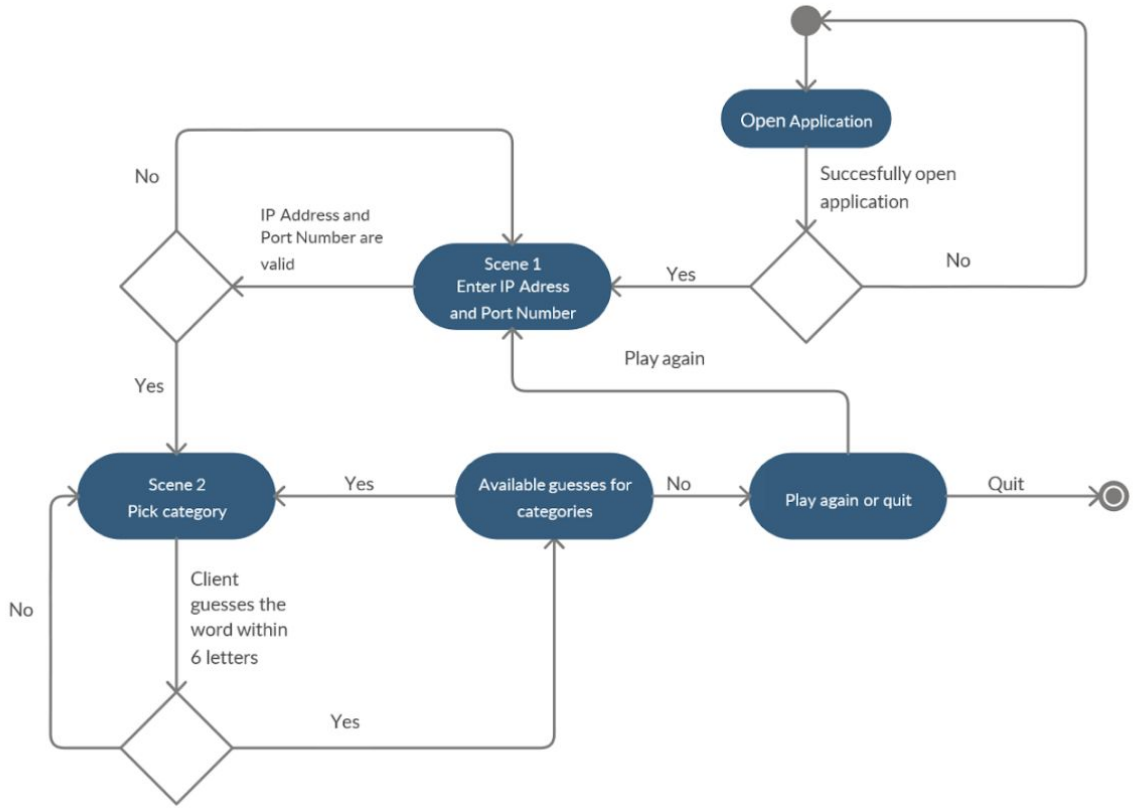
Client UML Class Diagram



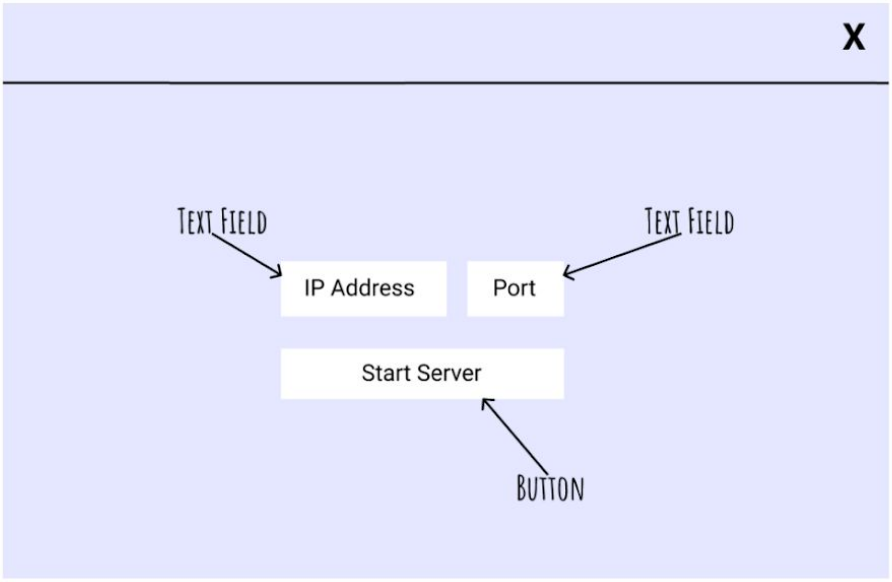
Server UML Class Diagram

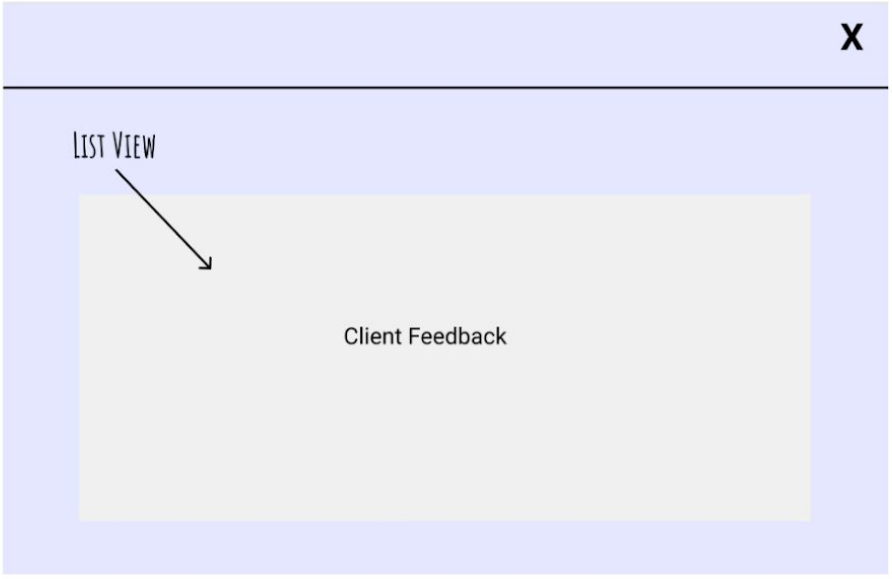


Client Activity Diagram

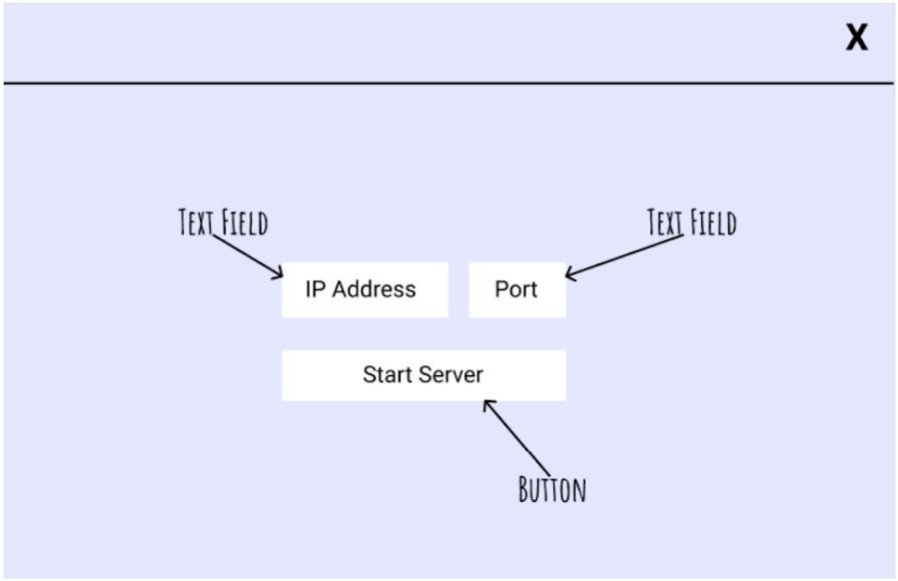


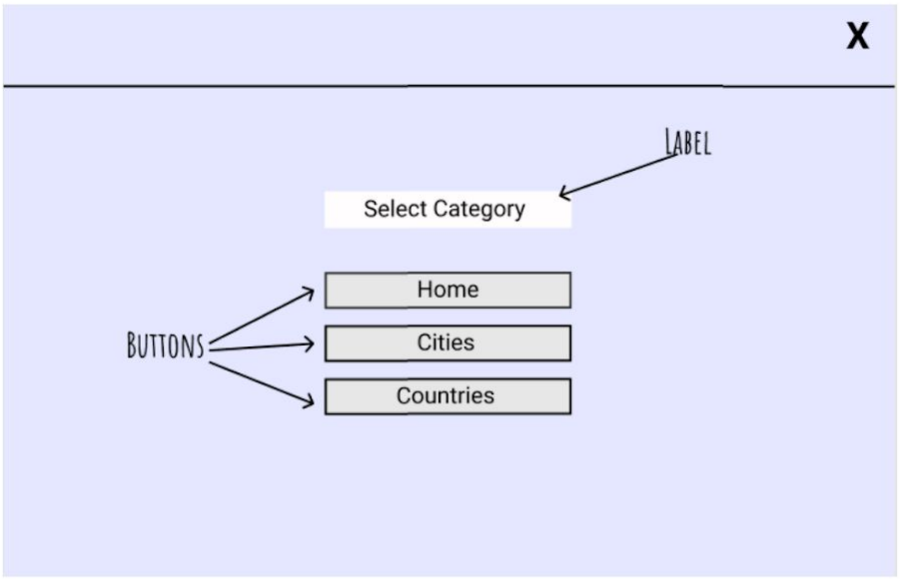
Server Wireframe

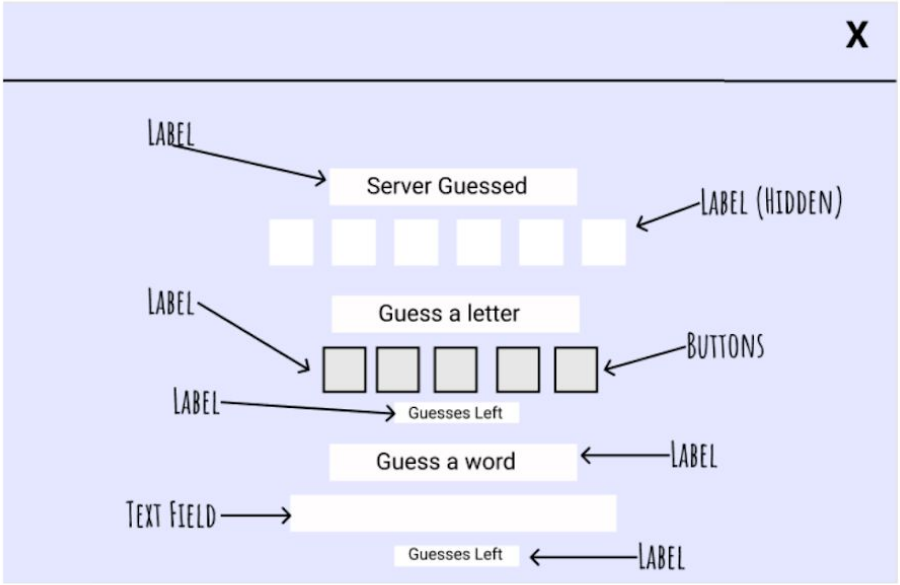




Client Wireframe






X
LABEL
Server Guessed
LABEL (HIDDEN)
LABEL
Guess a letter
BUTTONS
LABEL
Guesses Left
Guess a word
LABEL
TEXT FIELD
Guesses Left
LABEL

General description of the server and client logic.

The whole server logic was fit into a single class with 3 functions.

generateWords() creates a list of pairs <category, word>, where words are randomly chosen from the predefined list. The function is used when the client connects to the server, the last one responds with the list of categories generated for a client.

generateCharacters(word) creates a pair <chars[], chars[]>, where the first array consists of 3 letters randomly chosen from the supplied word + 3 letters not in the word, and the second array holds characters all the remaining characters from the alphabet. The first array is being sent to the player when it choses a category. The client is able to use his letter guesses by checking if one of those letters is present in the word. After checking the letter, the server sends the character from the second array, therefore the player always is able to guess some of the characters.

getWordFromCategory(category, blacklist[]) returns a new word from the category given. When the player was not able to guess the word in the category within given attempts, the function supplies the client with the new word in that same category.

The server keeps track of each player and communicates to their client programs using the predefined classes. For each request there is a class, which handles that specific request. To fit all of them in our game flow, on the clients' part, for example, when players choose a category, we are sending LettersRequest to get the number of letters in the word and an array of the characters displayed for the player. Along with the request, the client is being supplied with the response handler, which is being called when we receive the data from the server and then continuing the program execution as we defined.

Development of the UI and UX of the project.

In this project we took into account all the problems we encountered in previous assignments. After discussing all of them as a team, we decided to use a Singleton class to keep all of the UI parts in one place and remain applications' start method clean. Static UIStatic class has a reference to the application stage and all the scenes available in the game. It provides an API for changing scenes as well as methods for the layout and image processing.

While the Server UI is pretty simple and straightforward, the Clients' part has twice as many scenes and complicated User Interface. The GameScene is a class where all the “magic” happens. We tried to find a balance between the impressive visual part of the game with all the animations applied and user experience, and looked like we were able to do that. The game looks interactive and dynamic while it still remains intuitively understandable for the player on what actions we are waiting for from him and how the game responds to the actions that were made.

All visual parts of the game were built using functional programming. Each animation is implemented in a way that it has an onFinish handler. Utilizing that approach we were able to nest all the animation flow in a sequence of actions, which lets us create complicated interaction mechanisms.

Keeping the game component independent from each other we managed to create a bunch of GUI elements which are easily reusable and interchangeable. That saved us a lot of time in creating servers’ graphical interface. When we finished the client UI, we utilized the exact same components on the server part and made the whole interface in minutes.

Team Work.

All the work was evenly divided between all the team members and each of us was responsible for specific components in the game. We built a plan for the project and optimized it before we started the actual coding. Therefore we managed to finish everything we planned and the game works as expected.

For collaboration we were using the Zoom application. Every 3-4 days we had “meetings” where we discussed what worked and what didn’t and were finding the ways on resolving the problems we were facing.

Viktor. I was responsible for creating the general structure of the project. I utilized my experience of planning that I gained in previous projects. Besides planning and guiding the work I implemented the client’s GUI and UX leaving the points for connecting it to the clients’ logic part that was created by Volodymyr.

Zain. I created a server logic class named GameLogic and the actual Server class which handles all the clients in a separate thread. The GameLogic is a static class that provides useful functions for managing

words and categories. Also I was responsible for creating UML Class Diagram and Wireframes.

Mykyta. I designed a communication mechanism between the client and server and implemented it in both of them utilizing a pre-built structure that was created by Zain and Volodymyr in their server and client instances. Besides that I did the Activity Diagram and I was helping with the report.

Volodymyr. I did a server GUI utilizing the objects created by Viktor. I created a client class that communicates to the server and helped with tying the logic and visual part in both client and server applications.