

Milestone 4

Group 19

Aqeel Mozumder, Adam Kwan, Andrew Polanyi

Table of Contents

| | |
|-----------------------------------|-----------|
| 1. Introduction | 3 |
| 1.1 Game Description/Rules | 3 |
| 1.2 Source Code | 4 |
| 1.3 How to Build | 4 |
| 2. Beyond Views | 6 |
| 2.1 Documentation Roadmap | 6 |
| 2.2 How a view is documented | 8 |
| 2.3 System Overview | 9 |
| 2.4 Mapping Between Views | 9 |
| 2.5 Rationale | 10 |
| 2.6 Glossary | 10 |
| 3. Views | 11 |
| 3.1 Module View | 11 |
| 3.1.1 Primary Presentation | 11 |
| 3.1.2 Element Catalog | 12 |
| 3.1.3 Context Diagram | 14 |
| 3.1.4 Variability Guide | 14 |
| 3.1.5 Rationale | 15 |
| 3.2 Component and Connector View | 15 |
| 3.2.1 Primary Presentation | 15 |
| 3.2.2 Element Catalog | 16 |
| 3.2.3 Context Diagram | 19 |
| 3.2.4 Variability Guide | 19 |
| 3.2.5 Rationale | 20 |
| 3.3 Allocation View | 20 |
| 3.3.1 Primary Presentation | 20 |
| 3.3.2 Element Catalog | 22 |
| 3.3.3 Context Diagram | 23 |
| 3.3.4 Variability Guide | 23 |
| 3.3.5 Rationale | 24 |
| 4. References | 25 |

1. Introduction

1.1 Game Description/Rules

Our Game consists of three rounds, each with an increasing amount of points that you can earn. The objective is to be the player with the most points at the end of the 3 rounds. Each player plays the game through their browser where they can join a game using a 6 character game code that the host can give out.

Round 1:

Once the host starts the game players are put into round 1 where they are given a question prompt to answer. All players are given the same prompt so they will want to make their answer funny or serious depending on what they think other players will vote for. After all answers are submitted, each player will get to vote for one of the other players answers to the question. When your own answer gets a vote you get rewarded 100 points per vote. After the voting is done, scores are displayed for a brief period of time then the game moves on to round 2.

Round 2:

In round two, each player is given a different randomly generated draw title and are asked to draw their interpretation of the title on the drawing canvas which appears under the draw title. They have 60 seconds to draw and submit their drawing. Players are not allowed to write letters or words. Once everyone has completed their drawings, or the time runs out, they will create a fake draw title for each of the other players drawings. This is one drawing at a time. Once every player has created a fake title for each of the other players drawings, they will move on to voting. In the voting step, for each other player's drawing they will be asked to select the real drawing title from a list which includes the fake answers. If they guess the correct title both the guesser and the drawer will be given 200 points. If they guess a fake title, only the fake title creator will be given 200 points. Each player also gets to give points for their favourite fake answer for their own drawing. After all votes are complete the score for after round 2 will appear and round 3 will begin.

Round 3:

In round three, each player is given a different randomly generated draw title but this time each player has only one mousedown to draw what the title is about. Meaning, if the mouse button is released at any point while drawing then the drawing is submitted. After the drawings are submitted, each player gets to choose on whose drawing they want to guess. Then the chosen player's drawing will be displayed to the user and if the user can guess the drawing title right then both the guesser and creator will receive 300 points. After all guesses are submitted, final scores will be calculated and will be displayed.

Notes while playing the game:

Due to the limited development time the game has been tested for mostly happy path testing so there are a few actions to avoid while playing the game

- Each player should have a unique name
- Only 1 game can exist at a time
- If a player leaves you will be unable to finish the game

If at any point the game becomes stuck, click on the 'Home' button to return to the home page and create a new game and have players join on the new game code. Doing this will reset the game and you can begin playing again.

1.2 Source Code

Source code: https://gitlab.csc.uvic.ca/courses/2020091/SENG350/teams/group_19/triqa

1.3 How to Build

To Play:

Hosted Site:

The web app for our game is hosted at younode.ca for a limited time, from there you can create and play a game. If it is not available at younode.ca, you will need to build it with one of the options below.

To Build:

Prerequisites:

- Access to a linux terminal or powershell terminal
You will need this to clone the repository and run the web application
- Have .NET installed
You can check by running `dotnet` or `dotnet.exe` in a linux terminal (git bash works) and you should see the following if installed

```
Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help           Display help.
  --info              Display .NET information.
  --list-sdks         Display the installed SDKs.
  --list-runtimes     Display the installed runtimes.

path-to-application:
  The path to an application .dll file to execute.
```

If it is not installed google how to install for your specific OS.

Instructions

Once you have the prerequisites installed you can do the following

1. Clone the repository for the project in whichever directory you choose with
`git clone`
https://gitlab.csc.uvic.ca/courses/2020091/SENG350/teams/group_19/triqa
2. Once it has finished cloning navigate to the triqa/TRIQA/TripleQA directory, in the example I cloned from the seng350test directory:

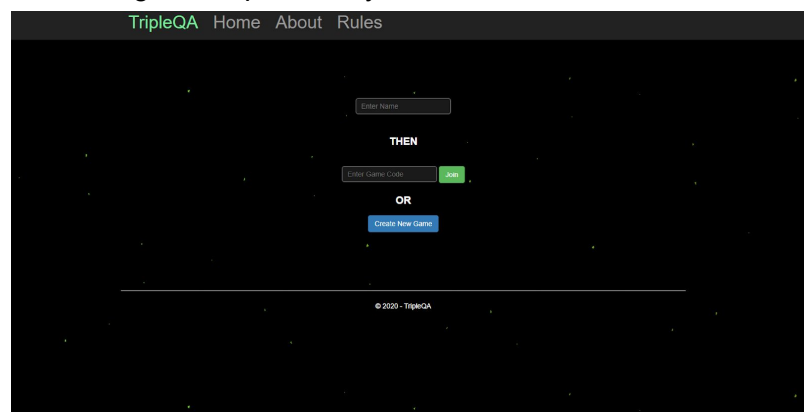
```
cd triqa/TRIQA/TripleQA
```

```
/seng350test/triqa/TRIQA/TripleQA
```

3. Once in this directory do either `dotnet run` OR `dotnet.exe run` and you should see the following:

```
$ dotnet run
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\adam8\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
Hosting environment: Development
Content root path: C:\Users\adam8\programming\seng350test\triqa\TRIQA\TripleQA
Now listening on: https://localhost:5001
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

4. Now open a browser of your choice and navigate to either of the localhosts and you should see the following if set up correctly



Alternative Method

This is an alternative method to run our game if you have Visual Studios downloaded and modified to have the following workloads downloaded: *.NET desktop development*, *Universal Windows Platform development*, and *ASP.NET and web development*. Once this is done you can clone the source code from our git repository listed above. Open Visual Studio and open the .sln file inside the first folder 'TRIQA'. Once it has loaded up you can click the IIS play button and it should build and run on your local host.

2. Beyond Views

2.1 Documentation Roadmap

- Scope and Summary:

The purpose of this document is to explain what the game is about and also assist the users to understand the implementation, functions of each element and client-server side communication. Thus, below in section 3, this document will elaborate on module, component & connectors and allocation views.

- How the Documentation is organized:

This document is clearly organized by the table of contents above.

Section 1 is where users can find what is the game about and describes each round of the game. Moreover, users can also find the link of the source code and steps on how to build and run the game.

Section 2 is about how the document is presented and a brief explanation of the architectural elements of the game by documenting the view.

Section 3 is where users can find more in depth details about each view of the game which are:

1. Module View

- ❖ Element types which are the implementation units of software that provide a coherent set of responsibilities.
- ❖ Relation and property types that show “is a part of” relationship between the submodule and the aggregate module.
- ❖ Modeling techniques on how the server and client module is created and how each module is connected with each other.

2. Component and Connectors View

- ❖ Elements types like components that are the principal processing units and data stores. Also, connectors which are connected via ports.
- ❖ Relation and property that show component ports that are associated with connector roles to yield a graph of component and connectors.
- ❖ A diagram of components and connectors that are associated with each other.

3. Allocation Structure View

- ❖ Software element properties that are required of the environment and environmental element properties that are provided to the software.
- ❖ How the game is related and allocated to an environmental element.
- ❖ Organisational structure involved with running our application on a cloud server machine.

- How stakeholders can use the documentation:

| Stakeholder Concerns | Scenarios | Addressed Section and Views |
|------------------------------------|---|---|
| Understanding and running the game | <i>"What is this game and how do i play this game"</i> <i>"Is this a single player or multiplayer game?"</i> | Section 1 of this Architectural Documentation |
| Client-Server Data | <i>"If a new script is needed to be added, where will it get added and how would the server module communicate with client modules"</i> | Section 3: Module View |
| Data Storage | <i>"How is the game accessing the player data"</i> | Section 3: Component and Connectors View |
| Limitations of PC specifications | <i>"Will it run smoothly on an old desktop? the old desktop's have trouble connecting to the wifi, will it affect the game?"</i> | Section 3: Allocation Structure View |

2.2 How a view is documented

Each view is documented using the standard template from 18.6 in the textbook [1]. The overview of this template is shown below:

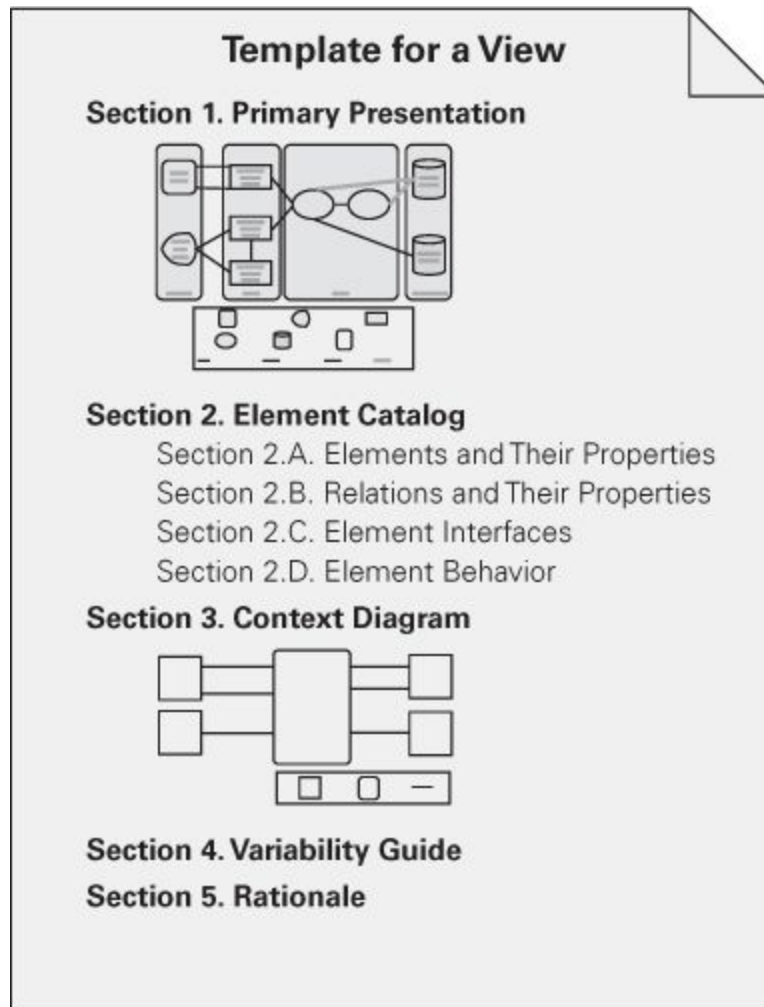


Figure 1. Template for documenting a view

A brief summary of each section:

- **Section 1. Primary Presentation:** This is used to show the elements and relationships within the given view. Usually some elements showing the normal operation of the system are shown, while excluding some error and exception handling. The primary presentation is usually graphical in the form of an informal diagram representing the particular view.
- **Section 2. Element Catalog:** This is a detailed list of the elements and relations depicted in the primary presentation. It is split into 4 subcategories of 2.A, 2.B, 2.C, and 2.D showing in Figure 1.
- **Section 3. Context Diagram:** The purpose of the context diagram is to show the view in relation to its surrounding environment and how it interacts with the overall system.

- **Section 4. Variability Guide:** This guide is used to show any variation points with the view being depicted.
- **Section 5. Rationale:** The rationale is to explain why the view is designed and why it was created this way. The rationale should be a convincing argument on why this design was the best choice.

2.3 System Overview

The system we developed consists of a web-hosted application which is completely server side meaning there is no database or added apis. The server keeps track of all of the current game information and distributes it from the backend which is written in C# to the frontend which is written in Javascript, JQuery, CSS, and HTML. During development we used Visual Studios with the development workloads of: *.NET desktop development*, *Universal Windows Platform development*, and *ASP.NET and web development*. This allowed us to run and test our web-app on the ISS locally hosted server. To host our web application we used one of our group members domains which has *Godaddy Ultimate Web Hosting* with *Plesk*. This is necessary for hosting *Microsoft asp.net* web applications. The front end communicates with the back end through post and get ajax calls which store and retrieve the game information respectively. After each section of a round is completed we use a recursive pooling function to check if the other players have also finished the section. This is needed to keep the players in the same round and allows the game to flow properly. For the two drawing rounds, we used a canvas element for drawing and then converted the canvas into a webp image and stored the src for that image in the server. Much of the server information including this canvas src are store in C# static class dictionaries with the player's name and then the data that we are temporarily storing. This is why we did not need to connect our game to a database. After a game is completed, another game can be started, and currently only one game can be played at any given time.

2.4 Mapping Between Views

The module view only contains the major functions used in each module, the component and connector view has a more detailed list of functions. The following table is a mapping between the module view and the component and connector view.

| Module Element | C&C Element | Mapping |
|-------------------|--------------------|--|
| HomeController.cs | HomeController.X() | All functions X are contained within the HomeController.cs module. |
| site.js | site.X() | All functions X are contained within the site.js module |

The following table is a mapping between module and allocation view.

| Module Element | Allocation Element | Mapping |
|----------------|--------------------|--|
| Client | Web Browser | The Client module is executing on the Web Browser software allocation unit which is operating on the User's PC |
| Server | Web Application | The Server module is the web application that is deployed using windows OS onto the Plesk Web Hosting server that is available at the younode.ca url |

2.5 Rationale

The design of this game was planned to achieve the best outcome which is having the Frontend views designed by HTML and CSS and scripted by JavaScript and JQuery. Furthermore, the Backend Server Controller manages the game flow and stores dynamic information for a specific game by using List with Dictionary. The architectural problem here was on how this design will store dynamic/runtime data about the players. We either had to choose to have a Database or to have a List. The rationale for choosing a List over Database is because our game does not need to retrieve the saved data after the game is over. The game is needed to store player data each game at a time and lose the data once the game is over. Therefore List was the simpler and faster way to solve this architectural problem. Moreover, in section 3 of this document each view will describe its own rationale on why each module's design was the best choice.

2.6 Glossary

IIS : Internet Information Services. Used to host websites and web apps.

CSS : Cascading Style Sheets is a frontend language which defines styles for classes.

HTML : Hypertext Markup Language is a frontend designing and displaying language.

MVC : Model View Controller is a type of framework from web applications

PC : Personal Computer

User : An individual using our web application through their PC

Webp : An image format with lossless compression

AJAX : Asynchronous Javascript and XML used to send and retrieve data from the server.



XML : Extensible Markup Language used as a set of rules for server requests

3. Views

3.1 Module View

3.1.1 Primary Presentation

The module view represented below is in an informal format showing the major modules and decomposition in class-like format. The following can be used as a key for the diagram:

-  Boxes are modules and submodules with attributes and operations listed
-  Boxes are high level modules
- Single solid lines show relations between modules
- Double solid lines show a interface between modules

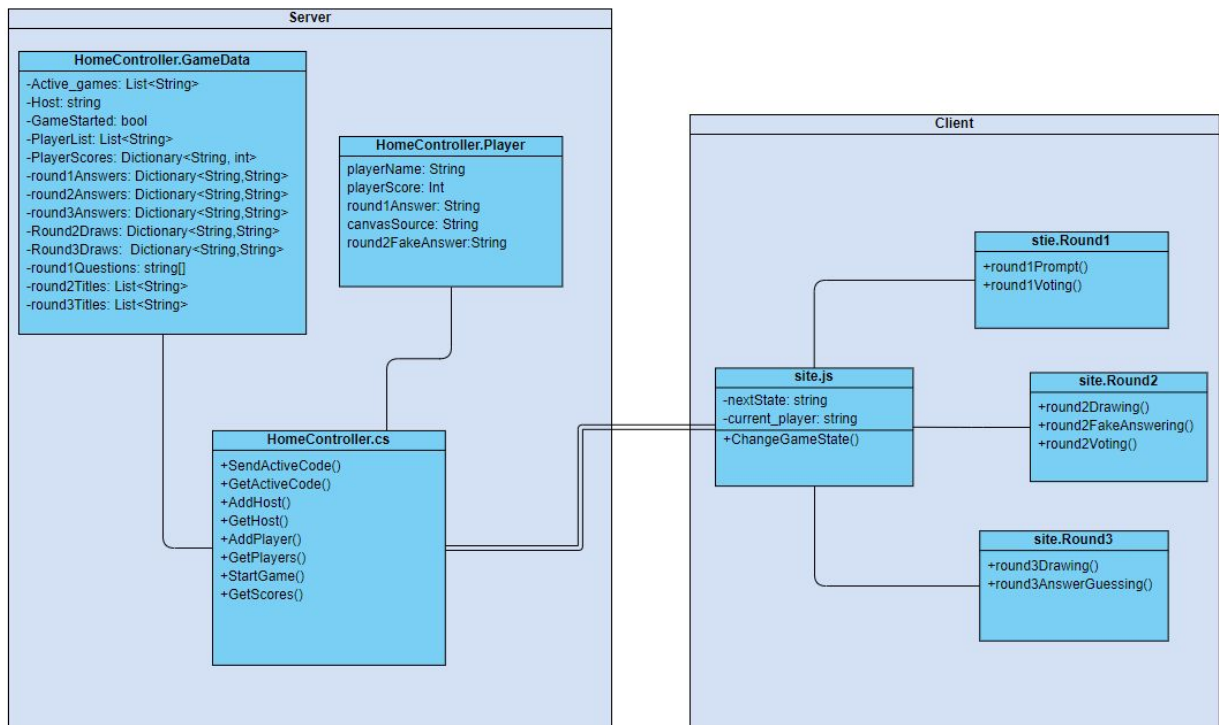


Figure 2. Module View Primary Presentation

3.1.2 Element Catalog

Elements and their properties

| Element | Properties |
|-------------------------|---|
| Server | The module encapsulating the submodules that operate on the server side |
| Client | The module encapsulating the submodules that operate on the client side |
| HomeController.cs | The C# controller that is responsible for organizing the game flow and storing information from each game. Contains functions responsible for storing data and changing the game state |
| HomeController.GameData | The GameData class is part of the HomeController and contains the data structures used to store information related to each game |
| HomeController.Player | The Player class is part of the HomeController and contains variables that are used to store information from the client for use in HomeController.cs |
| site.js | Site.js is the javascript control for game flow using the roundX modules on the client side. It interacts with the GameController.cs to synchronize all clients. Contains functions responsible for displaying the game to users and interacting with the user to get data and send it to the server. |
| site.Round1 | Round1 is part of site.js and is responsible for the round 1 control and game flow |
| site.Round2 | Round2 is part of site.js and is responsible for the round 2 control and game flow |
| site.Round3 | Round3 is part of site.js and is responsible for the round 3 control and game flow |

Relations and their properties

| Relation | Properties |
|--|--|
| HomeController.GameData to HomeController.cs | This is a "Is Part Of" relation showing that HomeController.GameData is a submodule of HomeController.cs |
| HomeController.Player to HomeController.cs | This is a "Is Part Of" relation showing that HomeController.Player is a submodule of HomeController.cs |
| site.Round1 to site.js | This is a "Is Part Of" relation showing that site.Round1 is a submodule of site.js |
| site.Round2 to site.js | This is a "Is Part Of" relation showing that site.Round2 is a submodule of site.js |
| site.Round3 to site.js | This is a "Is Part Of" relation showing that site.Round3 is a submodule of site.js |

Element Interfaces

| Elements | Interface |
|-------------------------------|--|
| HomeController.cs and site.js | This is a interface showing the two way communication between modules using HTTP POST and GET requests |

Element Behaviour

| Element | Behaviour |
|-------------------|--|
| HomeController.cs | This C# controller is used to handle all the POST and GET requests from site.js. The POST and GET requests use the HomeController.Player class to get information from the request and uses HomeController.Game to store any needed information. |
| site.js | The site.js implementation is used to send the HTTP POST and GET requests and used the round 1,2 and 3 modules to determine what needs to be sent in the POST and received in the GET requests |

3.1.3 Context Diagram

The following context diagram is to show how the server module is seen and how the client module is used. The key for the below diagram is the following:

- Hollow Arrow is mapping from module to a user context

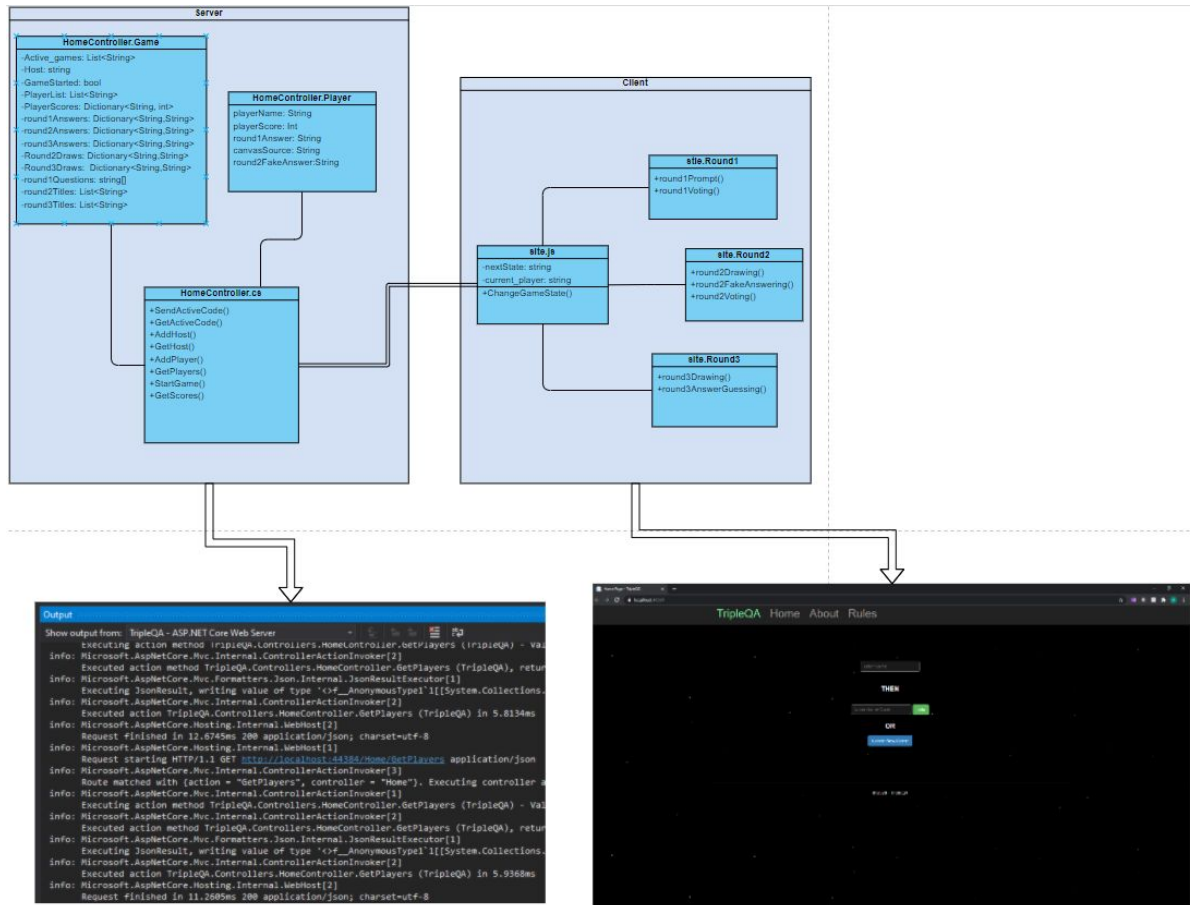


Figure 3: Module View Context Diagram

In the above context diagram the left hollow arrow is showing the server handling a GET request, the right hollow arrow is the client view on the home page.

3.1.4 Variability Guide

Looking at the high level modules of client and server there are a few variability points that can be exercised. The client was created using javascript and html making it portable to web browsers supporting these languages, the main 3 being Google Chrome, Mozilla Firefox, and Microsoft Edge. The server module is portable and able to run on any system able to clone the repository using IIS Express and having asp.net installed.

3.1.5 Rationale

The module view described in this section was created from the original planning that had been done with the class and object diagrams. A few changes were made during implementation because it became apparent that some of our planning was inaccurate and needed to be improved in the real implementation. The strategy pattern that we had planned to be used was changed as one less layer of abstraction was used in the implementation for the round modules. The structure was changed because the inputs and outputs of each round were not generalizable so the abstract round class was not able to accomplish its purpose. Instead the main pattern used was the State pattern as the site module used a change state controller that relied on knowing the next state to go to. This pattern was used because each round had multiple states so controlling the current state of the client became the main control of the system.

3.2 Component and Connector View

3.2.1 Primary Presentation

This particular view will convey information about the primary elements and relations of this game. These structures Below is an overview of the system's components and connectors:

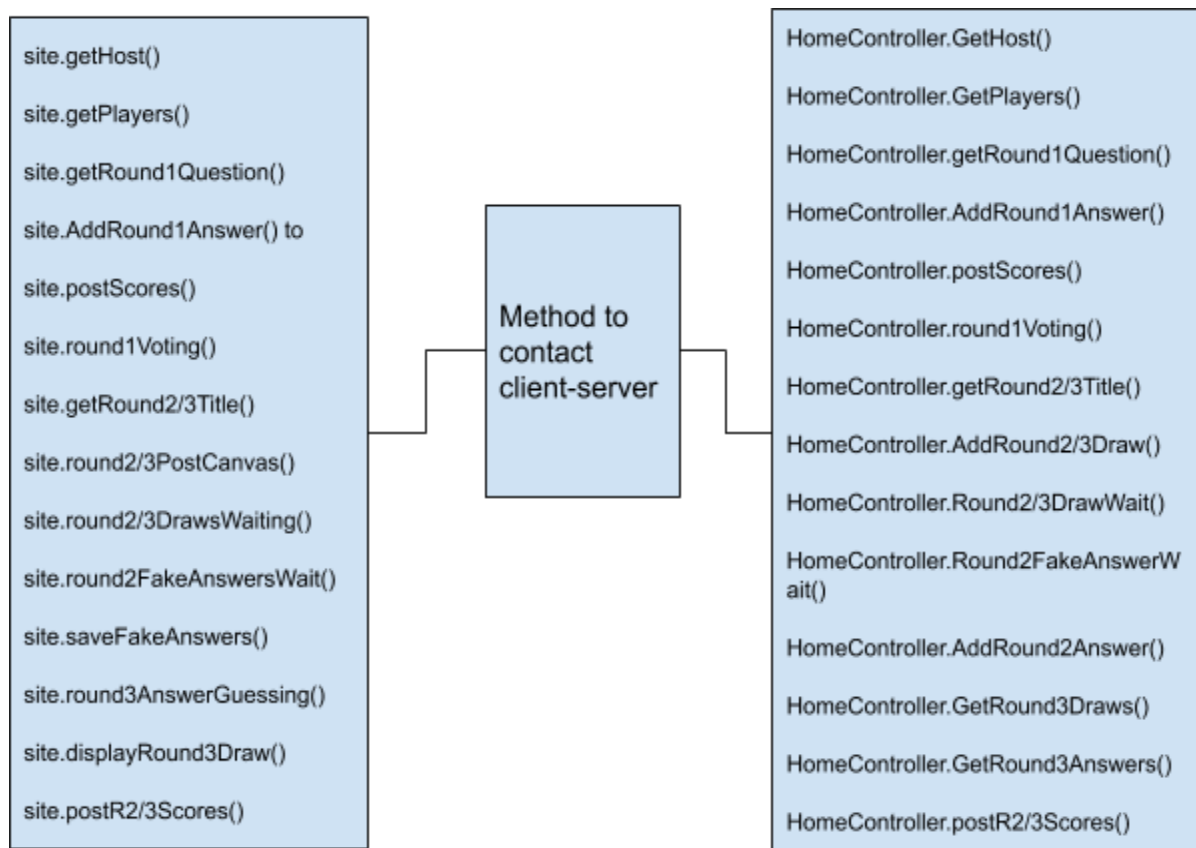


Figure 4. C&C Primary Presentation

3.2.2 Element Catalog

| Elements | Properties |
|--|---|
| HomeController.GameData.PlayerList | A Component which has properties of <i>"Reliability"</i> as the player data will determine the system's availability |
| HomeController.GameData.PlayerScores | A Component which has properties of <i>"Modifiability"</i> as the player score will constantly evolve after each round |
| Site.js All Wait functions | A Component which has properties of <i>"Concurrency"</i> as the system will constantly be in a deadlock when players are in a different state of the game |
| Game.cshtml onload | A Component which has properties of <i>"Performance"</i> as the body and the functions will load faster or slower depending on the user's latency |
| HomeController.GameData.roundXAnswers | A Component which has properties of <i>"Reliability"</i> as the players roundxanswers will determine each round's availability |
| HomeController.GameData.Round2Draws HomeController.GameData.Round3Draws | A Component which has properties of <i>"Reliability"</i> as the payer Draws will determine round 2 and 3 availability |
| HomeController.round1Questions HomeController.round2Title HomeController.round3Title | A Component which has properties of <i>"Resource Requirements"</i> as each rounds resources are required from these components |
| All ajax functions in site.js | Connectors which have properties of <i>"Roles"</i> , as they invoke-services and provide-services by reading and posting data. |
| Success functions in all ajax functions in site.js | Ports that defines the kind of behavior that can take place at that point of client-server interaction |

| Relations | Properties |
|--|---|
| <p>site.getHost() to HomeController.GetHost()</p> <p>site.getPlayers() to HomeController.GetPlayers()</p> <p>site.getRound1Question() to HomeController.getRound1Question()</p> <p>site.AddRound1Answer() to HomeController.AddRound1Answer()</p> <p>site.postScores() to HomeController.postScores()</p> <p>site.round1Voting() to HomeController.round1Voting()</p> <p>site.getRound2/3Title() to HomeController.getRound2/3Title()</p> <p>site.round2/3PostCanvas() to HomeController.AddRound2/3Draw()</p> <p>site.round2/3DrawsWaiting() to HomeController.Round2/3DrawWait()</p> <p>site.round2FakeAnswersWait() to HomeController.Round2FakeAnswerWait()</p> <p>site.saveFakeAnswers() to HomeController.AddRound2Answer()</p> <p>site.round3AnswerGuessing() to HomeController.GetRound3Draws()</p> <p>site.displayRound3Draw() to HomeController.GetRound3Answers()</p> <p>site.postR2/3Scores() to HomeController.postR2/3Scores()</p> | <p>All these relations are components connecting with each other via roles and ports.</p> <p>All of the client functions use AJAX functions as a connector role that interacts with the server.</p> <p>All Server functions return the data that are stored by formatting the data into JSON format.</p> <p>Each client function has ports that allow the client components to determine what action will be taken after reading or posting data.</p> <p>Therefore, all these relations are “<i>Attachment</i>” as they are associated with a component’s port to a connector’s role.</p> |

Element Interfaces

| Elements | Interface |
|---|---|
| HomeController.cs functions and site.js functions | Almost all the functions's interface show a client-server communication by AJAX |

Element Behaviour

| Element | Behaviour |
|-------------------|---|
| HomeController.cs | This C# controller is used to handle all the POST and GET requests from site.js. The POST and GET requests use the HomeController components to get information from the request and to store any needed information. |
| site.js | The site.js implementation is used to send the HTTP POST and GET requests and uses the round 1,2 and 3 success functions to determine what action is needed to be taken after receiving and posting the data. |

3.2.3 Context Diagram

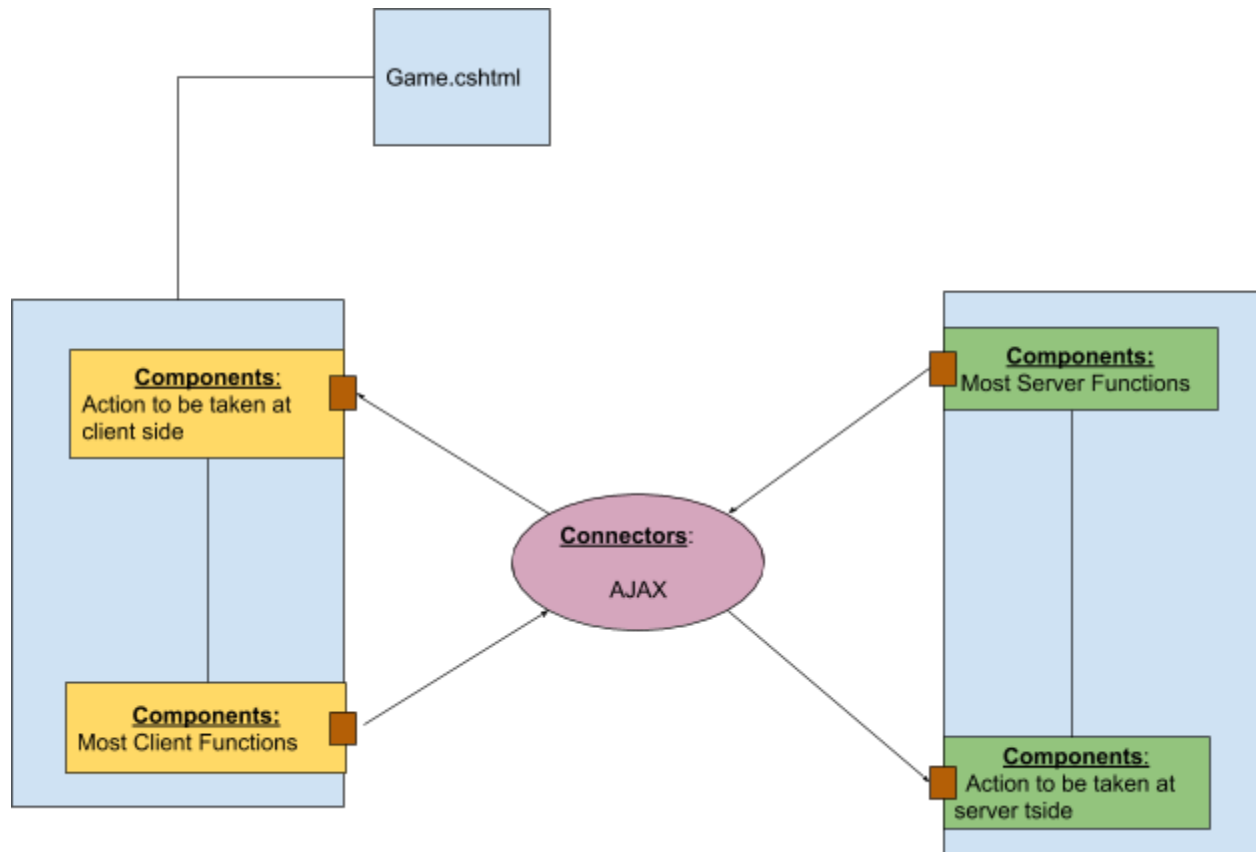


Figure 5. C&C Context Diagram

Figure 5 shows a context diagram on how the system is depicted in this view. This context diagram will explain the basic game flow of Components and Connectors. First The Game.cshtml calls 3 functions to load the host and players from the client-server. The game then constantly has client-server communication to make sure all players are in the same state of the game. To be able to have a client-server communication the game uses Ajax as a connector which is a way that the client components can read and post data into the server. The Server components will send and return the data that are stored in a list in JSON format.

3.2.4 Variability Guide

The client-server interaction can be varied depending on the user's performance. Any variation can be exercised by modifying and varying the deadlocks in the systems client and server components.

3.2.5 Rationale

This C&C design and pattern was chosen to achieve the best results in a small period of time. The main architectural problem in this view was about choosing a method to have an efficient client-server communication with no database. Thus, choosing AJAX was the best choice as the instead of WebSockets because AJAX is asynchronous, meaning the client components can send any number of requests without having to wait for a response [2] thus making the performance more efficient.

3.3 Allocation View

3.3.1 Primary Presentation

The allocation of software units can be seen below as the User's PC accesses our web application through their web browser with a secure SSL connection with the server. This shows how software units are mapped on a server level. This is similar to the explanation given in the system overview section 2.3.

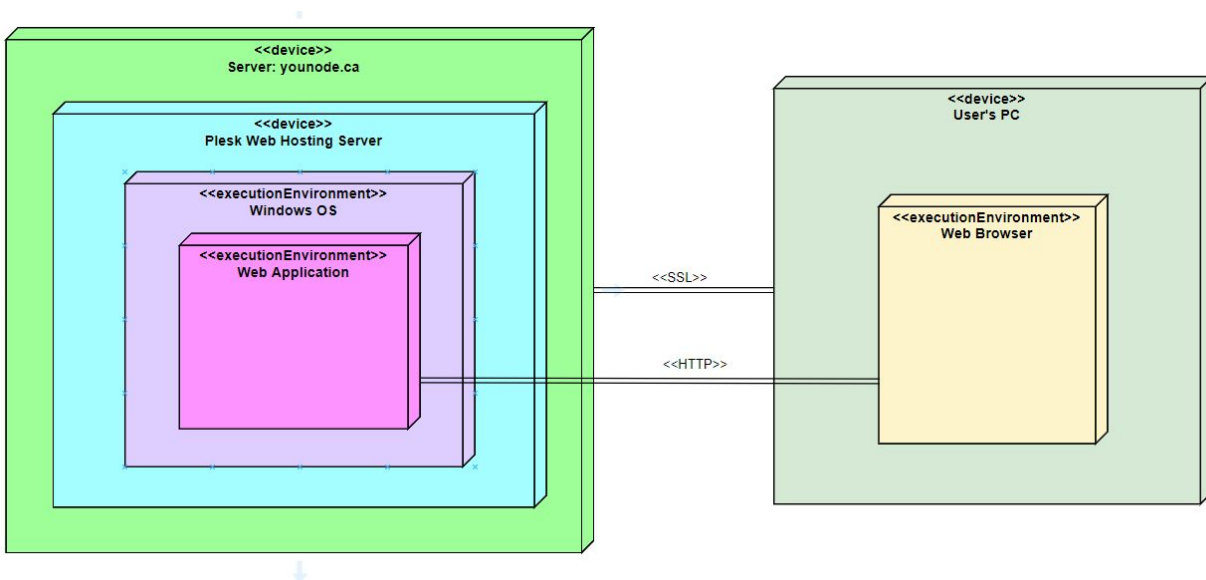


Figure 6 : Allocation Primary Presentation

The figure below shows a software architecture breakdown of the frontend and backend languages used and how they communicate within the server.

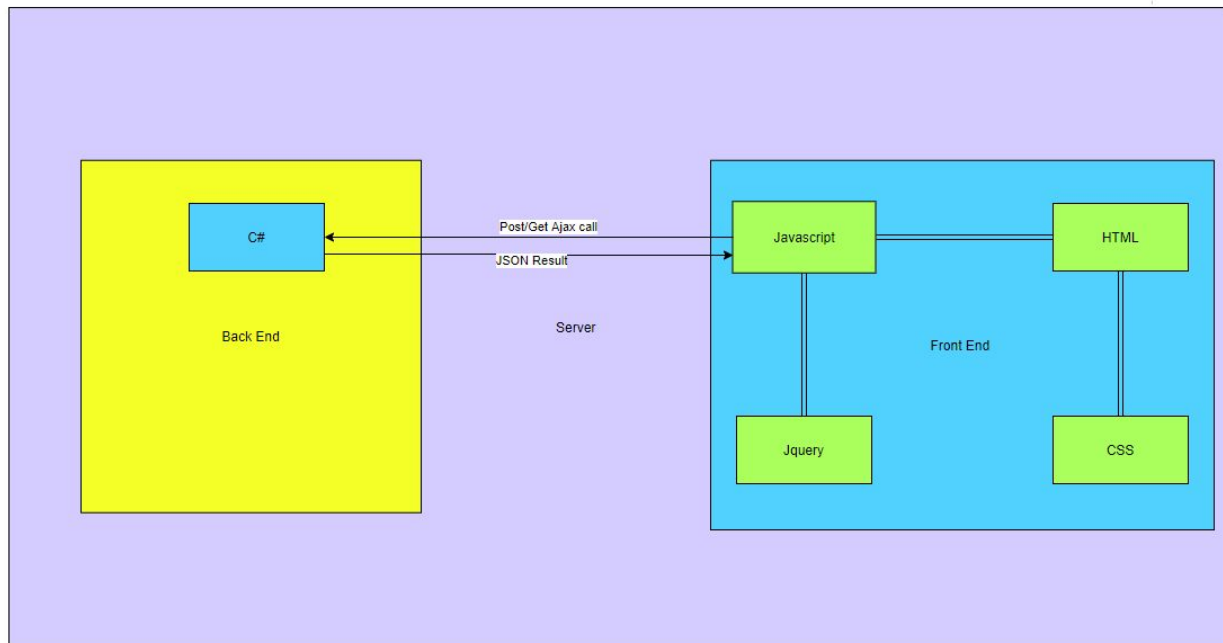


Figure 7. Software Language Allocation Structure

For project management software allocation during this project, we would split the work evenly between our 3 groups members in a way which focuses first on the most important development needs for maximum efficiency. At the beginning of Milestone 4, we built a medium fidelity prototype which helped us understand what the views would look like and the game flow. For development we each had different tasks and we organised these tasks on an online project management tool called *Trello*. *Trello* allowed each of us to have a space to drag our current tasks we were working on. Once we finished our tasks we would drag the task into the done space and choose the next task from the task left to do. Below is an example of our group's *Trello* board at some point during development.

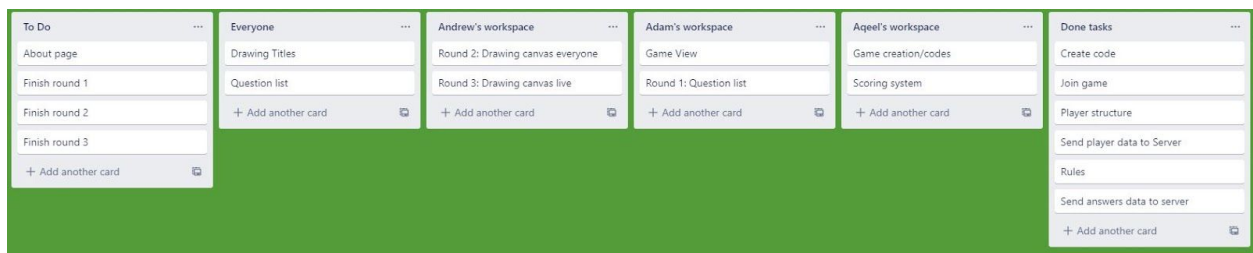


Figure 8. Development Team Task Allocation on Trello

3.3.2 Element Catalog

| Elements | Properties |
|-------------------|---|
| Server | Cloud server hosted by <i>GoDaddy</i> with <i>plesk</i> |
| User | Client using and connected to the web-application. |
| Web Application | Application which is hosted in the form of a website. Web Application type is MVC asp.net core. |
| PC | Users PCs are connected to the internet and access our application through our web server. |
| Plesk Web Hosting | Web hosting service provider that allows window OS .NET Web Applications |
| Windows OS | Operating system needed to run windows applications. |
| Trello | Project management software used to allocate tasks to team members |

3.3.3 Context Diagram

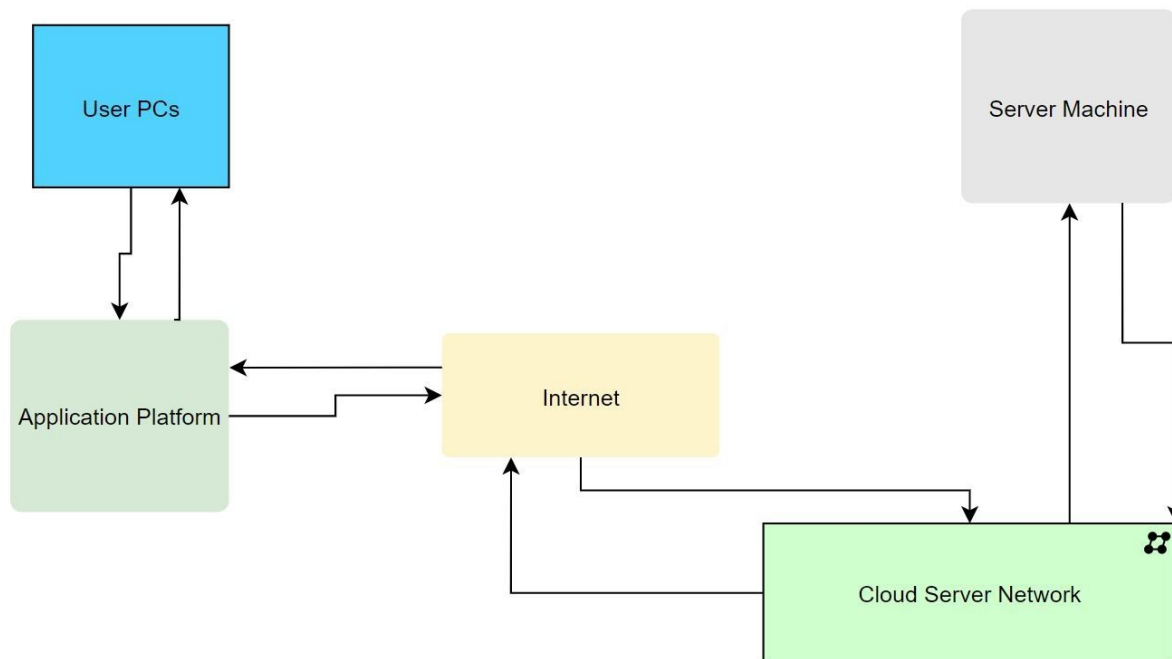


Figure 9. Allocation Context Diagram

The following diagram is a context diagram which shows the organisational structure involved with running our application on a cloud server machine. The basic flow starts with the user starting or joining a game on their PC by accessing our online web application platform via their browser. The application platform communicates through the internet with the cloud server network, which in turn, updates the game variables on the server.

3.3.4 Variability Guide

There are many different points of variability such as the user's PC specifications, internet connection speed and strength, and internet browser used. User's PC specifications such as RAM, OS, and Processor will either cause the web application to run more smoothly or most inconsistently. This is even more the case with variability in internet connection. If a user's internet connection is slow it will cause the get and post requests to the server to slow down or even fail. If the user is disconnected from the internet while playing a game they will be unable to rejoin the game as we did not have time to create a monitor class to rejoin them into the game. There are also many different types of browsers which users will use to access our web application. Browsers have different default fonts and button styles so the application will display differently with each browser. We recommend using Google Chrome.

3.3.5 Rationale

The structure of development was created in this way to allow users to access and play our game anywhere in the world for no cost and without downloading. We used SSL server certificates to ensure that the user has a safe connection with the server. We also chose to develop our game as a web application, because they are the fastest way to get a product to the user for acceptance testing of the idea without the expense of building a downloadable copy. As for the project development technique we chose to do an evolutionary prototyping based on the structure of the UML diagrams we made in the previous Milestones. We chose to start with a medium fidelity prototype so that each group member had a consistent understanding of the game flow and views that need to be developed. We separated the development into tasks and used *Trello* which really helped as we did not have any overlapping development. Overall, the methods we used for allocation of software units was successful.

4. References

- [1] L. Bass, R. Kazman, and P. Clements, *Software Architecture in Practice, Third Edition*. Addison-Wesely Professional, 2012.
- [2] Wilfred Nilsen, “*AJAX over WebSockets*” Web Dev Zone, Nov, 2018