International Hellenic University

Department of Information and Electronic Engineering

Development of a web application to use multiple application queries as a way of teaching for eLearning.

Thesis paper of

Tsougantzalis Nikolaos (144246)

Supervisors: Sidiropoulos Antonis, Assistant Professor

Kaplanoglou Pantelis, Adjunct Assistant Professor

THESSALONIKI, JUNE 2021

***Responsible Statement**: I certify that I am a writer of this thesis and that any help I had to prepare is fully recognized and refers to the thesis. I have also mentioned any sources of which I have used data, ideas or words, whether they refer to exactly either paraphrased. I also confirm that this thesis was prepared by me personally especially for the requirements of the Department of Informatics, Computer and Telecommunications Department of the International University of Greece.*

Abstract

When the Covid-19 pandemic swept across the world in 2020, schools closed and education was moved to students' homes. The necessity of taking online classes during the pandemic led us into the creation of this thesis project. We created a web application and that enhances the process of eLearning. Emphasized on control, model, view prototype, we will focus on the connection of our front end with the back end and finally with a non-relational database like MongoDB. Node.js, express.js as a form of server will be extensively analyzed till the final result of our thesis project. Intriguing is the synchronization of the users through socket.io java script library. Moreover, we will describe from theory to practice on how we combined and implemented technologies and libraries like jQuery to build our quiz app application. Of course, user experience, theory of color, risk analysis, project management and case game-based learning are terms that concerned us a lot. In conclusion, we aspire for new releases, we sharing the importance and the impact that this project have as a tool in digital education.

Contents

Tables

Figures

Keywords: node.js, socket.io, MVC, agile, express.js, mongo dB, bootstrap, npm, database, frontend, backend, theory of color, risk, analysis, responsive, user, experience.

1. Introduction

Web Applications became a part of our daily life for many years now. We can observe online trainings web applications, entertaining web applications or in our case an educational online web application that will upgrade the procedure of learning through online classes of the university. Since the beginning of Covid-19 in 2020, students of universities all over the world, started to attend classes online and this created the need of new tools that will provide immediacy and interaction between the professors and the students.

This thesis will present the creation of an educational related web application using modern web development technologies. Moreover, we will compare and discuss about architectures of coding and what was the proper choice for our project so we can have future installations and expansions. Please note the importance of time scheduling when you start your own creation of a web application, as we will mention the procedure of how to create a project with a Gannt chart or using other online tools like trello that provide help on scheduling your project.

On the second chapter we will discuss about the technologies, frameworks and databases that we used so we can have our final project. The importance of JavaScript and the evolution into modern frameworks like express.js, node.js, moment.js is just an example of what we're going to analyze on this chapter. From the data storing side, we will have the opportunity to check the Mongo DB as an alternative way of storage in our web application. Synchronizing users with a host will be achieved with socket.io with web socket and functions that allows the feature of having multiplayers. Last but not lease we will build a user interface (UI) with bootstrap rules so we can have a responsive application.

Following the second chapter, after presenting the tools that we will include in our project, we will analyze the functionality of them and their role to our thesis project. Decision and procedures about creation of backend and frontend. Main concerns will be the design of a UI that will emphasize to the best user experience. In this chapter we will discuss about the connection of the backend and frontend and how we achieve web socket connectivity, meaning the multiuser functionality and how someone can be the host of a game. Mentioning the word game for the first time, we can finally describe and name our app as a multiplayer quiz app for online classroom purposes.

Finally, on chapter four we will mention ideas for future work, we will share testing results and how we achieve the reduction of some errors. On the final testing process, before the final deployment, we will focus on user errors that we can avoid in our application. Additionally, as our project growths, we want to share the possibilities of expansion and conversion in to versions of deployment. As C.A.R Hoare mentioned, "The most important property of a program is whether it accomplishes the intention of its user." and this will be our goal in this thesis project.

2. Theoretical Background

2.1 Project Management

The purpose of project management is to ensure that all the key processes and activities work in harmony. Building successful Web-based applications requires close coordination among various efforts involved in the Web development cycle. Managing a large, sophisticated Web development project is a difficult endeavor that necessitates diverse talents and differs in several ways from managing typical IT/software projects. [1].

Significant role to our purpose is the variable of time. Time management completed through Project 2019 Microsoft (Figure 1) that is free to IHU students through university access via Azure. Microsoft Project is a program that aids in the creation of project plans, as well as real-time and scheduled completion dates. [2].

The early endeavours of project management and scheduling date back to the development of the Gantt chart by Henry Gantt (1861–1919). The Gantt chart is a horizontal-bar schedule showing activity start, duration, and completion.[3].

Additionally, we worked on a web platform called trello (Figure 2) that help us on task managing and communication with other team members, the professor in our case. Project ideas, future work or task completion were handled there via posting images, workflow or other material. Also, the feature of labelling is very important and help us understand the separation between sectors of workflow that is required to create our thesis.

Last but not least, we used Zotero for the citation. Zotero is a free, easy-to-use tool to help you collect, organize, cite, and share research. Focused on the bibliography structure, Zotero helped us on time management as we didn't handwrite every part of the necessary citations.

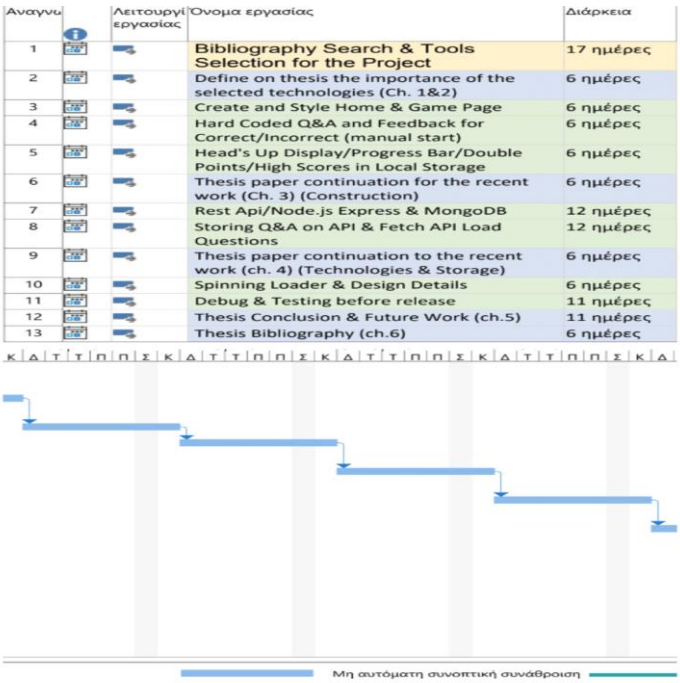


Figure 1:Partly representation of our first attempt of Gantt Scheduling

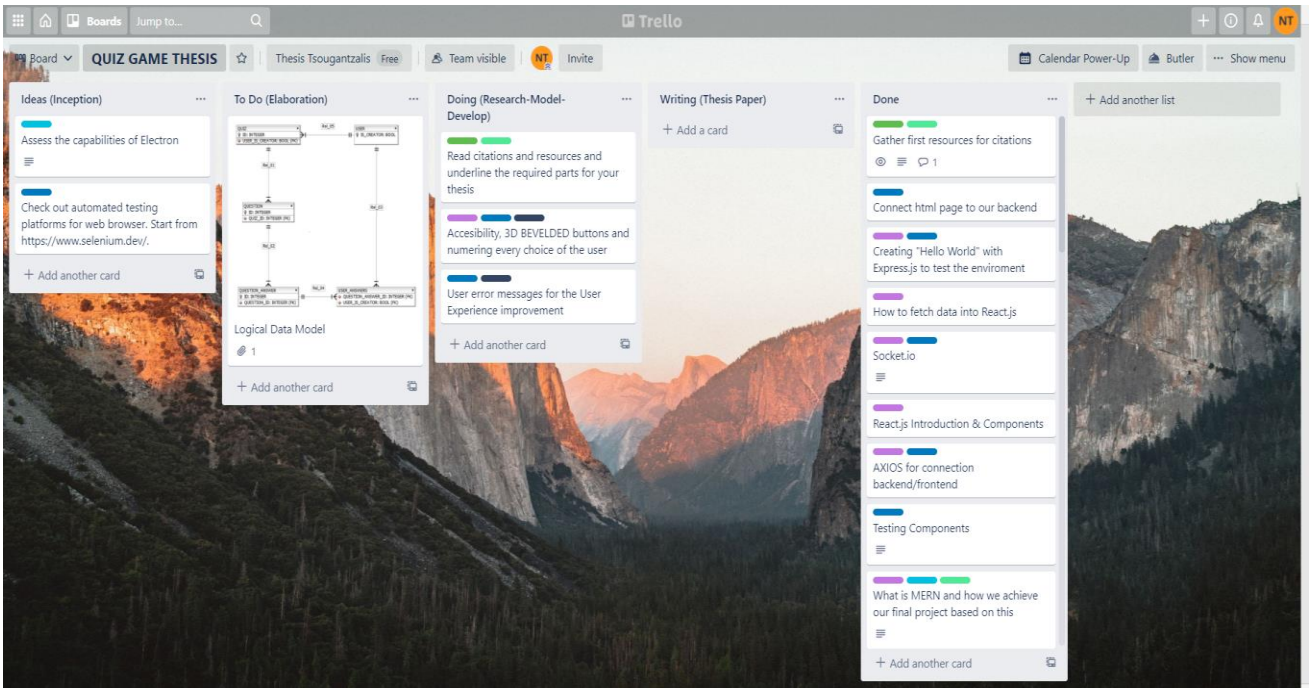


Figure 2: Representation of our thesis tasks in Trello

2.1.1 Agile Manifesto/ Software Development

Agile development is built on the concept of incremental and iterative development, in which different phases of a development life cycle are examined repeatedly. It converges on solutions iteratively by leveraging consumer input to enhance software.[4] As stated in t Agile Manifesto, the fundamental factors of agile rules include the following four:

- Early customer involvement
- Iterative development
- Self-organizing teams
- Adaptation to change

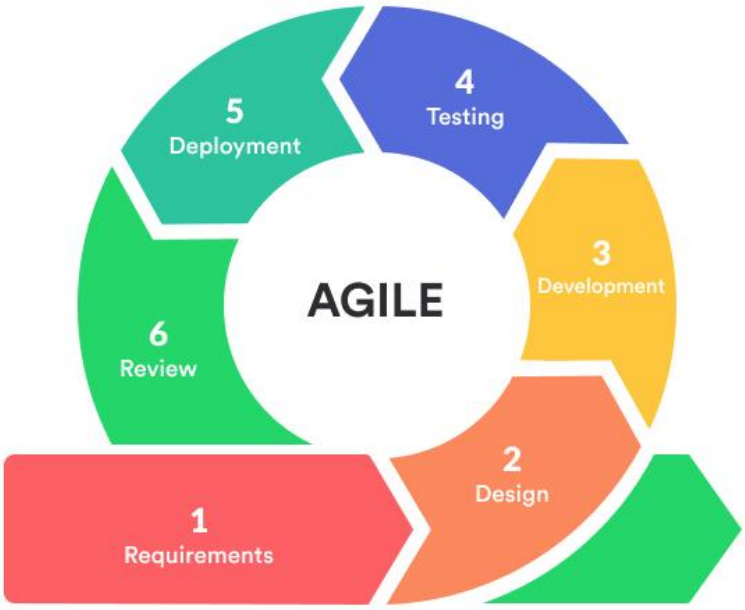


Figure 3:Steps for Agile Development [5]

In other words, we used agile development so we can review our project again and again until the final result. The importance of the agile methodology over the traditional way of development can be visible on the customer satisfaction levels and the collaboration you built with your customer. Another advantage that agile method provides is the feature of responses to changes. At the start of a project, traditional software development approaches entail creating and documenting a reliable set of requirements. [4] . On Table 1 we found the main differences of traditional and agile development so we can have a clear image of the comparison.

Table 1:Comparison of Agile and Traditional Approaches [4]

	AGILE	TRADITIONAL
User requirement	Iterative acquisition	Detailed user requirements are well-defined before coding/implementation
Rework cost	low	high
Development direction	Readily changeable	Fixed
Testing	On every iteration	After coding phase completed
Customer involvement	high	low
Extra quality required for developers	Interpersonal skills & basic business knowledge	Nothing in particular
Suitable Project scale	low to medium-scaled	Large-scaled

2.1.2 Model-view-controller (MVC) Framework

Model, View, and Controller (MVC) is an acronym for Model, View, and Controller. MVC is a popular method of coding organization. The core concept underlying MVC is that each portion of your code has a distinct purpose. Some of your code contains your app's data, part of your code makes it seem great, and some of your code regulates how it works.[6]

To determine each word of the MVC framework, we must first get the three basic notions included inside it:

- Model: Model code is used to represent real-world objects. This code might include raw data or the components of your program.
- Controller: This section of your code describes what to do with the user input that is received. The controller serves as a link between the view and the model.
- View: View code is how the built functions interact with our users directly. This is the section of your code that gives it style and makes it appear great; it represents how people view and interact with your program.

2.2 Visual studio code as developing environment for web applications

A lightweight yet capable source code editor, Visual Studio Code is running on your desktop and accessible for Windows, MacOS and Linux purposes. It has JavaScript, TypeScript and Node.js support and a wide range of additional language extensions (e.g., C++, C#, Java, Python, PHP, Go, etc.) as well as runtimes (e.g., .NET and Unity) [7].

In the first place and on a tiny footprint VS code is an editor who prides itself. You may adapt the device to the technology you take care of, unlike standard IDEs that prefer to incorporate everything other than a kitchen sink. [7]

Of course, VS code (Figure 4) is free and serves the purposes of this thesis. Mentioning the above from the FAQ page of the visual studio, we conclude that we can combine all the required technologies, frameworks, tools and extension into an IDE that is easy to use and perfectly created for web developing.

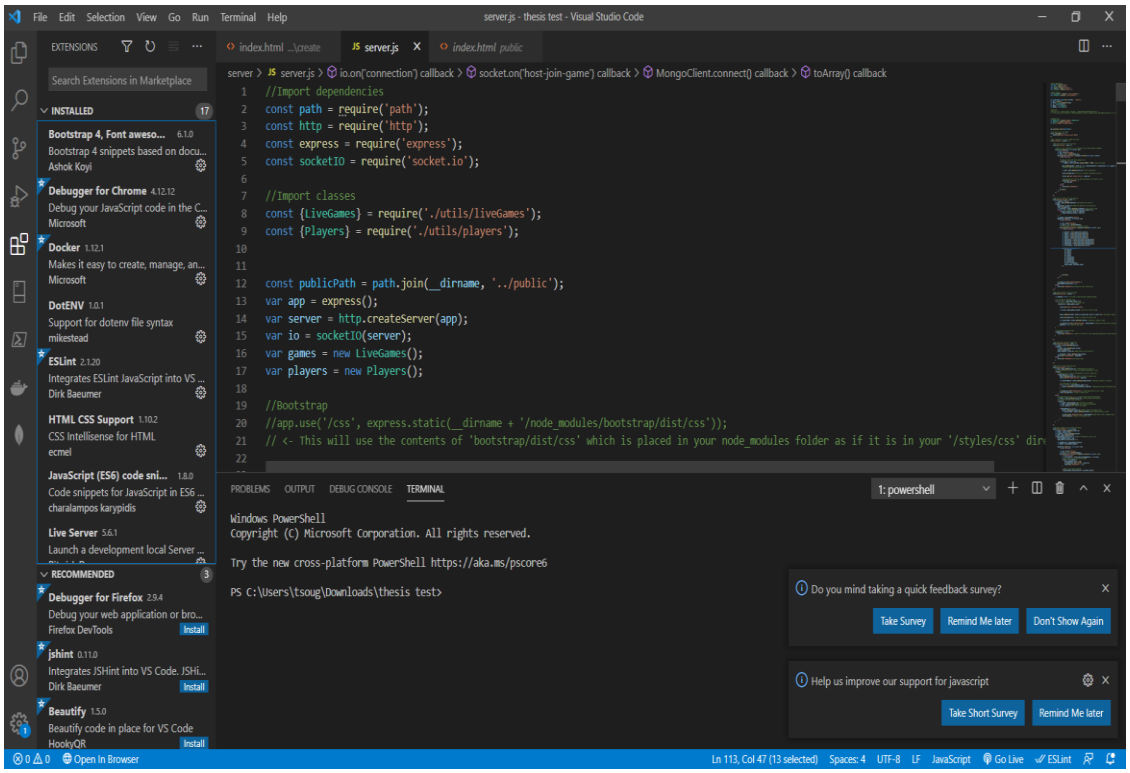


Figure 4:Presentation of our VS code environment on extensions page

2.3 Applied JavaScript Technologies for the creation of a web app

During the project management process, we had to decide the basic necessities for the build of a web application. Fundamental was the fact of using a technology that will depict the modernity of coding and how we can achieve into one environment with zero cost. Based on the npm.js we will have a combination of installations for the completion of a web application (Figure 5)

Figure 5:Dependencies info from our package.json file

2.3.1 npm as world's largest software registry

Starting by saying that npm is the world's largest software registry. Open-source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well. Npm consists of three distinct components:

- the website
- the Command Line Interface (CLI)
- the registry

Use the website to discover packages, set up profiles, and manage other aspects of your npm experience. For example, to manage access to public or private packages, you might build up organizations. The CLI runs from a device and the interaction between most developers with npm. The registry is a big public database containing the meta-information about JavaScript applications. [8]

In addition, we can define the benefits and the functionality of this invention in the software registry by declaring that [8]:

- Adapt or integrate code packages for your applications.
- You may immediately download standalone tools.
- Download packages using npx without running.
- Every npm user should share code, wherever.
- Specific developers restrict code.
- Create package maintenance, coding, and developer coordination organizations.
- Form virtual teams through the use of organizations.
- Multiple code versions and code dependencies are managed.
- Update apps easily when underlying code is updated.
- Discover several methods in which a puzzle can be resolved.
- Find others developers working on comparable projects and issues.

On Figure 6 we can check the existence of npm in well-known companies as a fundamental tool for their functionality in the modern world. There is a variety of applications in our everyday life built with the help of npm.js.

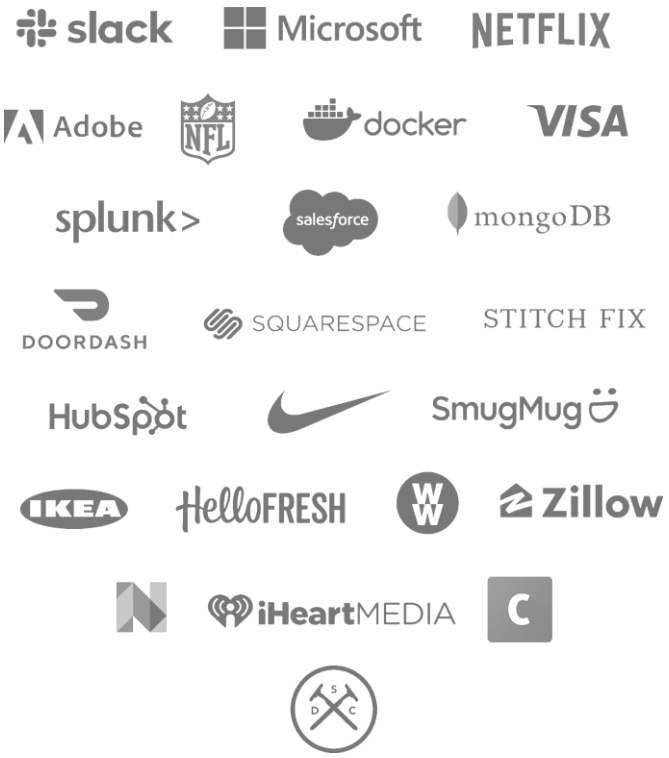


Figure 6:Gratefully serving everyone from solo developers to the Fortune 500 [9]

2.3.2 Node.js

A server-side JavaScript environment is Node.js, which is sometimes known as the Node. It is based on the execution of Google — the properly titled V8 engine. V8 and Node, which emphasize on performance and memory low, are generally implemented in C and C++ [10].

You may use a npm registry in the Node.js environment to distribute a Node.js library. Node.js can manage quick, high-performance applications and massive data flows in real-time. One example is the development of a gadget which can deliver control commands to your workstation. You may install Node.js on your IoT board or you can preinstall Node.js on your device. Then you would create and deploy the application logic in JavaScript. [11] .

You can build the following types of applications by using Node.js:

- HTTP web servers
- Microservices or serverless API back ends
- Drivers for database access and querying
- Interactive command-line interfaces
- Desktop applications
- Real-time Internet of Things (IoT) client and server libraries
- Plug-ins for desktop applications
- Shell scripts for file manipulation or network access
- Machine learning libraries and models

2.3.3 Express.js

Express is a web application framework basic and adaptable to Node.js which offers a rich collection of web-based and mobile functionalities. The creation of a rich API is quick and straightforward with a host of HTTP utility methods and middleware available to you. Express offers a thin shell of essential web application capabilities that you know and like without obscuring Node.js.[12]

```
const express = require('express' 4.17.1 )
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

Figure 7: "Hello World" example of express.js

Based on figure 7, we can comment about the most famous first app, "Hello World". We will have the opportunity to analyze how we use the express.js in our code on the following chapters. For now, let's explain the simplest educational test project.

This app (figure 7) starts a server and listens on port 3000 for connections. The app responds with “Hello World!” for requests to the root URL (/) or route. For every other path, it will respond with a 404 Not Found. [13]

In summary, Express.js builds on the underlying capability of Node, by providing a web application server framework. This framework provides a wrapper around a lower-level Node interface: giving the developer, a convenient means to handle routing and HTTP operations (such as GET and POST). Express.js makes it easier and smarter to create these services directly by utilizing Node than to (re-)build these services. [14]

In fact, we decided for the routing function with express.js. Routing relates to how the customer requests reply to application endpoints (URIs). [15]. We will extensively explain the functionality of routing and middleware on our project on the following chapter.

2.3.4 jQuery

We will now refer to libraries in JavaScript that assist us develop our code and perform certain operations. jQuery is a quick, compact and richly featured JavaScript library from now on. It makes things lot easier with an easy to use API that works across a range of browser applications, such as HTML, event handling, animation, and Ajax. [16] A combination of versatility and extensibility, jQuery has changed the way of writing code in JavaScript.

2.3.5 Socket.io

Socket.IO is a library that enables real-time, bidirectional and event-based communication between the browser and the server (Figure 8). It consists of:

- a Node.js server: Source | API

- a Javascript client library for the browser (which can be also run from Node.js): Source | API

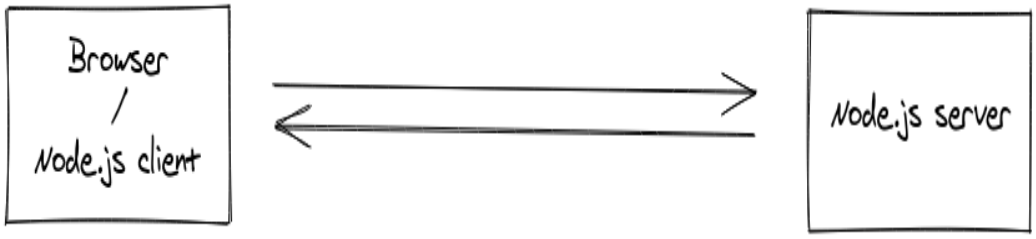


Figure 8:: Representation on how simply socket.io works [17]

If feasible, the client will attempt to connect to WebSocket, and if not, will revert to HTTP long surveys. WebSocket is a communication protocol that gives the browser and server a full-duplex and low-latency link. In the optimum situation, therefore, granted that:

- the browser supports WebSocket (97% of all browsers in 2020)
- there is no element (proxy, firewall, …) preventing WebSocket connections between the client and the server

On Figure 9, we can see that we will have on the client side. To be mentioned, the API on the server-side is similar with some Node.js extensions.

```
const socket = io("ws://localhost:3000");

socket.on("connect", () => {
  // either with send()
  socket.send("Hello!");

  // or with emit() and custom event names
  socket.emit("salutations", "Hello!", { "mr": "john" }, Uint8Array.from([1, 2, 3, 4]));
});

// handle the event sent with socket.send()
socket.on("message", data => {
  console.log(data);
});

// handle the event sent with socket.emit()
socket.on("greetings", (elem1, elem2, elem3) => {
  console.log(elem1, elem2, elem3);
});
```

Figure 9: API on client side on socket.io [17]

Socket.IO is NOT an implementation of WebSocket. While Socket.IO really utilizes WebSocket as a transport, it adds extra information to each message. That is why a WebSocket client cannot connect successfully with a Socket.IO server, nor can a Socket.IO client connect to a single WebSocket server.[17]

Here are the features provided by Socket.IO over plain WebSockets [17]:

- reliability (fallback to HTTP long-polling in case the WebSocket connection cannot be established)
- automatic reconnection
- packet buffering
- acknowledgments
- broadcasting to all clients or to a subset of clients (“Room”)
- multiplexing (what we call “Namespace”)

In this thesis and as a word in socket.io useability the concept of a room is highly crucial. A room is a channel that may be connected to and removed by sockets. It can be used in a subset of customers to broadcast events[18]:

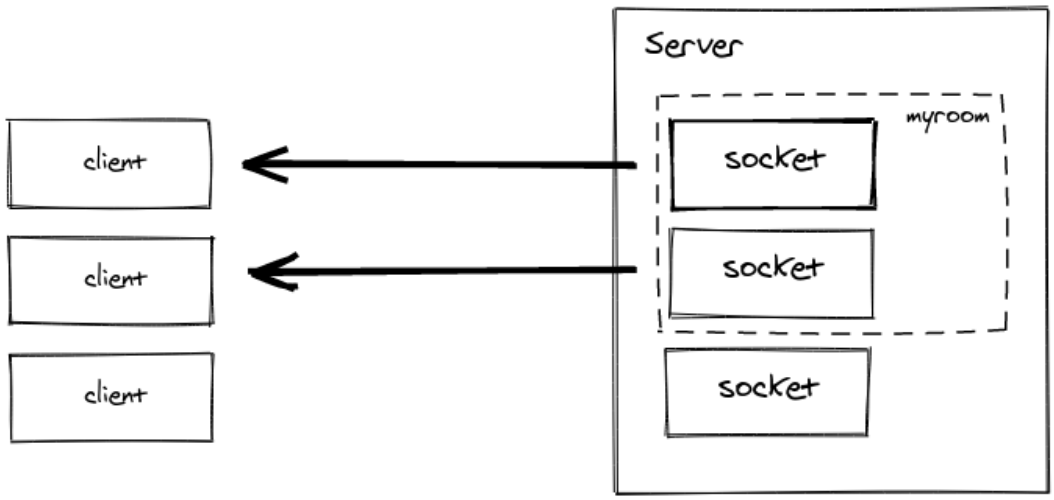


Figure 10: Rooms on socket.io [18]

Please note that rooms are a **server-only** concept (i.e., the client does not have access to the list of rooms it has joined).

2.4 Alternative way of storing data

In a world dominated by SQL relational databases, NoSQL database solutions are becoming increasingly popular. NoSQL databases were created with the goal of providing database solutions for massive amounts of unstructured data..[19]

2.4.1 MongoDB

MongoDB was chosen as our NoSQL database for this project. MongoDB is a document database that combines scalability and flexibility with the querying and indexing capabilities that you want. In MongoDB, a record is a document (Figure 11), which is a data structure made up of field and value pairs. JSON objects are comparable to MongoDB documents. Other documents, arrays, and arrays of documents can be used as field values.[20]. One of the major concepts behind MongoDB's architecture is that actions that may be passed to the client, such as object id creation and other operational solutions, are transferred from the server to the client. MongoDB is a general-purpose database that offers a variety of unique capabilities in addition to the ability to create, read, update, and remove data.[21]

The advantages of using documents are[20]:

- Documents (i.e. objects) correspond to native data types in many programming languages.
- Embedded documents and arrays reduce need for expensive joins.
- Dynamic schema supports fluent polymorphism.
- Indexes support faster queries and can include keys from embedded documents and arrays.
- Support for embedded data models reduces I/O activity on database system

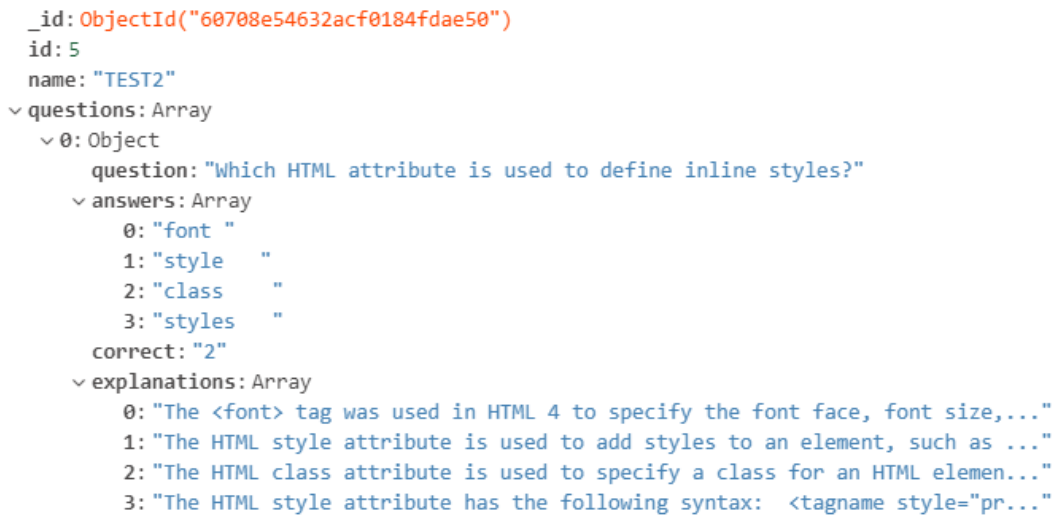


Figure 11: Our first test question mongo DB record on thesis project

After research we gathered some reasons that other developers and authors chose a MongoDB database for their project [19]:

- During the development phase, MongoDB enables flexibility.
- It quickly imports some items from various language difficulties into the database (like java script objects or python objects). It does not require any conversion.
- MongoDB is a database management system that is more responsive. This is the database to use if you want a basic database that responds quickly.
- The map-reduce function may be used to aggregate results for reporting.
- Because it is open source, you may create plugins to make it easier to use.

2.4.2 Mongoose

Mongoose is a tool for modelling MongoDB objects in an asynchronous context. Promises and call-backs are both supported by Mongoose. To represent your application data, Mongoose offers a simple schema-based approach. Out of the box, it features type casting, validation, query creation, business logic hooks, and more. [22]

A Schema is the foundation of Mongoose. Each schema corresponds to a MongoDB collection and determines the document form for that collection.[23]

```
import mongoose from 'mongoose';
const { Schema } = mongoose;

const blogSchema = new Schema({
  title:  String, // String is shorthand for {type: String}
  author: String,
  body:   String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs:  Number
  }
});
```

Figure 12: Our first test question mongo DB record on thesis project

The permitted Schema Types are:

- String
- Number
- Date
- Buffer

- Boolean
- Mixed
- ObjectId
- Array
- Decimal128
- Map

Summing up the way of connection between of MongoDB and our Node.js is through the mongoose.js library that is free to install it through npm, as everything else in our project.

2.5 User Interface (UI) & User Experience (UX)

The user interface is denoted by the acronym UI. It refers to the method in which consumers engage with the mobile app. All of the app's controls, buttons, blocks, and objects are part of the user interface.

The user interface's main goal is to make interaction between the user and the program as simple, entertaining, and effective as possible. Selecting a color, corporate identity, and current design concepts are all part of the UI development process. [24]

The acronym is the term "user experience" refers to how a person feels when using a product. The goal of UX is to create the greatest user experience possible, including simplicity of use and a sense of how the user influences your company's value. Defining how a product runs and serves the needs of users is part of creating a UX. The user experience (UX) must be clear, easy, and user-friendly, with the goal of converting consumers into loyal clients.. [24]

In the modern world, many applications have the same or similar functionality but everyday users, with no related background, can’t understand what makes them different. It’s the UI/UX design factor and the combination of technologies that used to achieve the captivation of the user. The following elements makes a modern application usable:

- Simplified and normal design
- Interactive
- Responsive
- Conventional

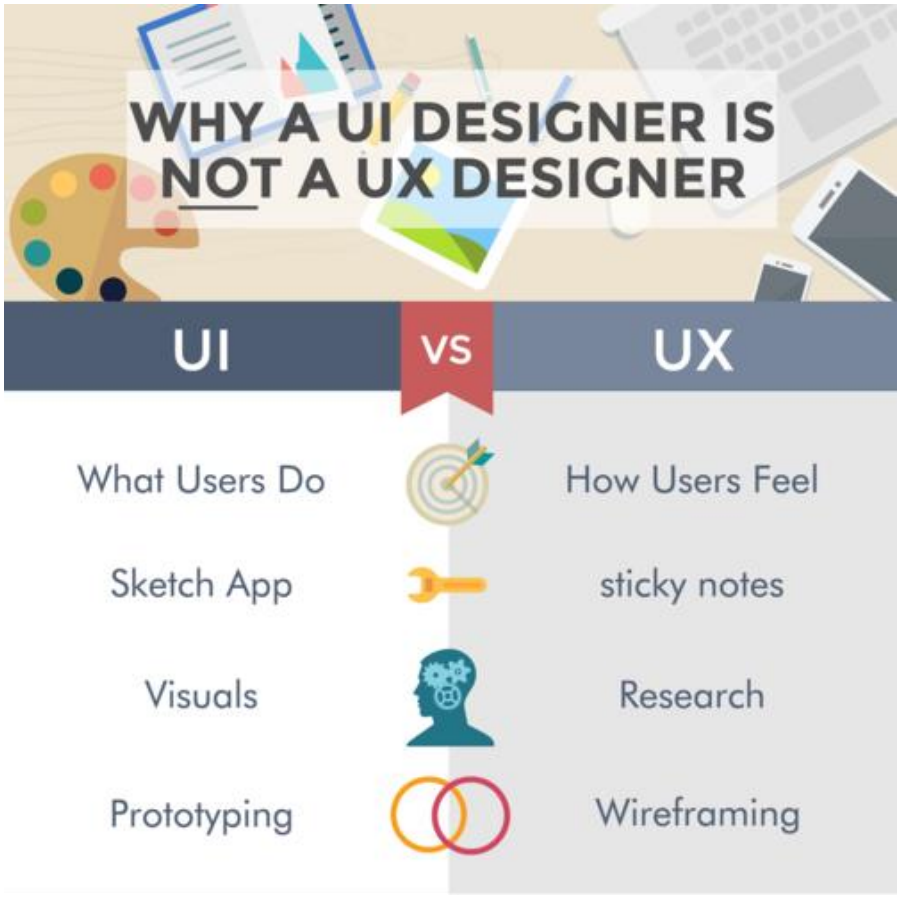


Figure 13: Differences between UI/UX

Looking at Figure 12, we can separate the two terms and specify the role of each one. We can use some key-words to define the role of each term:

- **UI:** color, illustration, photography, typography, icons, visual designs, graphics
- **UX**: usability testing, psychology, interaction design, content strategy, wireframes, prototypes, information architecture, understanding the problem and design and user research

At this point, there is no necessity to analyze more about the UI/UX because we will have time to improve it on the near future. Understanding the importance of these two existing terms is what matters. We will continue by explaining the technologies that we applied for the completion of our "view" side of the project.

2.5.1 HTML – the Structure of a Web Page

Tim Berners-Lee created HTML while working at CERN, and the Mosaic browser built at NCSA popularized it. It exploded in popularity throughout the 1990s, thanks to the rapid rise of the Internet. HTML has been enhanced in a variety of ways throughout the years. The Internet relies on Web page writers and providers adhering to the same HTML rules. This has sparked collaboration on HTML standards. [25]

Hyper-Text Markup Language is the abbreviation for Hyper-Text Markup Language. Now, before we go into the details of how HTML works, let's first define what HTML stands for:

- Hyper-Text refers to hyperlinks that redirect to another page, file, document, or section of a web page from text, an image, or a bookmark.
- A markup language is a computer language that includes tags that specify document components. The headline of a blog, which is usually expressed as a h tag, is an example of a tag. There are other tags, some of which we shall learn about later.[26]

HTML defines the following web page elements: headers, paragraphs, lists, and tables. It also includes code samples and character forms such as boldface. Tags are used to designate certain components and formats inside HTML texts. Each tag has its own name and is separated from the rest of the text by a note that I'll go over later.[27]

2.5.2 CSS-Cascade Style Sheets

Cascading Style Sheets (CSS) allow you to add complex formatting to HTML elements [27], despite the fact that HTML doesn't tell anything about how a page looks when it's viewed. As a result, we must conclude that HTML and CSS are unquestionably partners in crime, with CSS relying on HTML for its existence.

Cascading Style Sheets (www.w3.org/Style/CSS/) gives Web designers more control over layout, something they've been clamouring for years. Surprisingly, CSS has taken a long time to gain popularity. CSS1 was originally published as a standard in late 1996, while CSS2 was released in 1998. [28]

CSS has changed over the years, with the most recent version being CSS3, which has added a lot of new capabilities to existing ones. Just like HTML, CSS has changed over the years, with the most recent version being CSS3, which has introduced a lot of new capabilities to existing ones. CSS3 has been at the vanguard of modern web design and development, alongside HTML5.

- The HTML describes the structure of a Web page, whereas the CSS defines the look and style of the Web page.
- CSS allows you to fine-tune the look of the entire page and each individual element. Weight, color, size, and shadow are examples of these qualities.
- CSS settings for individual HTML tags may be controlled from a single location, affecting the look of the targeted elements.
- Though HTML and CSS are closely related, HTML is made up of Markups, and CSS is made up of rulesets that target particular HTML components.
- When you view a site, the webserver gives you the HTML file as well as the CSS code for that particular file (internal or external).
- The file is interpreted by the Web Browser, which then applies the CSS styles to the HTML components using its own rendering engine. As a result, you see a well-designed homepage with styles, fonts, colors, and shadows, among other things. [29]

2.5.3 Bootstrap

One difficulty with simple HTML design is that the site may seem differently depending on the browser or device used (e.g., mobile, tablet and laptop). As a result, we may need to change the code depending on the browser or device. Using Bootstrap, the problem may be simply fixed. Bootstrap is a web design framework that makes use of HTML, CSS, and JavaScript. All major browsers, such as Firefox, Opera, and Chrome, support it. Furthermore, Bootstrap comes with a number of present classes for quick layouts, such as dropdown buttons, navigation bars, and alerts. Finally, it is responsive in nature, which means that the layout changes automatically depending on the device, such as a mobile phone, laptop, or any other internet-connected device. [30]

Bootstrap was created by Mark Otto and Jacob Thornton at Twitter in the past. In August 2011, it was made available as an open-source project on GitHub.

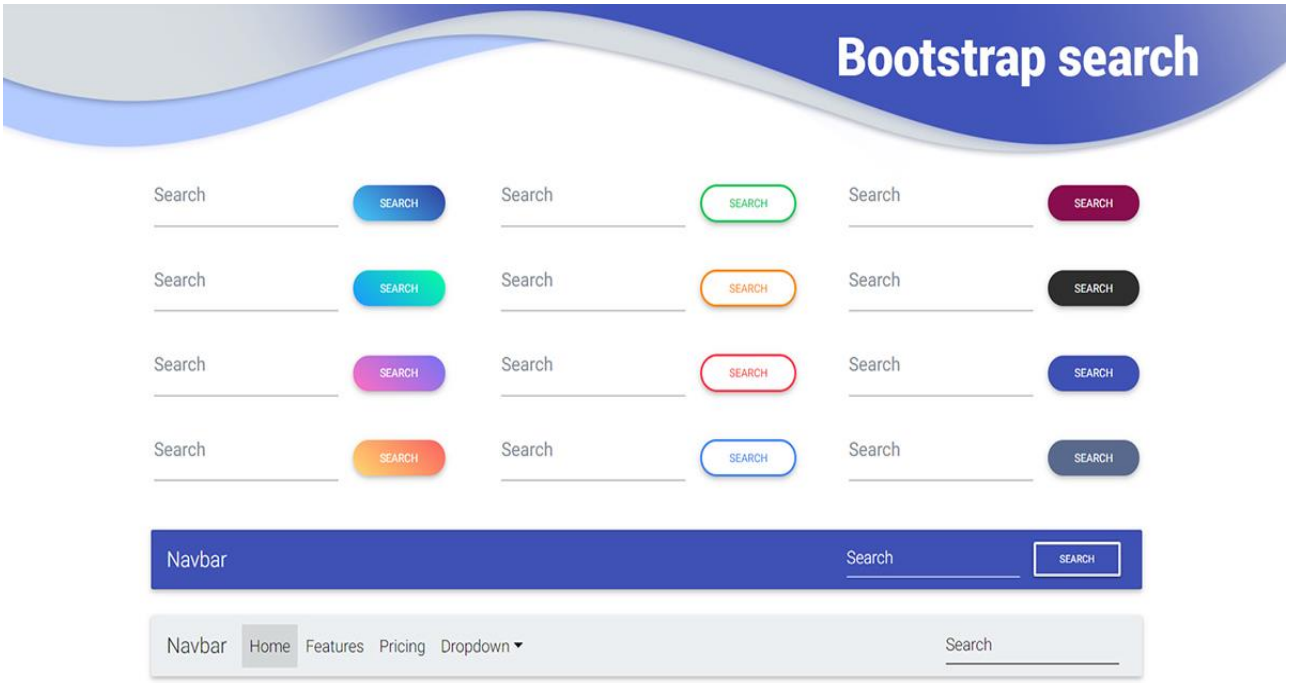


Figure 14: Bootstrap search button and nav bars examples [31]

2.5.4 Overall Project Depiction

On the previous paragraphs we referred on many tools, technologies, libraries and useful methodologies that will provide the help of the completion of our thesis. However, we want the reader of this thesis to have a clear image on how is a structure of a web application. On Figure 15 we can have an overall idea of how the combination of the technologies will be.

On chapter three we're going to attach some of our code, make comments and explain how strategically we worked for the best possible result with zero cost on this academic level thesis.

Figure 15: Depict of our technologies on a real-life example working mall

3. Creation of a Quiz App

Basic concept of our project will be explained shortly so the readers will have a depiction on what we're going to analyze.

Implementation of technologies, combination of methodologies and our designing ideas for the user interface will be presented in this chapter. We will show how we managed to create our server side or let's say backend with node.js and express.js, the connection to MongoDB database and finally the relation to our frontend through socket.io. At the end we will discuss about the view part and how we made some css//bootstrap tricks to improve the appearance of our project.

3.1 Presentation of the main concept of our project

Let's come to a halt and begin showcasing our project's core feature. We need to establish what we accomplished to develop before we enter into the deep understanding of coding section. Of course, at the end of our project, we will have our final outcomes, but for now, it is crucial for the reader to understand the primary principle of our plan.

Case-Based Games Learning (CBGL) is an active learning technique in which students apply knowledge and analytical skills to a real-life complicated issue (contextual) that is relevant to topic teachings. Giving pupils case-based tasks or questions on a regular basis might assist them in solving case-based mathematical issues [32]. Please look at Figure 16 and consider the four different forms of engagement that CBGL stragedy might cause. Let's take a look at some of the benefits and consequences of digital gamification assessment:

- A gamified e-quiz might be effective in evaluating learning performance.
- A gamified e-quiz can be an alternative solution for a formative assessment system.
- Game concepts can be a promising tool to engage students in attractive competition.
- Quiz competitions after lectures motivated students to compete with one another.
- Feeling of fun, interest, enthusiasm, and curiosity are the characteristics of game concepts. [33]

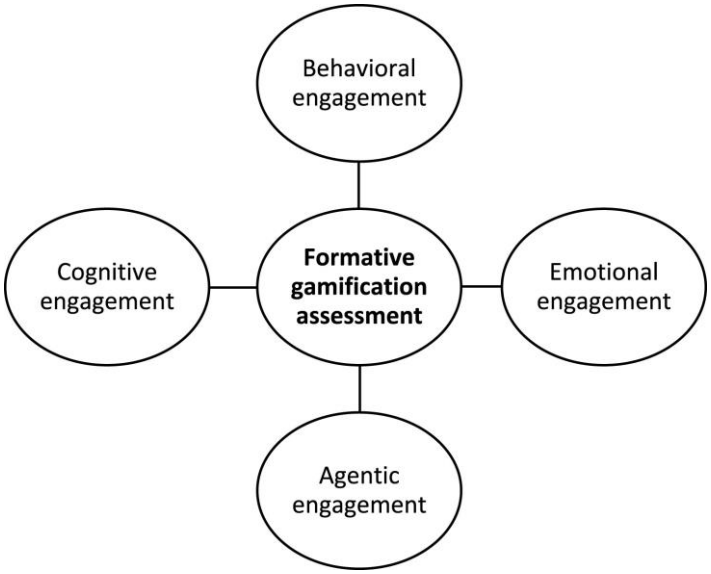


Figure 16: Types of engagement on formative gamification assessment

We need to think like we are in an online classroom like Zoom, WebEx or other educational online video streaming platforms. The main idea is that the professor will create an online quiz, with questions and explanations to the correct and wrong answers. Besides that, the achievement of the participation of the whole class will be through a unique id that will be generated automatically and it will be given to the students so they can participate from the main user interface.

Ensuring that the students will participate to the same quiz by id, then it will have a button only option that will register students' answers. Questions will be visible on the main screen via share screen or in a main screen in case we are in a classroom. After all the students have made their entries, the right answer will be visible to the main screen. For educational purposes, we will also have a twist here. Explanations of each answer will be shown, so the class will discuss their answer. This way learning process will have the feature of validation of knowledge and deep understanding.

Lastly, on the main screen we will have graphs that in the end of all submitted answers will show statistically how many answers were given for each option. We

will also have scores, winners and a timer to complete the functionality and intrigue the competition spirit between students.

On the following figures we summing up every word mentioned in pictures of our quiz app application. Of course, on the process of creating we will have explained the code knowledge part and how we achieved this final result.

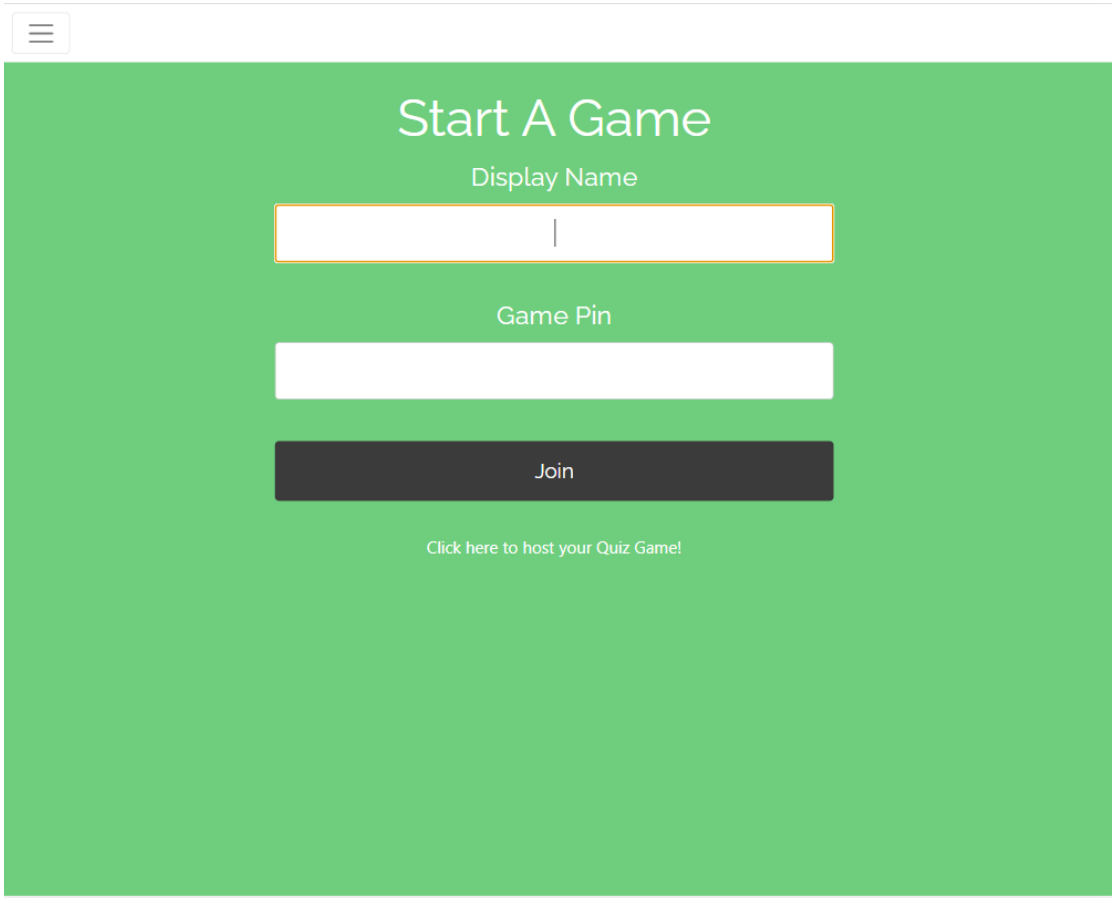


Figure 17: Main page

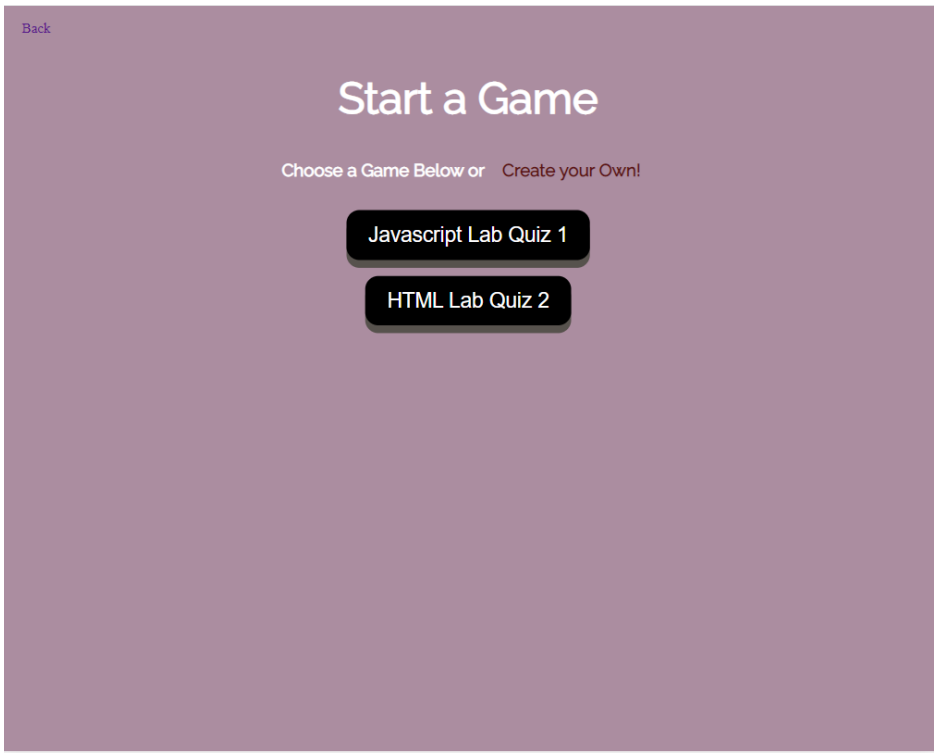


Figure 19: Host an existing game

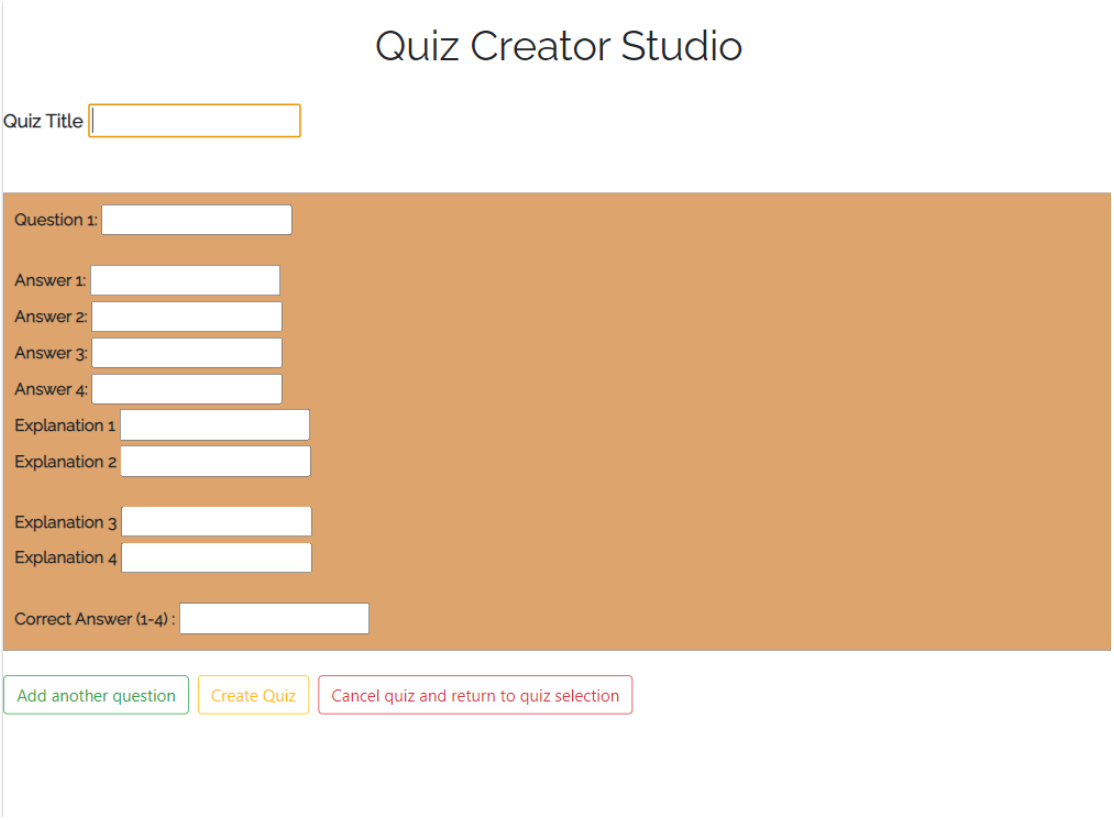


Figure 18: Quiz Creator Studio

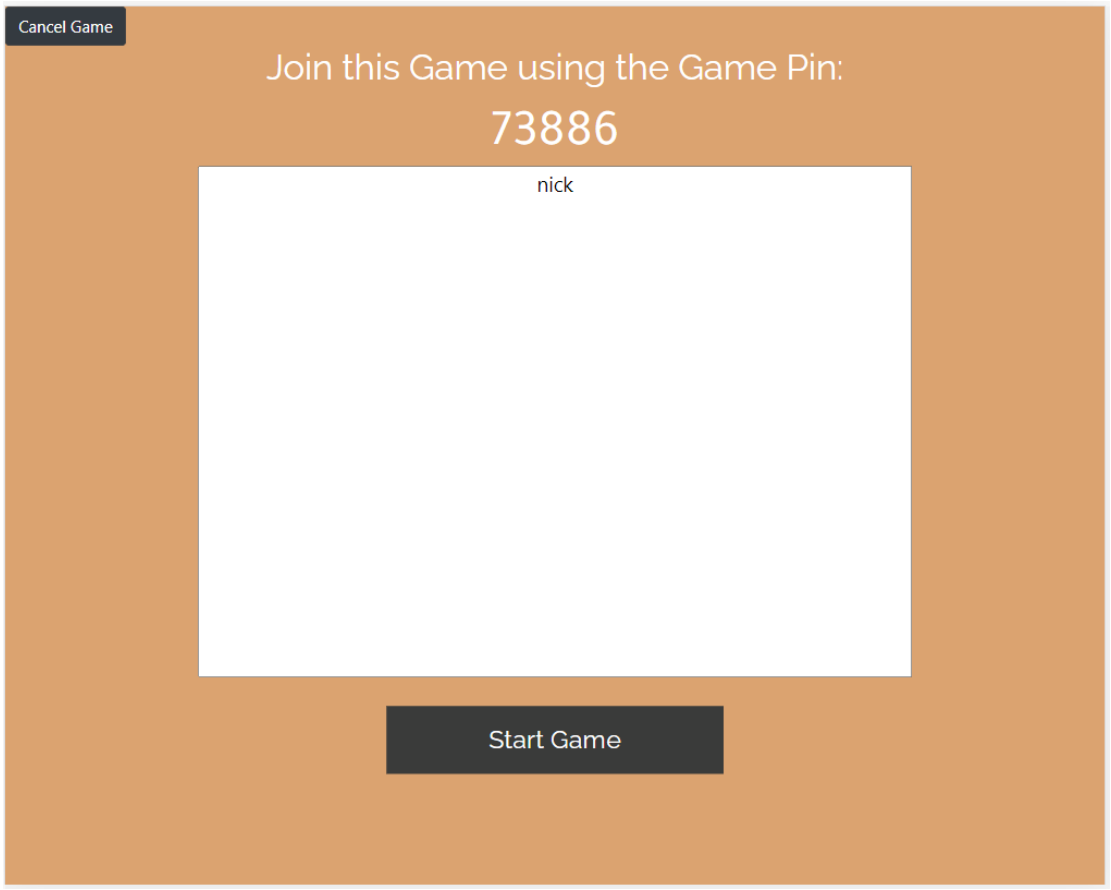


Figure 20:: Lobby of host view

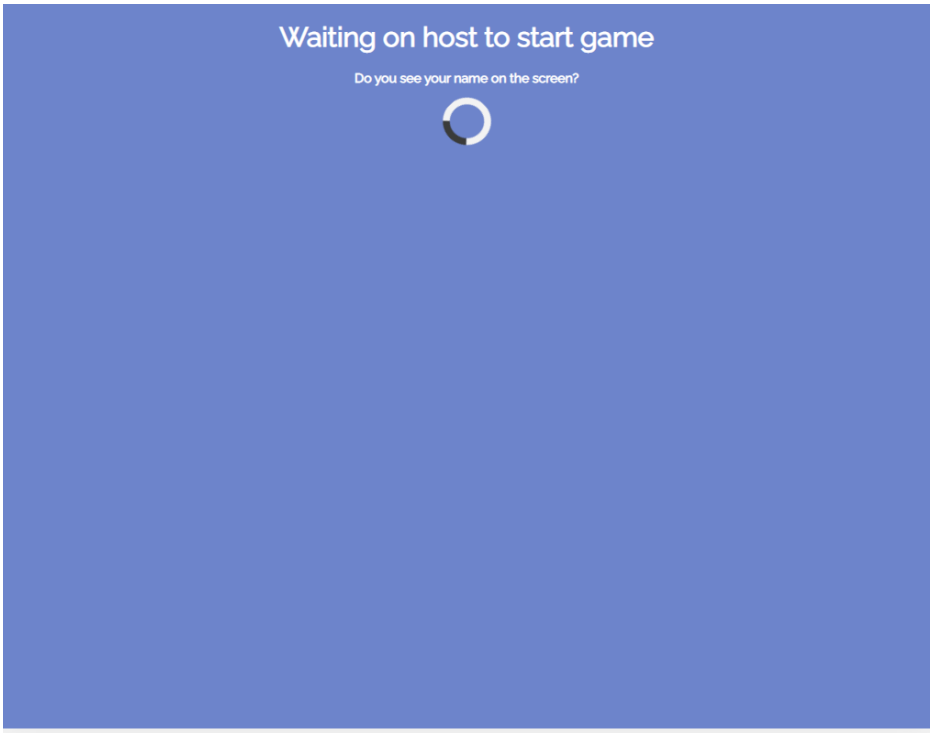


Figure 21: Player waiting host to start game

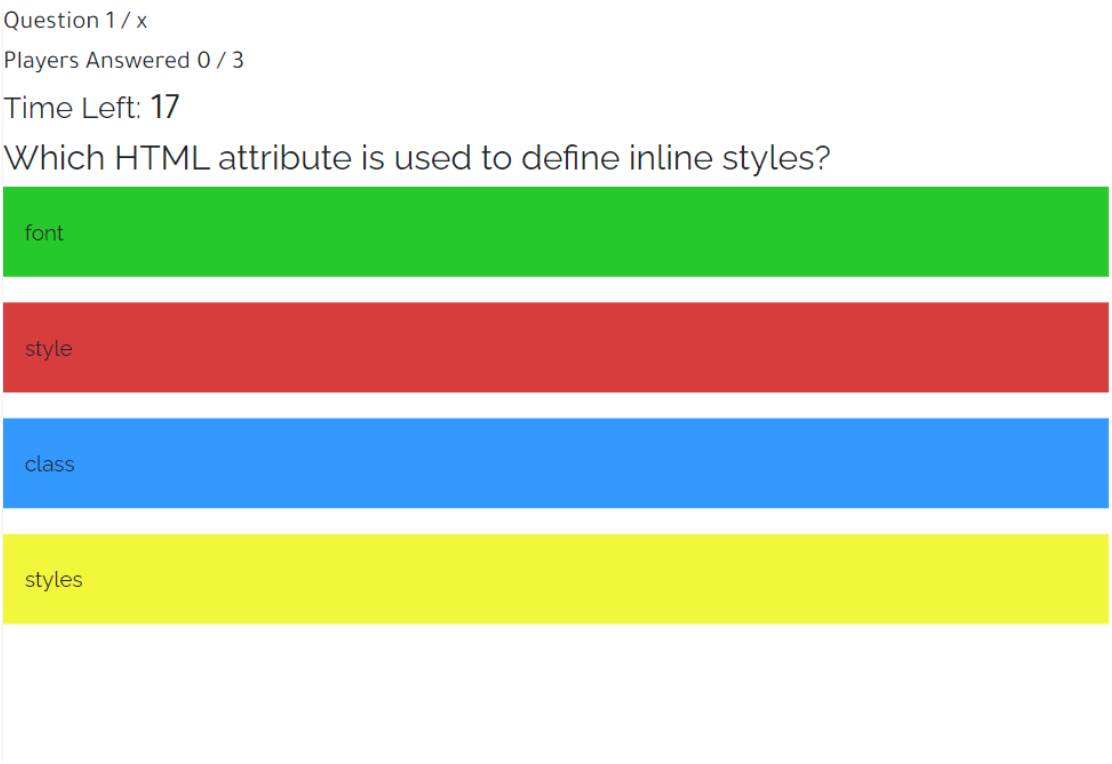


Figure 23: Questions on main screen (before answers submission)

Figure 22: Questions on main screen (after answers submission)

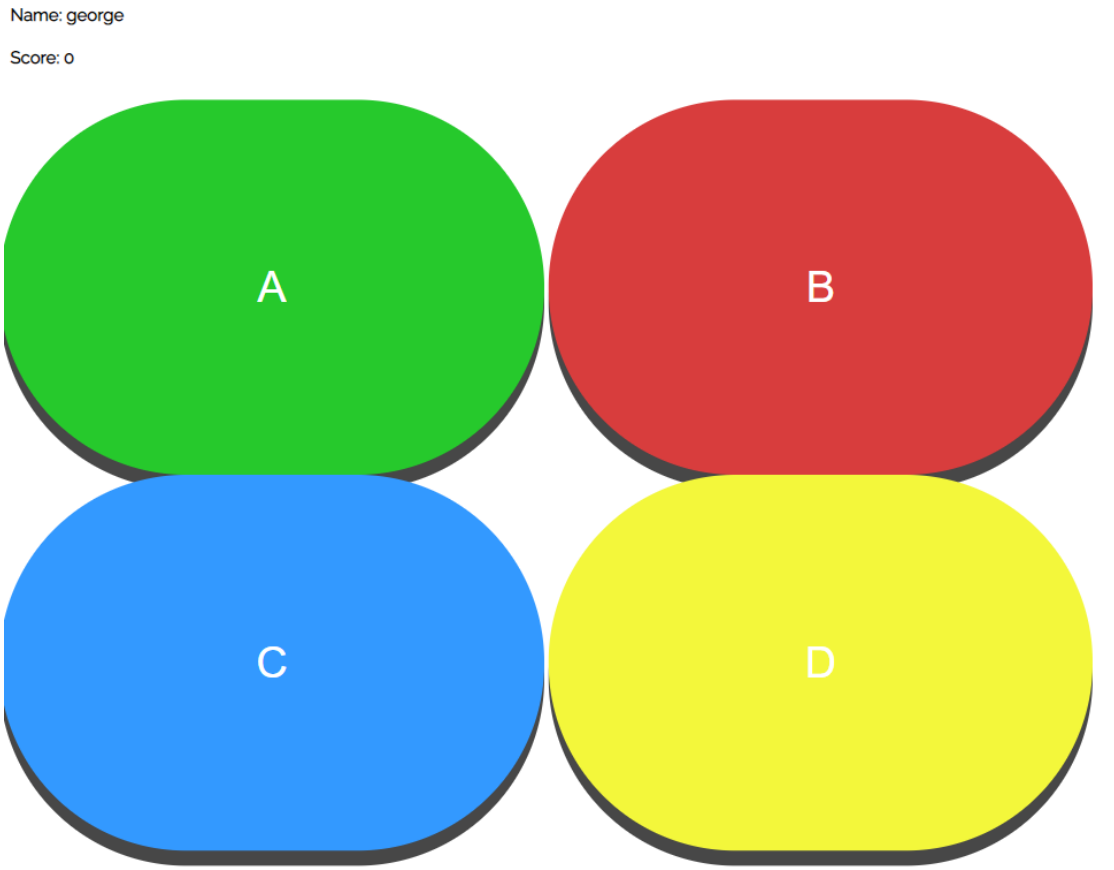


Figure 25: Button options on player view

Figure 24: Game Over page (sample without data)

3.2 Basic Server Setup

The first step is to open a terminal and create a new project directory, then run npm init to start a node application. A npm script is just a set of terminal commands that you can run with npm by entering npm run script name>. There are a few script names that don't need the inclusion of the "run" keyword; one of these script names is "start."[34]. After that, we can finally begin to install our dependencies. Here we used express and socket.io. On the following paragraphs, socket.io will be explained on how it works and its role on our project.

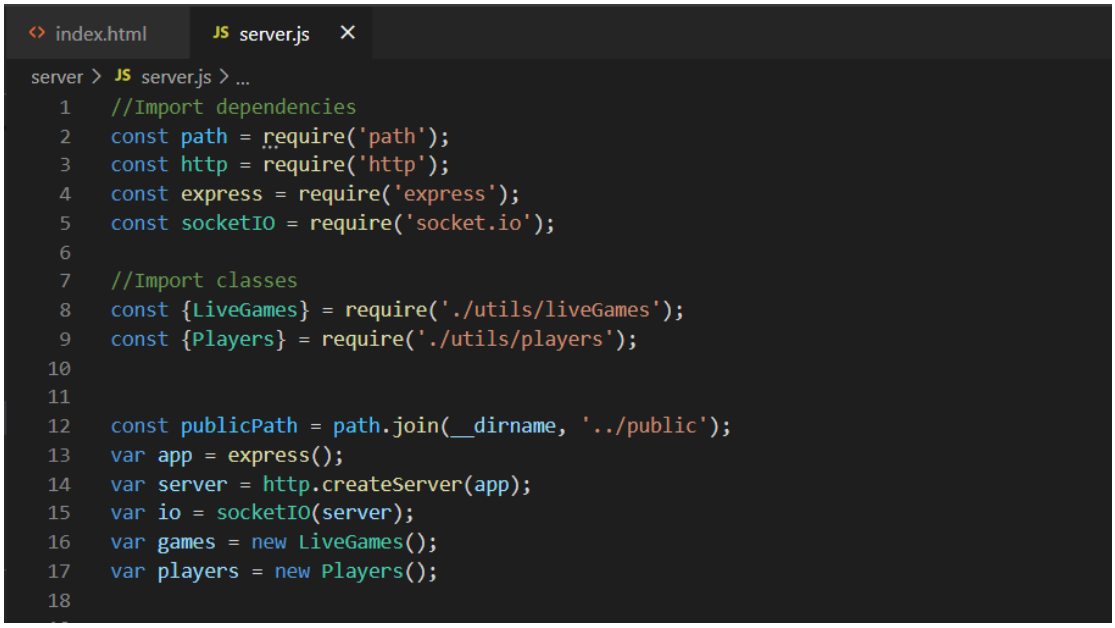


Figure 26: Import of dependencies, http.createServer(app)

Inside our server.js file, we imported the express module and then created an instance of it. On Figure 26 we can observe the dependencies and the classes we created as requirements for our server side. Let's stand on the creation of our HTTP server with a single command as the socket.io documentation suggested to do. Also as we can see on Figure 26 and 27, the server listens to port 3000 and *app.use(express.static(publicPath));* command is the one that with *const publicPath = path.join (__dirname, '../public');* ensured the connection with our public folded. This way can someone connect his static html file to a server side of express.js. However, we had to

implement more to get our final result of the connection between frontend and backend.

```
//Mongodb setup
var MongoClient = require('mongodb').MongoClient;
var mongoose = require('mongoose');
var url = "mongodb://localhost:27017/";



app.use(express.static(publicPath));

//Starting server on port 3000
server.listen(3000, () => {
    console.log("Server started on port 3000");
});
```

Figure 27: MongoDB setup & server port

Finally, for the basic setup of MongoDB we included these three lines of code so we can have access to a local database (Figure 26). More information will be given on how we can create, delete and update data on the following paragraphs.

3.2.1 Importance of socket.io on server side

The library of socket.io helped us complete the idea for interaction between host and clients. So, we have to understand two basic terms. Someone will send a message and someone is always listening to these messages. In socket.io this is translated as:

- socket.emit - This method is responsible for sending messages.
- socket.on - This method is responsible for listening for incoming messages.

Additionally, to this, we have to establish the connection so we can have our connection. It is listening for a ‘connection’ event and will run the provided function

anytime this happens. The setup listens to port 3000 as mentioned above. On figure 27 we can observe that every functionality is included on the brackets of io.on('connection', (socket) => , a function that supports everything about connection.

```
//When a connection to server is made from client
io.on('connection', (socket) => {

    //When host connects for the first time
    socket.on('host-join', (data) =>{

        //Check to see if id passed in url corresponds to id of thesis game in database
        MongoClient.connect(url, function(err, db) {
            if (err) throw err;
            var dbo = db.db("thesisDB");
            var query = { id:  parseInt(data.id)};
            dbo.collection('thesisGames').find(query).toArray(function(err, result){
                if(err) throw err;

                //A game was found with the id passed in url
                if(result[0] !== undefined){
                    var gamePin = Math.floor(Math.random()*90000) + 10000; //new pin for game

                    games.addGame(gamePin, socket.id, false, {playersAnswered: 0, questionLive: false, gameid: data.id, question: 1});
                    //Creates a game with pin and host id

                    var game = games.getGame(socket.id); //Gets the game data

                    socket.join(game.pin);//The host is joining a room based on the pin

                    console.log('Game Created with pin:', game.pin);

                    //Sending game pin to host so they can display it for players to join
                    socket.emit('showGamePin', {
                        pin: game.pin
                    });
                }else{
                    socket.emit('noGameFound');
                }
                db.close();
            });
        });

    });
```

Figure 28: socket.io build on server.js file

Looking at figure 28, and we come to a result that for every usability we wanted for our project, we created a function depended if we want to send or listen a message. Following we will have a table 2 that will sum up the socket.on and socket.emit functions. We have variables of reason of use with an explanation and type for mentioning the sender or listener practicality.

```
//When player connects for the first time
socket.on('player-join', (params) => {

    var gameFound = false; //If a game is found with pin provided by player

    //For each game in the Games class
    for(var i = 0; i < games.games.length; i++){
        //If the pin is equal to one of the game's pin
        if(params.pin == games.games[i].pin){

            console.log('Player connected to game');

            var hostId = games.games[i].hostId; //Get the id of host of game


            players.addPlayer(hostId, socket.id, params.name, {score: 0, answer: 0}); //add player to game

            socket.join(params.pin); //Player is joining room based on pin

            var playersInGame = players.getPlayers(hostId); //Getting all players in game

            io.to(params.pin).emit('updatePlayerLobby', playersInGame);//Sending host player data to display
            gameFound = true; //Game has been found
        }
    }
```

Figure 29: Example of when a player connection on server side

Taking as an example figure 29, we will explain step by step the method of player-join. The rest of the methods server.js were built similar to this. Changes and imports alert messages and dB connections will be explained later separately on another paragraph.

On player-join function on figure 29 we have the usability of a player that connects for the first time. First we create the function through socket.on and we have our Boolean gameFound variable that will help us change the status in case of a game is found with pin provided by player. The following loop runs for each game (figure 28) on the game class (figure 30), checks if the pin is equal to one of the game's pin. Then, it gets the if of the host of the game and with existing characteristics that were defined on LivePlayers class (figure 31), we added the player to the game. Following our code we have socket.join that means the entrance of a player into a room. Room as explained on chapter two isan arbitrary channel that sockets can join and leave.

In our case player is entering the room based on his id (param.id). After all the players participation, we're getting all the players who participated with the same id into the

game. At the same time, we send data to update the lobby of the host with the list of the players. Finally, Boolean variable gameFound turns into true.

We mentioned before LiveGames and now it's time to declare what exactly are there two small classes on liveGames.js and livePlayers.js. Simply, we can tell that it's the description of what actions can do with a player with functions like addPlayer, removePlayer, getPlayer and for get Players. Inside of the functions (figure 30) we have the constructed variables for the player (addPlayer). Also, the filter() method creates a new array with all elements that pass the test implemented by the provided function [35]. In particular, module.exports is used for defining what a module exports and makes available through require() [36]. Using the same logic, we designed the LiveGames class (figure 31).

```
class Players {
    constructor () {
        this.players = [];
    }
    addPlayer(hostId, playerId, name, gameData){
        var player = {hostId, playerId, name, gameData};
        this.players.push(player);
        return player;
    }
    removePlayer(playerId){
        var player = this.getPlayer(playerId);

        if(player){
            this.players = this.players.filter((player) => player.playerId !== playerId);
        }
        return player;
    }
    getPlayer(playerId){
        return this.players.filter((player) => player.playerId === playerId)[0]
    }
    getPlayers(hostId){
        return this.players.filter((player) => player.hostId === hostId);
    }
}

module.exports = {Players};
```

Figure 30: class LivePlayers on server side

Figure 31: liveGames.js class

In short, we gathered on the following table all the socket.io functions that used on the server side with their operability:

Table 2: Functions on server.js file

Function	Type
host-join	*Callback*
Host-join-game	*Callback*
Player-join	*Callback*
Player-join-game	*Callback*
Disconnect	*Callback*
playerAnswer	*Callback*
getScore	*Callback*

Time	*Callback*
TimeUp	*Callback*
nextQuestion	*Callback*
startGame	*Callback*
requestDbName	*Callback*
newQuiz	*Callback*

Accordingly, on table 2 we have a variety of functions that we need to explain their function. Explanation will be provided on the following bullet points for each function:

- **host-join**: when host connects for the first time. It checks if the id passed in URL corresponds to id of thesis game in database. Creates a game with pin and host id in case it finds a game passed with id same as the url. Obtains the game data and the host is joining a room based always on id. Sends game pin to host so they can display it for players to join.
- **host-join-game:** when the host connects from game view. Gets game with old host id. Changes the game host id to new host id. Gets player in game. In case of no game found, redirects the user.
- **player-join:** when player connects for the first time. If game is found with pin provided by player. Looping for each game in game class and checks if the pin is equal to one of the game's pin. Then, it gets the id of host game, player is joining the room based on pin. Sends host player data to display. In case of not matching the pin, player is sent to "join" page.
- **player-join-game:** when player connects from game view. It just updates player id with socket id. Also, we have the case of no player found, or to be specific no game found.
- **disconnects:** When a host or player leaves the site. Finding game with socket.id. If a game hosted by that id is found, the socket disconnected is a host. Checks to see if host was disconnected or sent to game view. Removes the game from game class. For each player in the game, removes it from player class and sent player back to "join" screen. Socket is leaving room. Another case is that there is no game, so it is player disconnection. Get all players socket.id and if someone has the one id, gets the id of the host of the

game, obtains game data based on host id and the pin of the game. Then after checking the status of the game, removes player from player class and the rest of the players remains on the lobby. Updated the lobby data. Player has left the room.

- **player-Answer:** Set's data in player class to answer from player. Checks if the question is still live, compare player's answer with the correct answer. Examine if all players answered and updates host screen of players answered.
- **get-Score:** sets the player data of score.
- **time:** default time for the players to answer
- **time-Up:** Countdown for the question to get answered and covers the question that is over and how to emit the answer based on game pin.
- **next-Question:** Reset current answer to 0. Obtains and loads data from thesisDb. as main characteristic of the function. Also have the ranking of the players and compares their score values. In case of Game Over, emits the scores based on the comparison that made. Also, emits the next question player based on the game pin.
- **startGame:** When the host starts the game. Gets the game based on socket.id. Tells the player and host that the game started.
- **requestDbNames:** Gives user game names data.
- **newQuiz:** a database call to thesisDB so we can have a new Quiz.

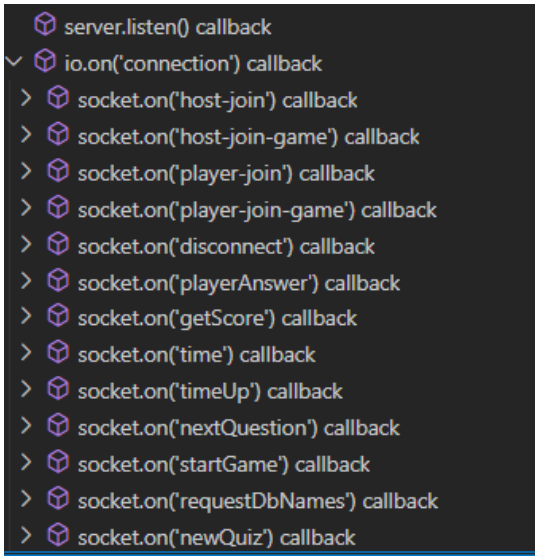


Figure 32: Visual Studio depiction help outline

As we can see on the figure 33, visual studio code helps us have a sum up of our outline on a section so we can browse and edit the function that we want any time that was necessary.

3.3 Database Setup

All this information about the questions and their explanations needs to be stored and restored every time we need to re-play the quiz. Of course, the pin for the game will be different but the questions and explanations can be the same. The same set of data will be visible to the players or students in our case for future use. This is helpful because in case of a professor that teaches the same class in too many laboratory subclasses, it saves time of his schedule.

Data is saved in MongoDB as documents. These documents are saved in JSON (JavaScript Object Notation) format in MongoDB. Embedded fields in JSON documents allow related data and lists of data to be kept in the document rather than in an external table..[37]

Let's start with the basics of a mongoDB setup on visual studio code through npm. We can control our database from our application through the offered UI (figure 33) inside of the application but for experienced users or professors familiar with development, we can manage our data from our visual studio code (figure 34) in case we are the owners or have access to main code.

Additionally, we can create our dataset and import it as json or csv file if we don't want to use the one-by-one option from the offered UI. This is optional, but the class material can be prepared from home with the use of VPN for educational purposes. On figure 35 we have the official MongoDB environment connected to our localhost port 27017.

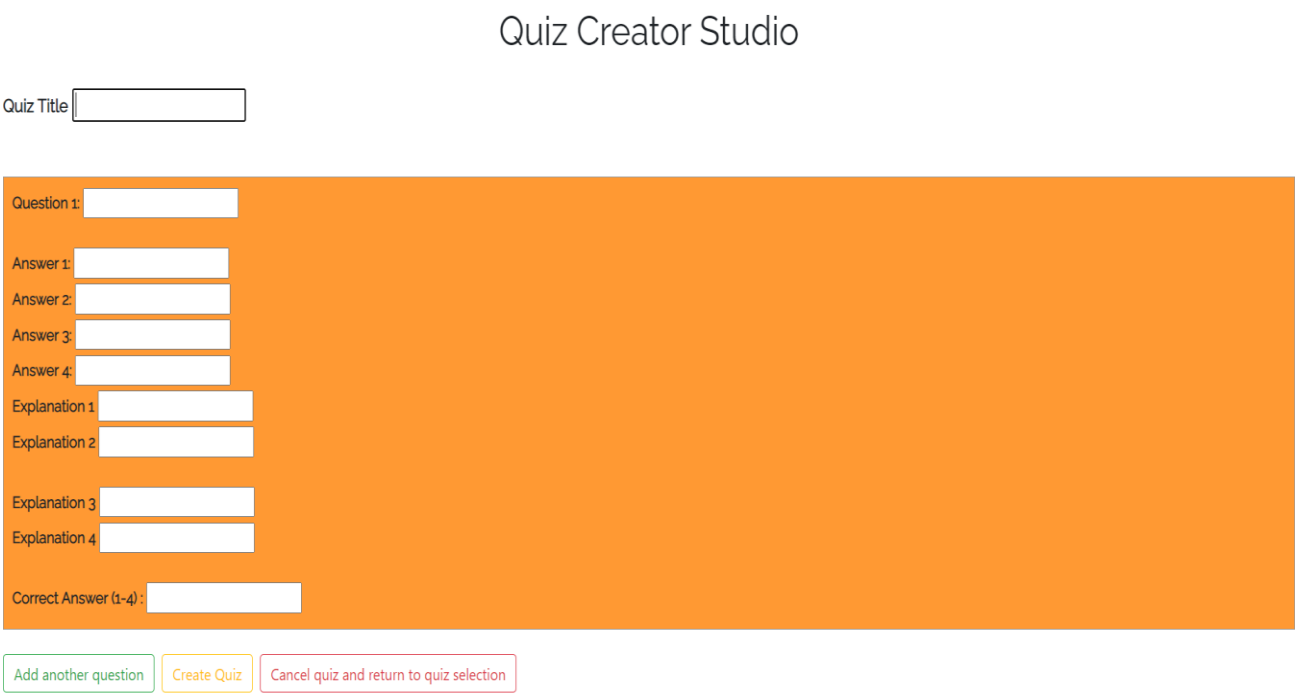


Figure 33: UI for creating a quiz

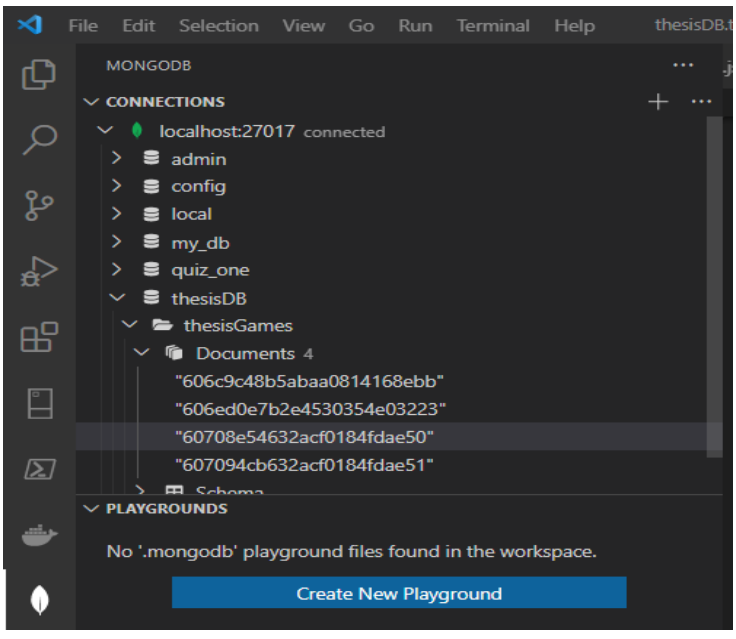


Figure 34: VS code MongoDB edit section

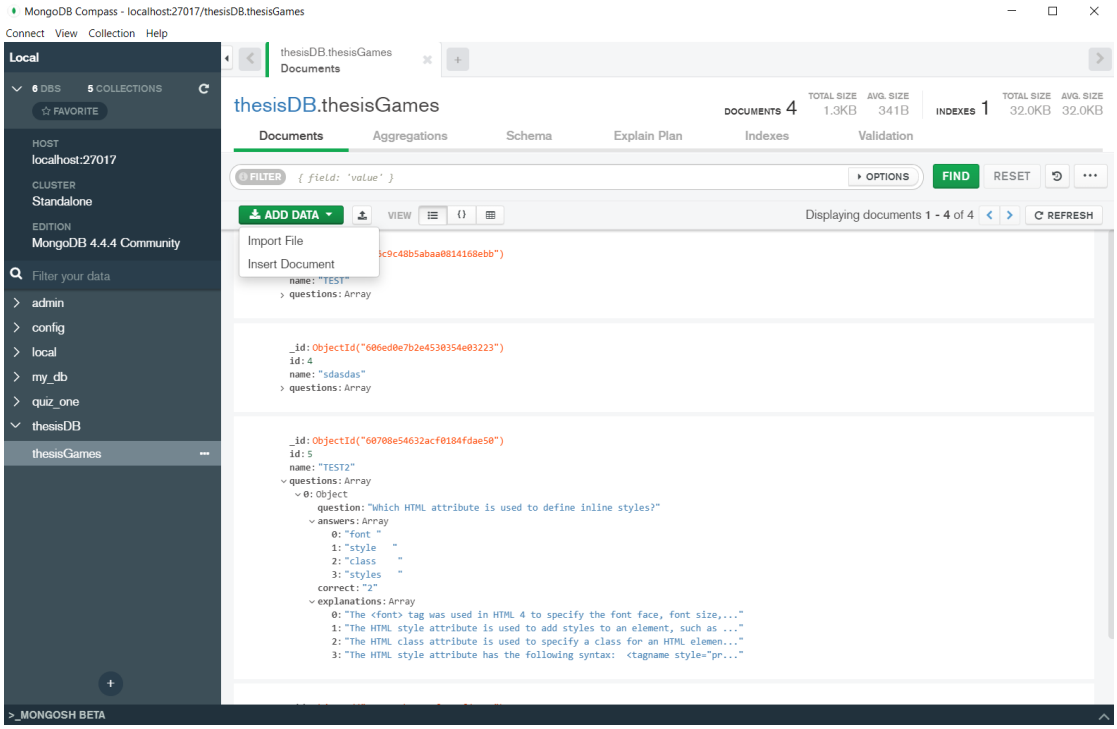


Figure 35: Official MongoDB Compass edit section

3.3.1 Schema

A document schema is a JSON object that allows you to define the shape and content of documents and embedded documents in a collection. You can use a schema to require a specific set of fields, configure the content of a field, or to validate changes to a document based on its beginning and ending states.[38] On MongoDB whenever we create a record on our database, an _id autogenerating number is created and it can be used for functionality reasons and the identification of our record. The _id field If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document. In the example above no _id field was specified, and as you can see from the result object, MongoDB assigned a unique _id for each document. If you *do* specify the _id field, the value must be unique for each document.[39]

Our thesis project has a specific schema as far as concerns the quiz creation. On the following table 3 we can observer the fields and types for the schema:

Table 3: Fields and types of our MongoDB schema

Field	Type
Id	*Integer*
Name	*String*
Questions	*Array*
Question	*Object*
Answers	*Array*
Explanations	*Array*
Correct	*Integer*

3.3.2 Database Functionality

Starting with the simplest thing to do to import and connect your server side to a MongoDB database. On figure 26 we observe that a few lines of code is only necessary for the existence of a non-relational database like Mongo.

Apart from that, we will explain how we implement some lines of code on our socket.io functions so we can create our quiz questions (figure 35). It's fundamental to mention that there is no need to add a delete option here as Mongo offers on his environment control center, all the proper options for clearing data or deleting the whole database if you ever wanted to make a change like that.

On table 2 we referred and described every socket.on function on our server.js file. Now we will focus on how we can create a record on our thesisDB. Combining the function of our server.js with the required lines of code on the quizCreator.js in our public or let's say the frontend part. As for the frontend setup, we will have our explanations later on this chapter. In the depiction of our code in figure 36 and figure 26 we can see the quizCreator.js with the updateDatabase() function and how it

triggers an emit for socket in the end of the function. It sends the "message" to the listener and we have the required result of updating and creating our record.

```
socket.on('newQuiz', function(data){
    MongoClient.connect(url, function(err, db){
        if (err) throw err;
        var dbo = db.db('thesisDB');
        dbo.collection('thesisGames').find({}).toArray(function(err, result){
            if(err) throw err;
            var num = Object.keys(result).length;
            if(num == 0){
                data.id = 1
                num = 1
            }else{
                data.id = result[num -1 ].id + 1;
            }
            var game = data;
            dbo.collection("thesisGames").insertOne(game, function(err, res) {
                if (err) throw err;
                db.close();
            });
            db.close();
            socket.emit('startGameFromCreator', num);
        });

    });
```

Figure 36: newQuiz function with thesisDB on server.js

```
function updateDatabase(){
    var questions = [];
    var name = document.getElementById('name').value;
    for(var i = 1; i <= questionNum; i++){
        var question = document.getElementById('q' + i).value;
        var answer1 = document.getElementById(i + 'a1').value;
        var answer2 = document.getElementById(i + 'a2').value;
        var answer3 = document.getElementById(i + 'a3').value;
        var answer4 = document.getElementById(i + 'a4').value;
        var correct = document.getElementById('correct' + i).value;
        var explanation1 = document.getElementById(i + 'e1').value;
        var explanation2 = document.getElementById(i + 'e2').value;
        var explanation3 = document.getElementById(i + 'e3').value;
        var explanation4 = document.getElementById(i + 'e4').value;

        var explanations = [explanation1, explanation2, explanation3, explanation4];
        var answers = [answer1, answer2, answer3, answer4];
        questions.push({"question": question, "answers": answers, "correct": correct, "explanations": explanations})

    }

    var quiz = {id: 0, "name": name, "questions": questions};
    socket.emit('newQuiz', quiz);
}
```

Figure 37: updateDatabase() on frontend side on quizCreator.js file

Furthermore, we can stand on the usability of appearing our questions from the database to our quiz. On server.js, our file that controls everything, exists a socket function called nextQuestion() that provides the data for the following question in our quiz application. Always working with the game id, the database starts var dbo = db.db('thesisDB'); and after that we state our query and how we want to absorb our data. On figure 37, we can observe every detail on how we synchronize and how we made the right checks so we can have our data to the game.

```
MongoClient.connect(url, function(err, db){
        if (err) throw err;

        var dbo = db.db('thesisDB');
        var query = { id:  parseInt(gameid)};
        dbo.collection("thesisGames").find(query).toArray(function(err, res) {
            if (err) throw err;

            if(res[0].questions.length >= game.gameData.question){
                var questionNum = game.gameData.question;
                questionNum = questionNum - 1;
                var question = res[0].questions[questionNum].question;
                var answer1 = res[0].questions[questionNum].answers[0];
                var answer2 = res[0].questions[questionNum].answers[1];
                var answer3 = res[0].questions[questionNum].answers[2];
                var answer4 = res[0].questions[questionNum].answers[3];
                var explanation1 = res[0].questions[questionNum].explanations[0];
                var explanation2 = res[0].questions[questionNum].explanations[1];
                var explanation3 = res[0].questions[questionNum].explanations[2];
                var explanation4 = res[0].questions[questionNum].explanations[3];
                var correctAnswer = res[0].questions[questionNum].correct;

                socket.emit('gameQuestions', {
                    q1: question,
                    a1: answer1,
                    a2: answer2,
                    a3: answer3,
                    a4: answer4,
                    e1: explanation1,
                    e2: explanation2,
                    e3: explanation3,
                    e4: explanation4,
                    correct: correctAnswer,
                    playersInGame: playerData.length
                });
```

Figure 38: example of database functionality on nextQuestion() on server.js

3.4 Frontend setup

As mentioned before on server.js exists a line of code, *const publicPath = path.join(__dirname, '../public');* , that ensures the connection our frontend with

backend. Every html file is categorized in files and called index as this command reads only index.html files. Otherwise, there is no recognition from the localhost what we want to bring to the surface. On Figure 39, we have the overview of the files and how we structured them based on our functionality, following always a process that will allow us to make changes in the future.

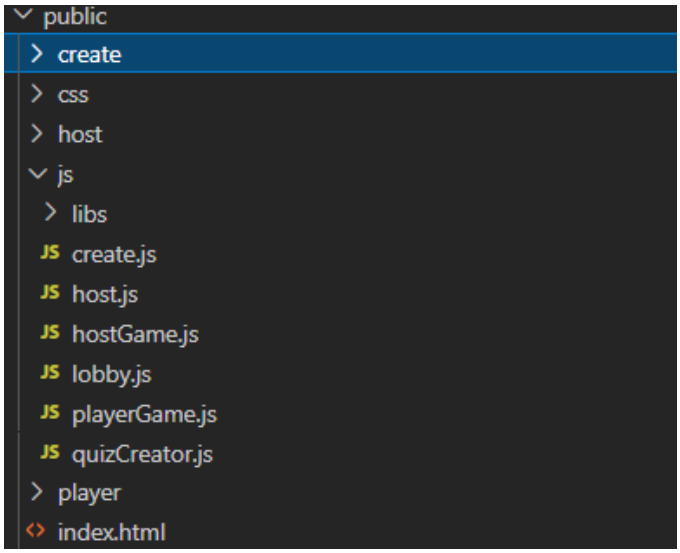


Figure 39: Overview of files of the frontend part of the project

Folders like create, host, player offers the view part of every page that concerns each category. On main index.html we have our main redirection or start page. Lastly, we have the js folder that includes libraries and that trigger or create control action in our project. The folder with java script files works in a combination with the backend so we can give action to the user interaction. As we can see in our project so far we have a model with data that is Mongo on server.js with their schema files, our view part on the folders of create, host and player and our controller that is a combination of backend and frontend. It's not the regular depiction of MVC but it stands close to the idea, because we separate every part and it's easy to make changes for future update.

3.4.1 JavaScript functionality on our frontend side

In this paragraph we will shortly explain the functionality of the js folder (figure 39), always have in mind what we explained for our server setup. On libs file, we have a

jquery functionality and a jQuery.deparam, the opposite of jQuery param. Creates an object of query string parameters [40] .

Starting with the create.js file, we have our socket.on gameNamesData() function that gives us the create page that have the existing games and the possibility of creating a new one. It gets triggered on main on create/index.html. Declaring that this is not the quiz creator page, but a previous page before we get in to creation of questions.

Secondly, we have the quizCreator.js, that has all the functionality that includes a part of inserting new data to our database. Updating our database or having the functionality of button option to add question, cancel quiz or create the quiz. On figure 32 we have a representation of the UI and might help to give an image to our description.

The host plays a sever role in our project and we need to clarify the host.js functionality. On host.js we have the following:

- When host connects to server (socket.on, connect), tell server that it is host connection (socket.emit, host-join).
- It shows the game pin (socket.on, showGamePin).
- Adds player's name to screen and updates player count (socket.on, updatePlayerLobby).
- Tell server to start game if button is clicked and redirects based on id when server start the game

The hostGame.js provides the usability on what we have while the game is on progress. On hostGame.js we have the following:

- When host connects to server (socket.on, connect), tell server that it is host connection from game view (socket.emit, host-join-game).
- On socket.on function gameQuestions() we have the load of the questions and the explanations, but with a trick of style we have it as non-visible. We declare it as style.display = ‘none’.

- On socket.on function updatePlayersAnswered(), we update the answers the submission of our players.
- On socket.on function questionOver(), we decide what explanation will appear under of every answer depending the correct answer. Shows user correct answer with effects on elements. Also, we create graphs based on values of the answers.
- On function nextQuestion(), we have the load of the next question that emit through socket.emit . We disappear the current elements with the trick of style.display, then we tell the server to start a new question.
- On socket on function GameOver() we the finale of our work and with style.display we disappear the elements and appear the 5 winners. On a class the 5 best will be visible for public view
- Functions like getTime() or Timer() completes the functionality as every question have a specific time declared to get answered.

The playerGame.js is responsible for the answers of the user. Submissions of an answer and the result and get the right messages for the answers. The user gets notified if the answer is correct or incorrect and take messages so they can follow the process of the game.

The lobby.js file is the screen that controls and maintain the functionality of multiplayers. Shortly includes the following:

- Socket.on (connect), when players connect to server and after it gets data from url, it tells the server that it is player connection.
- Socket.on (noGameFound), it boots player back to join screen if game pin has no match.
- Socket.on(hostDisconnect), if the host disconnects, then the player is booted to main screen.
- Socket.on(gameStartedPlayer), when the host clicks start game, the player screen changes.

3.4.2 User Interface setup

The user interface is very important because reduces the risks of misguidance of the user. The user interface of apps is important because that is how people interact with your product to achieve their needs and goals. It's not all about the UI. Functionality and good visuals are important in establishing your brand and relationship with customers [41].

We will present the technologies, functions and elements that helped to improve our interface. Of course, everything will be presented through code, because this is on what we emphasize in this chapter.

Referring on our structure of interface, we used HTML that also provides sometimes the key to the functionality of the frontend setup. Continuing with used CSS to add the desired font, background color, text color, padding, spacing, align and more characteristics that can be added on our CSS file. The interface completed with the setup of bootstrap. On the following paragraphs we will focus on achievements such as responsiveness, background color change and minimal design with buttons and navigation bars. Finally, color theory will be analyzed, as it helped us taking the app interface to another level.

Responsive design is a graphic user interface (GUI) design approach used to create content that adjusts smoothly to various screen sizes. Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones) [42].

Ensuring the responsiveness, we added the following meta command to every index.html:

```
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

Furthermore, we added the bootstrap setup so we can add a navigation guide to the user and the buttons to be interactive:

```
  <button onclick = "addQuestion()" class="btn btn-outline-
success mr-1 "> Add another question</button>
```

```
 <nav class="navbar sticky-top navbar-
light" style="background-color: #ffffff;">
<button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarTogglerDemo01" aria-
controls="navbarTogglerDemo01" aria-expanded="false" aria-
label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
              </button>
              <div class="collapse navbar-
collapse" id="navbarTogglerDemo01">
                <a class="navbar-
brand" href="/create/">Create a Quiz</a>
                <ul class="navbar-nav mr-auto mt-2 mt-lg-0">
                  <li class="nav-item active">
                    <a class="nav-
link" href="#">How to Play <span class="sr-
only">(current)</span></a>
                  </li>
                  <li class="nav-item">
                    <a class="nav-link" href="#">About</a>
                  </li>
                </ul>
                </div>
          </nav>
```

Adding more information, we can include some details that we put on our user interface:

- Changing background color and gives a fluid sensation to the application. On figure 40, we can the two functions that gives the result of the background color change.

```
function randomColor(){

    var colors = ['#4CAF50', '#f94a1e', '#3399ff', '#ff9933'];
    var randomNum = Math.floor(Math.random() * 4);
    return colors[randomNum];
}

function setBGColor(){
    var randColor = randomColor();
    document.getElementById('question-field').style.backgroundColor = randColor;
}
```

Figure 40: Functions that provides the background colour swapping.

.

In conclusion, bootstrap main advantages are that it is responsive by design, it maintains wide browser compatibility, it offers consistent design by using re-usable components, and it is very easy to use and quick to learn. From my experience, I totally recommend the use of this technology to beginners' developers on an academic level.

4. Testing and Risk Analysis

On this chapter we will demonstrate the testing we made to avoid user errors. We wanted to focus on user errors and how everything will be adaptable knowledge to our users. We don't want to stress our users with options and capabilities because in the end of the day, it's a tool for educational purposes.

Secondly, we will present our risk analysis management results and what are the risks for making user errors and what are the impacts to our thesis project. Hazards will be named and analyzed. Risk analysis is something that will upgrade our project and it will be a good preparation for real-life projects on a working company.

4.1 Testing responsiveness

Nowadays, requirements of building and has grown and the necessities are even larger. New devices added to our everyday life. We wanted to cover every device width, every screen that our game can be played during a class. Fortunately, Chrome

has made it easier to test responsive design through Developer Tools by integrating a powerful emulation feature: device mode. Device mode can emulate a mobile environment to test a website's responsiveness in different devices. ... You can also test your site with different throttling options. On the following image (figure 41) we will have three different device screens, iPhone 6/7/8 models, pad and our computer screen.

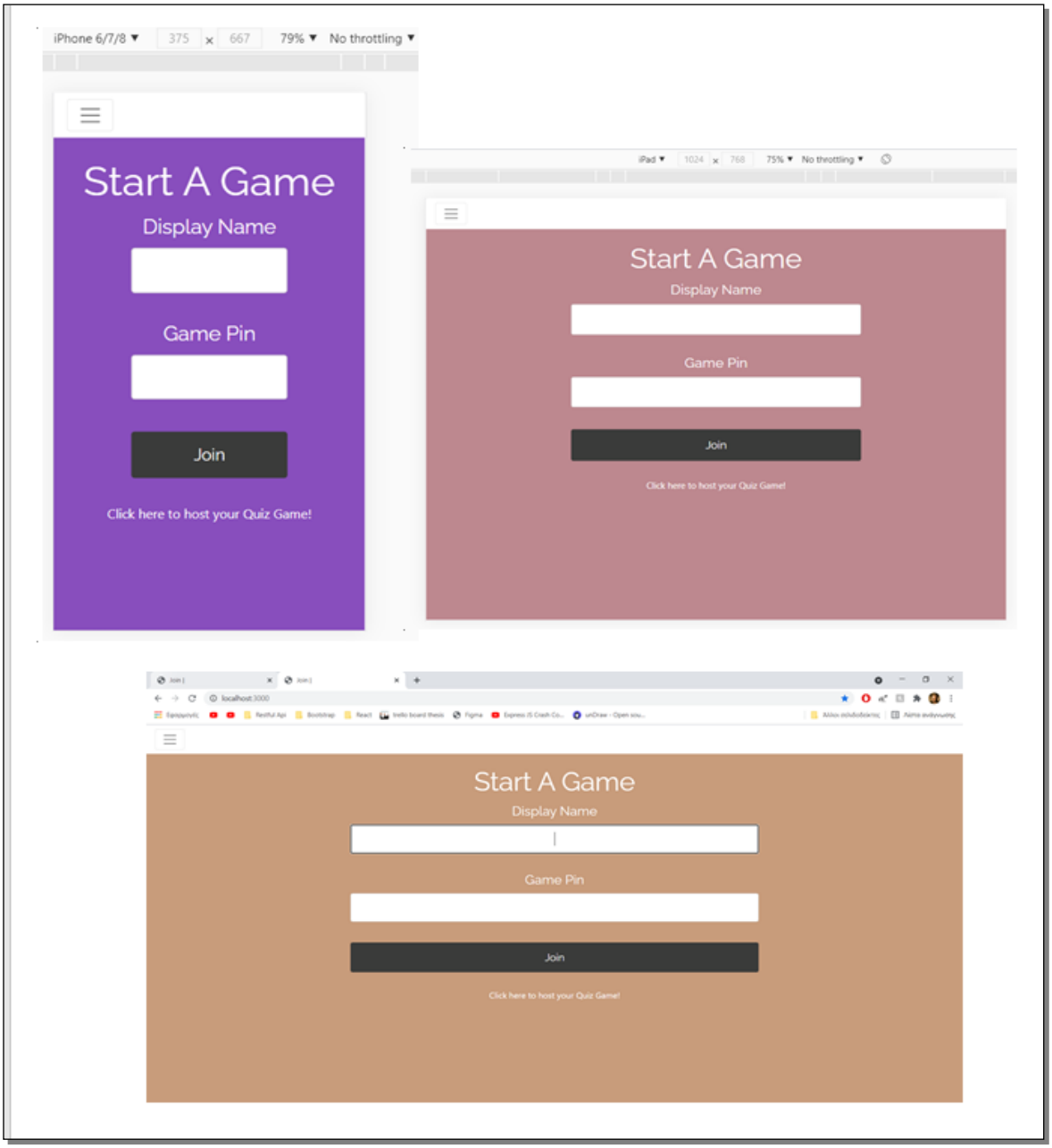


Figure 41: UI responsiveness on iPhone, iPad and pc computer screen

The biggest challenge is to test the limits of your responsiveness by testing in to an old small device, as users may not have the comfort to own an expensive device. We used developer dev tools and we set our sample device as iphone 5/SE with resolution 328px X 568px and the result was surprising good.

Closing we have to admit that for every product digital or nor we have the follow principals for the user interface:

- Valuable- It doesn't matter if an app is functional and beautiful, it first and foremost needs to be addressing a user need or else it's useless. If people want to use your product, it comes from understanding their problems and solving them. Don't think about the details and intricacies first until you determine specific problem and user base.
- Accessible- If you want your app to reach lots of people, you can't be thinking about edge cases.
- Intuitive- When people interact with apps, they shouldn't think about what they are doing, but to just do what they need to do in achieving their goals.
- Invisible- Following intuition, good design is when users don't even need to think about
- Beautiful- If you have an app that is based on a core need and is functional, you are 50% there. If you want to have a competitive edge over other products and have your customers invested in your brand, the app needs to be elegantly designed. What they are using, let alone how it is designed.

4.2 Theory of Color

Based on the user interface, we will get deeper on the design of UI world as we expand our knowledge and upgrade our project with the theory of color. When someone studies software engineer, it's almost unbelievable for the students the use of color theory. However, colors affect the market we work as it has connection with the human interaction. When someone is using your app, his emotions get triggered and psychology as we know is a severe factor for a successful application.

We buy something based on our favorite color, we ask someone to describe something to us based on color, and there are numerous more examples in our everyday lives that demonstrate the importance of color as a psychological aspect. Clarity, excellent layout, excellent visual hierarchy, appropriate use of color, and simple navigation are the five principles of compelling web design. Balance and color, on the other hand, are design principles and elements that impact harmony in a sequential manner. When a website adopts a pleasing color scheme, it engages users and creates a pleasing visual experience. Otherwise, if the website design isn't balanced, it might become disorganized and monotonous..[43]

As color for triggers emotions, we want to have red to give attention when something needs to be cancelled or green that expressed stability and user feels safety to continue the creation part.

On the following figure 42 we summing some of the basic colors and what emotions or what is the interpretation of each color in life.

TABLE I
COLOR EMOTION ON WEB UI DESIGN.

Color	Promotes
Red	Importance, power, youth
Orange	Uniqueness, friendliness, arise energy and a sensation of movement
Yellow	Happiness, enthusiasm, antiquity (darker shades)
Green	Growth, stability, financial themes, and environmental themes
Blue	Safety, calm, openness (lighter shades), strength and reliability (darker shades)
Purple	Luxury, romance (lighter shades), mystery (darker shades)
Black	Power, edginess, sophisticated and timeless
White	Simplicity, cleanliness, virtue,
Gray	Formality, neutrality, melancholy
Ivory	Elegance, simplicity, comfort
Beige	Traits of surrounding colors, humility, a secondary or background color

Figure 42: Colour emotion on web UI design[43]

If we follow fundamental color theory, finding a complimentary color is one of the greatest strategies to pick a color for quick attention. On the color wheel, a complimentary color to another may be found on the other side of the color wheel.[44]

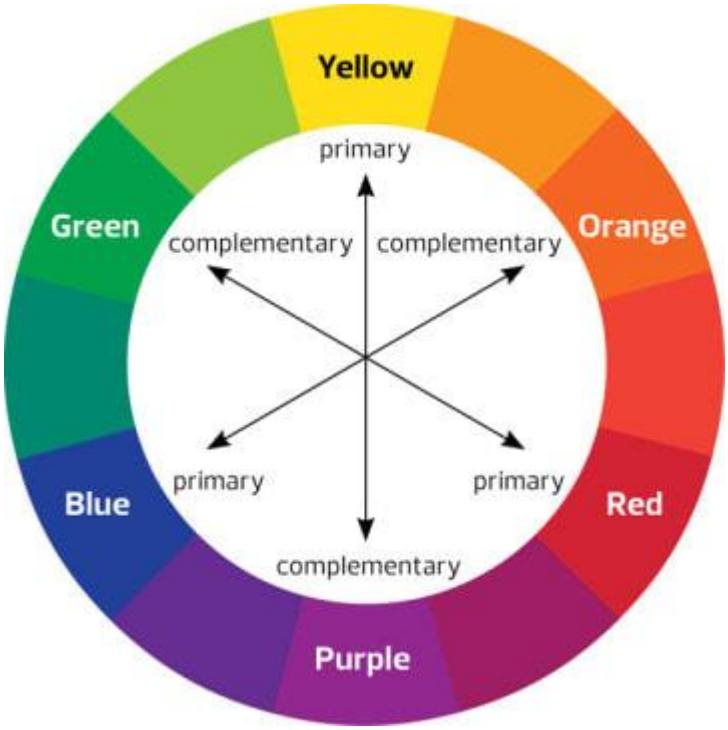


Figure 43: Complementary colours [44]

Each main color has a counterpart color that is one of the secondary colors. The secondary color matches the main color it does not utilize, which is a simple method to remember which colors to match. Red's complimentary color, for example, is green, which is made up of blue and yellow.[44]

Theory of color has three important variables to take into consideration, saturation, hue and brightness. Saturation refers to the purity of color, relative to the amount of white light with which mixed. The price of brightness depends on:

- From the amount of lighting that falls in surface and
- From its reflectivity, the rate i.e., reflected radiation in relation with the incident, which for ideal mirror is 100%.

Figure 44: Hue, saturation and brightness.

Last but not least, mentioning so many important terms about the theory of color but it's always beneficial the impact of the knowledge in our thesis project. All of the background color transition is depending on the complementary theory of color we mentioned before. At the heart of color theory, complementary colors are the opposite hues on the color wheel. In their most basic form, they are one primary color and the secondary color that is created by mixing the other two primaries. Specifically, we used a double-split complementary color scheme that is a tetradic scheme because it includes four colors., Based on the RGB codes we picked the following on the figure 45:

	HEX	RGB			CMYK			
	HEX:	R:	G:	B:	C:	M:	Y:	K:
	#de6d6f	**222**	**109**	**111**	**0**	**51**	**50**	**13**
	HEX:	R:	G:	B:	C:	M:	Y:	K:
	#6fde6d	**111**	**222**	**109**	**50**	**0**	**51**	**13**
	HEX:	R:	G:	B:	C:	M:	Y:	K:
	#6d71de	**109**	**113**	**222**	**51**	**49**	**0**	**13**
	HEX:	R:	G:	B:	C:	M:	Y:	K:
	#dea46d	**222**	**164**	**109**	**0**	**26**	**51**	**13**

Figure 45: Double split complementary colours of our thesis background colour

4.3 User error Messages

When building a page or a web-based utility, it's easy to disregard error warnings. Error notifications are sad because they get so little attention. It's the little things that count. They provide an opportunity to boost a company's overall brand and user experience. They also provide an opportunity to turn a frustrating situation into something enjoyable.[45]

Facebook's Jonathan Colman argued that error messages should explain things simply and plainly at the Delight Conference in Portland. [45]:

- What just happened
- Where the problem happened (did the user do something, or did the tool do something)
- What to do next and who should do it

Summing up, we understand the importance of user error messages. We gathered the following principles as we also stick to them on the making of our thesis project.

- Clarity is essential. When it comes to error messages, the most important thing to remember is to be as clear as possible. You must explain what happened, why it happened, and what the user may do to resolve the situation. The message should be expressed in simple language so that both the problem and the solution are easily understood by the intended audience. Avoid using technical language and avoiding abstract error messages. [46]

- Write succinctly yet accurately. In all aspects of your GUI, including error messages, try to keep the text to a minimum. Your objective is to create error messages that are brief but informative [46]. Because you'll be speaking to a range of folks with varying levels of technology and digital capabilities, don't overcommunicate the issue.

- Avoid using terms like "you did," "your activity resulted in," and "your activity resulted in." Instead of saying "You entered an erroneous login or password," say "Your login and password do not match." Use lower case text and (or) exclamation marks sparingly in your messaging to avoid irritating the user and leading to failure and abandonment of your product.

- Provide them with a solution rather than explaining each and every button on the screen. They only care about getting a quick answer and are uninterested in how your system operates. As a result, we believe that you do not have to disclose mistakes that your users are uninterested in. Developers may do this for their own reasons, but in the end, you must remove any unnecessary messages that may cause the user to get confused.

On figure 35 and 36 we have some examples for do and don’ts for the user errors. In our thesis we build a design stragedy based on words of UX designers. Error message design might seem like an insignificant part of information architecture, but it can have a tremendous impact on user experience. By reducing friction, you keep users on track and help them complete what they’ve planned.[46]

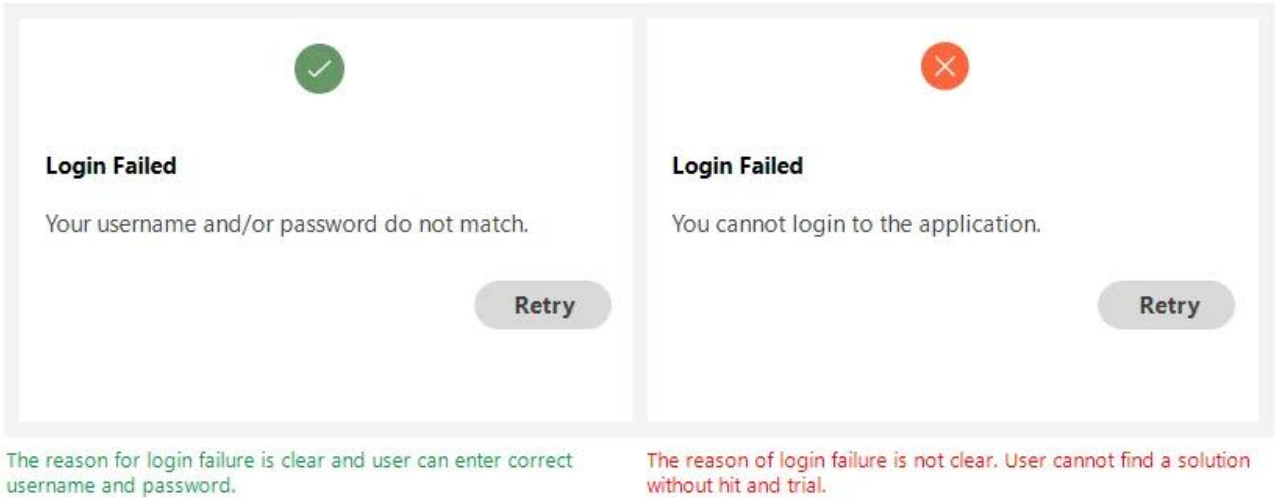


Figure 46: Login Failed example of user error messages

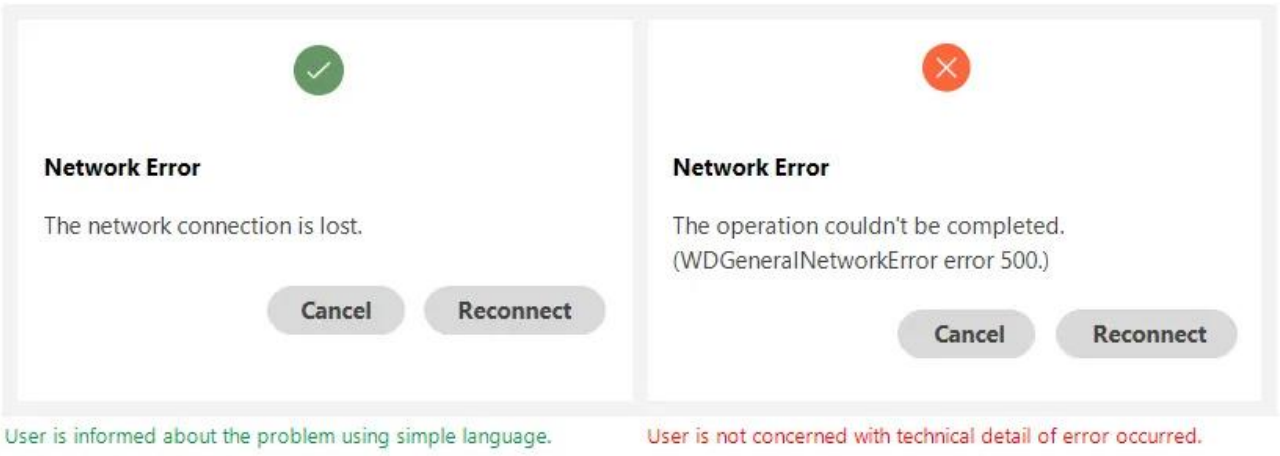


Figure 47: Network Error example of user error messages

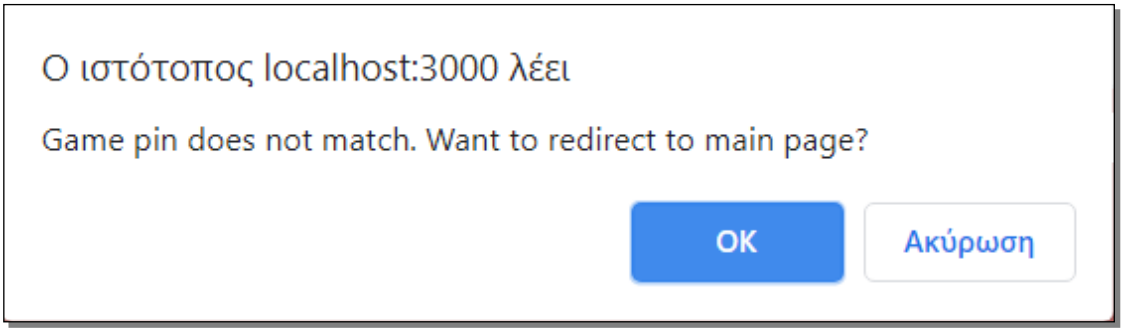


Figure 48: Error message example from our thesis project

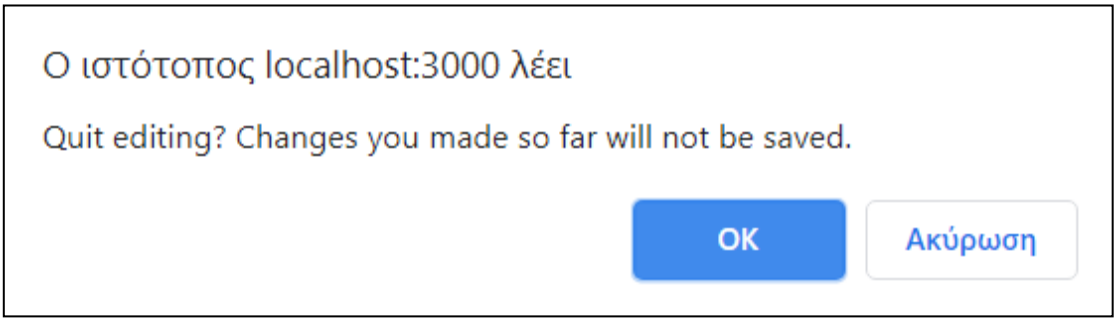


Figure 49: Error message example from our thesis project

Taking as an example the figures 48 and 49 we submit our effort to make a friendly automated message when he enters a pin that doesn't match and when he presses the cancel button on quiz creation accordingly. We could just redirect the user to the main page without any warning. Without these messages the user would never know what he did wrong or he might repeat his actions and make a loop of the same actions. In addition, we want to notice the user when his effort of quiz creation might disappear after pressing the cancel button. A case scenario is also that he might press cancel by accident and to get redirected as an automated action, might not be a good user experience.

4.4 Risk Management

Although, our thesis project has covered all the essential steps to create a clear vision of the upcoming platform, it is obvious that we will be confronted with several obstacles and it consequential to predict as many as possible. Our application has no existing data to proceed to quantitative analysis. As a result, we will focus on identifying potential risks regarding our steps and potential difficulties we might face prior or after the launching of the web application. In the following tables (table 5, table 6 and table 7) we are presenting a qualitative analysis by using the matrix scale of likelihood-severity as well as we determine the volume of the impact and probability to each risk and also define possible corrective actions.

The matrix scale presents the level of risk concerning the likelihood and severity scales.

Risk category provides a list of areas that have more potential to appear risks. These areas are categorized in the following four parts:

- Technical: Requirements, Technology, Interfaces, Quality.
- External: User
- Organizational: Project dependencies
- Project Management: Planning, Schedule, Estimation, Controlling, Communication

Although all the steps are defined observantly, there is a chance that we might face some barriers during the implementation of the project. It is consequential to predict some of the obstacles, determine the volume of the impact and also define possible corrective actions.

As shown in the following tables, there is a briefly presentation of risks that might affects the project:

Table 4: Risk management Table [47]

Hazard	Who is at risk	Severity
Instructions not be explained comprehensive enough	Users	Major Injuries
Trying to appeal to too many audiences might be effectiveless	Application	Minor Injuries
Time management problems and outline deadlines	Developer of the project	Fatality
User interface is not intuitive enough	Users	Minor Injuries
Failure to learn new technologies or tools	Developer of the project	Minor Injuries
Non responsive application enviroment for variety of devices	Users	Minor Injuries
Testing on a real audience before the first release	Application	Major Injuries
Illness/Accidents	Developer and Thesis Manager	Negligible Injuries
Unforseen family circumstances	Developer and Thesis Manager	Negligible Injuries
Potential users might not want to participate to the research	Application	Negligible Injuries

Likelihood	Risk Impact	Person Responsible	Recommended Action
Unlikely	Medium	Nick Tsougantzalis	Enhance "How to play" page with simple steps on how to play the game
Likely	Medium	Nick Tsougantzalis	Narrow down target audience and specify the needs
Unlikely	High	Nick Tsougantzalis	Trying to plan with sufficent based on Gantt chart timeline
Unlikely	Medium	Nick Tsougantzalis	Carry out friends and family to play the quiz app before thesis presentation
Unlikely	Medium	Nick Tsougantzalis	Ask for help from your professors or watch online tutorials
Unlikely	Medium	Nick Tsougantzalis	Test your project on Google Dev Tools to check responsiveness in many devices
Likely	High	Nick Tsougantzalis	Try to encourage people from your close circle to participate before the release day
Likely	Medium	Nick Tsougantzalis	Plan ahead of timeline so you can be on time in case of emergencies
Likely	Medium	Nick Tsougantzalis	Plan ahead of timeline so you can be on time in case of emergencies
Highly Unlikely	Low	Nick Tsougantzalis	Offer incecitives and encourage people to participate

Table 5: Risk Likelihood Distribution [47]

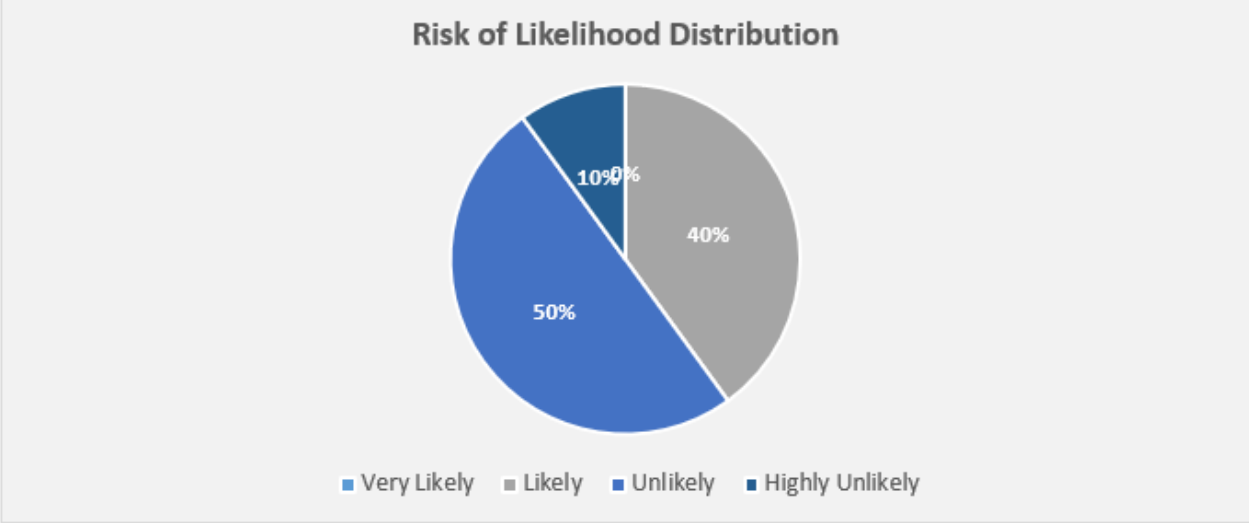
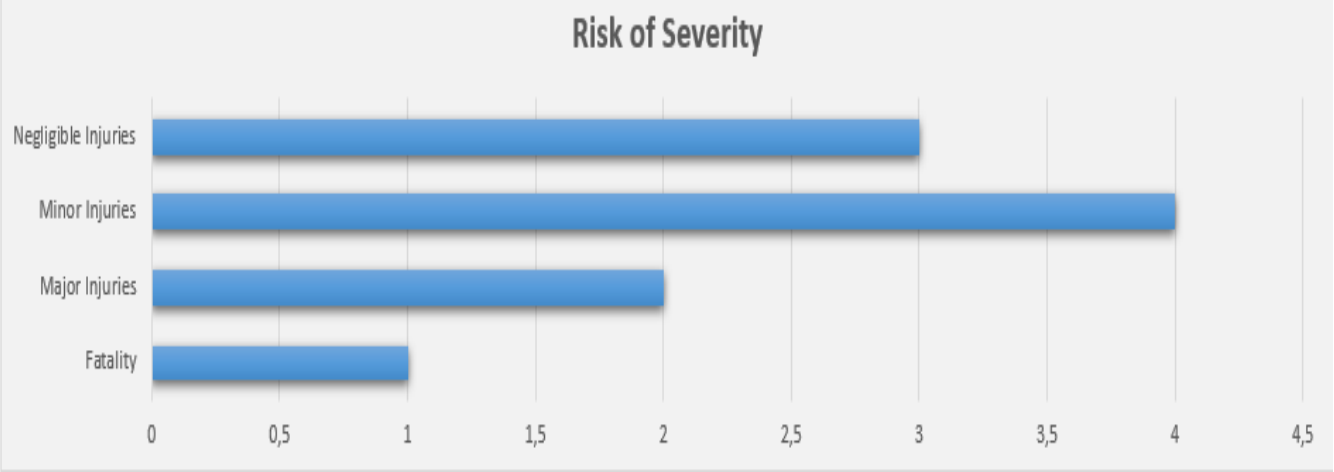


Table 6: Risk of Severity [47]



5. Conclusions and Future work

Schools were shuttered and instruction was shifted to pupils' homes as the Covid-19 epidemic ravaged the globe in 2020. As a result, several nations were confronted with an unplanned and rapid shift to online learning. Teachers were able to educate pupils at a distance utilizing systems that allowed for both synchronous and asynchronous contact with full classes, groups, and individual children or young people, as well as access to learning resources and interactive and collaborative activities.[48] Classroom of universities might be empty but this was an opportunity and a challenge for the creation of tools and applications that will improve the education on the pre and post covid era. Of course, we hope to overcome this pandemic but we also hope that we will keep the advantages of these tools.

The quiz application we created on this thesis project can be applied on a real or online classroom. Our purpose was to motivate the students to actively participate in an online classroom. The combination of technologies used above helped us create a modern application that focus on the user experience. Simplicity, usability and accessibility were the three features we sticked on while we were developing this application.

Our final conclusion is that gaming integrated within case studies has the opportunity to amplify user’s engagement and learning by permitting students to inspect challenging processes in a digital environment.

We will have new versions, more releases, and improve our user experience even more, and the design will follow the latest modifications to decrease the risks or obstacles. Hopefully, we will have the opportunity and time flexibility to examine real-time case scenarios in the near future. We will strive to absorb data from IHU students or other institutions and schools using questionnaires and the appropriate assessment technique in order to obtain real-time samples of user experience. Risk management attempted to address all hazards prior to deployment, but we still require people's reactions depending on their age or educational background in order to draw additional conclusions that will help us better.

Last but not least, the user interface should be improved on a regular basis to meet the demands of our quiz app. The addition of photos and videos to the questions and answers area is a future concept. Furthermore, we may make a change so that at the end of the quiz, users will be able to view films with the necessary theoretical background depending on the questions given. This necessitates a new database structure and a more sophisticated server configuration, but we've already created a system that can be modified to future adjustments.

6. Bibliography

[1] S. Murugesan, “Web Application Development: Challenges And The Role Of Web Engineering,” in *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, Eds. London: Springer, 2008, pp. 7–32. doi: 10.1007/978-1-84628-923-1_2.

[2] “Mishra and Mishra - 2013 - Software project management tools a brief compara.pdf.” Accessed: Apr. 29, 2021. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2464526.2464537

[3] M. Vanhoucke, *Project Management with Dynamic Scheduling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-40438-2.

[4] Y. B. Leau, W. K. Loo, W. Y. Tham, and S. F. Tan, “Software Development Life Cycle AGILE vs Traditional Approaches,” p. 6.

[5] “Insights to Agile Methodologies for Software Development | Hacker Noon.” https://hackernoon.com/a-case-study-type-insight-into-agile-methodologies-for-software-development-cd5932c6 (accessed Apr. 29, 2021).

[6] “MVC: Model, View, Controller,” *Codecademy*. https://www.codecademy.com/articles/mvc (accessed May 08, 2021).

[7] “Documentation for Visual Studio Code.” https://code.visualstudio.com/docs (accessed May 08, 2021).

[8] “About npm | npm Docs.” https://docs.npmjs.com/about-npm (accessed May 12, 2021).

[9] “npm.” https://www.npmjs.com/ (accessed May 12, 2021).

[10] S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, Nov. 2010, doi: 10.1109/MIC.2010.145.

[11] manekinekko, “What is Node.js? - Learn.” https://docs.microsoft.com/en-us/learn/modules/intro-to-nodejs/2-what (accessed Apr. 29, 2021).

[12] “Express - Node.js web application framework.” https://expressjs.com/ (accessed May 14, 2021).

[13] “Express ‘Hello World’ example.” https://expressjs.com/en/starter/hello-world.html (accessed May 14, 2021).

[14] A. J. Poulter, S. J. Johnston, and S. J. Cox, “Using the MEAN stack to implement a RESTful service for an Internet of Things application,” in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec. 2015, pp. 280–285. doi: 10.1109/WF-IoT.2015.7389066.

[15] “Express routing.” https://expressjs.com/en/guide/routing.html (accessed Nov. 28, 2020).

[16] J. F.- js.foundation, “jQuery API Documentation.” https://api.jquery.com/ (accessed May 15, 2021).

[17] D. Arrachequesne, “Introduction,” *Socket.IO*, May 14, 2021. https://socket.io/docs/v4/index.html (accessed May 15, 2021).

[18] D. Arrachequesne, “Rooms,” *Socket.IO*, May 19, 2021. https://socket.io/docs/v3/rooms/index.html (accessed May 21, 2021).

[19] Z. Parker, S. Poe, and S. V. Vrbsky, “Comparing NoSQL MongoDB to an SQL DB,” in *Proceedings of the 51st ACM Southeast Conference*, Savannah, Georgia, Apr. 2013, pp. 1–6. doi: 10.1145/2498328.2500047.

[20] “Introduction to MongoDB — MongoDB Manual,” *https://github.com/mongodb/docs-bi-connector/blob/DOCSP-*

3279/source/index.txt. https://docs.mongodb.com/manual/introduction/ (accessed May 15, 2021).

[21] L. Liang, L. Zhu, W. Shang, D. Feng, and Z. Xiao, "Express supervision system based on NodeJS and MongoDB," in *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, May 2017, pp. 607–612. doi: 10.1109/ICIS.2017.7960064.

[22] "Mongoose ODM v5.12.9." https://mongoosejs.com/ (accessed May 15, 2021).

[23] "Mongoose v5.12.9: Schemas." https://mongoosejs.com/docs/guide.html (accessed May 15, 2021).

[24] S. Mishra, "Significance of UI/UX design in mobile applications," *Medium*, Dec. 12, 2019. https://uxdesign.cc/significance-of-ui-ux-design-in-mobile-application-f90b2c56c204 (accessed May 18, 2021).

[25] "HTML 4.0 Specification," p. 369.

[26] "Web Development for Beginners – Learn Basic HTML and CSS to Build Your First Web Page," *freeCodeCamp.org*, May 14, 2021. https://www.freecodecamp.org/news/web-development-for-beginners-basic-html-and-css/ (accessed May 18, 2021).

[27] C. Grannell, *The Essential Guide to CSS and HTML Web Design*. Apress, 2008.

[28] "Powell and Powell - 2010 - HTML & CSS the complete reference.pdf."

[29] "CSS to CSS3 (Evolution of CSS)." http://www.tutorialspark.com/css3/CSS_CSS3.php (accessed May 18, 2021).

[30] M. K. Patel, "HTML, CSS, Bootstrap, Javascript and jQuery," p. 72.

[31] "A responsive Bootstrap Search Box that really searches." https://mdbootstrap.com/articles/miscellaneous/a-responsive-bootstrap-search-box-that-really-searches/ (accessed May 18, 2021).

[32] A. M. I. T. Asfar and A. M. I. A. Asfar, "Case-based games learning strategies to improve conceptual understanding in mathematics," *J. Phys. Conf. Ser.*, vol. 1663, p. 012060, Oct. 2020, doi: 10.1088/1742-6596/1663/1/012060.

[33] Z. Zainuddin, M. Shujahat, H. Haruna, and S. K. W. Chu, "The role of gamified e-quizzes on student learning and engagement: An interactive gamification solution for a formative assessment system," *Comput. Educ.*, vol. 145, p. 103729, Feb. 2020, doi: 10.1016/j.compedu.2019.103729.

[34] "Create an API with Express JS," *Hungry Turtle Code*. https://hungryturtlecode.com/projects/1-express-basic-setup/ (accessed May 20, 2021).

[35] "Array.prototype.filter() - JavaScript | MDN." https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter (accessed May 21, 2021).

[36] "Modules: CommonJS modules | Node.js v16.2.0 Documentation." https://nodejs.org/api/modules.html#modules_exports_shortcut (accessed May 21, 2021).

[37] "Structure your Data for MongoDB," *https://github.com/mongodb/docs-bi-connector/blob/master/source/server/introduction.txt*. https://docs.mongodb.com/guides/server/introduction/ (accessed May 25, 2021).

[38] "Document Schemas — MongoDB Realm,"

https://github.com/mongodb/docs-bi-connector/blob/DOCSP-3279/source/index.txt. https://docs.mongodb.com/realm/mongodb/document-schemas/ (accessed May 25, 2021).

[39] "Node.js MongoDB Insert." https://www.w3schools.com/nodejs/nodejs_mongodb_insert.asp (accessed May 25, 2021).

[40] "JavaScript regex trick: Parse a query string into an object « Steven Benner's Blog." https://stevenbenner.com/2010/03/javascript-regex-trick-parse-a-query-string-into-an-object/ (accessed May 30, 2021).

[41] T. Eaton, "The power of good UI and how it enhances engagement," *Medium*, May 25, 2018. https://uxdesign.cc/the-power-of-good-user-interface-and-how-it-enhances-engagement-the-new-currency-in-the-digital-43a59bcd9bda (accessed May 30, 2021).

[42] "HTML Responsive Web Design." https://www.w3schools.com/html/html_responsive.asp (accessed May 30, 2021).

[43] W. Swasty and A. R. Adriyanto, "Does Color Matter on Web User Interface Design," *CommIT Commun. Inf. Technol. J.*, vol. 11, no. 1, Art. no. 1, Aug. 2017, doi: 10.21512/commit.v11i1.2088.

[44] "Basic Color Theory for Web Developers," *DEV Community*. https://dev.to/nikacodes/basic-color-theory-for-web-developers-15a0 (accessed Jun. 01, 2021).

[45] D. Kenedy, "The Importance of Error Messages," *HB Design*, Jun. 24, 2015. https://www.hbdesign.com/importance-error-messages/ (accessed Jun. 02, 2021).

[46] "How to Write & Design User-Friendly Error Messages | Adobe XD Ideas," *Ideas*. /ideas/process/information-architecture/error-message-design-ux/ (accessed Jun. 02, 2021).

[47] "Excel Templates | Excel Spreadsheets | Someka.net." https://www.someka.net/ (accessed Jun. 03, 2021).

[48] L. Starkey, M. Shonfeld, S. Prestridge, and M. G. Cervera, "Special issue: Covid-19 and the role of technology and pedagogy on school education during a pandemic," *Technol. Pedagogy Educ.*, vol. 30, no. 1, pp. 1–5, Jan. 2021, doi: 10.1080/1475939X.2021.1866838.