

Dissertation

**STUDY OF INTERNET TECHNOLOGIES AND THE AVAILABLE
TOOLS FOR THE IMPLEMENTATION OF AN INTERACTIVE
WEB-BASED SYSTEM FOR DISPLAYING SENSITIVE DATA IN
REAL TIME**

Antonis Katsiantonis

UNIVERSITY OF CYPRUS



DEPARTMENT OF COMPUTER SCIENCE

August 2021

UNIVERSITY OF CYPRUS
DEPARTMENT OF COMPUTER SCIENCE

**Study of Internet Technologies and the Available Tools for the Implementation of an
Interactive web-based System for Displaying Sensitive Data in Real Time**

Antonis Katsiantonis

Supervisor

Dr. Giorgos Papadopoulos

Co-Supervisors

Christos Mettouris

Evangelia Vanezi

The Individual Diploma Thesis was submitted towards partially meeting the
requirements for obtaining the degree of Computer Science of the Department of
Computer Science of the University of Cyprus

August 2021

Acknowledgments

I would like to thank my Professor of Computer Science Dr. Giorgos Papadopoulos for giving me the opportunity to work on this dissertation, which improved my knowledge on internet technologies which are associated with real-time web apps, and also improved my experience on web app implementation. Special acknowledgment for the consistent supervising on the progress of my dissertation goes to Mr. Christos Mettouris and Mrs. Evangelia Vanezi. Their impact was crucial to the writing of this dissertation.

Finally, I would like to thank my family for supporting me throughout the entire time I was studying at the University of Cyprus and achieving my goals.

Abstract

Internet Technologies are a group of technologies that allow people to communicate and interact with each other, whether it is the forming of online communities on social platforms, being informed via news and media, researching with the use of search engines, or buying products using online marketplaces.

Today a variety of internet technologies are used, such as communication protocols and tools for the implementation of web applications, that interact with users in real time. Protocols are the system rules that allow multiple users and services to transmit information, while tools allow developers to develop, maintain and scale web applications and platforms.

The subject of this dissertation is the study of the internet technologies associated with the communication between nodes inside the internet, while taking into consideration the scenarios in which each technology is suited. After the study of the relevant internet technologies, the dissertation extends with a second part, which is an implementation, closely related to the first part of the dissertation. The purpose of the dissertation's second part is to update an existing platform, which receives sensitive data in real-time for analysis. Specifically, the goal is to study the technologies used by this platform for receiving data, in order to update/extend them if needed to improve the rate at which the platform can receive data, as well as the quality of the data. The study concluded that no further improvements needed to be made on the technologies used on the existing platform, and thus the second part of this dissertation presents a different approach in improving the existing platform, which is the implementation of a new platform that makes use of the data that the existing platform stores. Specifically, this new platform will present this data using graphical formats, such as charts, tables etc.

For the first part of the dissertation, scientific papers were studied, regarding internet technologies used for communication. Also, several protocols and tools were studied, that could be used specifically for the improvement of the platform which receives data analytics. These protocols and tools were studied in order to acquire a deeper understanding on how the selected protocols operate and examine whether the most appropriate protocols and technologies have been selected for the development of the existing platform. This dissertation concludes that the protocols and technologies used currently by the existing platform are well suited for its purpose, and that no changes have to be made on the existing technologies.

For the second part of the dissertation, as no improvements of the existing platform were needed regarding the web technologies used, the existing platform is extended by the implementation of a new web analytics platform, where it's purpose is the presentation of users' analytics data using visual aids. The web analytics platform is helpful to both researchers and teachers, as it assists in identifying the overall performance of students, displaying various stats, each one contributing in tracking where students can be improved.

Table of Contents

Chapter 1	Introduction.....	1
	1.1 Introduction	1
	1.2 Motivation	1
	1.3 Methodology	2
	1.4 Structure	3
 Chapter 2	 Related Work.....	 15
	2.1 Introduction	15
	2.2 Web Services	15
	2.3 Client-Server Model	15
	2.4 Application Layer	18
	2.5 Conclusion	20
 Chapter 3	 Internet Technologies for Real-Time Web-Based System.....	 7
	3.1 Introduction	7
	3.2 Methodology	8
	3.3 Internet Technologies	8
	3.3.1 Communication Protocols	9
	3.3.1.1 HTTP	9
	3.3.1.2 WebSocket	10
	3.3.2 Data Formats	12
	3.3.2.1 XML	12
	3.3.2.2 JSON	13
	3.3.3 Architectural Styles	14
	3.3.3.1 REST	15
	3.3.3.2 JSON-RPC	17
	3.3.3.3 GraphQL	19
	3.4 Internet Technologies for Real-Time Web-Based System	21
	3.4.1 Communication Protocols	22

	3.4.2 Data Formats	23
	3.4.3 Architectural Styles	23
	3.5 Conclusion	25
Chapter 4	Design and Implementation of Data Presentation Platform.....	26
	4.1 Introduction	26
	4.2 Data Protection	26
	4.3 Hardware	27
	4.4 Software	28
	4.4.1 Server	29
	4.4.2 Markup	29
	4.4.3 Styling	30
	4.4.4 Client-Side Script	30
	4.4.5 Server-Side Scripts	33
Chapter 5	Use-Case Demonstration.....	38
	5.1 Introduction	38
	5.2 Dashboard	38
	5.3 User Sessions	39
	5.4 Users Comparison	41
	5.5 Session Details	42
Chapter 6	Conclusion	44
	6.1 Introduction	44
	6.2 Conclusion	44
	6.3 Future Work	45
	Bibliography.....	45
	Appendix A.....	47
	Appendix B.....	48

Appendix C.....	5 5
Appendix D.....	6 2
Appendix E.....	7 7

Chapter 1

Introduction

1.1 Introduction	1
1.2 Motivation	1
1.3 Methodology	2
1.4 Structure	3

1.1 Introduction

This dissertation studies the internet technologies for the implementation of a web-based system which records data for analysis in real-time. Web-based systems are the systems that are usually accessed using a web browser. The term real-time means that the system responds fast enough so that the person using the system will perceive the response as immediate. This dissertation demonstrates the protocols and web technologies needed for the implementation of a web-based system that receives data for analysis and responds back in real-time. After the conclusion that the existing platform is already using the proper internet technologies, the dissertation proceeds to demonstrate the implementation of a new web analytics platform, which presents the data analytics to teachers using graphical formats.

Specifically, the purpose of the web-based system is to gather data analytics from video games. Some of the video games are implemented with Unity, while others are web-based. These video games use MySQL database for the storage of the data analytics.

1.2 Motivation

The purpose of this dissertation is to study the internet technologies that are related with the communication inside the web. Several protocols, tools, techniques and frameworks are covered. The dissertation compares the internet technologies which are studied with the technologies that were used for the implementation of the existing platform, and then conclude on which technologies are more suited for the implementation. A visual representation of the interaction between the games and the existing platform is displayed in

figure 1.1. Also, this dissertation aims for the improvement of the existing platform by making some use of the data analytics that this platform already stores.

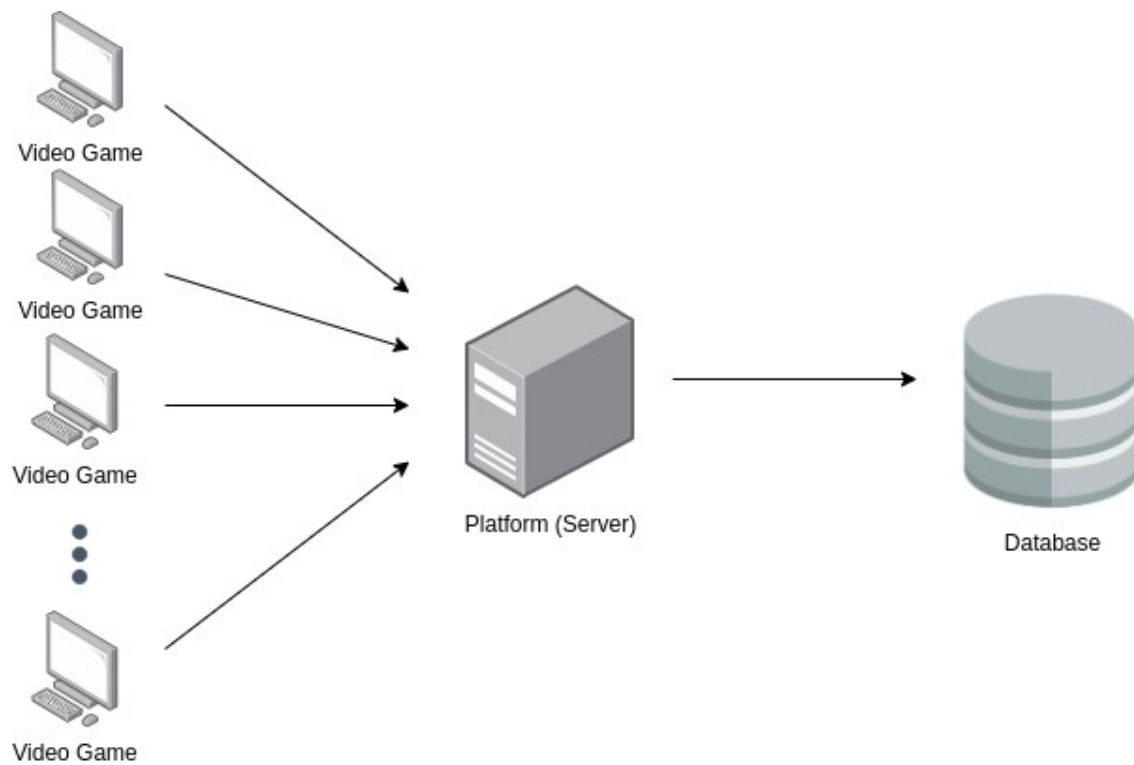


Figure 1.1: Games interacting with the existing platform

1.3 Methodology

The methodology that is followed in this dissertation consists of the 4 following steps:

- A. Study of the protocols used today for data transfer between nodes, in order to have a deeper understanding of how data is transferred throughout the web.
- B. Study of the internet technologies which can be used for the communication between the video games and the platform.
- C. Improvement of the platform using the internet technology that was decided to be the most suitable, if improvements are needed.
- D. Extension of the existing platform with the implementation of a data presentation platform, which presents the sensitive data to the teachers and researchers using graphical formats.

1.4 Structure

This dissertation is divided in 6 chapters. Chapter 1 consists of the introduction, which describes the motivation and the methodology which is followed for this dissertation. In Chapter 2 the related work is described, which is the work that is already been done on this subject, and this is where our work is based on. In Chapter 3, several internet technologies are covered. Each Internet Technology is studied and compared to the standard technologies that are used for the implementation of the existing platform that receives sensitive data. Chapter 4 presents the design and implementation of a data presentation platform, which is an extension of the existing platform that gathers data analytics, and presents this data in a way that the researchers and teachers can easily keep track of the students' performance. In Chapter 5, all use-cases of the platform are demonstrated, for a deeper understanding of how the data presentation platform is used and what information can be presented. Finally, Chapter 6 displays the conclusions of this dissertation. At the end of this dissertation, the bibliography and the appendices can be found.

Chapter 2

Related Work

2.1 Introduction	4
2.2 Web Services	4
2.3 Client-Server Model	5
2.4 Application Layer	6
2.5 Conclusion	6

2.1 Introduction

The purpose of this chapter is to study papers which are related to the subject of this dissertation. The papers which are studied in this chapter provide a basis of knowledge for the material which is covered in the following chapters of the dissertation. By covering the subjects in the following subsections, we acquire a deeper understanding on how communication happens over the world wide web. Initially, this chapter covers the concept of web services. This concept is explained with the use of definitions and examples of today's web services. This chapter also covers the client-server model. This concept is also explained using definitions, and there are also uses of examples for better understanding the application of the client-server model. Finally, this chapter gives a definition along with a brief explanation of the application layer, which is an important concept to understand in order to keep up with other concepts such as protocols and internet technologies.

2.2 Web Services

Usually, a Web Service is an application accessed by other applications over the web, but this definition is not precise, because it refers to other entities in the Web that are not web services. A more precise definition of a web service is characterized as a self-contained, modular business application which has open, Internet-oriented, standards-based interfaces [1]. In simpler terms, a Web Service is a service provided from one electronic device to other electronic devices, with each electronic device having the ability to communicate with other electronic devices via the world wide web. It can also be a server running on a computer

device, serving requests of other computers either by sending web documents, or processing information. In a Web service, various internet technologies such as protocols (HTTP, WebSocket) and other high-level tools are used for the transfer of information, which is structured in certain formats (XML, JSON).

One of the most popular kind of web services are the search engines. This kind of services are accessed and used billions of times per day. They are hosted by hundreds of thousands of servers which are located across the world. They are known to provide the service of easily finding information with the use of keywords.

2.3 Client-Server Model

Client-Server model is a system that performs both the functions of client and server so as to promote the sharing of information between them [2]. It allows The Client-Server Model describes the procedure in which providers of a resource or service known as server, interact with service requestors, known as clients. Servers are software which runs on special hardware, which are capable of serving multiple requests, and Clients are software run on devices used by end-users such as applications on smart phones, or websites that run on browsers. Usually, clients and servers communicate over the network on different devices, but they could also operate in the same system. Servers run multiple programs which are accessed by clients with the use of interfaces. The client does provide access to programs to other devices, but it requests resources and access to server programs. Therefore, clients establish connections with servers in order to send requests in order to access services, and servers respond to clients, either by providing, or denying services. The client does not have to know how the server operates internally in order to respond to the client's request, but it rather perceives the server as a black box that provides resources.

In order for the client and the server to communicate with each other is to have a common set of rules, along with a common language. Also, the client and the server should both have an interface, which one can refer to in order to know the structure of the messages to send to the other.

The client-server model can be demonstrated by taking as example an end-user who has access to a search engine using a web browser to navigate to it. In this scenario, the machine which runs the browser represents the client and the machine running the search engine service represents the server. The end-user requests for the service by typing the website which the search engine is located in on the browser's address bar. Then the machine hosting the search engine receives the request by the browser of the end-user and sends the website containing the search engine back to the browser, so that the browser can display the website to the end-user.

2.4 Application Layer

The application layer is made up of the communication protocols and interface methods which are directly used by end-user software that request resources and services, such as web browsers, email clients, smart phone applications, as well as server programs that provide resources and services to clients. The Application Layer is the most important layer in computer networks, and it's the layer which is the most visible to the end-user. [3]

2.5 Conclusion

This chapter has studied various sources that aid in acquiring a deeper understanding of some concepts. The concepts studied in the previous subsections are important to understand in order to be able to follow the concepts which are presented in the next chapters of this dissertation. By studying papers that cover more general subjects regarding communication and information transfer on the world wide web, it is now easier to follow along with more advanced subjects which follow in the next chapters.

Chapter 3

Internet Technologies for Real-Time Web-Based System

3.1 Introduction	7
3.2 Methodology	8
3.3 Internet Technologies	8
3.3.1 Communication Protocols	9
3.3.1.1 HTTP	9
3.3.1.2 WebSocket	10
3.3.2 Data Formats	12
3.3.2.1 XML	12
3.3.2.2 JSON	13
3.3.3 Architectural Styles	14
3.3.3.1 REST	15
3.3.3.2 JSON-RPC	17
3.3.3.3 GraphQL	19
3.4 Internet Technologies for Real-Time Web-Based System	21
3.4.1 Communication Protocols	22
3.4.2 Data Formats	23
3.4.3 Architectural Styles	23
3.5 Conclusion	25

3.1 Introduction

When studying the protocols for data transfer, we have a deeper understanding of how communication works on the web. When we study the internet technologies, it will be discovered that the technologies that will be seen next are based on the protocols that were studied.

After the study of the protocols which are related with communication, the study extends to the internet technologies for the implementation of a web-based system which records data for analysis in real-time. The purpose of this study is to find the internet technology that will

be used for the improvement of the platform, which is demonstrated in the next chapter. Finally, we conclude on the choice that was made for that internet technology.

3.2 Methodology

Initially, a group of Internet Technologies were chosen to be studied. The Internet Technologies that were chosen for this dissertation are the ones listed below:

1. HTTP (Hypertext Transfer Protocol)
2. WebSocket
3. REST (Representational state transfer)
4. SSE (Server-sent events)
5. SOAP (Simple Object Access Protocol)
6. GraphQL
7. RPC (Remote procedure call)
8. QUIC
9. WebRTC (Web Real-Time Communication)
10. OData
11. JSON
12. XML

These Internet Technologies were chosen because of their popularity as internet technologies such as protocols, APIs, data formats, etc. that serve for the communication on the web. For each of the internet technologies that were chosen, reliable resources were studied in order to have a deeper understanding of their purpose and the way that they work in detail. After understanding these internet technologies, one is chosen for the second part of the dissertation, which is the implementation of the platform that receives data from video games.

3.3 Internet Technologies

In order for two or more web applications to communicate with each other, internet technologies have been developed throughout the history of the web, each having a different purpose. These internet technologies are separated into categories in order to clarify their purpose. One of these categories that are be studied in this chapter is the communication protocols, which define the set of rules that multiple systems use to communicate with each other. Another category of internet technologies which is studied is the data formats, which define, as the name suggest, the format of the data when they travel from one application to another. The final category of internet technologies that is be shown in this chapter is the API

Architectural Styles. The APIs are used as interfaces to make web applications communicate in an elegant way. In the following sections, these three categories are explained, and for each category, the corresponding internet technologies are studied.

3.3.1 Communication Protocols

As was said in the previous section, a communication protocol is a set of rules which defines how two or more systems communicate with each other. In order to have a clearer understanding of the communication protocol's purpose, a real life example is used. Inside a classroom, in order for the students to communicate with the teacher, they have to have a predefined way to draw the teacher's attention, so when they want to *request* to speak, it's a convention for them to raise their hands. The student will then draw the teacher's attention, and the teacher will *respond* by giving permission to the student to speak. The student will then say what he wants, and then the teacher will respond to what the student said.

Without this simple convention between the students and the teacher, the communication inside the class would not be easy. If a convention is not defined, then each student would try a different method for drawing the teacher's attention. For example, while one student would raise his hand, some other student would try to stand up, and in this case the teacher would ask for the student to sit down instead of giving him permission to speak.

In the internet world, the students could be browsers, while the teacher could be a web server. In order for web browsers to be able to connect to a web server, they must be communicating using a common communication protocol.

In the following subsections, two communication protocols will be seen, HTTP and WebSocket, each suggesting a different solution for systems to communicate.

3.3.1.1 HTTP

HTTP (Hypertext transfer protocol) is one of the most widely used communication protocols today. It is mainly used for communication on the web, mostly in form of hypertext documents, and hyperlinks that link to other hypertext documents, but it can also be used for the transfer of other types of data as well. It is a text-based protocol, in which the client sends a request and the server returns a response. [3]

The HTTP protocol is based on the client-server model, where the client sends a request, meaning that it asks the server to take a certain action, and the server sends a response, meaning that it notifies the client about the action that is for the corresponding request of the client. The server can receive requests from multiple clients at the same time, and the server will respond to the clients in a certain way. A visual representation of an HTTP connection is displayed in the figure 3.1. The real life student-teacher example which is illustrated in the

above section is ideal for understanding this protocol, where the students and the teacher represent the clients and the server respectively. When the students need to say something, they raise their hand for permission, and the teacher gives permission to the students to speak. Due to the possibility of too many clients requesting on a single server, the server uses scheduling techniques to serve multiple clients at the same time. Also, at some occasions, a server may limit the amount of requests that receives from clients in order to prevent denial of service (DOS) attacks.

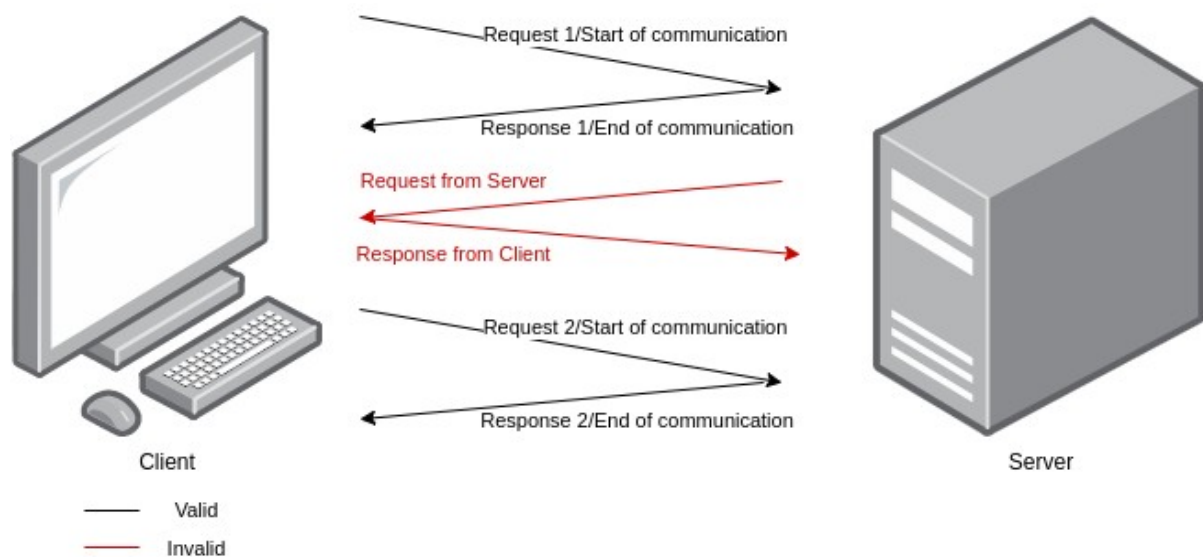


Figure 3.1: HTTP connection establishment

We can use the login interface on an online banking application as an illustrative example of an HTTP-based system on the internet. Clients have access to the banking system using a web browser. The login screen is reached using the address bar, which is used to request hypertext from web servers. Then the online banking server then sends the login screen to the client. After that the client enter his credentials, and by logging in he request from the server to be authenticated. The server then responds according to the validity of the credentials.

3.3.1.2 WebSocket

WebSocket is another communication protocol which is also used along with the more popular protocol, HTTP. However, the WebSocket protocol differs from the HTTP protocol when it comes to the way the communication works. The WebSocket protocol is defined as a

full-duplex, bidirectional, single-socket connection. The WebSocket replaces an HTTP request with a single request to open a WebSocket connection, and reuses the same connection from the client to the server, and the server to the client [4].

When the client and the server communicate using the WebSocket protocol, the communication is bidirectional, meaning that both the client and the server have the ability to send and receive data to/from the other side. There is less communication overhead compared to the unidirectional protocols, such as HTTP, because once a socket has been initiated from any of the two sides, then both the client and the server can send data until the socket is closed by any side. This way, the server can send data to the client without the client having to make a request. The WebSocket connection is visually presented in the figure 3.2.

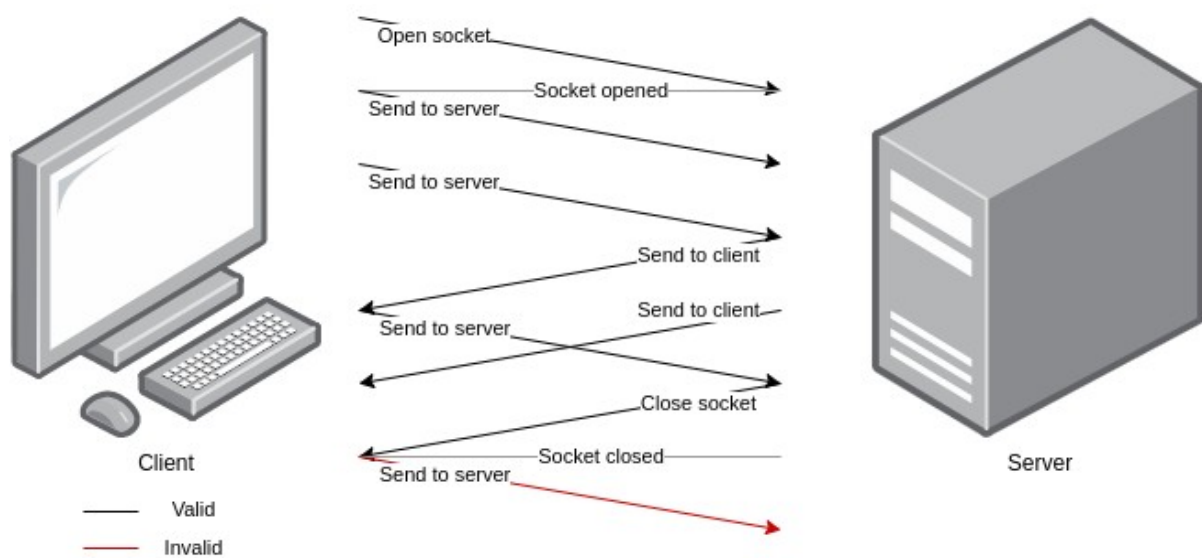


Figure 3.2: WebSocket connection establishment

Here, the real life student-teacher example is not very suitable, but we can visually illustrate the protocol using another example. Instead of the students being in the classroom, they are having a break at the schoolyard during the lunch period. In this scenario, each child can strike a conversation with another child, and the conversation can go on without any of the two children having to ask for permission, because the conversation is already started. Finally, each child has the ability to interrupt the conversation at any time. In this example, one child plays the role of a client, and the other plays the role of a server. It is observed that both children can exchange words once the conversation is started.

It is recommended for the WebSocket protocol to be used for applications where information need to be transferred in real-time. One example where the WebSocket is suitable, is a chat application, where messages are expected to be sent immediately after their composition.

Each person (client) will be connected with the chat application (server) using a socket. Whenever someone wants to send a direct message, the message goes through the socket which connects the sender with the application, and then the application sends the direct message to the recipient using the socket which connects the application with the recipient. Here, the application can send the message to the recipient, without the recipient having to ask the server if there are any new messages. The application has the ability to independently send messages to the users in real-time.

3.3.2 Data formats

In the previous section, we discussed the need for communication protocols, due to the fact that two systems cannot communicate with each other if they don't have a common set of communication rules that both systems understand and agree on. In this section we are discussing the need for data formats. In order for two or more systems to understand the content of the messages that they are sending to each other, a predetermined format for the messages has to be established.

For a better understanding of the necessity of data formats, the teacher-student example is mentioned. Suppose that the classroom has a predefined set of rules that determines how the students will communicate with the teacher. Also, let's assume that everyone speaks only English, except for one kid who speaks only Spanish. This means that, although the kid is able to ask the teacher for permission by raising his hand, he won't be able to tell her what he wants because they speak different languages. In this example, the raise of the student's hand represents the communication protocol, and the language represents the data format.

This example resembles two or more systems that, despite of having a common communication protocol, their messages have different formats, and therefore they are unable to communicate. The sending system will successfully send a message to the receiving system, but the receiver will not be able to understand the message, and therefore will not be able to respond appropriately. Thus, for two or more systems to communicate successfully, a common data format has to be determined.

In this subsection, we are looking at two different data formats, XML and JSON.

3.3.3.1 XML

XML is a Markup Language that is mainly used for the distribution of data throughout the internet. They are made up of storage units called entities, which contain either parsed or

unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constants on the storage layout and logical structure [5]. Although XML was primarily designed for documents, it is commonly used for the representation and distribution of structured data across the web. This markup language is known for its simplicity and readability, due to the fact that its syntax is identical to the HTML markup language, which a lot of people are familiar with. Also, it is also widely used because of its compatibility with various platform and programming languages. XML supports Unicode, which means that the content of an XML data structure supports most written languages in the world and it's not limited to a certain amount of available characters, such as English letters, numbers and symbols. An example of an XML file is displayed in the figure 3.3.

```
<email>
  <to>Andreas</to>
  <from>Antonis</from>
  <heading>Party</heading>
  <body>Hey, don't forget to come to the party tomorrow!</body>
</email>
```

Figure 3.3: XML file format example

Despite the benefits that the XML data format provides, it is not as preferred as JSON due to some of the drawbacks that it carries with it. The syntax of XML, while simple to read and understand, it is also verbose and redundant compared to other data formats. This means that an XML structure requires more storage space, it visually appears larger and its size negatively affects the communication speed between systems on the web. Finally, when using XML, arrays are not supported, in contrast to other data formats.

3.3.3.2 JSON

JSON is a file-independent data format that is used to store and transfer structured data in key-value pairs. It enables applications to communicate over a network. Today, it is widely used as a data format for the communication between clients and servers. Its syntax was derived from JavaScript, but now most modern programming languages have the ability to encode and decode JSON structured messages[6]. An example of a JSON file is displayed in the figure 3.4.

```
{
  "email": {
    "email_id": 7389217049,
    "from": "Antonis",
    "to": "Andreas",
    "cc": ["Nikos", "Giorgos", "Vasilis", "Christos", "Andrea"],
    "heading": "Party",
    "body": "Hey, don't forget to come to the party tomorrow!",
    "important": false,
    "attachments": null
  }
}
```

Figure 3.4: JSON file format example

One advantage that JSON has over XML is that its syntax is simpler and lighter, which makes this data format easier to read, smaller in storage size, and it also reduces the communication time between systems on the web. The JSON data format is a lot more popular than XML due to the popularity of JavaScript, which uses JSON for sending data messages to servers [7].

3.3.3 Architectural Styles

An Architectural Style is the set of rules and disciplines applied so that the systems will interact with each other inside the web in a certain way. Certain architectural styles along with some protocols are used interchangeably to describe the same concept, but there is a major difference between these two terms. The main difference between a protocol and an architectural style is that, a protocol is a set of rules which are implemented to define the way messages travel from one system to the other, while the architectural style encapsulates the implementation of the protocol and establishes conventions that systems agree on in order for them to interact with each other.

In order to distinct the concepts of protocols and architectural styles, we will go back to the teacher-student example. In this scenario, the implementation of the protocol can be represented by the student raising his hand along with the teacher responding by calling the name of the student. In contrast, the architectural style can be represented by more general conventions that apply to the whole class, such as the students lowering their hands when a student was given permission, or the rest of the students waiting for the teacher to finish answering to a question before they raise their hand again.

In the next sections, several architectural styles are studied, in order to get a deeper understanding on the disciplines of each one. The architectural styles were chosen for their popularity, and are presented in chronological order.

3.3.3.1 REST

REST stands for representational state transfer, and it's the most widely used architectural style today. It is built on the HTTP protocol, which means that it is based on the client-server model, and it is used to create interactive web systems that communicate with each other. When a groups of web systems use the REST architectural style, they are called RESTful. When a web system follows the REST guidelines, it provides an interface to clients which can be used to read or modify the state of the system that is providing the interface. Specifically, the REST architectural style provides an interface which can be used for creating, retrieving, updating and deleting information in databases [8].

Each time the client and the server need to communicate, each one must follow a series of steps in a particular order. Initially, the client starts the communication by sending a request. There are certain information which are required to be included inside the request message. First, an endpoint URL is required, which determines the domain, the port, the path and the query of the request. The domain refers to the address of the server where the request will travel, the port selects one of the applications that the server provides, the path is related to the action that is requested to be taken by the server, and the query is used to pass parameters to be used for the action called. For the construction of a request message, an HTTP method must also be defined. The HTTP method defines the type of action that is going to be performed, and it can only take a set of values, with GET, POST, PUT, DELETE being the most important. The GET method request information from the server, the POST method requires that information will be stored in the server, the PUT method requires for data to be updated in the server, and finally DELETE requires for data to be deleted from the server. The Headers is another piece of information that the request needs, which contains additional information that is needed for the communication between the client and the server. Finally, the request may contain a body, where additional data can be placed, such as data to be stored or updated.

This architectural style was created in need for performance, simplicity and scalability, and for these needs to be satisfied, the appropriate guidelines were defined for the REST web systems to follow. Initially, the REST architectural style is Stateless, which means that a

system on the web that receives a message, will process that message a certain way regardless of the system's state. Each message is isolated from other messages received by the web system, thus increasing performance by removing server load, especially on large-scale systems. Another guideline that is provided by REST is Isolation, meaning that each system on the web operates independently of the other systems. Both the client and the server are independent systems, and one system doesn't have to know anything about the implementation of the other. This means that the implementation of each of the systems can be changed without having any issues, as long as the interface that each system provides remains unchanged, that is, the format of the message that each side expects to receive remains the same. This way, each system can be evolved without having to worry about compatibility with other systems, because the communication interfaces which are provided by each system, remain unchanged. Layered systems is an additional guideline, where a system can be broken up to multiple sub-systems, and each of them has it's own purpose. That way, each sub-system can be developed independently from one another. The layered systems guideline is visually presented in the figure 3.5.

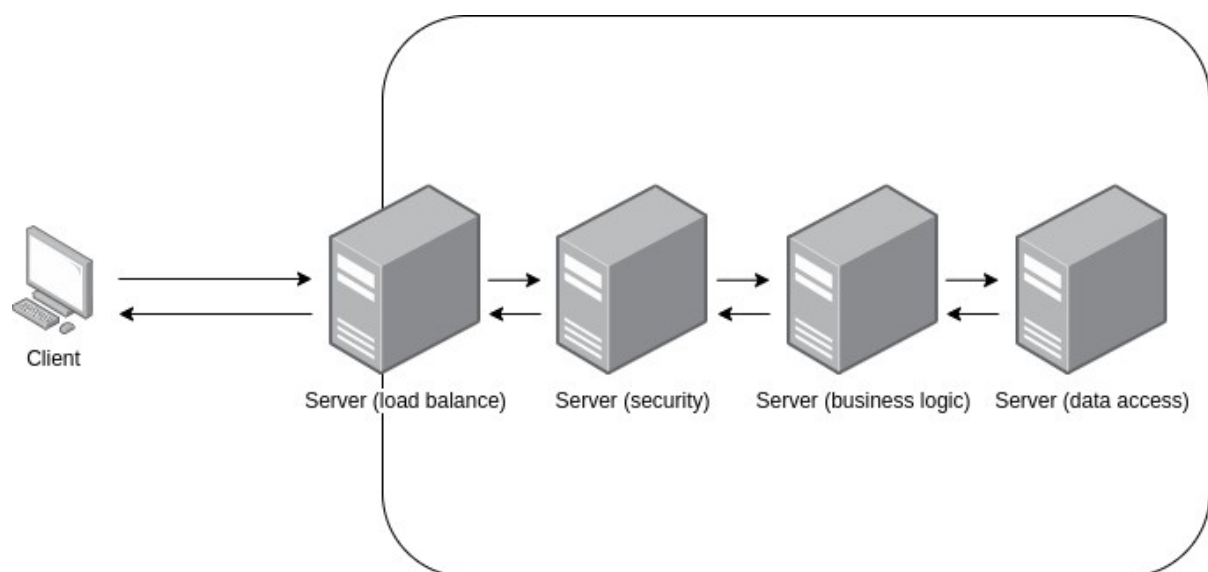


Figure 3.5: client communicating with RESTful service with layered systems guideline

For example, a system can be divided into multiple sub-systems, the one can be taking care of security, the second one could take care of business logic, another could handle the data access to the database, and so on.

3.3.3.2 JSON-RPC

JSON-RPC is another architectural style which is used for calling functions remotely and the transfer of messages across the web. This architectural style is also based on the client-server model, where clients request for information or actions from the server, and the server responds appropriately [9]. The connection between the client and the server with JSON-RPC is visually presented in figure 3.6.

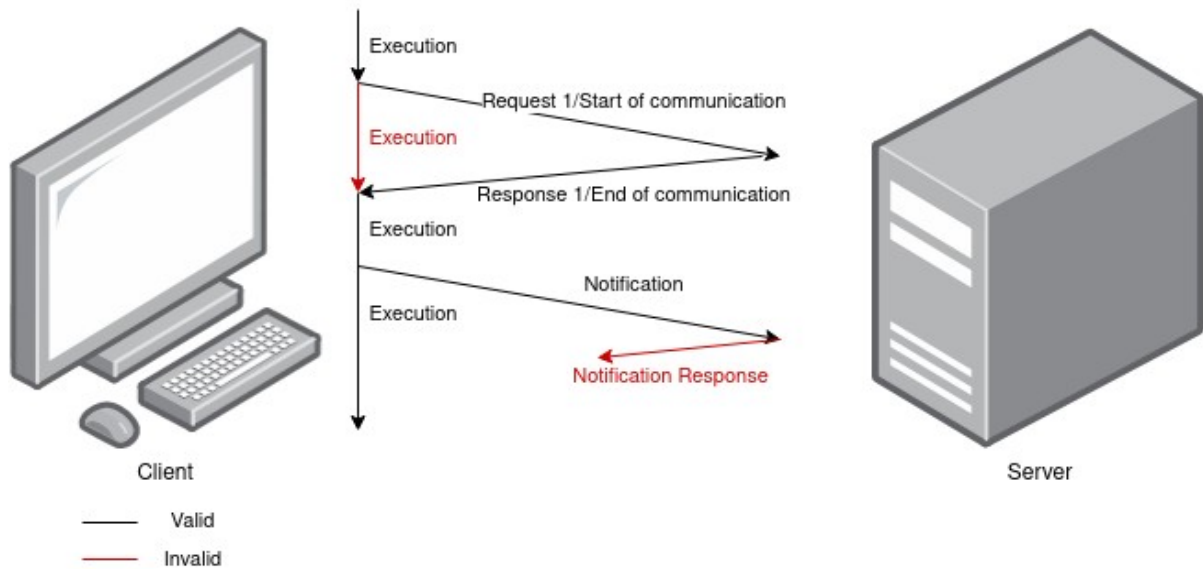


Figure 3.6: JSON-RPC connection establishment

It is stateless, meaning that each client's request happens in isolation, meaning that the server does not rely on previous request to handle the current one. The term JSON was covered in the data formats section, and it specifies the format that the messages take in order to travel between the client and the server. The term RPC stands for Remote Procedural Call, and it expresses the main idea behind this architectural style. The procedure behind an RPC-based protocol is that the client calls a function or a procedure that is located to a remote system, as if the function/procedure is located on the client's local environment. Just like in the REST architectural style, the client and the server communicate in a specific way. Initially, the client sends a JSON object to a server that is based on JSON-RPC. The request message has a predefined structure and must contain fields that are necessary for the communication to work. The method is one of the fields specified inside the request, and it determines the function/method of the server that will be executed. Additionally, the params field must be specified, which specifies the values of the arguments that will be passed in the defined method. Finally the id field must be specified, which is responsible for the uniqueness of the request, and it's also useful for the matching of each response to the appropriate request. That

is, if there are multiple requests to a server from multiple clients, we wouldn't know to which client each request should travel, and for a particular client, we wouldn't know the match that client's requests with the server's responses. After the server receives the client's request message, it responds with a response which has a similar structure, but with different fields. The first field is named result and contains the response data that the client requested for. This field is included inside the response with the assumption that the method was executed successfully. The second field is named error, and it is included in the response message only if there was an error when calling the method, and it contains an error code along with an error message that explains the reason of the error. The third field that is included is the id, which has the same value as the value of the id inside the request message. As explained above, this field is helpful for the client to know to which request does this response answer. What was just explained was the request-response model for JSON-RPC, but there is another way for the client to interact with the server using this architectural style. Instead of the client sending requests to the server, he can instead send notifications, which have similar structure as a request, but it lacks an id field. As the server receives the message from the client, it notices that there is no id inside the message and perceives it as a notification, and it does not respond to the client. This way, the client sends a notification and does not have to wait for the server to respond in order to continue executing.

The simplicity of the JSON-RPC architectural style makes it lightweight, meaning it is ideal for simple client-server interactions. Also, it is faster for web interactions where the server's response is not necessary and the client does not have to wait for response to continue executing.

3.3.3.3 GraphQL

GraphQL is language for APIs and it's used as a language for retrieving and manipulating data using queries, and it is also used for the execution of these queries on existing data. The syntax of this language allows for the client to request for data in specific structure, while the server returns the data that was requested in the same structure that the client requested them. This method of data transfer minimizes the amount of data that travels across the web, which in turn makes the communication faster between the client and the server. GraphQL supports operations such as reading from and writing, and also supports subscription to changes on a server, which means that whenever there is a change on given data on the server, the client will be informed about that change in real-time, as long as the client is subscribed to that data. The ability to subscribe to specific data which is given by GraphQL, is implemented using the WebSocket protocol that is mentioned in earlier sections. Examples of GraphQL request and response can be found in figure 3.7.

```
REQUEST:

query {
  student(username: "akatsiantonis") {
    department
    courses(semester: 1)
    average_grade(semester: 1)
  }
}

RESPONSE:

{
  "data": {
    "student": {
      "department": "Computer Science",
      "courses": [
        {"course_code": "CS101", "title": "Programming Principles"},
        {"course_code": "CS102", "title": "Discrete Structures"},
        {"course_code": "LAN100", "title": "General Advanced English"},
        {"course_code": "MAS011", "title": "Calculus 1"}
      ],
      "average_grade": 8.50
    }
  }
}
```

Figure 3.7: GraphQL request and response structure examples

As we mentioned earlier, in order to retrieve and manipulate data using GraphQL, the client sends a message that contains the exact fields that need to be read or manipulated. Then the server sends a JSON response that follows the same structure of the client's request, and with the same fields that were required by the client.

In order to better understand how GraphQL works in practice, we will look at a basic example where we read and manipulate the data of an undergraduate student, and we will compare how GraphQL works compared to the REST architectural style. First, let's assume that we want to retrieve the student's department that he is studying in. When using a REST API, usually the client would have to make a request using an endpoint which takes a student as a parameter and returns its department. Secondly, let's assume that we also want to retrieve the courses that the student was registered in. In that case, a REST API would also have an endpoint that receives a student as a parameter, and then returns the courses for that student. Finally, assume that we also want to get the average grade for all the courses that the student has completed. In the REST API scenario, the client would call another endpoint that is responsible for sending the average grade, given a student as a parameter. In this example, when using the REST API, we would make three separate calls to the server in order to gather all the required data, which means that a relatively large overhead should be expected. Another problem that occurs using the REST API is that these three endpoints could possibly send more data than required, which adds to the overhead. For example, there is a chance that the server provides an endpoint that returns all the courses for a given student, but not for a given semester. In that case, if a client wants to retrieve the courses of a student for a given semester, then he would have to gather the courses from all the semesters from the server, and then discard the courses of the semesters that he doesn't need to know. In that case, the job is done, but more data was retrieved than needed which further adds to the overhead. GraphQL, compared to the REST architectural style, does not have the liabilities that were mentioned above. In the student example, the client would only have to make a single request, which describes with accuracy the data that are needed by the client, and the server brings exactly the data that were requested. In conclusion, the fewer requests that were made, and the fewer information that traveled from the server to the client implies less overhead, which makes the GraphQL more efficient [10]. Figure 3.8 displays a comparison of connection establishments between REST and GraphQL. When using GraphQL, because the client does not rely on the implementation of the interfaces of the server, it asks exactly for what it needs, in contrast to REST where the interface of the server forces the client to make multiple requests in order to retrieve the information that he needs.

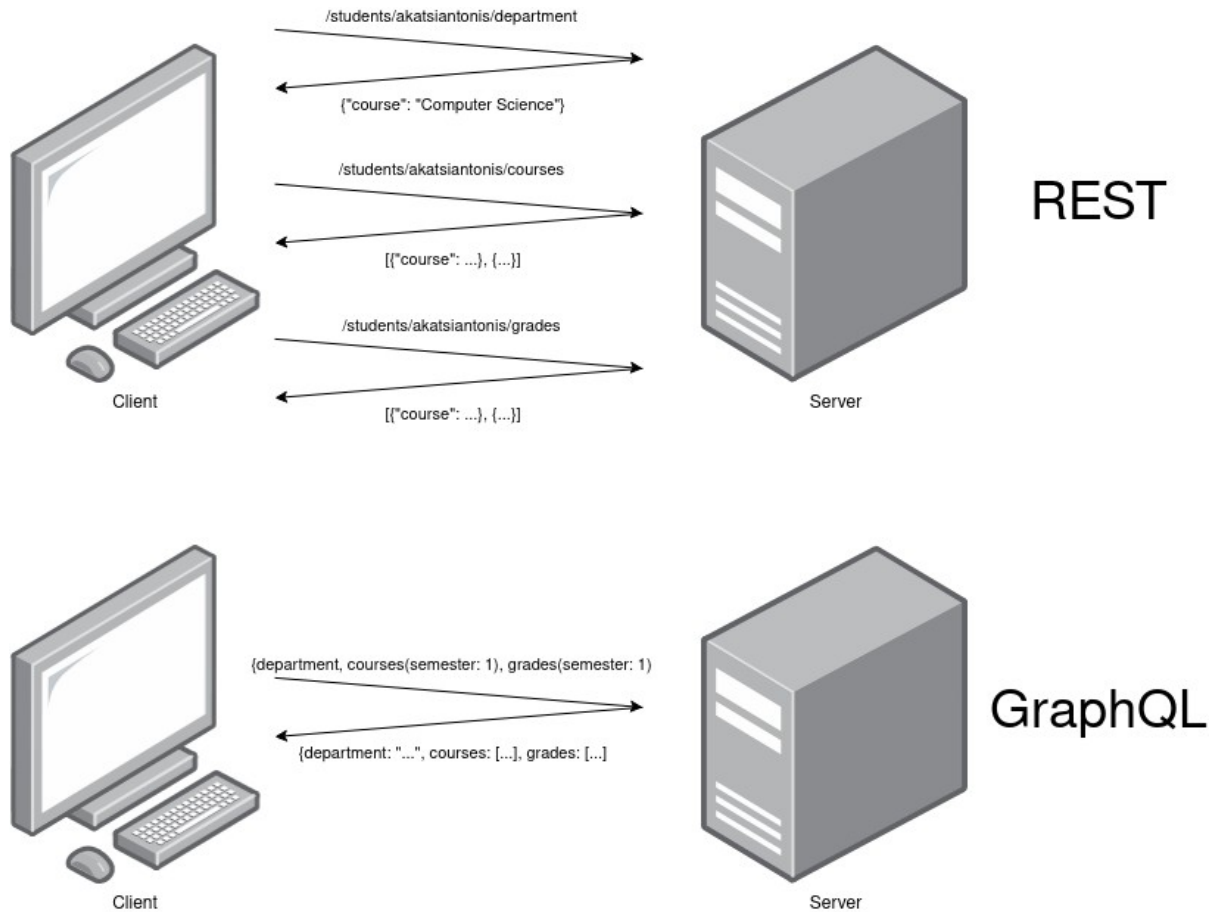


Figure 3.8: Comparison of GraphQL and REST connection establishments

3.4 Internet Technologies for a Real-Time Web-Based System

In the previous section many internet technologies were discussed in terms of availability and capability, and they were divided into three categories: the communication protocols, the data formats, and the architectural styles. For each of these internet technologies, their strengths and weaknesses were pointed out, that is, where each of these should or should not be considered. In this section, these internet technologies are discussed in the context of the platform implementation which is shown in the next chapter.

As mentioned in previous chapters, the purpose of the platform (server) is to gather data for analysis from video games (clients) and store them in a database in real-time. If we look back on the internet technologies that were seen earlier in this chapter, everything that was seen

can be useful for the implementation of this platform, but some technologies might fit better than others.

In the following sections the internet technologies are compared for each category, and then the ideal ones are chosen for the implementation of the platform.

3.4.1 Communication Protocols

In this chapter, two communication protocols were discussed, the HTTP and the Websocket protocols. Today, both protocols are widely used, each one for different scenarios. In order to choose which one is a better choice for the implementation of the platform, we need to think how each of the communication protocols would work if they were to be used.

In the case where HTTP is used for the implementation of the platform, the client would make an HTTP request to the server, sending a message that contains the data for analysis, and then the server would send an HTTP response containing the status message, that is, whether the storing of the data was successful or not. The client would then receive the response of the server, and continue operating based on the response message. Although the client can process the response message of the server, it could also ignore it by assuming that the data for analysis was stored successfully. In this case, The client waiting for the server is considered an overhead, which adds up, assuming that there are multiple games, and each game makes multiple requests in a fixed amount of time. Another overhead that occurs, is the overhead of opening and closing connections for each message that is to be sent from the client to the server, which again adds up based on the number of clients that send to the server, and on how many message does each client send.

If the platform were to be implemented using a WebSocket protocol, then the client would open a persistent connection with the server (or vice versa). Then, each time the client would want to send a request message, he would use the same connection. When the server receives the request message of the client, then it could either respond, or do nothing. In this case, it would be ideal for the server to not respond in order to minimize the network overhead. Also, the prevention of connections to be opened for each of the client's messages is another overhead that is avoided.

In conclusion, both of the communication protocols can be used for the implementation of the platform. Based on what was mentioned above, WebSocket would be more efficient than HTTP for multiple video games sending multiple data for analysis to the platform. However, WebSocket might come with development complications, such as finding stable tools for such implementation, which are also compatible with the programming languages used in the to develop the platform that gathers sensitive data for analysis. The protocol that is chosen for the communication between the platform and the video games, also depends on the

architectural style that is chosen later, but generally, both protocols are well suited for the communication between platform and games.

3.4.2 Data Formats

In this chapter, two data formats were discussed, XML and JSON. Generally, both of these data formats can be used for the implementation of the platform. Of course, the selection of the data format can depend on the architectural style that is chosen later, because some architectural styles only support one of the two data formats. Despite that fact, this subsection determines which data format is a better fit for the implementation.

Although both XML and JSON are decent options, the discussion that was made in the Data Formats subsection clearly states that JSON is a better option. One explanation as to why JSON is more suitable than XML is that, JSON was meant to be a data format, whereas XML is not just a data format, but it is also used for the visual representation of the data as well. Also, as mentioned before, the syntax of JSON is more minimal, which makes the JSON file lighter and easier to read. Furthermore, because of the relatively smaller sizes of the JSON files, less data is transferred across the web, which makes the communication faster between clients and servers. This concludes that JSON is a more efficient data format than XML for the implementation of the platform due to JSON being more efficient than XML [7].

3.4.3 Architectural Styles

In this chapter, three architectural styles were discussed, REST, JSON-RPC and GraphQL. For each of these architectural styles, we explained how they work using client-server paradigms and then talked about the benefits for choosing to use them. In this subsection, we discuss how these architectural styles can be applied for the implementation of the platform, and what benefits do we get from using each of these architectural styles, followed by their drawbacks that occur by using them for the implementation.

First, we discuss how the platform would operate using the REST architectural style. Initially, the video game would send an HTTP request to the platform, and then wait for the response of the platform. Then, the platform would store the data for analysis, and then create an HTTP response message containing a status message, and send it back to the video game. Then the client receives the response message and continues execution. This solution would be functional, but it would probably be inefficient, due to network overheads. When using the REST architectural style, for each piece of information from each of the video games that is sent to the platform, a persistent connection needs to be opened and closed, which adds to the time needed for the platform to complete the procedure of storing the data. Also, for each

message that is sent to the platform, the platform needs to respond to the video game in order to inform it that the storing is done. This step could also be considered to be an overhead, because the platform does not send data that is crucial for the video game execution, and thus it should ideally be ignored. Although the process of the data storing is relatively small, it will be called multiple times, and it should be implemented as efficiently as possible, therefore overheads should be reduced as much as possible. One solution would be to minimize the responses that the platform sends in order to minimize the network traffic, which in turn reduces the time needed for the server to respond to the client. However, we cannot reduce the overhead of the multiple connections opening and closing along with the request and response messages.

Now that we've seen how the platform operates using the REST architectural style, we will discuss how differently the platform would operate using JSON-RPC. The video games have the choice to send either a request, or a notification to the platform. The request contains the function name that the server is going to be executing in order to store the data for analysis, the parameters which contains the actual data that is going to be stored, and also the id of the request which distinguishes the request from the rest of the requests. When the video game sends a request, the platform gets the video game's data and proceeds to store them. The platform then reads the id of the request, and returns a response to the video game with the same id as the one received, in order for the video game to distinct the platform's response from the rest of the responses received from the platform. Also, the response contains a status message that notifies the video game whether the function has been successfully executed. Alternatively, the client could send a notification instead of a request, which does not contain an id field, and therefore the client does not have to wait for a response message from the platform. This reduces some of the overhead that occurs when the platform sends a response message to the video game. However, like the REST architectural style, there still needs to be a connection opening and closing when a message travels from the video game to the platform. But in general, JSON-RPC makes the procedure of data storing a lot lighter than using the REST architectural style.

Now that we have seen how REST and JSON-RPC, it's time to discuss GraphQL, and how it performs compared to the other two architectural styles. Similarly to the other cases, the video game creates a request containing the exact mutation that needs to be made, that is, the fields that need to be stored. This method of data transfer could be relatively faster than any of the other architectural types, because it eliminates the problem of over-fetching and under-fetching data. This means that the server could be receiving more data than necessary, which increases the network traffic, or the server could be receiving less data than needed, which then means that the video game will have to make multiple requests in order to send all the

data that needs to be stored. In conclusion, this architectural style is light-weight compared to the REST architectural style, and is a good choice for the implementation of the platform.

3.5 Conclusion

As discussed above, all the technologies that were mentioned are well suited for the improvement of the communication between the platform and the video games. However, the REST(HTTP) architectural style, along with the JSON data format are chosen due to their compatibility with the programming languages on which the platform and the video games are written. The main issue that occurs when integrating technologies other than REST (HTTP) to the existing platform and the games, is that the programming languages in which the existing platform and the games are written, do not support libraries for these technologies, and for technologies where the associated libraries exist, the libraries are not maintained enough in order to be as stable as the libraries and tools for the integration of the more popular REST(HTTP) technology.

In the next chapter, we proceed to improve the existing platform by making use of the data analytics that this platform stores. The existing platform will be extended by implementing a new platform that takes the data analytics and presents them with visual aids.

Chapter 4

Design and Implementation of Data Presentation Platform

4.1 Introduction	26
4.2 Data Protection	26
4.3 Hardware	27
4.4 Software	28
4.4.1 Server	29
4.4.2 Markup	29
4.4.3 Styling	30
4.4.4 Client-Side Script	30
4.4.5 Server-Side Scripts	33

4.1 Introduction

This chapter presents the design and implementation of the Data Presentation Platform. As mentioned in previous chapters, the purpose of this platform is to read the data analytics of students which is stored by an already existing platform, and then present them graphically, so that the end users of this platform can come into conclusions regarding the performance of students on various games. The users of this platform are the researchers and the teachers who are helping students with reading difficulties. The Data Presentation Platform makes easier for researchers and teachers to track students' performance and come into more accurate conclusions. In this chapter, the technologies used for the implementation of this platform is discussed.

4.2 Data Protection

The purpose of the Data Presentation Platform is to receive sensitive data from a database and use graphical formats to display useful information to the end user. One thing that has to be taken into consideration, is the protection of these sensitive data on the writing of this dissertation. Under the General Data Protection Regulation (GDPR), the data that will be displayed in this dissertation have to go through a form of anonymization[11]. The method

that was used for this process is the Pseudonimization method. In order to have a better understanding of this method, an example will be used. Inside the database of the existing platform, each student is identified by a user id and the name of the user. The process of pseudonimization changes the information that identifies the user, which in our case, is a unique number or name. Each user id will be mapped into a different one, which means that if the user is identified by a unique name, then that user will have his name changed. This way, the user cannot be identified, unless the mappings are known by someone.

In order to perform pseudonimization on the data for the purposes of the dissertation, the data of the existing platform is replicated. Then, the replicated data is anonymized using the pseudonimization method which is described above. Finally, the new platform which is implemented in this dissertation will use the replicated anonymized data. That way, the implementation has realistic data, without exposing the students' performance inside the games.

4.3 Hardware

In order to start with the implementation of the platform, we need to determine where the source code and the data of the platform will be stored and running. For this dissertation, it was required for the Data Presentation Platform to be implemented in the same environment that the data storing platform was implemented. In order to be able to integrate the Data Presentation Platform inside the existing one, we should use the same programming languages and technologies that the existing one used. For this dissertation, it was not possible to develop directly on the existing platform, and for that reason, the existing platform had to be replicated on a different environment, so that we could work on the replica. The replicated platform is running on a Linux machine running CentOS, which is provided by the University of Cyprus for the purpose of this dissertation, and this is where the new platform is integrated.

4.4 Software

It has been decided for the Data Presentation Platform to use the same software technologies that were used in the existing platform. In figure 4.1, the architecture of this new platform is visually presented. When the end-user enters the website of the platform in the browser's address bar, the browser requests the platform from the apache server. By default, the apache server looks for the "index.php" PHP file and executes it. The "index.php" PHP file then renders the "home.html" HTML file, which also includes stylings and client-side scripts. When dynamic content needs to be rendered with data from the database, the client-side script makes calls to the server-side scripts, which in turn make calls to the database in order to retrieve or manipulate data.

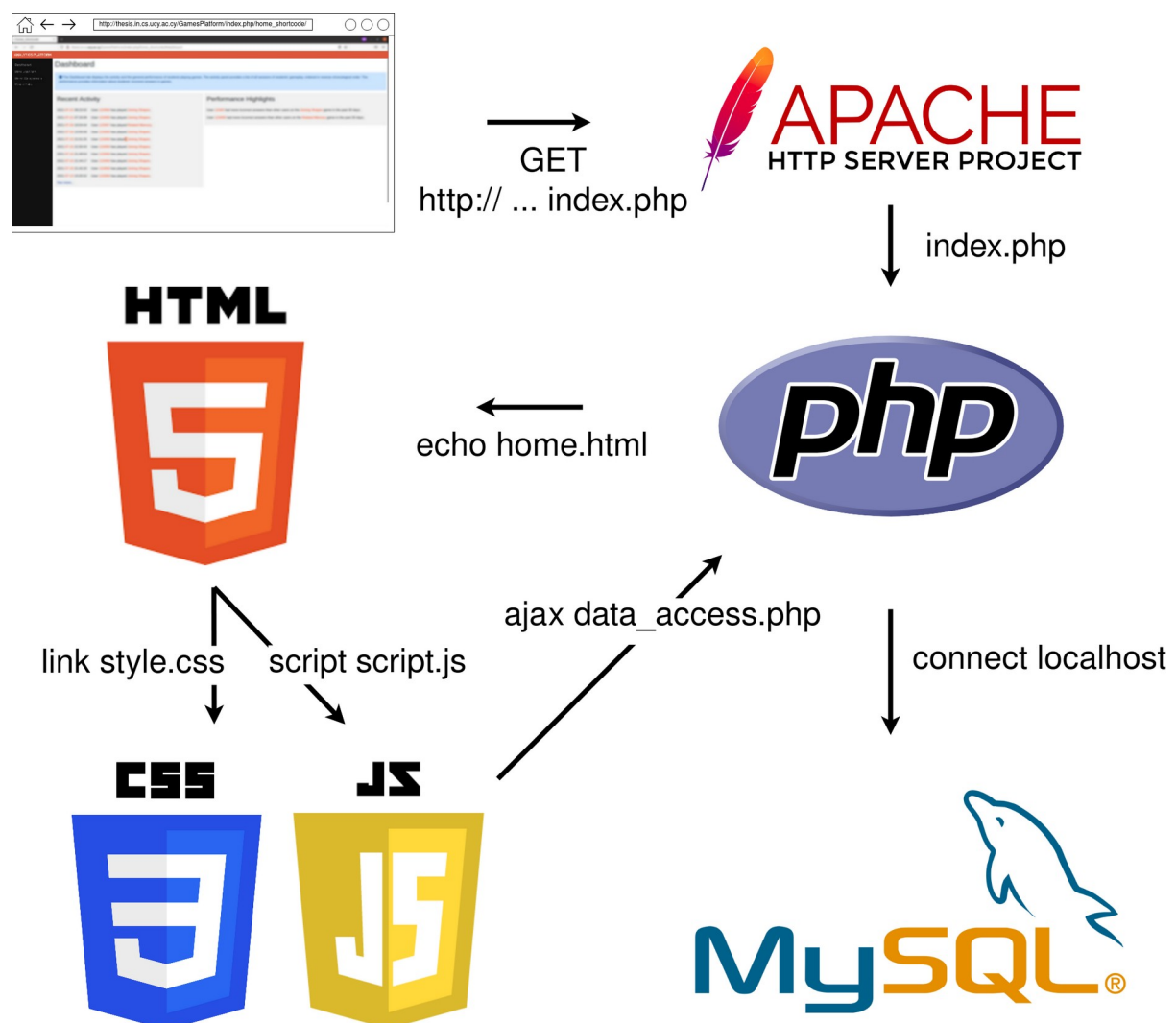


Figure 4.1: Architecture of the data presentation platform

4.4.1 Server

The server software which is used for the existing platform is the Apache server. For data access and manipulation, the MySQL query language is used, along with phpMyAdmin for the administration of the database.

After setting up the Apache server and the MySQL database, the next step is the installation of WordPress on the server. WordPress is a content management system which is integrated inside a server in order to provide features like plugins and templates. The reason that WordPress is used in the platform is to easily integrate source code inside a page using a WordPress feature which is called “shortcode”.

After setting up WordPress, all that is left to be done is the implementation of the platform using the same markup and programming languages that the existing platform was using. Initially, we start with an “index.php” file which is the default file that the Apache server looks for when the server is running.

4.4.2 Markup

The “index.php” file contains a function that renders an HTML file named “home.html”, where the markup of the platform is written. The function that renders the platform is mapped to a shortcode, which is used for the integration of the platform inside WordPress. The source code of the “index.php” file can be found in the Appendix A. Inside the “home.html” file, the initial structure of the platform is implemented. Along with the initial structure, several files are imported into the HTML file, including styling and scripts for the better design and proper functionality of the platform. For styling, the “Bootstrap” library was included [12] along with a custom CSS file, which was manually implemented. For functionality, the “Chart.js” script was included [13], which builds charts based on given data. Also, a manually implemented script file was included for the rest of the platform’s functionality. After the inclusion of styling and functionality follows the structure of the platform, which takes the form of a dashboard. The dashboard consists of 3 main elements, the top navigation bar, the sidebar, and the main window. The dashboard consists of 4 tabs, each giving different information to the end user. The tab functionality was manually implemented using HTML and JavaScript. The HTML markup defines the structure of each tab, while the JavaScript source implements the tab functionality, where clicking one tab switches between tabs correctly, displaying the contents of the tab inside the main window. In figure 4.2, a visual representation of the platform's structure is displayed.

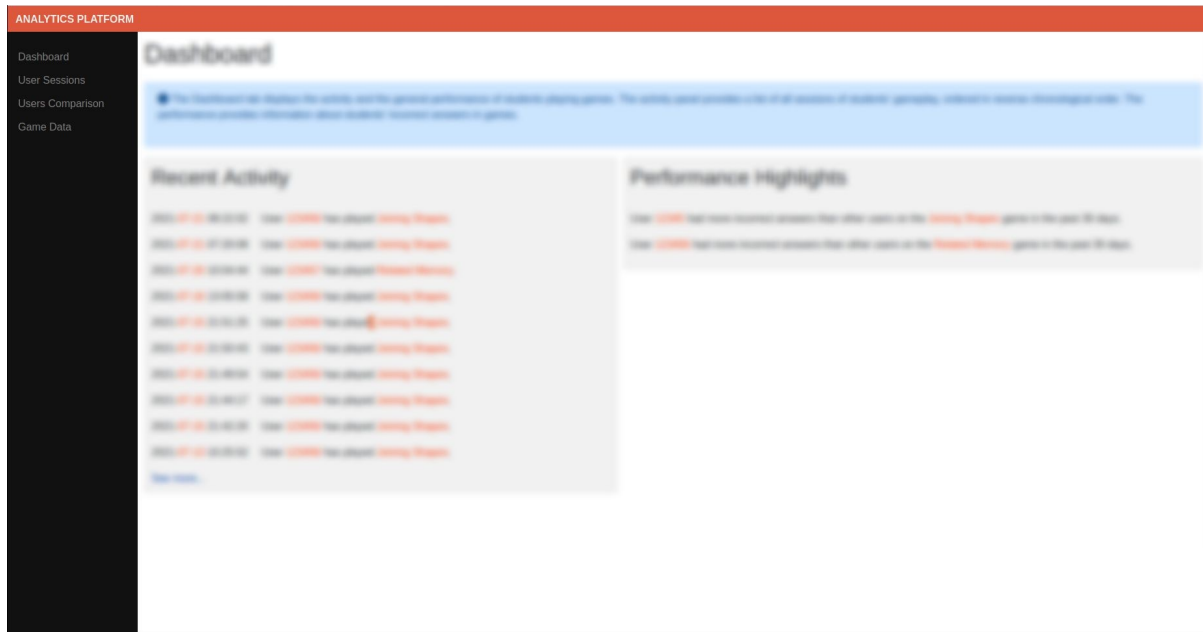


Figure 4.2: Initial structure of the data presentation platform

The first tab displays the main dashboard window, where general information is displayed about the students' activity. The second and third tab consist of a form that needs to be filled with desired information, and when filled, a series of charts are displayed below the form. The final tab consists of a table which provides more detailed information to the end user. The source code of the "home.html" file are given in the Appendix B.

4.4.3 Styling

The styling of the platform consists of two components, which is the external Bootstrap styling library, paired with the manually implemented CSS file named "style.css". Initially, the Bootstrap library changes the default styling format of the browser. Additionally, Bootstrap classes are used inside the HTML file for styling buttons and input boxes, along with tables and icons that appear on the platform. The rest of the styling is determined by the manually implemented "style.css" file. This file styles the majority of the platform, as it determines the placement, size and colors of nearly all components inside of it. The source code of the styling of the website can be found inside Appendix C.

4.4.4 Client-Side Script

All the client-side scripts that the platform executes are placed inside a single JavaScript file named "script.js". This script file determines the behavior of all windows, forms and buttons inside the platform. It is also responsible for building dynamic elements that are not initially

existent inside the platform, such as logs, charts, tables and other information received from a database. This script file is also responsible for receiving data analytics from the server-side scripts, which is covered in the next section. It uses logic to manipulate this data analytics and turn them into useful information in order for them to be presentable inside the platform.

For the activity panel in the dashboard tab, the client-side script requests the data from the server-side script about the activity of users. The request is made using the "data_access_call" function that was implemented for requesting data more easily.

```
function data_access_call(action, params) {  
    var xhttp = new XMLHttpRequest();  
    var response;  
    xhttp.onreadystatechange = function () {  
        if (this.readyState == 4 && this.status == 200) {  
            response = JSON.parse(xhttp.responseText);  
        }  
    };  
    xhttp.open("GET", "http://thesis.in.cs.ucy.ac.cy/GamesPlatform/wp-content/plugins/ucy-analytics-platform/data_access", true);  
    xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
    xhttp.send();  
    return response;  
}
```

Figure 4.3: Source code of the function that calls the server-side scripts for data access

Then, it uses the data requested from the server-side script in order to build the contents inside the activity panel. The way the content is built is by appending HTML markup inside elements that work as placeholders, which is written using the data retrieved from the database.

```
function build_dashboard(full_activity = false, full_performance = false) {  
    var activity = data_access_call('activity', {});  
  
    var dashboard_activity = document.getElementById('dashboard-activity');  
    var dashboard_performance = document.getElementById('dashboard-performance');  
    dashboard_activity.innerHTML = '<h2>Recent Activity</h2><br>';  
    dashboard_performance.innerHTML = '<h2>Performance Highlights</h2><br>';  
  
    var activity_counter = 0;  
    for (var a of activity) {  
        dashboard_activity.innerHTML += `  
        <p>${a.timestamp.slice(0, 5)}<span class="scarlet">${a.timestamp.slice(5, 10)}</span>  
        ${a.timestamp.slice(10, 19)}<span class="scarlet">${a.userid}</span> has played  
        <span class="scarlet">${a.game}</span></p>  
        `;  
        activity_counter++;  
        if (activity_counter == 10 && !full_activity) {  
            dashboard_activity.innerHTML += `<a id="activity-more" onclick="build_dashboard(true, ' + full_performance + '>View More</a>`;   
            break;  
        }  
    }  
}
```

Figure 4.4: Source code for the dynamic build of the activity panel

The same procedure happens when building the performance panel, only this time, further logic executed on the retrieved data for determining the performance of students.

For the user sessions tab, the client-side script receives data regarding the performance of a student for each of his sessions. The data is retrieved using the “data_access_call” function that was mentioned earlier. Then, it uses the retrieved data to build the charts and present additional information regarding the student’s last session.

```

var charts_info = document.getElementById('charts-info')
charts_info.innerHTML = "<br><h2>User's last session results:</h2>";
if (game == 'joining_shapes') {
    if (option == 'user_progress') {
        var user_mistakes_per_session = data_access_call('charts/joining_shapes/user_mistakes_per_session', data_access_call);
        var user_timing_per_session = data_access_call('charts/joining_shapes/user_timing_per_session', data_access_call);
        document.getElementById('compare_users_charts').innerHTML = '';
        document.getElementById(option + '_charts').innerHTML = `
            <div class="chart-outer-div"><div class="chart-inner-div"><canvas class="chart" id="chart1"></canvas></div>
            <div class="chart-outer-div"><div class="chart-inner-div"><canvas class="chart" id="chart2"></canvas></div>
        `;
        build_chart('chart1', 'User Incorrect Moves Per Session', user_mistakes_per_session['x'], user_mistakes_per_session['y']);
        build_chart('chart2', 'User Timing Per Session', user_timing_per_session['x'], user_timing_per_session['y']);

        var chart_percentages = get_chart_percentages(user_mistakes_per_session);
        charts_info.innerHTML += (chart_percentages == 'no_previous_sessions') ? "" : '<p>The user\'s <span class="s'
        var chart_percentages = get_chart_percentages(user_timing_per_session);
        charts_info.innerHTML += (chart_percentages == 'no_previous_sessions') ? "" : '<p>The user\'s <span class="s'

    } else if (option == 'compare_users') {
        var users_mistakes_comparison = data_access_call('charts/joining_shapes/users_mistakes_comparison', data_access_call);
        var users_timings_comparison = data_access_call('charts/joining_shapes/users_timings_comparison', data_access_call);
        document.getElementById('user_progress_charts').innerHTML = '';
        document.getElementById(option + '_charts').innerHTML = `
            <div class="chart-outer-div"><div class="chart-inner-div"><canvas class="chart" id="chart1"></canvas></div>
            <div class="chart-outer-div"><div class="chart-inner-div"><canvas class="chart" id="chart2"></canvas></div>
        `;
        build_chart('chart1', "Average Users' Incorrect Moves Comparison", users_mistakes_comparison['x'], users_mistakes_comparison['y']);
        build_chart('chart2', "Average Users' Timings Comparison", users_timings_comparison['x'], users_timings_comparison['y']);
    }
}

```

Figure 4.4: Source code for charts building for each game

For the users comparison tab, the same methods are used for building the charts and for displaying the additional information regarding the student’s last session.

For the session details tab, the “data_access_call” function is used again in order to retrieve data for the table. Then, the table is built by appending HTML markup inside the table elements.

The source code for the client-side script can be found in Appendix D.


```

function build_chart(chart_id, label, labels, data, is_boolean = false, is_percentage = false) {
  var config = {
    type: 'line',
    data: {
      labels: labels,
      datasets: [{
        label: label,
        backgroundColor: 'rgb(255, 99, 132)',
        borderColor: 'rgb(255, 99, 132)',
        data: data,
        tension: 0.1
      }]
    },
    options: is_boolean ? {
      tooltips: {
        callbacks: {
          label: tooltipItem => tooltipItem.value < 0.5 ? 'false' : 'true'
        }
      },
      scales: {
        yAxes: {
          ticks: {
            callback: value => {
              if (value == 0) {
                return 'Incorrect';
              } else {
                return value == 1 ? 'Correct' : '';
              }
            }
          }
        }
      }
    } : is_percentage ? {
      scales: {
        y: {
          min: 0
        }
      }
    } : {}
  };

  new Chart(chart_id, config);
}

```

Figure 4.5: Function called for building charts

4.4.5 Server-Side Scripts

The server-side scripts are composed using the PHP scripting language and they are responsible for fetching data from the database. These files fetch several types of data such as student logs, form field metadata, chart values and table data. Minimal data processing is happening inside the server-side scripts, as their purpose is to just fetch data and send them to the client-side script, while the client-side does the majority of the processing.

The scripts used for data retrieval use a common script which creates a connection, and then each script uses this connection to retrieve different data.

```

<?php

ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

$DB_NAME = "gamesPlatform";
$MYSQL_USERNAME = "gamesPlatform";
$MYSQL_PASSWORD = "DBSbUAcbBzuMUJxs";
$HOST_SERVER = "localhost";

$conection = mysqli_connect($HOST_SERVER, $MYSQL_USERNAME, $MYSQL_PASSWORD, $DB_NAME) or die ('Error connecting to data

```

Figure 4.6: Server-side script for creating database connection

For the dashboard tab, there are two scripts responsible for retrieving data, one for the recent activity panel and two for the performance panel. For the activity panel, an SQL query is executed, which retrieves the last month's sessions from the two games that the platform currently supports. Then, it sorts the sessions of both games, from the most recent sessions to the oldest. Finally, it returns the information to the client-side script.

```
<?php
include 'connection.php';

$activity = $connection->query("
SELECT
    userid,
    game,
    timestamp
FROM (
    SELECT
        userid,
        'Joining Shapes' AS game,
        timestamp
    FROM
        joiningshapespro_analytics
    UNION
    SELECT
        userid,
        'Related Memory' AS game,
        timestamp
    FROM
        memorypro_analytics
) AS activity
WHERE
    timestamp >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
ORDER BY
    timestamp DESC"
)->fetch_all(MYSQLI_ASSOC);

echo json_encode($activity);
?>
```

Figure 4.7: Server-side script for receiving game activity

For the performance panel, two server-side scripts are called, one for each game, each one collecting the number of incorrect moves of students for the particular game. The SQL query calculates for each student the average amount of incorrect moves that he had throughout his sessions in the last 30 days. Then, the script changes the form of the data for the client-side script to process it more easily, and then if finally returns it.

```

?php
include 'connection.php';

$query = "
    SELECT
        |   userid,
        |   AVG(numberOfIncorrectUserMoves) AS average_incorrect_moves
    FROM
        |   joiningshapespro_analytics
    GROUP BY
        |   userid
    ORDER BY
        |   userid
";

if (!($query_execution = $connection->query($query))) {
    echo "SQL Error: " . $connection->error;
    die();
};

$data = $query_execution->fetch_all(MYSQLI_ASSOC);

$final = array('x' => array(), 'y' => array());

foreach ($data as $record) {
    array_push($final['x'], $record['userid']);
    array_push($final['y'], $record['average_incorrect_moves']);
}

echo json_encode($final);
?

```

Figure 4.8: Source code for receiving average number of incorrect moves for each user

For the user sessions and the users comparison tabs, there is a need for scripts that retrieve data for the charts. For each game and for each statistic, a different script is executed for the retrieval of chart data, however, all scripts are very similar with each other. The student's number of incorrect moves per session for the joining shapes game, will be seen as an example. The server-side script receives the data from the form that the end user filled, and then uses it to build the SQL query. Then, the query fetches the timestamp and the amount of incorrect moves of each session for the student specified inside the form. The sessions that are fetched must also be inside the range of the dates that are specified inside the form, and must also be of the game's level that is selected inside the form. After the execution of the query, the script changes the form of the data in order for it to be properly received from the client-side script, and finally the data is sent.

```

<?php

include '../connection.php';

$dL = $_GET["dL"];
$item = $_GET["item"];
$user_id = $_GET["user_id"];
$from_date = $_GET["from_date"];
$to_date = $_GET["to_date"];

$query = "
    SELECT
        timestamp,
        numberOfIncorrectUserMoves
    FROM
        joiningshapespro_analytics
    WHERE
        DL = ' " . $dL . " '
        AND itemSet = ' " . $item . " '
        AND userid = ' " . $user_id . " '
        ((($from_date) ? (' AND timestamp >= ' " . $from_date . " ') : '')) .
        ((($to_date) ? (' AND timestamp <= ' " . $to_date . " 23:59:59") : '')) . " '
    ORDER BY timestamp
";

if (!($query_execution = $connection->query($query))) {
    echo "SQL Error: " . $connection->error;
    die();
};

$data = $query_execution->fetch_all(MYSQLI_ASSOC);

$final = array('x' => array(), 'y' => array());

foreach ($data as $record) {
    array_push($final['x'], str_replace(' ', ' ', substr($record['timestamp'], 0, -3)));
    array_push($final['y'], $record['numberOfIncorrectUserMoves']);
}

echo json_encode($final);

?>

```

Figure 4.9: Server-side script for retrieving number of incorrect moves for each session of a single user

For the session details tab, there are two scripts fetching table data, one for each game, but both scripts have similar form. For demonstration, the script for the joining shapes game will be explained. Initially, an SQL query is executed, which gathers many fields from a table that stores information about sessions of the joining shapes game, such as the session's user id, the level of the game, the time required for finishing the session, the amount of incorrect moves executed in that session, and finally the session's timestamp. After fetching the data, it then sends it to the client-side script, without doing further processing of the data.

```

?php
include 'connection.php';

$query = "
    SELECT
        userid,
        DL,
        itemSet,
        userTime,
        numberOfIncorrectUserMoves,
        timestamp
    FROM
        joiningshapespro_analytics
    ORDER BY
        timestamp DESC
";

if (!($query_execution = $connection->query($query))) {
    echo "SQL Error: " . $connection->error;
    die();
};

$data = $query_execution->fetch_all(MYSQLI_ASSOC);

echo json_encode($data);
?

```

Figure 4.10: Server-side script for retrieving data in more detail for the joining shapes game

For the related memory game, the data is retrieved in a very similar way, but having the data fetched from a different table.

A subset of the source code for the server-side scripts can be found on the Appendix E.

Chapter 5

Use-Case Demonstration

5.1 Introduction	38
5.2 Dashboard	38
5.3 User Sessions	39
5.4 Users Comparison	41
5.5 Session Details	42

5.1 Introduction

This chapter provides a use-case demonstration, where the full functionality of the platform is demonstrated. Each functionality has a different purpose, that is, each tab provides different tools that help the end user find useful information. The platform has a form of a dashboard where each tab provides different tools, and each tool provides information with the appropriate graphical formats. In the next sections, each tab is demonstrated, their purpose is explained along with how they must be used by the end user.

5.2 Dashboard

The first tab is the default tab which is initially shown to the end user when he loads the website. This tab displays general information about the activity of the user. At the top of the main window, the title of the tab is displayed, along with explanation on what kind of information is displayed on this tab.

Below, two windows are displayed side by side. The left window displays a log that lists recent activity from users' gaming sessions. The recent activity log lists activity from the last 30 days, and the order in which this information is listed is reverse chronological. This window might be useful when the end user wants to know which students have recent activity. That way, the end user can track the performance of only the students who have played recently.

The right window displays information about the performance of students' gaming sessions. Specifically, it lists information about students who are having more incorrect answers than other students in any game in the last 30 days. This window can be useful to the end users when they want to identify which students need more attention than others.

Generally, this tab helps the end user catch up with the student's recent activity, it gives an idea about which students are active, and which ones need more attention. The figure 5.1 shows how the dashboard is displayed along with the activity and performance panels.

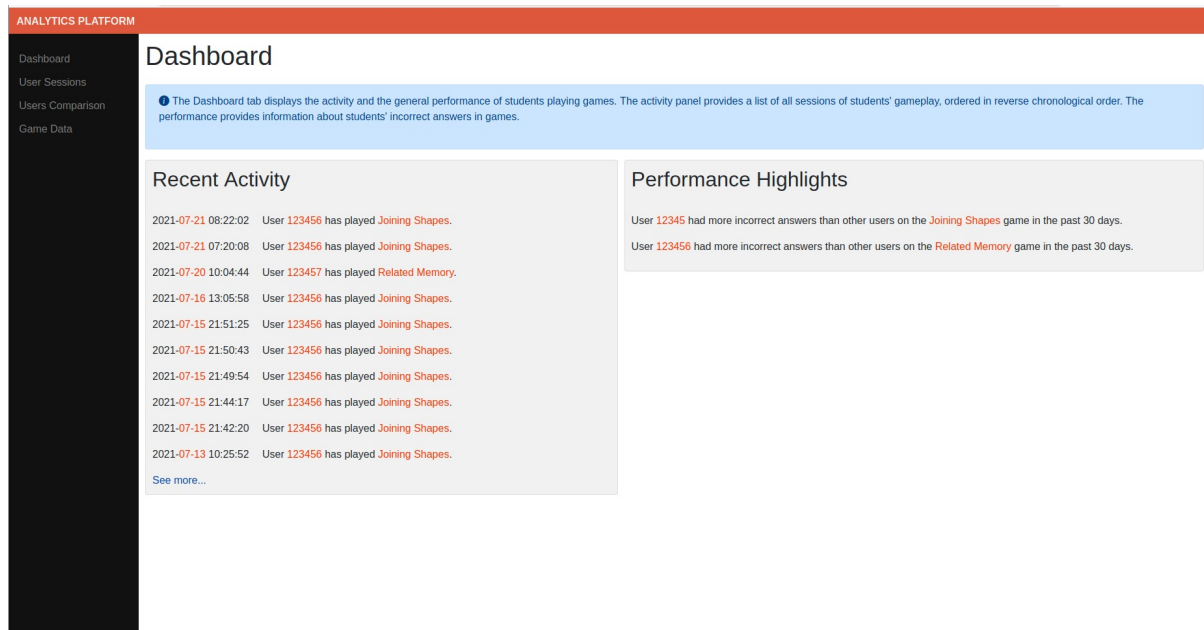


Figure 5.1: Activity and performance panels

5.3 User Sessions

The user sessions tab displays information about gaming sessions for a specific student. The end user fills a form that requires the student's id, the desired game for which we want information, a level of that game, and optionally, a date range. Then, the end user presses the "View" button, which will then display charts regarding the performance of the student on the specified game and level on each of the student's sessions.

Figure 5.2: The form which is filled for retrieving information for the sessions of a user

Because each game is of different nature, it makes sense for the information displayed for each game to be different. In order to demonstrate the information which is presented by the tool, a single student will be chosen, along with a game that he played a few times.

The image above displays the results that the tool displays after the end user fills the form with the desired filters and presses the “View” button. The tool provides a set of charts, each one displaying the values of a single statistic throughout the sessions of the student. Examples of statistics can be the amount of incorrect moves that a single student has made, or the time it took for the student to finish the session. Each point of a chart represents the score on a statistic for a single session. It is noticed that the sessions are sorted in the charts in chronological order, which visually helps the end user understand whether the performance on a single statistic of a student has improved or not. Additionally, the tool provides some conclusions about the student’s last session, compared to the previous sessions. These comments also help the end user to understand how the performance of a student is changing throughout the sessions.

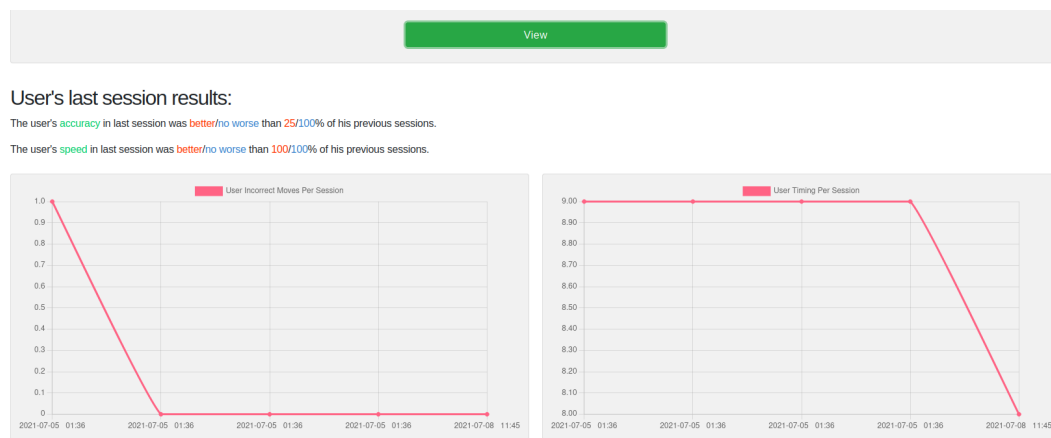


Figure 5.3: The information displayed for the user's sessions when submitting the form

5.4 Users Comparison

The users comparison tab has a similar purpose to the user sessions tab, which is to help the end user keep track of the students' performance. However, this tool, instead of comparing the sessions of a single student, it compares the average performance of all students. The users comparison tool requires from the end user to fill a form with the desired game, the game's level, and optionally, a date range. When the end user fills the form with the desired information and presses the "View" button, the relevant charts will appear.

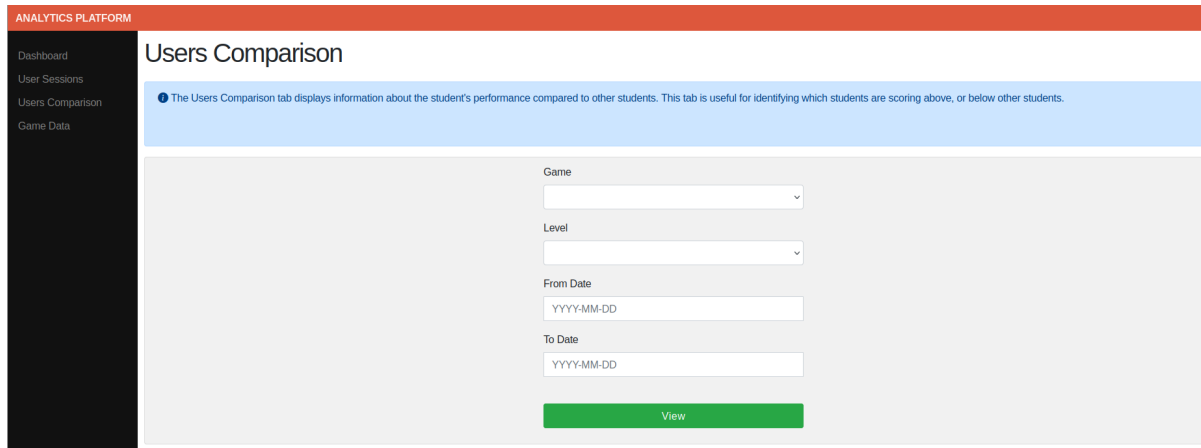


Figure 5.4: The form that needs to be filled for retrieving information that compares users

Each chart represents the average scores of each student for a single statistic. Again, examples of statistics are the amount of incorrect moves that the student has made in a session, or the time it took for the student to finish a session. These examples are taken from one of the games that was integrated into the new platform, Joining Shapes. This way, the end user can easily notice which students are having trouble catching up with the rest of the students, so that they can spend more time with those students.



Figure 5.5: The information displayed that compares users based on statistics for the Joining Shapes game

However, not all games have the same statistics, due to each game having different nature. Another example that will be shown is the Related Memory statistics.

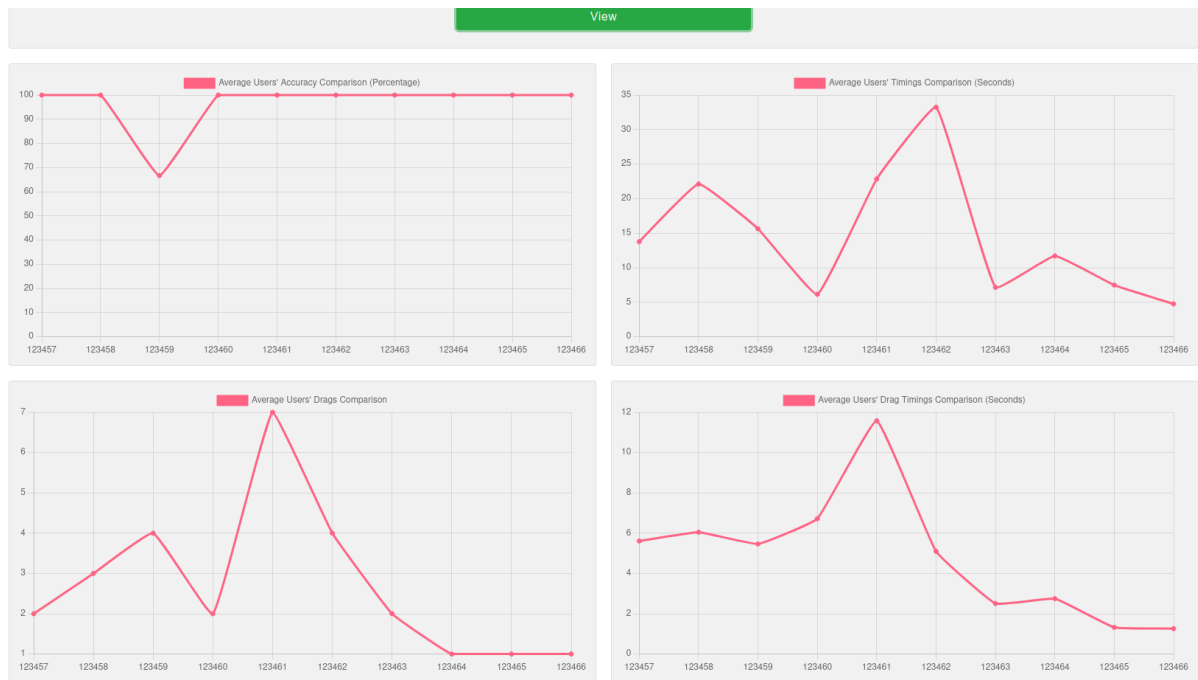


Figure 5.6: Statistics displayed for the users' comparison on the Related Memory game

5.5 Session Details

The session details tab provides a more raw form of information. This tool is used for the end users who did not find the information they needed in the other tabs. In order to use this tool, the end user simply needs to select a game, and then press “View”.

After that, a table is displayed to the end user, where rows represent sessions. Additionally, the table provides search boxes, where the end user can search keywords for a specific field. By default, the search boxes look for rows where the value contains the keyword, unless the character “=” is added as a prefix to the keyword, where the search box searches for exact matches.

Game Data

The Game Data tab displays information about the students' gameplay in more detail. In case that the information required cannot be found in other tabs, they can be found here.

Game: Joining Shapes

View

In order to search for exact matches, use the "~" character before a keyword.

Search for users... Search for DLs... Search for Items... Search for Timings... Search for Incorrect mov Search for Dates...

User ID	DL	Item Set	User Time	Incorrect Moves	Date
123456	DL1	1a	11	0	2021-07-21 08:22:02
123456	DL1	1a	10	0	2021-07-21 07:20:08
123456	DL1	1a	10	0	2021-07-16 13:05:58
123456	DL1	2a	13	0	2021-07-15 21:51:25
123456	DL1	2a	16	1	2021-07-15 21:50:43
123456	DL1	2a	34	0	2021-07-15 21:49:54
123456	DL1	2a	35	0	2021-07-15 21:44:17
123456	DL1	1a	9	0	2021-07-15 21:42:20

Figure 5.7: Session Details tab displaying information in more detail

Chapter 6

Conclusion

6.1 Introduction	44
6.2 Conclusion	44
6.3 Future Work	45

6.1 Introduction

This chapter concludes this dissertation by summarizing the overall work that has been done in the previous chapters of the dissertation. There is a brief discussion about the internet technologies that have been studied. Also, this chapter talks through the implementation of the data presentation platform. Finally, this chapter discusses the future work on this subject, that is, the ways in which the platform can be expanded and further improved.

6.2 Conclusion

In this dissertation, several internet technologies have been studied. These internet technologies have been divided into categories based on their purpose, and for each category, the more appropriate ones were supposed to be chosen for the improvement of the platform that stores data analytics. However, it was concluded that there is no such need for improvement, due to the fact that the existing platform is using the appropriate technologies that were chosen. Afterwards, the dissertation moved on with the implementation of a new platform that uses the data stored from the existing platform, and uses graphical formats to present them to the teachers and researchers, so that they can get useful information about their students' state. The new platform was implemented, and it displays information which helps the end users keep track of the students' performance, and have a sharper image on which students need more time and attention than others.

6.3 Future Work

For future work, the platform that was implemented in this dissertation can be extended with any tools that the teachers and researchers think would be necessary. A login system can be added into the new platform, in order for teachers to be identified by the new platform. Each teacher can have his own set of students, and the new platform could be giving each teacher information only for his own students. Also, the platform can be extended with more games and levels for each game, which can be added with just a few lines of code. Additionally, with the extension of the new platform with new games, new information will need to be displayed, and probably new graphical formats might need to be introduced as well. Generally, the implementation of the data presentation platform was made in a way that it can be extended efficiently with new features and games, which makes the future work of this dissertation easier.

Bibliography

- [1] Springer Verlag, “Web Services Concepts, Architectures and Applications”, [Online], Available:
<https://people.inf.ethz.ch/alonso/WebServicesBook.html>
- [2] Haroon Shakirat Oluwatosin, “Client-Server Model”, [Online], Available:
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1083.8741&rep=rep1&type=pdf>
- [3] Olivier Bonaventure, “Computer Networking: Principles, Protocols and Practice”, [Online], Available:
http://www.opentextbooks.org.hk/system/files/export/3/3547/pdf/Computer_Networking_Principles_Protocols_and_Practice_3547.pdf
- [4] Vanessa Wang, Frank Salim, Peter Moskovits, “The Definitive Guide to HTML5 WebSocket”, [Online], Available:
[http://sd.blackball.lv/library/The_Definitive_Guide_to_HTML5_WebSocket_\(2013\).pdf](http://sd.blackball.lv/library/The_Definitive_Guide_to_HTML5_WebSocket_(2013).pdf)
- [5] W3C, “Extensible Markup Language (XML) 1.0 (Second Edition)”, [Online], Available:
<https://www.w3.org/TR/2000/REC-xml-20001006.pdf>
- [6] Tom Marrs, “JSON at Work”, [Online], Available:
<http://projanco.com/Library/JSON%20at%20Work.pdf>
- [7] Alen Šimec, Magdalena Magličić, “Comparison of JSON and XML Data Formats, [Online], Available:
https://www.researchgate.net/publication/329707959_Comparison_of_JSON_and_XML_Data_Formats

- [8] MarkLogic, “MarkLogic Server: Rest Application Developer’s Guide”, [Online], Available:
<https://docs.marklogic.com/guide/rest-dev.pdf>

- [9] Microsemi, “Application Note JSON-RPC”, [Online], Available:
http://ww1.microchip.com/downloads/en/Appnotes/ENT-AN1126-4.3_VPPD-04260.pdf

- [10] GraphQL, “Introduction to GraphQL”, [Online], Available
<https://graphql.org/learn/>

- [11] Nils Gruschka, Vasileios Mavroeidis, Kamer Vishi, Meiko Jensen, “Privacy Issues and Data Protection in Big Data: A Case Study Analysis under GDPR”, [Online], Available
<https://arxiv.org/pdf/1811.08531.pdf>

- [12] Bootstrap, “Build fast, responsive sites with Bootstrap”, [Online], Available
<https://getbootstrap.com/>

- [13] Chart.js, “Simple yet flexible JavaScript charting for designers & developers”, [Online], Available
<https://graphql.org/learn/>

Appendix A

index.php:

```
1. <?php
2. /**
3.  * Plugin Name: Analytics Platform
4.  * Plugin URI: http://thesis.in.cs.ucy.ac.cy/GamesPlatform/analytics-platform
5.  * Description: Description for analytics platform
6.  * Version: 1.0
7.  * Author: Antonis Katsiantonis
8.  * Author URI: http://www.google.com
9.  */
10.
11. add_shortcode('home_shortcode', 'home');
12.
13. function home() {
14.     echo file_get_contents('http://thesis.in.cs.ucy.ac.cy/GamesPlatform/wp-content/
plugins/ucy-analytics-platform/home.html');
15. }
16.
17. ?>
```


Appendix B

home.html:

1. <!DOCTYPE html>
2. <html lang="en">
- 3.
4. <head>
5. <meta charset="UTF-8" name="viewport" content="width=device-width, initial-scale=1">
- 6.
7. <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
8. integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
- 9.
10. <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css">
- 11.
12. <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
- 13.
14. <link rel="stylesheet"
href="http://thesis.in.cs.ucy.ac.cy/GamesPlatform/wp-content/plugins/ucy-analytics-platform/style.css">
- 16.
17. <script type="text/javascript"
src="http://thesis.in.cs.ucy.ac.cy/GamesPlatform/wp-content/plugins/ucy-analytics-platform/script.js"></script>
- 19.
20. <title>Analytics Platform</title>
21. </head>
- 22.
23. <body onload="switch_option('dashboard');">
- 24.

```

25. <div id="navigation-bar">
26.     <span id="navbar-title"><a id="title"
27.
href="http://thesis.in.cs.ucy.ac.cy/GamesPlatform/index.php/home_shortcode/">ANALYTIC
S
28.         PLATFORM</a></span>
29. </div>
30.
31. <div class="sidenav">
32.         <a id="dashboard-button" href="#dashboard"
onclick="switch_option('dashboard');">Dashboard</a>
33.         <a href="#user-sessions" onclick="switch_option('user_progress');">User
Sessions</a>
34.         <a href="#users-comparison" onclick="switch_option('compare_users');">Users
Comparison</a>
35.         <a href="#game-data" onclick="switch_option('game_data');">Session Details</a>
36. </div>
37.
38. <div id="main" class="main">
39.     <div id="dashboard-content" hidden>
40.         <h1 class="header">Dashboard</h1>
41.         <div id="dashboard-user-info" class="tab-info">
42.             <div class="alert fixed-size-alert alert-primary">
43.                 <i class="bi bi-info-circle-fill"></i> The Dashboard tab displays the activity
and the general
44.                 performance of students playing games. The activity panel provides a list of
all sessions of
45.                 students' gameplay, ordered in reverse chronological order. The performance
provides information
46.                 about students' incorrect answers in games.
47.             </div>
48.         </div>
49.
50.         <div id="dashboard-activity-container">
51.             <div id="dashboard-activity"></div>
52.         </div>

```

```

53.     <div id="dashboard-performance-container">
54.         <div id="dashboard-performance"></div>
55.     </div>
56. </div>
57.
58. <div id="user-progress-content" hidden>
59.     <h1 class="header">User Sessions</h1>
60.     <div id="user-progress-user-info" class="tab-info">
61.         <div class="alert fixed-size-alert alert-primary">
62.             <i class="bi bi-info-circle-fill"></i> The User Sessions tab displays the
progress of students
63.             throughout their sessions. Depending on the game that will be chosen,
different charts will be
64.             displayed which aid in tracking the performance of students. The data in the
charts are ordered
65.             in chronological order.
66.         </div>
67.     </div>
68.
69.     <div class="form">
70.         <form>
71.             <div class="form-container">
72.                 <div id="user" class="form-group">
73.                     <label for="user_progress-user-id">User ID</label>
74.                     <input type="text" list="users-list" class="form-control"
id="user_progress-user-id"
75.                         placeholder="Testing users: 123456, 123458">
76.                     <datalist id="users-list">
77.                         <option value="123456"></option>
78.                         <option value="123458"></option>
79.                     </datalist>
80.                 </div>
81.                 <div id="user_progress-game" class="form-group">
82.                     <label for="user_progress-game-select">Game</label>
83.                     <select class="form-control" id="user_progress-game-select"
84.                         onchange="load_levels('user_progress');">

```

```

85.         <option value="joining_shapes">Joining Shapes</option>
86.         <option value="related_memory">Related Memory</option>
87.         <option disabled selected value=""></option>
88.     </select>
89. </div>
90. <div id="user_progress-level" class="form-group">
91.     <label for="user_progress-level-select">Level</label>
92.     <select class="form-control" id="user_progress-level-select">
93.         <option disabled selected value></option>
94.     </select>
95. </div>
96. <div id="user_progress-from-date" class="form-group">
97.     <label for="user_progress-from-date">From Date</label>
98.     <input type="text" class="form-control" id="from-date-user_progress"
99.         placeholder="YYYY-MM-DD">
100. </div>
101. <div id="user_progress-to-date" class="form-group">
102.     <label for="user_progress-to-date">To Date</label>
103.     <input type="text" class="form-control" id="to-date-user_progress"
placeholder="YYYY-MM-DD">
104. </div>
105.     <button id="view-button" type="button" class="btn btn-success option"
106.         onclick="view_charts('user_progress')">View</button><br><br>
107. </div>
108. </form>
109. </div>
110.
111. <div id="charts-info"></div>
112.
113.     <div id="user_progress_charts"></div>
114. </div>
115.
116. <div id="compare-users-content" hidden>
117.     <h1 class="header">Users Comparison</h1>
118.     <div id="user-progress-user-info" class="tab-info">
119.         <div class="alert fixed-size-alert alert-primary">

```

120. <i class="bi bi-info-circle-fill"></i> The Users Comparison tab displays
information about the

121. student's performance compared to other students. This tab is useful for
identifying which students

122. are scoring

123. above, or below other students.

124. </div>

125. </div>

126.

127. <div class="form">

128. <form>

129. <div class="form-container">

130. <div id="compare_users-game" class="form-group">

131. <label for="compare_users-game-select">Game</label>

132. <select class="form-control" id="compare_users-game-select"

133. onchange="load_levels('compare_users');">

134. <option value="joining_shapes">Joining Shapes</option>

135. <option value="related_memory">Related Memory</option>

136. <option disabled selected value=""></option>

137. </select>

138. </div>

139. <div id="compare_users-level" class="form-group">

140. <label for="compare_users-level-select">Level</label>

141. <select class="form-control" id="compare_users-level-select">

142. <option disabled selected value></option>

143. </select>

144. </div>

145. <div id="compare_users-from-date" class="form-group">

146. <label for="compare_users-from-date">From Date</label>

147. <input type="text" class="form-control" id="from-date-
compare_users"

148. placeholder="YYYY-MM-DD">

149. </div>

150. <div id="compare_users-to-date" class="form-group">

151. <label for="compare_users-to-date">To Date</label>

```

152.         <input type="text" class="form-control" id="to-date-compare_users"
placeholder="YYYY-MM-DD">
153.     </div>
154.     <button id="view-button" type="button" class="btn btn-success option"
155.         onclick="view_charts('compare_users')">View</button><br><br>
156. </div>
157. </form>
158. </div>
159.
160. <div id="compare_users_charts"></div>
161. </div>
162.
163. <div id="game-data-content" hidden>
164.     <h1 class="header">Game Data</h1>
165.     <div id="user-progress-user-info" class="tab-info">
166.         <div class="alert fixed-size-alert alert-primary">
167.             <i class="bi bi-info-circle-fill"></i> The Game Data tab displays
information about the students'
168.             gameplay in more detail. In case that the information required cannot be
found in other tabs, they
169.             can be found here. <br>
170.         </div>
171.     </div>
172.
173. <div class="form">
174.     <form>
175.         <div class="form-container">
176.             <div id="game_data-game" class="form-group">
177.                 <label for="game_data-game-select">Game</label>
178.                 <select class="form-control" id="game_data-game-select">
179.                     <option value="joining_shapes">Joining Shapes</option>
180.                     <option value="related_memory">Related Memory</option>
181.                     <option disabled selected value=""></option>
182.                 </select>
183.             </div>
184.             <button id="view-button" type="button" class="btn btn-success option"

```

```
185.         onclick="view_table();">View</button><br><br>
186.     </div>
187. </form>
188. </div>
189.
190.     <div id="loading-label" hidden>Loading...</div>
191.
192.     <div id="games-table-search-container"></div>
193.
194.     <div id="games-table-container"></div>
195. </div>
196.
197. </div>
198.
199. </body>
200.
201. </html>
```

Appendix C

style.css:

```
1. #navigation-bar {
2.     z-index: 2;
3.     position: fixed;
4.     width: 100%;
5.     height: 40px;
6.     padding: 8px 0px 0px 12px;
7.     background-color: rgba(221, 87, 60, 1);
8.     color: rgba(255, 255, 255, 0.9);
9. }
10.
11. #navbar-title {
12.     font-weight: bold;
13. }
14.
15. #title,
16. #title:hover {
17.     color: inherit;
18. }
19.
20. .sidenav {
21.     margin-top: 40px;
22.     height: 100%;
23.     width: 200px;
24.     position: fixed;
25.     z-index: 1;
26.     top: 0;
27.     left: 0;
28.     background-color: #111;
29.     overflow-x: hidden;
30.     padding-top: 20px;
31. }
```



```
32.
33. .sidenav a {
34.     padding: 6px 8px 6px 16px;
35.     text-decoration: none;
36.     font-size: 16px;
37.     color: #818181;
38.     display: block;
39. }
40.
41. .sidenav a:hover {
42.     color: #f1f1f1;
43. }
44.
45. .main {
46.     margin-left: 200px;
47.     padding-top: 40px;
48. }
49.
50. .tab-info {
51.     margin: 10px;
52. }
53. .header {
54.     padding: 10px;
55. }
56.
57. .form {
58.     border: 1px solid #ddd;
59.     margin: auto;
60.     margin: 10px;
61.     background-color: #f1f1f1;
62.     border-radius: 4px;
63. }
64.
65. .form-container {
66.     margin: auto;
67.     padding-top: 10px;
```

```
68.    max-width: 100%;
69.    width: 400px
70. }
71.
72. #page-title {
73.    text-align: center;
74.    margin: 50px;
75. }
76.
77. #form-options {
78.    text-align: center;
79. }
80.
81. .option {
82.    margin: 0px 0px 20px 0px;
83.    width: 100%;
84. }
85.
86. #form-fields {
87.    margin: auto;
88. }
89.
90. #view-button {
91.    margin: 25px 0px 0px 0px;
92. }
93.
94. #dashboard-activity-container {
95.    width: 45%;
96.    padding-right: 5px;
97.    padding-left: 10px;
98.    float: left;
99. }
100.
101. #dashboard-performance-container {
102.    width: 55%;
103.    padding-right: 10px;
```

```
104. padding-left: 5px;
105. float: right;
106. }
107.
108. #dashboard-activity {
109. border: 1px solid #ddd;
110. float: left;
111. background-color: #f1f1f1;
112. border-radius: 4px;
113. padding: 10px;
114. width: 100%;
115. }
116.
117. #dashboard-performance {
118. border: 1px solid #ddd;
119. float: left;
120. background-color: #f1f1f1;
121. border-radius: 4px;
122. padding: 10px;
123. vertical-align: text-top;
124. width: 100%;
125. }
126.
127. #games-table-container,
128. #games-table-search-container,
129. #loading-label {
130. max-width: 1300px;
131. width: 100%;
132. margin: auto;
133.
134. }
135.
136. .table-search-container {
137. float: left;
138. width: 16.6666%;
139. width: calc(100% / 6);
```

```
140. padding: 10px;
141. }
142.
143. .table-search-container-leftmost {
144. padding-left: 0px;
145. }
146.
147. .table-search-container-rightmost {
148. padding-right: 0px;
149. }
150.
151. .table-search {
152. float: left;
153. margin: 0px 10px 10px 0px;
154. }
155.
156. .chart-outer-div {
157. width: 50%;
158. float: left;
159. padding: 10px;
160. }
161.
162. .chart-inner-div {
163. border: 1px solid #ddd;
164. background-color: #f1f1f1;
165. border-radius: 4px;
166. padding: 10px;
167. }
168.
169. #charts-info {
170. padding-left: 10px;
171. }
172.
173. .fixed-size-alert {
174. min-height: 100px;
175. }
```

```

176.
177. .scarlet {
178.   color: rgba(255, 51, 0, 1);
179. }
180.
181. .light-blue {
182.   color: rgba(51, 127, 204, 1);
183. }
184.
185. .spring-green {
186.   color: rgba(0, 204, 102, 1);
187. }
188.
189. #activity-more, #performance-more {
190.   color: rgb(6, 69, 173);
191. }
192.
193. #activity-more:hover, #performance-more:hover {
194.   cursor: pointer;
195. }
196.
197. @media only screen and (max-width: 1400px) {
198.   .chart-outer-div {
199.     width: 100%;
200.   }
201. }
202.
203. /* While I was writing the dissertation, the font sizes of some elements decided to
change on their own.
204. I can't figure out why this happened, but the css below is a quick fix */
205.
206. html {
207.   font-size: 100%;
208. }
209.
210. input[type="text"] {

```

```
211. padding: .375rem .75rem;
212. font-size: 1rem;
213. }
214.
215. label, table{
216. font-size: 1rem;
217. }
```

Appendix D

script.js

```
1. function a() { return 'hello' }
2.
3.
4. function sidebar_navigate(option) {
5.     var sidebar_tabs = document.getElementById('main').children;
6.     for (var tab of sidebar_tabs) {
7.         if (tab.id === option.replaceAll('_', '-') + '-content') {
8.             tab.removeAttribute('hidden');
9.         }
10.        else {
11.            tab.setAttribute('hidden', 'true');
12.        }
13.    }
14. }
15.
16.
17. function switch_option(option) {
18.     sidebar_navigate(option);
19.
20.     if (option === 'dashboard') {
21.         build_dashboard();
22.     }
23. }
24.
25.
26. function data_access_call(action, params) {
27.     var xhttp = new XMLHttpRequest();
28.     var response;
29.     xhttp.onreadystatechange = function () {
30.         if (this.readyState === 4 && this.status === 200) {
31.             response = JSON.parse(xhttp.responseText);
```

```

32.     }
33. };
34.                                     xhttp.open("GET",
"http://thesis.in.cs.ucy.ac.cy/GamesPlatform/wp-content/plugins/ucy-analytics-platform/
data_access/" + action + '.php?' + new URLSearchParams(params).toString(), false);
35.  xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
36.  xhttp.send();
37.  return response;
38. }
39.
40. const asc = arr => arr.sort((a, b) => a - b);
41.
42. const sum = arr => arr.reduce((a, b) => a + b, 0);
43.
44. const mean = arr => sum(arr) / arr.length;
45.
46. const std = (arr) => {
47.   const mu = mean(arr);
48.   const diffArr = arr.map(a => (a - mu) ** 2);
49.   return Math.sqrt(sum(diffArr) / (arr.length - 1));
50. };
51.
52. const quantile = (arr, q) => {
53.   const sorted = asc(arr);
54.   const pos = (sorted.length - 1) * q;
55.   const base = Math.floor(pos);
56.   const rest = pos - base;
57.   if (sorted[base + 1] !== undefined) {
58.     return sorted[base] + rest * (sorted[base + 1] - sorted[base]);
59.   } else {
60.     return sorted[base];
61.   }
62. };
63.
64. const q75 = arr => quantile(arr, .75);
65.

```



```

66.
67. function build_dashboard(full_activity = false, full_performance = false) {
68.   var activity = data_access_call('activity', {});
69.
70.
71.   var dashboard_activity = document.getElementById('dashboard-activity');
72.   var dashboard_performance = document.getElementById('dashboard-performance');
73.   dashboard_activity.innerHTML = '<h2>Recent Activity</h2><br>';
74.   dashboard_performance.innerHTML = '<h2>Performance Highlights</h2><br>';
75.
76.   var activity_counter = 0;
77.   for (var a of activity) {
78.     dashboard_activity.innerHTML += `
79.       <p>${a.timestamp.slice(0, 5)}<span class="scarlet">${a.timestamp.slice(5,
100.         10)}</span>
101.           ${a.timestamp.slice(10, 19)}&emsp; User <span
102. class="scarlet">${a.userid}</span> has played
103.       <span class="scarlet">${a.game}</span>.</p>
104.     `;
105.     activity_counter++;
106.     if (activity_counter == 10 && !full_activity) {
107.       dashboard_activity.innerHTML += '<a id="activity-more"
108. onclick="build_dashboard(true, ' + full_performance + ')">See more...</a>';
109.       break;
110.     }
111.   }
112. }
113.
114.
115.   var joining_shapes_performance = data_access_call('joining_shapes_performance',
116. {});
117.   var related_memory_performance = data_access_call('related_memory_performance',
118. {});
119.   var js_q75 = q75([...joining_shapes_performance.y]);
120.   var rm_q75 = q75([...related_memory_performance.y]);
121.
122.   var performance_counter = 0;

```

```

97.
98.   for (i in joining_shapes_performance.x) {
99.       if (joining_shapes_performance.y[i] > js_q75) {
100.           if (performance_counter == 10 && !full_performance) {
101.               break;
102.           }
103.           performance_counter++;
104.           dashboard_performance.innerHTML += `
105.               <p>User <span class="scarlet">${joining_shapes_performance.x[i]}</span>
had more incorrect answers than
106.                   other users on the <span class="scarlet">Joining Shapes</span> game in the
past 30 days.</p>
107.               `;
108.
109.           }
110.       }
111.   for (i in related_memory_performance.x) {
112.       if (related_memory_performance.y[i] > rm_q75) {
113.           if (performance_counter == 10 && !full_performance) {
114.               dashboard_performance.innerHTML += '<a id="performance-more"
onclick="build_dashboard(' + full_activity + ', true)">See more...</a>';
115.               break;
116.           }
117.           performance_counter++;
118.           dashboard_performance.innerHTML += `
119.               <p>User <span class="scarlet">${related_memory_performance.x[i]}</span>
had more incorrect answers than
120.                   other users on the <span class="scarlet">Related Memory</span> game in the
past 30 days.</p>
121.               `;
122.           }
123.       }
124.   }
125.
126.
127. function load_levels(option) {

```

```

128.   var game = document.getElementById(option + '-' + 'game-select').value;
129.   var levels = data_access_call('levels', { 'game': game });
130.
131.   var level_select = document.getElementById(option + '-' + 'level-select');
132.   level_select.innerHTML = '<option disabled selected value></option>';
133.   for (var level of levels) {
134.       var option_label = level['dl'] + ' ' + level['item'];
135.       level_select.innerHTML += '<option value="' + option_label + ">' + option_label
+ '</option>';
136.
137.   }
138. }
139.
140.
141. function view_charts(option) {
142.
143.   var user_id = (option != 'user_progress') ? " : document.getElementById(option + '-' +
'user-id').value;
144.
145.   if (option == 'user_progress' && !user_id) {
146.       console.log('ERROR: No user ID.');
```

```

147.       return;
148.   }
149.
150.   var game = document.getElementById(option + '-' + 'game-select').value;
151.
152.   if (!game) {
153.       console.log('ERROR: Game not selected.');
```

```

154.       return;
155.   }
156.
157.   var level = document.getElementById(option + '-' + 'level-select').value;
158.
159.   if (!level) {
160.       console.log('ERROR: No level.');
```

```

161.       return;

```

```

162. }
163.
164. var dl = level.split(' ')[0];
165. var item = level.split(' ')[1];
166.
167. var data_access_args = {
168.     'dl': dl,
169.     'item': item,
170.     'user_id': user_id,
171.     'from_date': document.getElementById('from-date-' + option).value,
172.     'to_date': document.getElementById('to-date-' + option).value
173. };
174.
175. var charts_info = document.getElementById('charts-info')
176. charts_info.innerHTML = "<br><h2>User's last session results:</h2>";
177. if (game == 'joining_shapes') {
178.     if (option == 'user_progress') {
179.
180.         var user_mistakes_per_session =
data_access_call('charts/joining_shapes/user_mistakes_per_session', data_access_args);
181.         var user_timing_per_session =
data_access_call('charts/joining_shapes/user_timing_per_session', data_access_args);
182.         document.getElementById('compare_users_charts').innerHTML = "
183.             <div class=\"chart-outer-div\"><div class=\"chart-inner-div\"><canvas
class=\"chart\" id=\"chart1\"></canvas></div></div>
184.             <div class=\"chart-outer-div\"><div class=\"chart-inner-div\"><canvas
class=\"chart\" id=\"chart2\"></canvas></div></div>
185.         `;
186.         build_chart('chart1', 'User Incorrect Moves Per Session',
user_mistakes_per_session['x'], user_mistakes_per_session['y']);
187.         build_chart('chart2', 'User Timing Per Session', user_timing_per_session['x'],
user_timing_per_session['y']);
188.
189.
190.         var chart_percentages = get_chart_percentages(user_mistakes_per_session);

```

```

191.         charts_info.innerHTML += (chart_percentages == 'no_previous_sessions') ? "" :
'<p>The user\'s <span class="spring-green">accuracy</span> in last session was <span
class="scarlet">better</span></span class="light-blue">no worse</span> than <span
class="scarlet">' + chart_percentages.better + '</span></span class="light-blue">' +
chart_percentages.no_worse + '</span>% of his previous sessions.</p>';
192.         var chart_percentages = get_chart_percentages(user_timing_per_session);
193.         charts_info.innerHTML += (chart_percentages == 'no_previous_sessions') ? "" :
'<p>The user\'s <span class="spring-green">speed</span> in last session was <span
class="scarlet">better</span></span class="light-blue">no worse</span> than <span
class="scarlet">' + chart_percentages.better + '</span></span class="light-blue">' +
chart_percentages.no_worse + '</span>% of his previous sessions.</p>';
194.
195.     }
196.     else if (option == 'compare_users') {
197.         var users_mistakes_comparison =
data_access_call('charts/joining_shapes/users_mistakes_comparison', data_access_args);
198.         var users_timings_comparison =
data_access_call('charts/joining_shapes/users_timings_comparison', data_access_args);
199.         document.getElementById('user_progress_charts').innerHTML = "";
200.         document.getElementById(option + '_charts').innerHTML = `
201.             <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart1"></canvas></div></div>
202.             <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart2"></canvas></div></div>
203.         `;
204.         build_chart('chart1', "Average Users' Incorrect Moves Comparison",
users_mistakes_comparison['x'], users_mistakes_comparison['y']);
205.         build_chart('chart2', "Average Users' Timings Comparison",
users_timings_comparison['x'], users_timings_comparison['y']);
206.
207.     }
208. }
209. else if (game == 'related_memory') {
210.     if (option == 'user_progress') {
211.         var user_timing_per_session =
data_access_call('charts/related_memory/user_timing_per_session', data_access_args);

```

```

212.                                     var    user_drags_per_session    =
data_access_call('charts/related_memory/user_drags_per_session', data_access_args);
213.                                     var    user_drag_time_per_session    =
data_access_call('charts/related_memory/user_drag_time_per_session', data_access_args);
214.                                     var    user_correctness_per_session    =
data_access_call('charts/related_memory/user_correctness_per_session', data_access_args);
215.
216.     document.getElementById('compare_users_charts').innerHTML = "";
217.     document.getElementById(option + '_charts').innerHTML = `
218.         <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart4"></canvas></div></div>
219.         <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart1"></canvas></div></div>
220.         <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart2"></canvas></div></div>
221.         <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart3"></canvas></div></div>
222.     `;
223.         build_chart('chart4', "Users' Correctness Per Session",
user_correctness_per_session['x'], user_correctness_per_session['y'], true, false);
224.         build_chart('chart1', 'User Timing Per Session', user_timing_per_session['x'],
user_timing_per_session['y']);
225.         build_chart('chart2', "User Drags Per Session", user_drags_per_session['x'],
user_drags_per_session['y']);
226.         build_chart('chart3', "User Drag Timings Per Session",
user_drag_time_per_session['x'], user_drag_time_per_session['y']);
227.         var correctness = (!user_correctness_per_session.y.length) ? "" :
((user_correctness_per_session.y[user_correctness_per_session.y.length - 1] == 1) ?
'correct' : 'incorrect');
228.         charts_info.innerHTML += (!user_correctness_per_session.y.length) ? "" :
'<p>The user\'s answer was <span class="" + ((correctness == 'correct') ? 'spring-green' :
'scarlet') + ">' + correctness + '</span> in last session.</p>';
229.         var chart_percentages = get_chart_percentages(user_timing_per_session);
230.         charts_info.innerHTML += (chart_percentages == 'no_previous_sessions') ? "" :
'<p>The user\'s <span class="spring-green">speed</span> in last session was <span
class="scarlet">better</span></span class="light-blue">no worse</span> than <span

```

```

class="scarlet">' + chart_percentages.better + '</span><span class="light-blue">' +
chart_percentages.no_worse + '</span>% of his previous sessions.</p>';
231.         var chart_percentages = get_chart_percentages(user_drag_per_session);
232.         charts_info.innerHTML += (chart_percentages == 'no_previous_sessions') ? "" :
'<p>The user\'s <span class="spring-green">number of moves</span> in last session were
<span class="scarlet">less</span><span class="light-blue">no more</span> than <span
class="scarlet">' + chart_percentages.better + '</span><span class="light-blue">' +
chart_percentages.no_worse + '</span>% of his previous sessions.</p>';
233.         var chart_percentages = get_chart_percentages(user_drag_time_per_session);
234.         charts_info.innerHTML += (chart_percentages == 'no_previous_sessions') ? "" :
'<p>The user\'s <span class="spring-green">dragging speed</span> in last session was <span
class="scarlet">better</span><span class="light-blue">no worse</span> than <span
class="scarlet">' + chart_percentages.better + '</span><span class="light-blue">' +
chart_percentages.no_worse + '</span>% of his previous sessions.</p>';
235.
236.     }
237.     else if (option == 'compare_users') {
238.         var users_correctness_comparison =
data_access_call('charts/related_memory/users_correctness_comparison', data_access_args);
239.         var users_timings_comparison =
data_access_call('charts/related_memory/users_timings_comparison', data_access_args);
240.         var users_drag_comparison =
data_access_call('charts/related_memory/users_drag_comparison', data_access_args);
241.         var users_drag_timings_comparison =
data_access_call('charts/related_memory/users_drag_timings_comparison',
data_access_args);
242.
243.         document.getElementById('user_progress_charts').innerHTML = "";
244.         document.getElementById(option + '_charts').innerHTML = `
245.             <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart4"></canvas></div></div>
246.             <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart1"></canvas></div></div>
247.             <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart2"></canvas></div></div>

```

```

248.         <div class="chart-outer-div"><div class="chart-inner-div"><canvas
class="chart" id="chart3"></canvas></div></div>
249.         `;
250.
251.         build_chart('chart4', "Average Users' Accuracy Comparison (Percentage)",
users_correctness_comparison['x'], users_correctness_comparison['y'], false, true);
252.         build_chart('chart1', "Average Users' Timings Comparison (Seconds)",
users_timings_comparison['x'], users_timings_comparison['y']);
253.         build_chart('chart2', "Average Users' Drags Comparison",
users_drags_comparison['x'], users_drags_comparison['y']);
254.         build_chart('chart3', "Average Users' Drag Timings Comparison (Seconds)",
users_drag_timings_comparison['x'], users_drag_timings_comparison['y']);
255.     }
256. }
257. }
258.
259.
260. function get_chart_percentages(values) {
261.     var num_previous_sessions = values.y.length - 1;
262.     if (num_previous_sessions <= 0) {
263.         return 'no_previous_sessions';
264.     }
265.     var last_session_mistakes = parseInt(values.y[values.y.length - 1]);
266.     var num_worse_sessions = 0;
267.     var num_no_better_sessions = 0;
268.     for (var i in values.y.slice(0, values.y.length - 1)) {
269.         var session_mistakes = parseInt(values.y[i]);
270.         if (session_mistakes > last_session_mistakes) {
271.             num_worse_sessions++;
272.         }
273.         if (session_mistakes >= last_session_mistakes) {
274.             num_no_better_sessions++;
275.         }
276.     }
277.     var num_worse_sessions_percentage = num_worse_sessions / num_previous_sessions
* 100;

```



```

278.         var num_no_better_sessions_percentage = num_no_better_sessions /
num_previous_sessions * 100;
279.     return {
280.         'better': Math.round(num_worse_sessions_percentage),
281.         'no_worse': Math.round(num_no_better_sessions_percentage)
282.     }
283.
284. }
285.
286.
287. function build_chart(chart_id, label, labels, data, is_boolean = false, is_percentage =
false) {
288.     var config = {
289.         type: 'line',
290.         data: {
291.             labels: labels,
292.             datasets: [{
293.                 label: label,
294.                 backgroundColor: 'rgb(255, 99, 132)',
295.                 borderColor: 'rgb(255, 99, 132)',
296.                 data: data,
297.                 tension: 0.1
298.             }]
299.         },
300.         options: is_boolean ? {
301.             tooltips: {
302.                 callbacks: {
303.                     label: tooltipItem => tooltipItem.value < 0.5 ? 'false' : 'true'
304.                 }
305.             },
306.             scales: {
307.                 yAxes: {
308.                     ticks: {
309.                         callback: value => {
310.                             if (value == 0) {
311.                                 return 'Incorrect';

```

```

312.             } else {
313.                 return value == 1 ? 'Correct' : "";
314.             }
315.         }
316.     }
317. }
318. }
319. } : is_percentage ? {
320.     scales: {
321.         y: {
322.             min: 0
323.         }
324.     }
325.
326.     } : {}
327. };
328.
329. new Chart(chart_id, config);
330. }
331.
332. function view_table() {
333.     var loading_label = document.getElementById('loading-label');
334.     loading_label.removeAttribute('hidden');
335.     var game = document.getElementById('game_data-game-select').value;
336.     var data = data_access_call(game + '_table', {});
337.
338.     var table_container = document.getElementById('games-table-container');
339.     table_container.innerHTML = '<table id="games-table" class="table table-bordered
table-striped"> <thead id="table-head"></thead> <tbody id="table-body"></tbody></table>';
340.     var table_head = document.getElementById('table-head');
341.     var table_body = document.getElementById('table-body');
342.     if (game == 'joining_shapes') {
343.         table_head.innerHTML = '<tr><th>User ID</th><th>DL</th><th>Item
Set</th><th>User Time</th><th>Incorrect Moves</th><th>Date</th></tr>';
344.         for (var d of data) {

```

```

345.         table_body.innerHTML += '<tr><td>' + d.userid + '</td><td>' + d.DL +
'</td><td>' + d.itemSet + '</td><td>' + d.userTime + '</td><td>' +
d.numberofIncorrectUserMoves + '</td><td>' + d.timestamp + '</td></tr>';
346.     }
347.     var search = document.getElementById('games-table-search-container');
348.     search.innerHTML = `
349.         <div class="alert alert-primary"><i class="bi bi-info-circle-fill"></i> In order to
search for exact matches, use the "=" character before a keyword.</div>
350.         <div class="table-search-container table-search-container-leftmost"><input
id="search-users" type="text" class="form-control table-search" aria-label="Large"
placeholder="Search for users..." aria-describedby="inputGroup-sizing-sm"
onkeyup="table_search(0);"></div>
351.         <div class="table-search-container"><input id="search-dls" type="text"
class="form-control table-search" aria-label="Large" placeholder="Search for DLs..." aria-
describedby="inputGroup-sizing-sm" onkeyup="table_search(1);"></div>
352.         <div class="table-search-container"><input id="search-items" type="text"
class="form-control table-search" aria-label="Large" placeholder="Search for Items..." aria-
describedby="inputGroup-sizing-sm" onkeyup="table_search(2);"></div>
353.         <div class="table-search-container"><input id="search-timings" type="text"
class="form-control table-search" aria-label="Large" placeholder="Search for Timings..."
aria-describedby="inputGroup-sizing-sm" onkeyup="table_search(3);"></div>
354.         <div class="table-search-container"><input id="search-incorrect-moves"
type="text" class="form-control table-search" aria-label="Large" placeholder="Search for
Incorrect moves..." aria-describedby="inputGroup-sizing-sm"
onkeyup="table_search(4);"></div>
355.         <div class="table-search-container table-search-container-rightmost"><input
id="search-dates" type="text" class="form-control table-search" aria-label="Large"
placeholder="Search for Dates..." aria-describedby="inputGroup-sizing-sm"
onkeyup="table_search(5);"></div>
356.     `;
357. }
358. else if (game == 'related_memory') {
359.     table_head.innerHTML = '<tr><th>User ID</th><th>DL</th><th>Item
Set</th><th>Information</th><th>Information Value</th><th>Date</th></tr>';
360.     for (var d of data) {

```

```

361.         table_body.innerHTML += '<tr><td>' + d.userid + '</td><td>' + d.DL +
'</td><td>' + d.itemSet + '</td><td>' + d.l + '</td><td>' + d.v + '</td><td>' + d.timestamp +
'</td></tr>';
362.     }
363.     var search = document.getElementById('games-table-search-container');
364.     search.innerHTML = `
365.         <div class="alert alert-primary"><i class="bi bi-info-circle-fill"></i> In order to
search for exact matches, use the "=" character before a keyword.</div>
366.         <div class="table-search-container table-search-container-leftmost"><input
id="search-users" type="text" class="form-control table-search" aria-label="Large"
placeholder="Search for users..." aria-describedby="inputGroup-sizing-sm"
onkeyup="table_search(0);"></div>
367.         <div class="table-search-container"><input id="search-dls" type="text"
class="form-control table-search" aria-label="Large" placeholder="Search for DLs..." aria-
describedby="inputGroup-sizing-sm" onkeyup="table_search(1);"></div>
368.         <div class="table-search-container"><input id="search-items" type="text"
class="form-control table-search" aria-label="Large" placeholder="Search for Items..." aria-
describedby="inputGroup-sizing-sm" onkeyup="table_search(2);"></div>
369.         <div class="table-search-container"><input id="search-timings" type="text"
class="form-control table-search" aria-label="Large" placeholder="Search for Timings..."
aria-describedby="inputGroup-sizing-sm" onkeyup="table_search(3);"></div>
370.         <div class="table-search-container"><input id="search-incorrect-moves"
type="text" class="form-control table-search" aria-label="Large" placeholder="Search for
Incorrect moves..." aria-describedby="inputGroup-sizing-sm"
onkeyup="table_search(4);"></div>
371.         <div class="table-search-container table-search-container-rightmost"><input
id="search-dates" type="text" class="form-control table-search" aria-label="Large"
placeholder="Search for Dates..." aria-describedby="inputGroup-sizing-sm"
onkeyup="table_search(5);"></div>
372.     `;
373.
374. }
375. loading_label.setAttribute('hidden', 'true');
376.
377. }
378.

```

```

379. function table_search(field_index) {
380.     var input, filter, table, tr, td, i, txtValue;
381.     search_id = document.getElementsByClassName('table-search')[field_index].id;
382.     input = document.getElementById(search_id);
383.     filter = input.value.toUpperCase();
384.     table = document.getElementById("games-table");
385.     tr = table.getElementsByTagName("tr");
386.     for (i = 0; i < tr.length; i++) {
387.         td = tr[i].getElementsByTagName("td")[field_index];
388.         if (td) {
389.             txtValue = td.textContent || td.innerText;
390.             if (
391.                 (filter[0] != '=' && txtValue.toUpperCase().indexOf(filter) > -1)
392.                 || (filter[0] == '=' && filter == '=' + txtValue)
393.             ) {
394.                 tr[i].style.display = "";
395.             } else {
396.                 tr[i].style.display = "none";
397.             }
398.         }
399.     }
400. }

```

Appendix E

connection.php

```
1. <?php
2.
3. ini_set('display_errors', 1);
4. ini_set('display_startup_errors', 1);
5. error_reporting(E_ALL);
6.
7. $DB_NAME = "gamesPlatform";
8. $MYSQL_USERNAME = "gamesPlatform";
9. $MYSQL_PASSWORD = "D8SbUAcbBzuMUJxs";
10. $HOST_SERVER = "localhost";
11.
12. $connection = mysqli_connect($HOST_SERVER, $MYSQL_USERNAME,
    $MYSQL_PASSWORD, $DB_NAME) or die ('Error connecting to database.');
```

activity.php

```
1. <?php
2.
3. include 'connection.php';
4.
5. $activity = $connection->query("
6. SELECT
7.     userid,
8.     game,
9.     timestamp
10. FROM (
11.     SELECT
```

```

12.     userid,
13.     'Joining Shapes' AS game,
14.     timestamp
15. FROM
16.     joiningshapespro_analytics
17. UNION
18. SELECT
19.     userid,
20.     'Related Memory' AS game,
21.     timestamp
22. FROM
23.     memorypro_analytics
24. ) AS activity
25. WHERE
26.     timestamp >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
27. ORDER BY
28.     timestamp DESC"
29. )->fetch_all(MYSQLI_ASSOC);
30.
31. echo json_encode($activity);
32.
33. ?>

```

joining_shapes_performance.php

```

1. <?php
2.
3. include 'connection.php';
4.
5. $query = "
6.     SELECT
7.         userid,
8.         AVG(numberOfIncorrectUserMoves) AS average_incorrect_moves

```

```

9. FROM
10.     joiningshapespro_analytics
11. GROUP BY
12.     userid
13. ORDER BY
14.     userid
15. ";
16.
17. if (!$query_execution = $connection->query($query)) {
18.     echo "SQL Error: " . $connection->error;
19.     die();
20. };
21.
22. $data = $query_execution->fetch_all(MYSQLI_ASSOC);
23.
24. $final = array('x' => array(), 'y' => array());
25.
26. foreach ($data as $record) {
27.     array_push($final['x'], $record['userid']);
28.     array_push($final['y'], $record['average_incorrect_moves']);
29. }
30.
31. echo json_encode($final);
32.
33. ?>

```

related_memory_performance.php

```

1. <?php
2.
3. include 'connection.php';
4.
5. $query = "

```



```

6.  (
7.      SELECT
8.          userid,
9.          0 AS num_mistakes
10.     FROM
11.         memorypro_analytics
12. )
13. UNION
14. (
15.     SELECT
16.         userid,
17.         COUNT(*) AS num_mistakes
18.     FROM
19.         memorypro_analytics
20.     WHERE
21.         action = 'Wrong'
22.         AND timestamp >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
23.     GROUP BY
24.         userid
25.     ORDER BY
26.         userid
27. )
28. ";
29.
30. if (!$query_execution = $connection->query($query))) {
31.     echo "SQL Error: " . $connection->error;
32.     die();
33. };
34.
35. $data = $query_execution->fetch_all(MYSQLI_ASSOC);
36.
37. $final = array('x' => array(), 'y' => array());
38.
39. foreach ($data as $record) {
40.     array_push($final['x'], $record['userid']);
41.     array_push($final['y'], $record['num_mistakes']);

```

```

42. }
43.
44. echo json_encode($final);
45.
46. ?>

```

user_mistakes_per_session.php

```

1. <?php
2.
3. include '../connection.php';
4.
5. $dl = $_GET["dl"];
6. $item = $_GET["item"];
7. $user_id = $_GET["user_id"];
8. $from_date = $_GET["from_date"];
9. $to_date = $_GET["to_date"];
10.
11. $query = "
12.     SELECT
13.         timestamp,
14.         numberOfIncorrectUserMoves
15.     FROM
16.         joiningshapespro_analytics
17.     WHERE
18.         DL = " . $dl . " '
19.         AND itemSet = " . $item . "
20.         AND userid = " . $user_id .
21.         ((($from_date) ? (" AND timestamp >= " . $from_date) : "")) .
22.         ((($to_date) ? (" AND timestamp <= " . $to_date . " 23:59:59") : "")) . "
23.     ORDER BY timestamp
24. ";
25.
26. if (!($query_execution = $connection->query($query))) {

```

```

27. echo "SQL Error: " . $connection->error;
28. die();
29. };
30.
31. $data = $query_execution->fetch_all(MYSQLI_ASSOC);
32.
33. $final = array('x' => array(), 'y' => array());
34.
35. foreach ($data as $record) {
36.     array_push($final['x'], str_replace(' ', ' ', substr($record['timestamp'], 0, -3)));
37.     array_push($final['y'], $record['numberOfIncorrectUserMoves']);
38. }
39.
40. echo json_encode($final);
41.
42. ?>

```

users_drag_timings_comparison.php

```

1. <?php
2.
3. include '../connection.php';
4.
5. $dl = $_GET["dl"];
6. $item = $_GET["item"];
7. $user_id = $_GET["user_id"];
8. $from_date = $_GET["from_date"];
9. $to_date = $_GET["to_date"];
10.
11. $query = "
12.     SELECT
13.         timestamp,

```

```

14.          SUM(CAST(SUBSTR(action, 11, LENGTH(action)) AS DOUBLE)) AS
sum_drag_time
15.  FROM
16.    memorypro_analytics
17.  WHERE
18.    action LIKE 'Drag_Time%'
19.    AND DL = "" . $dl . ""
20.    AND itemSet = "" . $item . ""
21.    AND userid = "" . $user_id .
22.    (($from_date) ? (" AND timestamp >= "" . $from_date) : "") .
23.    (($to_date) ? (" AND timestamp <= "" . $to_date . " 23:59:59") : "") . ""
24.  GROUP BY
25.    userid,
26.    usersessionid,
27.    usergamesessionid
28.  ORDER BY
29.    timestamp
30. ";
31.
32. if (!$query_execution = $connection->query($query)) {
33.   echo "SQL Error: " . $connection->error;
34.   die();
35. };
36.
37. $data = $query_execution->fetch_all(MYSQLI_ASSOC);
38.
39. $final = array('x' => array(), 'y' => array());
40.
41. foreach ($data as $record) {
42.   array_push($final['x'], str_replace(' ', ' ', substr($record['timestamp'], 0, -3)));
43.   array_push($final['y'], $record['sum_drag_time']);
44. }
45.
46. echo json_encode($final);
47.
48. ?>

```

related_memory_table.php

```
1. <?php
2.
3. include 'connection.php';
4.
5. $query = "
6.     SELECT
7.         userid,
8.         DL,
9.         itemSet,
10.        IF(action IN ('Correct', 'Wrong'), 'answer', 'time') AS l,
11.        IF(action IN ('Correct', 'Wrong'), action, CAST(SUBSTR(action, 11,
LENGTH(action)) AS DOUBLE)) AS v,
12.        timestamp
13.    FROM
14.        memorypro_analytics
15.    WHERE
16.        action IN ('Correct', 'Wrong')
17.        OR action LIKE 'Drag_Time%'
18.    ORDER BY
19.        timestamp DESC
20. ";
21.
22. if (!$query_execution = $connection->query($query)) {
23.     echo "SQL Error: " . $connection->error;
24.     die();
25. };
26.
27. $data = $query_execution->fetch_all(MYSQLI_ASSOC);
28.
29. echo json_encode($data);
```

30.

31. ?>