CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

The main aim of this web page is to develop a game of memory. Match each square by color. we can click the squares to flip them. Score multipliers are applied if one or both of your matching squares were viewed for the first time. In this project, we can check the player have a perfect memory, the memory game can be seen as a game of strategy. In this application, the player has to match each square by color. Click the squares to flip them. Score multipliers are applied if one or both of your matching squares were viewed for the first time. In this project we can check whether the player have a perfect memory, the game can be seen as a game of strategy. In this application the player has to match each square by color. Click the squares to flip them. Score multipliers are applied if one or both of your matching squares were viewed for the first time. By playing this game one can come to know about the capacity to memorize. This helps him out to improve his memory by playing a simple game. It improves his memory capacity.

Time management played a large role because in order to create a software application a specific timeline needed to be laid out in order to get certain aspects of the project done on time so that future work could be built upon its foundation. Time management is an important skill to learn and refine because it is applied in all kinds of jobs and different situations. Also team work and responsibility are traits which are desired when working in the real world because, in most applications, one individual is not responsible for a project, but a group of individuals is.

By learning how to work in a team atmosphere and having the responsibility to get the designated work done, you can apply these traits in a favorable way to future opportunities. The design of this software game helped to greatly improve the programming skills of the team. Being given the opportunity to become familiar with Windows SDK is an excellent opportunity to add to the team's computer knowledge, which is very useful in a professional engineering environment. When researching the way in which this design software package would be implemented, a lot of research needed to be performed and an assessment had to be made about the best way in which to approach the project.

These in itself provided valuable experience to the team because being able to research and learn how a system works, and then determining the best way to go about completing the task, is a very practical skill that can be used throughout life. Overall, the skill sets which were learned and applied during the design process of this project provided valuable experience to the team. By refining such traits as leadership, teamwork, and time management the team was able to work well and stay on task in order to complete the design phase for the project. Also a practical application of learning how a system works and applying it to solve a problem was learned. These skills provide valuable experience for implementing what was learned in the classroom into everyday life, and help provide a base which can be built upon for future careers.

Angular is a platform and framework for building client applications in HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps. The basic building blocks of an Angular application are NgModules, which provide a compilation context for components. NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules. An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.

Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data. Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient. Both components and services are simply classes, with decorators that mark their type and provide metadata that tells Angular how to use them.

The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display. The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).

An app's components typically define many views, arranged hierarchically. Angular provides the Router service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.

1.2 SCOPE

The main scope of this game is the guessing game which is used by color tiles.We can Match each square by color. we can click the squares to flip them. Score multipliers are applied if one or both of your matching squares were viewed for the first time.

1.3 PROJECT FEATURES

- **User login**: This allows only the registered users to login.
- **Board**: User after logged in, can choose the size of the game like 4x4 or 6x6.
- **Start**: After choosing the game automatically get started.
- **Profile:** Registered user’s details will be shown here.
- **Accomplishments:** Top scores will be shown here with the time utilized to finish.

CHAPTER 2

LITRATURE SURVEY

Title: "MindTactics"

Author: Kenneth oum, Hasan Ayaz

Year: 2010

Description: During the last couple of decades, there has been an exponential improvement in neuroimaging technologies that allowed researchers noninvasive assessment of cognitive workload, short term memory, and spatial/navigational behaviors in humans. Through the use of new experimental paradigms and brain imaging devices, researchers have gained deeper insight into the neural correlates of emotion, cognition and motor control. Brain Computer Interface (BCI) systems utilize neuroimaging tools to detect brain activation evoked by a specific thinking process and convert it to a command. We have developed a game environment called "MindTactics" as a test platform for BCI devices to conceptualize experimental cognitive paradigms in ecologically valid environments as well as test BCI gameplay paradigms. Unlike traditional video games where the challenge to the user is the designed game mechanics, BCI based gameplay also involves mastering the use of the BCI device itself. The purpose of this project is to develop compelling BCI game design methods. MindTactics is capable of integrating data from multiple devices including the optical brain imaging based BCI developed at Drexel University, and it records behavioral log files for further analysis.

Brain-Computer Interfaces (BCI) systems have matured significantly from their earlier days and even commercial products are now available or under development for entertainment purposes. These new systems provide a novel control mechanism for video games. By utilizing the capability of these BCI devices, video games can be uniquely designed to be challenging and adaptive to the user in ways that current video games are not. However, BCI also presents inherent challenges so that a new gameplay paradigm is required by the devices due to the uncertainty in the functioning of both the devices and the brain signals that they measure. In traditional video games, the challenge to the user is simply the designed game mechanics. BCI based gameplay does not only involve the game mechanics based challenges, but also mastering the use of the BCI device itself. The purpose of this project is to develop compelling game design methods that will also serve as a platform for the testing of BCI devices. This

includes creating a 3D game environment that presents challenges such as mental distractions and direct multi-user competition. Functional near infrared spectroscopy, or fNIR, is a noninvasive optical technique that measures neural activity related hemodynamic response within the cortex. fNIR technology uses light within 700nm to 900nm, introduced at the scalp, to enable the noninvasive and safe measurement of changes in the relative ratios of deoxygenated hemoglobin (deoxy-Hb) and oxygenated hemoglobin (oxyHb) in the capillary beds during brain activity. Both, oxy-Hb and deoxy-Hb are correlates of brain activity through oxygen consumption by neurons. fNIR technology allows the design of portable, safe, affordable, noninvasive, and minimally intrusive monitoring systems. These qualities make fNIR suitable for the study of hemodynamic changes due to cognitive and emotional brain activity under ecologically valid, natural conditions.

Title: "Evaluate children learning experience of multitouch flash memory game"

Author: Lau Siong Hoe

Year: 2014

Description: Flash memory game based on multitouch technology is designed with richness of artistic effects which enhances children learning experience. It offers an opportunity for children to learn more effectively in a fun and enjoyable environment which improve their thinking, memorization and social skills. The objective of this paper is to provide an evaluation of children learning experience of multitouch flash memory game. An active learning experience offered by this game may contribute to the right brain functions development, thus unlocked children's innate intelligence. This preliminary study was conducted with four preschool children. Our results shown that the children had an average learning experience with the game. They had exhibited strong interest at playing the game but the excitement wore off gradually over time. This study had shown that the children exposure to such game in their education setting remains a challenge today.

According to research of Glenn Doman et al. from the Institutes for the Achievement of Human Potential, he had proved that children who did not have predestined inborn genius capabilities, were still able to build these abilities by presenting clear, accurately rendered pictures and information to them at the fastest possible speed. In addition, young children could be taught of anything and everything if they were provided the right environment and techniques. Dr. Makoto Shichida described the child's innate intelligence could be unlocked by placing more emphasis on the child's right brain function development. He had indicated that extraordinary learning abilities could be developed in ordinary children by quickly displaying flash cards with one object and word at a time. Instructors at Shichida's Asian Learning academies had documented abilities such as rapid calculation, mass sequential memory and photographic memory which emerged in many young children through the Shicida method. This was due to the right brain's mass-memory, automatic processing capability which was at work instantly absorbed and recorded the images and sounds presented. On the other hand, the left brain dealed with systematic, logical and orderly thought functions. Once these two types of brain functions had increased, accelerated learning could take place. Our research is based on multitouch technology with flash memory game which aims to improve preschool children's short term memory and speed recognition skills. Moreover, flash game stimulates part of their

brain to be more responsive and attentive. Subsequently, if the children were to play this flash memory game consistently, their concentration energy could be boosted.

In conclusion, multitouch edutainment is the act of learning through the utilization of multitouch technology, game design and education. The development of edutainment environment is also intended to implement technological innovations in education. Interactive multitouch flash memory game improved children's learning experiences with technology, and therefore provide an engaging environment for learning and development. It offers the opportunity for children to learn more effectively in a fun and enjoyable way. The multitouch flash memory game would improve children's brain function thus enhance their concentration, memory, reading and reasoning skills.

Title: "Visuospatial working memory game and measured memory performances at various ages"

Author: Takahiro Miura

Year: 2016

Description: Rapid population aging necessitates the urgent development of adjustable systems and interfaces that take into account the physical and cognitive functions of individuals with either mild or severe impairments. Various multi-functional systems have evolved, and developing and refining these systems in response to user requirements has become increasingly challenging. In particular, some elderly people resist current interfaces due to a decrease in interest in novelty as well as a decrease in cognitive function. To resolve these issues, it is necessary to characterize an interface using a ludic design that can induce hedonic effects in the users and decrease the cognitive workload of the interface. Thus, design implications and evaluation criteria for such senior-friendly and disability-friendly interfaces need to be established.

Because of rapid population aging, it is necessary to design and develop senior-friendly or disability-friendly interfaces that can decrease the cognitive workload caused by an interface. At that time, the design implications and evaluation criteria of an interface should be needed for creating senior-friendly and disability-friendly interfaces. One of the elements that relate to memory functions for manipulating interfaces include working memory. However, rough standard of visuospatial memory remains unknown. Particularly, there are little reports about the relations between the age and the volume of visuospatial memory. In this paper, we aim to clarify this relations for proposing interface implications by using a visual pattern span test. For effective measurement of the memory, we implemented a gamified measurement application based on visual pattern span tests. The results indicated that the median numbers of memorable buttons on visuospatial memory were and 7.0. Also, the number of memorable buttons increases as the age increases until the age ranged 21-25 and then decreases gradually as the age increases after 21-25 years. Our evaluation suggest that it can be effective to measure memory performance of, especially, children by the applications that are designed based on the entertainment concepts including a gamification.

However, rough standard of visuospatial memory remains unknown. Particularly, there are few reports on the relationship between visuospatial WM volume and age. In this paper, we aim to

clarify the relationship between the visuospatial WM volume and age for guiding interface design using a visual pattern span test. We administered a gamified measurement application based on existing visual pattern span tests to over 300 participants. We employed this approach because Komarov et al. reported that there are no significant differences and similar effect sizes in evaluating user interface performance between crowdsourced-based and laboratory-based experimental methodologies. This finding suggests that similar results can be obtained from small-scale laboratory experiments and large-scale field-based experiments.

CHAPTER 3

SYSTEM ANALYSIS

3.1 INTRODUCTION

The goal of the application is to improve the user's ability to recall common names based on visual prompts, typically a photograph. The main consideration of design is to make the program as accessible as possible to users of the User role type. This is due to the anticipated variability in users' prerequisite knowledge of computers and ability to interact with user interfaces. The design that was chosen to be implemented utilizes Microsoft Window's Software Development Kit (SDK).

3.2 DISADVANTAGES OF EXISTING SYSTEM

- In the existing system, the interface of the applications is not user-friendly and responsive. The user interface is much complex when compared to UX and UI.
- Making integration with the cloud makes the server side complex when compared to having a local server.
- In the existing system it is much harder to set up integration with API like Cloud API, Google API but with firebase and angular connecting to external API makes it much simpler.
- In existing system are time-consuming and it is much harder to control multiple data information coming from the database.
- It has a more complex code and difficult to understand by the developer.
- In the Existing system it's very difficult to create Mobile Application like android, IOS , blackberry or any.

In the Existing system, it's very difficult to create Desktop Applications like Windows 32- Bit /64- Bit, IOS or any.

3.3 PROPOSED SYSTEM

- In this system firstly user who wants to view the website with description of that image has to register and login to the webpage. User can also done this process by using Google sign-in & Sign-out options also by using Google email id to login.All the data

what we have used for logged in is stored in cloud database No-SQL storage on firestore. This application is protected by Auth-Guard Feature to let the user use services only if they login. Once they are logged in then only user can see or view the website. Only the authenticated user can view the web site. In this application, we have decreased the loading time of the user interface and made user interface much simpler.

- The developer can code complex applications with much easier code readability compared to existing web frameworks.
- Using angular application, we can reduce the files sizes by 18% - 45% compared to Java, PHP, and Ajax.
- Angular conveniently relieves the developer of actively manipulating DOM (Data Object Model).
- Angular simplifies linking data and user interface.
- It reduces development time code consistency.
- This application which we have built it is present and future ready with large developer customer support.
- It has cloud Integration, which makes application server less and makes the application to render fast.
- Database integration with cloud and it includes offline support and makes it ready for the desktop, mobile, and web ready.
- Using Google Authentication & Authorization The user can Login with Google Login/Signup, Guest Login/Signup, Direct Login/Signup.
- It can also integrate with Facebook, GitHub, and other online services easily.
- In this application, we can create Android Application, Desktop Application, IOS Applications, Windows Applications by adding Interfaces, API's, in a much-simplified pattern.

CHAPTER 4

SYSTEM REQUIRMENTS AND ANALYSIS

4.1 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

Technologies Used - Libraries of Npm & amp, Angular Cli, Yarn, AuthGuard, Angular material, rxjs, HttpClientModule, External API's, Chocolatey, Cordova, Cordova-android, ElectronJs.

Language – Html, Css, AngularJs, Jquery, Libraries of Npm & amp, Ng , MaterialJs

Software – NPM, Angular CLI , Firebase, GitHub, Android Studio ,Java 1.8.0+ SDK .

Tool –Visual Studio Code

Database – Firestore using Firebase Cloud

4.2 HARDWARE COMPONENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the system do and not how it should be implemented.

Operating System – Windows

Processor – 2 GHZ or Higher

Ram – 4 GB

Hard Disk – 20 GB

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

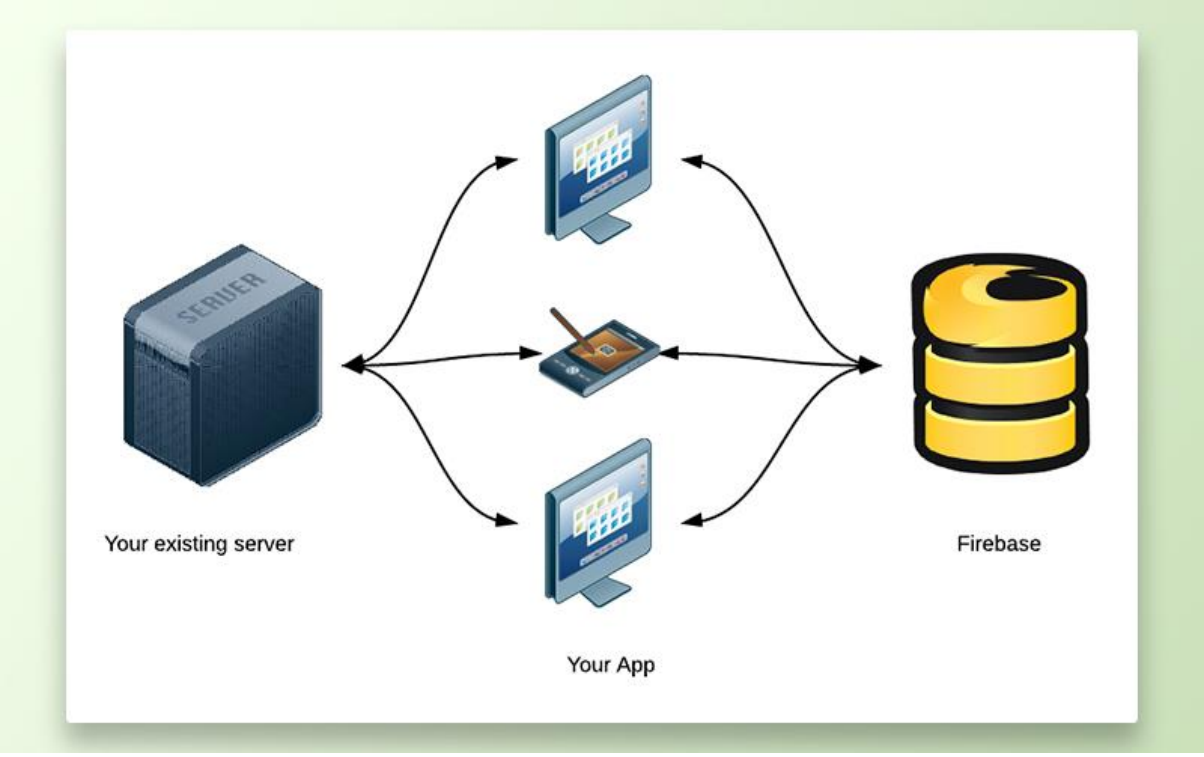

Fig. 5.1 System Architecture

Firebase Cloud Messaging

Formerly known as Google Cloud Messaging (GCM), Firebase Cloud Messaging (FCM) is a cross-platform solution for messages and notifications for Android, iOS, and web applications, which as of 2016 can be used at no cost.

Firebase Auth

Firebase Auth is a service that can authenticate users using only client-side code. It supports social login providers Facebook, GitHub, Twitter and Google (and Google Play Games). Additionally, it includes a user management system whereby developers can enable user authentication with email and password login stored with Firebase.

Realtime database

Firebase provides a real time database and backend as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase's cloud. The company provides client libraries that enable integration with Android, iOS, JavaScript, Java, Objective-C, Swift and Node.jsapplications.

The database is also accessible through a REST API and bindings for several JavaScript frameworks such as AngularJS, React, Ember.js and Backbone.js. The REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Developers using the real time database can secure their data by using the company's server-side-enforced security rules. Cloud Firestore which is Firebase's next generation of the Realtime Database was released for beta use.

Firebase Storage

Firebase Storage provides secure file uploads and downloads for Firebase apps, regardless of network quality. The developer can use it to store images, audio, video, or other user-generated content. Firebase Storage is backed by Google Cloud Storage.

Firebase Hosting

Firebase Hosting is a static and dynamic web hosting service that launched on May 13, 2014. It supports hosting static files such as CSS, HTML, JavaScript and other files, as well as support through Cloud Functions. The service delivers files over a content delivery network (CDN) through HTTP Secure (HTTPS) and Secure Sockets Layer encryption (SSL). Firebase partners with Fastly, a CDN, to provide the CDN backing Firebase Hosting. The company states that Firebase Hosting grew out of customer requests; developers were using Firebase for its real-time database but needed a place to host their content.

5.2 DATA FLOW DIAGRAM

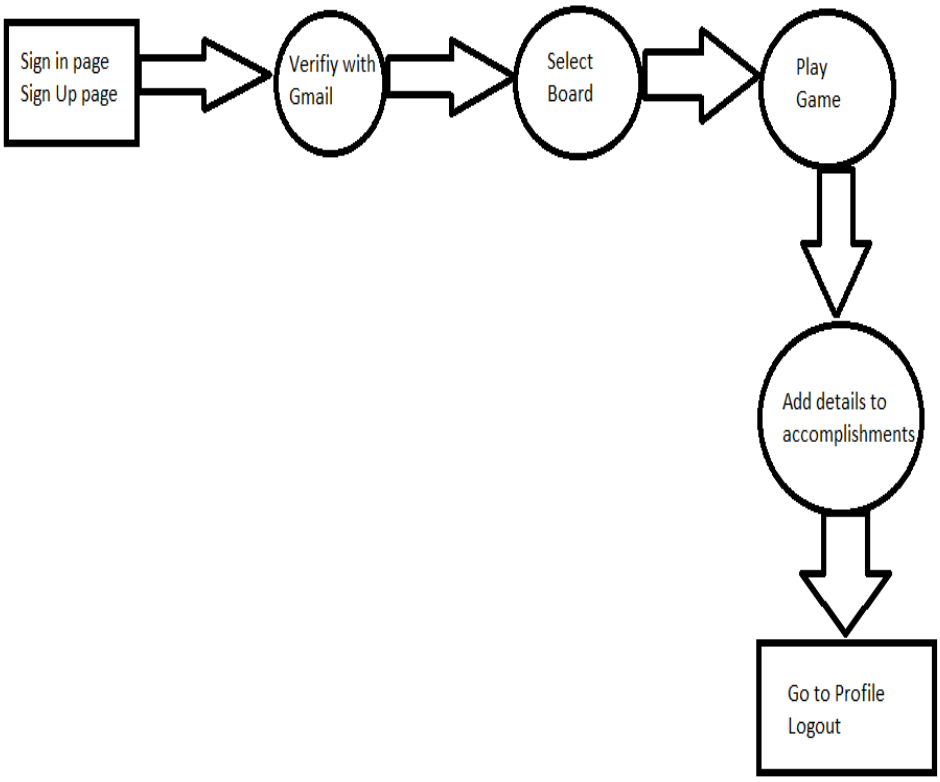

Fig. 5.2 Data Flow Diagram

DFD consists of processes, flows, warehouses, and terminators. There are several ways to view these DFD components.

Process

The process (function, transformation) is part of a system that transforms inputs to outputs. The symbol of a process is a circle, an oval, a rectangle or a rectangle with rounded corners (according to the type of notation). The process is named in one word, a short sentence, or a phrase that is clearly to express its essence.

Data Flow

Data flow (flow, dataflow) shows the transfer of information (sometimes also material) from one part of the system to another. The symbol of the flow is the arrow. The flow should have

a name that determines what information (or what material) is being moved. Exceptions are flows where it is clear what information is transferred through the entities that are linked to these flows. Material shifts are modelled in systems that are not merely informative. Flow should only transmit one type of information (material). The arrow shows the flow direction (it can also be bi-directional if the information to/from the entity is logically dependent - e.g. question and answer). Flows link processes, warehouses and terminators.

Warehouse

The warehouse (datastore, data store, file, database) is used to store data for later use. The symbol of the store is two horizontal lines, the other way of view is shown in the DFD Notation. The name of the warehouse is a plural noun (e.g. orders) - it derives from the input and output streams of the warehouse. The warehouse does not have to be just a data file, for example, a folder with documents, a filing cabinet, and optical discs. Therefore, viewing the warehouse in DFD is independent of implementation. The flow from the warehouse usually represents the reading of the data stored in the warehouse, and the flow to the warehouse usually expresses data entry or updating (sometimes also deleting data). Warehouse is represented by two parallel lines between which the memory name is located (it can be modelled as a UML buffer node).

Terminator

The Terminator is an external entity that communicates with the system and stands outside of the system. It can be, for example, various organizations (eg a bank), groups of people (e.g. customers), authorities (e.g. a tax office) or a department (e.g. a human-resources department) of the same organization, which does not belong to the model system. The terminator may be another system with which the modelled system communicates.

5.3 MODULES AND THEIR DISCRIPTION

Angular NgModules differ from and complement JavaScript (ES2015) modules. An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.

Every Angular app has a root module, conventionally named `AppModule`, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.

Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the `Router` NgModule.

Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of lazy-loading—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

For a more detailed discussion, see [Introduction to modules](guide/architecture-modules).

Every Angular application has at least one component, the root component that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.

The Component() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Decorators are functions that modify JavaScript classes. Angular defines a number of decorators that attach specific kinds of metadata to classes, so that the system knows what those classes mean and how they should work.

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic, and binding markup connects your application data and the DOM.

There are two types of data binding. Event binding lets your app respond to user input in the target environment by updating your application data. Property binding lets you interpolate values that are computed from your application data into the HTML.

Before a view is displayed, Angular evaluates the directives and resolves the binding syntax in the template to modify the HTML elements and the DOM, according to your program data and logic. Angular supports two-way data binding, meaning that changes in the DOM, such as user choices, are also reflected in your program data.

Your templates can use pipes to improve the user experience by transforming values for display.

For example, use pipes to display dates and currency values that are appropriate for a user's locale. Angular provides predefined pipes for common transformations, and you can also define your own pipes.

The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app. It is modeled on the familiar browser navigation conventions. Enter a URL in the address bar and the browser navigates to a corresponding page. Click links on the page and the browser navigates to a new page.

Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

The router maps URL-like paths to views instead of pages. When a user performs an action, such as clicking a link, that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies.

If the router determines that the current application state requires particular functionality, and the module that defines it hasn't been loaded, the router can lazy-load the module on demand.

The router interprets a link URL according to your app's view navigation rules and data state. You can navigate to new views when the user clicks a button or selects from a drop box, or in response to some other stimulus from any source. The router logs activity in the browser's history, so the back and forward buttons work as well.

To define navigation rules, you associate navigation paths with your components. A path uses a URL-like syntax that integrates your program data, in much the same way that template syntax integrates your views with your program data. You can then apply program logic to choose which views to show or to hide, in response to user input and your own access rules.

For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class. A service class definition is immediately preceded by the Injectable() decorator. The decorator provides the metadata that allows other providers to be injected as dependencies into your class.

Dependency injection (DI) lets you keep your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

A MODULE IN NODE.JS

Consider modules to be the same as JavaScript libraries.

A set of functions are to be included in application.

INCLUDE MODULES

To include a module, require() function with the name of the module is used

```
var http = require('http');
```

Now application has access to the HTTP module, and is able to create a server

```
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

CREATE OWN MODULES

Own modules can be created, and can be easily included in applications.

The following example creates a module that returns a date and time object:

Example

```
exports.myDateTime = function()

{

Return Date();

};
```

INCLUDE OWN MODULE

Example:

```
var http = require('http');

var dt = require('./myfirstmodule');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});

  res.write("The date and time is currently: " + dt.myDateTime());

  res.end();

}).listen(8080);
```

Node.js HTTP Module

THE BUILT-IN HTTP MODULE

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

To include the HTTP module, require() method can be used

```
var http = require('http');
```

Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

createServer() method can be used to create an HTTP server:

Example:

```
var http = require('http');

//create a server object:

http.createServer(function (req, res) {

  res.write('Hello World!'); //write a response to the client

  res.end(); //end the response

}).listen(8080); //the server object listens on port 8080
```

Add an HTTP Header

Response from the HTTP server is supposed to be displayed as HTML, then include an HTTP header with the correct content type:

Example:

```
var http = require('http');

http.createServer(function (req, res) {

  // add a HTTP header:

  res.writeHead(200, {'Content-Type': 'text/html'});

  res.write('Hello World!');

  res.end();

}).listen(8080);
```

The first argument of the res.writeHead() method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

READ THE QUERY STRING

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).This object has a property called "url" which holds the part of the url that comes after the domain name demo_http_url.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

Split the Query String

There are built-in modules to easily split the query string into readable parts, such as the URL module.

Example

Split the query string into readable parts:

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

NODE.JS FILE SYSTEM MODULE

NODE.JS AS A FILE SERVER

The Node.js file system module allows to work with the file system on computer.

To include the File System module, use the require() method:

```
var fs = require('fs');
```

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

READ FILES

The fs.readFile() method is used to read files on your computer.

Assume the following HTML file (located in the same folder as Node.js):

demofile1.html

```
<html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

Example:

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  //Open a file on the server and return its content:
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHe
ad(200, {'Content-Type': 'text/html'});
```

```
    res.write(data);

    return res.end();

  });

}).listen(8080);
```

CREATE FILES

The File System module has methods for creating new files:

- fs.appendFile()
- fs.open()
- fs.writeFile()

Writing File

Use fs.writeFile() method to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

Signature

fs.writeFile(filename, data[, options], callback)

ParameterDescription

filename: Full path and name of the file as a string.

Data: The content to be written in a file.

options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".

callback: A function with two parameters err and fd. This will get called when write operation completes.

The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created

Example

```
var fs = require('fs');

//create a file named mynewfile1.txt:
```

```
fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

UPDATE FILES

The File System module has methods for updating files:

- fs.appendFile()
- fs.writeFile()

The fs.appendFile() method appends the specified content at the end of the specified file

Example

```
var fs = require('fs');

//append content at the end of the file:

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {
  if (err) throw err;
  console.log('Updated!');
});
```

DELETE FILES

To delete a file with the File System module, use the fs.unlink() method.

The fs.unlink() method deletes the specified file

Example

```
var fs = require('fs');

//Delete the file mynewfile2.txt:

fs.unlink('mynewfile2.txt', function (err) {
```

```
  if (err) throw err;

  console.log('File deleted!');

});
```

RENAME FILES

To rename a file with the File System module, use the fs.rename() method.

The fs.rename() method renames the specified file

Example

```
var fs = require('fs');

//Rename the file "mynewfile1.txt" into "myrenamedfile.txt":

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {

  if (err) throw err;

  console.log('File Renamed!');

});
```

ANGULARJS

ANGULARJS INTRODUCTION

AngularJS is a **JavaScript framework** and can be added to an HTML page with a <script> tag.

AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

ANGULARJS EXTENDS HTML

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

ANGULARJS MODULES

AngularJS module defines an application.

The module is a container for the different parts of an application.

The module is a container for the application controllers.

Controllers always belong to a module.

CREATING A MODULE

A module is created by using the AngularJS function angular.module

```
<div ng-app="myApp">...</div>

<script>

var app = angular.module("myApp", []);

</script>
```

The "myApp" parameter refers to an HTML element in which the application will run.

Now controllers, directives, filters, and more, can be added to AngularJS application.

ADDING A CONTROLLER

Add a controller to application, and refer to the controller with the ng-controller directive:

ADDING A DIRECTIVE

AngularJS has a set of built-in directives which you can use to add functionality to application.

DATA BINDING

The {{ firstName }} expression, in the example above, is an AngularJS data binding expression.

Data binding in AngularJS binds AngularJS expressions with AngularJS data.

{{ firstName }} is bound with ng-model="firstName".

REPEATING HTML ELEMENTS

The ng-repeat directive repeats an HTML element

The ng-repeat directive actually **clones HTML elements** once for each item in a collection.

The ng-repeat directive used on an array of objects

THE NG-APP DIRECTIVE

The ng-app directive defines the **root element** of an AngularJS application.

The ng-app directive will **auto-bootstrap** (automatically initialize) the application when a web page is loaded.

THE NG-INIT DIRECTIVE

The ng-init directive defines **initial values** for an AngularJS application.

Normally, ng-init is not used. A controller or module is used.

THE NG-MODEL DIRECTIVE

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

The ng-model directive can also:

- Provide type validation for application data (number, email, required).
- Provide status for application data (invalid, dirty, touched, error).
- Provide CSS classes for HTML elements.
- Bind HTML elements to HTML forms.

CREATE NEW DIRECTIVES

In addition to all the built-in AngularJS directives, you can create your own directives.

New directives are created by using the .d function.

To invoke the new directive, make an HTML element with the same tag name as the new directive.

When naming a directive, use a camel case name, w3TestDirective, but when invoking it, use - separated name, w3-test-directive:

Invoke a directive by using:

- Element name
- Attribute
- Class
- Comment

RESTRICTIONS

Restrict directives to only invoke by some of the methods.

THE NG-MODEL DIRECTIVE

Use the ng-model directive to bind data from the model to the view on HTML controls (input, select, text area)

TWO-WAY BINDING

Data binding in AngularJS is the synchronization between the model and the view.

When data in the model changes, the *view* reflects the change, and when data in the *view* changes, the *model* is updated as well. This happens immediately and automatically, which makes sure that the model and the view is updated at all times.

ANGULARJS CONTROLLER

Applications in AngularJS are controlled by controllers. Because of the immediate synchronization of the model and the view, the controller can be completely separated from the view, and simply concentrate on the model data. Thanks to the data binding in AngularJS, the view will reflect any changes made in the controller.

ANGULARJS CONTROLLERS

AngularJS applications are controlled by controllers.

The **ng-controller** directive defines the application controller.

A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

CONTROLLER METHODS

The example above demonstrated a controller object with two properties: lastName and firstName.

A controller can also have methods (variables as functions):

CONTROLLERS IN EXTERNAL FILES

In larger applications, it is common to store controllers in external files.

5.4 UML DIAGRAMS:

UML diagrams, in this case, are used to communicate different aspects and characteristics of a system. However, this is only a top-level view of the system and will most probably not include all the necessary details to execute the project until the very end.

- **Forward Design** – The design of the sketch is done before coding the application. This is done to get a better view of the system or workflow that you are trying to create. Many design issues or flaws can be revealed, thus improving the overall project health and well-being.
- **Backward Design** – After writing the code, the UML diagrams are drawn as a form of documentation for the different activities, roles, actors, and workflows.

Blueprint

In such a case, the UML diagram serves as a complete design that requires solely the actual implementation of the system or software. Often, this is done by using CASE tools (Computer Aided Software Engineering Tools). The main drawback of using CASE tools is that they require a certain level of expertise, user training as well as management and staff commitment.

Pseudo Programming Language

UML is not a stand-alone programming language like Java, C++ or Python, however, with the right tools, it can turn into a pseudo programming language. In order to achieve this, the whole system needs to be documented in different UML diagrams and, by using the right software, the diagrams can be directly translated into code. This method can only be beneficial if the time it takes to draw the diagrams would take less time than writing the actual code.

Despite UML having been created for modelling software systems, it has found several adoptions in business fields or non-software systems.

CLASS DIAGRAM:

Class UML diagram is the most common diagram type for software documentation. Since most software being created nowadays is still based on the Object-Oriented Programming paradigm, using class diagrams to document the software turns out to be a common-sense solution. This happens because OOP is based on classes and the relations between them.

In a nutshell, class diagrams contain classes, alongside with their attributes (also referred to as data fields) and their behaviours (also referred to as member functions). More specifically, each class has 3 fields: the class name at the top, the class attributes right below the name, the class operations/behaviours at the bottom. The relation between different classes (represented by a connecting line), makes up a class diagram.

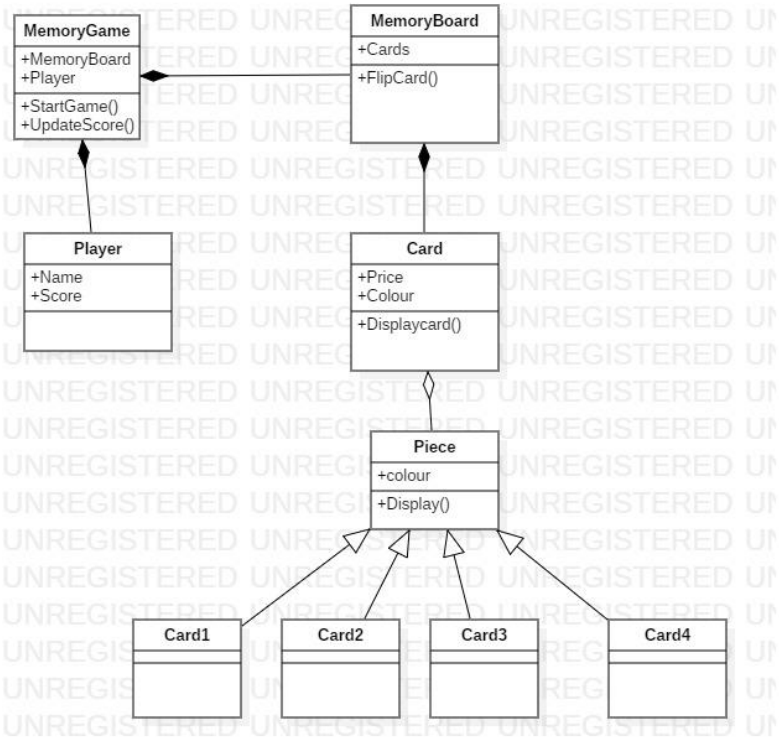

Fig. 5.4.1 Class Diagram

In the above class diagram we have six different classes named as memory game, memory card, player, card and piece. All these classes have their respective instances and operations.

In the memory game class we have two instances. These two are the memoryboard and player. The operations in the class named MemoryGame are StartGame and Update Score. In the MemoryBoard class we have one instance and one operation. The instance present in the MemoryBoard class is cards. The operations present in the class is FlipCard.

In player class we have two instances and four operations. The instances present in the player class are name and score. The operations present in this class are login, register, play and logout.

In the Card class we have two instances present and four operations. The instances in the Card are piece and colour. The operations present in this class is display.

USE CASE DIAGRAM:

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

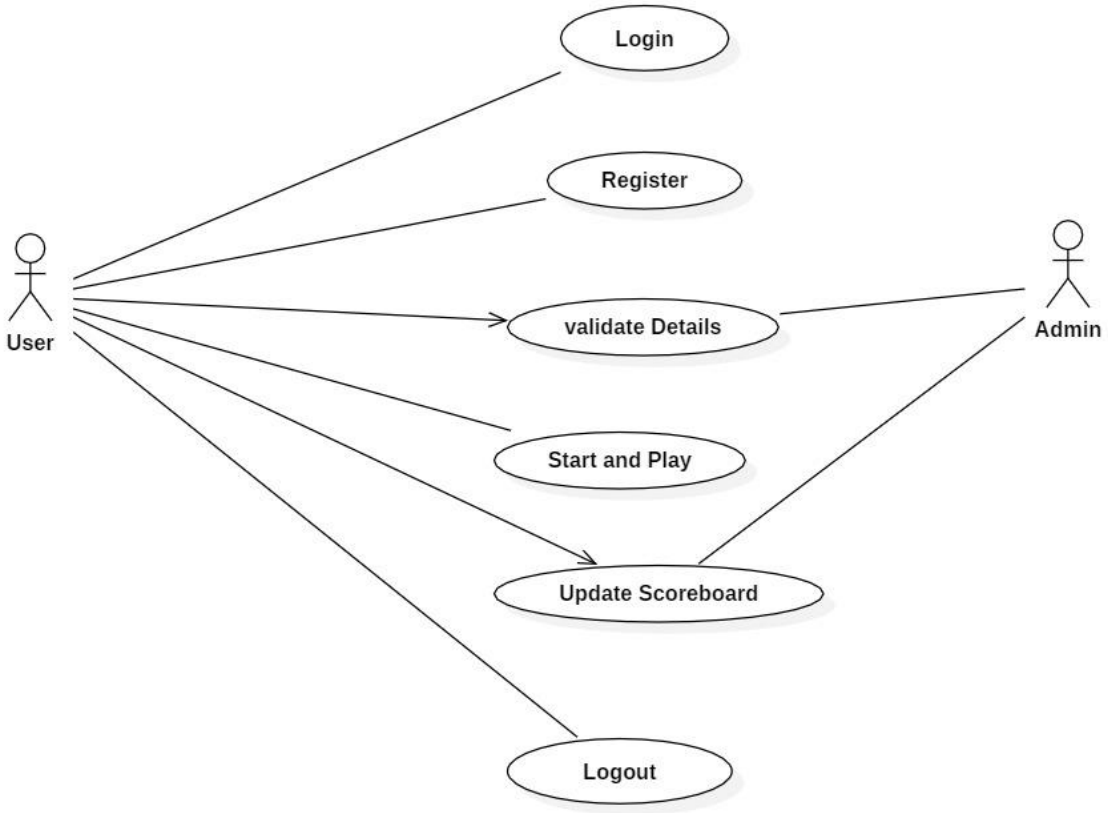

Fig. 5.4.2 Use Case Diagram

In the usecase diagram represented above we have two actors and six usecases. The actors are named as user and admin. The actor named used performs four actions. They are login, register, start and play and logout. The actors named admin performs the actions such as validate details and update Scoreboard.

SEQUENCE DIAGRAM:

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

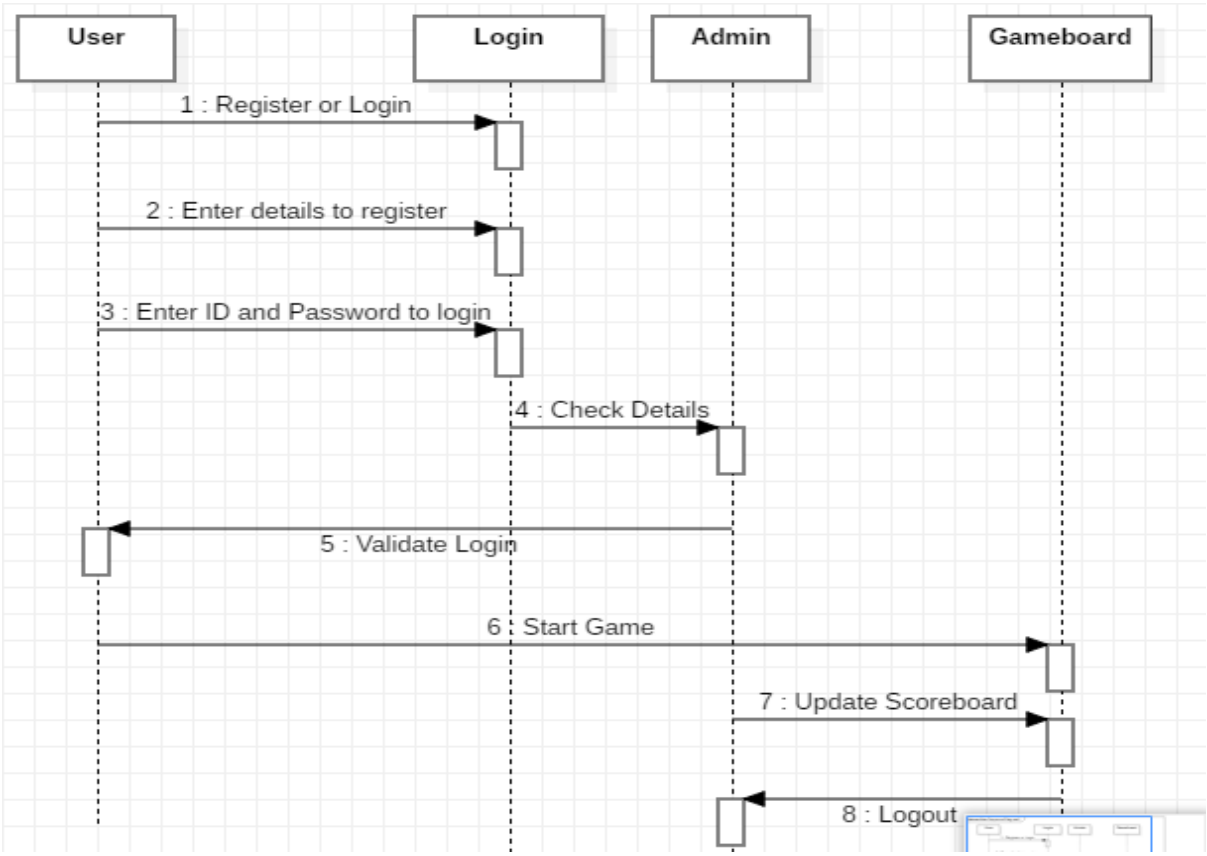

Fig. 5.4.3 Sequence Diagram

In the above sequence diagram we have four objects and 8 messages. The four objects are named as User, Login, Admin and Gameboard. The first message called Register or Login is passed between user and login. The second message called Enter details to register is passed between User and Login. The third message called Enter id and password to login is passed between User and Login. The fourth message called check details is passed between login and admin. The fifth message called validate login is passed between admin and user. The sixth message called start game is passed between user and gameboard. The seventh message called

update scoreboard is passed between admin and gameboard. The eight message called logout is passed between gameboard and admin.

COLLABORATION DIAGRAM:

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing.

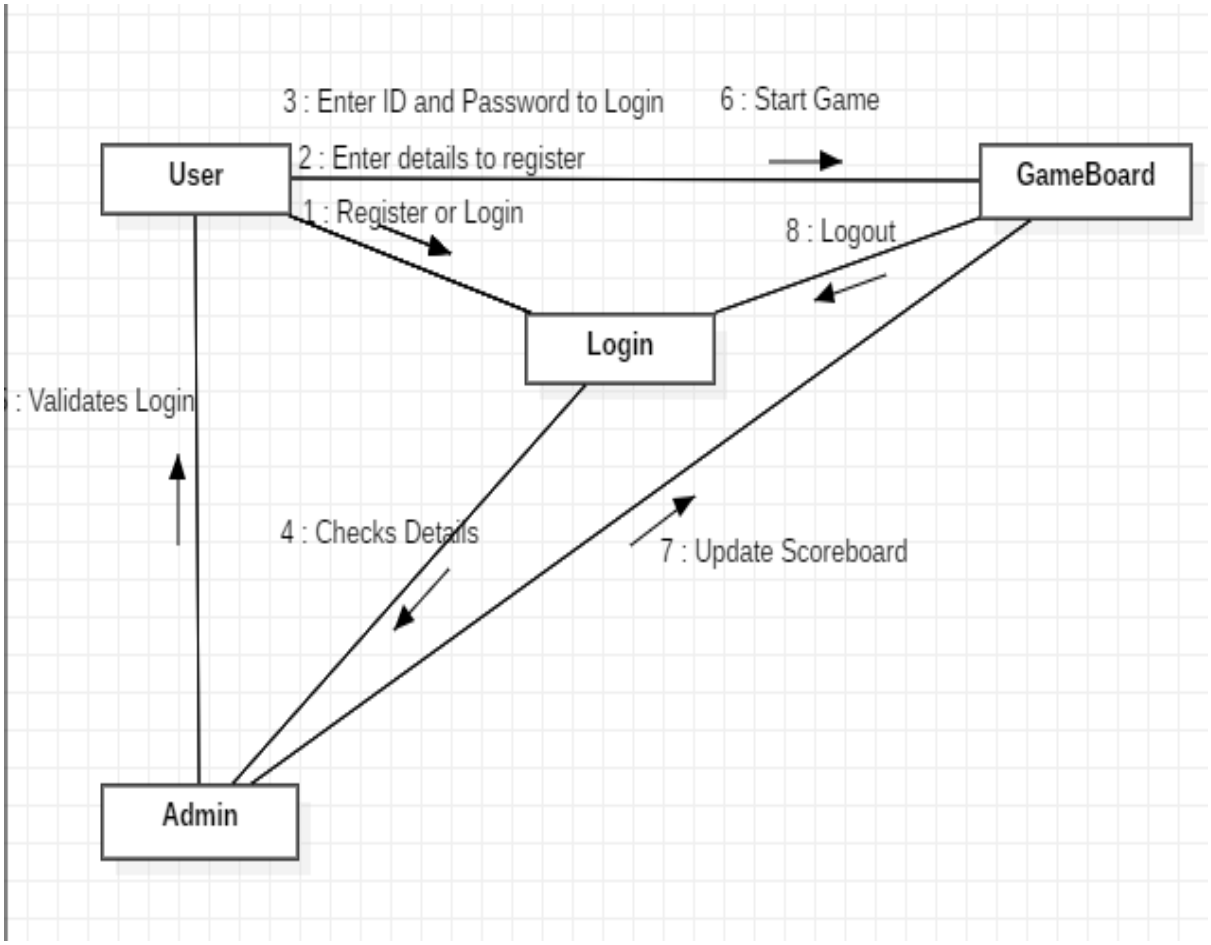

Fig. 5.4.4 Collaboration Diagram

In the above sequence diagram we have four objects and 8 messages. The four objects are named as User, Login, Admin and Gameboard. The first message called Register or Login is

passed between user and login. The second message called Enter details to register is passed between User and Login. The third message called Enter id and password to login is passed between User and Login. The fourth message called check details is passed between login and admin. The fifth message called validate login is passed between admin and user. The sixth message called start game is passed between user and gameboard. The seventh message called update scoreboard is passed between admin and gameboard. The eight message called logout is passed between gameboard and admin.

ACTIVITY DIAGRAM:

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

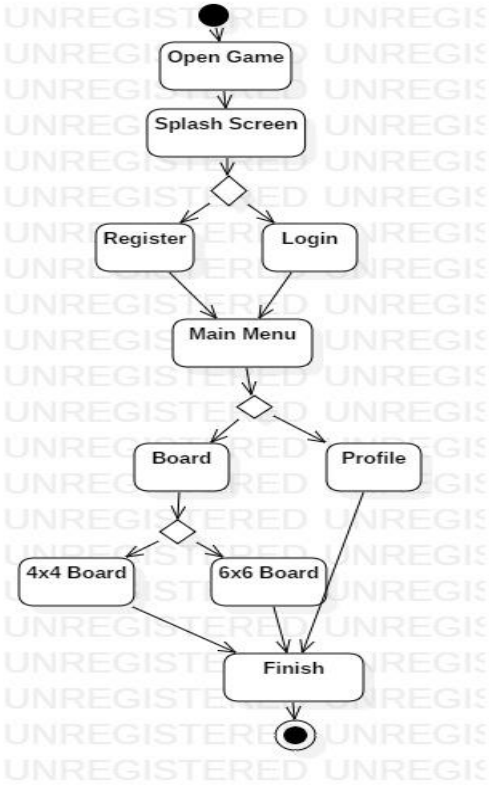

Fig. 5.4.5 Activity Diagram

In the above activity diagram we have different activities represented. From the initial point in the activity diagram we have the first activity as the open game, the second activity as the Splash game. This is forked as two activities named as register and login. These two activities are joined as Mainmenu activity. This activity is further forked as two activities named Board and profile. The board activity is forked as 4x4 board and 6x6 board. All the activities are joined as finish activity at the end.

CHAPTER 6

CODING

TECHNICAL DETAILS

This chapter is about the software language and the tools used in the development of the project. The platform used here is HTML,CSS,JVASCRIPT,ANGULAR and NODE.

FEATURES OF TECHNOLOGIES

NODEJS

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Node.js uses asynchronous programming

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request. Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

NODE.JS FILE

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

ANGULARJS SERVICES

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.AngularJS has about 30 built-in services. One of them is the $location service.The $location service has methods which return information about the location of the current web page:

THE $HTTP SERVICE

The $http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

THE $TIMEOUT SERVICE

The $timeout service is AngularJS' version of the window.setTimeout function.

THE $INTERVAL SERVICE

The $interval service is AngularJS' version of the window.setInterval function.

CREATE YOUR OWN SERVICE

To create your own service, connect your service to the module:

Create a service named hexafy:

```
app.service('hexafy', function() {
  this.myFunc = function (x) {
    return x.toString(16);
  }
});
```

To use your custom made service, add it as a dependency when defining the controller:

ANGULARJS API

API stands for **A**pplication **P**rogramming **I**nterface.

ANGULARJS GLOBAL API

The AngularJS Global API is a set of global JavaScript functions for performing common tasks like:

- Comparing objects
- Iterating objects
- Converting data

The Global API functions are accessed using the angular object.

Below is a list of some common API functions

angular.lowercase()

Example

```
<div ng-app="myApp" ng-controller="myCtrl">
  <p>{{ x1 }}</p>
  <p>{{ x2 }}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.x1 = "JOHN";
  $scope.x2 = angular.lowercase($scope.x1);
});
</script>
```

ANGULARJS ROUTING

ROUTING IN ANGULARJS

Navigate to different pages in your application, and also want the application to be a SPA (Single Page Application), with no page reloading, then ngRoute module can be used.

The ngRoute module *routes* application to different pages without reloading the entire application.

Example

```
<!DOCTYPE html>

<html>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>

<body ng-app="myApp">

<p><a href="#/!">Main</a></p>

<a href="#!red">Red</a>

<a href="#!green">Green</a>

<a href="#!blue">Blue</a>

<div ng-view></div>

<script>

var app = angular.module("myApp", ["ngRoute"]);

app.config(function($routeProvider) {

  $routeProvider

  .when("/", {

    templateUrl : "main.htm"

  })

  .when("/red", {

    templateUrl : "red.htm"

  })

  .when("/green", {
```

```
    templateUrl : "green.htm"

  })

  .when("/blue", {

    templateUrl : "blue.htm"

  });

});

</script>

<p>Click on the links to navigate to "red.htm", "green.htm", "blue.htm", or back to "main.htm"</p>

</body>

</html>
```

To make the applications ready for routing, include the AngularJS Route module:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>
```

Then add the ngRoute as a dependency in the application module:

```
var app = angular.module("myApp", ["ngRoute"]);
```

Now application has access to the route module, which provides the $routeProvider.

Use the $routeProvider to configure different routes in application:

```
app.config(function($routeProvider) {
 $routeProvider
 .when("/", {
  templateUrl : "main.htm"
 })
 .when("/red", {
  templateUrl : "red.htm"
 })
 .when("/green", {
  templateUrl : "green.htm"
 })
 .when("/blue", {
  templateUrl : "blue.htm"
```

```
  });
});
```

$ROUTEPROVIDER

$routeProvider can be used to define what page to display when a user clicks a link.

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>
<body ng-app="myApp">
<p><a href="#/!">Main</a></p>

<a href="#!london">City 1</a>
<a href="#!paris">City 2</a>
<p>Click on the links to read about London and Paris.</p>
<div ng-view></div>
<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
    $routeProvider
    .when("/", {
        templateUrl : "main.htm"
    })
    .when("/london", {
```

```
        templateUrl : "london.htm"
    })
    .when("/paris", {
        templateUrl : "paris.htm"
    });
});
</script>
</body>
</html>
```

CONTROLLERS

$routeProvider can also define a controller for each "view".

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>
<body ng-app="myApp">
<p><a href="#/!">Main</a></p>
<a href="#!london">City 1</a>
<a href="#!paris">City 2</a>
<p>Click on the links.</p>
<p>Note that each "view" has its own controller which each gives the "msg" variable a value.</p>
<div ng-view></div>
```

```
<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
    $routeProvider
    .when("/", {
        templateUrl : "main.htm",
    })
    .when("/london", {
        templateUrl : "london.htm",
        controller : "londonCtrl"
    })
    .when("/paris", {
        templateUrl : "paris.htm",
        controller : "parisCtrl"
    });
});
app.controller("londonCtrl", function ($scope) {
    $scope.msg = "I love London";
});
app.controller("parisCtrl", function ($scope) {
    $scope.msg = "I love Paris";
});
</script>
```

```
</body>

</html>
```

TEMPLATE

In the previous examples the templateUrl property in the $routeProvider.when method is used.Template property can also be used, which allows to write HTML directly in the property value, and not refer to a page.

Example

```
<!DOCTYPE html>

<html>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>

<body ng-app="myApp">

<p><a href="#/!">Main</a></p>

<a href="#!banana">Banana</a>

<a href="#!tomato">Tomato</a>

<p>Click on the links to change the content.</p>

<p>The HTML shown in the ng-view directive are written in the template property of the $routeProvider.when method.</p>

<div ng-view></div>

<script>

var app = angular.module("myApp", ["ngRoute"]);

app.config(function($routeProvider) {

  $routeProvider

  .when("/", {
```

```
    template : "<h1>Main</h1><p>Click on the links to change this content</p>"
  })
  .when("/banana", {
    template : "<h1>Banana</h1><p>Bananas contain around 75% water.</p>"
  })
  .when("/tomato", {
    template : "<h1>Tomato</h1><p>Tomatoes contain around 95% water.</p>"
  });
});
</script>
</body>
</html>
```

Application Code:

index.html:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Memory Game by Moshe</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
```

```
</html>
```

board.component.html:

```
<div
class="container-
fluid">

                    <!-- SELECT BOARD SIZE SECTION -->
                    <mat-form-field class="full-width">
                      <mat-select placeholder="Select a Board Size:"
                    [(ngModel)]="boardSize" [disabled]="isGameRunning">
                        <mat-option value="4">4x4</mat-option>
                        <mat-option value="6">6x6</mat-option>
                      </mat-select>
                    </mat-form-field>

                    <hr />

                    <!-- START AND STOP BUTTONS SECTION  -->
                    <div class="btn-group btn-group-justified">
                      <div class="btn-group">
                        <button mat-raised-button color="primary" class="btn-start"
                    (click)="onStart()" [disabled]="isGameRunning">START</button>
                      </div>
                      <div class="btn-group">
                        <button mat-raised-button color="warn" class="btn-stop"
                    (click)="onStop()" [disabled]="!isGameRunning">STOP</button>
                      </div>
                    </div><br />

                    <hr />

                    <!-- THE BOARD SECTION -->
```

```
 <mat-card class="back-img">

   <!-- THE HEADER -->
   <mat-card-header>
     <mat-card-title><h1 class="head-style">Board Size:
{{boardSize}}x{{boardSize}}</h1></mat-card-title>
   </mat-card-header>

   <hr>

   <!-- HERE IS WHERE THE CARDS PLACED -->
   <mat-card-content>

     <!-- PROGRESS BAR (showing when loading finished (3
seconds after clicking on START) ) -->
     <mat-progress-bar mode="indeterminate"
*ngIf="showLoading"></mat-progress-bar>

     <!-- THE COUNT TIME -->
     <p class="counter-text">{{minutes | countPipe}}:{{seconds |
countPipe}}</p>

     <hr>

     <!-- HERE IS THE CARDS -->
     <table class="table table-responsive table-condensed">
       <tr *ngFor="let tr of cards; let row = index" class="borderless">
         <td *ngFor="let card of cards[row]">
           <app-card [card]="card" (click)="onClick(card)"></app-card>
         </td>
       </tr>
     </table>
```

```
                    <!-- WHEN THE IS OVER -->
                    <div *ngIf="!isGameRunning && isGameOver" [@in]>
                      <p class="good-job" >GOOD JOB!!</p>
                    </div>

                  </mat-card-content>

                </mat-card>
                <ul>
                  <button class="common-btn btn"
                routerLink='/game'>Finish</button></ul>



                </div>
```

board.component.css:

```
.card-
panel
{
        display: inline-block;
        background-color: transparent;
      }

      .card-body {
        padding: 0;
        height: 150px;
        cursor: pointer;
      }

      .img-width {
        width: 110px;
```

```
}

.real-img {
  margin: 20% 0;
}

@media screen and (max-width: 768px) {
  .card-body {
    height: auto;
  }

  .img-style {
    margin-bottom: 20%;
  }
}
.back-img {
  background-image: url(../../assets/images/back-img.jpg);
  background-repeat: no-repeat;
  background-attachment: fixed;
  }

.good-job {
  text-align: center;
  color: #38d6f186;
  font-size: 75px;
  text-shadow: 0 0 15px wheat;
  }
  .counter-text {
    text-shadow: 0 0 10px gold;
    font-family: fantasy;
    color: aquamarine;
    opacity: 0.8;
    text-align: center;
```

```
  font-size: 35px;
}
.head-style {
  text-shadow: 0 0 10px gold;
  color:rgba(23, 122, 134, 0.7);
  }
```

board.component.ts:

```
import {
Component,
Input } from
'@angular/core';
                    import { trigger, state, style, transition, animate } from
                    '@angular/animations';

                    import { Card } from '../shared/card.model';
                    import { GameService } from '../game.service';
                    import { MatDialog } from '@angular/material';
                    import { Accomplishment } from '../shared/accomplishment.model';
                    import { HttpErrorResponse } from '@angular/common/http';
                    import { PlayerDialogComponent } from '../dialogs/player-
                    dialog/player-dialog.component';
                    // import { Router } from "@angular/router";

                    @Component({
                      selector: 'app-board',
                      templateUrl: './board.component.html',
                      styleUrls: ['./board.component.css'],
                      animations: [
                        trigger('in', [
                          transition('void => *', [
```

```
        style({
          transform: 'rotate(360deg) scale(0)'
        }),
        animate(1000, style({
          transform: 'rotate(0) scale(1)'
        }))
      ])
    ])
  ]
})
export class BoardComponent {

  constructor(private gameService: GameService ,
    private dialog: MatDialog  ) { }
  boardSize = '4'; // stored as string to set a default value to Select input
(if as a number it won't work).
  cards: Card[][] = [];
  showLoading = false;
  isGameRunning = false;
  isGameOver = false;
  clickedCard: Card; // this is the first card was clicked, after well
compared with the second card.
  clicks = 0; // count the clicks. (maximum 2)
  delayTimer;

  players: Accomplishment [] = [];
  startedTime: number; // this will hold the started time as milliseconds.

  countInterval: any;
  seconds = 0;
  minutes = 0;
```

```
  ngOnInit(): void { // getting the accomplishments from the server (firebase) if there is.
    this.gameService.getAccomplishments().
    subscribe(
      (players) => {
       if (players !== null) {
         this.players = players;
       }
      }, (error: HttpErrorResponse ) => { // log to the console in error case.
        console.log(error.message);
        console.log(error);
      }
    );
  }
  onStart(): void {
    const tempCards = this.gameService.getCards(+this.boardSize); // get the cards as a completely new instances.
    this.gameService.suffleCards(tempCards, +this.boardSize); // then suffle the Cards.

    this.isGameOver = false;
    this.showLoading = true;
    this.isGameRunning = true;

    this.delayTimer = setTimeout(() => {
        this.cards = tempCards; // setting the 'Real Cards' to the tempCards.
        this.showLoading = false;
        this.startedTime = new Date().getTime(); // current time stored as milliseconds as we said.
```

```
        this.countInterval = setInterval(() => { // just a simple timer to out
put to the user.
          this.seconds++;
          if (this.seconds === 59) {
            this.seconds = 0;
            this.minutes++;
          }
        }, 1000);

      }, 3000);

  }
  onStop(): void { // reset all
    this.cards = [];
    this.isGameRunning = this.showLoading = false;
    this.clicks = this.seconds = this.minutes = 0;
    clearTimeout(this.delayTimer);
    clearInterval(this.countInterval);
  }
  onClick(card: Card): void {
    // if the user click twice and the card didn't succeed
    if (this.clicks < 2 && !card.succeed) {
      this.clicks++;
      card.showImage = true; // show the 'Real' card image
      if (this.clicks === 1) { // in the first click save the card to property
'clickedCard'
        this.clickedCard = card;
      } else if (this.clicks === 2) { // if it's the second click...
        if (this.clickedCard === card) { // if the user clicked on exact the
same card (checking by its reference object)
          this.clicks--; // so he be able to click on another card.
          return;
        }
```

```
      if (this.clickedCard.ID === card.ID) { // on success

        setTimeout(() => {
          this.clicks = 0;
          this.clickedCard.succeed = true; // put the card as succeed state to true
          card.succeed = true; // put the card as succeed state to true
          if (this.isGameFinished () === true) { // check every time on success if the game is finished (all cards succeed)
            this.onFinished();
          }
        }, 400);

      } else { // on fail

        setTimeout(() => {
          this.clicks = 0; // reset the clicks
          this.clickedCard.showImage = false; // reverse the card
          card.showImage = false; // reverse the card
        }, 800);

      }
    }
  }


 }
 isGameFinished(): boolean {
   if (this.cards[0] === undefined) { // this is to avoid error which happens at the beginning.
     return false;
   }
```

```
    for (let i = 0; i < +this.boardSize; i++) { // run on all rows.

      if (this.cards[i].every((card: Card) => { // run every card in that
row.
        if (card.succeed === false) { // if some card didn't succeed yet
return false, which mean the game is not over.
          return false;
        }
        return true; // if the card succeed return true to keep checking in
the next card.
      }) === false) { // if the function every return false which mean
some card didn't succeed return false and exit from the function.
        return false;
      }

    }
    return true; // if all passed success (no return false happened) return
true, which mean the game IS Over!.
  }

  // get the fastest Accomplishment.
  getTheFastest(boardSize: number): number {
    const accomplishment = this.players.find((player) => {
      return player.size === boardSize;
      // remember the array is already sorted, so all we need is the find
the first player with same boardSize which is the fastest.
    });

    if (accomplishment === undefined) { // if there is no
accomplishment return some bigggg number.
      return 99999999999999;
    }
```

```
    return accomplishment.totalTime;
  }

  onFinished(): void {
    const min = this.minutes;
    const sec = this.seconds;

    this.isGameOver = true;
    this.isGameRunning = false;
    this.clicks = this.seconds = this.minutes = 0;
    this.cards = [];
    clearInterval(this.countInterval);
    const tTime = new Date().getTime() - this.startedTime; // totalTime
as milliseconds.

    if (tTime < this.getTheFastest(+this.boardSize)) { // if the new
accomplishment is faster.
      const dialogRef = this.dialog.open(PlayerDialogComponent , {
hasBackdrop: false, data: min + ':' + sec });

      dialogRef.afterClosed().subscribe(
        (name: string) => {
          if (name.length > 3) {
            this.players.push({ // push a new Accomplishment
              name: name,
              size: +this.boardSize,
              totalTime: tTime,
              minutes: min,
              seconds: sec
            });
            this.gameService.saveAccomplishments (this.players); // save
all
          }
```

```
        });
      }

    }
  }
```

game.component.css:

```
.borderless
td,
.borderless
th {
    border: none;
}

.head-style {
text-shadow: 0 0 10px gold;
color:rgba(23, 122, 134, 0.7);
}

.btn-start {
width: 100%;
}

.btn-stop {
width: 100%;
}

.back-img {
background-image: url(../../assets/images/back-img.jpg);
background-repeat: no-repeat;
```

```
background-attachment: fixed;
}

.counter-text {
text-shadow: 0 0 10px gold;
font-family: fantasy;
color: aquamarine;
opacity: 0.8;
text-align: center;
font-size: 35px;
}

.good-job {
text-align: center;
color: #38d6f186;
font-size: 75px;
text-shadow: 0 0 15px wheat;
}
```

game.component.html:

```
<div
class="jumbotron">
                    <h1 class="text-center">Memory Game</h1><br><br>

                      <img src="../favicon.ico" alt="logo" class="img-responsive
                  center-block img-thumbnail" /><br><br>

                      <!-- ACCOMPLISHMENTS SECTION -->
                      <label class="text-info" for="list-
                  accomplishments">Accomplishments:</label>
```

```
    <p class="text-warning" *ngIf="players.length === 0">there is
    no accomplishments yet!</p>
    <ul class="list-group" style="width: 290px;" id="list-
    accomplishments">
      <li class="list-group-item" *ngFor="let player of players |
    sortPlayers">
        {{player.name}}, finished at -
        <span class="text-warning">{{player.minutes |
    countPipe}}:{{player.seconds | countPipe}}</span>
        , on - <span class="text-danger">{{player.size === 4 ? '4x4' :
    '6x6'}}</span>.</li>
    </ul>
    <ul>
        <button class="common-btn btn"
    routerLink='/board'>board</button></ul>
    </div>
```

game.component.ts:

```
import {
Component,
Input } from
'@angular/core';
                    import { trigger, state, style, transition, animate } from
                    '@angular/animations';

                    import { Card } from '../shared/card.model';
                    import { MatDialog } from '@angular/material';
                    import { GameService } from '../game.service';
                    import { Accomplishment } from '../shared/accomplishment.model';
                    // import { PlayerDialogComponent } from '../dialogs/player-
                    dialog/player-dialog.component';
```

```
import { HttpErrorResponse } from '@angular/common/http';
import { Router } from '@angular/router';

@Component({
  selector: 'app-game',
  templateUrl: './game.component.html',
  styleUrls: ['./game.component.css'],
  animations: [
    trigger('in', [
      transition('void => *', [
        style({
          transform: 'rotate(360deg) scale(0)'
        }),
        animate(1000, style({
          transform: 'rotate(0) scale(1)'
        }))
      ])
    ])
  ]
})
export class GameComponent {


  constructor(private gameService: GameService ,private router:
Router ,
    private dialog: MatDialog  ) { }
  //boardSize = '4'; // stored as string to set a default value to Select
input (if as a number it won't work).
  //cards: Card[][] = [];
  showLoading = false;
  isGameRunning = false;
  isGameOver = false;
```

```
  clickedCard: Card; // this is the first card was clicked, after well
compared with the second card.
  clicks = 0; // count the clicks. (maximum 2)
  delayTimer;

  players: Accomplishment [] = [];
  startedTime: number; // this will hold the started time as milliseconds.

  countInterval: any;
  seconds = 0;
  minutes = 0;

  ngOnInit(): void { // getting the accomplishments from the server
(firebase) if there is.
    this.gameService.getAccomplishments().
    subscribe(
      (players) => {
       if (players !== null) {
         this.players = players;
       }
      }, (error: HttpErrorResponse ) => { // log to the console in error
case.
       console.log(error.message);
       console.log(error);
      }
    );
  }
  getTheFastest(boardSize: number): number {
    const accomplishment = this.players.find((player) => {
      return player.size === boardSize;
      // remember the array is already sorted, so all we need is the find
the first player with same boardSize which is the fastest.
    });
```

```
    if (accomplishment === undefined) { // if there is no
accomplishment return some bigggg number.
      return 99999999999999;
    }

    return accomplishment.totalTime;
  }

  // start (board){
    // this.router.navigateByUrl('board');
  // }

  //public onStart() {
    // this.router.navigate(['/board']).then( (e) => {
      // if (e) {
        // console.log("Navigation is successful!");
      // } else {
        // console.log("Navigation has failed!");
      // }
    // });
// }

}
```

app.component.ts:

```
import {
Component}
from
'@angular/core';
                    // tslint:disable-next-line:import-blacklist
                    import 'rxjs/Rx';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],

})
export class AppComponent {
  constructor() {}


}
```

app.module.ts:

```
import {
NgModule }
from
'@angular/core';
                    // Auth service
                    import { AuthService } from './Auth_1/shared/services/auth.service';
                    import { BrowserModule } from '@angular/platform-browser';
                    import { DashboardComponent } from
                    './Auth_1/components/dashboard/dashboard.component';
                    import { ForgotPasswordComponent } from
                    './Auth_1/components/forgot-password/forgot-password.component';
                    import { AngularFireAuthModule } from '@angular/fire/auth';
                    //import { AngularFireModule } from '@angular/fire';
                    import { AngularFirestoreModule } from '@angular/fire/firestore';
```

```
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { ReactiveFormsModule } from '@angular/forms';
import { SignInComponent } from './Auth_1/components/sign-in/sign-in.component';
import { SignUpComponent } from './Auth_1/components/sign-up/sign-up.component';
import { VerifyEmailComponent } from './Auth_1/components/verify-email/verify-email.component';
import { environment } from '../environments/environment';
import { AppComponent } from './app.component';
import { AppRoutingModule1 } from './Auth_1/shared/routing/app-routing.module';
import { CardComponent } from './card/card.component';
import { SortPlayersPipe } from './pipes/sort-players.pipe';
import { PlayerDialogComponent } from './dialogs/player-dialog/player-dialog.component';
import { CountPipe } from './pipes/count.pipe';
import { AngularMaterialModule } from './angular-material.module';
import { importType } from '@angular/compiler/src/output/output_ast';
import { from } from 'rxjs';
import { AngularFireModule } from '@angular/fire';
import { GameComponent } from './game/game.component';
import { BoardComponent } from './board/board.component';
import { GameModule } from './game/game.module';

@NgModule({
  declarations: [
    AppComponent,
    CardComponent,
    SortPlayersPipe,
    CountPipe,
```

```
    PlayerDialogComponent,
    SignInComponent,
    SignUpComponent,
    DashboardComponent,
    ForgotPasswordComponent,
    VerifyEmailComponent,
    GameComponent,
    BoardComponent


  ],
  imports: [
    BrowserModule,
    AppRoutingModule1,
    AngularFireModule.initializeApp(environment.firebase ),
    AngularFireAuthModule,
    AngularFirestoreModule,
    ReactiveFormsModule,
    FormsModule,
    HttpClientModule,
    AngularMaterialModule,
    GameModule
  ],
  entryComponents: [
    PlayerDialogComponent
  ],
  providers: [AuthService],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

CHAPTER 7

TESTING

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

7.1 SOFTWARE VALIDATION

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software?".
- Validation emphasizes on user requirements.

7.2 SOFTWARE VERIFICATION

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications?"

- Verifications concentrate on the design and system specifications.

7.3 TARGET OF THE TEST AREA

- **Errors** -These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, considered as an error.
- **Fault -** When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

7.4 BLACK-BOX TESTING

It is carried out to test functionality of the program. It is also called 'Behavioral' testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok', and problematic otherwise.

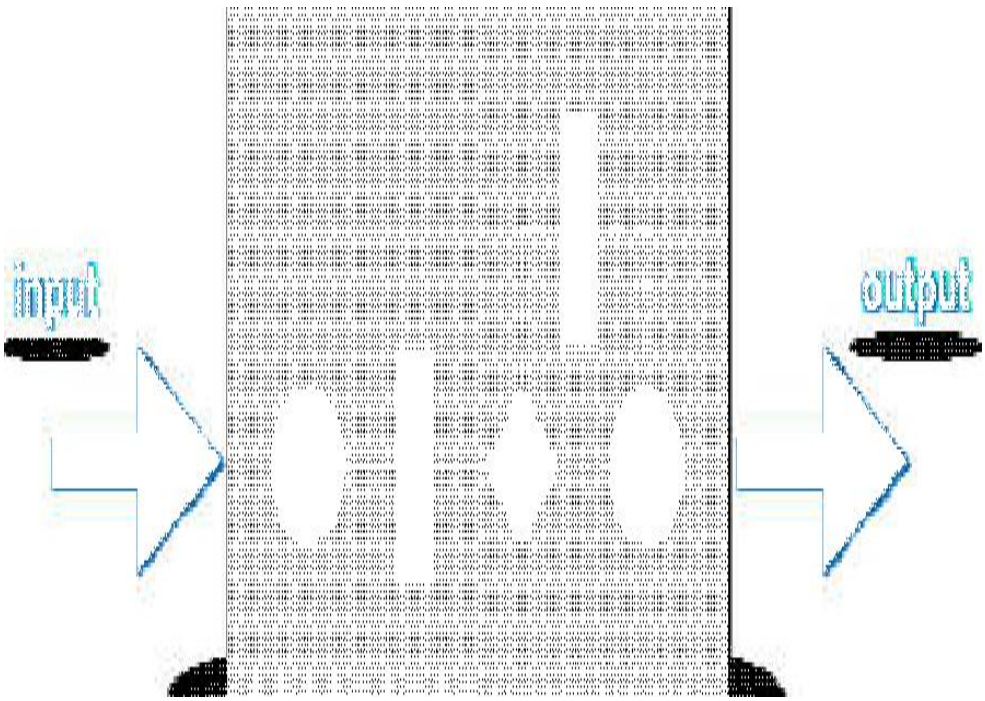

Fig 7.1:Black box testing

In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

7.4.1 BLACK-BOX TESTING TECHNIQUES

- **Equivalence class** - T he input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.
- **Boundary values** - T he input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
- **Cause-effect graphing** - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.
- **Pair-wise Testing** - The behavior of software depends on multiple parameters. In pair wise testing, the multiple parameters are tested pair-wise for their different values.
- **State-based testing** - The system changes state on provision of input. These systems are tested based on their states and input.

7.5 WHITE-BOX TESTING

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as ‘Structural’ testing.

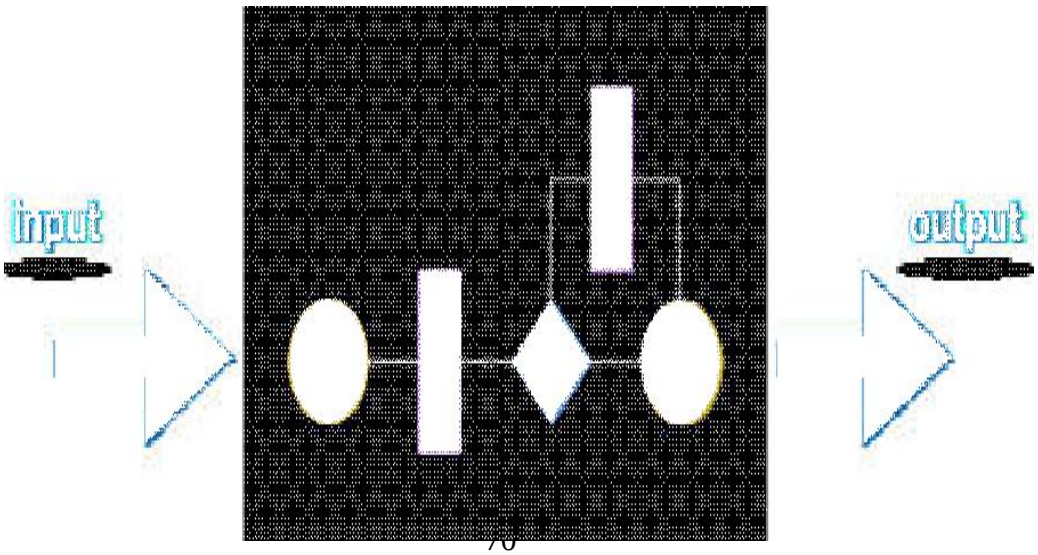

Fig 7.2: White box testing

In this testing method, the de sign and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The following are some White-box testing techniques

- **Control-flow testing** - The purpose of the control-flow testing to set up a test case which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.
- **Data-flow testing** - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

7.6 TESTING LEVELS

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified. Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels -

7.6.1 UNIT TESTING

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

7.6.2 INTEGRATION TESTING

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updating etc.

7.6.3 SYSTEM TESTING

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- **Functionality testing** - Tests all functionalities of the software against requirement.
- **Performance testing** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.
- **Security & Portability** - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

7.6.4 ACCEPTANCE TESTING

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

- **Alpha testing** - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.

- **Beta testing** - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

7.6.5 REGRESSION TESTING

Whenever a software product is updated with new code, feature or functionality, it is tested thoroughly to detect if there is any negative impact of the added code.

7.7 TEST YOUR APP

Android Studio is designed to make testing simple. With just a few clicks, you can set up a JUnit test that runs on the local JVM or an instrumented test that runs on a device. Of course, you can also extend your test capabilities by integrating test frameworks such as Mockito to test Android API calls in your local unit tests, and Espresso or UI Automator to exercise user interaction in your instrumented tests. You can generate Espresso tests automatically using Espresso Test Recorder.

7.8 LOCAL UNIT TESTS

Located at module-name/src/test/java/.

These are tests that run on your machine's local Java Virtual Machine (JVM). Use these tests to minimize execution time when your tests have no Android framework dependencies or when you can mock the Android framework dependencies.

At runtime, these tests are executed against a modified version of android.jar where all final modifiers have been stripped off. This lets you use popular mocking libraries, like Mockito.

7.9 INSTRUMENTED TESTS

These are tests that run on a hardware device or emulator. These tests have access to Instrumentation APIs, give you access to information such as the Context of the app you are testing, and let you control the app under test from your test code. Use these tests when writing integration and functional UI tests to automate user interaction, or when your tests have Android dependencies that mock objects cannot satisfy.

Because instrumented tests are built into an APK (separate from your app APK), they must have their own AndroidManifest.xml file. However, Gradle automatically generates this file during the build so it is not visible in your project source set. You can add your own manifest file if necessary, such as to specify a different value for `minSdkVersion` or register run listeners just for your tests. When building your app, Gradle merges multiple manifest files into one manifest.

7.10 RUN A WEB PAGE

To run a test, proceed as follows:

1. Be sure your project is placed in Angular Folder.

2. Run your test in one of the following ways:

    - In the Project folder , open command prompt window

    - In the command prompt window, run ng serve

    - It will automatically open the browser on ip address localhost:4200

7.11 USER TESTING

Along with the other testing methods in this guide, user testing can provide specific and valuable insights about the usability of your app.

To find users who can test your app, use methods such as the following:

- Reach out to local organizations, colleges, or universities that provide training for people with disabilities.

- Ask your social circle. There might be people with disabilities who are willing to help.

- Ask a user testing service (such as usertesting.com) if they can test your app and include users with disabilities.

- Join an accessibility forum, such as Accessibility or Eyes-free, and ask for volunteers to try your app.

7.12 TEST CASES AND RESULTS

Table No.7.1 Test cases and Results

TEST ID	TEST CASES	TEST RESULTS
1	Enter valid user name and valid password to login	User has access to main menu or use.
2	Use invalid email in login page	Shows that the user is invalid.
3	Use already used email in registration	Shows alter that the email already exists.
4	Select the board size	Selected board is displayed with relevant cards.
5	Add accomplishments	Accomplishments are added to the home page of the game.
6	Verification of email id	The game page is opened only after verification of the email.

CHAPTER 8

SCREEN SHOTS

8.1 DESKTOP APPLICATION

SIGN-IN PAGE

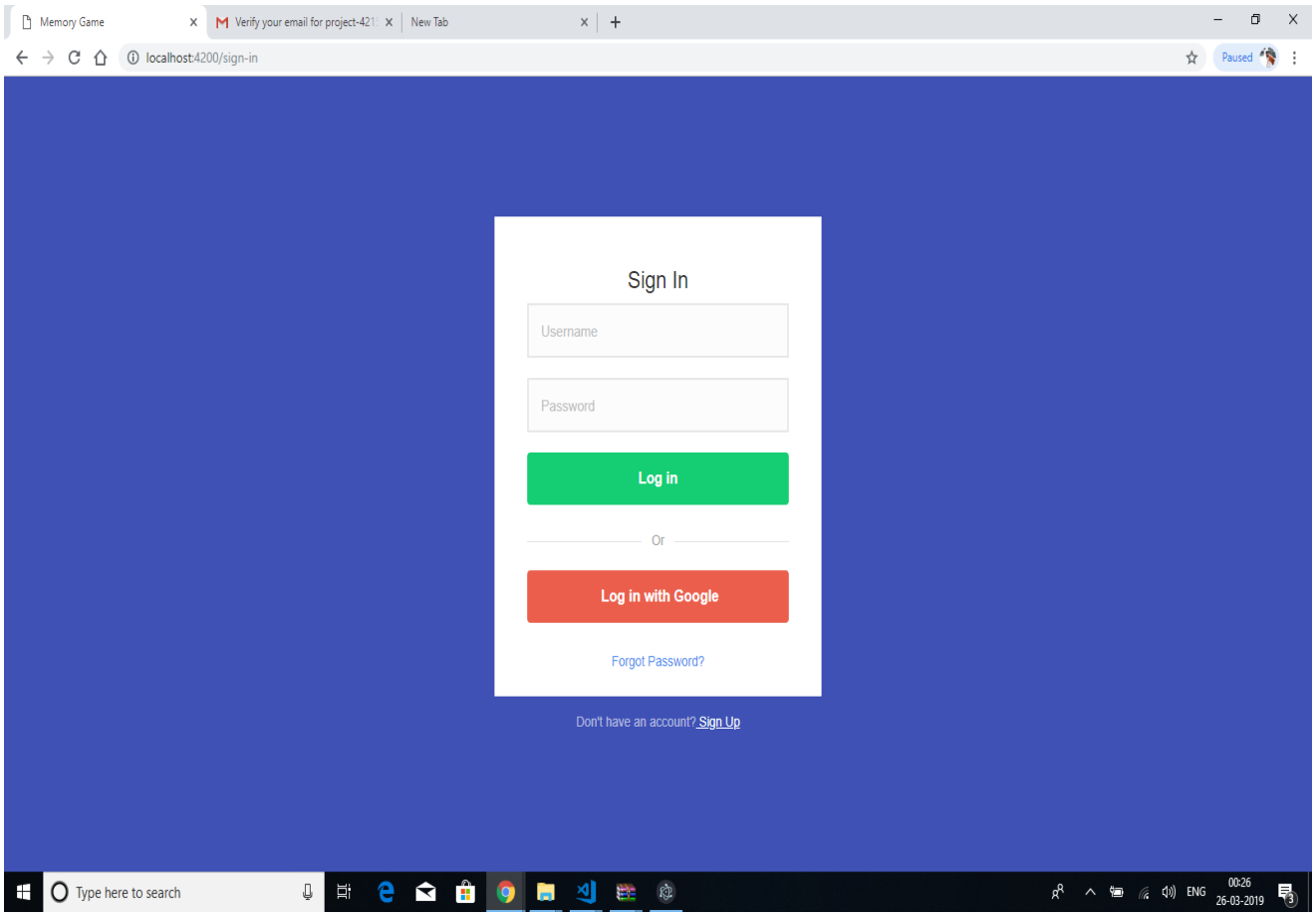

Fig. 8.1.1 Sign-In page

In the sign In page we have the two different options such as Login or Login in the google .By selecting one of the option we will be able to Sign in into our existing account. The existing

account is the one which is created prior and it is being used by the registered user. Once one of the above options is selected ,then we use our username which being a registered username for the account. Later we enter the password for the respective registered account. Then we will be taken into our registered account. In this page ,we do have another option as forgot password. This option is used when the user do not remember his account password. By using this option he can get a verification code to our gmail account. By using this verification code we can set a new password to our account. In this page ,we have an option called sign up for the users those who have not yet registered.

SIGN UP PAGE

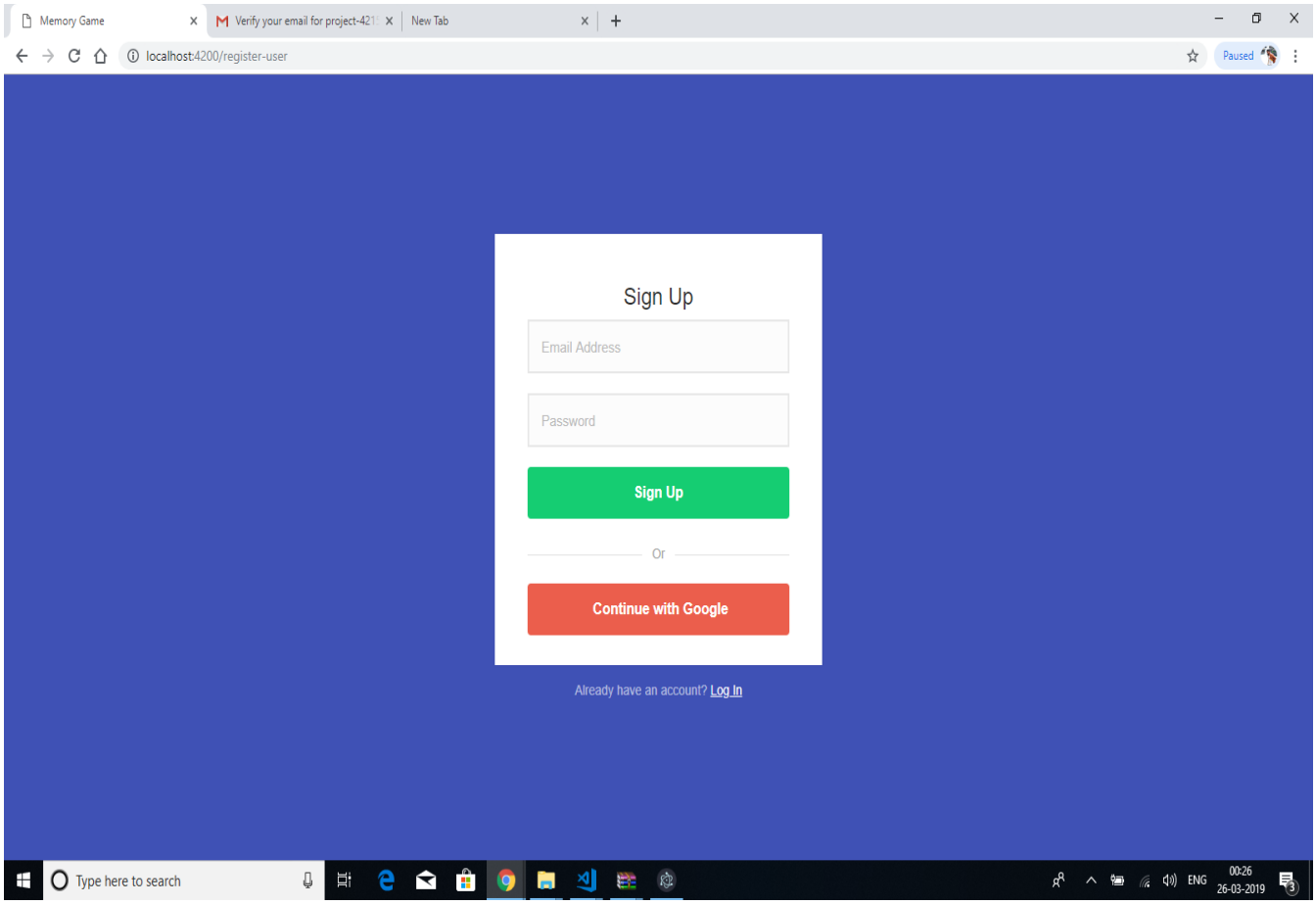

Fig. 8.1.2 Sign-Up page

For the Sign-Up page, we use our email address and set our account password. Here we also have an option called Log In for the people those who have already registered. By clicking this option they are directed to the Sign In page, where they can sign in into their registered account.

USER PROFILE PAGE

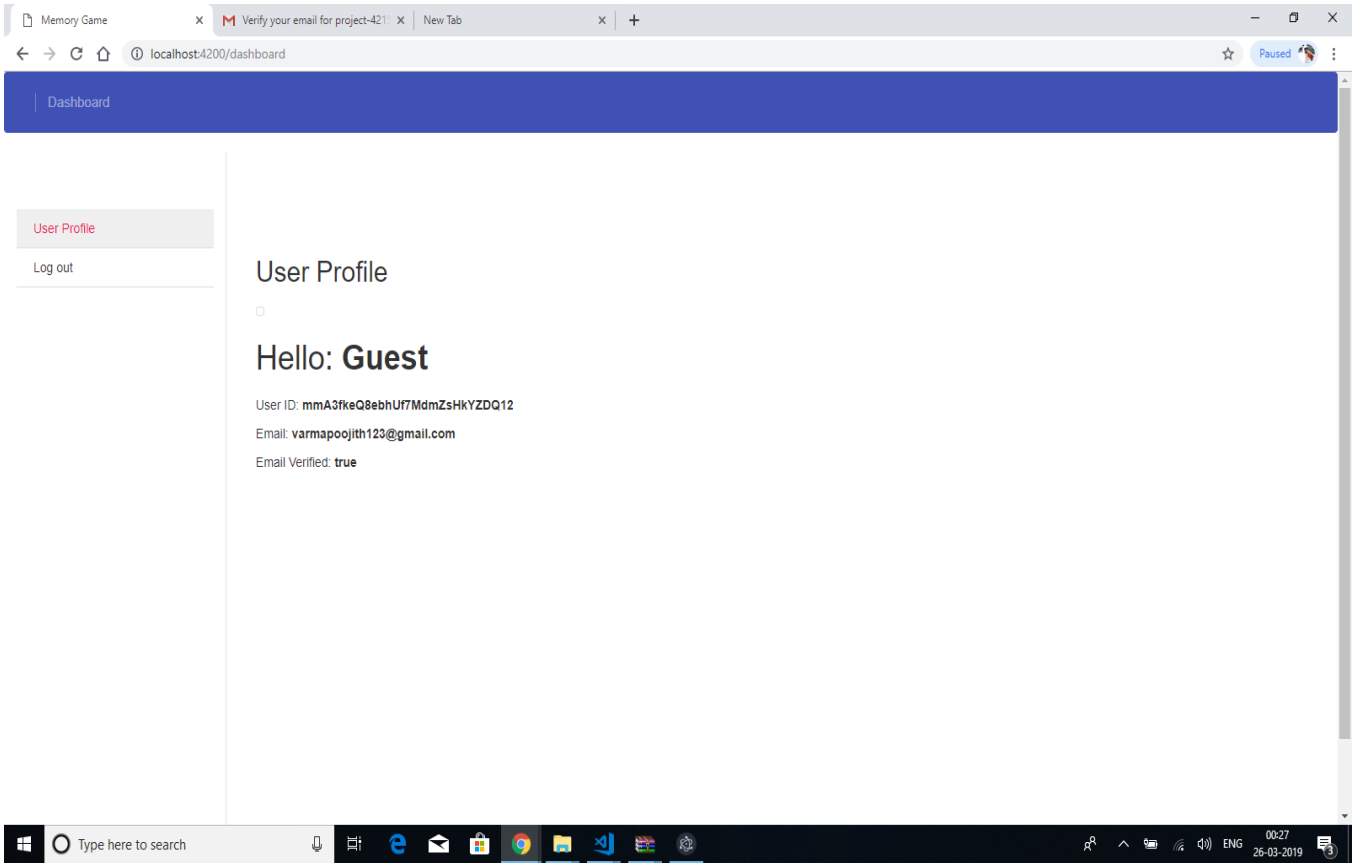

Fig. 8.1.3 User Profile Page

Here in the user profile page, we have the information regarding the user. It displays the registered user mail id, his user ID, and it also displays whether the email has been verified or not. It also consists of the option called as Log Out to get out the profile page.

HOME PAGE

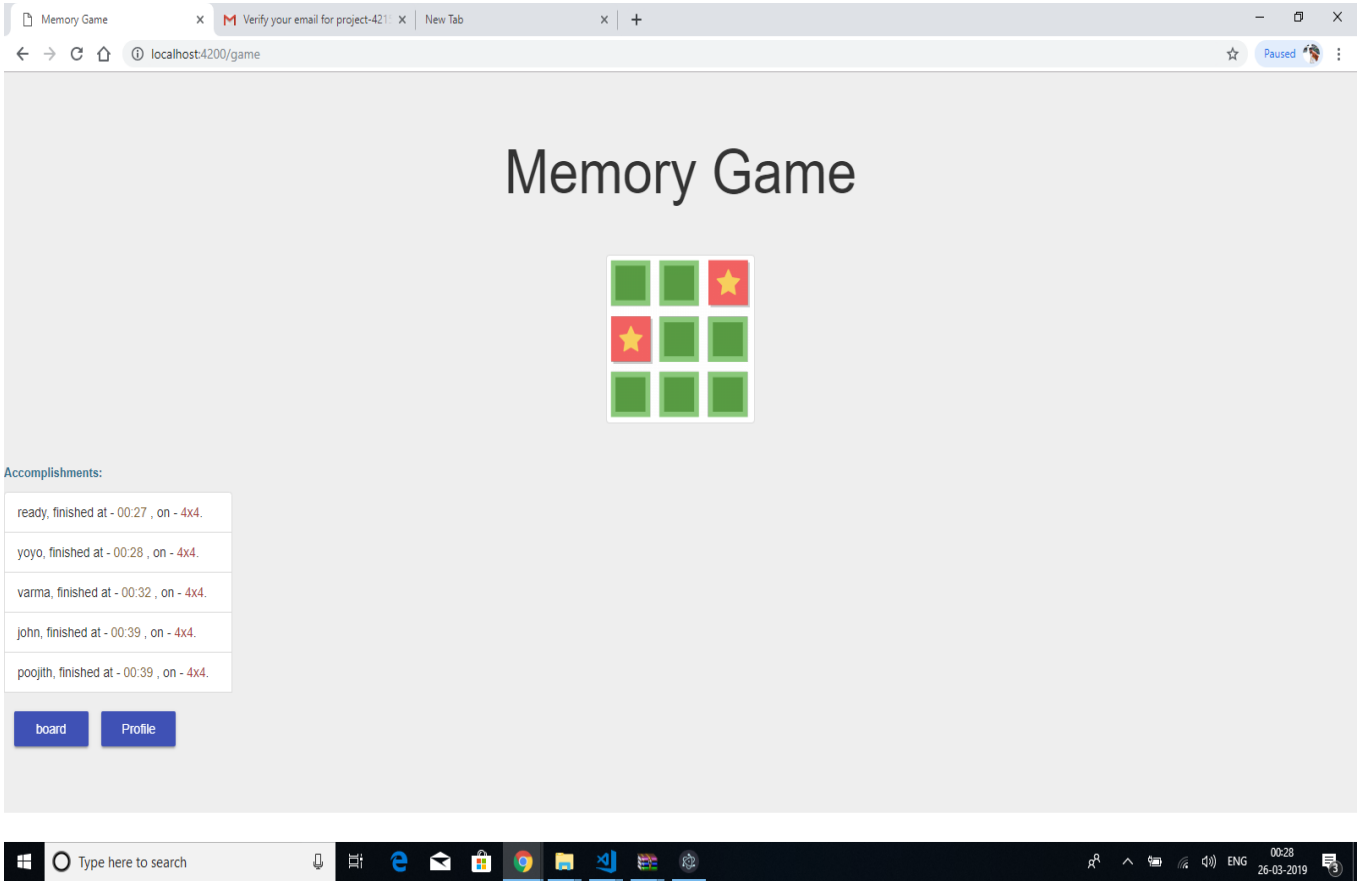

Fig. 8.1.4 Home Page

The home page consists of the game logo. It also consists of the top five accomplishments. These accomplishments are gathered from the various players games. Among them only top five are displayed over the homepage.

Here we also have other options like board and profile.

By clicking the board option, we can select the 4x4 board or 6x6 board. we can also use profile option to check out the profile of the user.

GAME PAGE SCREEN 1

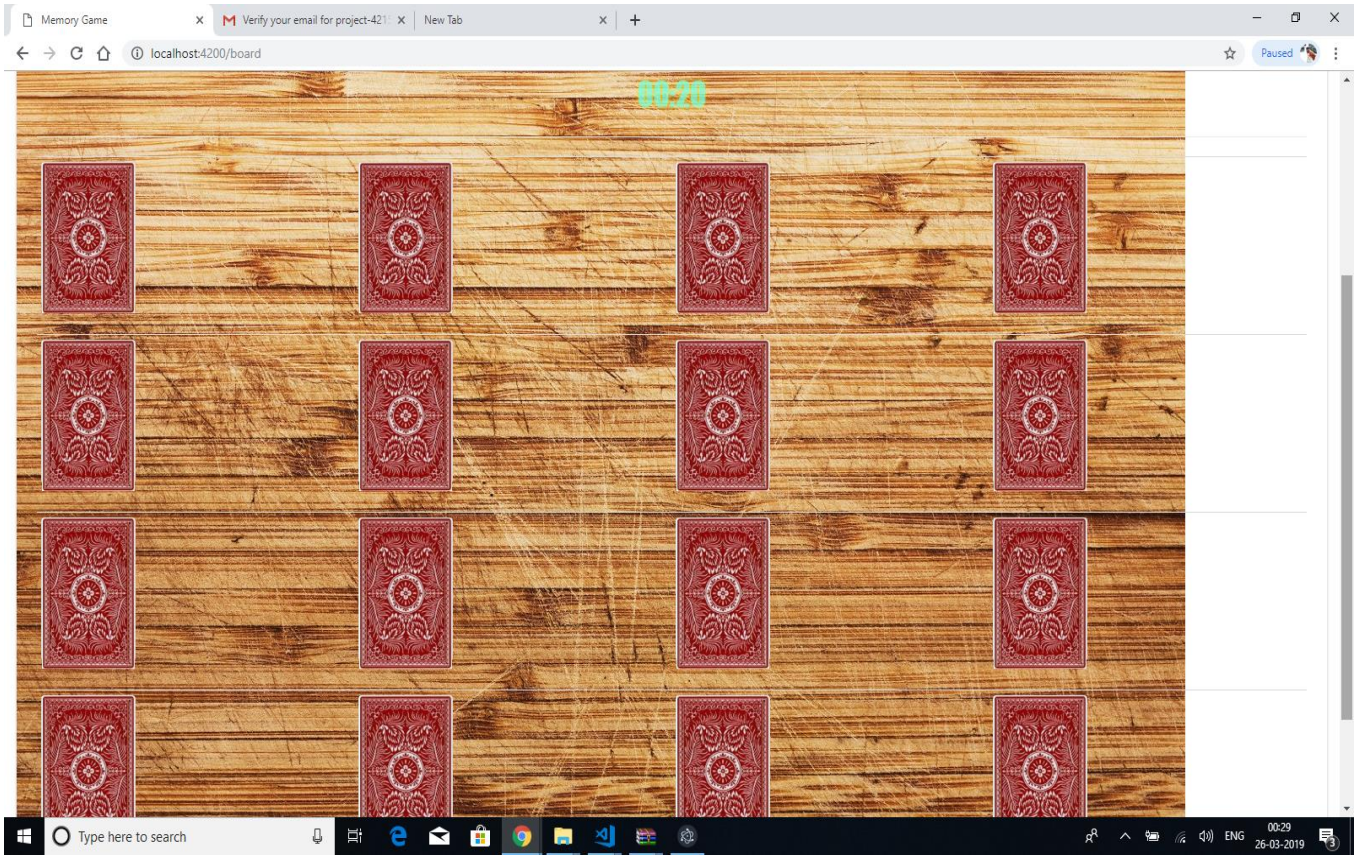

Fig. 8.1.5 Game Page Screen 1

It consists of the cards that are of the size 4x4 or 6x6.In 4x4 board, we have total 16 cards. In the 6x6 board, we have total 36 cards. We also have time duration that indicates the time that we have utilised in order to complete the game.

GAME PAGE SCREEN 2

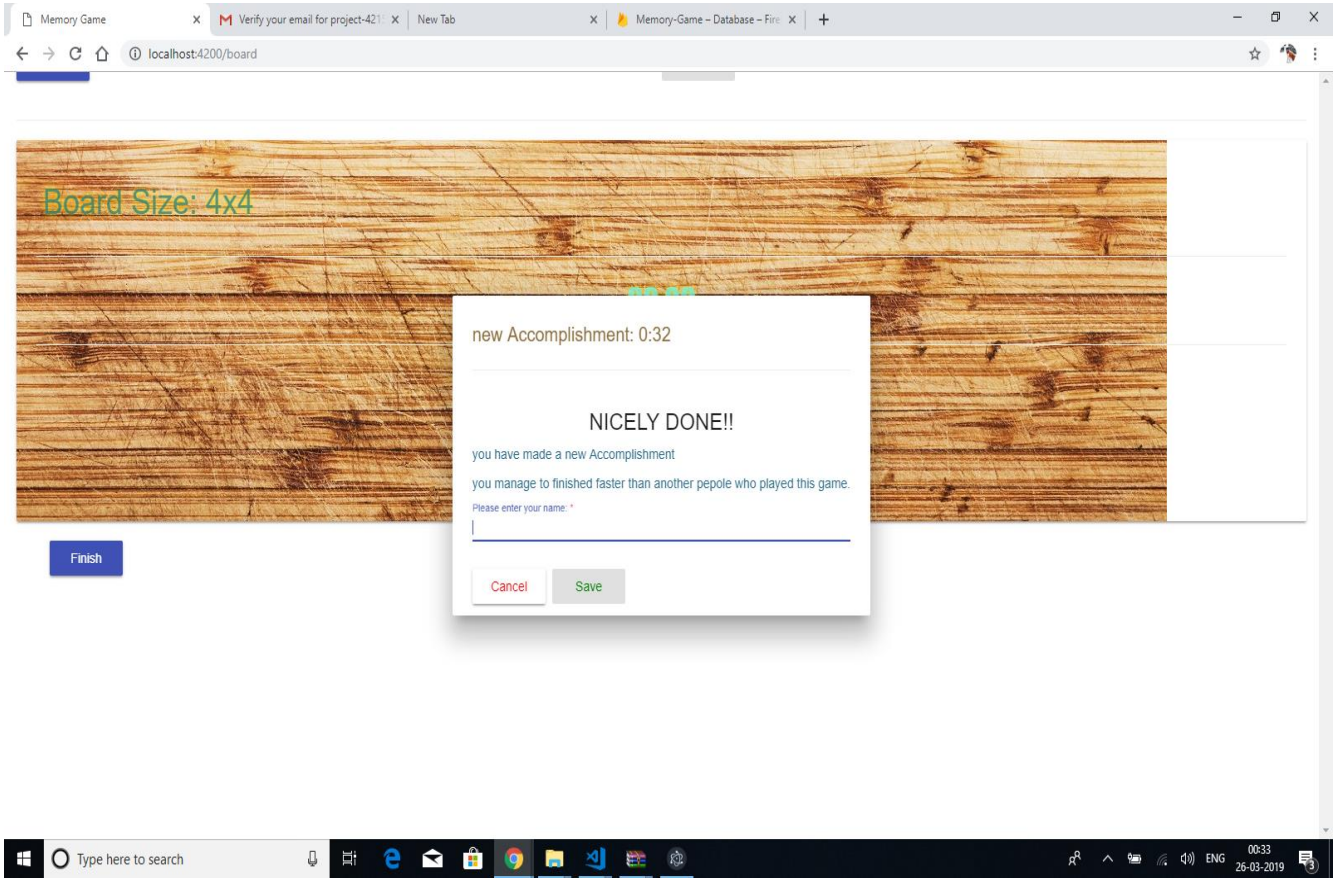

Fig. 8.1.6 Game Page Screen 2

Here in this page, if we have made one of the top five accomplishments, then it pops up a window mentioning our time taken to complete the game and asks us to enter our name and save. Orelse we can cancel it by using cancel option. We have Finish button to complete the game.

8.2 ANDROID APPLICATION

WELCOME PAGE

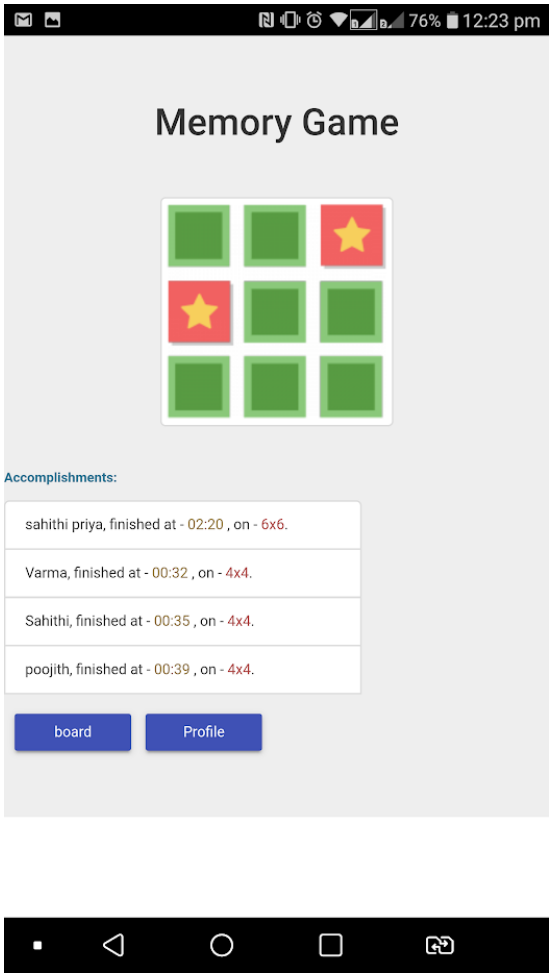

Fig. 8.2.1 Welcome Page

The home page consists of the game logo. It also consists of the top five accomplishments. These accomplishments are gathered from the various players games. Among them only top five are displayed over the homepage.

Here we also have other options like board and profile.

By clicking the board option, we can select the 4x4 board or 6x6 board. we can also use profile option to check out the profile of the user.

GAME PAGE SCREEN 1: Game Page for 4x4 board:

Fig. 8.2.2 Game Page Screen 1

It consists of the cards that are of the size 4x4. In 4x4 board, we have total 16 cards. We also have time duration that indicates the time that we have utilised in order to complete the game.

GAME PAGE SCREEN 2: Game Page for 6x6 board

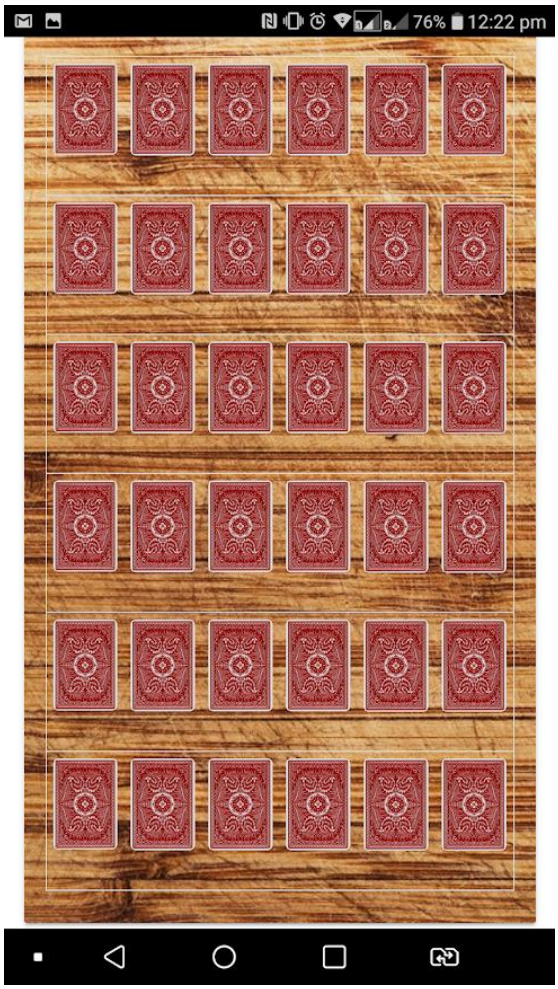

Fig. 8.2.3 Game Page Screen 2

It consists of the cards that are of the size 6x6. In the 6x6 board, we have total 36 cards. We also have time duration that indicates the time that we have utilised in order to complete the game.

GAME PAGE SCREEN 3: After completion of game

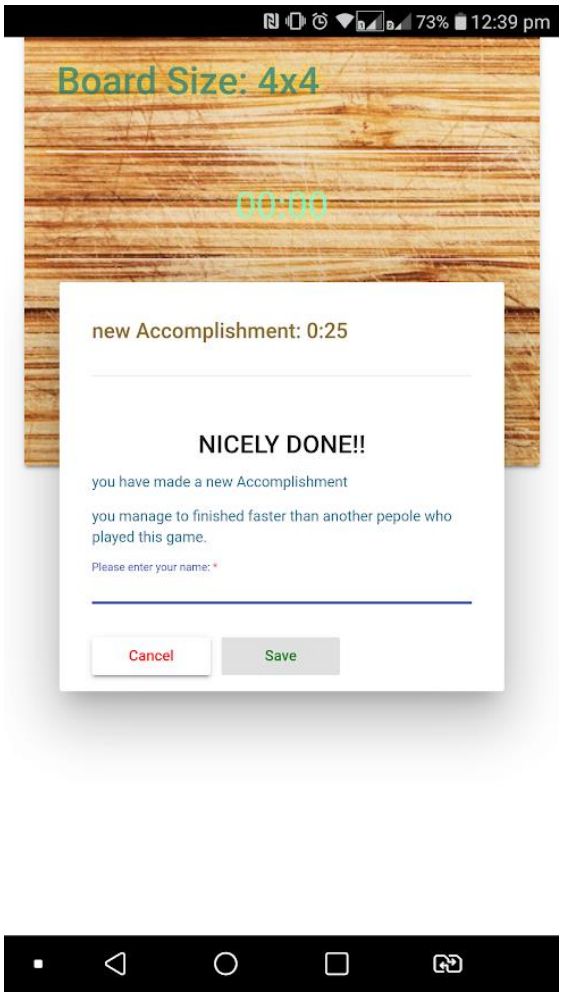

Fig. 8.2.4 Game Page Screen 3

Here in this page, if we have made one of the top five accomplishments, then it pops up a window mentioning our time taken to complete the game and asks us to enter our name and save. Orelse we can cancel it by using cancel option. We have Finish button to complete the game.

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENT

9.1 CONCLUSION

The Developed web page shows how is the working functionality of guessing game. The conclusion is to develop a game of memory. Match each square by color!we can click the squares to flip them. Score multipliers are applied if one or both of your matching squares were viewed for the first time. In the existing system,we have created the guessing game only. The Developed web page shows how is the working functionality of guessing game. The conclusion is to develop a game of memory. Match each square by logo! we can click the squares to flip them. Score multipliers are applied if one or both of your matching squares were viewed for the first time.

Technologies like Libraries of Npm & amp, Angular Cli, Yarn, AuthGuard, Angular material, rxjs, HttpClientModule, External API's, Chocolatey, Cordova, Cordova-android, ElectronJs create environments for executing the application in different platforms and also the modules of angular help in segregating the each method for execution.

9.2 IMPORTANCE OF RESEARCH

Based on the features that these games are designed with, people can easily learn faster without any stress attached. For instance, matching games have been found to be the most online console available for people. This type of game is simply designed to help people match similar images with respect to color, size, and structure. Since children can easily learn from things they see, the matching games will be of a great source of boosting their brain functionality effectively.

Few technologies are used in project, among them Angular and Node is very important as final application should run on this platform. We have spent many days learning Angular and Node as it was a new technology. Our application will be successfully built on this as we were able to use many built-in features of angular.

9.3 FUTURE ENHANCEMENT

We have used few technologies in our project, among them Angular and Node is very important as our final application should run on this platform. Using these technologies a project can be developed on different platforms like web, desktop and mobile app. Our application will be successfully built on this as we were able to use many built-in features of Angular.

REFERENCES:

[1]MindTactics: A Brain Computer Interface gaming platform, Kenneth Oum ; Hasan Ayaz ; Patricia A. Shewokis ; Paul Diefenbach 2010 2nd International IEEE Consumer Electronics Society's Games Innovations Conference.

[2]Evaluate children learning experience of multitouch flash memory game, Liew Tze Hui ; Lau Siong Hoe ; Hishamuddin Ismail ; Neo Han Foon ; Goh Kah Ong Michael 2014 4th World Congress on Information and Communication Technologies (WICT 2014).

[3]Visuospatial working memory game and measured memory performances at various ages, Takahiro Miura ; Ken-ichiro Yabu ; Kenichi Tanaka ; Kazutaka Ueda ; Tohru Ifukube 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC).

[4]Teaching game AI through Minicraft mods. Jessica D. Bayliss. 2012 IEEE International Games Innovation Conference.

[5]The development of 3d education game to maximize children memory. Dania Eridani ,Paulus Insap Santosa 2014 The 1st International Conference on Information Technology, Computer, and Electrical Engineering.

[6]Comparision of game experience and preferences between young and elderly. Jung-Ying Wang 2014 International Conference on Audio, Language and Image Processing.

[7]Equilibrium in repeated Stackelberg Public Goods game with two-leaders-one-follower and one-step-memory. Yifen, M. (2015). 2015 34th Chinese Control Conference (CCC).

[8]Taslihan Virtual Reconstruction- Interactive Digital Story or a Serious Game. Selma Rizvic, Irfan Prazina. 2015 7th International Conference on games and virtual words for serious applications(vs-games).

[9]Prototype of an educational game for teaching and learning in paged virtual memory. Darielson Araujo de Souza.2015 International Symposium on Computers in Education(SIE).

[10]An Agent-Based Game Platform for Exercising People Prospective Memory. Han Lin, Jinghua Hou. 2015 IEEE/WIC/ACM International Conference on web Intellience and Intelligent Agent Technology(WI-IAT). Volume-3.