

IB Computer Science IA :

Web Application Development Project

1. Summary & Evaluation
2. Criteria B: DESIGN
3. Criteria C: DEVELOPMENT

- Summary & Evaluation -

Summary

This document is broken down into two sections; Design & Development. The Design section includes UI designs of the webpages, structure of database, as well as algorithms that I have created for the app. The Development section shows a detailed documentation of the code that I have written, categorised by techniques.

Self- Evaluation

The screen and algorithm designs in section 1 are highly original, highlighting my creativity skills. They also include detailed annotations which makes them easy to understand and follow. The programming techniques shown in section 2 are very advanced, as it includes complex algorithms and advanced technologies.

Teacher Evaluation

Text copied from email attached on next page :

Through her IA that spanned for about 5 - 6 months, Marie has worked extremely hard to understand her client's requirements and design an aesthetically pleasing website based on her client's requirements. Throughout every stage of the product life cycle, she has been able to provide well-detailed documentation explaining complex techniques used in a clear and succinct manner. Marie has also made certain to test against her client's requirement specifications to gather feedback for the product from her client. Overall, Marie has done a good job with her course work and what I find impressive most is her ability to grasp, pick up a new web technology, despite having no prior knowledge in order to build a suitable product based on client needs.



Manoj Alex Varghese

to me ▾

7:45 AM (1 hour ago)



Good morning Marie,

Here's my comments about your IA:

Through her IA that spanned for about 5 - 6 months, Marie has worked extremely hard to understand her client's requirements and design an aesthetically pleasing website based on her client's requirements. Throughout every stage of the product life cycle, she has been able to provide well-detailed documentation explaining complex techniques used in a clear and succinct manner. Marie has also made certain to test against her client's requirement specifications gather feedback for the product from her client. Overall, Marie has done a good job with her course work and what I find impressive most, is her ability to grasp, pick up a new web technology, despite having no prior knowledge in order to build a suitable product based on client needs.

Hope this helps :)

Best,

Mr V

M. Alex Varghese

Teacher of Computer Science

Computer Science | UWC South East Asia



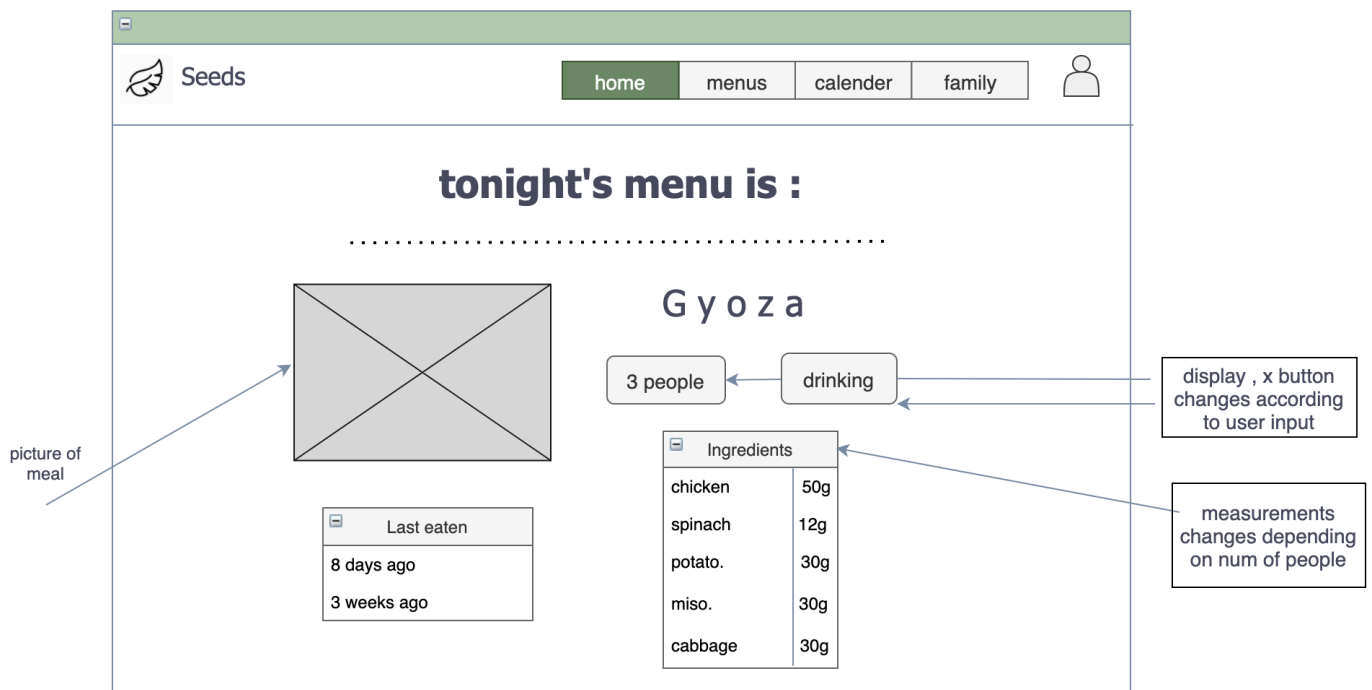
Criterion B: Design

Table of Contents

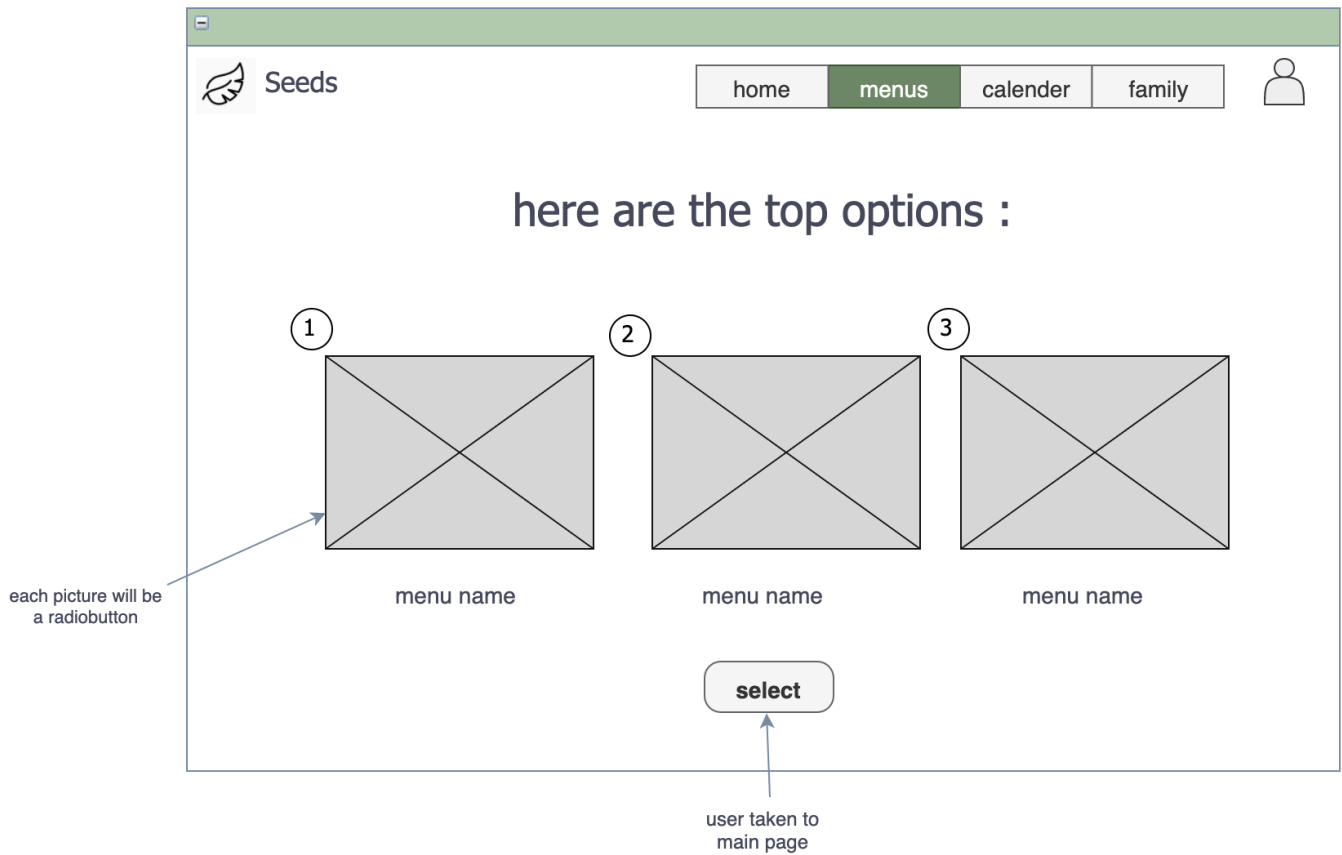
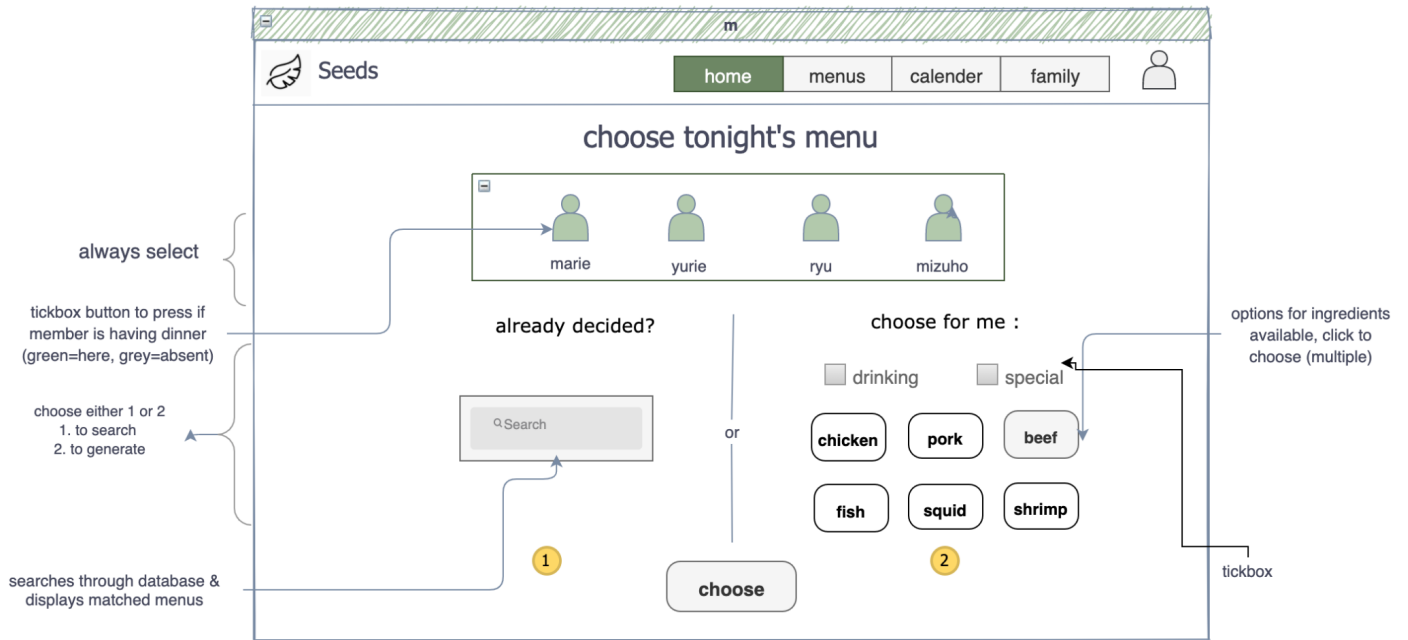
Screen Designs	2
ER Diagram	7
System Flowcharts	8

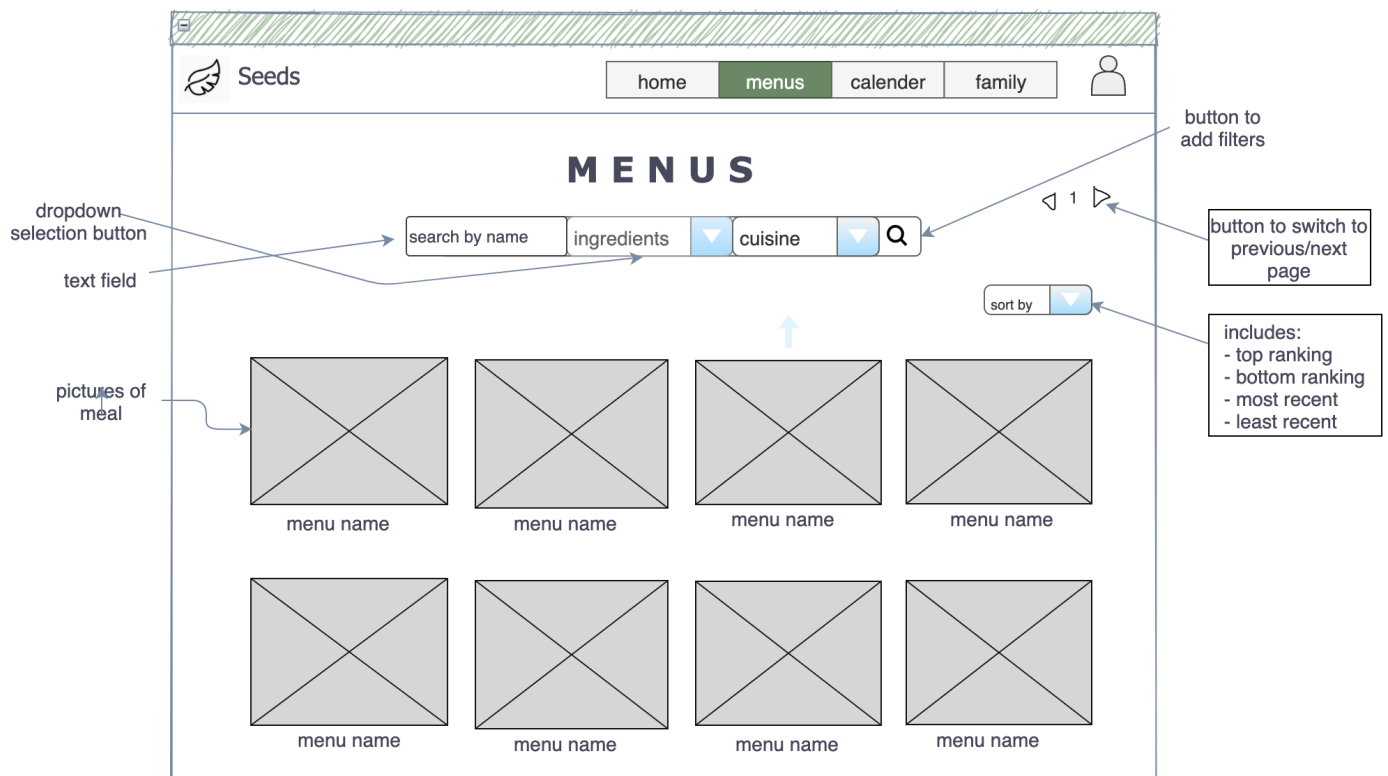
Screen Designs

Main menu: if menu is chosen

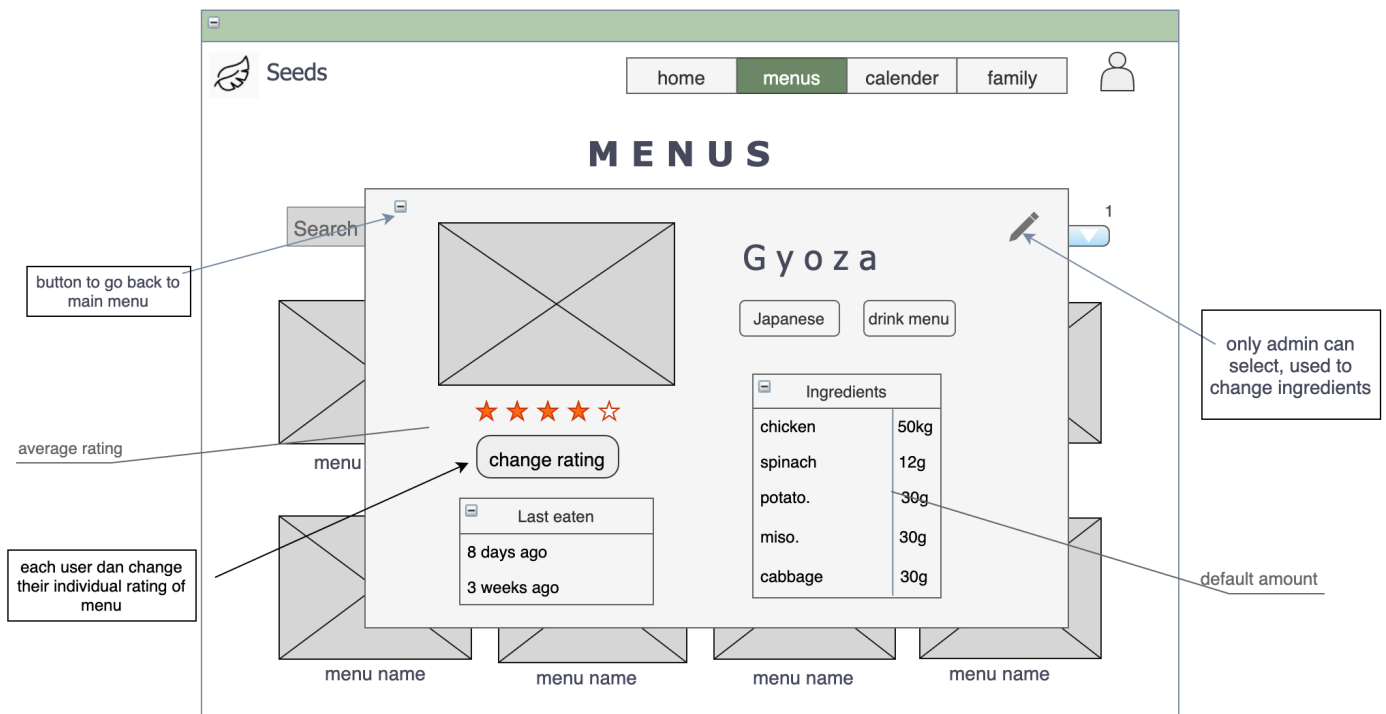


Main menu: if menu is not chosen

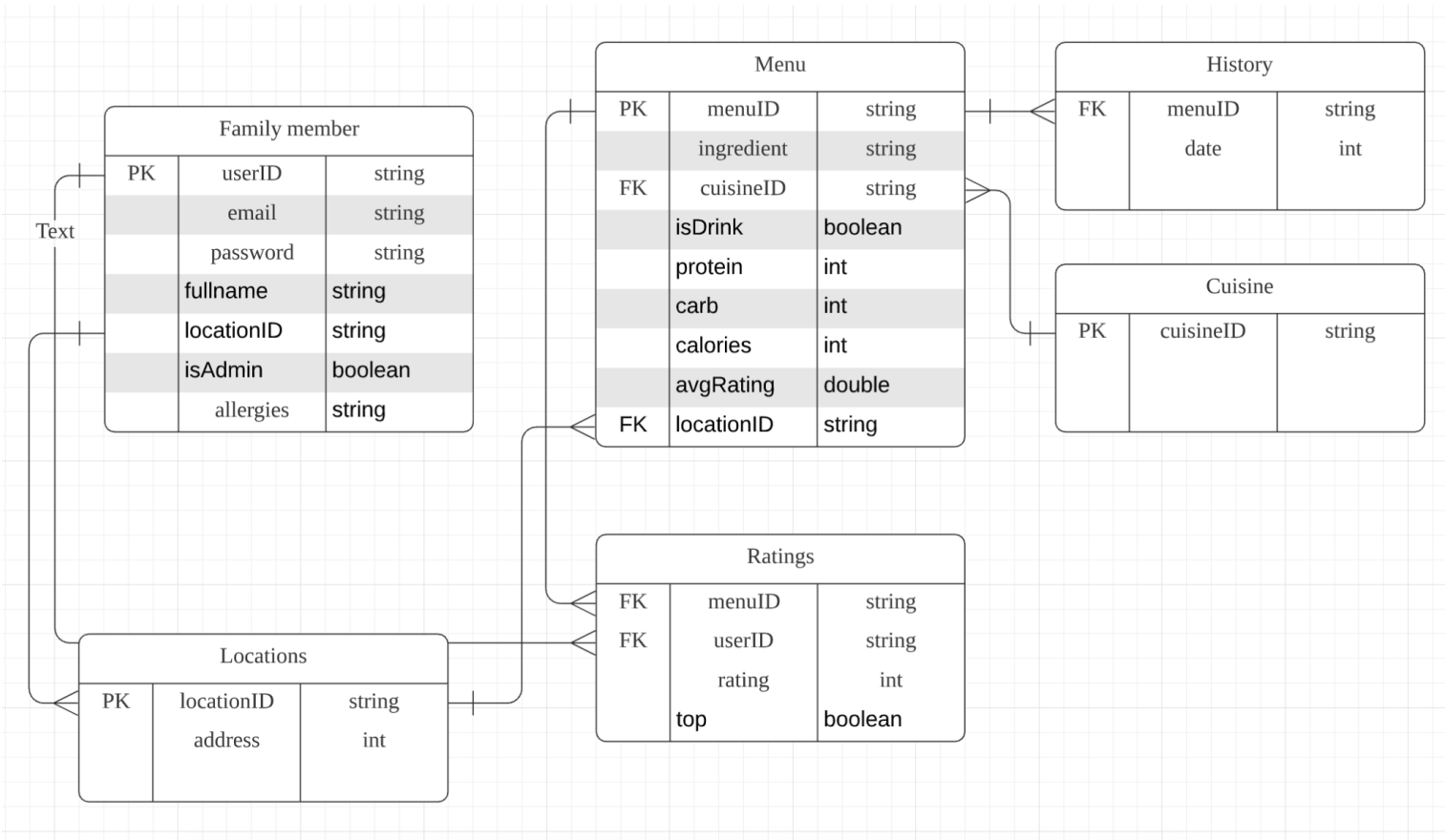




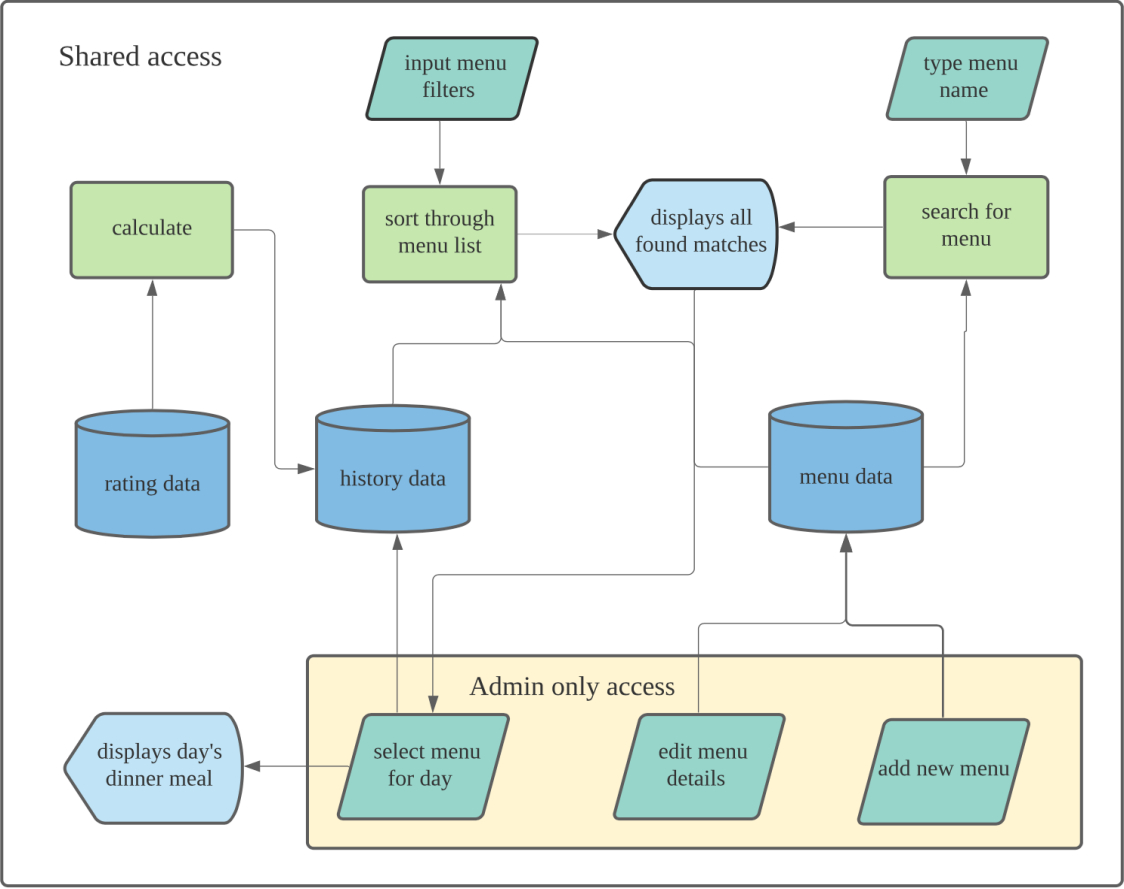
pop-up menu when a meal is clicked



ER Diagram



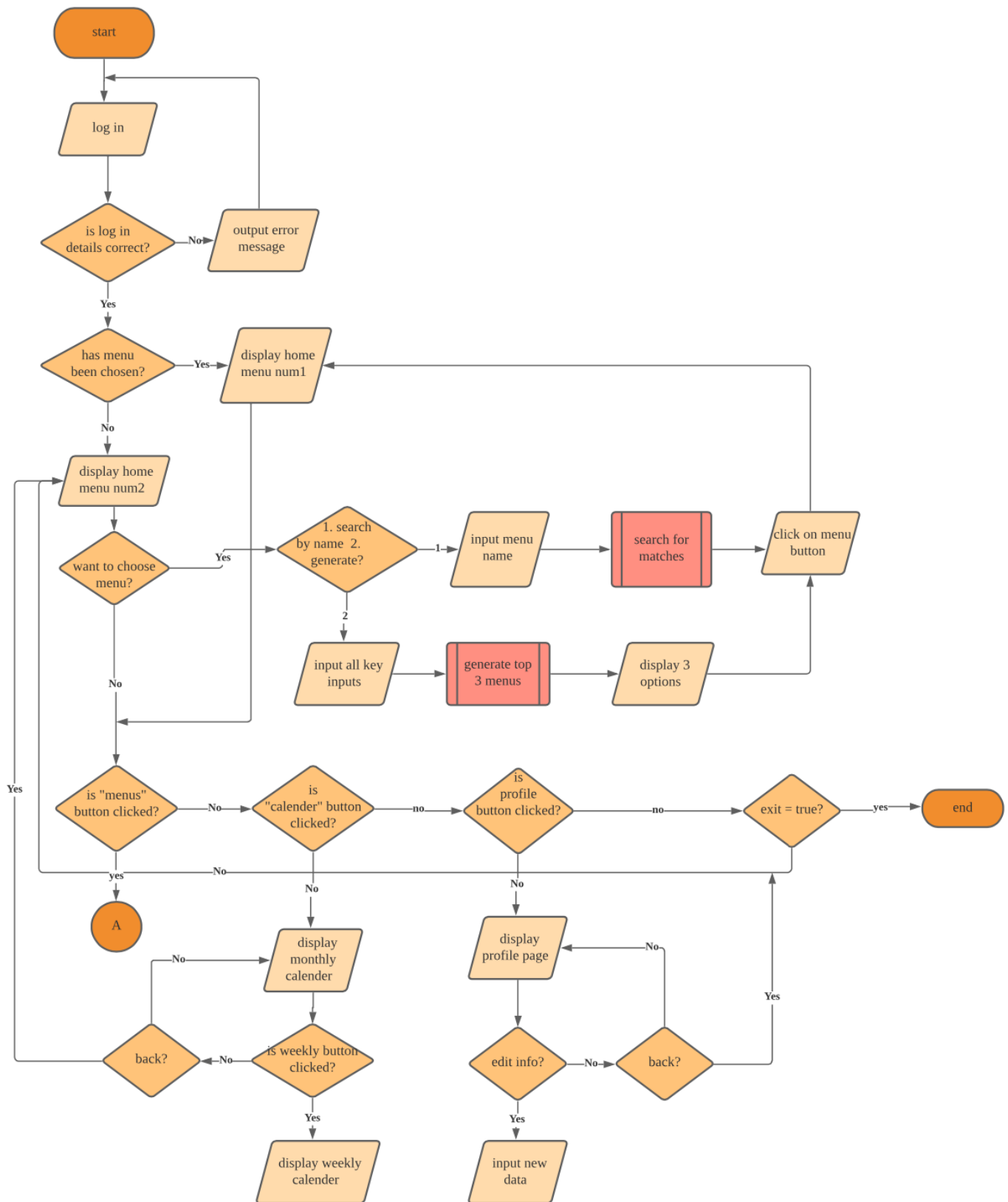
System Flowchart



Program flowcharts & Pseudocode

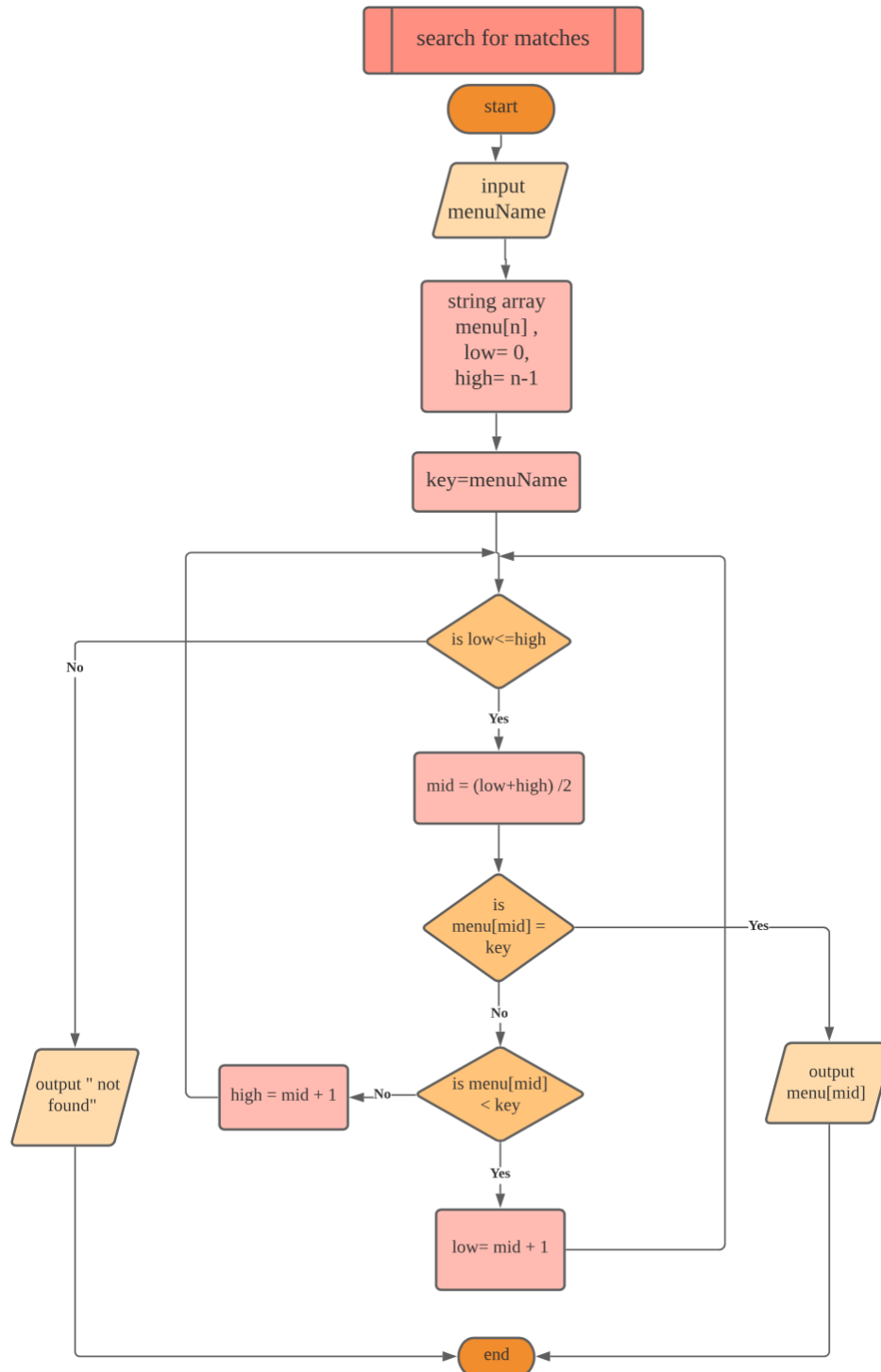
Menu Navigation Flowchart

Steps taken by user after log-in

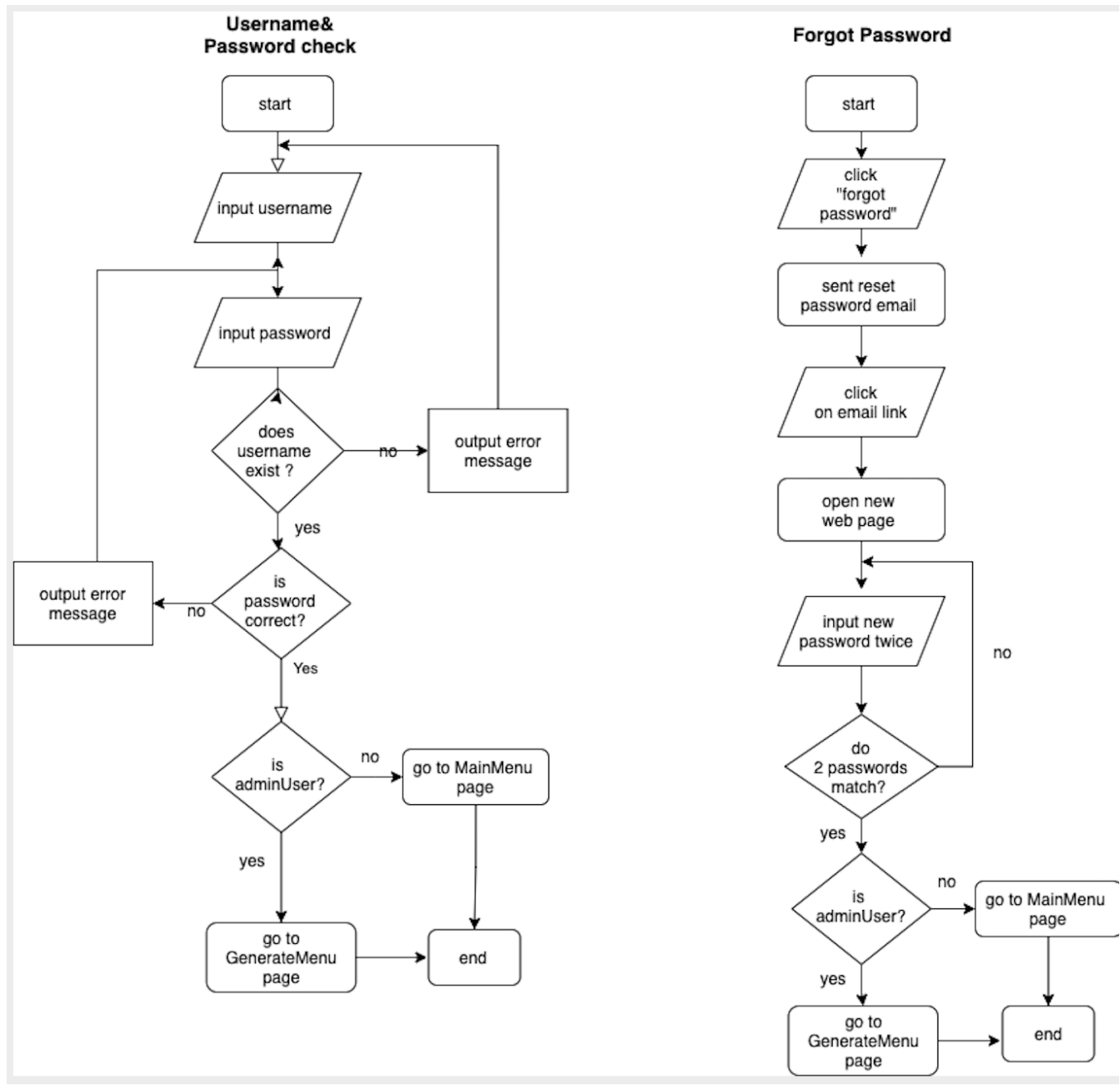


Search menu Flowchart

Binary Search Algorithm used when searching for menus



Log-in Flowchart



GenerateMenu pseudocode

Algorithm created by me to generate a menu

```
DECLARE inputIng : ARRAY of string
DECLARE inputPeople : ARRAY of string
DECLARE choices : COLLECTION of string
DECLARE finalThree : ARRAY of string
DECLARE sortedRating: ARRAY of int
DECLARE rating
DECLARE temp : int
DECLARE num : int
DECLARE sum : int
DECLARE average : decimal
```

function **generateMenu (**

 INPUT inputIng

FOR i = 0 to menuArray.length

 A = menuArray[i]

 FOR i = 0 to inputIng.length

 IF(a.ingredient = inputIng[i])

 IF (day = sunday)

 IF (a.type = "don" AND a.drink = isDrinking)

 choices.push(a)

 IF (day = friday / saturday)

 IF (a.type = "special" AND a.drink = isDrinking)

 choices.push(a)

 ELSEIF (a.drink = isDrinking)

 choices.push(a)

 ENDIF

```
ENDIF
END
END
calculateRating(choices)
sortedRating = selectionSort(rating)
FOR i = 0 to 3
    finalThree.push( sortedRating[i] )
END
OUTPUT finalThree
```

```
function calculateRating(arr)
    INPUT inputPeople
    n = arr.length
    FOR i = 0 to n
        rating.push(arr[i])
        FOR i = 0 to inputPeople.length
            Temp = Rate data of choices[i].inputPeople[i]
            Sum = sum + temp
            Num = num + 1
        Avg = sum / num
        rating.push(avg)
    Reset sum , num
Return rating
```

Test Plan

screen	what to test	test data/ inputs	expected outcome
log-in	username presence check	"Marie"	checks if password is correct
		" "	outputs error message
		8888	outputs error message
	password correct check	"Marieopi"	allows access to next page
		" "	outputs error message
		89812	outputs error message
home page no.2	click on multiple icons of members	click on all	tick marks appear on icons clicked
		click on none	outputs "choose members" when choose button is clicked
	menu name typed is existing in database / correct format / correct spelling	"ahijo"	display matched menu (s)
		8888	"no results"
		"pancakes"	
	allows only input of menu name or filters	type in search box	filter section unable to click
	clicking "choose button" links to corresponding page	typed in search box	jump to <i>home page no.1</i>
		chosen filters	jump to <i>choosing from 3 menus</i>
choosing from 3 menus	click on only one menu option	-	tick mark appear on menu clicked
	clicking on "select" jumps to home page no.1		
browsing menu-main	when menu image clicked, pop-up menu appears	click on menu image	jump to hyperlink of pop-up of correct menu
	change sorting method from 5 options	click on any buttons below "change sort"	page refreshes & displays new sorted items
	when filter (cuisine&	click on button	table list of all types of

	ingredients) drop down button pressed, display all options	next to "cuisine"	cuisines displayed
	menu display changes to new display when filter applied	click on "pork" under ingredients & "western" under cuisine , press go	display menus;
browsing menu-popup	go back to main browser page when cross button clicked	click on the button	back by one page, before menu pop-up
	correct data type when editing menu details (ingredients)	input "carrots"	allow user to click on confirm button
		input "4.092"	output "please enter a string"
	change rating	"3"	allow user to click on confirm button
		range check: "800"	"rate from 1 to 5"
		datatype check: "3.8"	"please enter an integer"
		"M"	

Criterion C: Development

Table of Contents

Data Structures	16
Cloud Database	17
Algorithms	24
User Interaction	27

Data Structures

Arrays : 1D

```
function generateMenu() {  
  let choices = []  
  let topThree = []  
  const today = new Date();  
  const day = today.getDay();  
  
  for (let i = 0; i < menuArray.length; i++) {  
    let a = menuArray[i]  
    for (let ii = 0; ii < inputIng.length; ii++) {  
      if (a.ingredient === inputIng[ii]) {  
        if (day == 0) {  
          if (a.type == "don" && a.isDrink === isDrinking) {  
            choices.push(a)  
          }  
        } else if (day == 1 || day == 2) {  
          if (a.type == "special" && a.isDrink === isDrinking) {  
            choices.push(a)  
          }  
        } else if (a.isDrink === isDrinking) {  
          choices.push(a)  
        }  
      }  
    }  
  }  
  calculateRating(choices)  
}
```

1D array to store menu options

inserting items from whole menuArray if conditions met

taken from GenerateMenu file

Arrays are used throughout the project to store data about menus. Above shows a 1d array, **choices**, being used to store the possible menu options, helping achieve success criteria 6.

Arrays : 2D

```
async function calculateRating(arr) {  
  let rating = []  
  let numPpl = 0  
  let sum = 0  
  let average = 0  
  let n = arr.length  
  for (let i = 0; i < n; i++) {  
    for (let j = 0; j < inputPeople.length; j++) {  
      database.ref("ratings/" + inputPeople[j] + "/" + arr[i])  
        .once('value')  
        .then((snapshot) => {  
          sum = sum + snapshot.val  
          numPpl = numPpl + 1  
        })  
    }  
    average = sum / numPpl  
    rating[i][0] = arr[i]  
    rating[i][1] = average  
    sum = 0  
    numPpl = 0  
  }  
  return rating;  
}
```

2D array to store String of menu choices & its average rating

inserting calculated values per row

taken from GenerateMenu file

Above shows a 2d array, **rating**, that is used when generating the menu. I have used this data structure because it allows the storage of two separate arrays in a way that stays linked after sorting. This helps achieve success criteria 6 & 7.

Cloud Database

I have used Firebase to give my web application the ability to store data persistently in the cloud. Cloud database allows me to scale my app dynamically.

Firestore configuration

```
<script>
  var firebaseConfig = {
    apiKey: "AIzaSyBgUXpyQXl08FHfazBIuyPaQMAvIiLMKds",
    authDomain: "seeds-69d3b.firebaseio.com",
    databaseURL:
      "https://seeds-69d3b-default-rtdb.asia-southeast1.firebaseio.com",
    projectId: "seeds-69d3b",
    storageBucket: "seeds-69d3b.appspot.com",
    messagingSenderId: "424175596523",
    appId: "1:424175596523:web:2c76bf56f465fd86465fae",
  };
  firebase.initializeApp(firebaseConfig);
</script>
```

taken from index.html file

Above shows the extract of the configuration used to connect the database to the web application in the IDE.

Write Methods

```
function writeUser(userid, email, isAdmin) {
  database.ref('users/' + userid).set({
    userid: userid,
    email: email,
    isAdmin: isAdmin,
  });
  console.log("Added user " + userid)
}

function writeMenu(menuName, ingredient, isDrink, cuisine) {
  database.ref('menu/' + menuName).set({
    menuid: menuName,
    menuName: menuName,
    ingredient: ingredient,
    isDrink: isDrink,
    cuisine: cuisine,
  });
  console.log("Added menu " + menuName)
}

function writeRating(userid, menuid, rating) {
  database.ref('rating/' + userid + "/" + menuid).set({
    rating: rating
  });
  console.log("Added rating " + userid + menuid + rating)
}
```

taken from dataBase.js file

The Firebase database consists of four data collections ;

1. User
2. Menu
3. Rating
4. History

The four Write functions above insert data into the database, where it takes in parameters and passes them into the attributes of each collection. I have chosen to log each time a data gets written, such as ;

```
console.log("Added rating " + userid + menuid + rating)
```

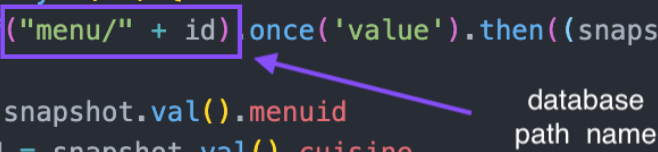
in order to check each time that the correct parameters have been inserted.

Read method

```
let menuid = localStorage.getItem("menubutton")
function readMenuById(id) {
  database.ref("menu/" + id).once('value').then((snapshot) => {
    menuN = snapshot.val().menuid
    cuisineN = snapshot.val().cuisine

    if (snapshot.val().isDrink) {
      drinkB = "drink"
    } else {
      drinkB = "not drink"
    }

    menuName.innerHTML = menuN
    cuisineDetail.innerHTML = cuisineN
    drinkDetail.innerHTML = drinkB
  })
}
readMenuById(menuid)
```



taken from BrowseMenu.js file

To print details of a menu on the browser, the Read function takes in an id of a menu and accesses the data at the given path. Each item at the path "menu/" holds 4 attributes , a) menu name b) cuisine c) ingredient d) isDrink , which are read and printed on the page using .innerHTML. This achieves Success Criteria 2.

Demo database

```
function resetDatabase() {
  database.ref().remove();
  console.log("reset database: success");
}

function buildDemoDatabase() {

  resetDatabase();

  writeUser("Marie", "marie@gmail.com", false);

  writeMenu("pizza", "pork", true, "special");

  writeRating("Marie", "pizza", 4);

  writeHistory("pizza", "2021-07-24", "Marie, Mizuho");
}
```

taken from dataBase.js file

To build a prototype of the web application, I have created a demo database using the previous Write functions . Above shows a snippet of the methods with each respective parameter inserted. I have done this to give consistent data for my web application during development. This step achieves Success Criteria 2,3 and 4. Additionally, the method resetDatabase() clears all data everytime the file is run, which avoids any risks of overridden data.

Snapshot processing

```
function snapshotToArray(snapshot) {  
  let arr = [];  
  snapshot.forEach(function (childSnapshot) {  
    let item = childSnapshot.val();  
    item.key = childSnapshot.key;  
    arr.push(item);  
  });  
  return arr;  
}  
  
function readMenu() {  
  database.ref("menu/").once('value').then((snapshot) => {  
    let arr = snapshotToArray(snapshot);  
    for (let i = 0; i < arr.length; i++) {  
      menuArray.push(arr[i].menuName)  
      drinkArray.push(arr[i].isDrink)  
      ingArray.push(arr[i].ingredient)  
      cuisineArray.push(arr[i].cuisine)  
    }  
  })  
}  
readMenu();
```

taking snapshot of each data branch

moving database snapshot to array

taken from MenuDetail.js file

In order to efficiently process a list of data in the database, I created a helper function which takes any snapshot of the data that is accessed and returns an array. In the readMenu() function, the snapshot array is used to process the data.

Asynchronous coding

```
async function calculateRating() {
  let rating = []
  let avgRate = []
  let people = ["Marie", "Yurie", "Mizuho", "Ryu"]

  for (let menuid = 0; menuid < menuArray.length; menuid++) {
    for (let id = 0; id < 4; id++) {
      database.ref("ratings/" + people[id] + "/" + menuArray[menuid])
        .once('value')
        .then((snapshot) => {
          rating.push([menuid, snapshot.val()])
        })
    }
  }
  await sleep(2000);

  for (let i = 0; i < rating.length; i++) {
    for (let j = 0; j < rating[i].length; j++) {
      console.log(rating[j][1])
      let sum;
      sum = sum + rating[i][1]
    }
    sum = 0;
  }
  return avgRate;
}

function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

taken from GenerateMenu.js file

Asynchronous coding is the concept of running a task separately from the primary application thread, to avoid time delay in data processing. In the code snippet above, the data at the path "ratings/" in the database is accessed, and adds each element in an array as soon as it has retrieved the data. However, since the code below continues whilst the data is still getting transferred into the array, I have used a sleep function. This puts the primary program on pause for a given amount of time, and allows the asynchronous code to finish executing.

Algorithms

Filtering Algorithm

```
<div class="w3-row-padding w3-container w3-center">
  <div class="w3-col s4 w3-section w3-center"></div>
  <div class="w3-col s4 w3-section w3-center">
    <div>
      <h5>search by name</h5>
      <input
        id="menuInput"
        class="w3-input w3-border w3-round-large"
        type="text"
      />
    </div>
    <div>
      <h5>ingredients</h5>
      <select id="ingredientList" class="w3-select" name="option">
        <option value="" disabled selected></option>
        <option id="fish" value="1">fish</option>
        <option id="chicken" value="2">chicken</option>
        <option id="pork " value="3">pork</option>
        <option id="prawn" value="4">prawn</option>
        <option id="beef" value="5">beef</option>
      </select>
    </div>
  </div>
```

taken from BrowseMenu.html file

A key feature of this web application is the ability to search a menu by using filters. The filters include menu name, ingredients and cuisine. The above HTML code shows the input elements , including a text field and two drop down buttons.


```

function createMenu() {
  database.ref("menu/").once('value').then((snapshot) => {
    menuArray = []
    let arr = snapshotToArray(snapshot);
    let value1 = document.getElementById("menuInput").value
    let value2 = ingredientList.options[ingredientList.selectedIndex].text;
    let value3 = cuisineList.options[cuisineList.selectedIndex].text;

    for (let i = 0; i < arr.length; i++) {
      const filter1 = arr[i].menuName
      const filter2 = arr[i].ingredient
      const filter3 = arr[i].cuisine

      if (value1 === "" && value2 === "" && value3 === "") {
        menuArray.push(arr[i].menuName)
      } else if (value1 === "" && value2 === "") {
        if (filter3.includes(value3)) {
          menuArray.push(arr[i].menuName)
        }
      } else if (value1 === "" && value3 === "") {
        if (filter2.includes(value2)) {
          menuArray.push(arr[i].menuName)
        }
      } else if (value2 === "" && value3 === "") {
        if (filter1.includes(value1)) {
          menuArray.push(arr[i].menuName)
        }
      } else if (value1 === "") {
        if (filter2.includes(value2) && filter3.includes(value3)) {
          menuArray.push(arr[i].menuName)
        }
      } else if (value2 === "") {
        if (filter1.includes(value1) && filter3.includes(value3)) {
          menuArray.push(arr[i].menuName)
        }
      }
    }
  })
}

```

value1 : user input in search box
value 2 & 3 : user choice in dropdown menu

holds menu data from database

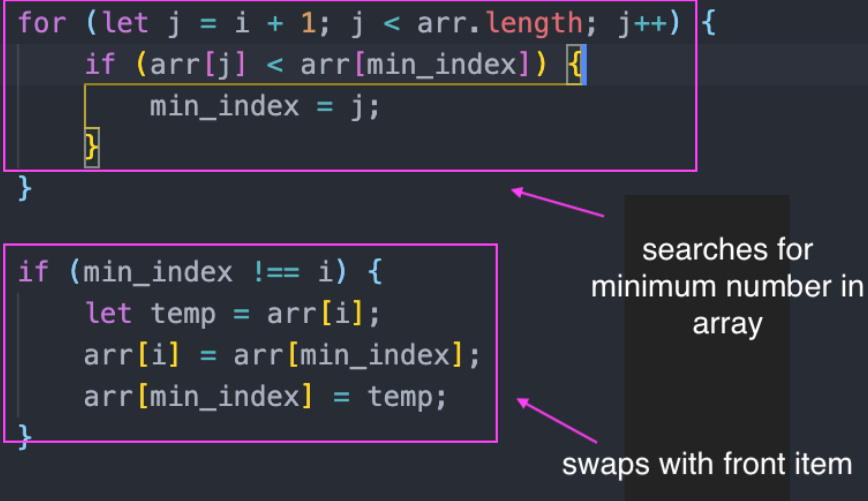
creates array based on whether user input & database matches

taken from BrowseMenu.js file

Above shows several two-layered conditional statements that filter menu options. The outer layer checks the presence of inputs in the three input elements, and the inner layer checks whether they match with the data. This achieves Success Criteria 5.

Selection Sort

```
function selectionSort(arr) {  
    for (let i = 0; i < arr.length - 1; i++) {  
        let min_index = i;  
        for (let j = i + 1; j < arr.length; j++) {  
            if (arr[j] < arr[min_index]) {  
                min_index = j;  
            }  
        }  
  
        if (min_index !== i) {  
            let temp = arr[i];  
            arr[i] = arr[min_index];  
            arr[min_index] = temp;  
        }  
    }  
    return arr;  
}
```



The diagram illustrates the Selection Sort algorithm with two key steps highlighted by pink boxes and arrows:

- searches for minimum number in array**: This annotation points to the inner loop (the second for loop) which iterates from `i + 1` to the end of the array, comparing `arr[j]` with `arr[min_index]` to find the smallest element.
- swaps with front item**: This annotation points to the swap logic (the if statement) which checks if the found minimum index (`min_index`) is different from the current position (`i`). If so, it swaps the element at `i` with the element at `min_index` using a temporary variable `temp`.

taken from GenerateMenu.js file

The selection sort algorithm sorts and displays the menu array in descending and ascending order in terms of name or rating.

RECURSION : Binary Search

```
function searchBinary(arr, x, start, end) {
  if (start > end) return false; ← base case
  let mid = Math.floor((start + end) / 2);
  if (arr[mid] === x) return true;
  if (arr[mid] > x)
    return searchBinary(arr, x, start, mid - 1);
  else
    return searchBinary(arr, x, mid + 1, end); ← repeats code with altered parameters
}

searchButton.addEventListener("click", () => {
  searchBinary(menuArray, menuSearch.val, 0, menuArray.length-1)
})
```

taken from GenerateMenu file

I have chosen a recursive approach as it reduces length and repetition of code. The value at x is compared with the mid value of the array and unless equal, calls itself with different parameters. The base case is given at the beginning of the method. This method is used to search for a menu when choosing at the GenerateMenu page, achieving success criteria 6.

User Interaction

Event handling

```
calculateRating()

alph.addEventListener("click", () => {
  menuArray = selectionSort(menuArray)
})

highRate.addEventListener("click", () => {
  menuArray = selectionSortD(rating)
})

lowRate.addEventListener("click", () => {
  menuArray = selectionSortA(rating)
})

createButton(menuArray)
})
```

← sorts menu by rating when "highRate" clicked

taken from dataBase.js file

To make a HTML button interactive, an event handler is attached to the button which executes a function when clicked. By adding **.addEventListener** to each sorting button, the displayed button on screen gets sorted in their respective ways.

Window Local Storage

```
function createButton(arr) {
  let html = ""
  for (let i = 0; i < arr.length; i++) {
    html += "<button class='w3-button' id='button' + i + '>' + arr[i] + "</button>"
  }
  menuGallery.innerHTML = html;

  for (let i = 0; i < arr.length; i++) {
    let button = document.getElementById('button' + i)
    button.addEventListener('click', () => {
      localStorage.setItem("menubutton", arr[i])
      window.location.href = "file:///Users/mariemuramatsu/Desktop/IA/CS%20ai-%20seeds/firebase"
    })
  }
}
```

saves name of button clicked onto local storage

jumps to menuDetail page

taken from BrowseMenu.js file

The localStorage property in js allows the key/value pairs to be stored in memory, facilitating and convenient data exchange in various parts of the application.

```
let menuid = localStorage.getItem("menubutton")
readMenuById(menuid)
readRatingById("Mizuho", menuid)
```

retrieves menu name saved in local storage

taken from MenuDetail.js file

Input validation

```
editRatingCheck.addEventListener("click", () => {  
    let x = document.getElementById("editRating").value;  
    let message;  
    if (isNaN(x)) {  
        message = "please enter a number";  
    } else if (x < 1 || x > 5) {  
        message = "please enter from 1 to 5";  
    } else {  
        message = "Input OK";  
        writeRating("Mizuho", menuId, x)  
        console.log(x)  
    }  
    document.getElementById("message").innerHTML = message;  
})
```

isNaN : check whether
input = number

range check

taken from MenuDetail.js file

When a user clicks on a menu button, a text field allows the user to input a new rating for the menu. The variable "x" holds the value that is inputted by the user, and two conditional statements are used to check whether it is valid;

1. data type check
2. range check

If user input is valid, writeRating rewrites the rating data in the database. This achieves Success Criteria 3.