Evidence Gathering Document for SQA Level 8 Professional Developer Award.

This document is designed for you to present your screenshots and diagrams relevant to the PDA and to also give a short description of what you are showing to clarify understanding for the assessor.

Fill in each point with screenshot or diagram and description of what you are showing.

Each point requires details that cover each element of the Assessment Criteria, along with a brief description of the kind of things you should be showing.

Week 1

Unit	Ref	Evidence
I&T	I.T.6	Demonstrate the use of a hash in a program. Take screenshots of: *A hash in a program *A function that uses the hash *The result of the function running

Description: Here is a hash called “friend1” with a collection of key-value paired attributes (name, age, job etc). The second screenshot shows the “get_name” function taking in an object as a parameter and returning the hash value for the “name” key. The third screenshot shows a test for the “get_name” function, passing the friend hash as an argument. The last screenshot shows the test passing.

```
@friend1 = {
  name: "Markus",
  age: 22,
  job: "graphic designer",
  friends: ["Julie","Pete","Maria"],
  favourites: {
    tv_show: "Buffy the Vampire Slayer",
    food: "pizza"
  }
}
```

```
def get_name(person)
  return person[:name]
end
```

```
def test_get_friend_name
  result = get_name(@friend1)
  assert_equal("Markus", result)
end
```

```
➜  specs ruby friends_spec.rb
Run options: --seed 16489

# Running:

.

Finished in 0.000736s, 1358.6957 runs/s, 1358.6957 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
➜  specs
```

Week 2

Unit	Ref	Evidence
I&T	I.T.5	Demonstrate the use of an array in a program. Take screenshots of: *An array in a program *A function that uses the array *The result of the function running

Description: Here is an array of hashes called “people”. This contains 3 objects, each representing a person, with a set of 3 attributes.
The function “total_cash” takes in the array of people to get the total count of the people’s money. First, a variable called “total_money” is set to serve as a counter (starting at 0). A for loop searches through the array and at each loop adds up to total_money each person’s value for the key “cash”. The function returns the value for the total money.
The last screenshot shows the result of the function’s test running.

```ruby
@people = [
  {
    name: "Chris",
    age: 26,
    cash: 65
  },
  {
    name: "Liliana",
    age: 15,
    cash: 100
  },
  {
    name: "Shona",
    age: 18,
    cash: 110
  }
]
```

```ruby
def total_cash(people)
  total_money = 0
  for person in people
    total_money += person[:cash]
  end
  return total_money
end
```

```ruby
  def test_get_total_cash
    assert_equal(275, total_cash(@people))
  end
```

```
➜  specs ruby people_array.rb
Run options: --seed 41972

# Running:

.

Finished in 0.001128s, 886.5248 runs/s, 886.5248 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
➜  specs
```

Week 3

Unit	Ref	Evidence
I&T	I.T.3	Demonstrate searching data in a program. Take screenshots of: *Function that searches data *The result of the function running

Description: The first screenshot shows a function called “customers”. This will return all the customers that have brought tickets for the film whose id is being passed as an argument. The function uses an SQL query (with an inner join) to search for rows in a database in which the id column on the Customer class matches the customer_id column on the Ticket class. An array of matching Customer objects is returned.
The second screenshot shows the result of the SQL query in action on Postico, returning all the customers who have booked for the movie with an id of 3.

```ruby
def customers()
  sql = "SELECT customers.*
  FROM customers
  INNER JOIN tickets
  ON customers.id = tickets.customer_id
  WHERE film_id = $1"
  values = [@id]
  customer_data = SqlRunner.run(sql, values)
  return Film.map_items(customer_data)
end
```

```sql
SELECT customers.*
    FROM customers
    INNER JOIN tickets
    ON customers.id = tickets.customer_id
    WHERE film_id = 3
```

Load Query... Save Query...

id	name	funds
5	Davide	20
6	Marianne	40
7	Jean	66

Unit	Ref	Evidence
I&T	I.T.4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running

Description: The first screenshot shows a class method for Customer that uses an SQL query to return a list of all the customers, and sorts this list by funds.
The second screenshot shows the SQL query in action on Postico, returning all the customers from poorer to richer.

```
def self.all()
 sql = "SELECT * FROM customers ORDER BY funds"
 customer_data = SqlRunner.run(sql)
 return Customer.map_items(customer_data)
end
```

```
1 SELECT * FROM customers ORDER BY funds
```

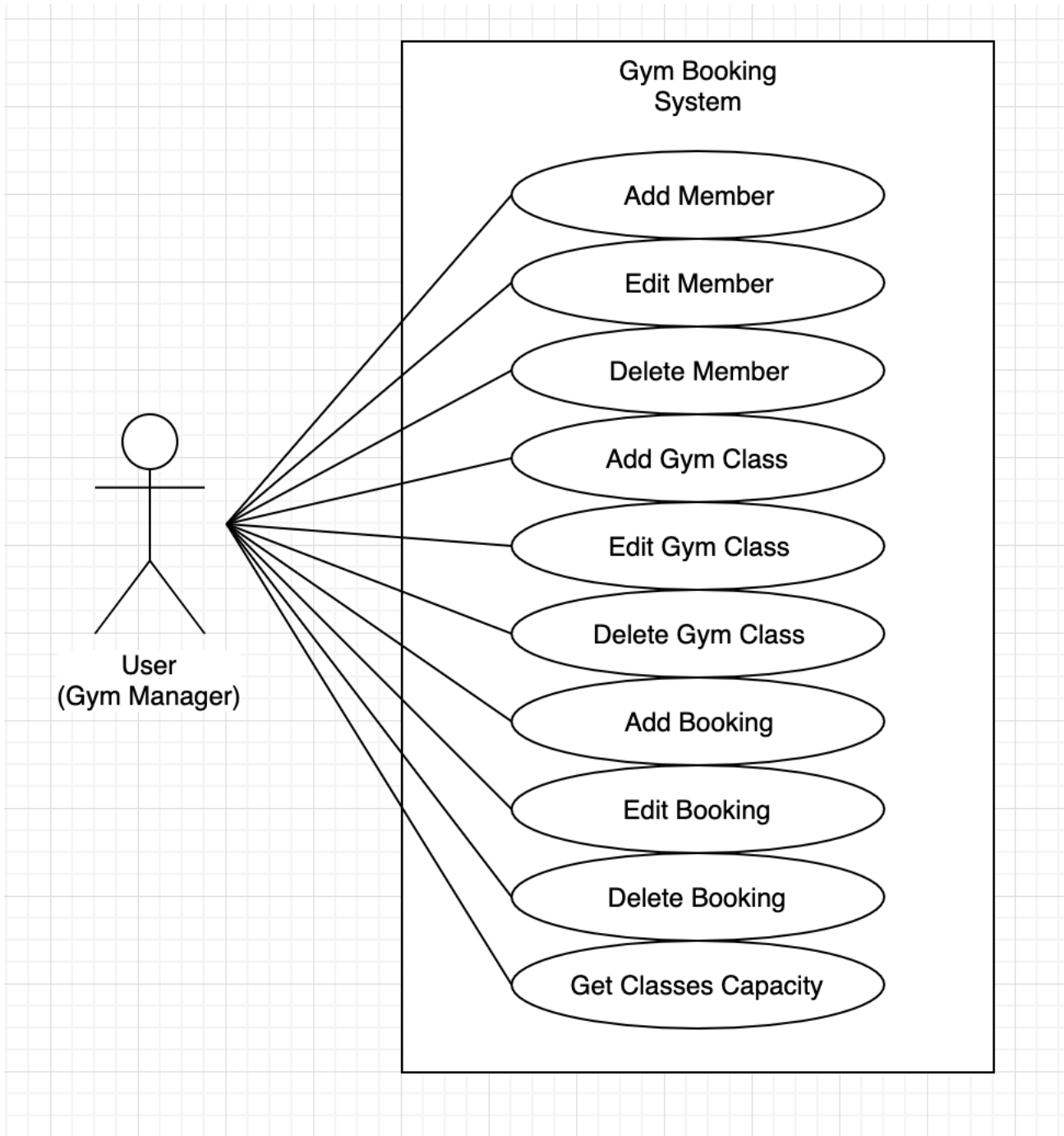
Load Query... Save Query...

id	name	funds
8	Davide	20
12	Monique	38
9	Marianne	40
11	Melissa	55
10	Jean	66
15	Jean	66
13	Mariusz	88
14	Patrick	120

Week 4

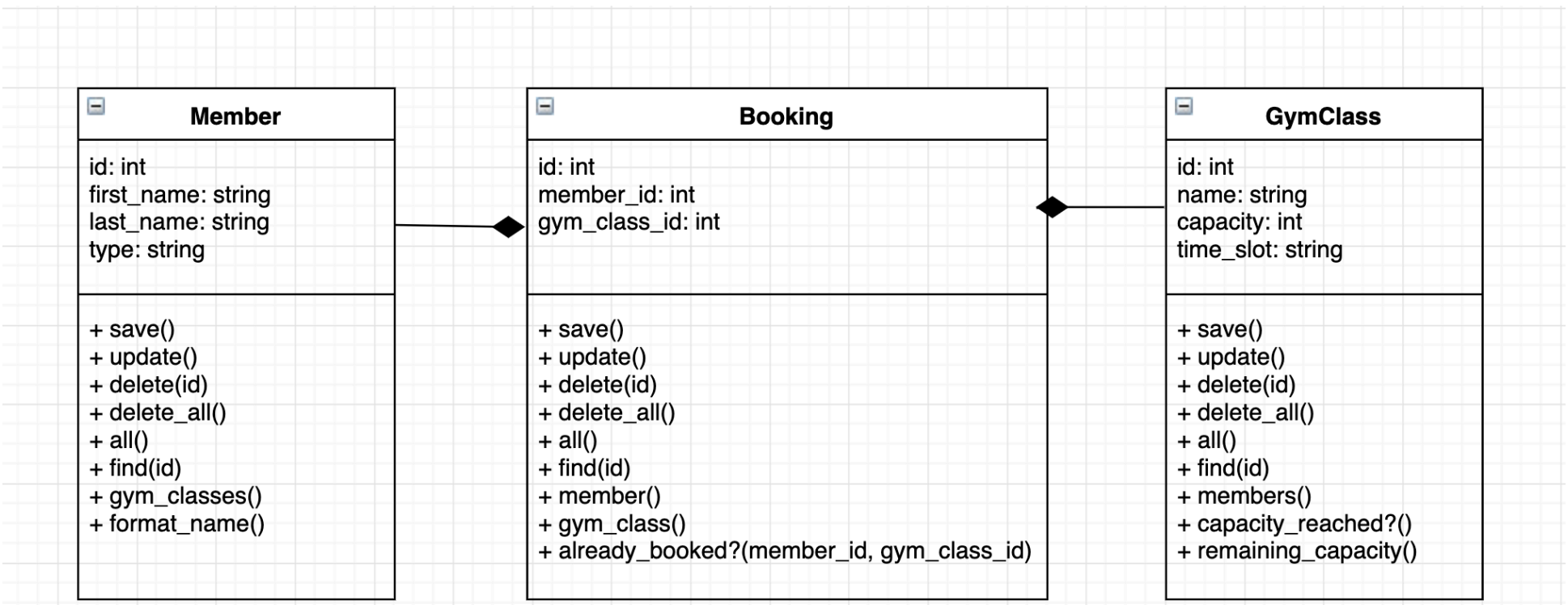
Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram

The diagram shows the use case for the Gym Manager of a Gym Booking System application



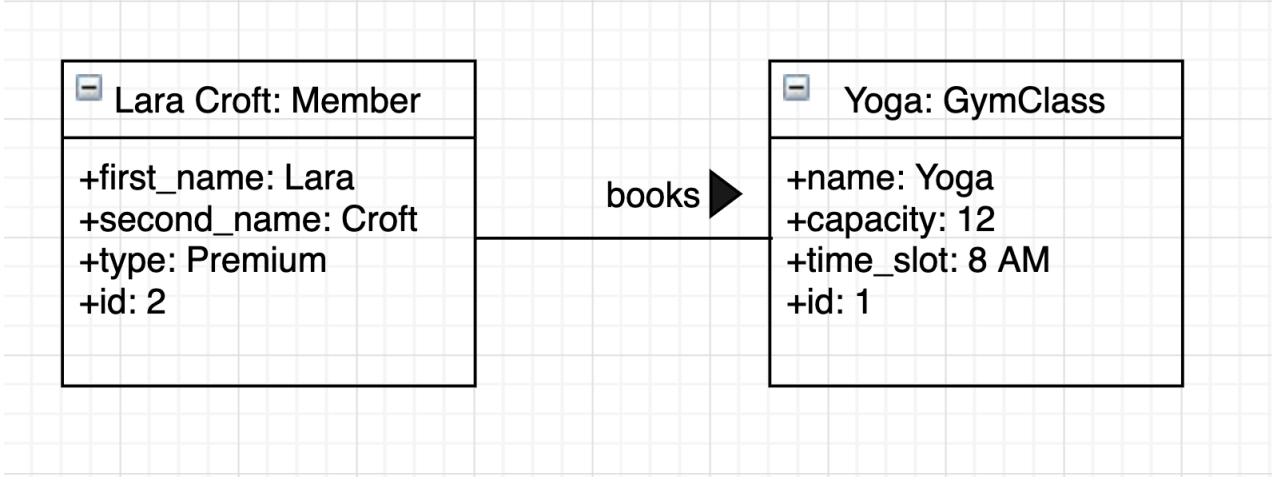
Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram

The diagram shows 3 classes (Member, Booking and GymClass), their attributes and methods.



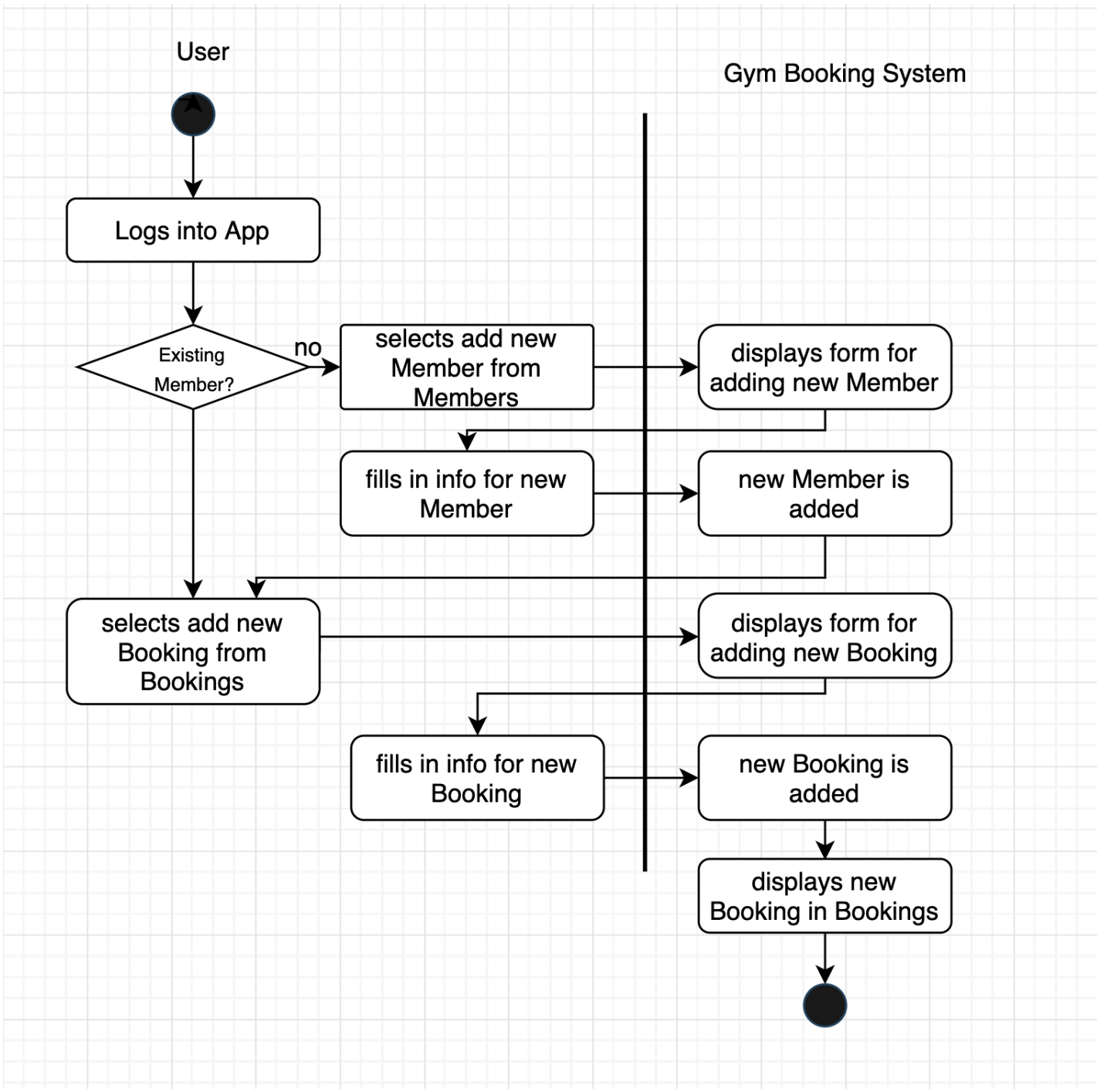
Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram

The diagram shows 2 objects and their attributes.



Unit	Ref	Evidence
A&D	A.D.4	An Activity Diagram

The diagram shows the user journey to add a new booking.
First, if it doesn't exist, a new member is added from the members menu, by filling in the relative form.
Then, the user adds a new booking from the bookings menu, by adding the relevant information.
The new booking is finally displayed in Bookings.

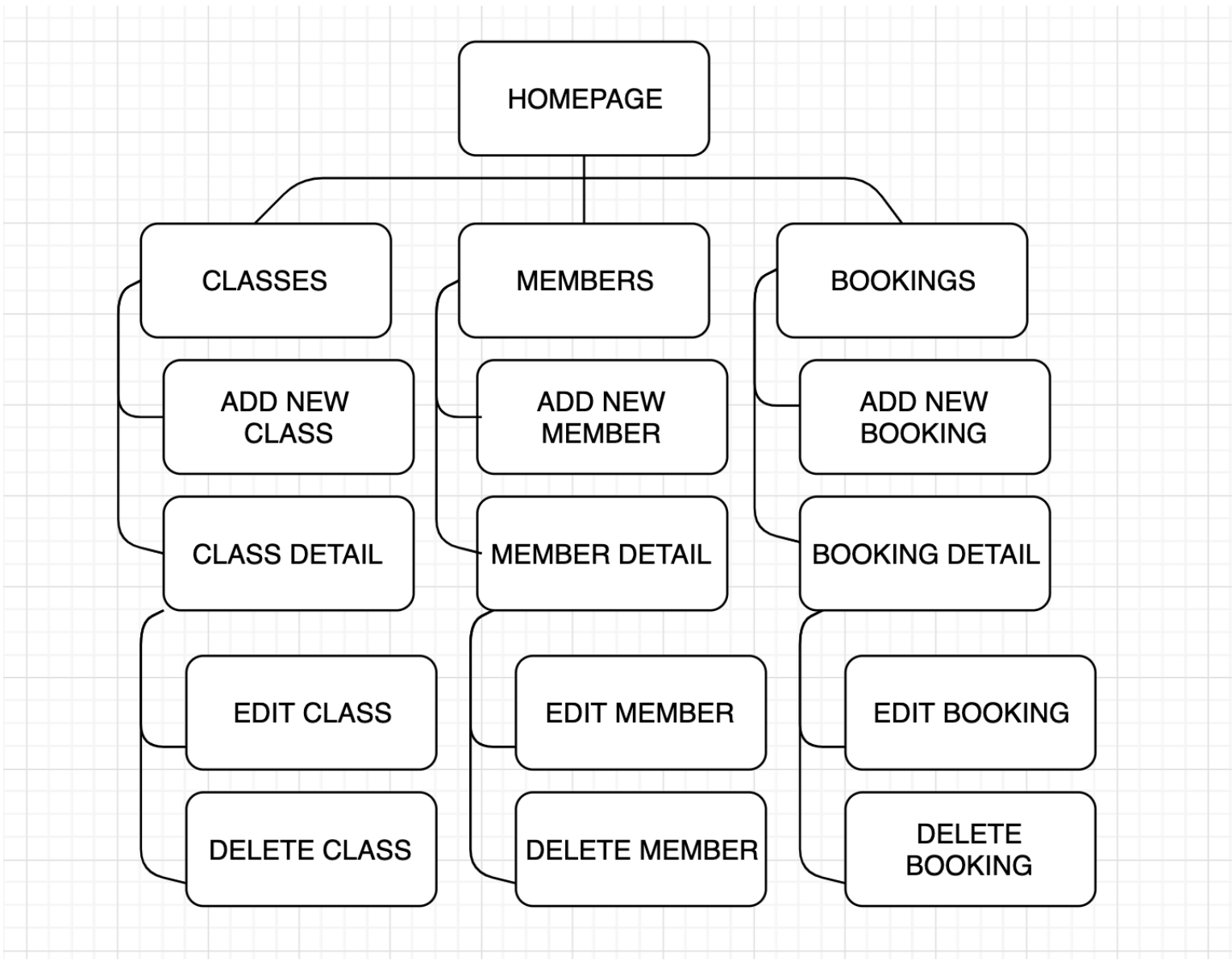


Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time

Constraint Category	Implementation Constraint	Solution
Hardware and software platforms	Built and tested on Apple laptop. Requires Ruby and Sinatra to work.	Make sure that the required technology is installed and up to date. The app should be tested on different platforms and browsers.
Performance requirements	Potentially loading a vast amount of data for members and bookings, the app could be slow when displaying the full list.	Requires good wifi connection, a search bar could be added to filter the data.
Persistent storage and transactions	Requires PSQL and a database to store data.	Create appropriately named tables in POSTgres database for different classes.
Usability	The Ruby file needs to run in order for the application to be used.	The app could be made server less and able to operate entirely in the cloud.
Budgets	N/A	Personal project completed as part of a web development course.
Time	Deadline set 6 days from the start	Prioritise MVP and build smaller features one at the time.

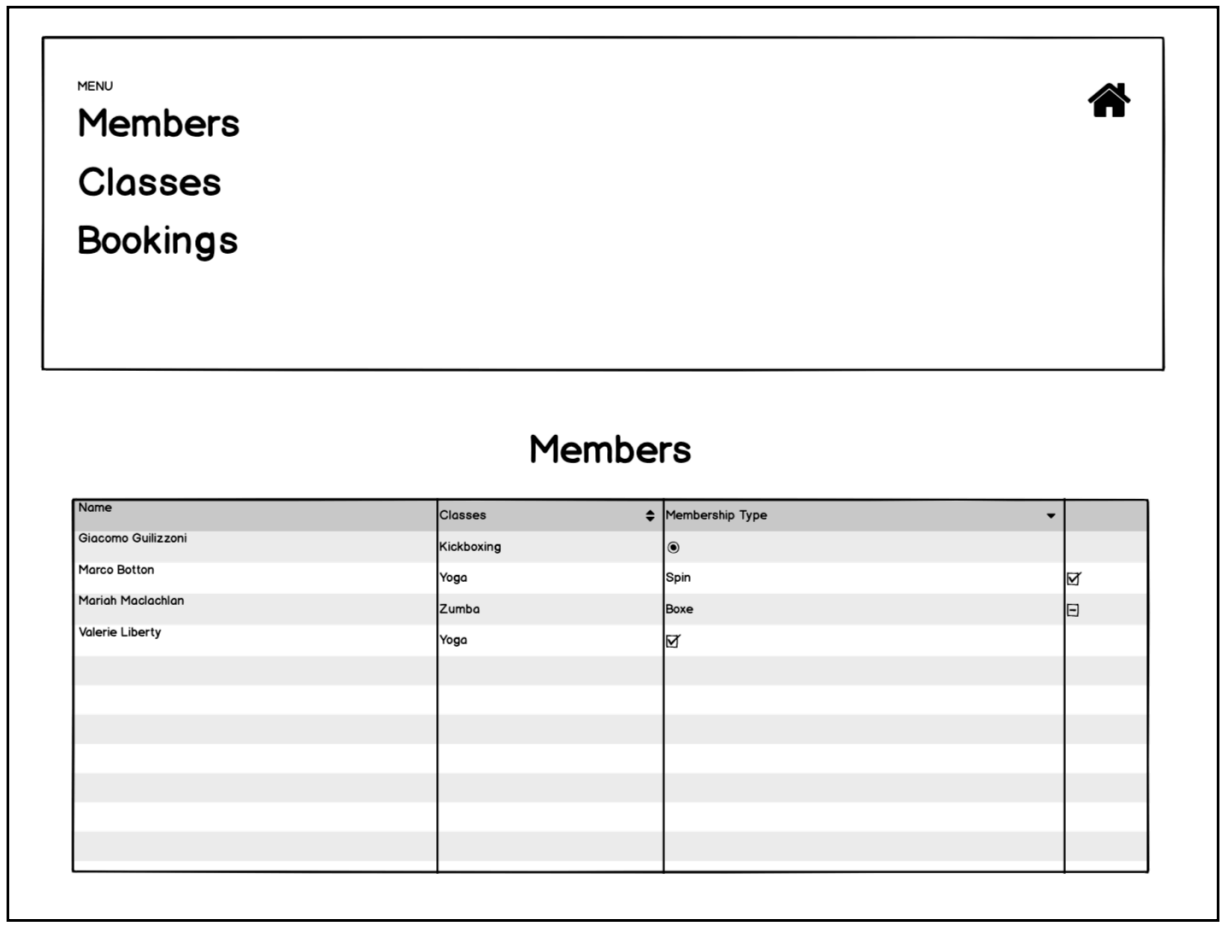
Unit	Ref	Evidence
P	P.5	User Site Map

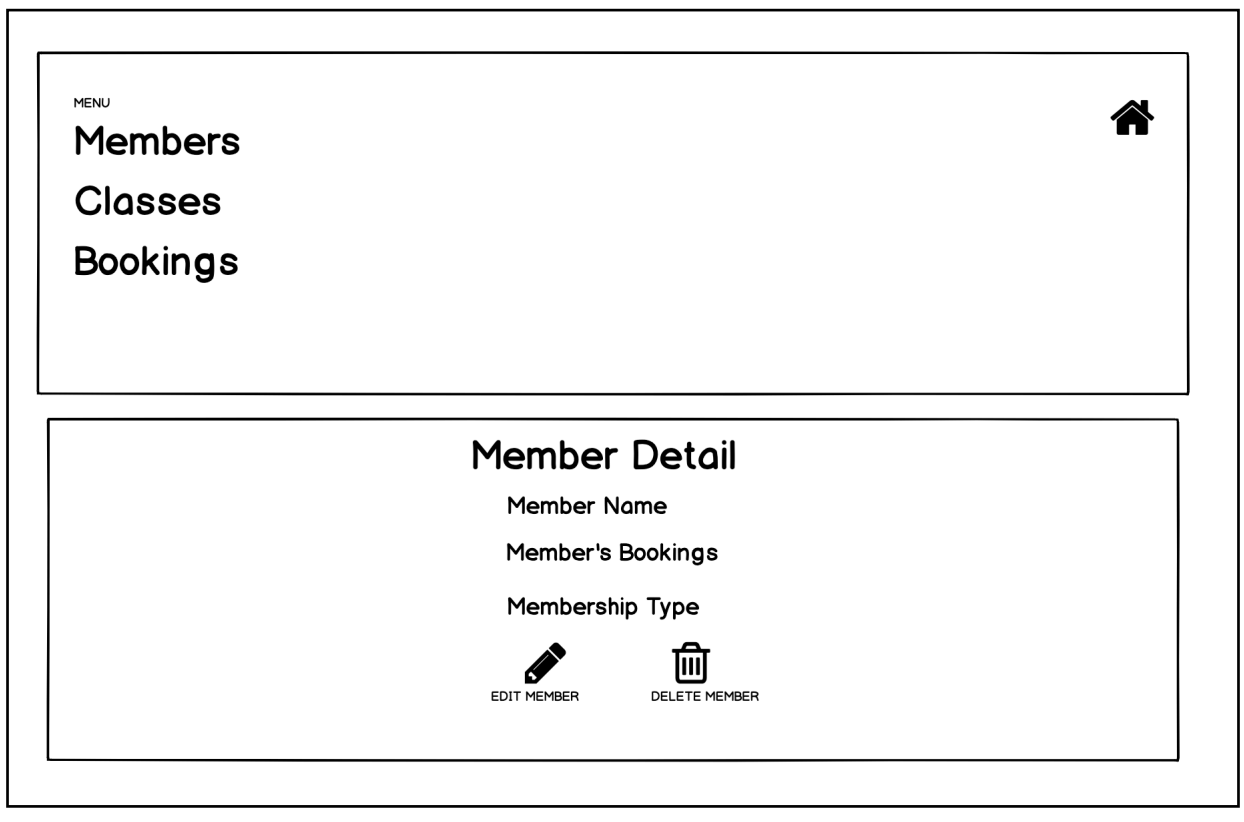
The user site map shows the structure for the Gym project. This has a menu with three main sections that are always one click away.
From these, new classes/members/bookings can be added and their details can be seen.
From the detail page, the stored data can be edited or deleted.



Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams

Here are the wireframes for my Gym App project.





Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method

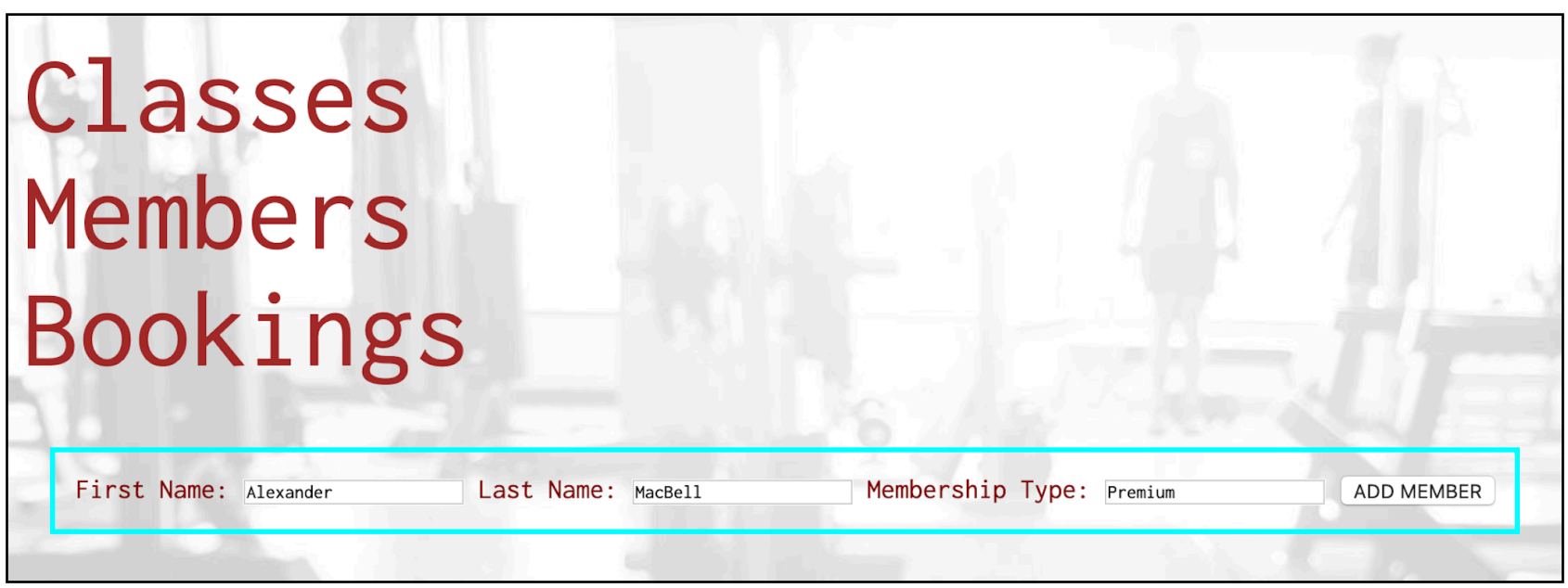
Here’s an example of Pseudocode used for a method in my Gym project.
The method already_booked? checks if a booking for a given member and a given gym class already exists.

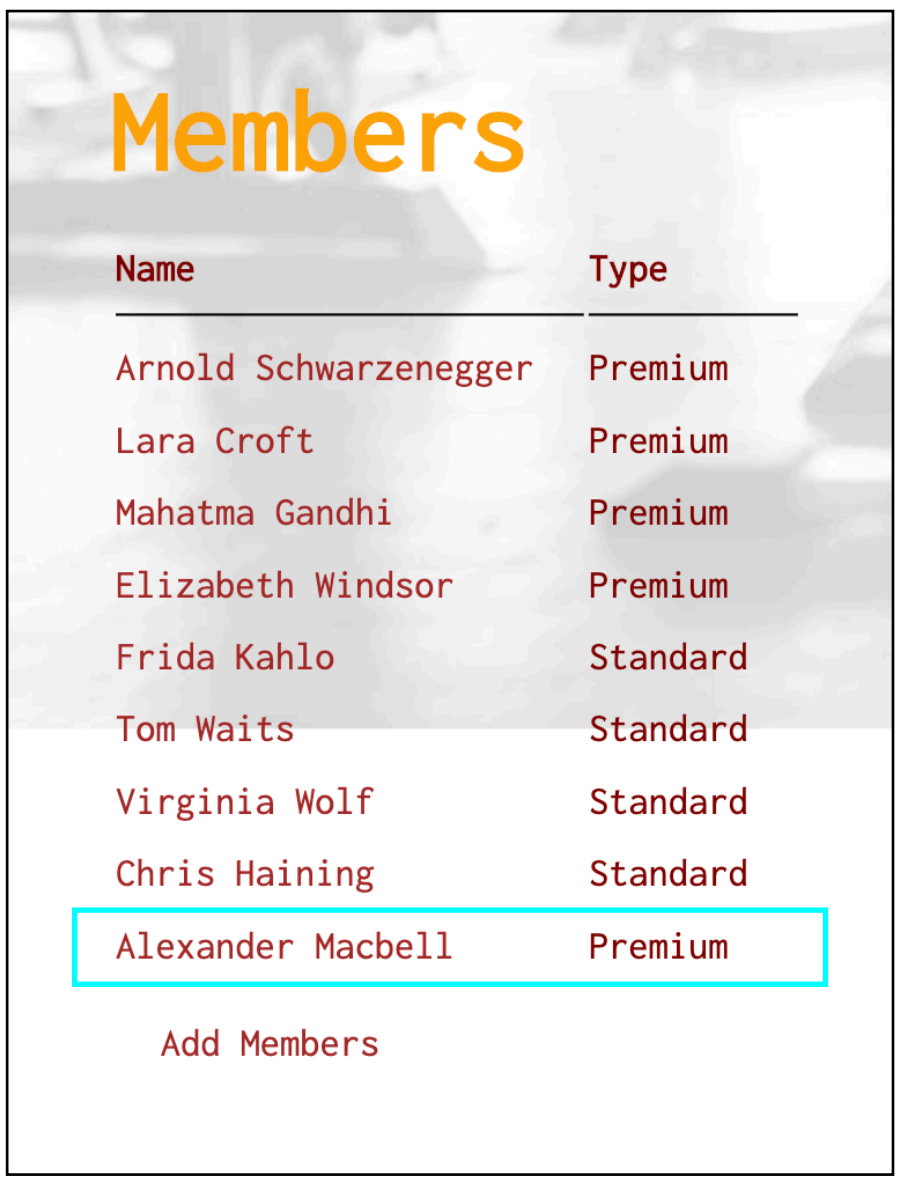
```
60
61  # method that checks if a booking already exists.
62  # It should take two params (member_id and gym_class_id).
63  # It should look for a booking with those two params and see if one exists in bookings.
64  # it should return a boolean (true for booked, false for non-booked)
65
```

```
def self.already_booked?(member_id, gym_class_id)
  sql = "SELECT * FROM bookings WHERE (member_id, gym_class_id) = ($1, $2)"
  values = [member_id, gym_class_id]
  no_booking_found = SqlRunner.run(sql, values).first.nil?
  return !no_booking_found
end
```

Unit	Ref	Evidence
P	P.13	Show user input being processed according to design requirements. Take a screenshot of: * The user inputting something into your program * The user input being saved or used in some way

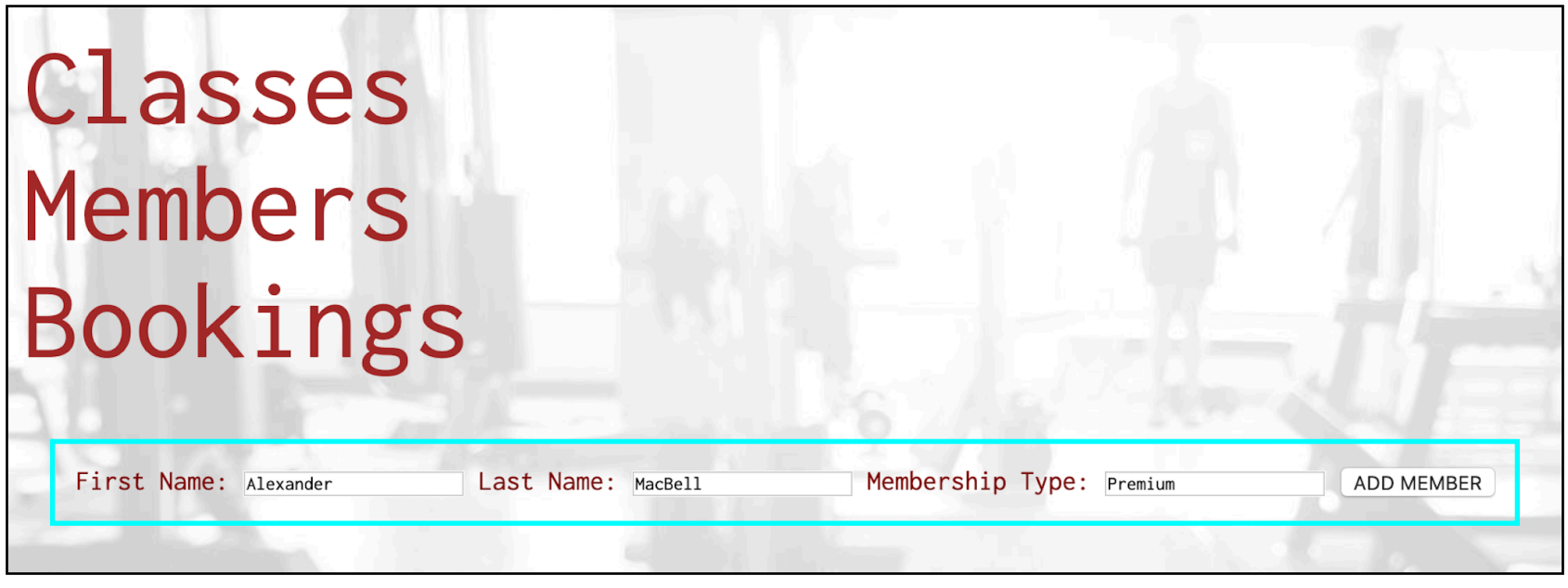
The first screenshot shows a new member being added by the user.
The second screenshot shows the newly added member at the bottom of the members list.

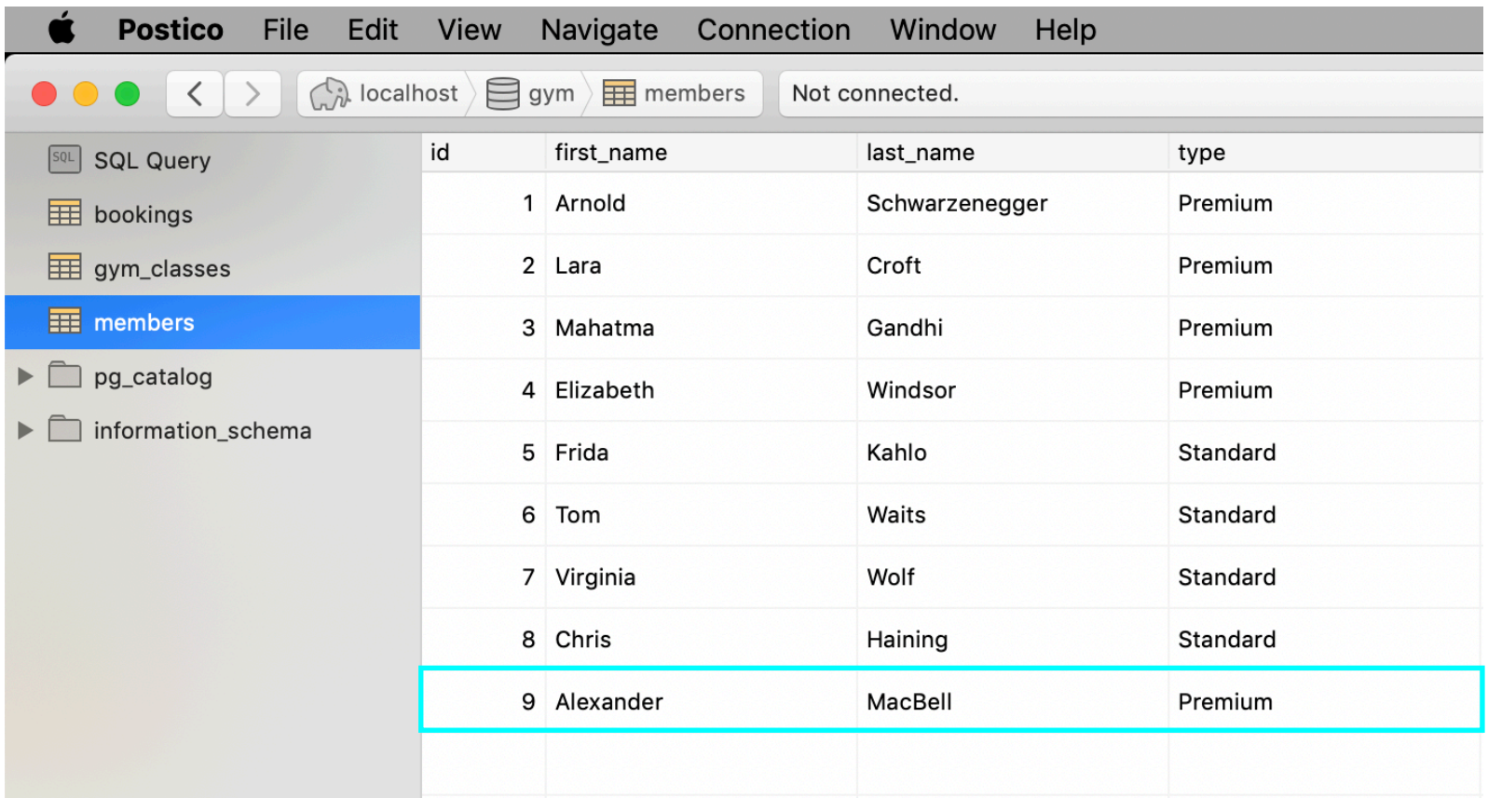




Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved

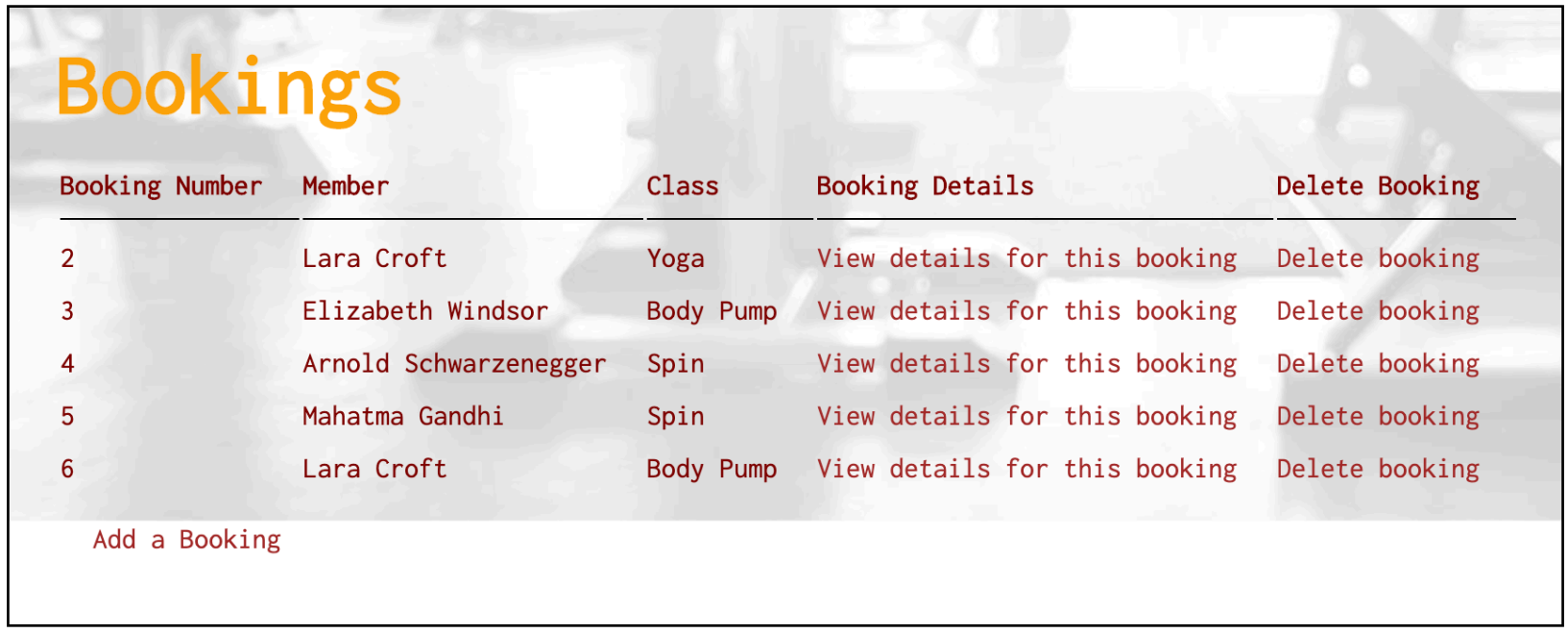
The first screenshot shows a new member being added by the user.
The second screenshot proves that the newly added member has been saved in the database.





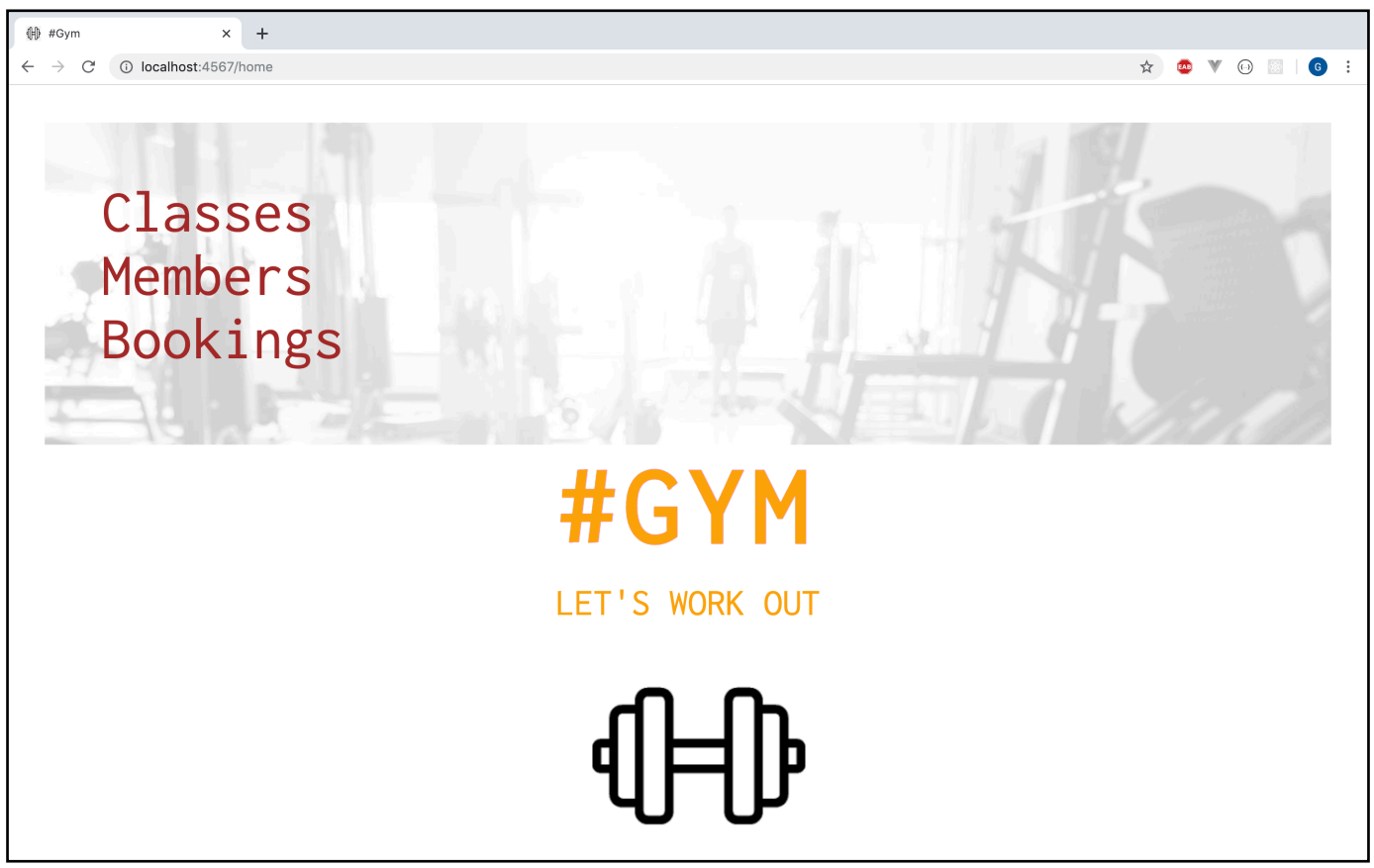
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program

The first screenshot shows the user performing a deleting action on a member object.
The second screenshot shows the effect of this action (the member object has been deleted).

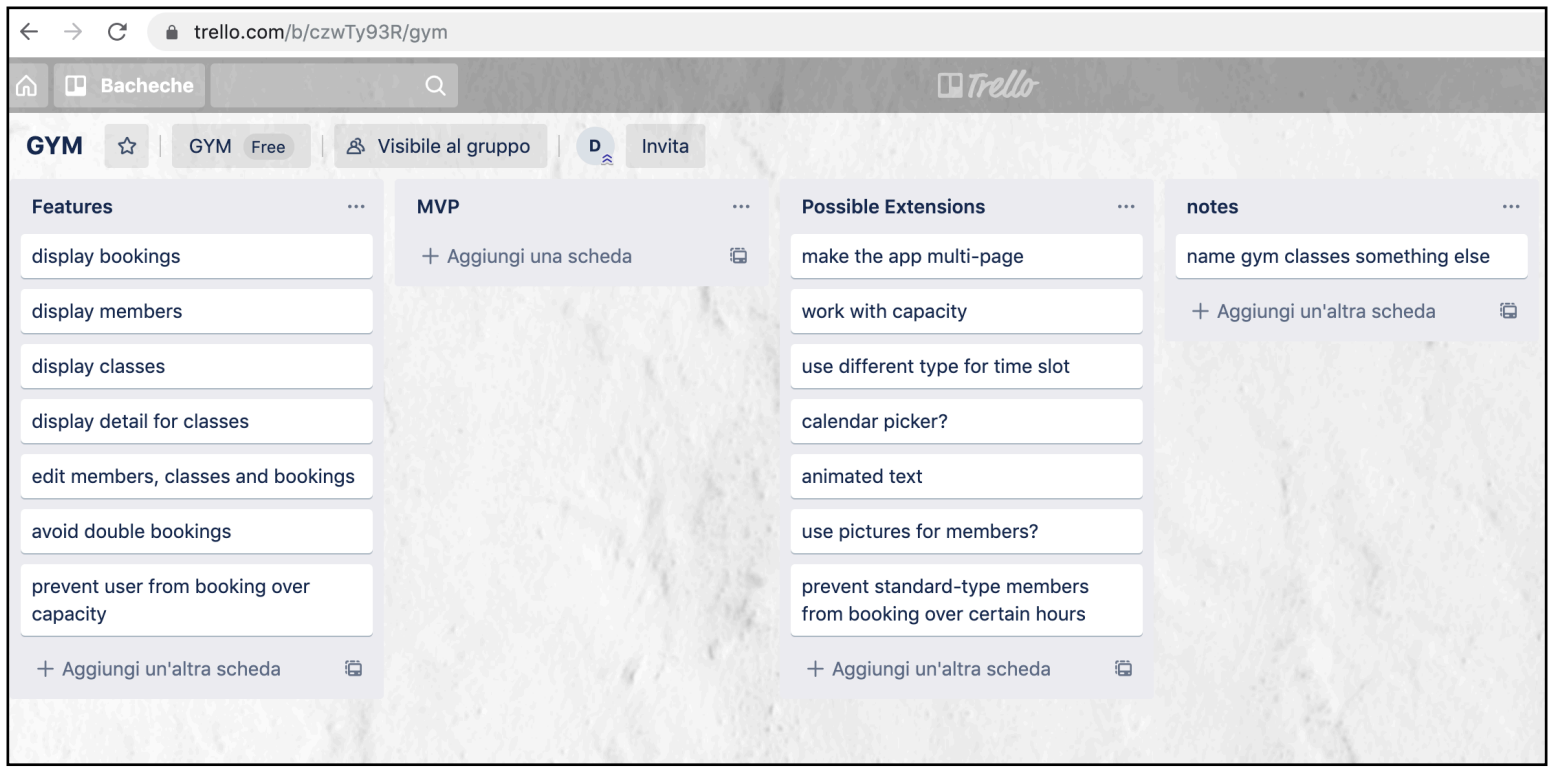


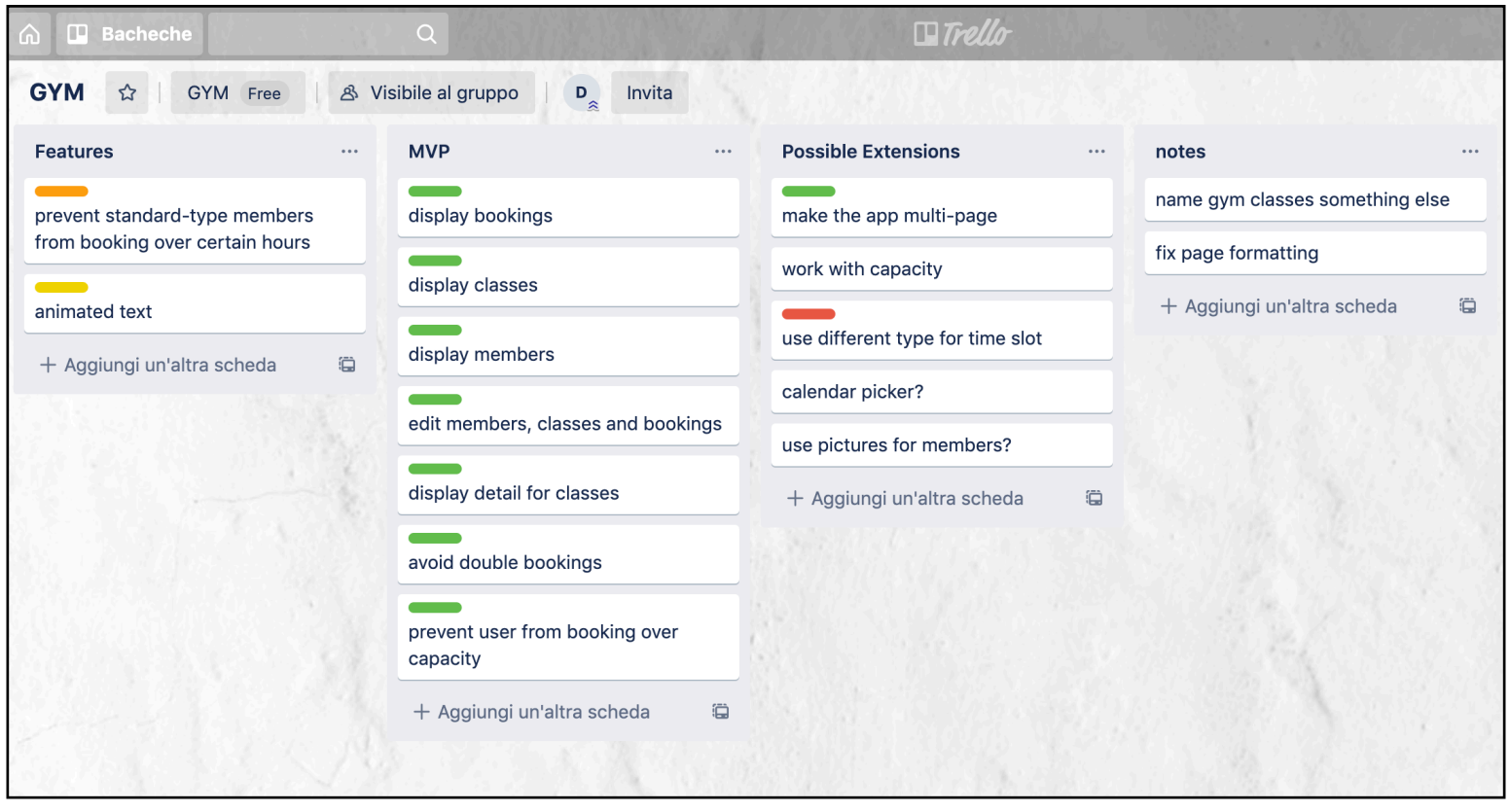
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.

https://github.com/DavCampagna/gym_project



Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.





Week 7

Unit	Ref	Evidence
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running

These screenshots shows the code implementing an API for anime films.

The API is fetched. Then, the result is transformed into JSON format.

The fetched data is used to populate a list of films. Also, the relevant details for the film are displayed when selecting from the list.

```
  mounted(){
  fetch('https://ghibliapi.herokuapp.com/films')
  .then(res => res.json())
  .then(films => this.films = films)
  eventBus.$on("film-selected", film => this.displayFilmInfo(film));
  eventBus.$on("favourite-added", film => this.markFavourite(film));
  eventBus.$on("favourite-removed", film => this.unmarkFavourite(film));
  eventBus.$on("film-watched", film => this.markWatched(film));
  eventBus.$on("watchlist-added", film => this.markWatchlistItem(film));
  eventBus.$on("watchlist-removed", film => this.unmarkWatchlistItem(film));
  eventBus.$on("film-unselected", film => this.hideFilmInfo(film));
  eventBus.$on("watchAgain-added", film => this.markWatchAgainItem(film));
  }
```


```
<script>
import FilmFilterForm from './components/FilmFilterForm.vue'
import { eventBus } from './main.js'
import FilmList from './components/FilmList.vue'
import FilmDetail from './components/FilmDetail.vue'
import FavouriteFilmList from './components/FavouriteFilmList.vue'
export default {
  name: "app",
  components: {
    "film-filter-form": FilmFilterForm,
    "film-list": FilmList,
    "film-detail": FilmDetail,
    "favourite-film-list": FavouriteFilmList
  },
  data() {
    return {
      films: [],
      selectedFilm: null,
      favourites: [],
      watched: [],
      watchlist: [],
      watchAgain: []
      // component: "favourite-film-list"
      // websiteTag: '<a href="https://www.studioghibli.com.au/"></a>'
    }
  },
  methods: {
    displayFilmInfo: function(film) {
      this.selectedFilm = film
      },
```

Week 8

Unit	Ref	Evidence
P	P.2	Take a screenshot of the project brief from your group project.

Educational App

The BBC are looking to improve their online offering of educational content by developing some interactive browser applications that display information in a fun and interesting way. Your task is to make an a Minimum Viable Product or prototype to put forward to them - this may only be for a small set of information, and may only showcase some of the features to be included in the final app.

MVP

A user should be able to:

- view some educational content on a particular topic
- be able to interact with the page to move through different sections of content

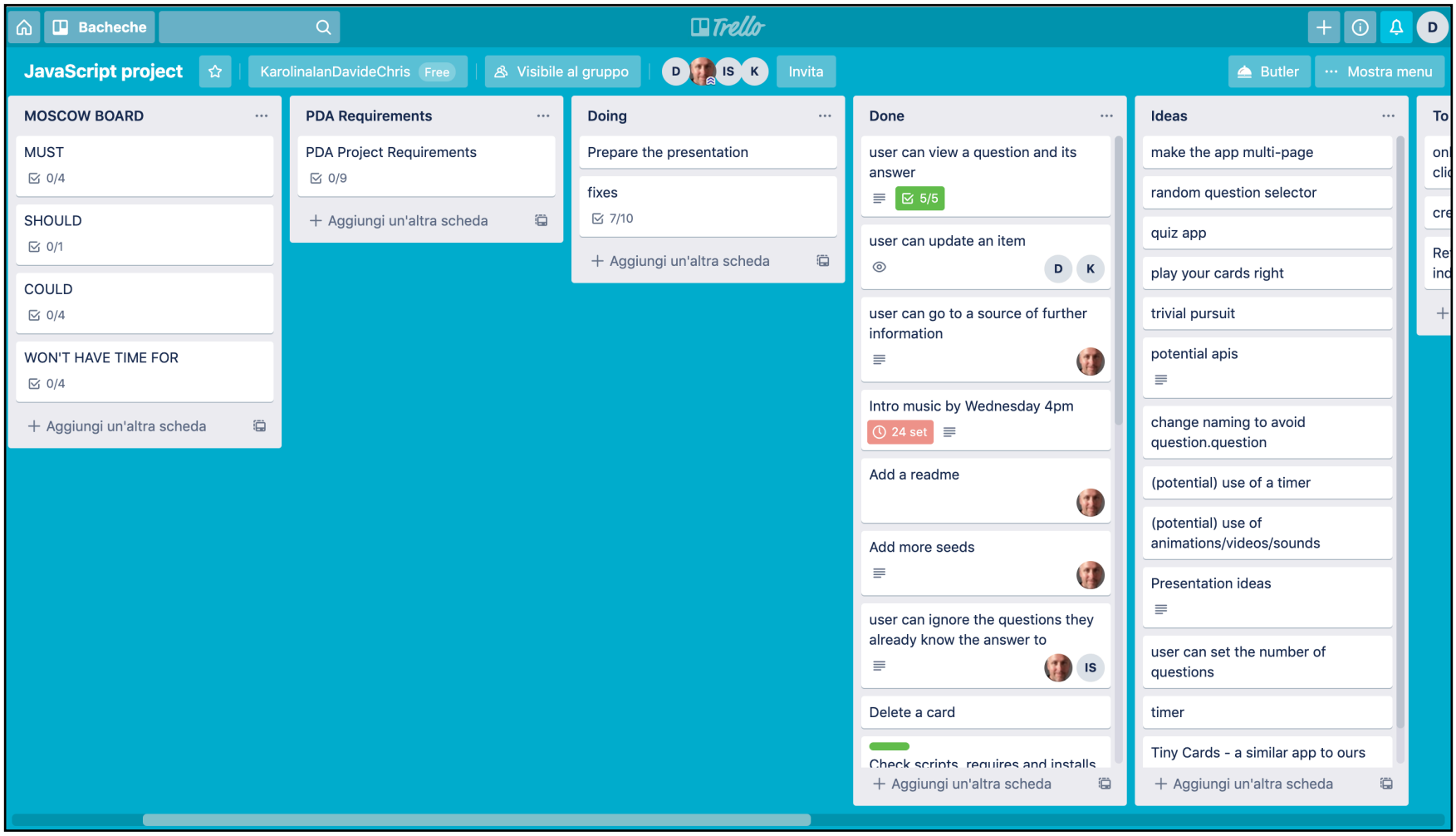
Example Extensions

- Use an API to bring in content or a database to store information.
- Use charts or maps to display your information to the page.

API, Libraries, Resources

- https://www.highcharts.com/ HighCharts is an open-source library for rendering responsive charts.
- https://korigan.github.io/Vue2Leaflet/#/ Leaflet is an open-source library for rendering maps and map functionality.

Unit	Ref	Evidence
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.



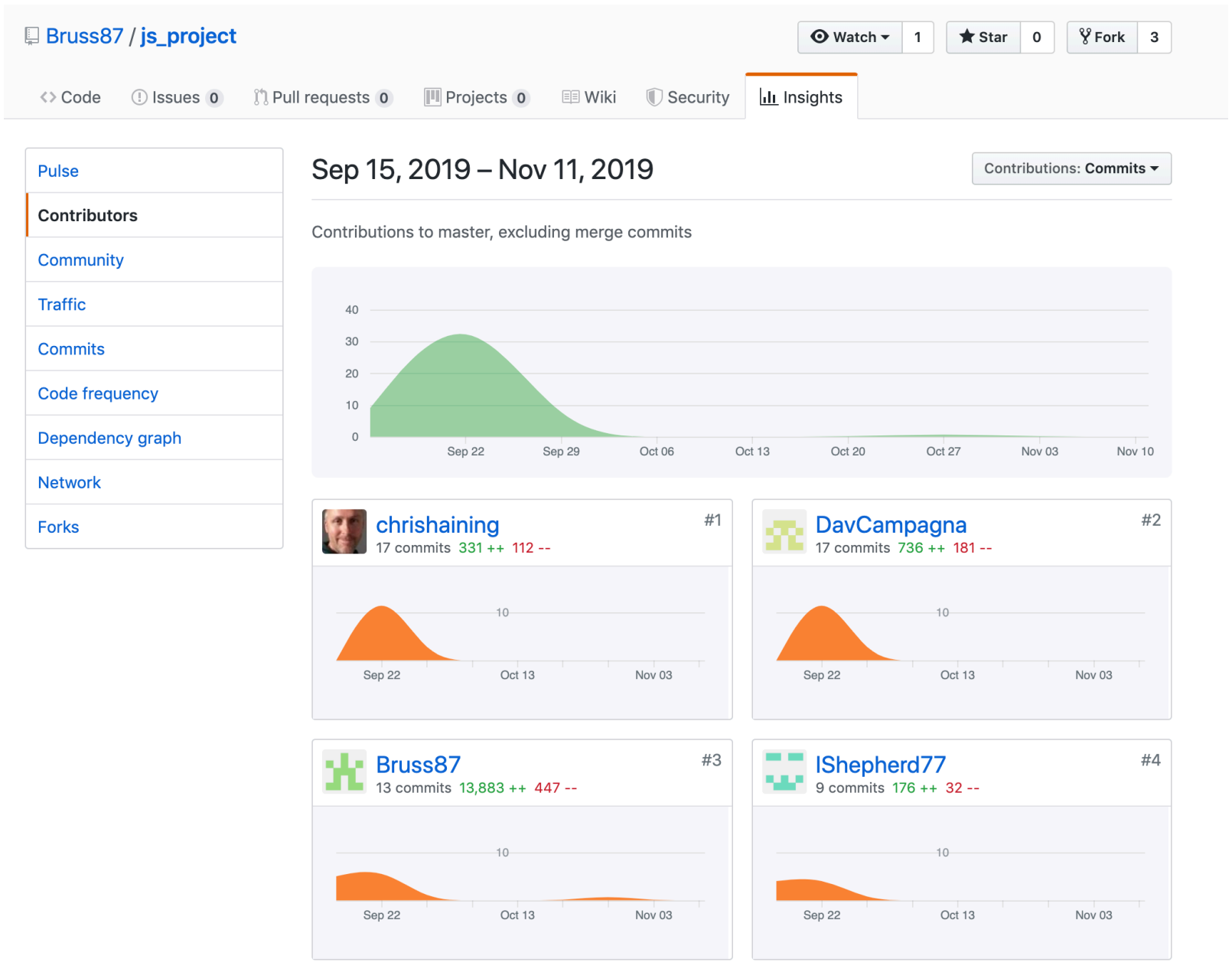
Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.

This acceptance criteria and test plan was built for my group project My Coding Pal, an educational app for the users to create and save their own questions, filter them by topic and grade them to keep track of their progress.

Acceptance Criteria	Expected Result	Pass/Fail
A user is able to	The … does … when	Pass/Fail
A user is able to see a list of questions	A list of questions is displayed when logging into the app	Pass
A user is able to add a new question	When clicking on “Add a question” button, a form to add the new question appears	Pass
A user is able to select a question from the list	When selecting an item from the list, the information about the selected question is displayed in the form of a flipping card that appears on the right side of the page	Pass
A user is able to see the correct answer to the selected question	When hovering over the question-card, this one flips and the answer is revealed on the back of the card.	Pass
A user is able to update a question	When selecting an item from the list, a form to update the question-card is displayed under the card itself.	Pass
A user is able to delete a question	When selecting an item from the list, a button to delete the question-card appears under the card itself.	Pass
A user is able to filter questions by topic	When logging into the app, at the top of the page a search bar allows the user to choose from a list of topics and filter the questions for the selected topic.	Pass
A user is able to mark questions that have been answered correctly	When selecting an item from the list, a radio button appears under the card item to allow the user to mark the selected question as “mastered”	Fail

Week 9

Unit	Ref	Evidence
P	P.1	Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.



Week 11

Unit	Ref	Evidence
P	P.18	Demonstrate testing in your program. Take screenshots of: * Example of test code * The test code failing to pass * Example of the test code once errors have been corrected * The test code passing

Here's an example of test code.
This tests a method called addBook which adds a Book object to a library (which starts empty).
The method is tested by an assertion between the value of 1 book added and the result of calling a bookCount() method.
The test initially fails to pass as the wrong return type (void) is being used by mistake instead of int.
Once this has been changed to int, the test passes.

```
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TestLibrary {

    private Book book1;

    private Library library;

    @Before
    public void setUp() throws Exception {
        book1 = new Book( title: "Mrs Dalloway",  author: "Virginia Wolf",  genre: "Novel");

        library = new Library( capacity: 1);

    }

    @Test
    public void canAddBook() {
        library.addBook(book1);
        assertEquals( expected: 1, library.bookCount());
    }
```

```
Messages:  Build
Information: java: Errors occurred while compiling module 'library'
Information: javac 1.8.0_212 was used to compile java sources
Information: 2019-11-11 16:11 - Build completed with 1 error and 0 warnings in 1 s 542 ms
/Users/user/codeclan_work/week_11/day_2/classes_homework_solution/src/main/java/Library.java
Error:(32, 5)  java: missing return statement
```

```
public void addBook(Book book) {
    if(hasCapacity()){
        this.collection.add(book);
        this.addToGenreHashmap(book);
    }
}
```

→

```
public int addBook(Book book) {
    if(hasCapacity()){
        this.collection.add(book);
        this.addToGenreHashmap(book);
    }
}
```

```
Run:  TestLibrary
Tests passed: 1 of 1 test – 4 ms
TestLibrary  4 ms
canAddBook  4 ms
/Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.

This screenshot shows encapsulation in Java. The Bedroom class is public and able to be instantiated outside of the class.
However, its properties (roomNo, capacity, type, nightly rate, guests) are private so that they are not available to be accessed or manipulated outside of the class.

```java
public class Bedroom {

    private int roomNo;
    private int capacity;
    private String type;
    private double nightlyRate;
    private ArrayList<Guest> guests;

    public Bedroom(int roomNo, int capacity, String type, double nightlyRate) {
        this.roomNo = roomNo;
        this.capacity = capacity;
        this.type = type;
        this.nightlyRate = nightlyRate;
        this.guests = new ArrayList<Guest>();
    }
```

Week 12

Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.

These screenshots shows a program using Polymorphism, the ability of an object to take on many forms.

The “Orchestra” class has a method “hire” that takes in a musician object of the type IPlay.

```
public void hire(IPlay musician) {
    if (freeSpaces() > 0) {
        musicians.add(musician);
    }
}
```

The classes Violinist and Pianist implement the interface IPlay, inheriting the IPlay behaviours.

```
public class Violinist implements IPlay{
    private String name;
    private int yearsOfTraining;
    private double salary;

    public Violinist(String name, int yearsOfTraining, double salary){
        this.name = name;
        this.yearsOfTraining = yearsOfTraining;
        this.salary = salary;
    }
```

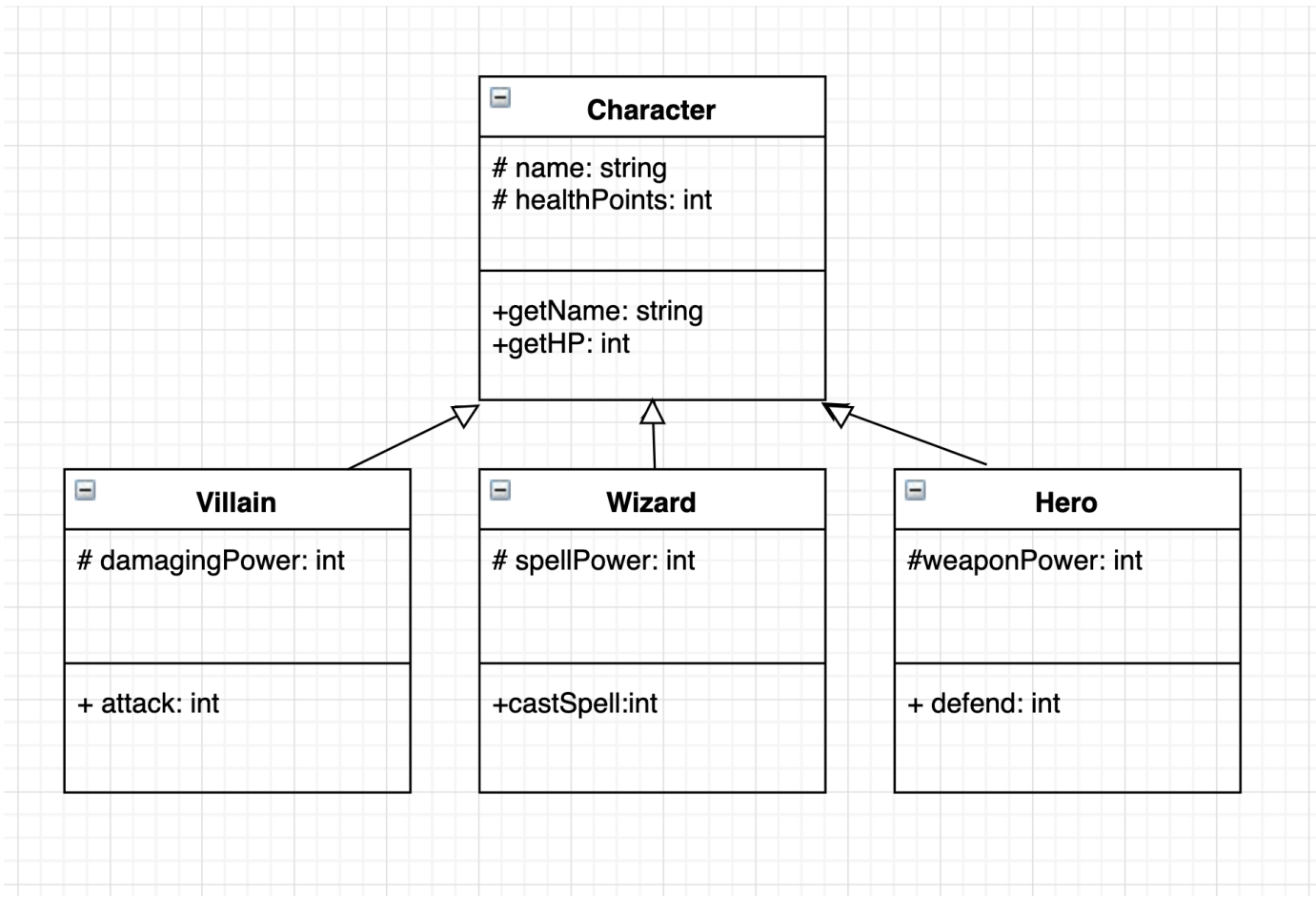
```
public class Pianist implements IPlay{
    private String name;
    private int yearsOfTraining;
    private double salary;

    public Pianist(String name, int yearsOfTraining, double salary){
        this.name = name;
        this.yearsOfTraining = yearsOfTraining;
        this.salary = salary;
    }
```

This screenshot shows the test written for the function “hire”. Here, a pianist an a violinist are added to the orchestra.

```
@Test
public void canHireMusicians(){
    orchestra.hire(pianist);
    orchestra.hire(violinist);
    assertEquals( expected: 2, orchestra.musiciansCount());
}
```

Unit	Ref	Evidence
A&D	A.D.5	An Inheritance Diagram

This diagram shows three classes (Villain, Wizard and Hero) are inheriting attributes and behaviours from the class Character.



Unit	Ref	Evidence
I&T	I.T.2	Take a screenshot of the use of Inheritance in a program. Take screenshots of: *A Class *A Class that inherits from the previous class *An Object in the inherited class *A Method that uses the information inherited from another class.

The class Pianist extends the class Musician, inheriting all its attributes. The class Musician implements the IPlay interface, therefore Pianist will also inherit the IPlay behaviours.

```java
public class Pianist extends Musician{

    public Pianist(String name, int yearsOfTraining, double salary, String instrument){
        super(name, yearsOfTraining, salary, instrument);
    }
}
```

```java
public abstract class Musician implements IPlay {
    private String name;
    private int yearsOfTraining;
    private double salary;
    private String instrument;

    public Musician(String name, int yearsOfTraining, double salary, String instrument){
        this.name = name;
        this.yearsOfTraining = yearsOfTraining;
        this.salary = salary;
        this.instrument = instrument;
    }

    public String getName() {
        return name;
    }

    public int getYearsOfTraining() {
        return yearsOfTraining;
    }

    public double getSalary() {
        return salary;
    }

    public String getInstrument() {
        return instrument;
    }

    public String play() {
        return "Playing the " + instrument;
    }
}
```

pianist is an instance of the Pianist class.

The last screenshot shows the test for the method “play” that the Pianist class inherits from Musician.

```java
Pianist pianist;

@Before
public void before(){
    pianist = new Pianist( name: "Aurelius Maggi", yearsOfTraining: 18, salary: 200.00, instrument: "piano");
}
```

```java
    @Test
    public void canPlay(){
        assertEquals( expected: "Playing the piano", pianist.play());
    }
}
```

Week 14

Unit	Ref	Evidence
P	P.9	Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.

The first screenshot shows an example of linear search algorithm applied to an array of numbers.
The function linearSearch takes in a number (55) and loops through every item in the array to find it, starting from the item at index 0 and moving on to the next one until the end of the array. A string with a message is returned if the number is not found.

The second screenshot shows the result of the function, having found the number 55 at the index 5.

```
let numbers = [1, 17, 8232, 1324, 4, 55, 324];

function linearSearch(arr, elem) {
  for(let i = 0; i < arr.length; i++) {
    if(arr[i] === elem){
      return i;
    }
  }
  return "Number not found"
}

console.log(linearSearch(numbers, 55));
```

```
➜  algorithms node linear_search.js
5
➜  algorithms
```

This screenshot shows an algorithm used to find a Car object by passing a String for the car model.

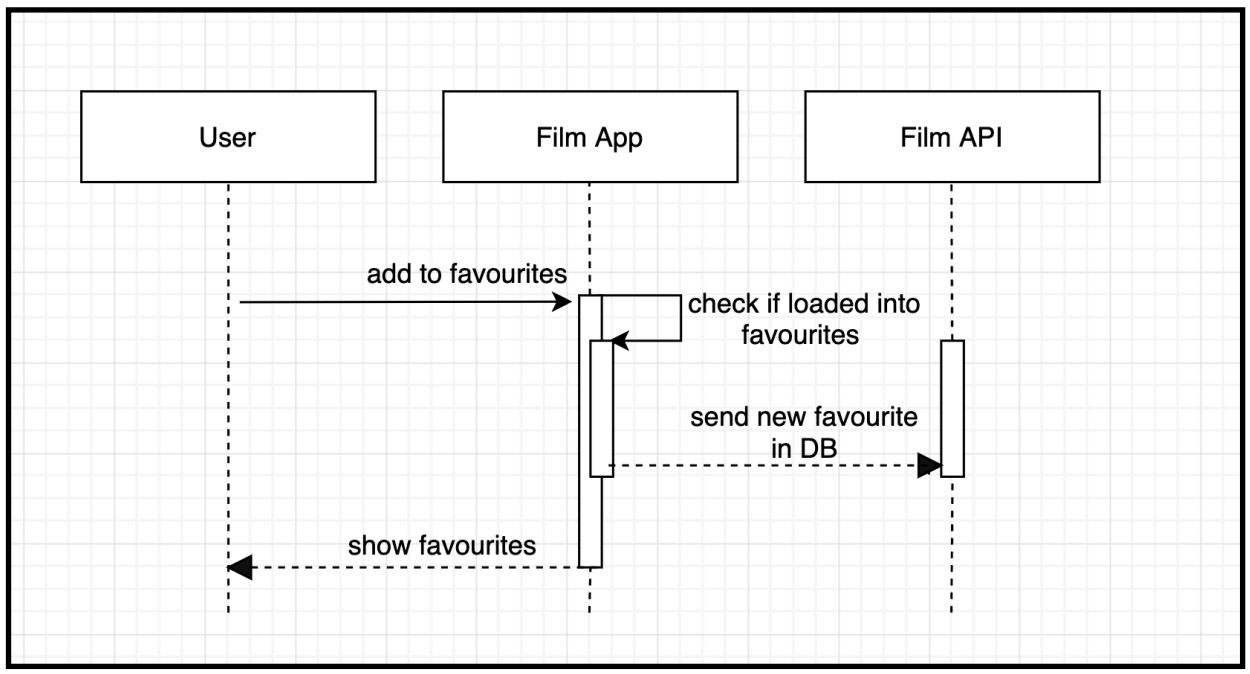
First, a foundCar variable is set to null.
Then, a for loop checks through the array of Car objects in the stock array for a car object whose model property is equal to the string that has been passed for model.

If a car object is found, this becomes the foundCar and is returned as such.

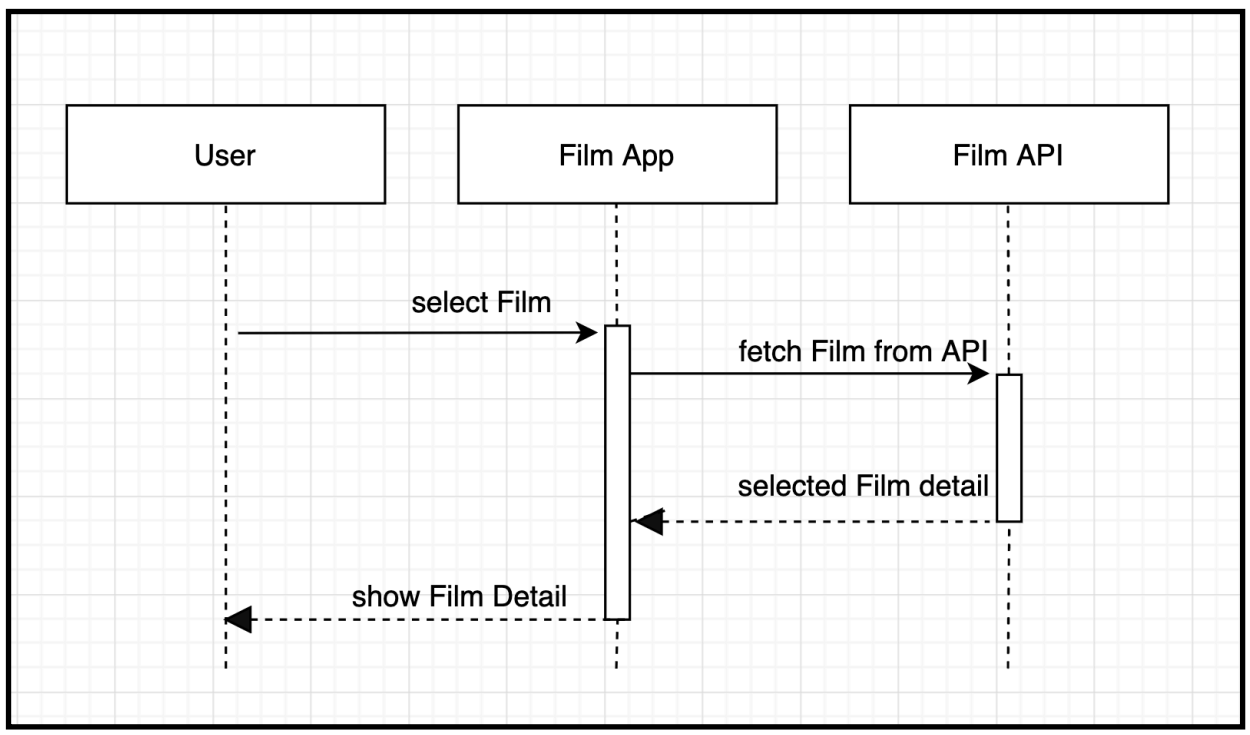
```
public Car getCarByModel(String model){
    Car foundCar = null;
    for(Car car : this.stock) {
        if(car.getModel() == model){
            foundCar = car;
        }
    }
    return foundCar;
}
```

Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).

The first diagram shows a sequence diagram for a project involving films. The diagram shows what happens when a user selects a film item from the list of films: the event is passed to the app, which sends a request to the database API. This returns the details for the selected film to the app, allowing the details to be displayed for the user.

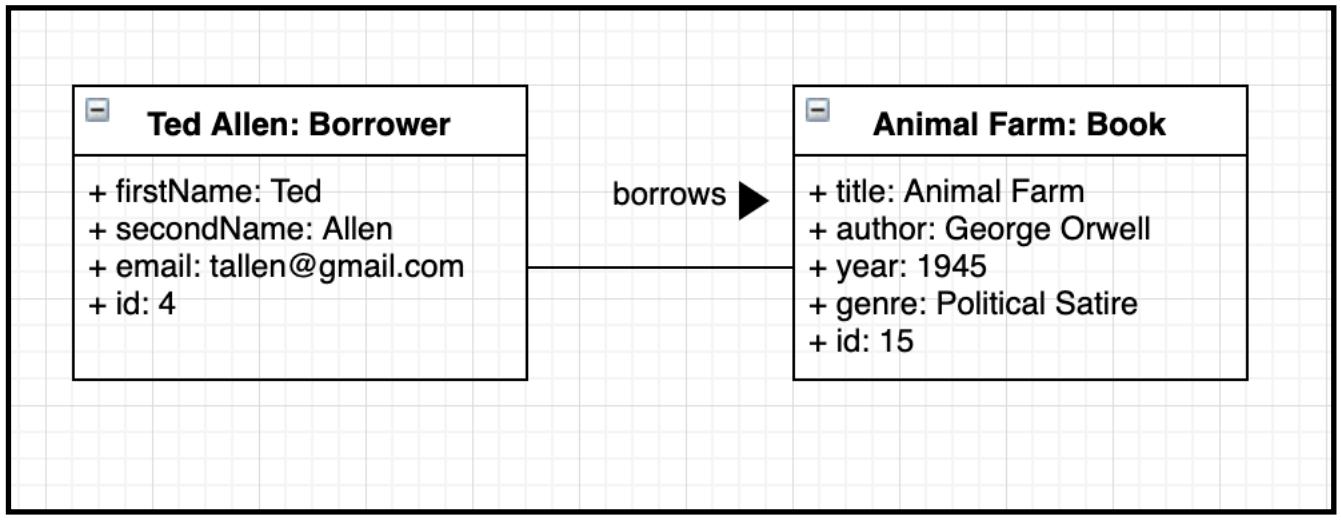


The second diagram shows the process of adding to favourites. First, the list of already loaded favourite films is checked, before sending the new favourite to the database.



Unit	Ref	Evidence
P	P.8	Produce two object diagrams.

The diagram shows 2 objects and their attributes.



Unit	Ref	Evidence
P	P.17	Produce a bug tracking report

Here is a table showing the bug tracking and fixes for the film project.

Bug/Error	Solution	Date
Details not showing when selecting film	Added event bus to FilmFilterForm file	10/11/2019
New Film not being saved into database	Fixed Post Method	10/11/2019
Film objects no longer appear in FilmList	Re-factored use of props in components	10/11/2019
FilmDetail not showing anymore	V-if added to component	11/11/2019
CSS menu and header would break the page	Wrapped elements in div tags	12/11/2019