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.

Each point that required details the Assessment Criteria (What you have to show) along with a brief description of the kind of things you should be showing.

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

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:

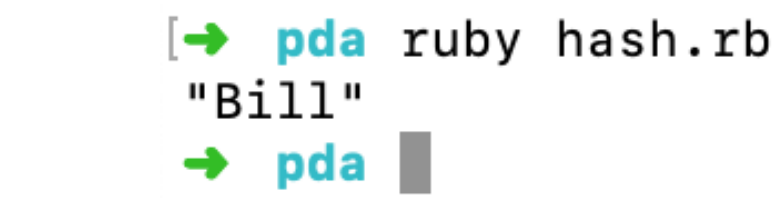
```
my_array = ["Andrew", "Bill", "Charles", "Daniel", "Edward"]

def print_elements_from_array(array)
  for element in array
    p element
  end
end
```

```
➜  pda ruby array.rb
"Andrew"
"Bill"
"Charles"
"Daniel"
"Edward"
➜  pda
```

My_array is an array of names which are just Strings and print_elements_from_array() is a function that can be passed the array mentioned. The results of the print function being called and passed the my_array as an argument are in the second screenshot. The function loops through every element of the array and outputs it to the console.

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:

```
my_hash_list = [
  {
    name: "Bill",
    age: 26,
    city: "Edinburgh",
    job: "Doctor"
  },
  {
    name: "Charles",
    age: 34,
    city: "Glasgow",
    job: "Teacher"
  }
]

def get_name_from_hash hash
  return hash[:name]
end

name = get_name_from_hash(my_hash_list[0])
p name
```



There is a hash shown in the first screenshot which has an array of hashes my_hash_list. Each hash contains keys (name, age, city, job) along with random details as their values. The function get_name_from_hash will be used with the first hash in the array as the argument passed and the result of calling this function is the name is output to console.

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:

```ruby
def favourite_film
  film_list = self.films.map{|film| film.title}
  frequency_hash = film_list.reduce(Hash.new(0)){
    |hash, title| hash[title] += 1; hash
  }
  max = frequency_hash.values.max
  print (frequency_hash.find_all{
    |title, frequency| frequency == max}).map{
      |title, frequency| title}
end
```

```
[1] pry(main)> customers[0].favourite_film
["Robocop"]=> nil
[2] pry(main)> customers[1].favourite_film
[]=> nil
[3] pry(main)> customers[2].favourite_film
["Robocop", "Jurassic Park"]=> nil
[4] pry(main)>
```

The favourite film method takes a list of films, for which a customer has bought tickets, and turns it into a hash using the film's name as the key and the value as a counter for whenever we see another occurrence of the same film name in the list. This counts the number of tickets the customer has bought for the same film. We then check what the max value is in the hash and use the find_all method from arrays to find only the films which have this max value as there may be multiple films the customer has seen the same number of max times. The result is shown in the screenshot above.

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:

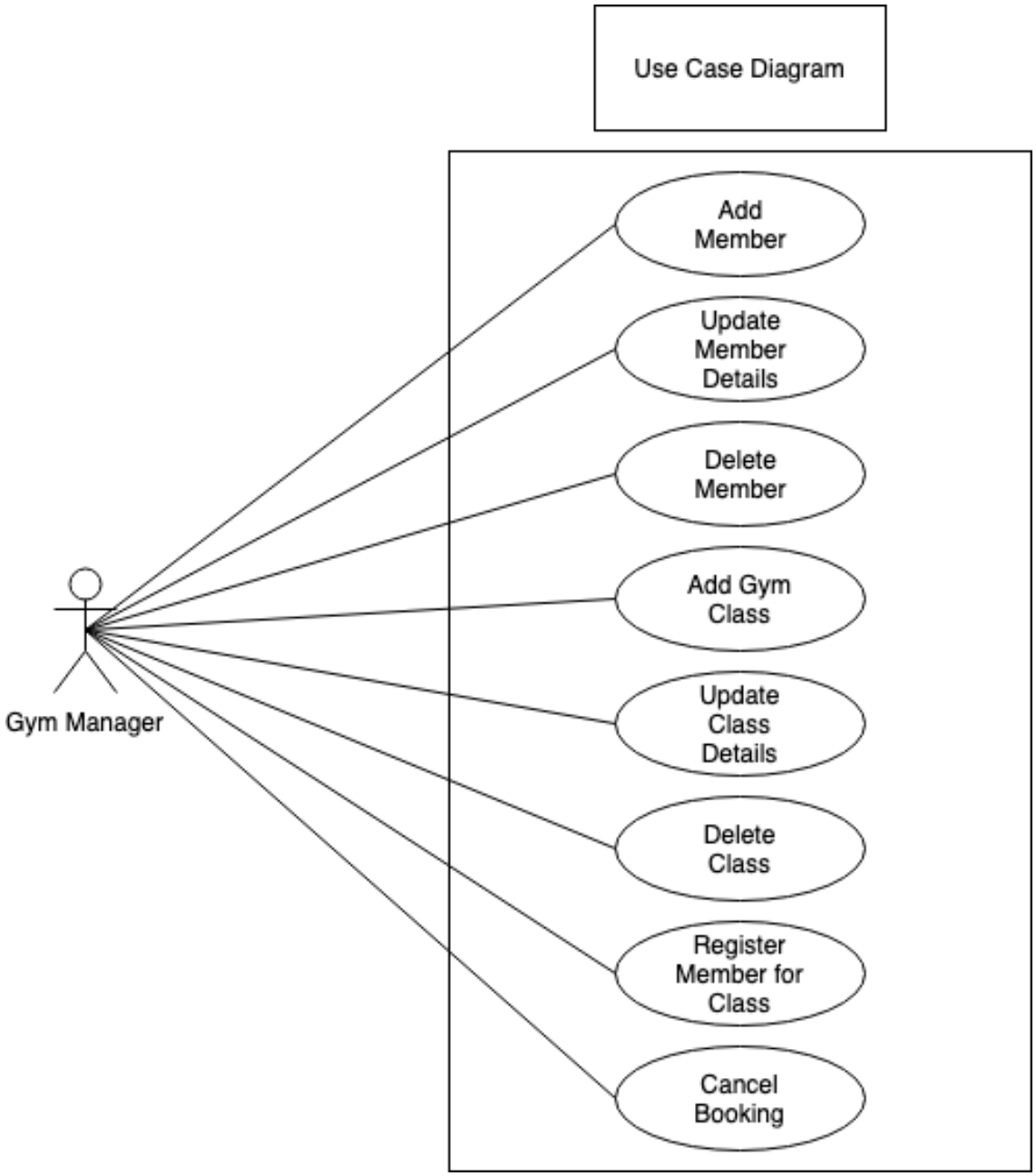
```ruby
def popular_show_time
  sql = "SELECT screenings.* FROM screenings INNER JOIN tickets
  ON screenings.id = tickets.screening_id
  WHERE screenings.film_id = $1"
  values = [@id]
  screenings_data = SqlRunner.run(sql, values)
  screenings = screenings_data.map{|screening|
  Screening.new(screening)}

  # ------------------------------------------------------------
  # This way will return multiple popular times
  show_times = screenings.map{|screening| screening.show_time}
  frequency_hash = show_times.reduce(Hash.new(0)){
    |hash, show_time| hash[show_time] += 1; hash}
  max = frequency_hash.values.max
  return (frequency_hash.find_all{
    |show_time, frequency| frequency == max}).map{
      |show_time, frequency| show_time}
end
```

```
[3] pry(main)> Film.all[0].popular_show_time
["18:00"]=> nil
[4] pry(main)>
```

The popular_show_time method is used to sort a list of movie screenings of one particular film into a hash which contains the screening show time as the key and the number of tickets bought as the value. In this way the hash is used to sort the screenings with the number of tickets bought as key-value pairs. The result of the method running is shown in the second screenshot.

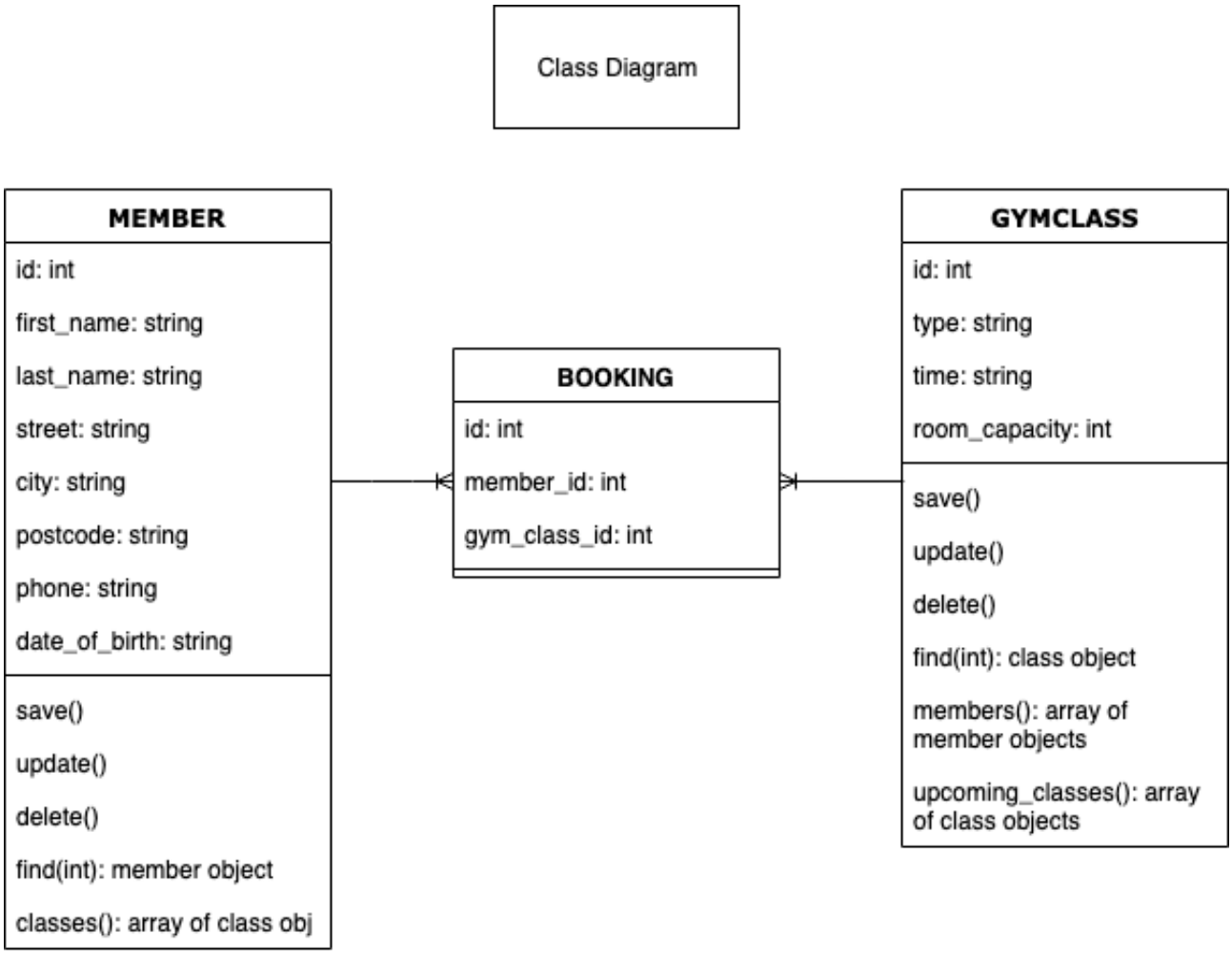
Week 5 and 6

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



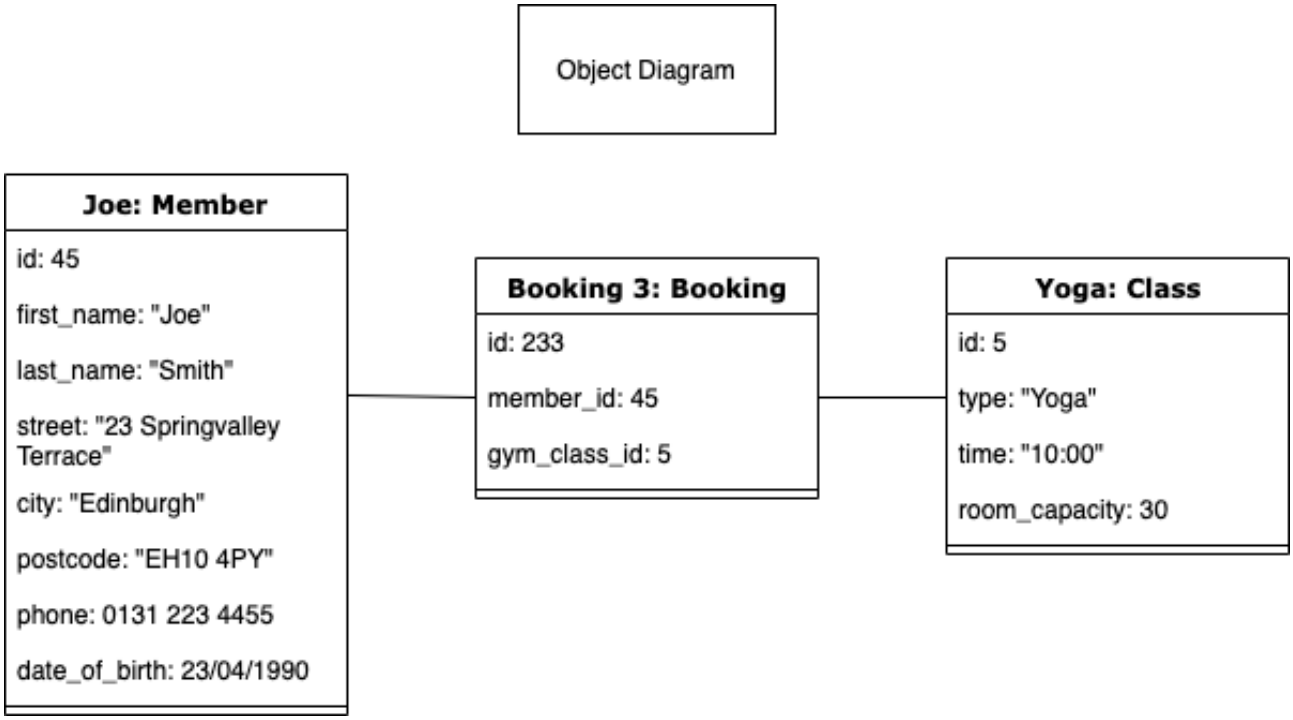
The above shows a use case diagram for a gym management web app. It shows the cases required to implement the MVP as described in the briefing by the intended end user, the gym manager. This shows all the functionality the gym manager needs for a basic product.

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



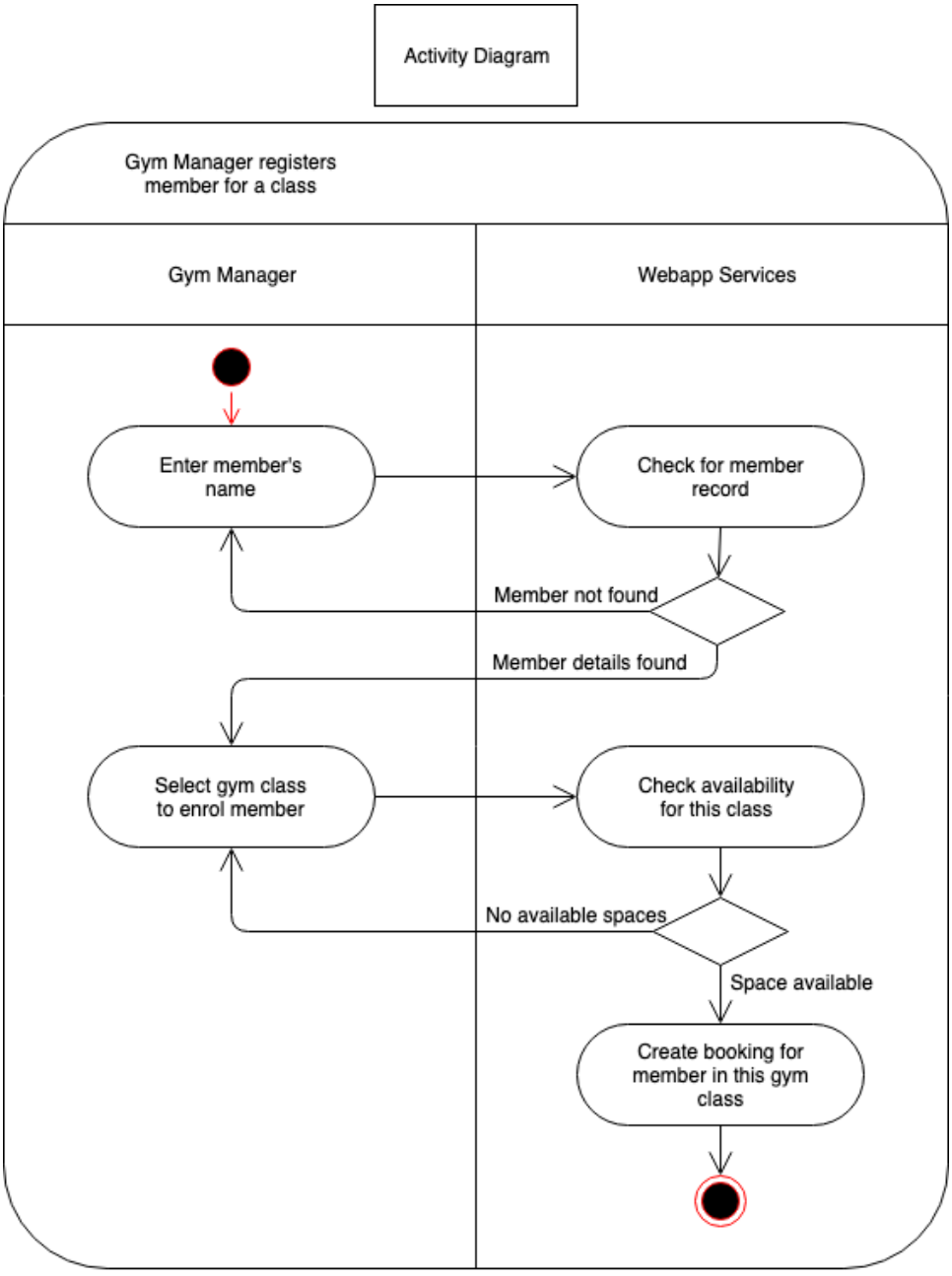
The above diagram shows the class diagram representing the 3 classes which would be required to implement the MVP requirements of the gym management web app. These class diagrams model all the required components to have a basic working app which fulfils the project brief.

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



The above diagram is an example of an object diagram with sample data that might be stored in the database of the gym management web app. This approximates the expected data the app should receive from user input via the forms available in each of the sections (i.e. the members, gym classes and bookings).

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



The diagram here represents an activity diagram of the processes that would be involved if a gym manager were to book a gym member into an upcoming gym class. These separate the steps the gym manager would take and how the booking system used would respond.

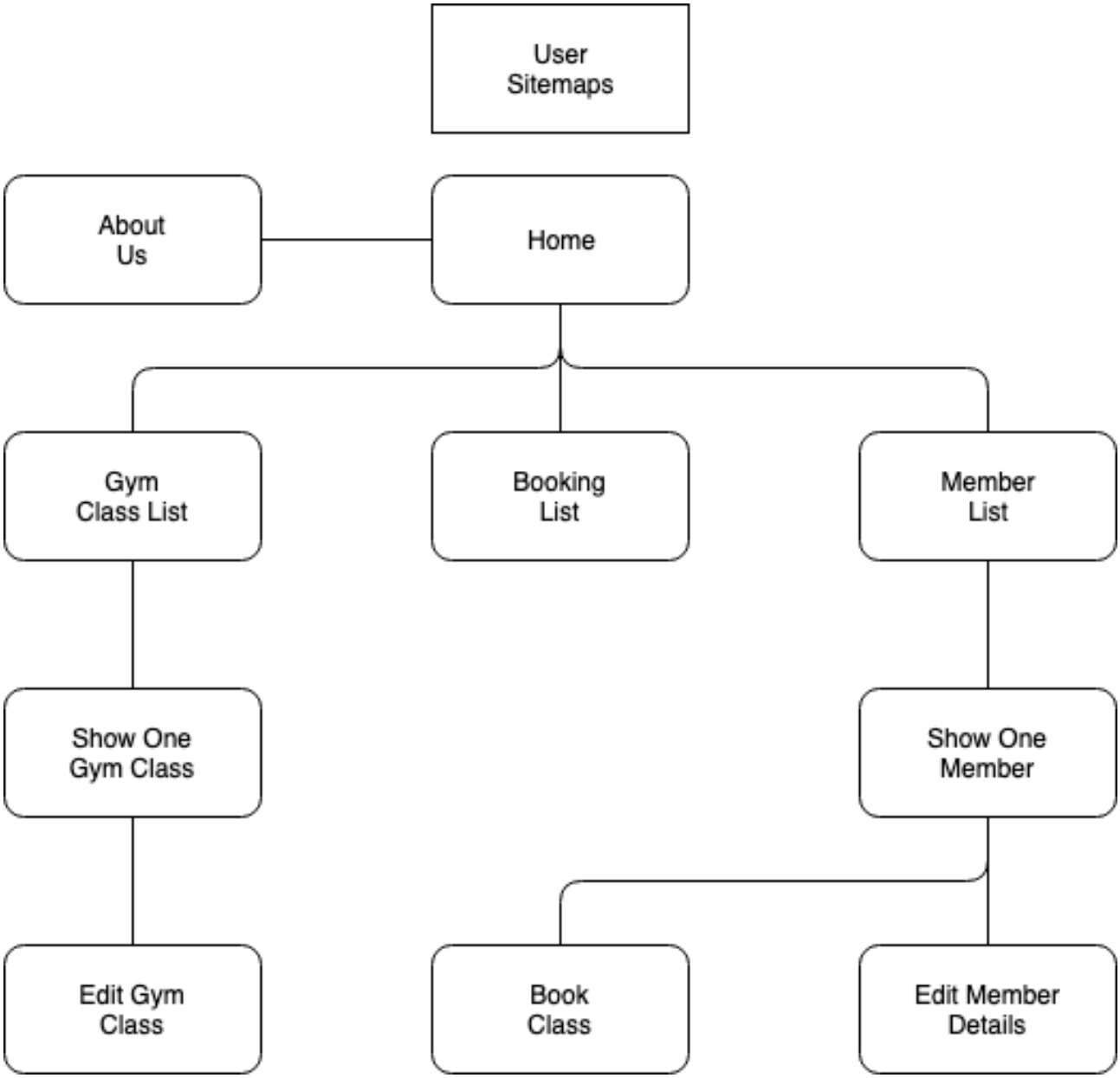
Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time
		Description:

Implementation Constraints Plan

Constraint Category	Implementation Constraint	Solution
Hardware and Software Platforms	The app will be designed by testing on a laptop display. The layout of the elements on the page won't be optimised for smaller sized displays like on tablets or mobile phones.	The way to solve this issue would be to test the layout on the smaller displays of those devices.
Performance Requirements	There could be an issue with scalability in that over time the system may have more entries than it was initially tested with. The system should not show a reduction in response time due to larger workloads.	There would need to be testing done for scalability specifications. Either additional hardware would need to be brought in for the tests or simulation models would need to be implemented.
Persistent Storage and Transactions	Persistent storage becomes a problem the longer the lifecycle of the product. As more and more entries are added to the database. The initial system used may not be suitable as time passes.	The solution would be to add more hardware or upgrade the hardware as the limits are being reached. A limit would be needed on the number of transactions to keep on the system.
Usability	The app may have issues if the design and layout is not similar to other apps. Users nowadays expect a certain standard layouts with apps of this nature. Users do not like learning new ways for just one app.	Ensure that the layout is one that is familiar to the users. The structure and flow of the app should be intuitive to the users. Keep the menus and links simple.
Budgets	Our budget is initially limited to what our investors have provided. In this case it is mainly ourselves that are the investors for this app. The client has provided an initial deposit for us to begin development.	Identify changes to the project early on preferably during the planning phase. Set realistic project goals. Any additional features that are not vital to the project should be considered expendable.
Time Limitations	The project is required to be completed in a period of one week. A presentation has been scheduled to showcase the main features requested along with any additional ones we manage to add.	Good planning and time management should ensure that the project will be completed within this time.

The above implementation constraint plan was used to analyse the possible problems which may be encountered during the project build for the gym management app along with any solutions to overcome them.

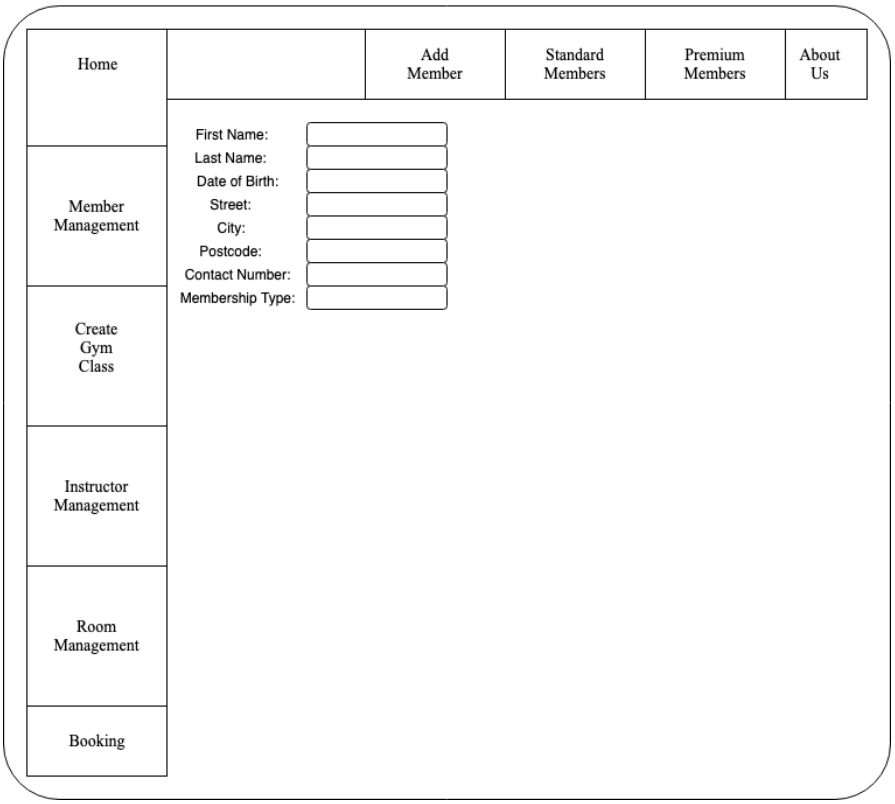
Unit	Ref	Evidence
P	P.5	User Site Map
		Description:



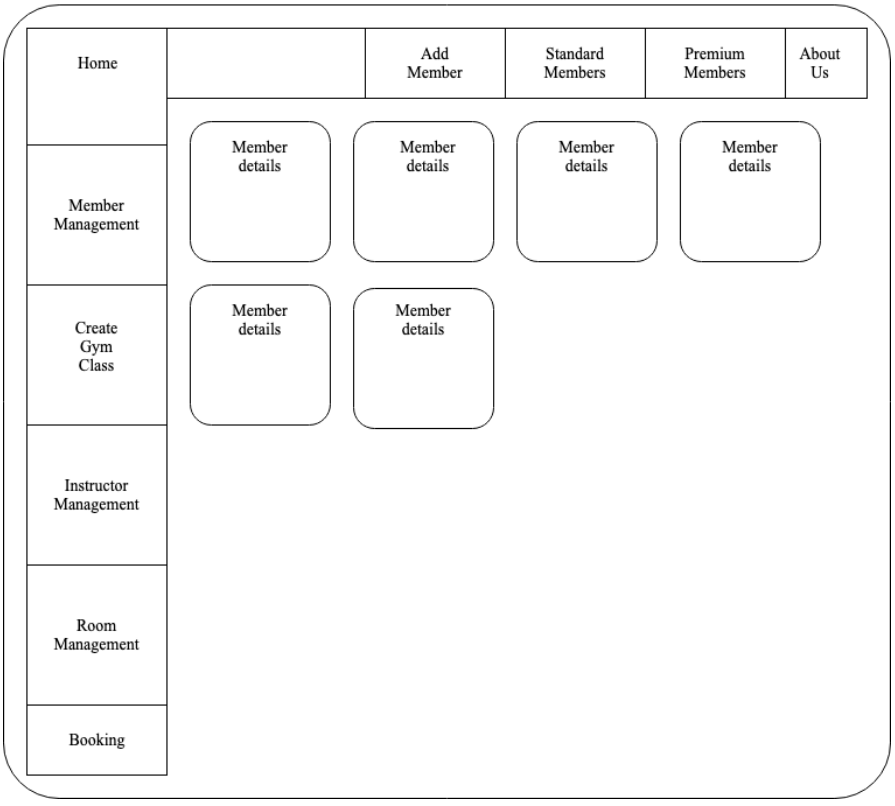
The above shows the user sitemap used in the design of the gym management web app. It shows the paths to reach each of the screens that will be implemented in the final app design. This can be used to analyse the usability of the app by seeing how many steps it would take to reach each view of the app.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams
		Description:

Web app wireframe model



Web app page layout



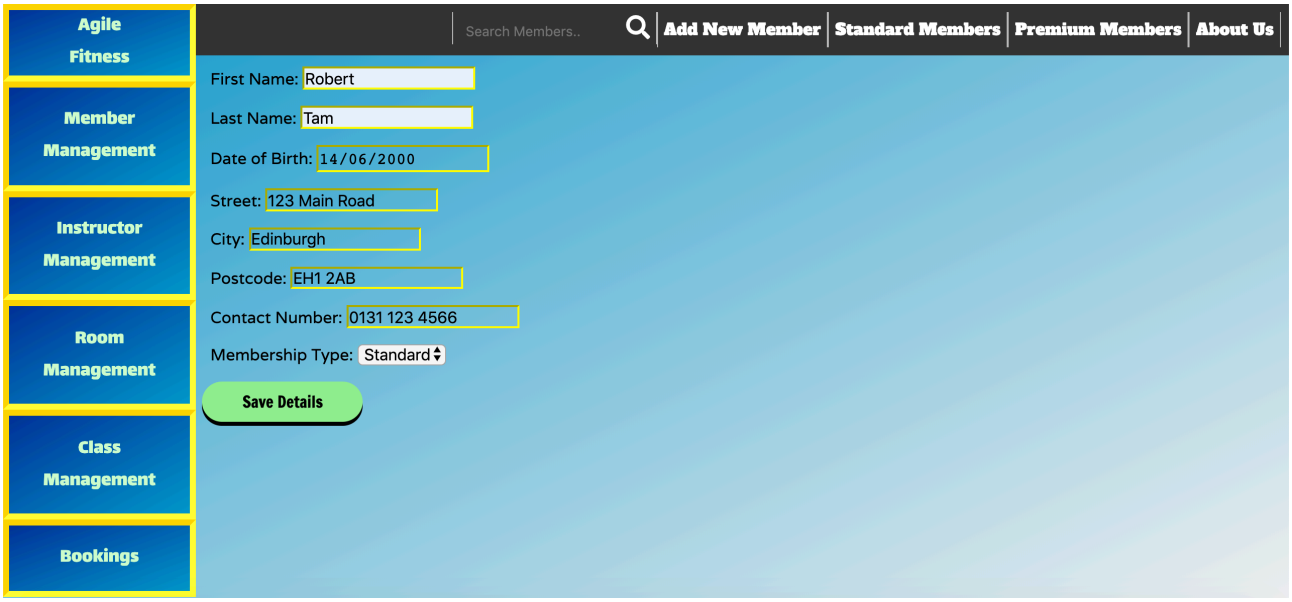
The two wireframes above detail the look of two pages that will be implemented in the gym management web app. The first diagram is a representation of the page used to add a member's details into the gym management app. The second shows the view of a list of all members currently stored in the app.

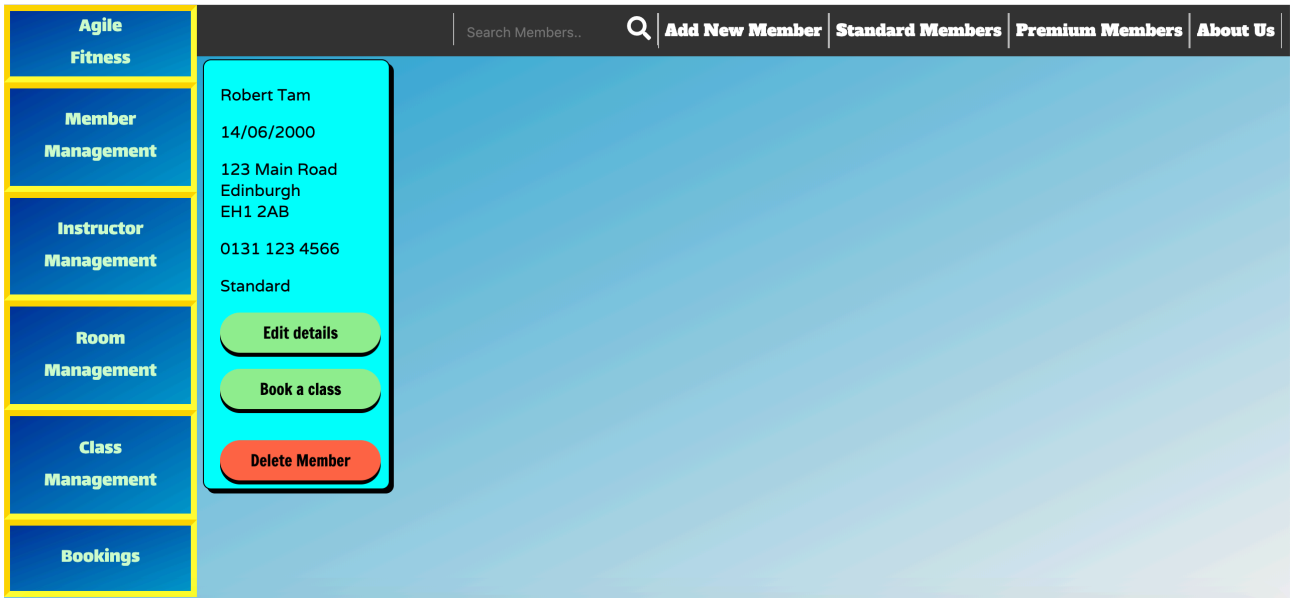
Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method
		Description:

```
- Declare function mostPopularDino() to find the most popular dinosaur
- Declare variable mostGuests and set starting value to zero
- Declare variable mostPopular, keep it undefined as we will use
  it to store the dino object which is most popular
- For every element (dinosaur) in the dinosaur list
- Check if the dinosaur's attribute guestsAttractedPerDay is higher
  than the current value of mostGuests
- If yes assign mostGuests to be the current dinosaur's guestsAttractedPerDay
  attribute to be used for next dinosaur's comparison
- Also assign mostPopular to be the current dinosaur
- Once the for loop finishes return mostPopular
```

The pseudo code above describes the method that will be implemented for looking up the dinosaur, from a list of dinosaurs, which has the most visitors. The image shows the variables and the algorithm which will be used to find the most popular dinosaur.

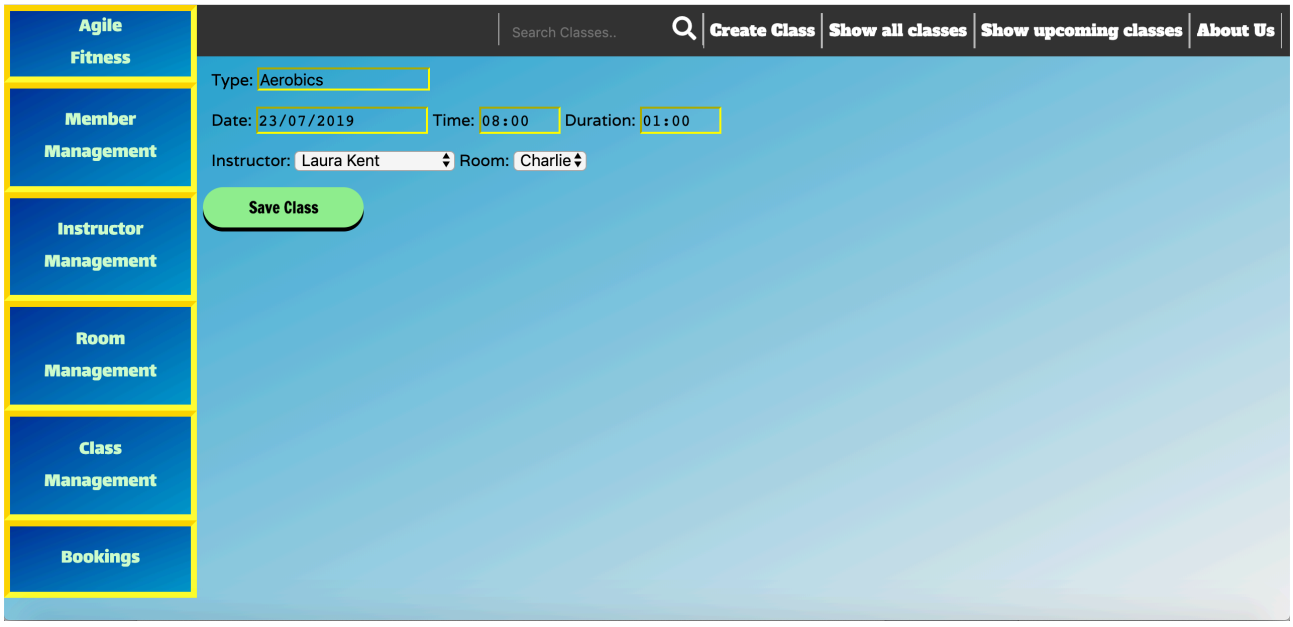
Unit	Ref	Evidence
P	P.13	Show user input being processed according to design requirements. Take a screenshot of: * The user inputting something into your program * The user input being saved or used in some way
		Description:

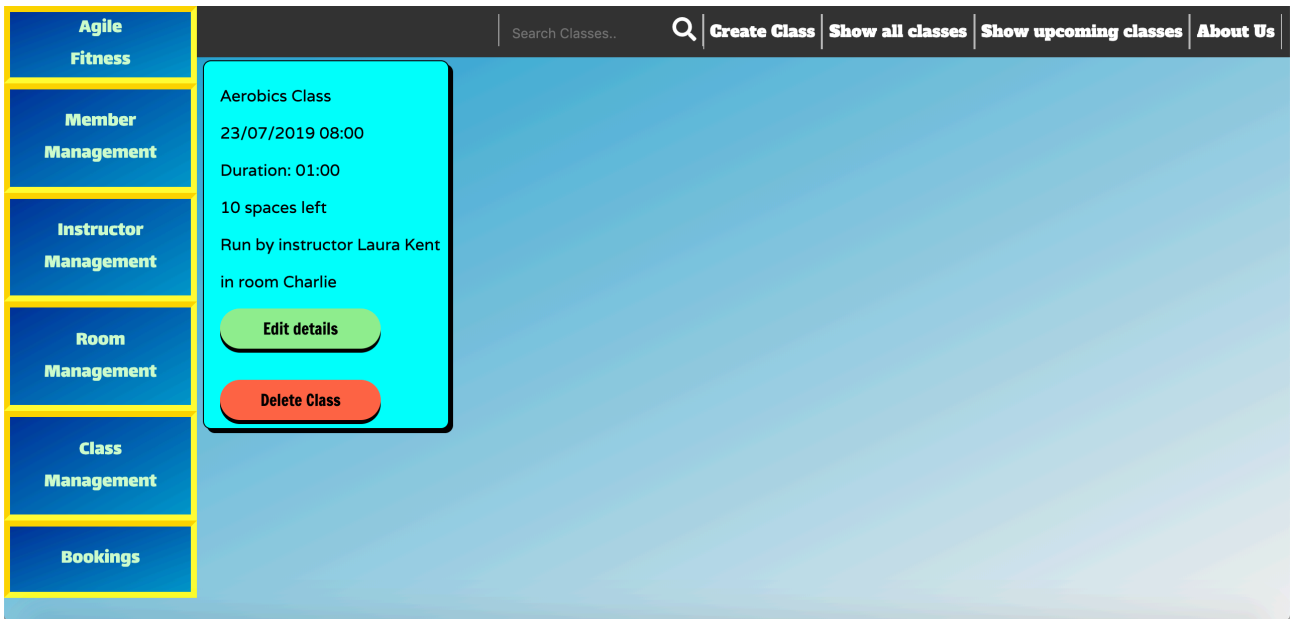




The above two images show some user input being entered into the gym web app and then the data being stored in the app. The first image shows some random personal data entered into the app. The second image is the screen which is shown after the user clicks the save details button.

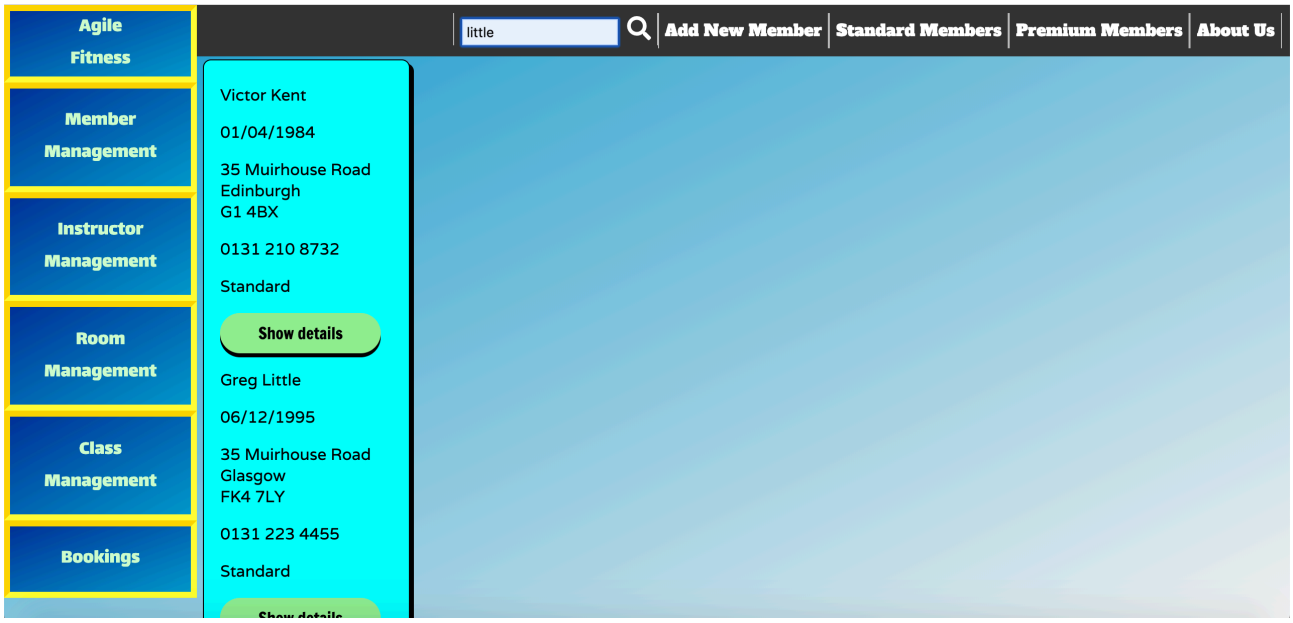
Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved
		Description:

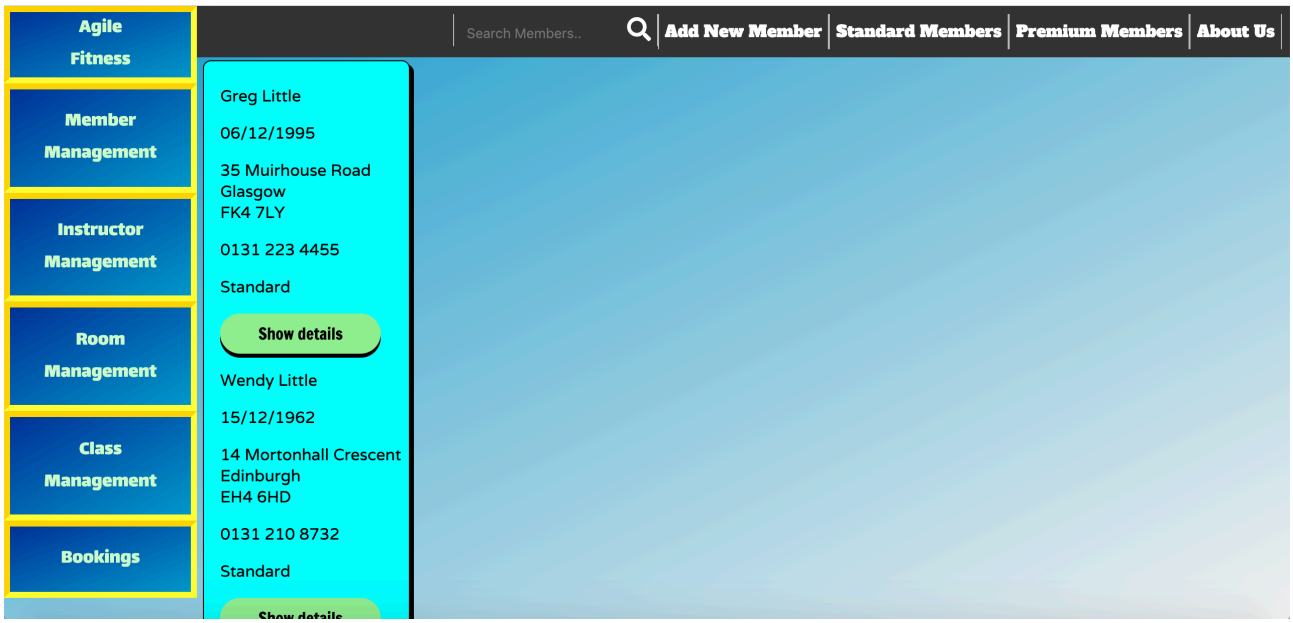




The above images show some random information entered to create a new gym class in my web app. A confirmation is shown when the user clicks save class to inform the user that the new information has been stored in the app's database.

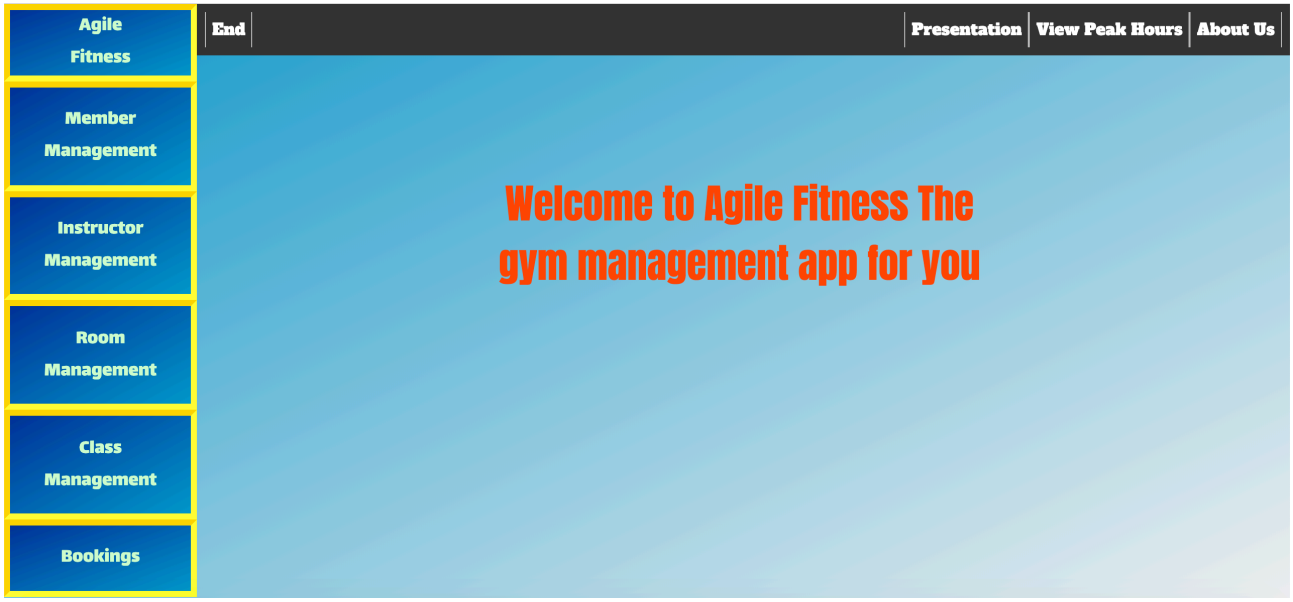
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program
		Description:





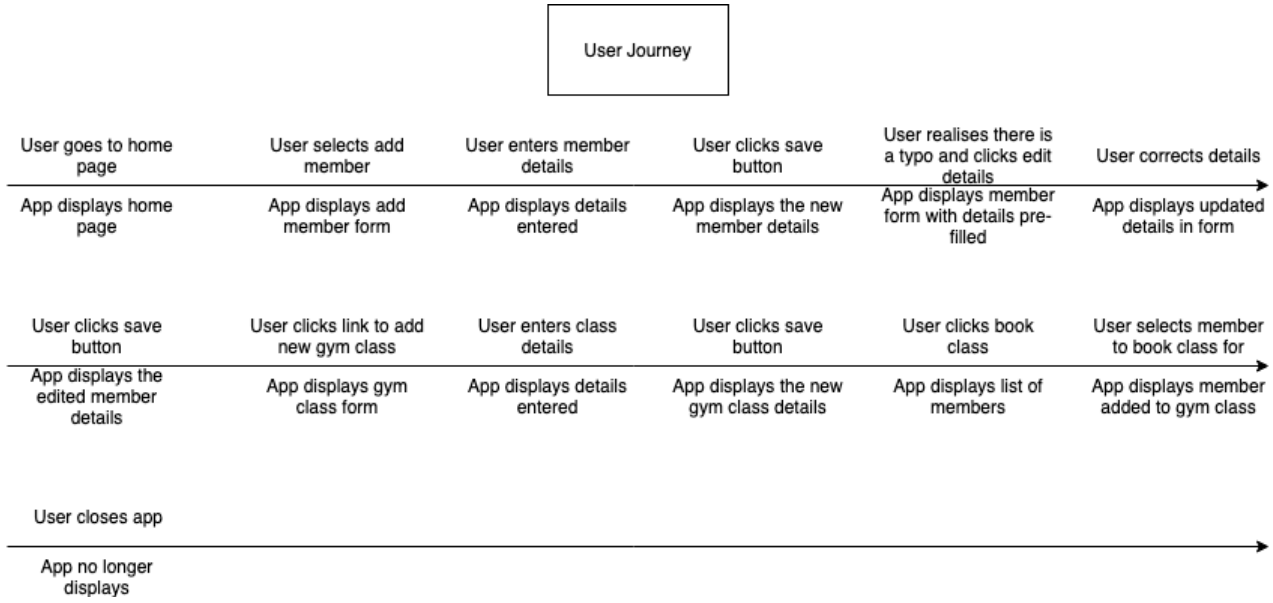
The above images show a user searching for members with the string little in their name using the search bar at the top. The results of the search are then displayed in the following image. The search ignores case sensitivity.

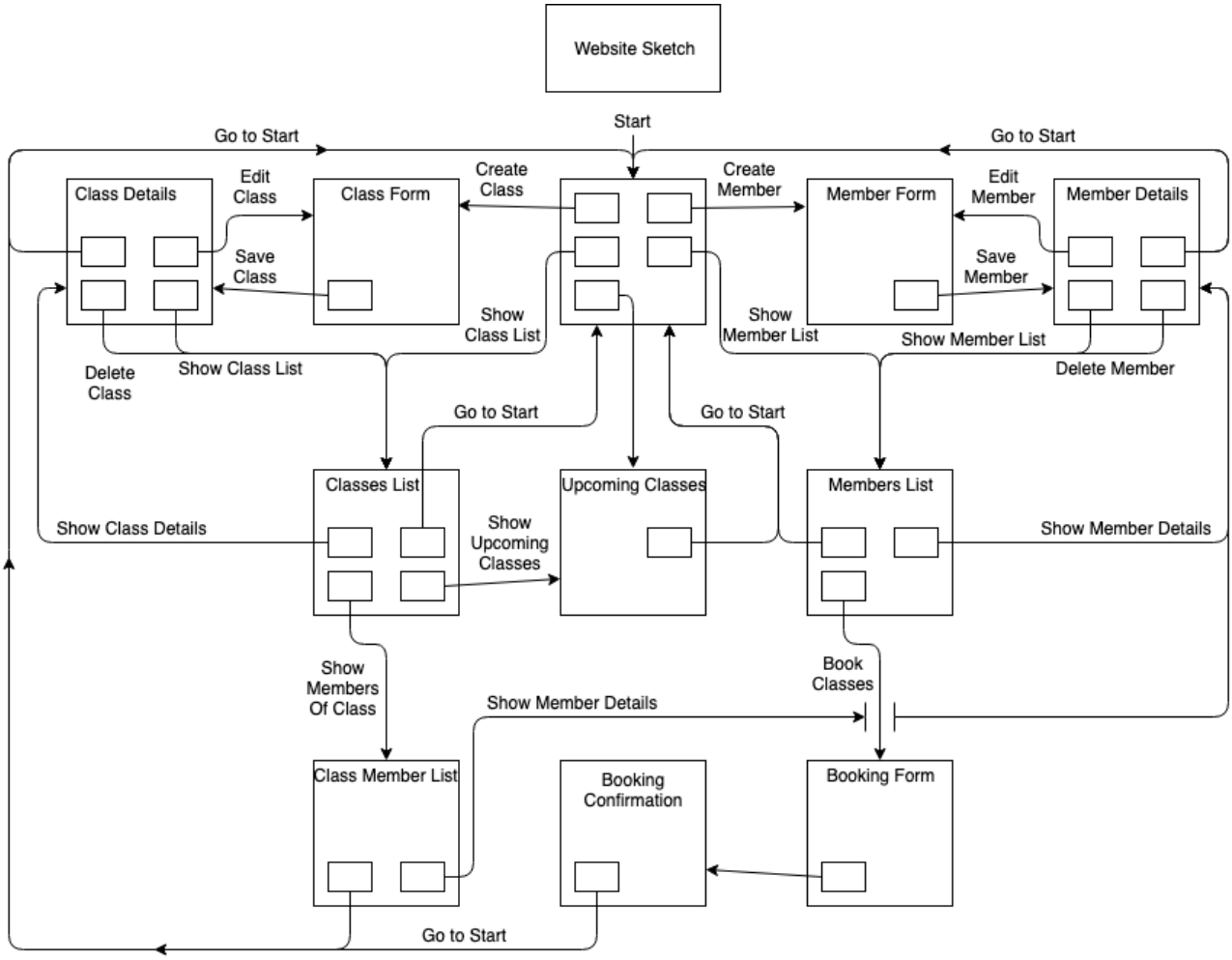
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.
		Description:



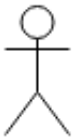
The image is the front page of my gym management app for my solo project. The app was written in Ruby and uses the Sinatra framework. The GitHub link to the project is https://github.com/gadgetguy82/gym_project

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





Proto Persona

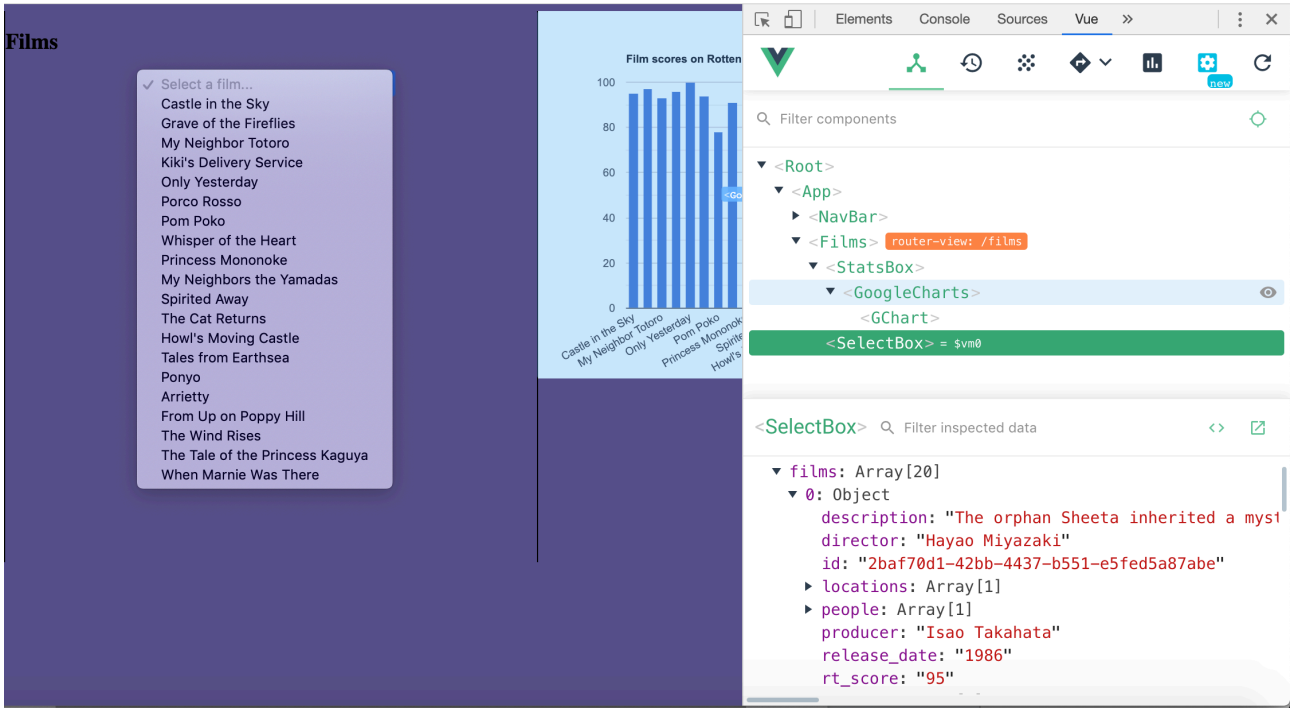
Personal/Demographic	Behavioural
 Gary Sparrow 45 year old male, married with two children. Gym owner/manager, worked in the fitness industry for 10 years. Prior to this worked in the retail sector. Comfortable with using modern technology.	Uses technology daily, but little knowledge on exploring new software solutions that can help with his gym management. He is therefore a little reluctant to try new out new software. Well-organised in all aspects of his life.
Pain Points	**Goals/Motivations**
Current system in his gym is unable to keep up with demand as his gym is expanding in member numbers. System has been the same for the past 8+ years and is slow to respond.	Requires a system that can handle the increased demand. Needs to keep track of member numbers for each gym class. Needs to make sure staff are scheduled for the classes.

The images above show the rest of the methods used to plan out the design of the gym management app. The first diagram is the user journey for how a gym manager might use the app in a daily routine. It describes the user actions and the app responses as the user enters a new member and then spots an error in the member details. The user then corrects the member details before creating a new gym class and booking the member on the new class. The next diagram shows the layout of the links to each view at a basic level. This was used to see if there were going to be any issues in designing the app to this layout. The third diagram is a proto persona used to see what an average user of the management app might be. It is used to see how this user might interact with the app, what the user might need and what issues may arise from using the app.

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
		Description:

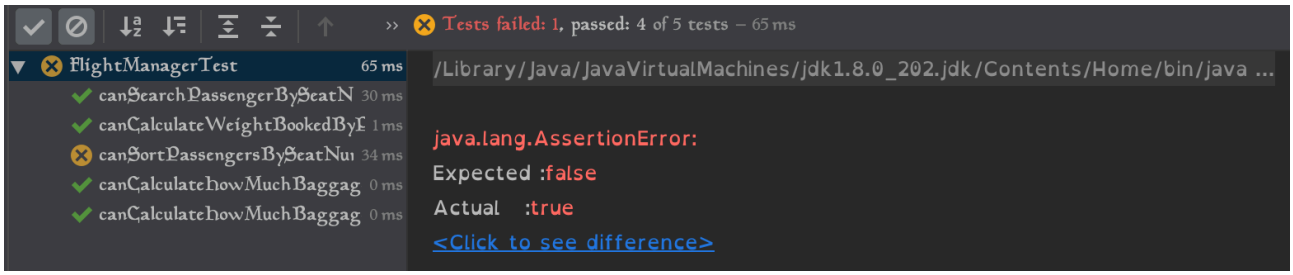
```
mounted() {
  fetch("https://ghibliapi.herokuapp.com/films")
  .then(res => res.json()).then(data => {
    this.films = data
    this.chartSettings.type = "ColumnChart";
    this.chartSettings.data = [["Title", "Rating"]]
    .concat(this.films.map(film => [film.title, parseFloat(film.rt_score)]));
  });

  eventBus.$on("selected-film", film => this.selectedFilm = film);
},
```



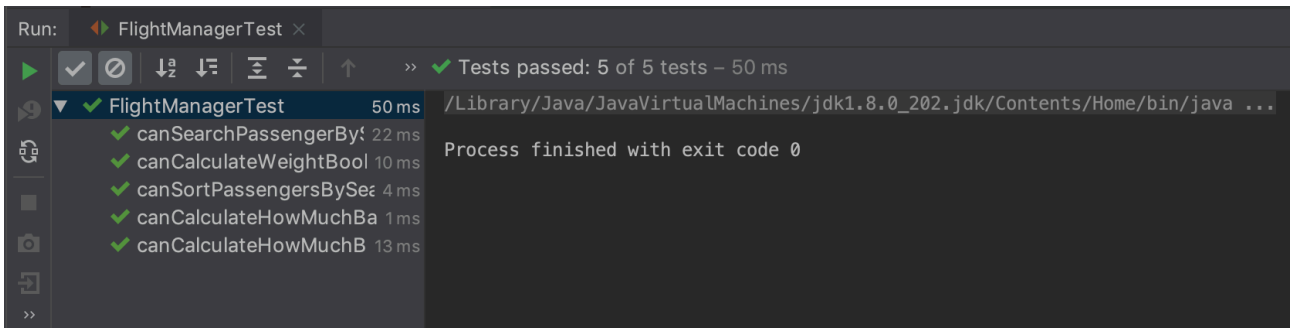
The above images show code which pulls data from the Studio Ghibli API and then extracting the data for use in my app. The Vue extension shows the original data from the API before being filtered in the selection box on the left.

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
		Description:

```java
@Test
public void canSortPassengersBySeatNumbers() {
  flightManager2.bubbleSort(flight2.getPassengers());
  boolean sorted = true;
  for (int i = 0; i < flight2.passengerCount() - 1; i++) {
    if (flight2.getPassenger(i).getSeatNumber() > flight2.getPassenger( index: i + 1).getSeatNumber()){
      sorted = false;
      break;
    }
  }
  assertEquals( expected: false, sorted);
}
```



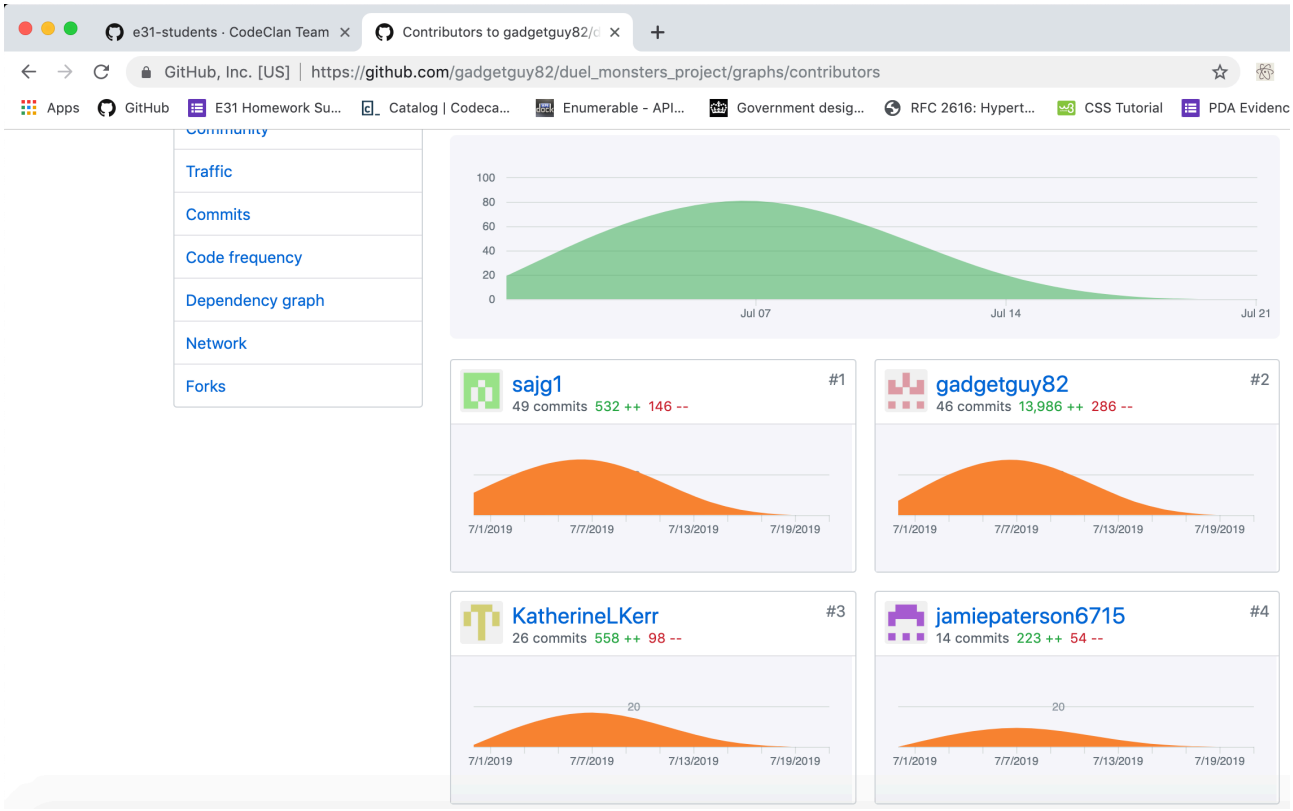

```java
@Test
public void canSortPassengersBySeatNumbers() {
  flightManager2.bubbleSort(flight2.getPassengers());
  boolean sorted = true;
  for (int i = 0; i < flight2.passengerCount() - 1; i++) {
    if (flight2.getPassenger(i).getSeatNumber() > flight2.getPassenger( index: i + 1).getSeatNumber()){
      sorted = false;
      break;
    }
  }
  assertEquals( expected: true, sorted);
}
```



The first image shows a test to check that a function can sort an array of passengers using their seat numbers. The function being tested uses the bubble sort algorithm. The second image shows the canSortPassengersBySeatNumber() test failing as the assertion has the wrong expected value. The third image shows the expected value changed from false to true. The test goes through the array and checks that the first object's seat number is less than the second one. If every object passes this check then sorted remains true and the assertion checks for this value. The last image are of the test code being run after the test is corrected and the test successfully passing.

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.
		Description:



The above is the contributors page for our group project written in JavaScript and using the Vue framework. My username is gadgetguy82 on GitHub. This was for our web app game.

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

Browser Game

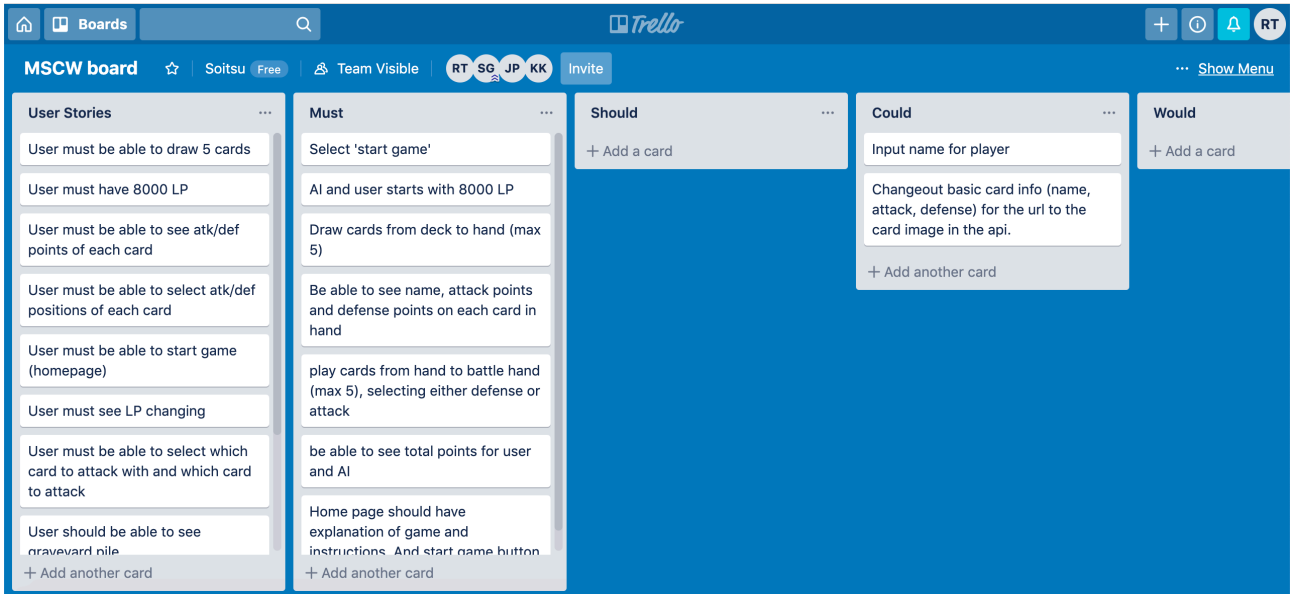
Create a browser game based on an existing card or dice game. Model and test the game logic and then display it in the browser for a user to interact with.

Write your own MVP with some specific goals to be achieved based on the game you choose to model.

You might use persistence to keep track of the state of the game or track scores/wins. Other extended features will depend on the game you choose.

Our chosen project brief was to implement a web browser game. As per the brief we chose to implement a card game which initiates battles by comparing the stats of cards. The game we chose to build was the Yugioh card game which uses monster, trap and spell cards. We set our MVP to have a basic working game using just the normal monster cards. Effects from monsters, traps and spells would be implemented as extensions if we got round to it.

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



Above is the Trello MOSCOW board we used during the planning stage. These were the targets we had to have in our app for a user to be able to play a basic game. We had a few cards in the under the could heading for possible future extensions.

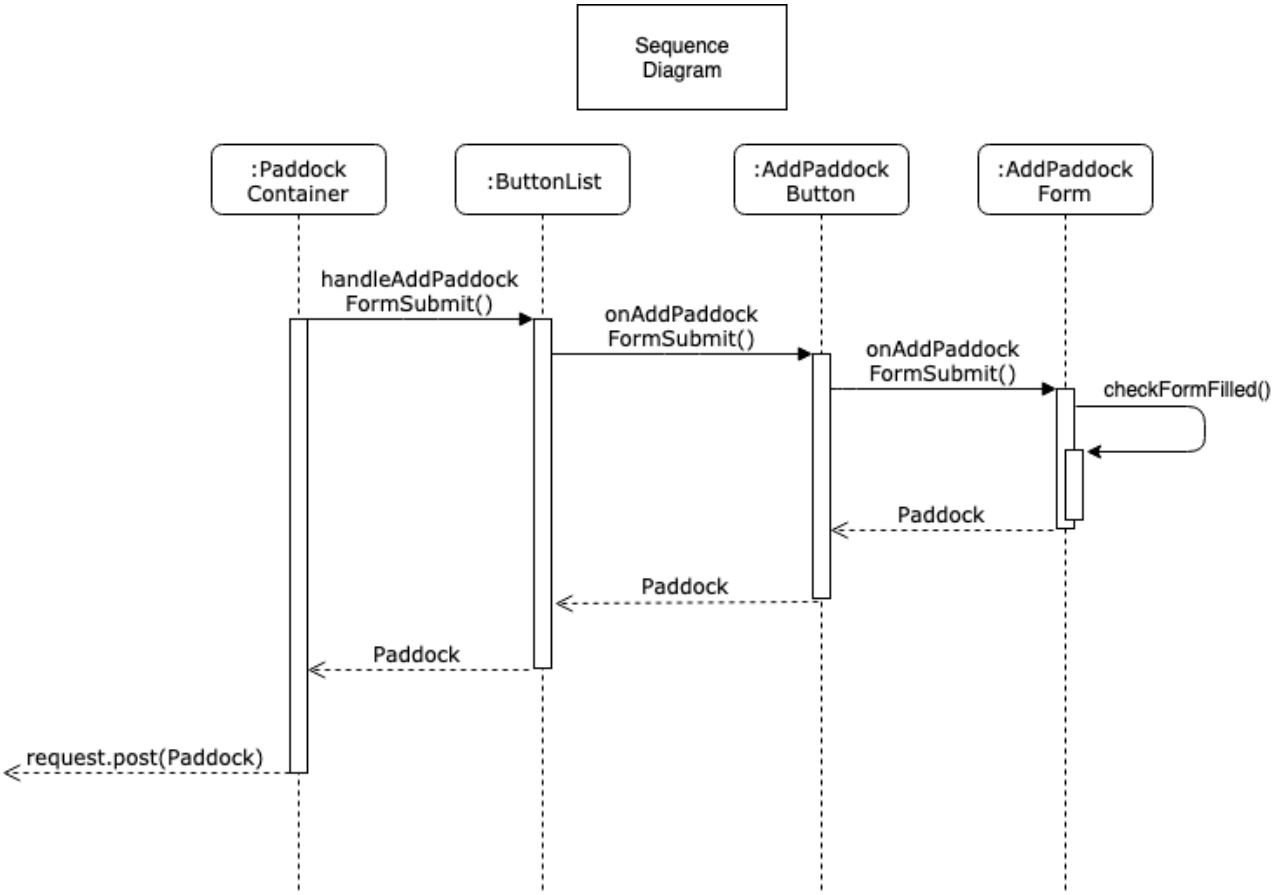
Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.

Acceptance Criteria and Test Plan

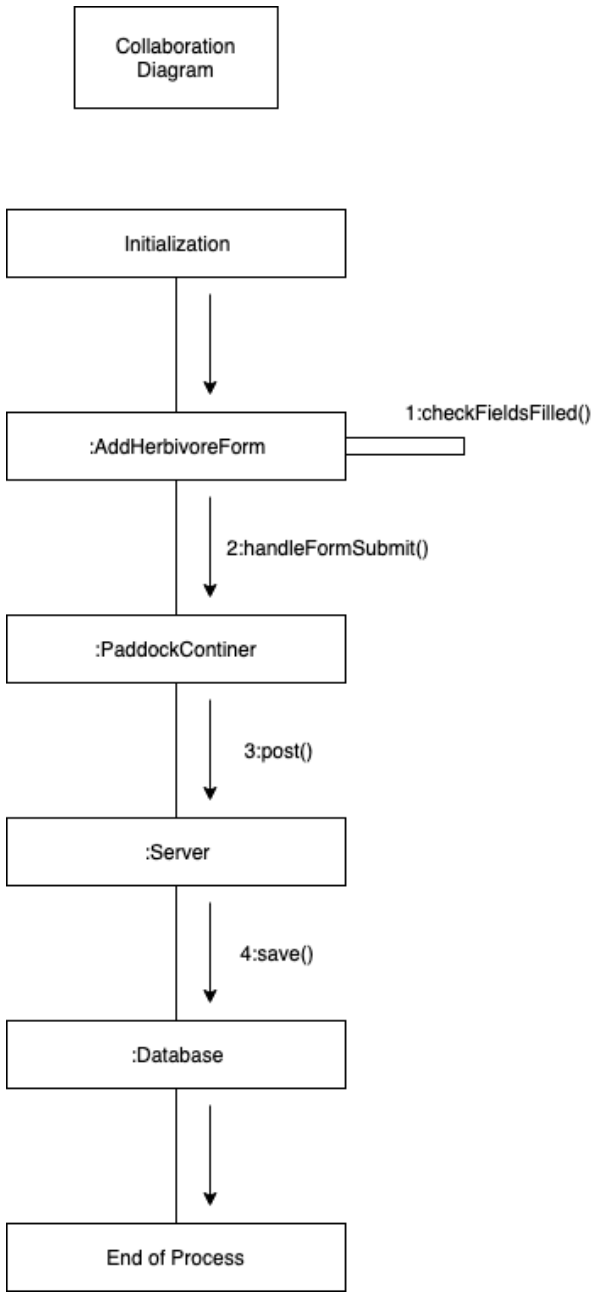
Acceptance Criteria	Expected Result	Pass/Fail
A user is able to draw cards from a playing deck	The app does draw a card from the playing deck into their hand when the user clicks on the playing deck.	Pass
A user is able to set cards to either attack or defence mode	The app does switch to using the attack or defence stats when the user clicks on either the attack or defence button below each card on the battle field.	Pass
A user is able to lower their opponent's life points by attacking with their cards stats	The app does reduce the opponent's life points when a monster card attacks the opponent's life points directly.	Pass
A user is able to see the stats on the cards in play	The app does display the stats of the card when the user hovers over each card with the mouse cursor.	Pass
A user is able to destroy opponent's monsters if their stats are lower and send the destroyed monster is sent to the graveyard	The app does send monster cards to the graveyard when they are involved in a battle and have lower stats than the winner.	Pass
A user is able to hide their hand during their opponent's turn	The app does turn over the cards in the playing hand when the user ends their turn.	Pass

The above acceptance criteria was used on our group project and states most of the functionality we needed for a MVP for the user to be able to play a basic game whereby they can draw cards and attack their opponent with the stats on the cards they put into play.

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

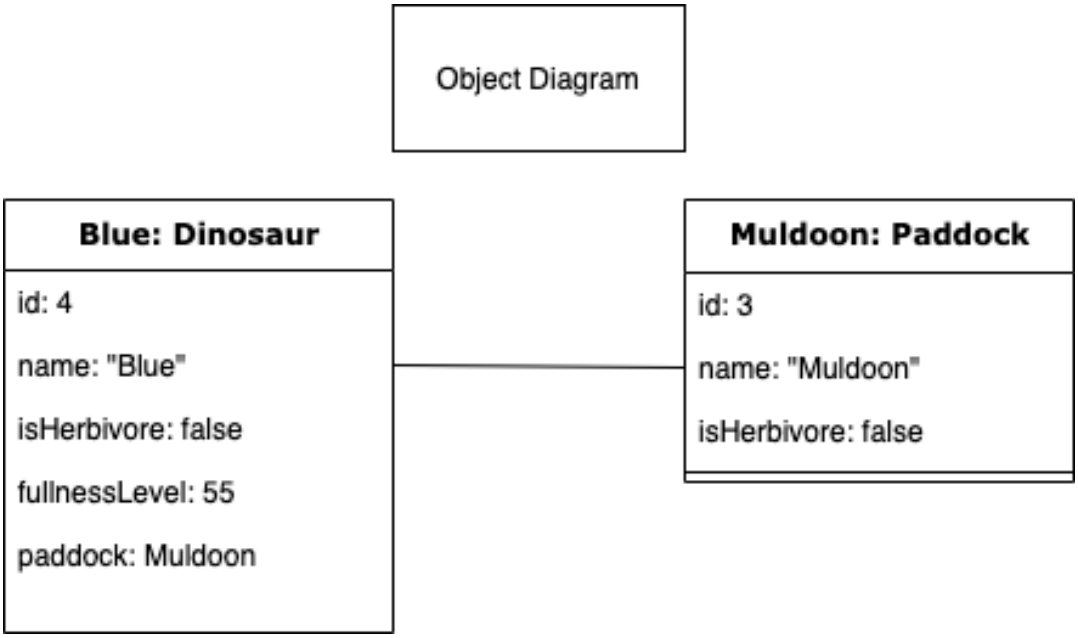


The above diagram is a sequence diagram from our group project to implement a dinosaur park management app. This shows how the handleAddPaddockFormSubmit() method is passed down through the child components to the AddPaddockForm component. After the user fills in the form the app passes the newly created paddock back up through the components in order for the PaddockContainer to post the new paddock to the database.

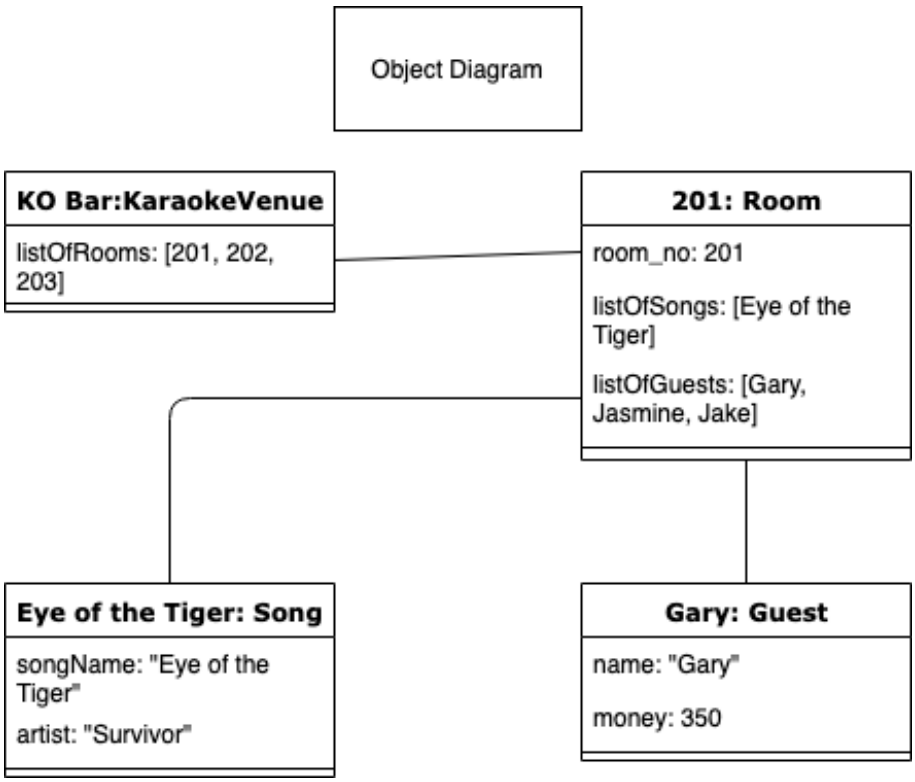


The above collaboration diagram shows the steps for adding a herbivorous dinosaur to the park's management app. It lists the steps and methods used starting from the AddHerbivoreForm which checks that the fields of the form have entries. It then passes this information to the PaddockContainer component which then posts the information to the server allowing the server to store the new dinosaur in the database.

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



The above is an object diagram from our group project. It shows sample data of a dinosaur's information which would available on the app.



The second object diagram models sample data from a Karaoke bar. A Karaoke venue will have list of rooms which in turn will have a list of songs and guests that are in the room.

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

Bug Tracking Report

Bug/Error	Solution	Date
No default constructor for Entity	Add a no parameter default constructor to the Herbivore class and Carnivore class so that Spring can build the models from the data in the database with the class setter methods.	16/08/2019
Error 4415 Cannot render error page for request [/dinosaurs] and exception [Could not write JSON: Infinite recursion StackOverflowError	Add annotations @JsonIgnoreProperties so that Spring will not continue looping through the relational properties of two models.	18/08/2019
Uncaught Type Error: Cannot read property 'map' of undefined	Typo found in variable name of props being passed down to child component. listofPaddocks changed to listOfPaddocks	19/08/2019
Transfer Form not submitting on user clicking button to submit form	The function handleTransferSubmit was not being passed down as props to child component. Added prop to child component.	20/08/2019
PUT http://localhost:8080/herbivores/405	Missing dinosaur.id property on the route. Added to route to access the correct route for updating on server side.	20/08/2019
Cannot read property 'map' of undefined	RemoveDinosaurButton was used in two places, one did not have the list of dinosaurs being passed down as props. Corrected by adding props to the this child component.	21/08/2019

These bugs in the report above were recorded during the last group project. These were the errors I found and then corrected as described in the solutions column above.

Week 12

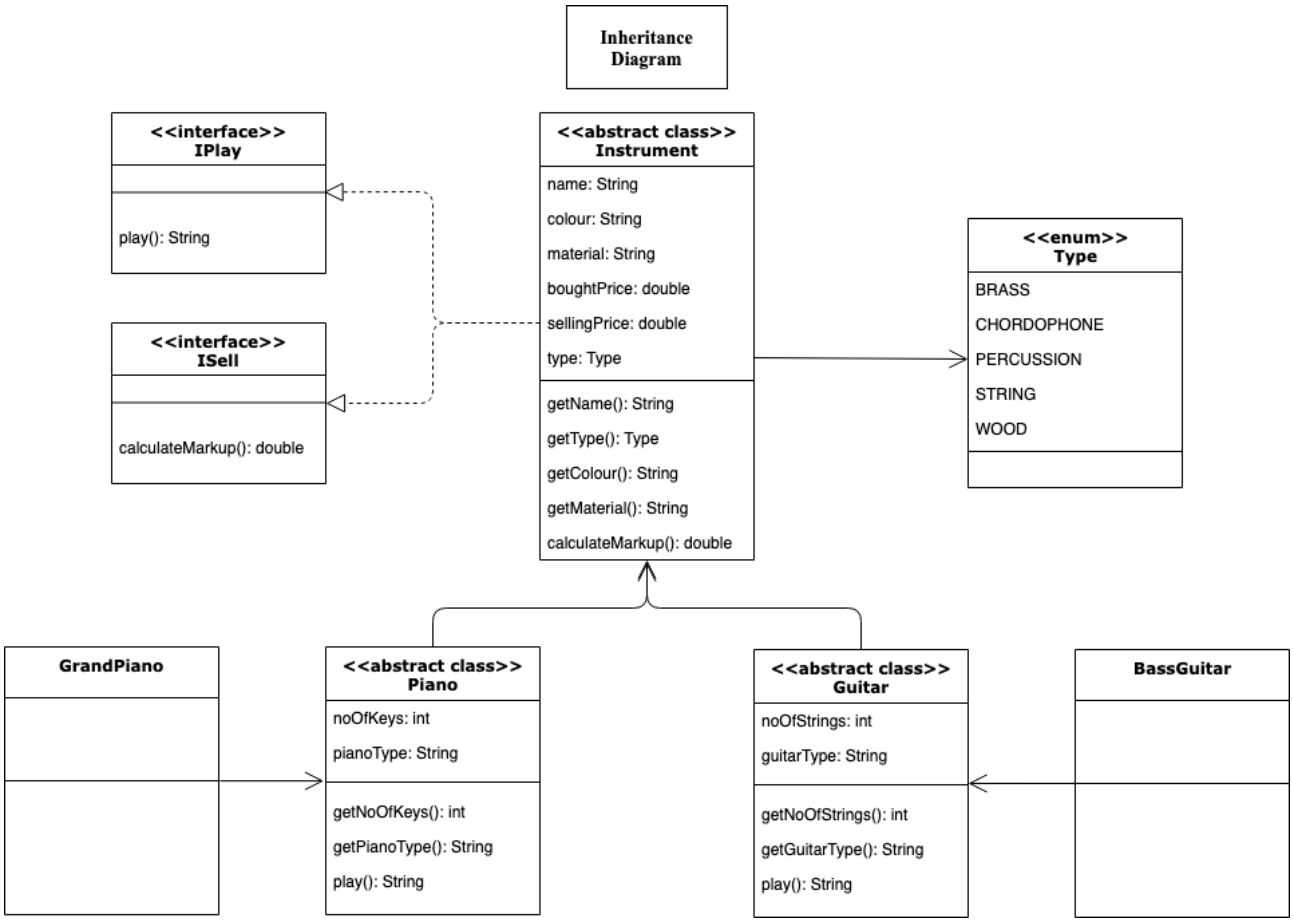
Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.
		Description:

```
public class ShopTest {
  Shop shop;
  BassGuitar bassGuitar;
  GuitarString guitarString;

  @Before
  public void setUp() {
    shop = new Shop( name: "Ray's Music Exchange");
    bassGuitar = new BassGuitar( name: "Fender", colour: "Red", material: "Basswood", noOfStrings: 4);
    guitarString = new GuitarString( description: "Guitar String", boughtPrice: 10.00, sellingPrice: 15.99);
  }
```

```
  @Test
  public void canRemoveItemsFromStock() {
    shop.addStock(bassGuitar);
    shop.addStock(guitarString);
    shop.removeStock(guitarString);
    assertEquals( expected: 1, shop.stockCount());
  }
```

The above shows an example of the use of polymorphism. Here bassGuitar is a BassGuitar object which inherits from abstract class Guitar and implements the ISell interface. guitarString is a GuitarString object which inherits from abstract class Accessory and also implements the ISell interface. This means that both these objects are of type ISell along with the classes they are an object of and inherited from. This means they can both be added to an array which takes ISell objects as can be seen in the test above where both are added to stock array of type ISell in the shop object.

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



This is the inheritance diagram used to plan out an implementation of the music instrument shop class. Here I have an abstract class Instrument from which all other instruments will inherit from and the properties and methods which will be associated with an instrument. All instruments will therefore inherit the two interfaces, IPlay and ISell and will be of those types. I have two child classes of Instrument, Guitar and Piano which are themselves abstract and then further child classes inheriting from them.

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

```
public class Passenger {
  private String name;
  private int noOfBags, seatNumber;
  private Flight flight;

  public Passenger(String name, int noOfBags) {
    this.name = name;
    this.noOfBags = noOfBags;
  }

  public String getName() { return this.name; }

  public int getNoOfBags() { return this.noOfBags; }

  public Flight getFlight() { return this.flight; }

  public int getSeatNumber() { return this.seatNumber; }

  public void addFlight(Flight flight) {
    this.flight = flight;
    this.seatNumber = this.flight.assignSeat();
  }
```

The above image shows encapsulation in a piece of code, the attributes are hidden by declaring the access modifiers as private. These attributes can be retrieved by the getter methods and set by the constructor and the addFlight method.

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.
		Description:

```java
import behaviours.ISell;

public abstract class Accessory implements ISell {
  private String description;
  private double boughtPrice, sellingPrice;

  public Accessory(String description, double boughtPrice, double sellingPrice) {
    this.description = description;
    this.boughtPrice = boughtPrice;
    this.sellingPrice = sellingPrice;
  }

  public String getDescription() { return this.description; }

  public double getBoughtPrice() { return this.boughtPrice; }

  public void setBoughtPrice(double boughtPrice) { this.boughtPrice = boughtPrice; }

  public double getSellingPrice() { return this.sellingPrice; }

  public void setSellingPrice(double sellingPrice) { this.sellingPrice = sellingPrice; }

  public double calculateMarkup() {
    return Math.round((this.sellingPrice - this.boughtPrice) * 100.0) / 100.0;
  }
}
```

```java
public class GuitarString extends Accessory {
  public GuitarString(String description, double boughtPrice, double sellingPrice) {
    super(description, boughtPrice, sellingPrice);
  }
}
```

The above images show an abstract class Accessory and the child class GuitarString inheriting the Accessory class.

```java
public class Shop {
  private String name;
  private ArrayList<ISell> stock;

  public Shop(String name) {
    this.name = name;
    this.stock = new ArrayList<ISell>();
  }

  public String getName() { return this.name; }

  public int stockCount() { return stock.size(); }

  public void addStock(ISell item) { this.stock.add(item); }

  public void removeStock(ISell item) { this.stock.remove(item); }

  public double totalPotentialProfit() {
    double total = 0.0;
    for (ISell item : this.stock) {
      total += item.calculateMarkup();
    }
    return total;
  }
```

```java
public class ShopTest {
  Shop shop;
  BassGuitar bassGuitar;
  GuitarString guitarString;

  @Before
  public void setUp() {
    shop = new Shop( name: "Ray's Music Exchange");
    bassGuitar = new BassGuitar( name: "Fender", colour: "Red", material: "Basswood", noOfStrings: 4);
    guitarString = new GuitarString( description: "Guitar String", boughtPrice: 10.00, sellingPrice: 15.99);
  }

  @Test
  public void canCalculateTotalPotentialProfit() {
    bassGuitar.setBoughtPrice(60.99);
    bassGuitar.setSellingPrice(89.99);
    guitarString.setSellingPrice(14.00);
    shop.addStock(bassGuitar);
    shop.addStock(guitarString);
    assertEquals( expected: 33.00, shop.totalPotentialProfit(), delta: 0.01);
  }
```

In the test above we show guitarString using the setSellingPrice method from its parent class, Accessory. After the setting the new selling price for guitarString we test that the method totalPotentialProfit() sees the new price and uses it for calculating potential profit instead of the sellingPrice set in the constructor.

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.
		Description:

```ruby
def my_flatten(array)
  final_array = []

  for element in array
    if (element.is_a?(Array))
      final_array += my_flatten(element)
    else
      final_array << element
    end
  end
  return final_array
end
```

The first algorithm I have chosen is the flatten function which is used to reformat an array which has nested arrays into just an array with all the elements of itself and the previous nested arrays. This function removes all nested arrays from the outermost array but keeps all the elements within.

```java
public static String capitalise(String string) {
  String[] wordArray = string.toLowerCase().split( regex: " ");
  String[] capitalisedWords = new String[wordArray.length];
  for(int i = 0; i < wordArray.length; i++) {
    String firstLetter = wordArray[i].substring(0, 1).toUpperCase();
    capitalisedWords[i] = firstLetter + wordArray[i].substring(1);
  }
  return String.join( delimiter: " ", capitalisedWords);
}
```

The second algorithm I have chosen is the capitalise function which is used to capitalise every word in a string. The method first turns all words to lowercase by using String's toLowercase() method and then using the split() method on a whitespace and assigning the result to wordArray. The method then uses a for loop to go through each word in wordArray and then takes the first letter as a substring on which toUpperCase() is called on. The capitalised letter is then concatenated back with the rest of the word and stored in a new array capitalisedWords. The method finally returns the words in capitalisedWords joined by a whitespace in between as one String.