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

Screenshot:

```
33  array_of_numbers = [1,2,3,4,5,6,7,8,9,16,36,15]
34
35  def calculate_total(numbers)
36    total = 0
37    for number in numbers
38      total = total + number
39    end
40    return total
41  end
42
43  p "This is the sum of the array elements: #{calculate_total(array_of_numbers)}"
```

```
➜  lesson ruby quiz.rb
"This is the sum of the array elements: 112"
```

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:

Screenshots:

```ruby
margaret = {name: "Margaret", age: 2, eggs: 3}
hetty = {name: "Hetty", age: 1, eggs: 2}
henrietta = {name: "Henrietta", age: 3, eggs: 1}
audrey = {name: "Audrey", age: 2, eggs: 4}
mabel = {name: "Mabel", age: 5, eggs: 1}

chickens = [margaret,hetty,henrietta,audrey,mabel]

def find_animal_by_name(animals,name)
  found = false
  for animal in animals
    if animal[:name]==name
      found = true
    end
  end
  return "The animal was found: #{found}."
end

p find_animal_by_name(chickens,"James")
```

I.T.6 1/2: **hash and function that uses it**

```
➜ lesson ruby loops_in_functions.rb
"The animal was found: false."
```

I.T.6 2/2 **execution result in terminal.**

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

Screenshot:

```ruby
def Property.find_by_id(id)
  db = PG.connect({dbname: "property_tracker", host: "localhost"})
  sql = "SELECT * FROM houses WHERE id = $1"
  values = [id]
  db.prepare("find_by_id",sql)
  house_array = db.exec_prepared("find_by_id",values)
  db.close()
  house = house_array[0]
  p house_array
  return Property.new(house)
end
```

I.T.3 1/2 SEARCH FUNCTION

```
The address you are searching for is:
#<Property:0x007feef19e5c18 @address="2/1
29 Hope Street Glasgow", @value="135000",
@bedrooms="3", @build="flat", @id=67>
➜  agency
```

I.T.3 2/2 OBJECT FOUND BY THE FUNCTION

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: PSQL selection: contents of a table are shown ordered by one specified column value.

Screenshot

```ruby
def self.all_by_quantity()
  db = PG.connect({ dbname: 'pizza_shop', host: 'localhost' })
  sql = "SELECT * FROM pizza_orders order by quantity"
  db.prepare("all", sql)
  orders = db.exec_prepared("all")
  db.close()
  return orders.map { |order| PizzaOrder.new(order) }
end
```

I.T.4 1/2 ORDER A TABLE BASED ON A COLUMN VALUE

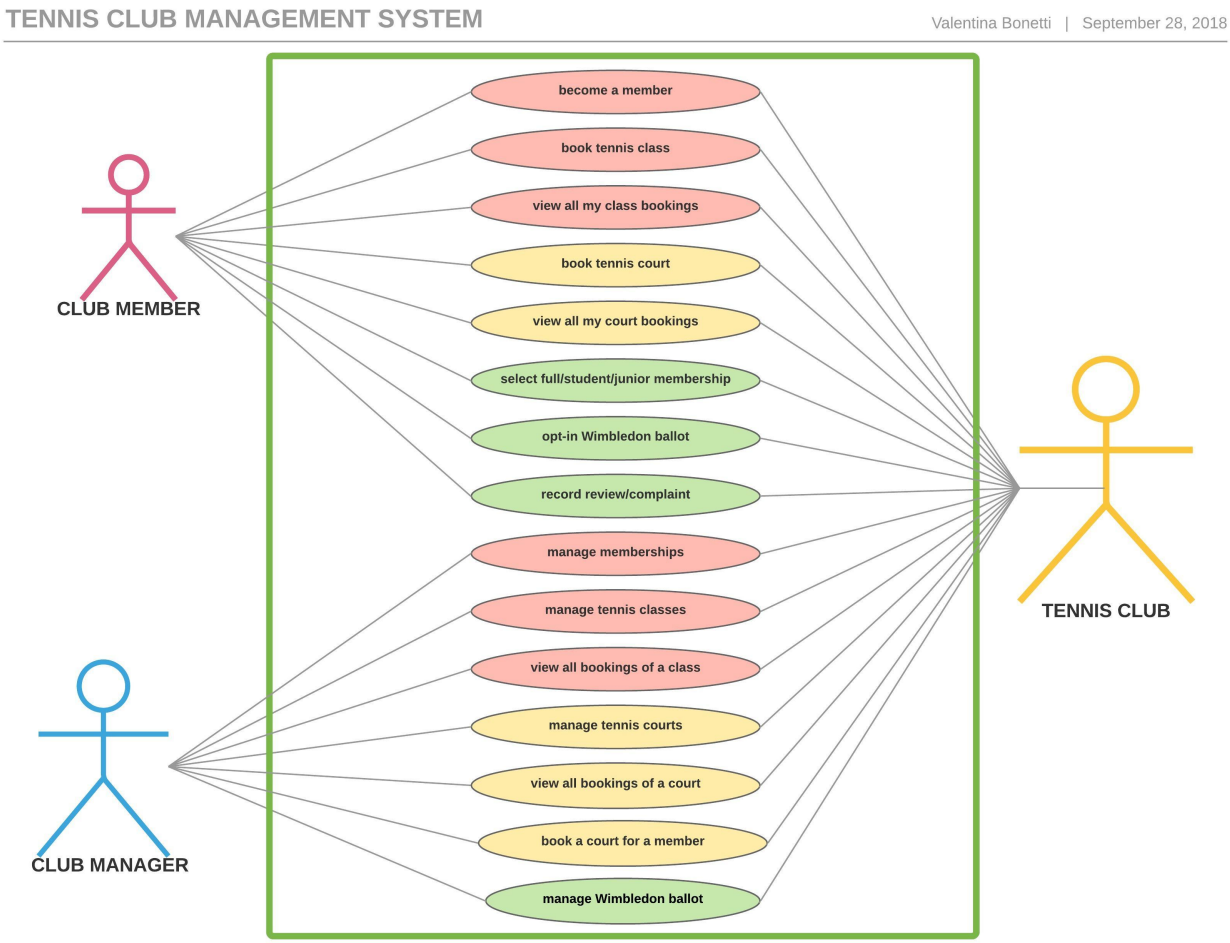
```
 id |  topping  | quantity | customer_id
----+-----------+----------+-------------
 16 | bufala    |        1 |           6
 17 | anchovies |        1 |           6
 14 | capperi   |        3 |           6
 15 | peppers   |        5 |           6
(4 rows)

pizza_shop=#
```

I.T.4 2/2 OUTPUT: PIZZA ORDERS ARE ORDERED BY QUANTITY RATHER THAN ID

Week 5 and 6

Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram
		Description: Tennis club management app, use case diagram

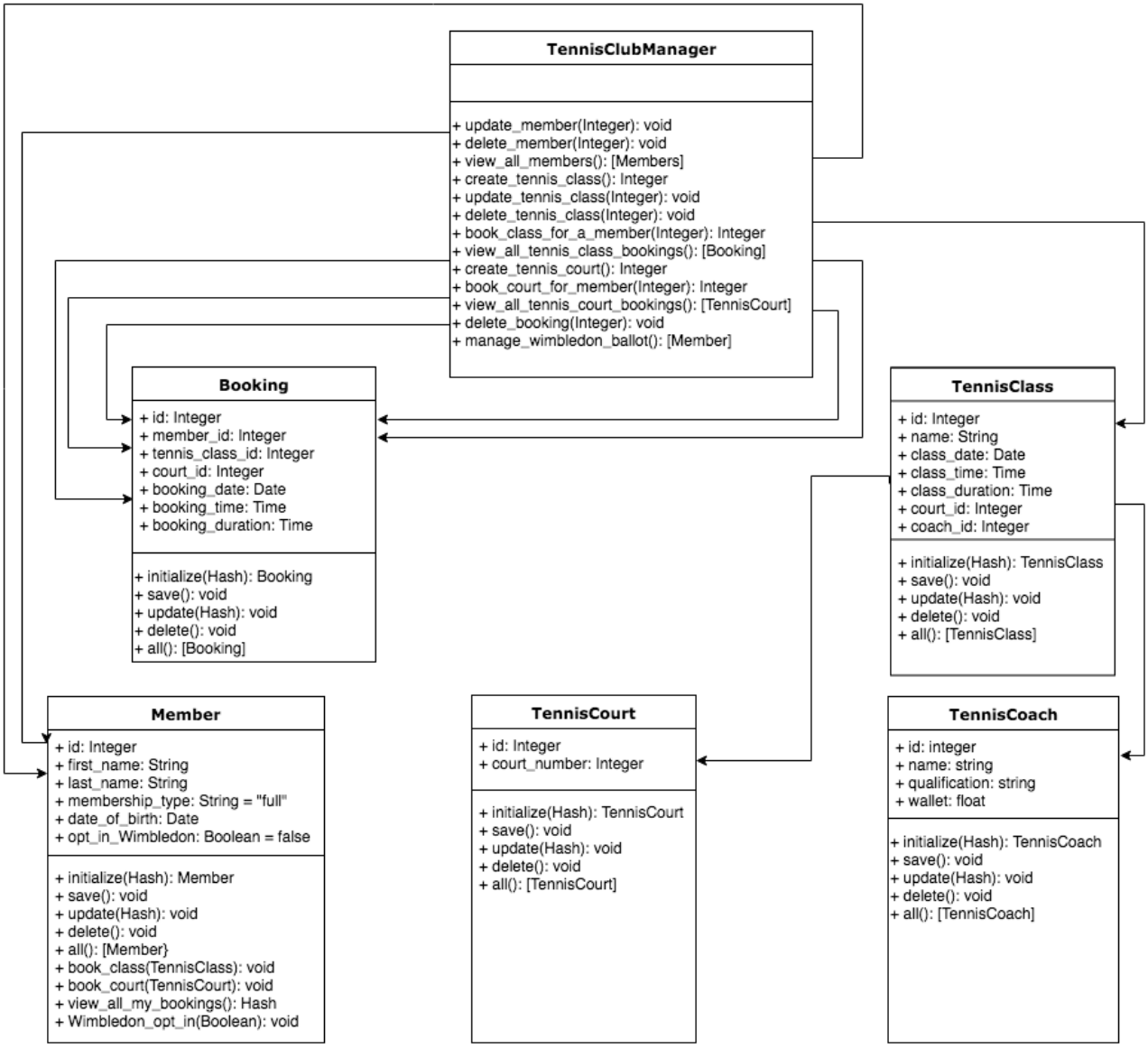
Screenshot:



A.D.1 WEEK 5 PROJECT: TENNIS CLUB MANAGEMENT SYSTEM

Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram
		Description: Tennis club management app, UML class diagram

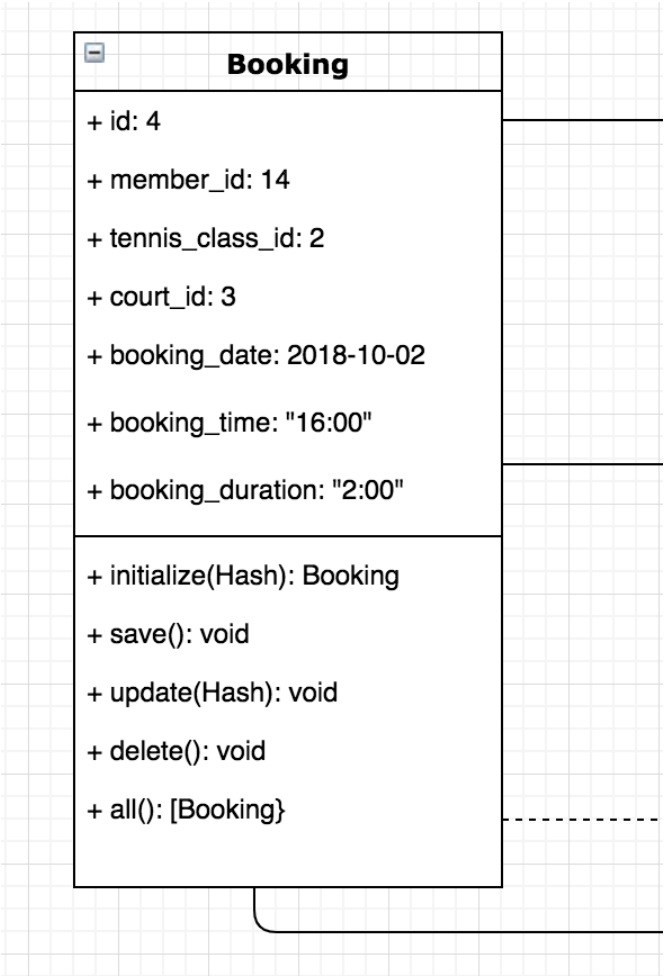
Screenshot:



A.D.2 WEEK 5 PROJECT: TENNIS CLUB MANAGEMENT SYSTEM, CLASS DIAGRAM.

Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram
		Description: UML diagram of one of the tennis club management app objects

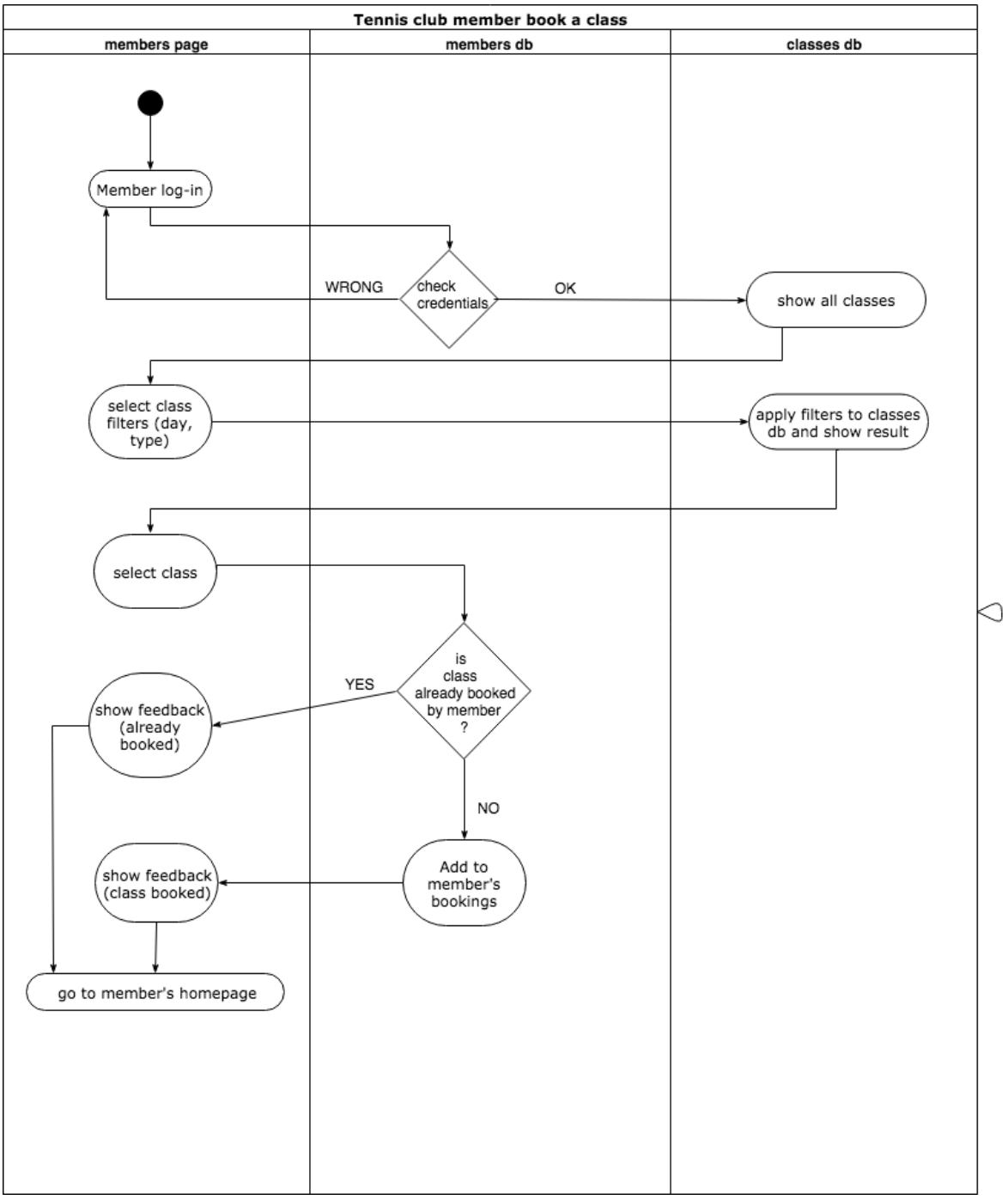
Screenshot:



A.D.3 A BOOKING OBJECT FROM THE TENNIS CLUB MANAGEMENT SYSTEM

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

Screenshot:



A.D.4 ACTIVITY DIAGRAM OF A TENNIS CLUB MEMBER BOOKING A TENNIS CLASS

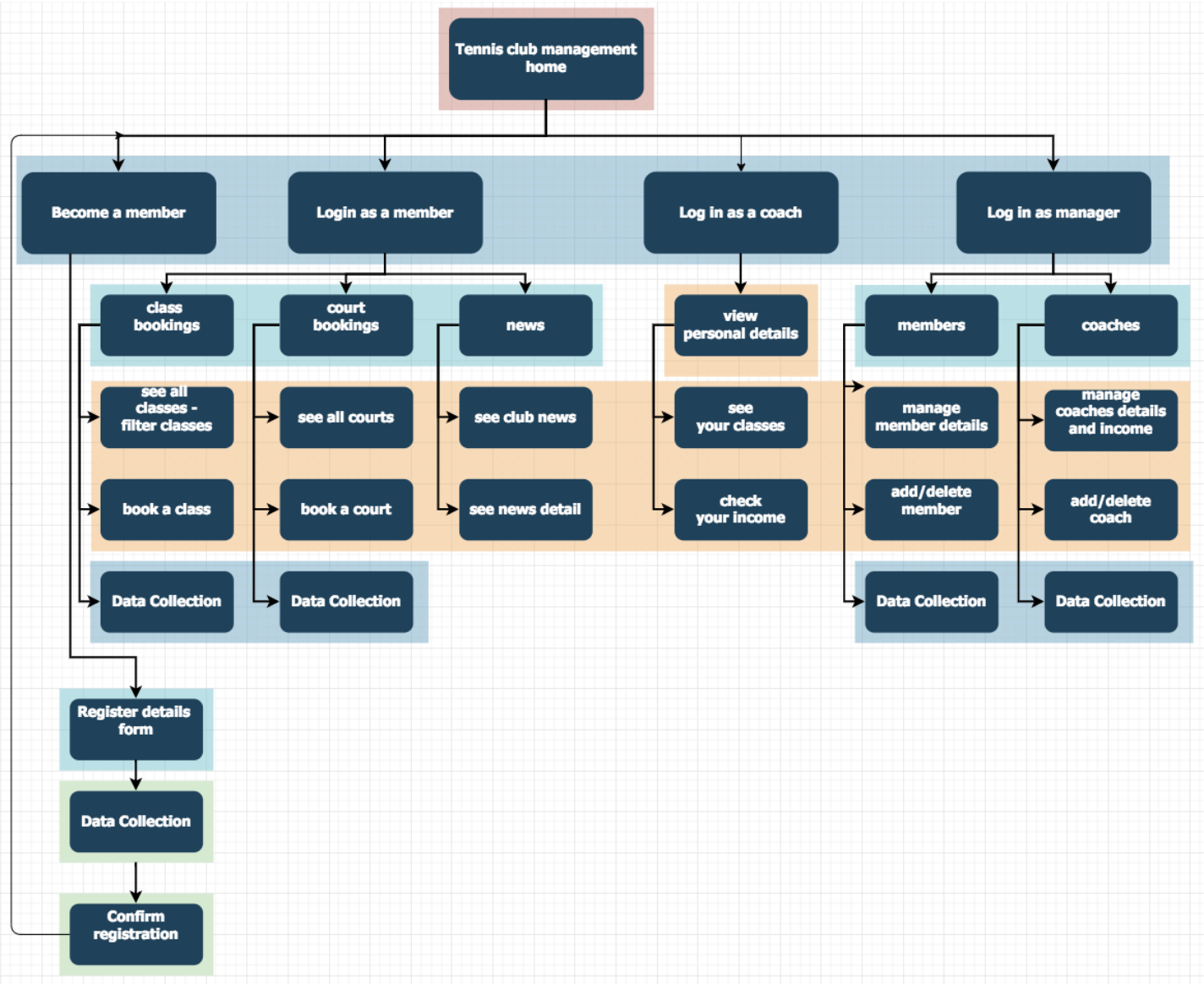
Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time

Screenshot:

	Possible effect of constraint	Plan
Hardware and software platforms	Should support both web/mobile	Test on both/Optimise with css
Performance requirements	Prevent double bookings and overbooking of a class	Insert checks at members and classes db levels and reject double bookings/ give feedback
Persistent Storage and Transactions	Data should be persistently saved in respective databases	Make sure all databases are updated following an action
Usability	There can be a lot of data on booking tables	Make each class details clickable for more info and larger font
Budgets	Low budget	Do best working solution remaining within resources
Time	Project deadline in 5 days	Test properly/ follow MVP plan first and grow app organically

Unit	Ref	Evidence
P	P.5	User Site Map

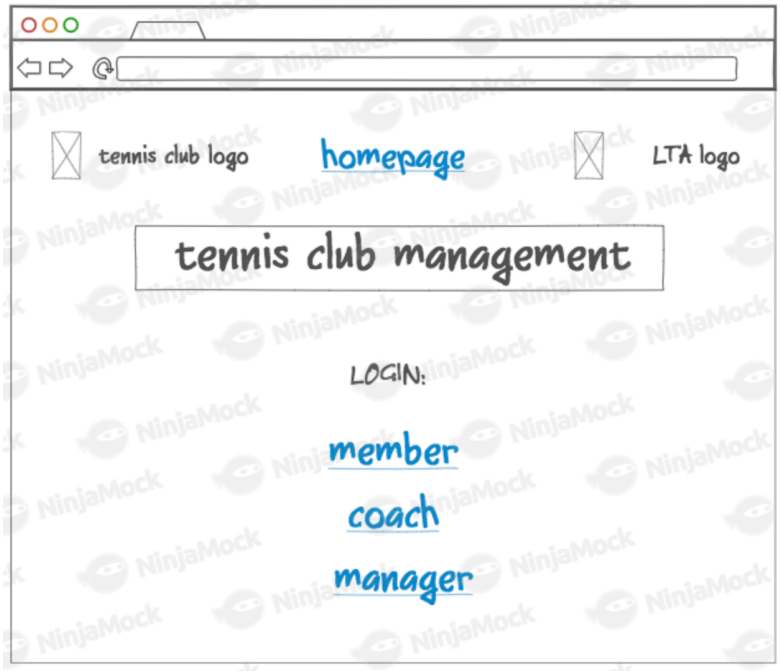
Screenshot:



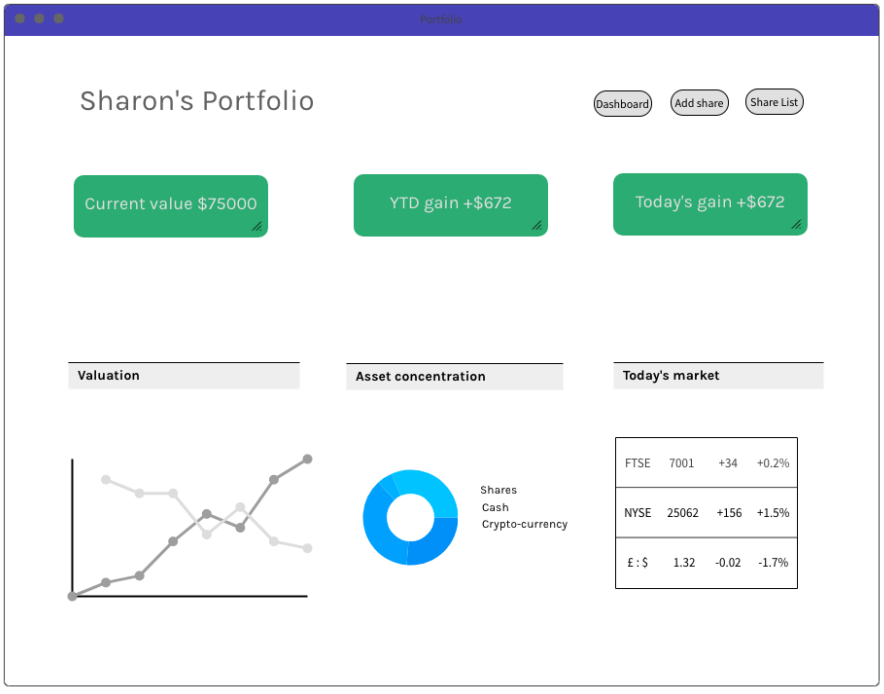
P.5 USER SITE MAP OF TENNIS CLUB MANAGEMENT APP

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

Screenshot:

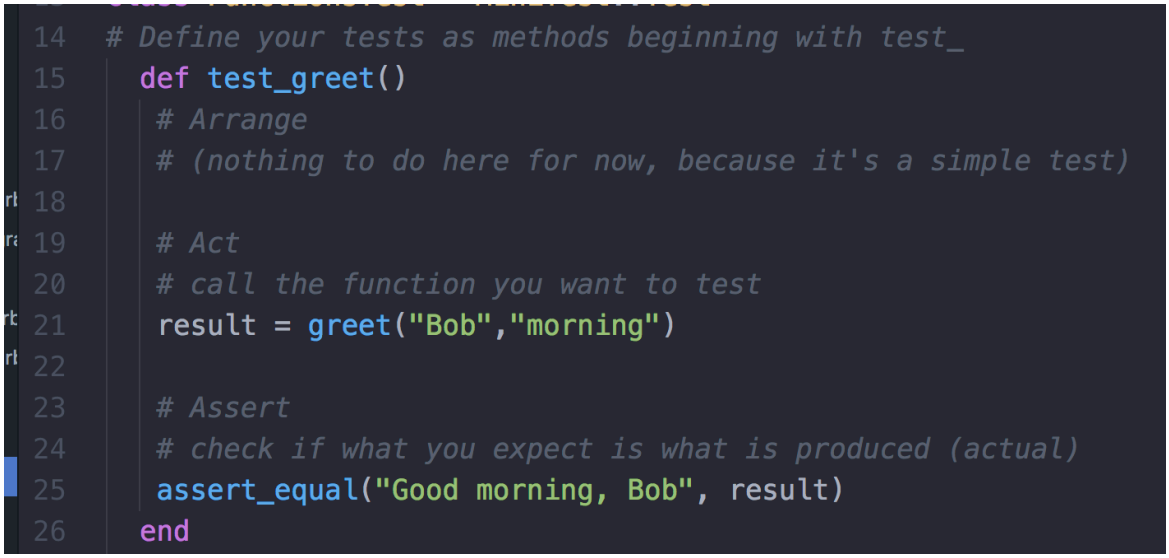


P.6 1/2 WEEK 5 PROJECT: TENNIS CLUB MANAGEMENT SYSTEM, HOMEPAGE WIREFRAME



P.6 2/2 PAIRED PROJECT: SHARES PORTFOLIO APP - HOMEPAGE WIREFRAME

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

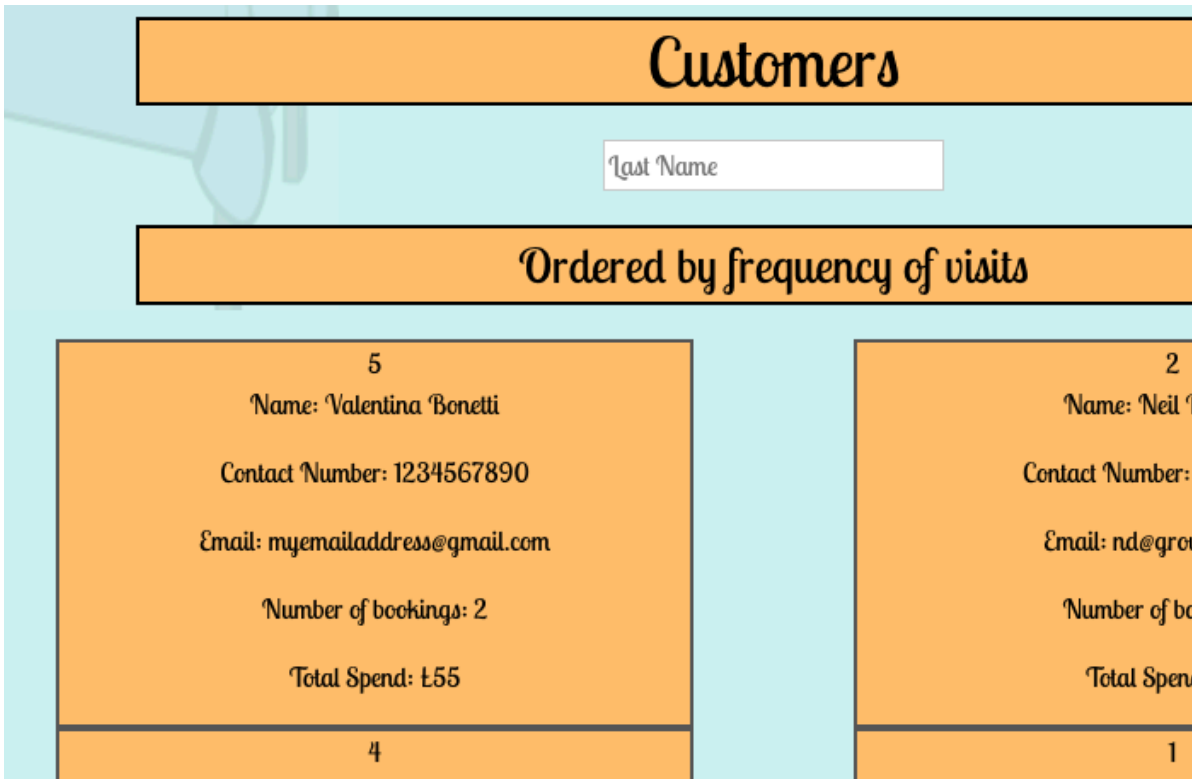


```
# Define your tests as methods beginning with test_
def test_greet()
 # Arrange
 # (nothing to do here for now, because it's a simple test)

 # Act
 # call the function you want to test
 result = greet("Bob","morning")

 # Assert
 # check if what you expect is what is produced (actual)
 assert_equal("Good morning, Bob", result)
end
```

PSEUDOCODE USED TO WRITE A TEST FUNCTION IN RUBY

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: see next page

P.13-1: USER INPUTTING CUSTOMER DETAILS INTO CUSTOMER FORM



P.13-2: RESULT OF USER INPUT - THE CUSTOMER IS SHOWN/SAVED INTO DATABASE

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

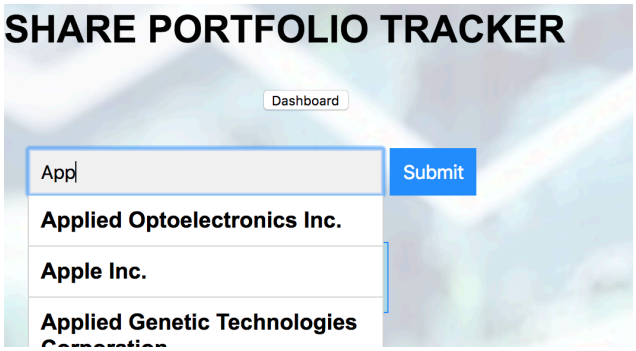
```
Customer customer1 = new Customer( firstName: "Jordan", lastName: "Davidson", email: "jd@growlmail.com", contactNumber: "749473829");
customerRepository.save(customer1);

Customer customer2 = new Customer( firstName: "Neil", lastName: "Davidson", email: "nd@growlmail.com", contactNumber: "749473003");
customerRepository.save(customer2);

Customer customer3 = new Customer( firstName: "Valentina", lastName: "Bonetti", email: "vb@growlmail.com", contactNumber: "749473034");
customerRepository.save(customer3);

Customer customer4 = new Customer( firstName: "Craig", lastName: "Dunlop", email: "cd@growlmail.com", contactNumber: "749475703");
customerRepository.save(customer4);

RestTable table1 = new RestTable( tableNumber: 1, maxCovers: 2);
restTableRepository.save(table1);

RestTable table2 = new RestTable( tableNumber: 2, maxCovers: 4);
restTableRepository.save(table2);

RestTable table3 = new RestTable( tableNumber: 3, maxCovers: 6);
restTableRepository.save(table3);

RestTable table4 = new RestTable( tableNumber: 4, maxCovers: 2);
restTableRepository.save(table4);
```

P.14-1: DATA BEING ENTERED AND SAVED IN THE PROGRAM (DATA LOADER FILE)

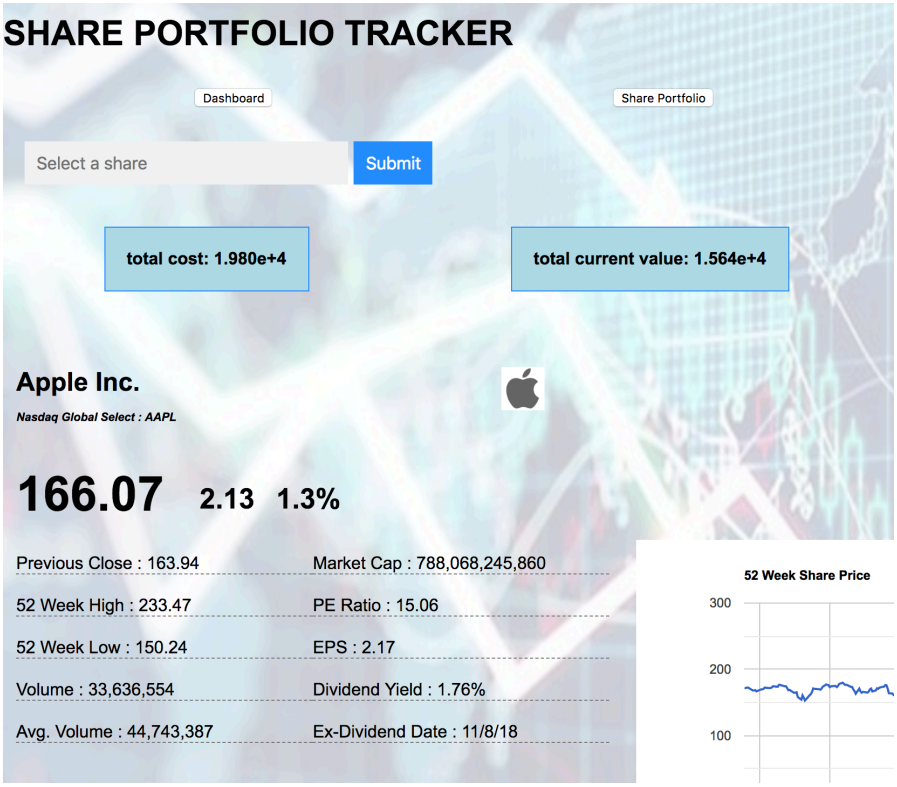
```
          List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | bookings  | table | postgres
 public | customers | table | postgres
 public | tables    | table | postgres
(3 rows)

restaurantService=# select * from customers;
 id | contact_number |      email       | first_name | last_name
----+----------------+------------------+------------+-----------
  1 | 749473829      | jd@growlmail.com | Jordan     | Davidson
  2 | 749473003      | nd@growlmail.com | Neil       | Davidson
  3 | 749473034      | vb@growlmail.com | Valentina  | Bonetti
  4 | 749475703      | cd@growlmail.com | Craig      | Dunlop
(4 rows)

restaurantService=#
```

P.14-2: PERSISTENT DATA APPEARING 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
		Description: Shares portfolio app. The user request info about a particular share select from the complete list of available shares, and the app retrieves the info from APIs and summarises it on a single page.

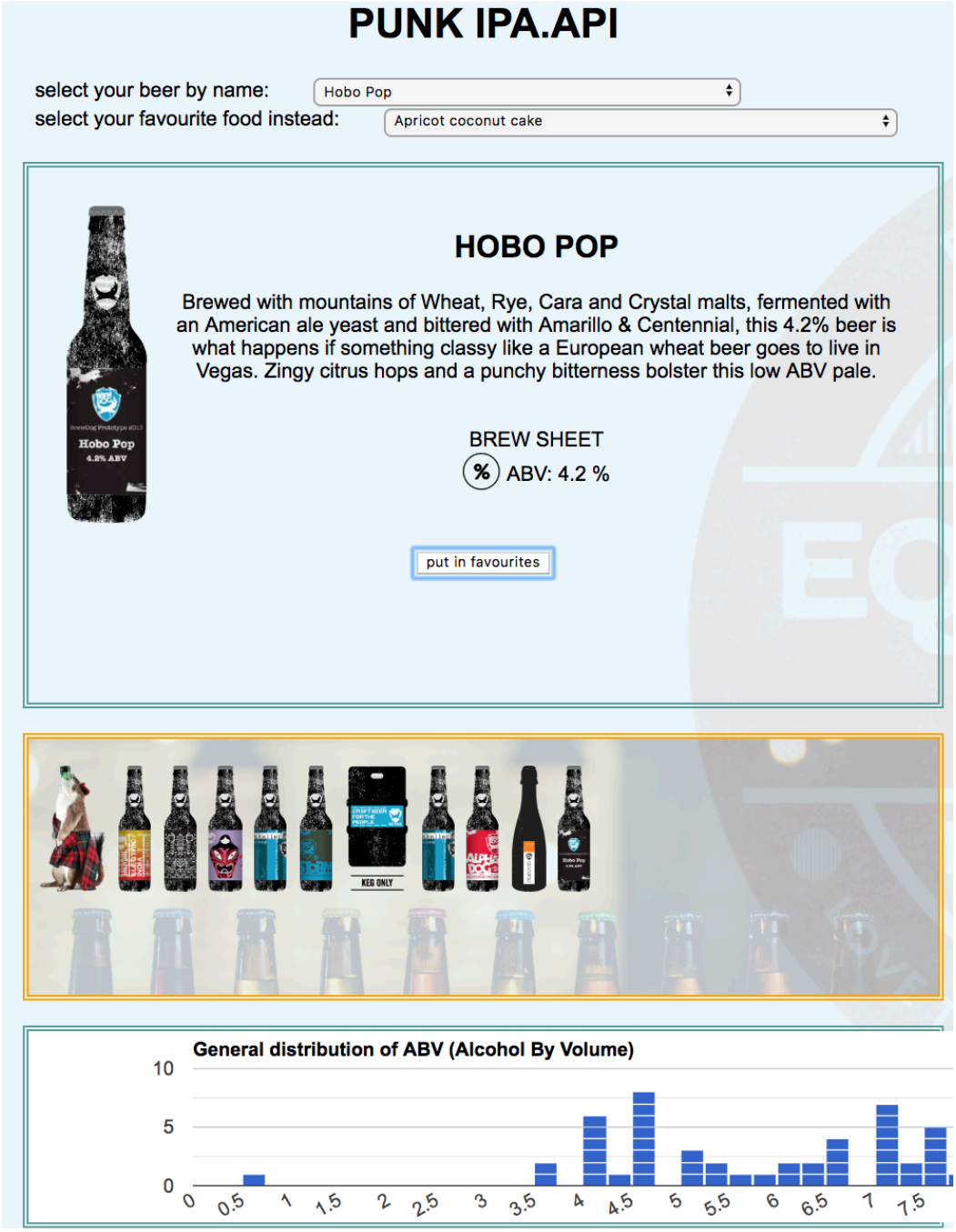


P.15-1: USER REQUESTING INFORMATION



P.15-2: ACTION BEING PERFORMED (SHARE INFO RETRIEVED FROM DIFFERENT APIS)

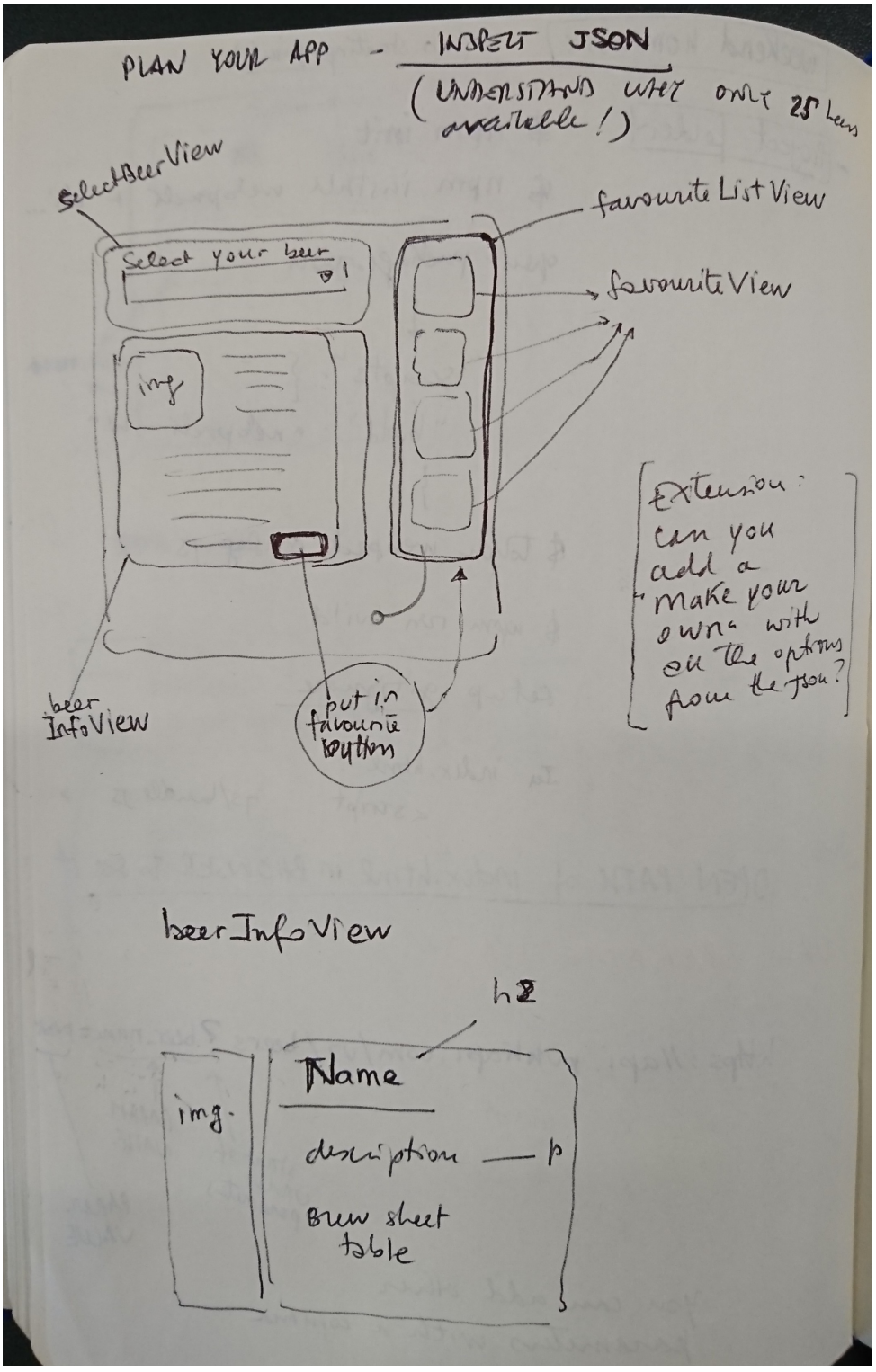
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.
		Description: Brewdog beers app: shows info, ABV and food combination for all the beers; allows creation of a favourite list with icons (all selectable/modifiable)



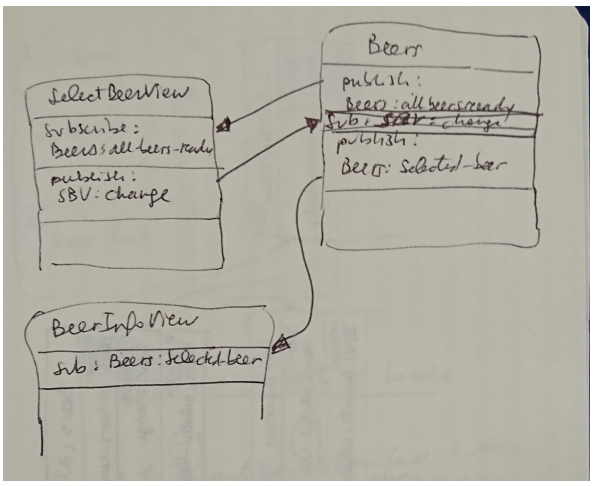
P.11 BEER API PROJECT (PUNK IPA API)

https://github.com/ValentinaBonetti/codeclan_weekend07_homework01

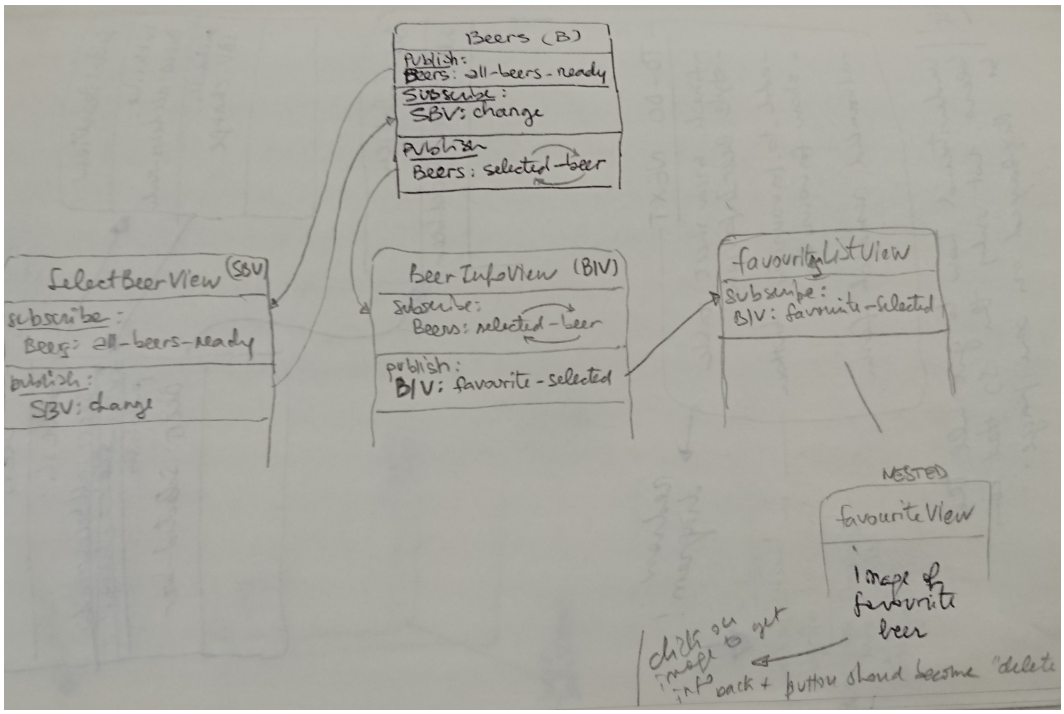
Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.
		Description: notebook screenshots



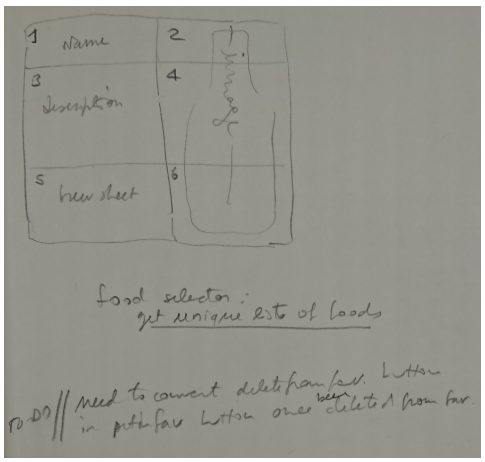
P.12-1/4: BEER PROJECT PLANNING



P.12-2/4: BEER PROJECT PUBS



P.12-3/4: BEER PROJECT PLANNING



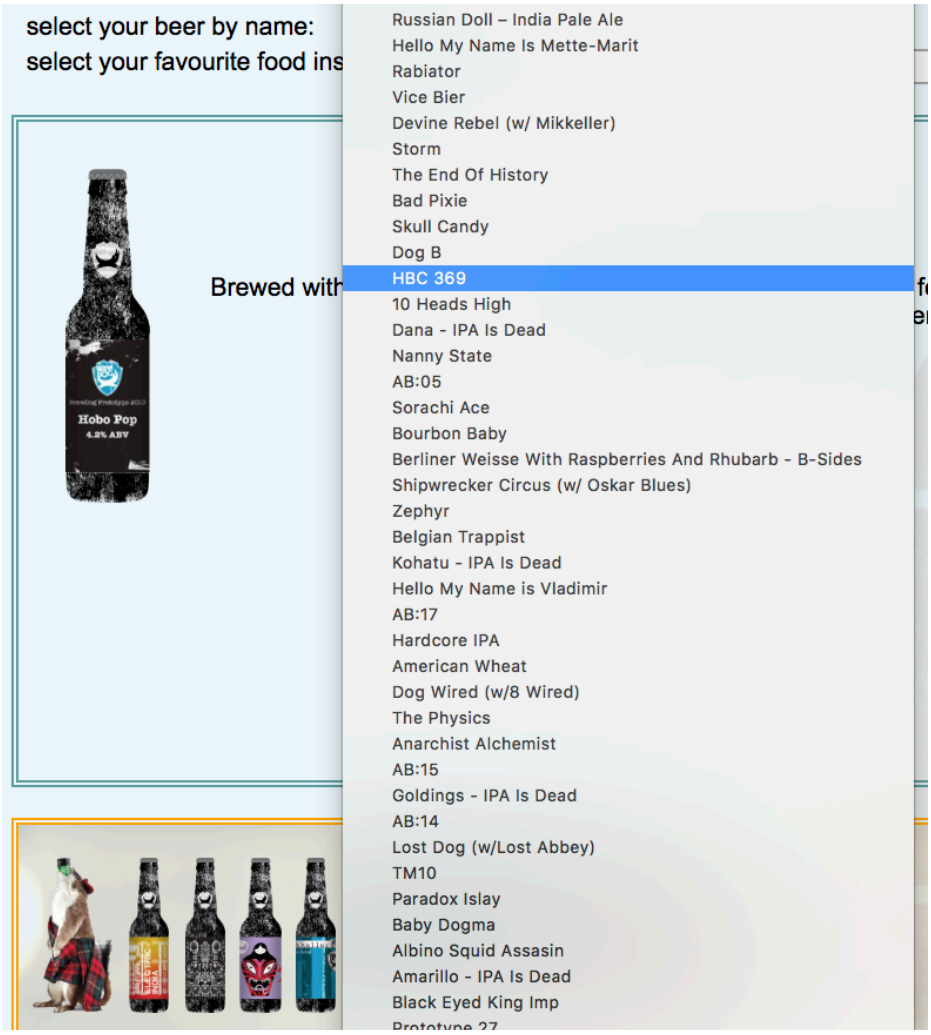
P.12-4/4: BEER PROJECT PLANNING

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: Brewdog beers app (description in P.11)

```
Beers.prototype.getData = function () {
  const request = new Request('https://api.punkapi.com/v2/beers?per_page=80');
  request.get().then(data => {
    this.beersData = data;
    PubSub.publish('Beers:all-beers-ready',this.beersData);

  })
};
```

P.16-1: API USED IN THE CODE



P.16-2: API RESPONSE SHOWN BY THE PROGRAM WHILE RUNNING

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

```
10
11  require_relative('card.rb')
12  class CardGame
13
14
15    def checkforAce(card)
16      if card.value = 1 # this needs to be == , otherwise always true
17        return true
18      else
19        return false
20      end
21    end
22
23    dif highest_card(card1 card2)
24    #"def" instead of "dif", comma missing between card1 and card2
25    if card1.value > card2.value
26      return card.name
27      # card object not defined
28      # card class method name not defined
29    else
30      # what if they are equal? equal condition missing
31      card2
32      # return keyword missing. It would work anyway, but better with
        return
33    end
34    # bad indentation of if statement
35  end
36  # bad indentation of method end
37  end
38  # this last end closes the class... it should be deleted
39
40  def self.cards_total(cards)
41    total
42    # total should be set to an initial value (0)
43    for card in cards
44      total += card.value
45      return "You have a total of" + total
46      # this statement returns the value of the first card and
47      # ends the for loop after the first iteration.
48      # It needs to be placed outside the for loop, after the following
        end.
49    end
50  end
51
```

P18.A 1/2 EXAMPLE OF CODE BEING TESTED

```ruby
  # tests for CardGame

  def test_checkforAce_true()
    expected = true
    actual = @cardgame.checkforAce(@card3)
    assert_equal(expected,actual)
  end

  def test_checkforAce_false()
    expected = false
    actual = @cardgame.checkforAce(@card1)
    assert_equal(expected,actual)
  end

  def test_highest_card()
    expected = @card1
    actual = @cardgame.highest_card(@card1,@card3)
    assert_equal(expected,actual)
  end

  # This ranking is used in the game of bridge:
  # spades (highest), hearts, diamonds, clubs (lowest)
  def test_highest_card_samevalue()
    expected = @card1
    actual = @cardgame.highest_card(@card1,@card2)
    assert_equal(expected,actual)
  end

  def test_highest_card_valueMoreImportantThanSuit()
    expected = @card2
    actual = @cardgame.highest_card(@card2,@card3)
    assert_equal(expected,actual)
  end

  def test_cards_total()
    expected = "You have a total of 13"
    actual = CardGame.cards_total(@cards)
    assert_equal(expected,actual)
  end
```

P18.A 2/2 - EXAMPLE OF TEST CODE

```
# Running:

...F....

Finished in 0.001297s, 6168.0797 runs/s, 6168.0797 assertions/s.

  1) Failure:
TestCardGame#test_cards_total [specs/testing_task_2_spec.rb:68]:
Expected: "You have a total of 13"
  Actual: "You have a total of 2"

8 runs, 8 assertions, 1 failures, 0 errors, 0 skips
```

P18.B - EXAMPLE OF A TEST FAILING

```ruby
require_relative('card.rb')
class CardGame


  def checkforAce(card)
    if card.value == 1
      return true
    else
      return false
    end
  end

  def highest_card(card1,card2)
    # bridge suit ranking low to high is alphabetical order:
    # low_to_high_suit_rank = ["clubs","diamonds","hearts","spades"]
    cards = [card1,card2]
    cards.sort_by! {|card| [card.value, card.suit]}
    cards.reverse!
    return cards[0]
  end


  def self.cards_total(cards)
    total = 0
    for card in cards
      total += card.value
    end
    return "You have a total of " + total.to_s
  end

end

```

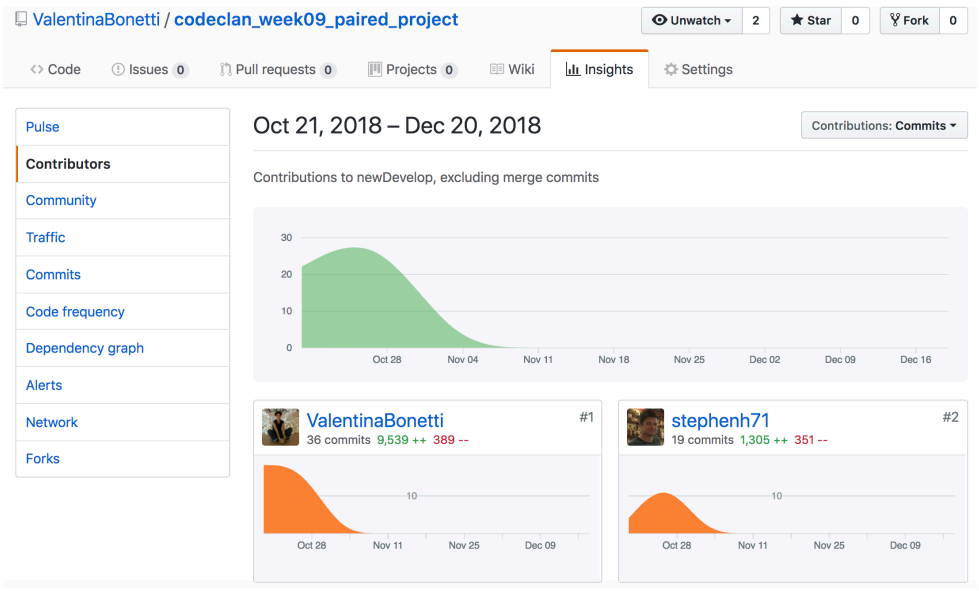
P18.C - EXAMPLE CODE FROM P.18 A CORRECTED TO MAKE TESTS PASS

```
ruby specs/testing_task_2_spec.rb
Run options: --seed 53349

# Running:

........

Finished in 0.001078s, 7421.1507 runs/s, 7421.1507 assertions/s.

8 runs, 8 assertions, 0 failures, 0 errors, 0 skips
```

P18.D - ALL TEST PASSED

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.



P.1 GITHUB CONTRIBUTORS PAGE FOR A GROUP PROJECT (SHARES PORTFOLIO APP)

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

Shares Portfolio Application

A local trader has come to you with a portfolio of shares. She wants to be able to analyse it more effectively. She has a small sample data set to give you and would like you to build a Minimum Viable Product that uses the data to display her portfolio so that she can make better decisions.

MVP

A user should be able to:

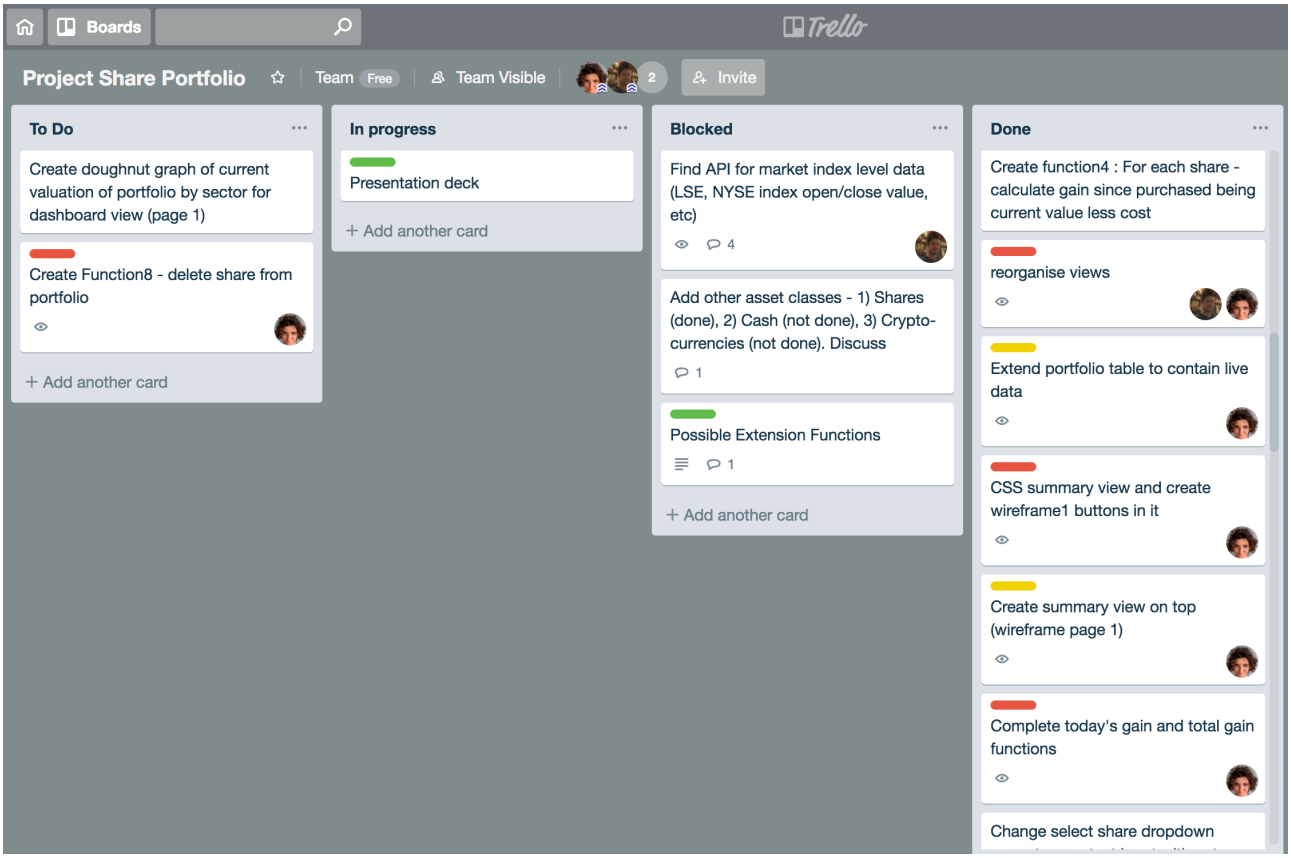
- view total current value.
- view individual and total performance trends.
- retrieve a list of share prices from an external API and allow the user to add shares to her portfolio.
- View a chart of the current values in her portfolio.

Example Extensions

- Speculation based on trends and further financial modelling using projections.

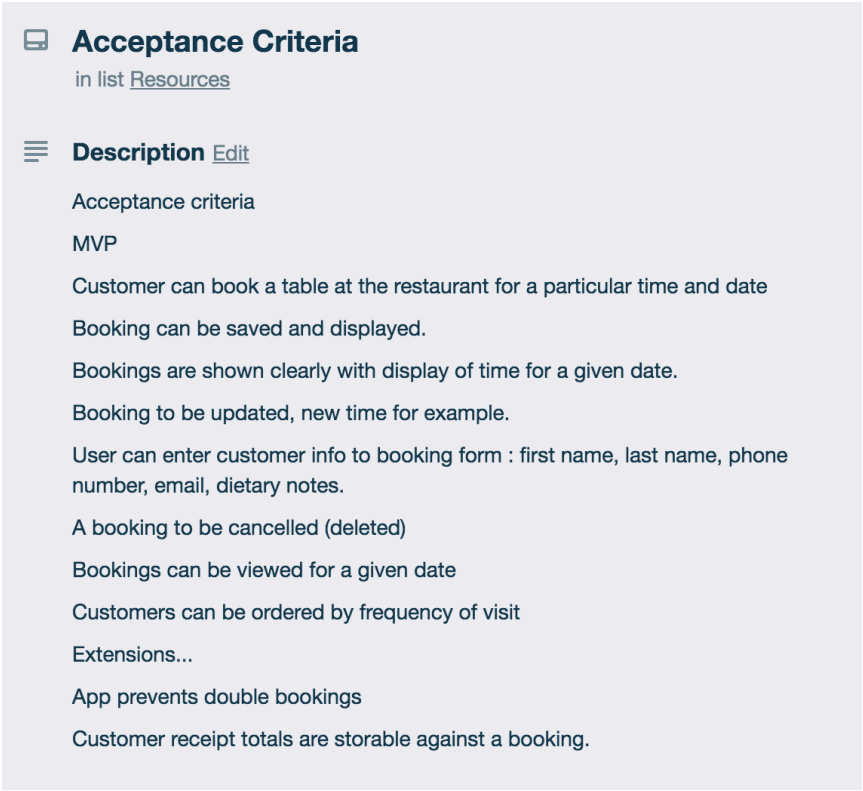
P.2 SCREENSHOT OF GROUP PROJECT BRIEF (SHARES PORTFOLIO APP)

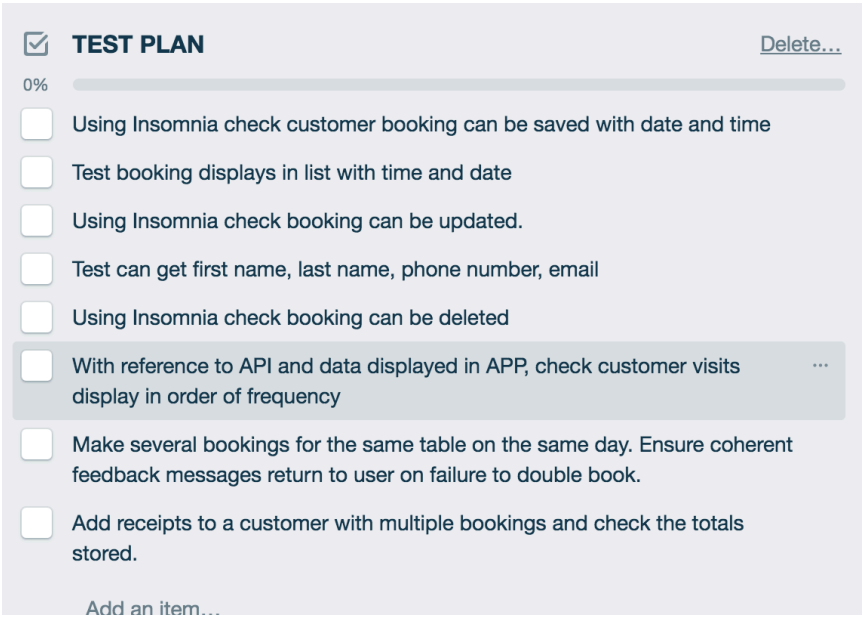
Unit	Ref	Evidence
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.
		Description: partial view of Trello board during final stages of the project



P.3 SCREENSHOT OF TRELLO BOARD USED FOR GROUP PROJECT (SHARES APP)

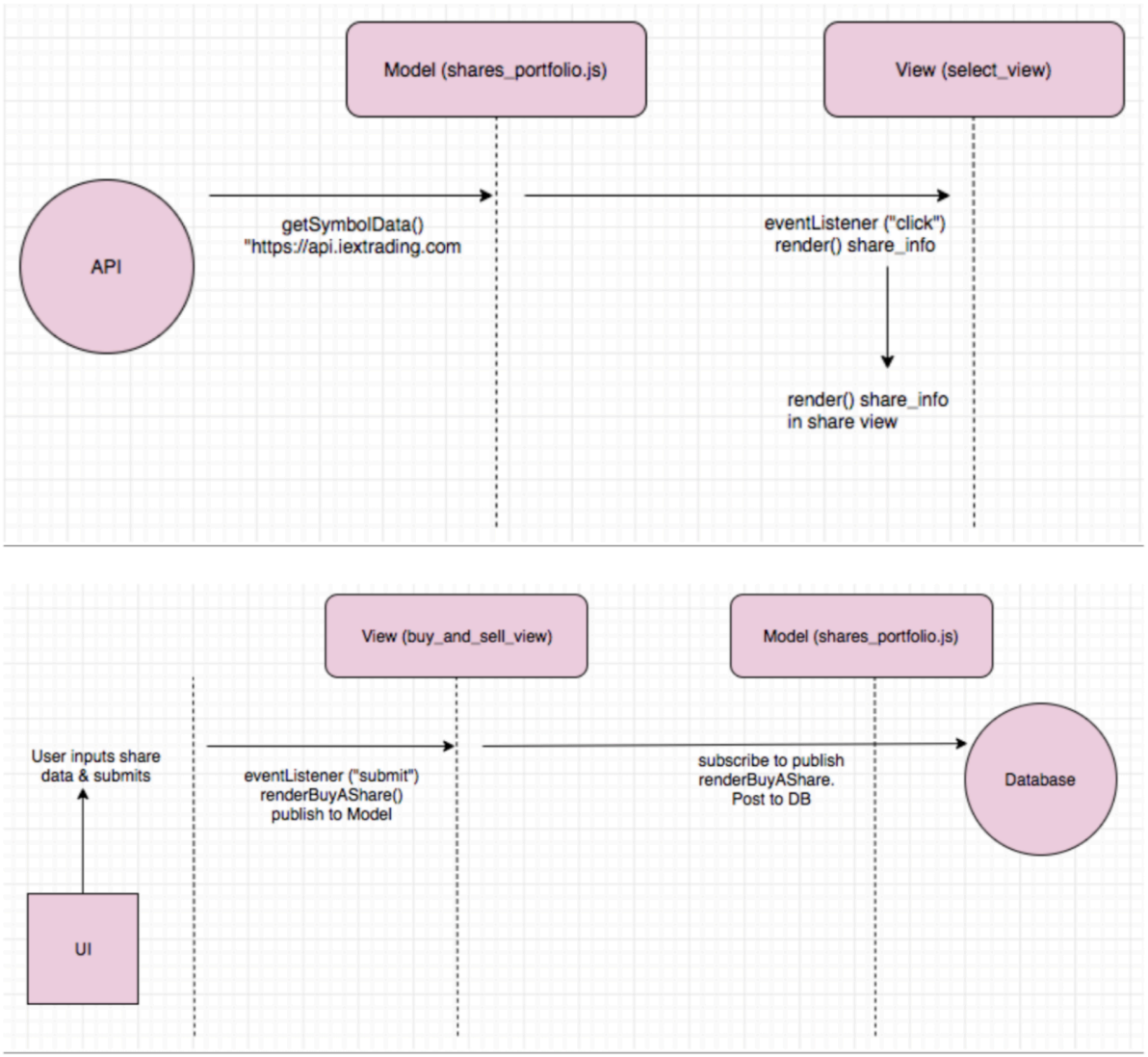
Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.





P.4 ACCEPTANCE CRITERIA AND TEST PLAN OF RESTAURANT BOOKING APP

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

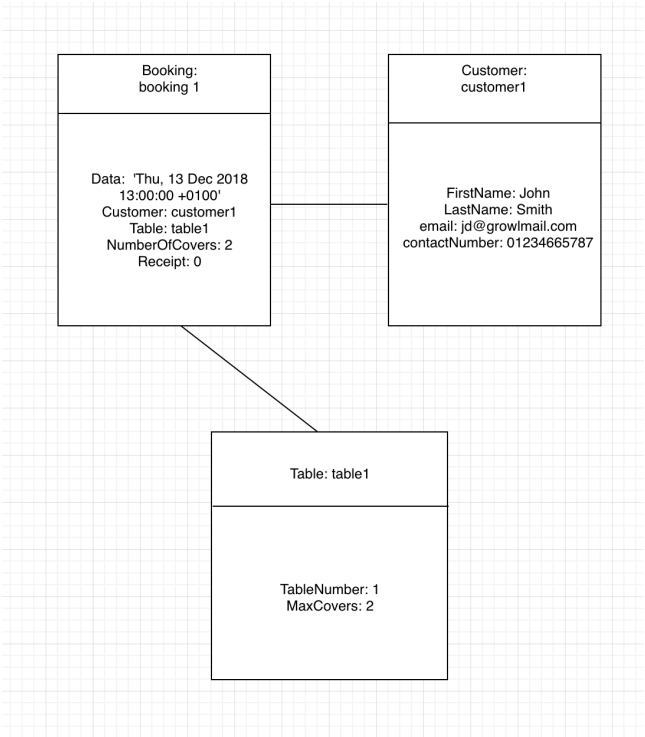


P.7 SYSTEM INTERACTION DIAGRAMS FOR SHARES PORTFOLIO APP.

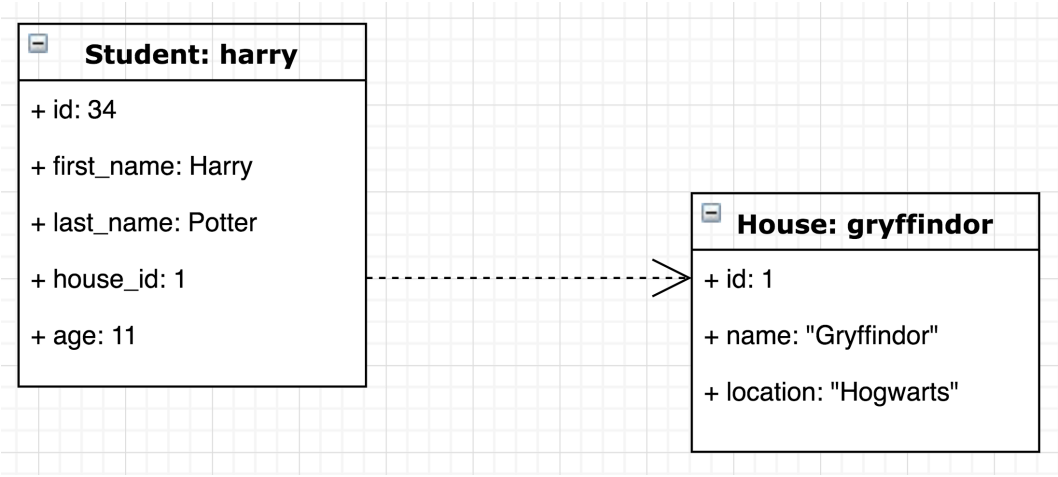
DIAGRAM 1: MODEL GETS DATA FROM MULTIPLE APIS AND SHOWS LIST OF SHARES, THEN USER SELECTS SHARE AND MODEL RETRIEVES INFO TO RENDER

DIAGRAM 2: USER INPUTS DATA IN BUY-SHARE FORM AND SHARE IS ADDED TO DB

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



P.8 1/2: OBJECT DIAGRAM FOR RESTAURANT BOOKING APP



P.8 2/2 OBJECT DIAGRAM OF HOGWARTS BOOKING PROGRAM

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

Bug / Error	Solution	Date
Form not showing	Changed event.booking.numberOfCovers to event.target.numberOfCovers etc	10/12/18
Table form: error relating to class.	Added export default TableForm	10/12/18
Booking Update form not populating date field	Revise date format to be in accordance with datetime-local formatting	10/12/18
Date showing incorrectly and difficult to read.	Import moment and change code to render date format intelligible	10/12/18
Date and time showing together, need them to show in different parts of form	Slice date data into two sections and display seperately	10/12/18

P.17 BUG TRACKING REPORT FOR RESTAURANT BOOKING APP

Week 12

Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.
		Description: A “Connectable” interface describes the methods that a class needs to implement to be considered of type “Connectable” (image 1/4). Two classes implementing Connectable are shown in images 2/4 and 3/4; a “Network” class (image 4/4) is thus able to include an ArrayList of Connectable devices that can contain any objects from different classes that implement “Connectable”.

Connectable.java

```
1  package behaviours;
2
3  public interface Connectable {
4
5      String connect(String data);
6
7  }
```

I.T.7 1/4: CONNECTABLE INTERFACE

InternetRadio.java

```java
import behaviours.Connectable;

public class InternetRadio implements Connectable {

    public String connect(String data) {
        return "Connected to radio: " + data;
    }

    public String tune(String channel) {
        return "You are listening to radio channel " + channel;
    }

}
```

I.T.7 2/4: “INTERNETRADIO” CLASS IMPLEMENTING CONNECTABLE INT.

```java

public class Printer implements Connectable {
    public String print(String string) {
        return "printing: " + string;
    }

    public String connect(String data) {
        return "connetting to network " + data;
    }
```

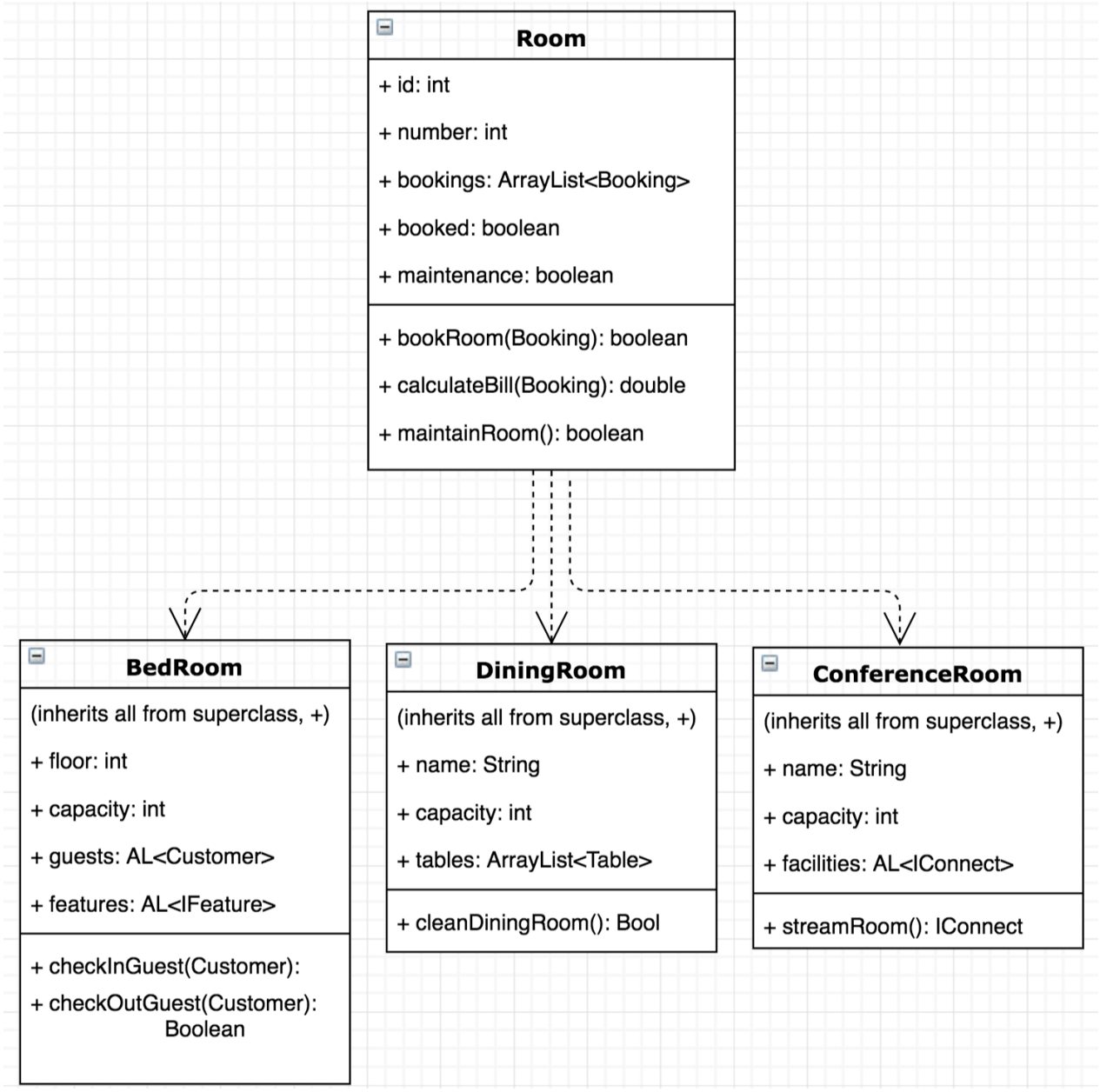
I.T.7 3/4: “PRINTER” CLASS IMPLEMENTING CONNECTABLE INTERFACE

Network.java

```java
import behaviours.Connectable;

import java.util.*;

public class Network {
    private String name;
    private ArrayList<Connectable> devices;
    private int max_connectable_devices;


    public Network(String name){
        this.devices = new ArrayList<Connectable>();
        this.name = name;
        this.max_connectable_devices = 10;
    }

    public String getName(){
        return name;
    }

    public int deviceCount(){
        return devices.size();
    }
```

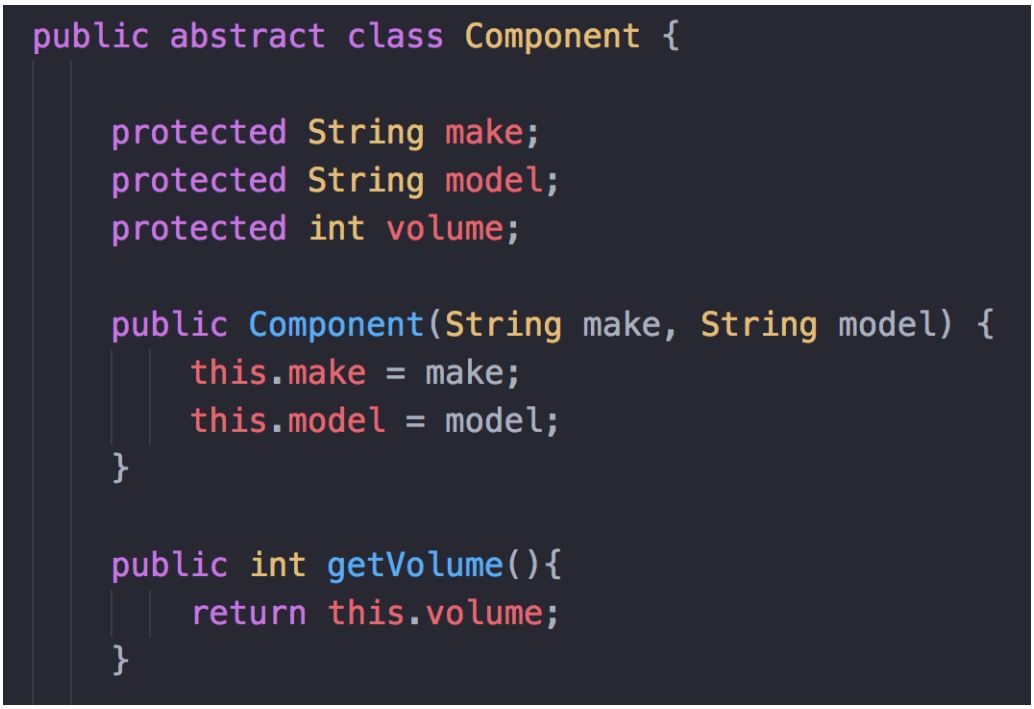
I.T.7 4/4: “NETWORK” CLASS CONTAINING AN ARRAYLIST OF CONNECTABLE

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



A.D.5 INHERITANCE DIAGRAM OF ROOM CLASSES FOR HOTEL MANAGEMENT PROGRAM

Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.
		Description: Example of encapsulation: the class contains the protected variable "volume", which is directly available only to subclasses. A getter method "getVolume" is needed to make the volume available to external components.



```
public abstract class Component {

    protected String make;
    protected String model;
    protected int volume;

    public Component(String make, String model) {
        this.make = make;
        this.model = model;
    }

    public int getVolume(){
        return this.volume;
    }
```

I.T.1 USE OF ENCAPSULATION IN A CLASS

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.

```java
package instruments;

import behaviours.IPlay;
import behaviours.ISell;

public abstract class Instrument implements IPlay, ISell {

    protected Integer idNumber;
    protected String make;
    protected String model;
    protected InstrumentType type;
    protected String material;
    protected Colour colour;
    protected Double buying_price, selling_price;

    public Instrument(Integer idNumber, String make, String model,
                      InstrumentType type, String material, Colour colour,
                      Double buying_price, Double selling_price)
                      {
        this.idNumber = idNumber;
        this.make = make;
        this.model = model;
        this.type = type;
        this.material = material;
        this.colour = colour;
        this.buying_price = buying_price;
        this.selling_price = selling_price;
    }

    public double calculateMarkup() {
        return this.selling_price - this.buying_price;
    }
```

I.T.2 A: INSTRUMENT CLASS

```java
package instruments;

public class Guitar extends Instrument {

    private Integer number_of_strings;

    public Guitar(Integer idNumber, String make, String model,
                  InstrumentType type, String material, Colour colour,
                  Double buying_price, Double selling_price, Integer number_of_strings) {
        super(idNumber, make, model, type, material, colour, buying_price, selling_price);
        this.number_of_strings = number_of_strings;
    }

    public String play() {
        return "Sdeooong sdeoooong";
    }
}
```

I.T.2 B: GUITAR CLASS INHERITS FROM INSTRUMENT CLASS

```java
    @Before
    public void setUp() throws Exception {
        guitar = new Guitar(2430752,"Yamaha","Classic 101",
                InstrumentType.STRING,"Wood", Colour.WHITE, 2340.0, 3456.60, 5);
    }
```

I.T.2 C: INSTANCE OF GUITAR CLASS (USED IN A TEST SETUP METHOD)

```java
    public double calculatePotentialProfit() {
        double total = 0.0;
        for(ISell saleable : stock){
            total += saleable.calculateMarkup();
        }
        return total;
    }
```

I.T.2 D: METHOD THAT USES “CALCULATEMARKUP” INHERITED FROM PARENT CLASS

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: The two algorithms I have chosen are a Ruby function that executes all the operations needed to sell a pet in a pet shop and a Java function that returns the total weight of the luggage that passengers have booked on a certain flight. I have chosen these particular algorithms because they are easy to follow even if the rest of the program is not shown, thanks to the selection of meaningful names for variables and functions.

```ruby
def sell_pet_to_customer(petshop,pet,customer)
  if pet != nil
    if find_pet_by_name(petshop,pet[:name]) != nil
      if customer_can_afford_pet(customer,pet)
        remove_customer_cash(customer,pet[:price])
        add_or_remove_cash(petshop,pet[:price])
        remove_pet_by_name(petshop,pet[:name])
        add_pet_to_customer(customer,pet)
        increase_pets_sold(petshop,1)
      end
    end
  end
end
```

P.9 1/2 RUBY FUNCTION USED IN A PET SHOP APP

```java
public double howMuchBaggageWeightIsBooked() {
    double total_booked_weight = 0;
    for(Passenger passenger: this.passengers) {
        total_booked_weight += passenger.number_of_bags()*this.planeType.getWeightPerBag();
    }
    return total_booked_weight;
}
```

P.9 2/2 JAVA FUNCTION USED TO CALCULATE THE TOTAL WEIGHT OF PASSENGERS' LUGGAGE ON A FLIGHT (FLIGHT MANAGEMENT APP)