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

Paste Screenshot here

Description here

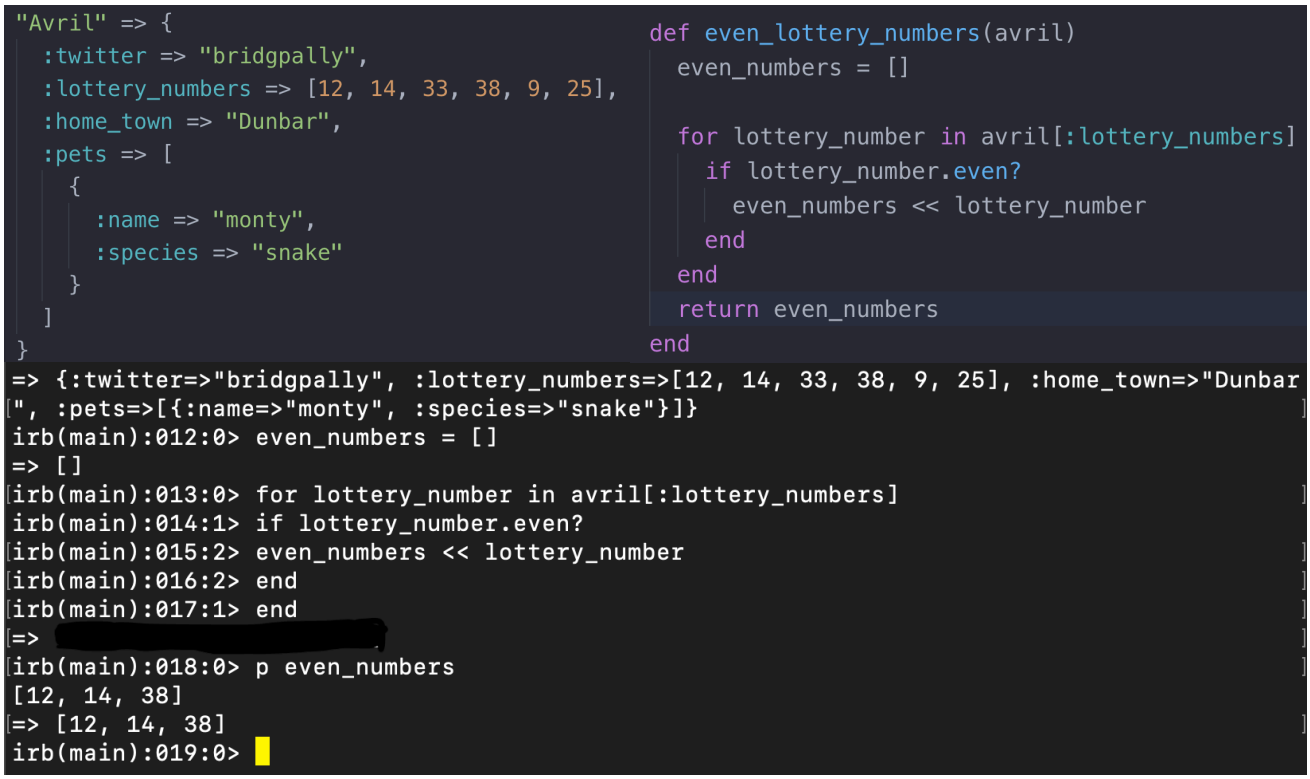
These screenshots are an example of Ruby hash and how they can be used (or can work) in a function. As it can be seen, chickens is an array of hashes. This array is a collection of key - value hashes (in this case keys are actually Ruby symbols). Each hash of chicken represent the following: name, age, eggs. It has been defined a function called count_eggs that takes the array of hashes chickens. This function has a variable (total_number_of_eggs) set to 0 that it is used as a counter and this will be incremented because of the for loop. The for loop is iterating each element in the array of hashes (that is, each chicken in chickens array) and it is selecting each

chicken's eggs quantity by accessing the key eggs (as in chicken[:eggs]) to end up adding those eggs quantity to the variable total_number_of_eggs which serves as a sort of counter storing the new quantity. Once it has been looped through each one and so it has finished, it has been return the variable total_number_of_eggs, resulting in a total of 4 eggs, as it can be seen in IRB.

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

Paste Screenshot here



Description here

This piece of code form part of collection of hashes assigned to the variable users, however I am only using it to show its functionality in the function even_lottery_numbers(…). Here, it can be seen an array of numbers (in this case of lottery numbers) for the key-symbol :lottery_numbers. Thus, the function even_lottery_numbers(avril) is looping through the array of numbers and accessing to them by using the key [:lottery_numbers]. Then, there is a condition (and if statement) that checks if that particular lottery_number is even?. If the condition is true (i.e. if the lottery_number that is being checked is even) then that lottery_number will be shovel into and empty array that has been declare as a set up a the beginning of the function. Finally, once it has been looped through all the number and thus, stored in the variable even_numbers, return the

new array containing only the even lottery numbers. As it can be seen lastly in the IRB, this function returns a new array [12, 14, 38] with only even numbers.

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

Paste Screenshot here

```ruby
category1 = Category.new({
  "type" => "Highlandwear"
  })
category1.save()

category2 = Category.new({
  "type" => "Confectionary"
  })
category2.save()

category3 = Category.new({
  "type" => "Winter Clothing"
  })
category3.save()

category4 = Category.new({
  "type" => "Summer Clothing"
  })
category4.save()

category5 = Category.new({
  "type" => "Lambswool, Cashmere &
  })
category5.save()

category6 = Category.new({
  "type" => "Jewellery"
  })
category6.save()
```

```ruby
def self.all()
  sql = "SELECT * FROM categories"
  categories = SqlRunner.run(sql)
  result = categories.map { |category| Category.new(category) }
  return result
end
```

```
From: /Users/matora/codeclan/codeclan_work/week_04/day_5/shop_invertory/db/seeds.rb @ line
 161 :

    156:
    157:
    158:
    159:
    160: binding.pry
 => 161: nil

[1] pry(main)> Category.all
=> [#<Category:0x007fcc3e8f3538 @id=17, @type="Highlandwear">,
 #<Category:0x007fcc3e8f3470 @id=18, @type="Confectionary">,
 #<Category:0x007fcc3e8f33a8 @id=19, @type="Winter Clothing">,
 #<Category:0x007fcc3e8f32b8 @id=20, @type="Summer Clothing">,
 #<Category:0x007fcc3e8f31f0 @id=21, @type="Lambswool, Cashmere & Wool">,
 #<Category:0x007fcc3e8f3100 @id=22, @type="Jewellery">,
 #<Category:0x007fcc3e8f3038 @id=23, @type="Sweaters, Jackets & Hoodied Tops">,
 #<Category:0x007fcc3e8f2f20 @id=24, @type="Small Gift & Soft Toys">]
[2] pry(main)>
```

Description here

The above screenshots show how to search data in a database. On the left, there have been created and saved instance objects of the class Category, they all have properties of unique id assigned by the database when they are saved and property type that takes a string and they are both collected in a hash. At the top, it can be seen a function that calls on the class itself (self.all() same as Category.all()). This function has declare a variable sql which is the one that contain the SQL query. This query is just saying to the database to SELECT * (all the columns) FROM the table categories. Then, this function is connecting to the database, performing the query and when finished it returns the data and it will be stored in the variable categories. SqlRunner.run(sql) is a Ruby helper file that acts as a set up to connect with the database and execute the specific query requested by each function. Once the query has been executed and the data has been collected in the variable, that data which it is returned in an array is then mapped as object of the class category and so creating a new array of Category objects. The result it is then displayed in the console.

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

Paste Screenshot here

```ruby
def manufacturers_products()
  sql = "SELECT manufacturers.*, products.* FROM manufacturers
         INNER JOIN products
         ON manufacturers.id = products.manufacturer_id
         WHERE manufacturers.id = $1
         ORDER BY manufacturers DESC"
  values = [@id]
  products_array = SqlRunner.run(sql, values)
  return products_array.map { |product| Product.new(product) }
end
```

```sql
SELECT manufacturers.*, products.* FROM  manufacturers
INNER JOIN products
ON manufacturers.id = products.manufacturer_id
ORDER BY manufacturers.phone DESC
```

Load Query... Save Query... Cancel Execute Statement

id	name	country	phone	email	id	name	description	stock_quantity	max_quantity	min_quantity	buying_cost	selling_price	manufacturer_id	category_id
24	Walkers LTD	UK	0131 871 555	info@walkersshortbread.com	34	Finger Shortbread	Traditional Butter Finger Shortbread	250	300	100	0.69	3.99	24	26
22	John Morrison Kiltmakers	UK	0131 556 473	info@johnmorrisonkilt.com	36	Sporran	Basic Leather Sporran	0	20	5	9.99	19.99	22	25
22	John Morrison Kiltmakers	UK	0131 556 473	info@johnmorrisonkilt.com	32	Kilt	8 Yards Polyester Kilt	20	25	5	24.9	49.99	22	25
23	Aran Tradition	Ireland	0121 006 885	info@arantradition.com	33	Woollen Hat	Winter Woollen Pom-Pom Hat	60	60	10	6.99	15.99	23	27
25	Safon	Spain	0034 972 26 83 04	info@safon.com	35	Nessie T-Shirt	Short Sleeves 100% Cotton T-Shirt	10	30	10	2.99	9.99	25	28

```ruby
books = ["The Silent Patient", "The Lost Girls of Paris", "Feeding you Lives", "Soft Skills", "Heart of Darkness", "Murderer in the Orient Express", "And Then There Were None"]

def sorting_library(books)
  books.sort{ |book1, book2| book1 <=> book2 }
end
puts sorting_library(books)
```

```
➜ learning-tests ruby sorting_data.rb
And Then There Were None
Feeding you Lives
Heart of Darkness
Murderer in the Orient Express
Soft Skills
The Lost Girls of Paris
The Silent Patient
➜ learning-tests
```

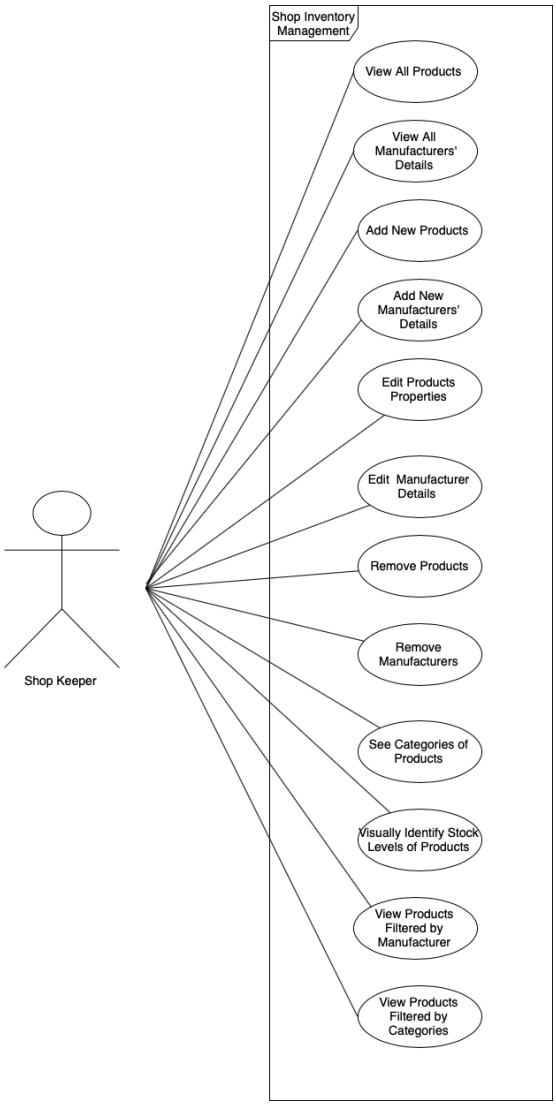
Description here

These screenshots are example of sorting datas. The picture at the top is a function that filters products by manufacturers and sorts the data by descending order of manufacturers' ids. As it can be seen in the picture below this function (which is a screenshot of Postico used to show how this query of sorting data works), the manufacturers have been sorted accordingly by ids and they also filters the products the share assigned manufacturer_id. The last image shows an array of books' title (as data type strings) and a function sorting_library (that will sort the books' title by alphabetical order). This function takes in the array of books as parameters; then, this array is passed the method sort and a block of code that gets the instructions of how to sort the array; and lastly, the function is printed. As it can be seen in the left, the terminal has printed the result of this sorting function when running the file.

Week 4

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

Paste Screenshot here

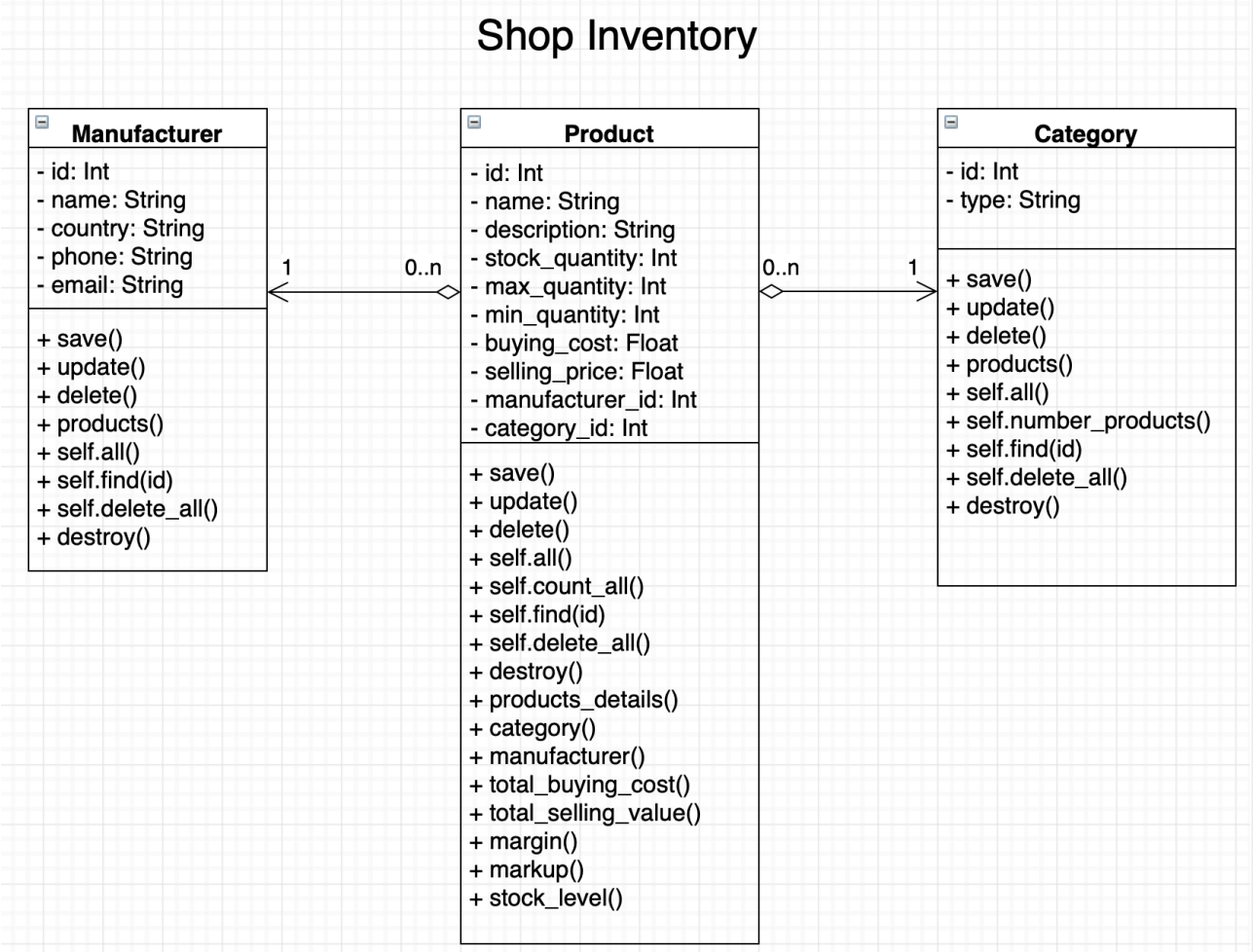


Description here

This is a case diagram that shows an overview of a shop inventory system. This system is only for stock control and admin/management control. So, here it can be seen the different cases a shop keeper can experience when using this stock control app. A shop keeper can view all products that are hold in the inventory at the moment. A shop keeper can view all manufacturers' contact details. A shop keeper can view products per category and all the categories. A shop keeper can add and edit the product list as well as add and edit manufacturers' contact details. A shop keeper filter the stock list by manufacturer or by categories and can visually identify levels of stock through colours when low quantity of stock.

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

Paste Screenshot here

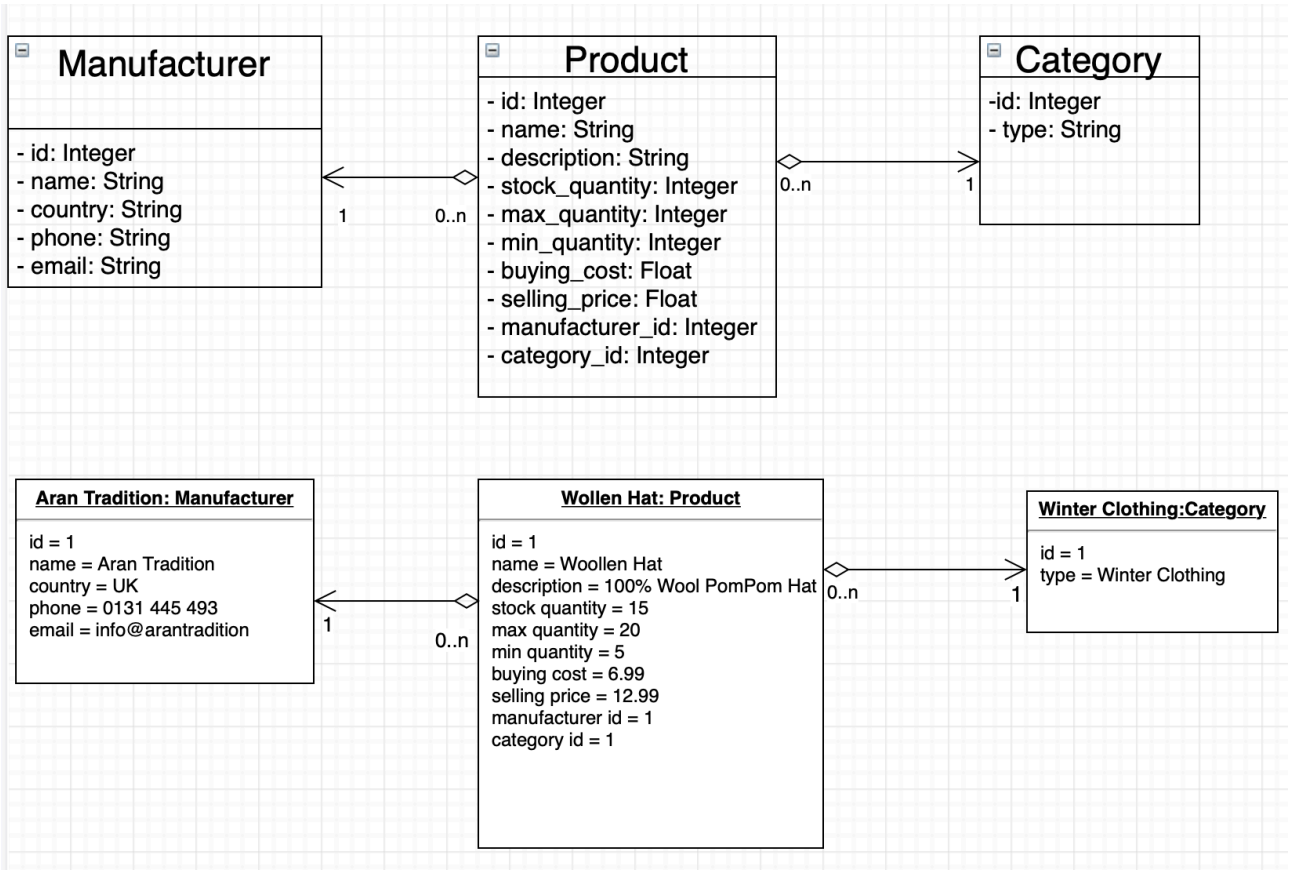


Description here

This is a class diagram model for a shop inventory app. So, here it can be seen a model for a stock control and stock management app of their products. There are three classes Product, Manufacturer and Category. Both Manufacturer and Category classes hold an association of one to many with Product class. This means that one manufacturer can have/ supplied one or more products but one product will only be supplied by a particular manufacturer. The same situation occurs to the class Category where one category can display more than one product but one product will be shown only in a particular category. From this diagram, it can also be seen that each class present different properties and methods. All three classes have been CRUD and specially products present some other methods like category( ) which filters the products by categories and manufacturers which filters the products by manufacturer, total cost and total price of products, margin and markup and the stock level of products depending on the quantity.

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

Paste Screenshot here

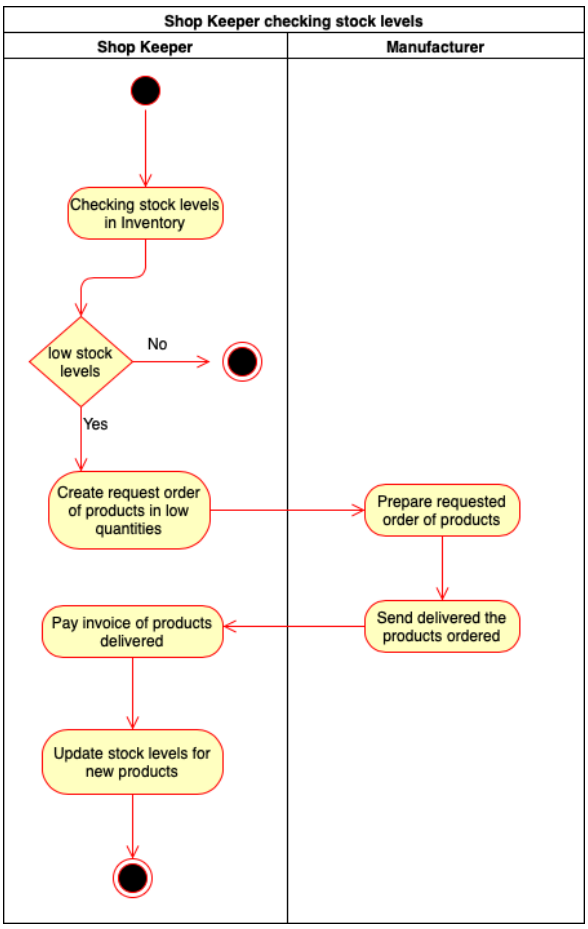


Description here

This is an object diagram where it can be seen a model of stock control and stock management. Three classes were model one for Manufacturer with the following properties: id (which is assigned when saved in the database), name of data type string, country of data type string, phone of data type string and email of data type string. The second class is Product which have the following properties: id, name (string), description (string), stock_quantity (integer), max_quantity (integer), min_quantity(integer), buying_cost (float), selling_price (float), manufacturer_id (Integer), category_id (Integer). And, lastly, class category which has an id (integer) and type (string). Just below each class diagram, there are their respective object diagram like Aran Tradition: Manufacturer, Woollen Hat: Product and Winter Clothing: Category.

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

Paste Screenshot here



Description here

Here, it can be seen an example of activity diagram prepared for the Ruby solo project. The project was to develop an app for stock control for shop keeper management only. In here, the shop keeper is checking the stock levels of his/her inventory, if the condition low stock level being true/yes the shop keeper will run an order to the manufacturer and the manufacturer will prepare and deliver the requested order. Thus, the shop keeper will finish this task by updating the inventory system. If the condition low stock would otherwise be false/no the shop keeper will finish the task there.

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

Paste Screenshot here

CONSTRAINT CATEGORY	IMPLEMENTATION CONSTRAINT	SOLUTION
Hardware and Software Platforms	The product is not supported across different platforms/devices as well as different OS. This issue can directly mean a constraint in usage of the software as well as preventing potential users of using it.	Plan, design and develop the software and hardware to be used across the main range of OS and devices to get to a much largest audience.
Performance Requirements	The software might require a fast speed range of Internet signal due to a very large data handling. This is a problem if the app is being used in areas where Internet signal is very poor and so it will not provide an effective and/or efficient experience of this software.	Well planned, designed and developed software to be compatible online and/or offline; otherwise, to be able to operate under a poor signal of Internet.
Persistent Storage and Transactions	A fully dependency on servers with extremely high volumes of traffic. This might be an issue because if the server fails, users of this product will not be able to access the data.	Solution for this constraint would be to implement a back up server for the software to ensure a good user experience.
Usability	The product lacks off a well structured HTML and a visualisation of data. This is a problem because people with disabilities will no longer be able to use this software.	Ensure the software is developed as accessible and inclusive as possible. This can be achieved by carrying out accessibility testing using assistive technology and usability testing.
Budgets	The product is provided with a budget that extends no more than a week. This is an issue because it might not cover extra features and functionalities as well as limiting the app to be upgraded.	A solution to this issue is to thoroughly plan and define achievable goals of the software.
Time Limitations	Developing time of the product covers just one week which constraints developers to add extra functionality to the software.	Ensure the product is developed to achieve basic goals agreed on MVP briefing.

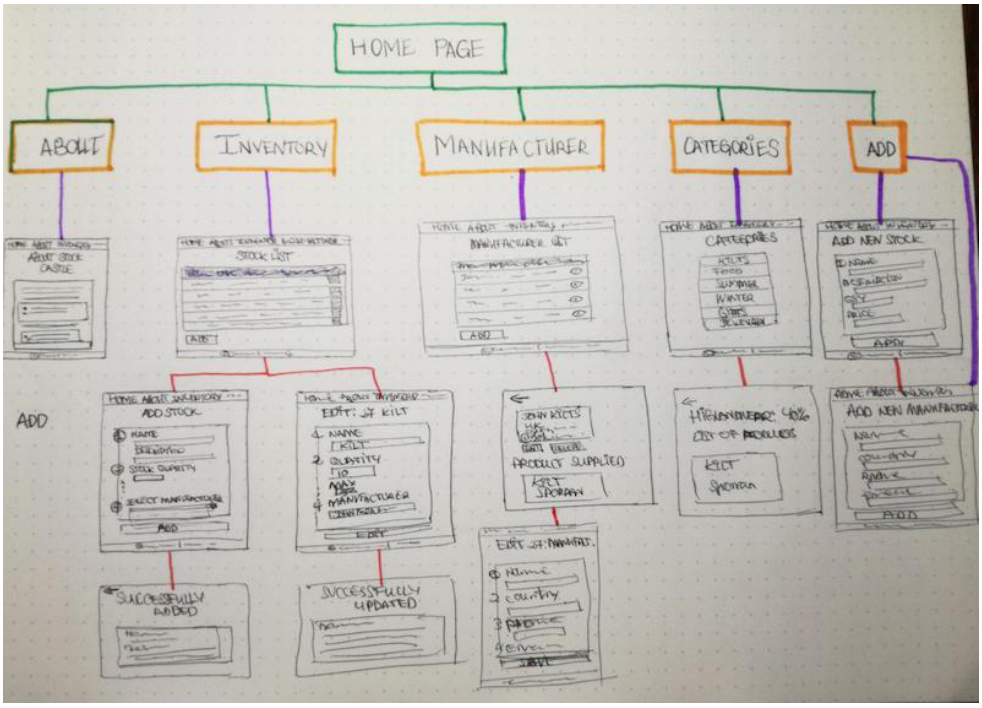
Description here

This is an Implementation Constraint plan with some of factors that could delay and/or even stop the software from developing. The following plan was defined for the Ruby solo project in which it has been developed an admin web application for management usage only that kept track of products by category and suppliers of those products. One of the major constraints when developing this app was the limitation of time which only allows the software to achieve basic functionality including CRUD (create, read, update and delete) data. Another major constraints to consider is usability; the current software has not being fully developed to be accessible and/or inclusive, partly due to the lack of planning as well as lacking tests to make the software

compatible with assistive technology, like: screen reader software, screen magnification software, screen recognition software, etc.

Unit	Ref	Evidence
P	P.5	User Site Map

Paste Screenshot here

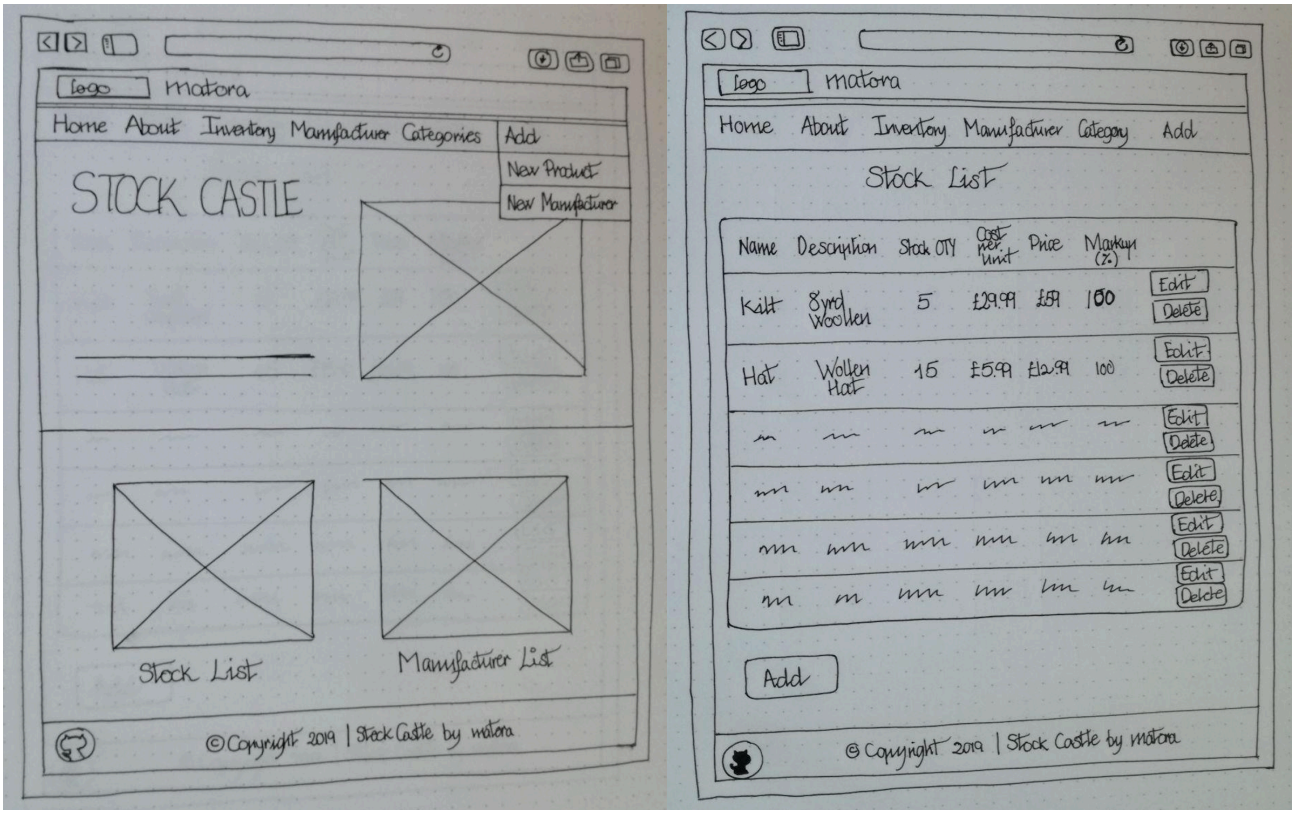


Description here

This is a user site map for an app to carry out stock control for a shop keeper usage. As it can be seen, this app will consist of a main Home Page from which any user/shop keeper can navigate to different pages of the app, like: About page (which shows some information of the app and how to use it), Inventory page (which shows the main functionality of the app: view of all products and options of edit, delete and add new products), Manufacturer page (which displays a list of the different manufacturers that supply products to the shop with address and contact details, and also options for edit, delete and add new manufacturers to the list), Categories page (which displays a list of categories for the products that exist at the moment in the stock of the shop) and finally a quick and fast path to access the form to add new products or new manufacturers.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams

Paste Screenshot here



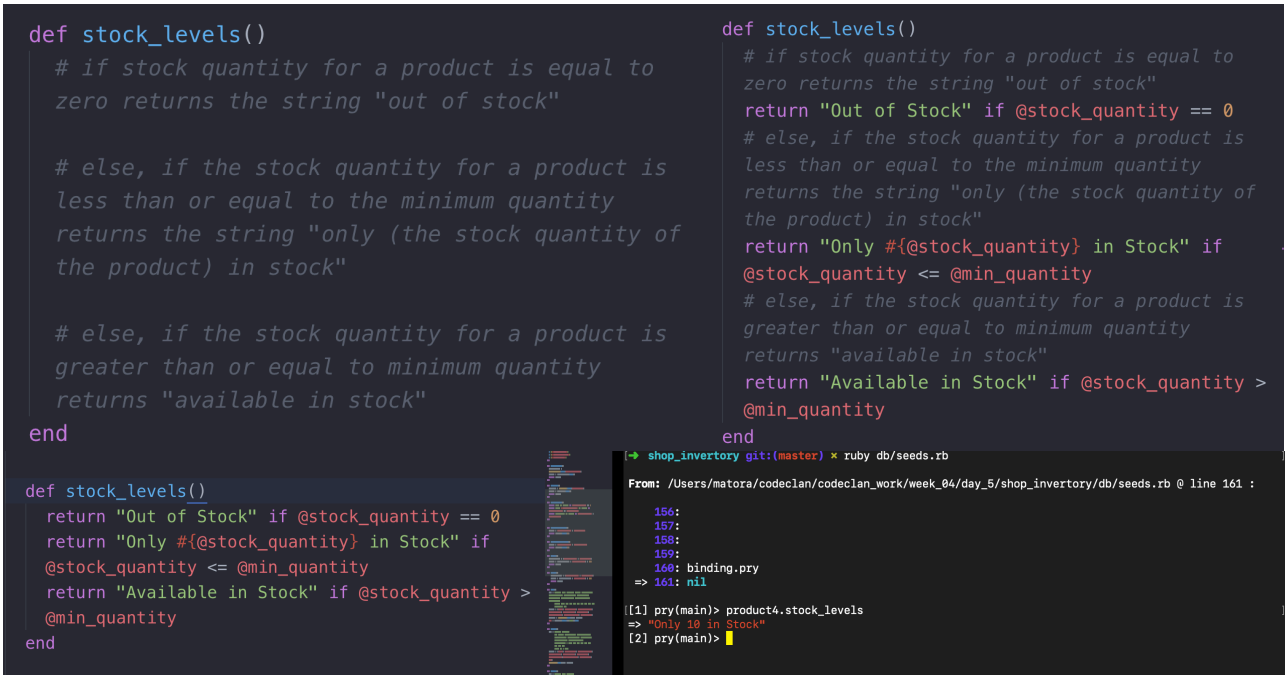
Description here

These two wireframe diagrams were designed as part of the planning for the Ruby solo project. The diagram on the left represents the main Home page for an app that tracks and manages products (much like the inventory of a shop). Thus, it can be seen a navigation bar a the top of the page with the different pages to which any user can navigate. In the main centre there are two main dashboard, the one above displays the name of the app with some description of the software and just beside main picture of the logo. Just below this dashboard, a section with two images: one that represents a check list of products which will link to the inventory page, and the second image represents a manufacturer building which will link to the manufacturers' list of contact details. Finally, at the bottom there is a footer with the copyright and name of the app.
The diagram on the right represents the Inventory page (stock list table). In this page, there is a main big table displaying the products that are hold in the shop to which you can access them, view individual products, edit each products of the list as well as deleting them. Just below the table the user of this app can also add new products to this table list by clicking in "Add" button which will take them to a very easy and simple form to cover the details of the new product added. Finally, the footer can be seen at the bottom.

Week 5

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

Paste Screenshot here

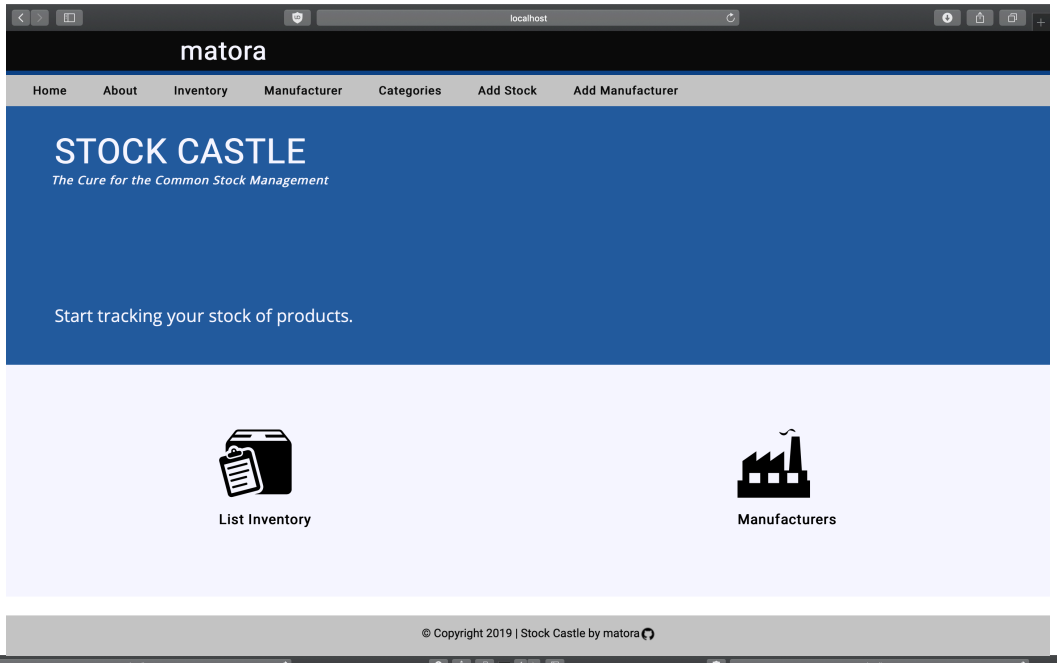


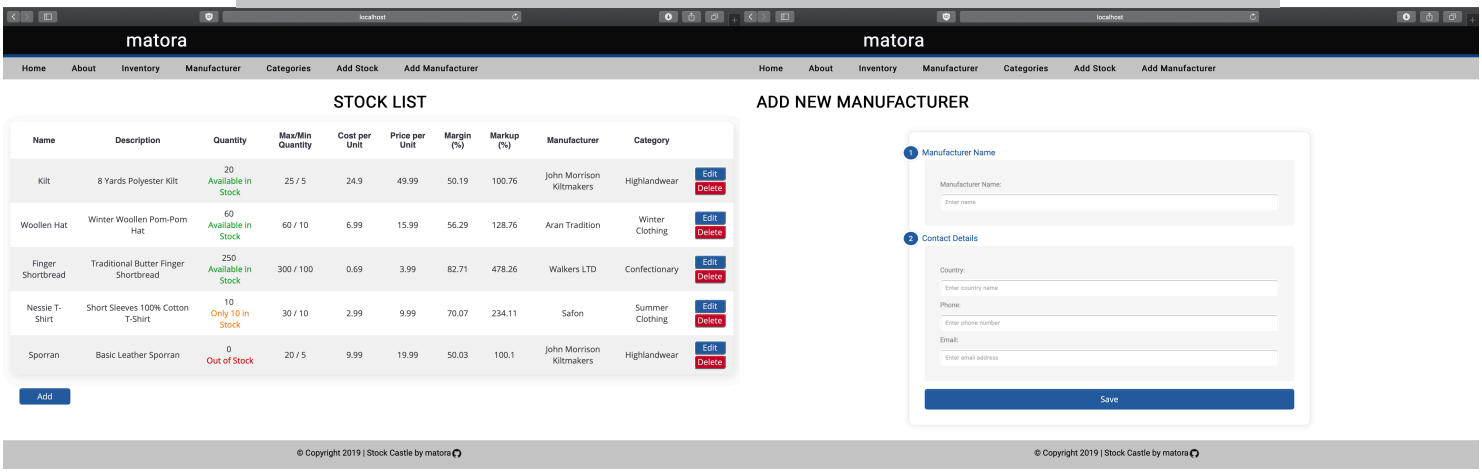
Description here

The screenshots shows a function defined to check the different stock levels for product' objects depending on the stock quantity. Here, it can be seen a piece of pseudocode to create this function which uses control flow statements (if…else) to check the different conditions that can occur and depending on the condition met will return one thing or the others. At the bottom there can be seen how this piece of code works in the console.

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

PasteScreenshot here





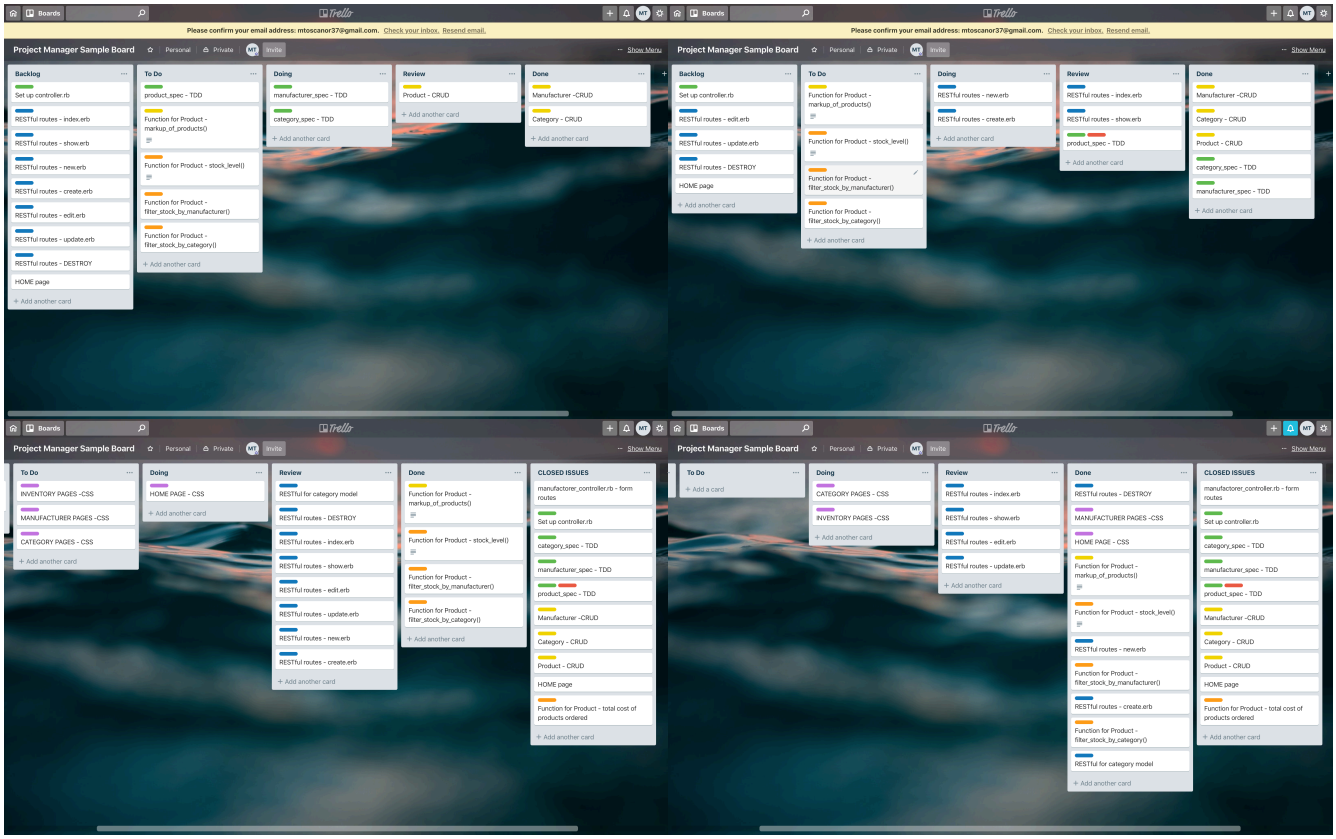
Description here

https://github.com/matosca/Ruby_Project_Shop_Inventory

As part of CodeClan's 16 weeks programming course, I was asked to create a web app using Sinatra, PostGRES SQL, HTML and CSS, and Ruby. Stock Castle is designed for stock management. This allows a shop keeper to track products easy and fast. It also keeps track of contact details of different manufacturers. The project started with the following MVP: the inventory should track individual products as well as individual manufacturers, the inventory should be able to edit/update, delete and create new products and manufacturers separately and finally, the inventory should display visually stock levels of products. The extensions for this project where: the inventory should display total cost, total price of products, the inventory can calculate the markup and margin of products, products could be filter by manufacturers and the inventory should have categories and be able to get products filtered by categories.

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

Paste Screenshot here

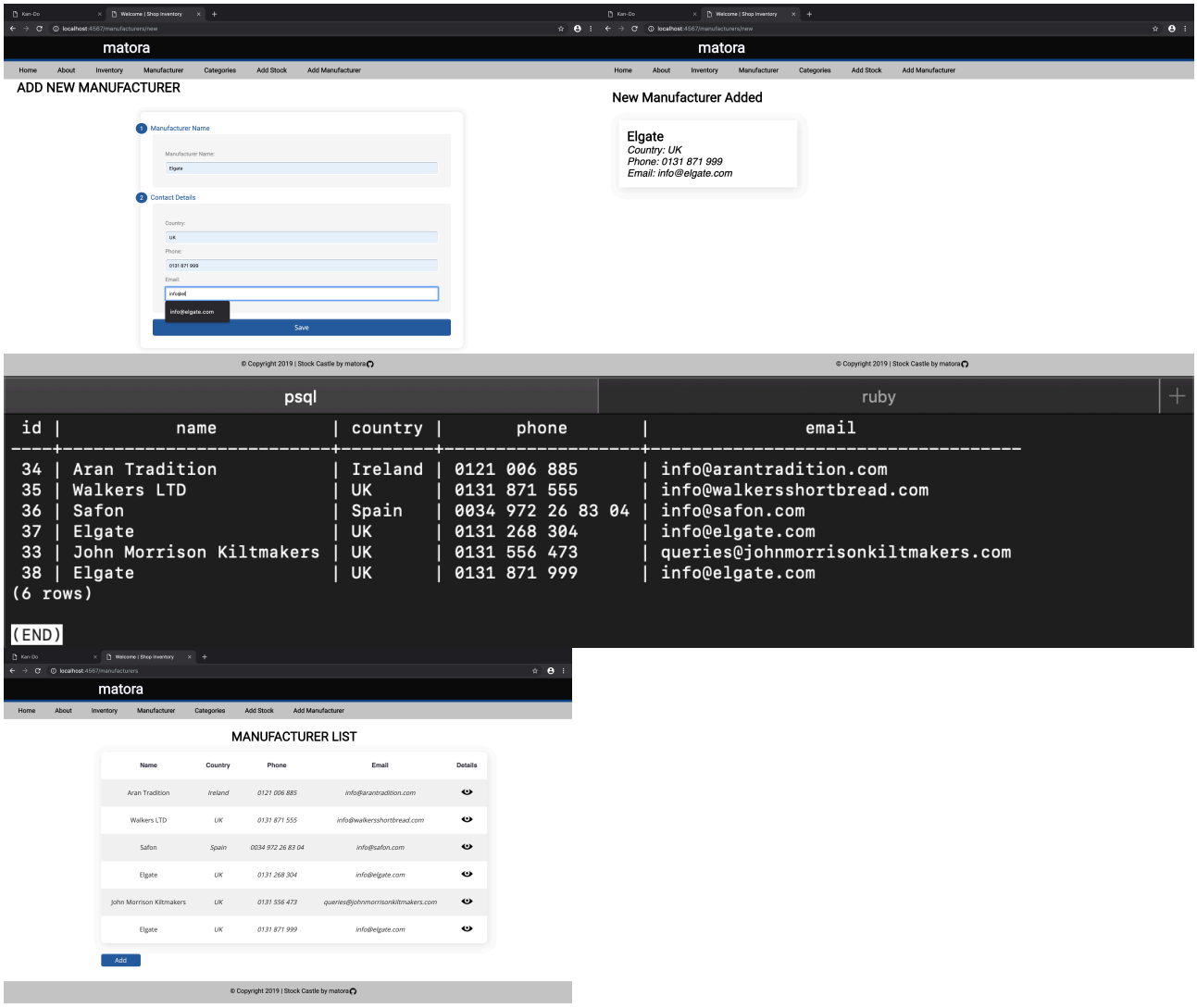


Description here

Here, it can be seen some screenshots from Trello. This system of Kanban board I implemented for managing the different stages of development of my solo Ruby project. I planned the week to do different small tasks and so I moved forward developing features of the software. As I was finishing the different tasks and achieving small goals that I set up to my plan I was able to progress fast and efficient along the whole process of development for this project.

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

Paste Screenshot here



Description here

These screenshots show user inputting data into the program. Here, the user has is required to fill a form to add new manufacturers to this web app which tracks products and manufacturers for a shop. Once, the user has inputted the required information and saved that information, a confirmation's view of the new manufacturer added is displayed with all details. Below, there is a screenshot of the terminal that shows our database table for manufactures including the recently added by the user through the form. And, as it can be seen in the last screenshot, the new manufacturer appears at the end of manufacturers' list table.

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

Paste Screenshot here

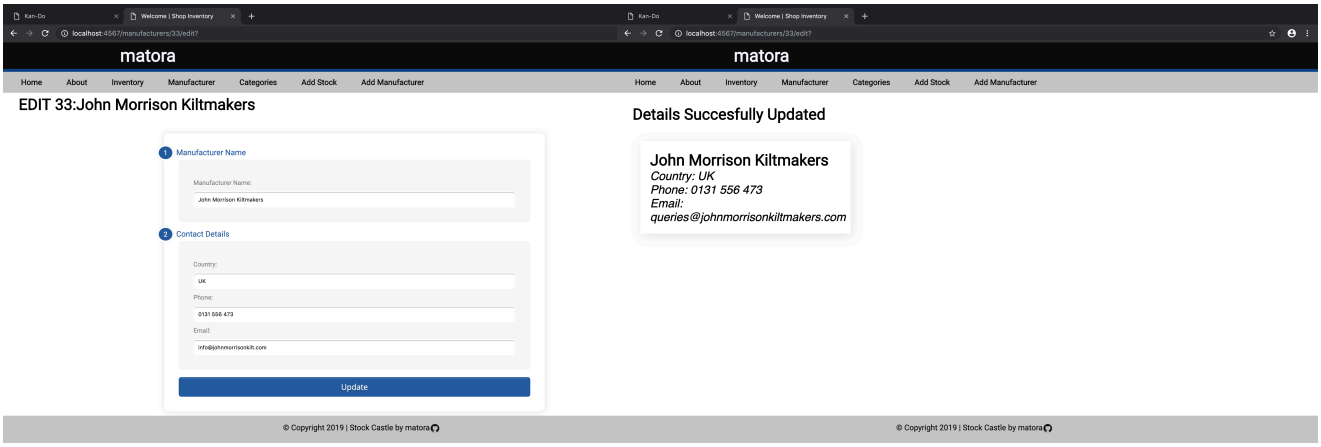
```
[➜  shop_invertory git:(master) ✗ psql -d shop_inventory
psql (11.1)
Type "help" for help.

[shop_inventory=# INSERT INTO manufacturers(name, country, phone, email) VALUES ('Toucan Jewellery', 'UK', '0127 342 8896',
  'info@toucan.com');
INSERT 0 1
[shop_inventory=# SELECT * FROM manufacturers;
 id |          name           | country |       phone       |              email
----+-------------------------+---------+-------------------+----------------------------------
 34 | Aran Tradition          | Ireland | 0121 006 885      | info@arantradition.com
 35 | Walkers LTD             | UK      | 0131 871 555      | info@walkersshortbread.com
 36 | Safon                   | Spain   | 0034 972 26 83 04 | info@safon.com
 37 | Elgate                  | UK      | 0131 268 304      | info@elgate.com
 33 | John Morrison Kiltmakers | UK      | 0131 556 473      | queries@johnmorrisonkiltmakers.com
 38 | Elgate                  | UK      | 0131 871 999      | info@elgate.com
 39 | Heather Gems            | UK      | 0131 871 7556     | info@elgate.com
 40 | Edinburgh Wool LTD      | UK      | 0241 998 7556     | queries@edinburghwool.com
 41 | Toucan Jewellery        | UK      | 0127 342 8896     | info@toucan.com
(9 rows)

shop_inventory=#
```

Description here

This is a screenshot of psql interpreter run in the terminal to demonstrate data persistence. Here, it can be seen how data has been inputed and saved into our shop_inventory database. Afterwards, a query to select all columns from the table manufacturers is shown to display the actual state and confirmation of the data saved in the table.

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

Paste Screenshot here



```
[➜  shop_invertory git:(master) ✗ psql -d shop_inventory
psql (11.1)
Type "help" for help.

[shop_inventory=# SELECT * FROM manufacturers;
 id |           name           | country |       phone       |               email
----+--------------------------+---------+-------------------+-----------------------------------
 34 | Aran Tradition           | Ireland | 0121 006 885      | info@arantradition.com
 35 | Walkers LTD              | UK      | 0131 871 555      | info@walkersshortbread.com
 36 | Safon                    | Spain   | 0034 972 26 83 04 | info@safon.com
 37 | Elgate                   | UK      | 0131 268 304      | info@elgate.com
 33 | John Morrison Kiltmakers | UK      | 0131 556 473      | queries@johnmorrisonkiltmakers.com
 38 | Elgate                   | UK      | 0131 871 999      | info@elgate.com
 39 | Heather Gems             | UK      | 0131 871 7556     | info@elgate.com
 40 | Edinburgh Wool LTD       | UK      | 0241 998 7556     | queries@edinburghwool.com
 41 | Toucan Jewellery         | UK      | 0127 342 8896     | info@toucan.com
(9 rows)

[shop_inventory=# UPDATE manufacturers SET email = 'info@heathergems.com' WHERE id = 39;
UPDATE 1
[shop_inventory=# SELECT * FROM manufacturers;
 id |           name           | country |       phone       |               email
----+--------------------------+---------+-------------------+-----------------------------------
 34 | Aran Tradition           | Ireland | 0121 006 885      | info@arantradition.com
 35 | Walkers LTD              | UK      | 0131 871 555      | info@walkersshortbread.com
 36 | Safon                    | Spain   | 0034 972 26 83 04 | info@safon.com
 37 | Elgate                   | UK      | 0131 268 304      | info@elgate.com
 33 | John Morrison Kiltmakers | UK      | 0131 556 473      | queries@johnmorrisonkiltmakers.com
 38 | Elgate                   | UK      | 0131 871 999      | info@elgate.com
 40 | Edinburgh Wool LTD       | UK      | 0241 998 7556     | queries@edinburghwool.com
 41 | Toucan Jewellery         | UK      | 0127 342 8896     | info@toucan.com
 39 | Heather Gems             | UK      | 0131 871 7556     | info@heathergems.com
(9 rows)

shop_inventory=#
```

Description here

Here, it can be seen the user requesting for information and modifying that information. At the top, there are two screenshots of the user editing data previously save in the database and a confirmation of this same data updated. As well, the screenshot below shows the interpreter psql run in the command line to demonstrate data being requested and processed correctly in the database.

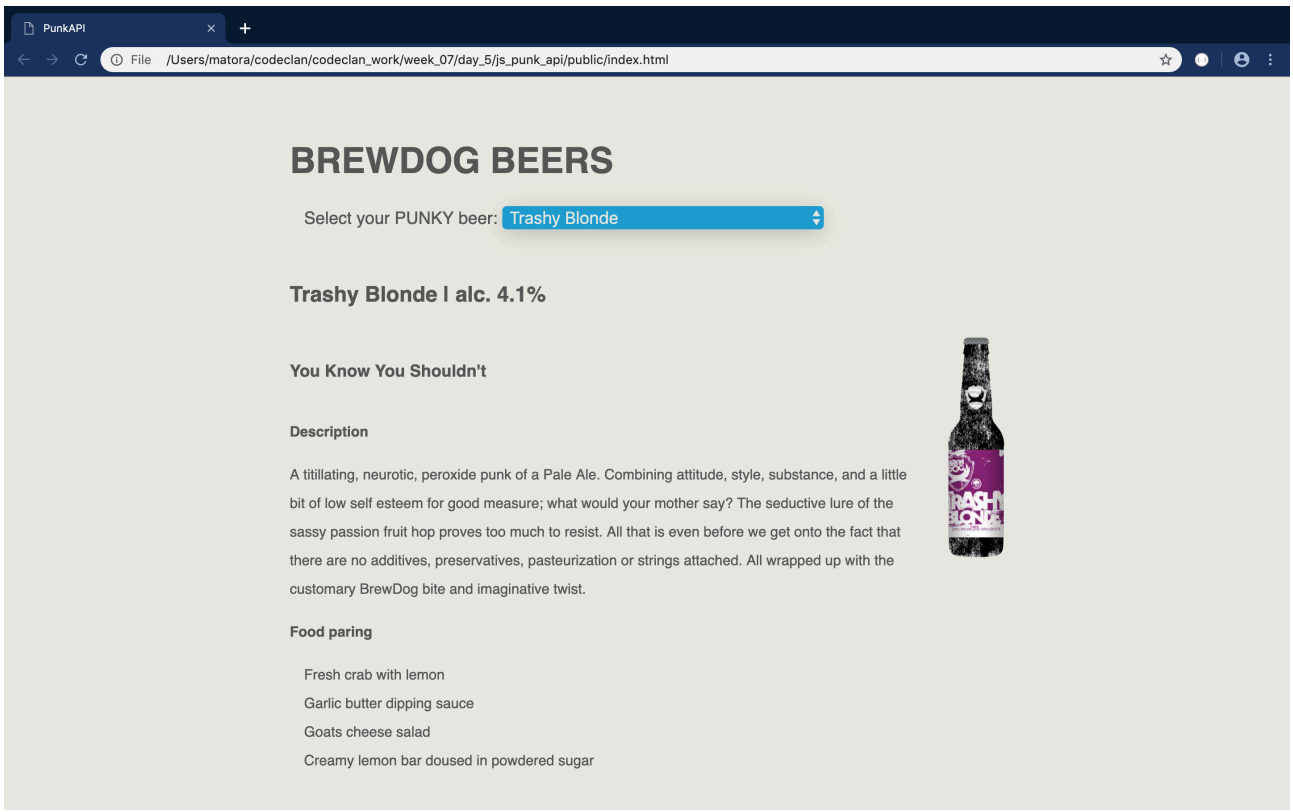
Week 7

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

Paste Screenshot here

```
const PubSub = require('../helpers/pub_sub.js');
const RequestHelper = require('../helpers/
request_helper.js');

const Beers = function () {
  this.data = null;
};

Beers.prototype.getData = function () {
  const requestHelper = new RequestHelper('https://
  api.punkapi.com/v2/beers');

  requestHelper.get( (beersData) => {
    this.data = beersData;
    PubSub.publish('Beers:all-beers-loaded',
    this.data);
  });
};
```

```
const RequestHelper = function (url) {
  this.url = url;
};

RequestHelper.prototype.get = function(onComplete) {
  const xhr = new XMLHttpRequest();

  xhr.addEventListener('load', () => {
    if (xhr.status !== 200) return;
    const jsonString = xhr.responseText;
    const data = JSON.parse(jsonString);
    onComplete(data);
  });

  xhr.open('GET', this.url);
  xhr.setRequestHeader('Accept', 'application/json');
  xhr.send();
};
```



Description here

These two screenshots show an API used for this project. The task here was to create an application that makes a request to an API and displays the data. As it can bee seen in the first image, this programs uses Brewdog beers API ('https://api.punkapi.com/v2/beers') to make the request to the server and get all the data or information to display what we are interested. So, on the right of the top image, we can see a constructor to handle the request and so it prototypes a function called get() in which we create a new XMLHttpRequest() and so it loads the data, if everything okay, then we JSON.parse the response, it passes the data to a callback function and finally, open the url (using GET method), sets the request header and sends it. On the left of the top picture, we can see the model for Beers that prototypes a method called getData(). This function create a new RequestHelper that takes as argument the url of the API to be implemented in our program through which the data is obtained. Thus, the picture below uses the information taken from the API and displays it in a plain JS frontend app. This mini project displays the information for the beer selected from the select dropdown element.

Week 8

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

Paste Screenshot here

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.

Guess You?

This is a full-stack JavaScript web application game based on the boardgame 'Guess Who?'.

MVP

A user should be able to:

- View all characters cards.
- Select a question from a list. The selected question by the user will affect the character card view.
- View the result of the game.

Extensions

A user should be able to:

- Have a turn time limit.
- Provide with some feedback after each selected question.
- View animation when card is out of play.
- Remove questions from the select dropdown after they have been asked.

Advanced Extensions

The app should be:

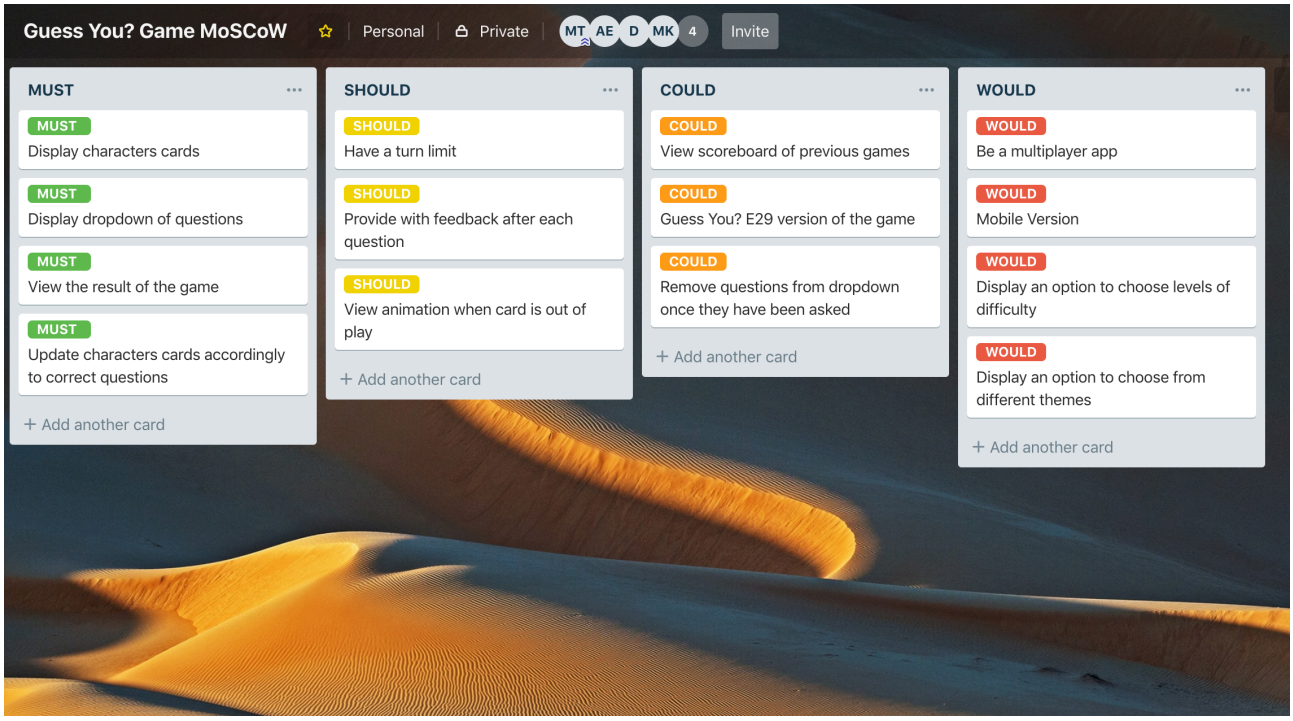
- View score board of previous games.
- Have an option to select levels of difficulty.
- A multiplayer game.

Description here

These two screenshots show the project brief chosen for the JS group project. We chose the brief to the browser game which allows us the freedom to theme it in whatever we would like to work on. We all decided then to elaborate our own brief based on the game ‘Guess Who?’ And, we decided to give it a twist into ‘Guess You? E29 version. Thus, we planned our MVP and Extensions. Due to the limited time, we could not work on advanced extension but we met the goals of MVP and extensions. So, when planning the MVP for the brief, we cover the most basic functionality and from the progress and keep adding features. Our MVP for the game should be able to display the cards with all characters, should be able to allow to select a question to ask and should be able to display result of the game.

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

Paste Screenshot here



Description here

This is a screenshot of the MOSCOW board for the group project. Here, we see the main features our game should as well as those we would like to have. We use this MOSCOW board in order to build and create the brief for the project game.

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

Paste Screenshot here

Acceptance Criteria	Expected Result	Pass/Fail
A user is able to add new items to the list of stock	The details of a new item are saved in the database when the user completes the form and clicks'Add' button.	Pass

Acceptance Criteria	Expected Result	Pass/Fail
A user is able to edit/modify details of existing stock items in the list of stock	When the user clicks the 'Edit' button, the details of that specific item are filled in the form and can be modified by the user and saved back to the database when clicking 'Save' button.	Pass
A user is able to delete items from the list of stock	The details of an item are deleted from the list and database when the user clicks the 'Delete' button for that item in the list.	Pass
A user is able to register new manufacturer to the list of contacts	The details of a new manufacturer are saved in the database when the user completes the form and clicks 'Add' button	Pass
A user is able to remove manufacturers from the list of contacts	When the user clicks the 'Delete' button for a specific manufacturer in the list, the details for that manufacturer is removed from the list and the database, too.	Pass
A user is able to assign manufacturers to stock items	The user is able to assigns a manufacturer to a new stock items when the form is being completed and the 'Add' button is clicked.	Pass
A user is able to view all items from the list of stock in a single page	The user is able to view in a single page the list of stock with all the items, when the user clicks in 'Inventory' at the top in the navigation bar.	Pass
A user is able to view stock levels indicated by different colours	The user can easily identify levels of stock determined by colours when the users clicks 'Inventory' in the nav bar and can view all items in the stock list.	Pass
A user is able to group the items in the stock list by categories	The user can view current categories of stock items when navigating to 'Category' page.	Pass

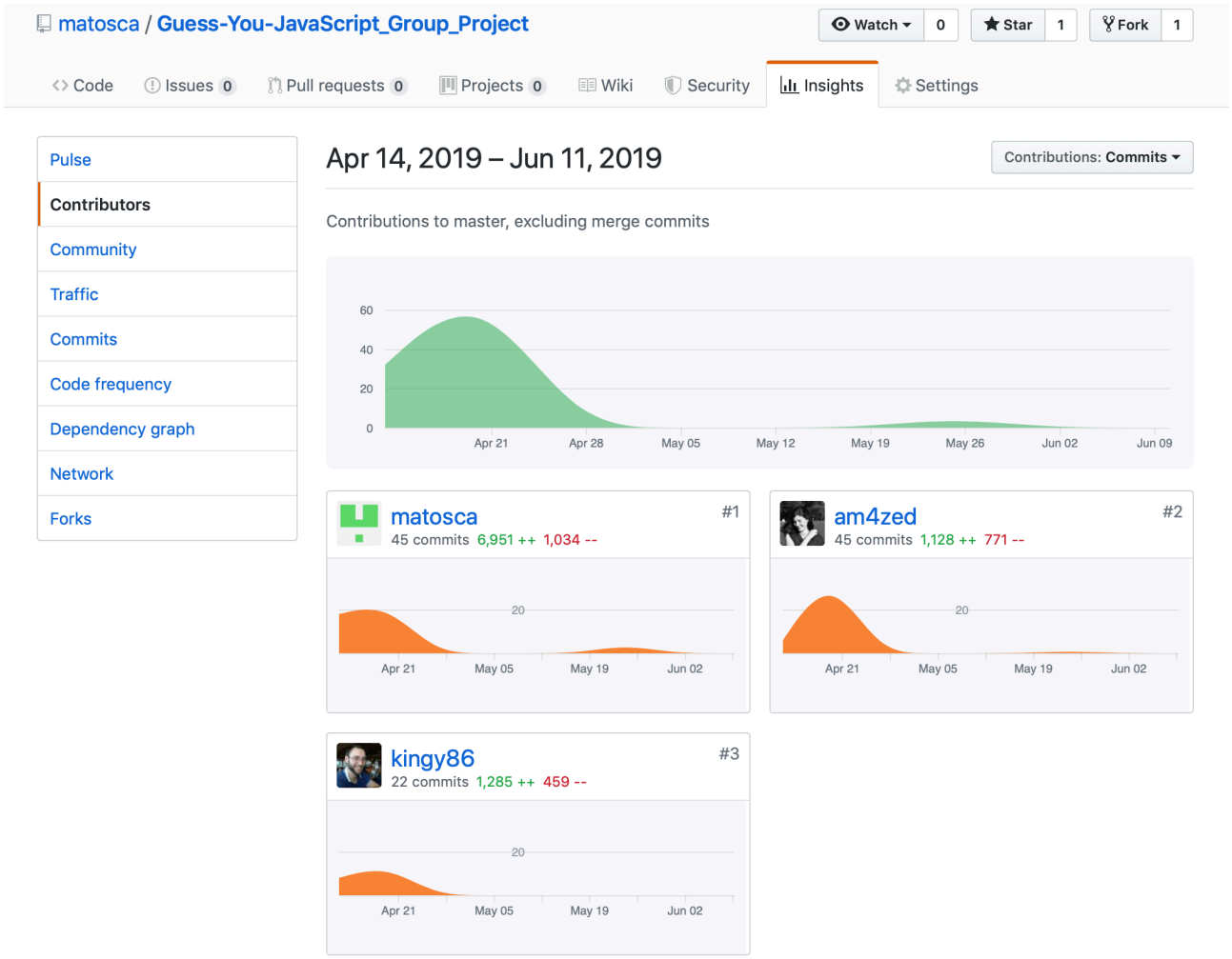
Description here

This is an acceptance criteria and test plan used to define the requirements for a stock management web app developed for my Ruby solo project. These statements are written from the point of view of the user (in this case, a shop keeper) to determine if the functionalities of the app are working as it is expected. This process is used closely with user stories. While user stories are used to explain the roles of users in a system and what they intend to accomplish when doing a task in the software, acceptance criteria plan ensures that those user stories are completed as expected.

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.

Paste Screenshot here



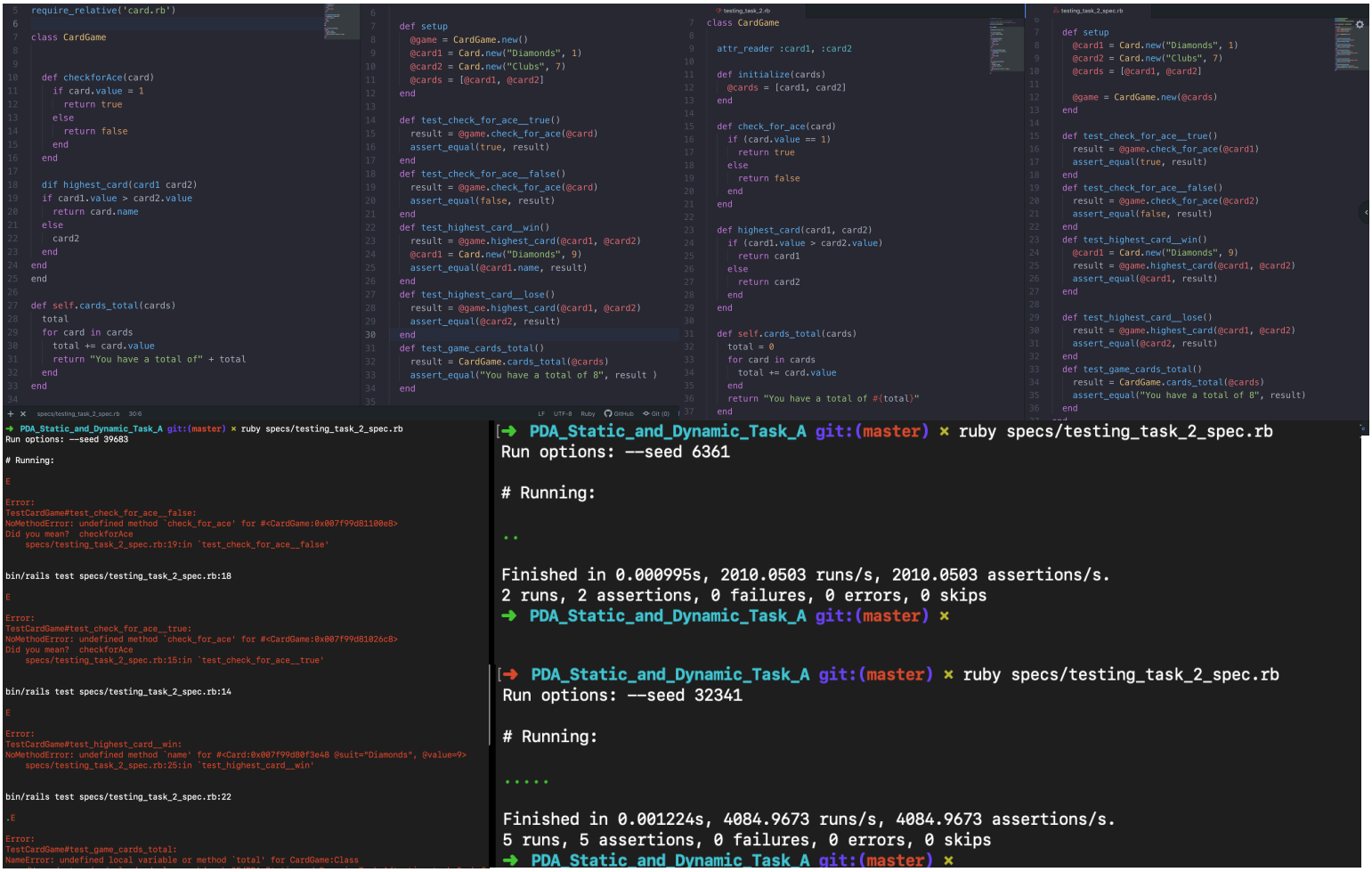
Description here

This is a screenshot of the contributor's page on Github from my group JavaScript project. My team and I worked for a period of a week developing a full-stack JavaScript web game app, Guess who?. During that spring, we developed a game based on cards using PubSub pattern with JS, Express.js and MongoDB in order to build our own API.

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

Paste Screenshot here



Description here

Here, a Static Testing was carried out where the main task was to comment the errors found in the Ruby file testing_task_2. Then, a Dynamic Testing was carried out where the errors commented were corrected. In order to correct the code from testing_task_2, a spec file was prepared where it can be seen that the tests are throwing errors and/or failing before correcting them; and then, the code has been correctly written and the tests are now passing.

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

Paste Screenshot here

```
public class Plane {
    private PlaneType planeType;
    private String airline;
    private ArrayList<Person> passengers;
    private int capacity;

    public Plane(PlaneType planeType, String airline, int capacity) {
        this.planeType = planeType;
        this.airline = airline;
        this.passengers = new ArrayList<>();
        this.capacity = capacity;
    }

    public PlaneType getPlaneType() { return planeType; }

    public String getAirline() { return airline; }

    public int getCountOfPassengers() { return passengers.size(); }

    public void boarding(Person passenger) { passengers.add(passenger); }

    public int getCapacity() { return this.capacity; }
}
public class Flight {

    private Plane plane;
    private String flightNumber;
    private String destination;
    private int flightCapacity;

    public Flight(String flightNumber, String destination, int flightCapacity) {
        this.flightNumber = flightNumber;
        this.destination = destination;
        this.plane = plane;
        this.flightCapacity = flightCapacity;
    }

    public Plane getPlane() { return this.plane; }

    public void setPlane(Plane plane) { this.plane = plane; }

    public String getFlightNumber() { return flightNumber; }

    public String getDestination() { return destination; }

    public int getFlightCapacity() { return this.flightCapacity; }
}
```

Description here

Encapsulation describes the idea of bundling the data (variables) and code acting on the data (methods) together as a single unit (e.g. a class in Java). This means that the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. In the piece of code above, encapsulation have been achieved by declaring private variables for the class Plane. This allows the Plane class only to accessed their own data. By providing getters and setters, the data (variables) of the Plane class can be accessed by others. In this case, only getters have been created to allow other classes that interacts in this program to read the data but not modify it. This exercise consisted in building a system for an airport to manage its flights. Here, the class Plane has been created with the following properties: planeType (Enum PlaneType), airline (String), passengers (array-list holding objects of class Person), and capacity (int). This will allow the Flight class to create new flight objects which will have a plane (Plane) assigned, and so the Airport to manage flights (Flight).

Week 12

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

Paste Screenshot here

```java
package FantasyAdventure.Intefaces;

public interface IDamage {

    void attack(IDefend defend);
}
```

```java
public class Game {

    private Room room;
    private ArrayList<IDefend> players;
    private ArrayList<IDamage> enemies;

    public Game(Room room) {
        this.room = room;
        this.players = players;
        this.enemies = enemies;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }

    public ArrayList<IDefend> getPlayers() {
        return players;
    }

    public void setPlayers(ArrayList<IDefend> players) {
        this.players = players;
    }

    public ArrayList<IDamage> getEnemies() {
        return enemies;
    }

    public void setEnemies(ArrayList<IDamage> enemies) {
        this.enemies = enemies;
    }
}
```

```java
public abstract class Enemy implements IDefend, IDamage {

    private int healthPoints;
    private int armour;
    private int damage;

    public Enemy(int healthPoints, int armour, int damage) {
        this.healthPoints = healthPoints;
        this.armour = armour;
        this.damage = damage;
    }

    public int getHealthPoints() { return healthPoints; }

    public void setHealthPoints(int healthPoints) {
        this.healthPoints = healthPoints;
    }

    public int getArmour() { return armour; }

    public int getDamage() { return damage; }

    public void defend(int damage) {
        int initialDamage = armour - damage;

        if (initialDamage > 0) {
            this.healthPoints -= initialDamage;
        } else {
            this.healthPoints += initialDamage;
        }
    }

    public void attack(IDefend player) { player.defend(damage); }
```

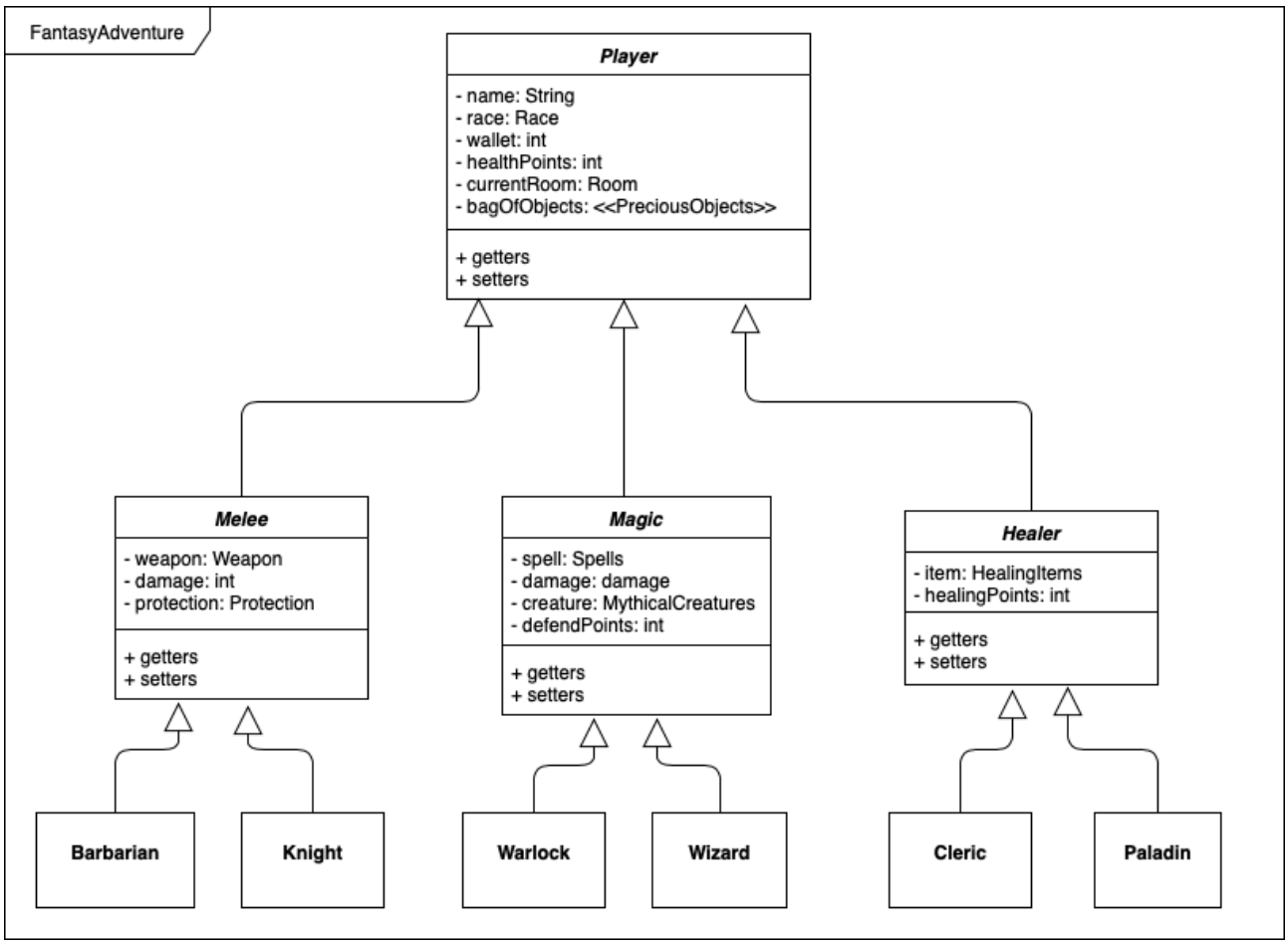
```java
public abstract class Melee extends Player implements IDamage, IDefend {

    private Weapons weapon;
    private int damage;
    private Protection protection;

    public Melee(String name, Race race, int wallet, Weapons weapon, Protection protection)
        super(name, race, wallet);
        this.weapon = weapon;
        this.damage = weapon.getDamage();
        this.protection = protection;
    }

    public Weapons getWeapon() { return weapon; }

    public void changeWeapon(Weapons weapon) { this.weapon = weapon; }

    public int getDamage() { return damage; }

    public Protection getProtection() { return this.protection; }

    public void attack(IDefend enemy ) { enemy.defend(damage); }

    public void defend(int damage){

        int initialDamage = protection.getArmourRating() - damage;

        if (initialDamage > 0) {
            setHealthPoints(getHealthPoints() - initialDamage);
        } else {
            setHealthPoints(getHealthPoints() + initialDamage);
        }
    }
```

Description here

Polymorphism is the ability of an object to take on many forms. This occurs when a parent class reference is used to refer to a child class object. In this example, we have created an interface called IDamage that has a method attack( ) that takes in another interface called IDefend. Then, the class Melee extends from Player and implements both interfaces IDamage and IDefend. Thus, Melee adopts the behaviour of that interface and applies the method attack(IDefend enemy). This IDefend parameter adopts the form of instance object of the class Enemy. Likewise, the superclass Enemy implements IDamage and IDefend and uses the method attack( ) too, but in this case attack takes in IDefend player being an instance object of the class Player.
Therefore, both superclasses have implemented IDamage and IDefend and so their behaviours.

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

Paste Screenshot here



Description here

This is an example of inheritance diagram that represents a part of the planning to build a Fantasy Adventure game. At the top, we have the superclass that have the properties that are common to the subclasses. Also, Player is an abstract class. Then, we have planned to differentiate three main categories for which we created each an abstract class too. Melee, Magic, and Healer inherits all from the parent class Player but, at the same time, they are parent classes. Finally, Barbarian and Knight are child classes that inherits all properties and behaviours from the superclass Melee, Warlock and Wizard are child classes that inherits all properties and behaviours too from the superclass Magic and, lastly, Cleric and Paladin are child classes that inherits all properties and behaviours from the superclass Healer.

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.

Paste Screenshot here

```java
package FantasyAdventure.Enemies;

import FantasyAdventure.Intefaces.IDamage;
import FantasyAdventure.Intefaces.IDefend;

public abstract class Enemy implements IDefend, IDamage {

    private int healthPoints;
    private int armour;
    private int damage;

    public Enemy(int healthPoints, int armour, int damage) {
        this.healthPoints = healthPoints;
        this.armour = armour;
        this.damage = damage;
    }

    public int getHealthPoints() { return healthPoints; }

    public void setHealthPoints(int healthPoints) {
        this.healthPoints = healthPoints;
    }

    public int getArmour() { return armour; }

    public int getDamage() { return damage; }

    public void defend(int damage) {
        int initialDamage = armour - damage;

        if (initialDamage > 0) {
            this.healthPoints -= initialDamage;
        } else {
            this.healthPoints += initialDamage;
        }
    }

    public void attack(IDefend player) { player.defend(damage); }

}
```

```java
package FantasyAdventure.Enemies;

public class Orc extends Enemy{

    public Orc(int healthPoints, int armour, int damage) {
        super(healthPoints, armour, damage);
    }
}
```

```java
public class OrcTest {

    Orc orc;
    Barbarian barbarian;

    @Before
    public void before() {
        orc = new Orc( healthPoints: 50,  armour: 8,  damage: 10);
        barbarian = new Barbarian( name: "Conan", Race.HUMAN,  wallet: 100, Weapons.SWORD,

    }

    @Test
    public void hasHealthPoints() { assertEquals( expected: 50, orc.getHealthPoints()); }

    @Test
    public void hasArmour() { assertEquals( expected: 8, orc.getArmour()); }

    @Test
    public void getDamage() { assertEquals( expected: 10, orc.getDamage()); }

    @Test
    public void canDefendFromAttackWhenDamageMoreThanArmour(){
        orc.defend( damage: 25);
        assertEquals( expected: 33, orc.getHealthPoints());
    }
```

Description here

These screenshots demonstrate the use of inheritance. Here, we can observe a superclass Enemy with the following properties: healthPoints, armour and damage. Enemy class also has getters and setters for their respective data (variables). Just below the first picture, it can be seen another screenshot for the subclass Orc which extends Enemy. By doing that, Orc inherits all properties and methods from the parent class. Finally, the last screenshots shows an instance object of type Orc and how this objects uses the behaviours inherited from the superclass. The methods getHealthPoints( ), getArmour( ), and getDamage( ) demonstrate the usage of the information inherited.

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.

Paste Screenshot here

```java
public void attack(IDefend enemy ) { enemy.defend(damage); }

public void defend(int damage){

    int initialDamage = protection.getArmourRating() - damage;

    if (initialDamage > 0) {
        setHealthPoints(getHealthPoints() - initialDamage);
    } else {
        setHealthPoints(getHealthPoints() + initialDamage);
    }
}
```

1

```java
public void fight(){
    if (getCurrentRoom() != null) {
        Room room = getCurrentRoom();
        Enemy enemy = room.getEnemy();
        while (enemy.getHealthPoints() > 0) {
            attack(enemy);
            defend(enemy.getDamage());
            if (enemy.getHealthPoints() < 0) {
                enemy.setHealthPoints(0);
            }
        }
    }
}
```

2

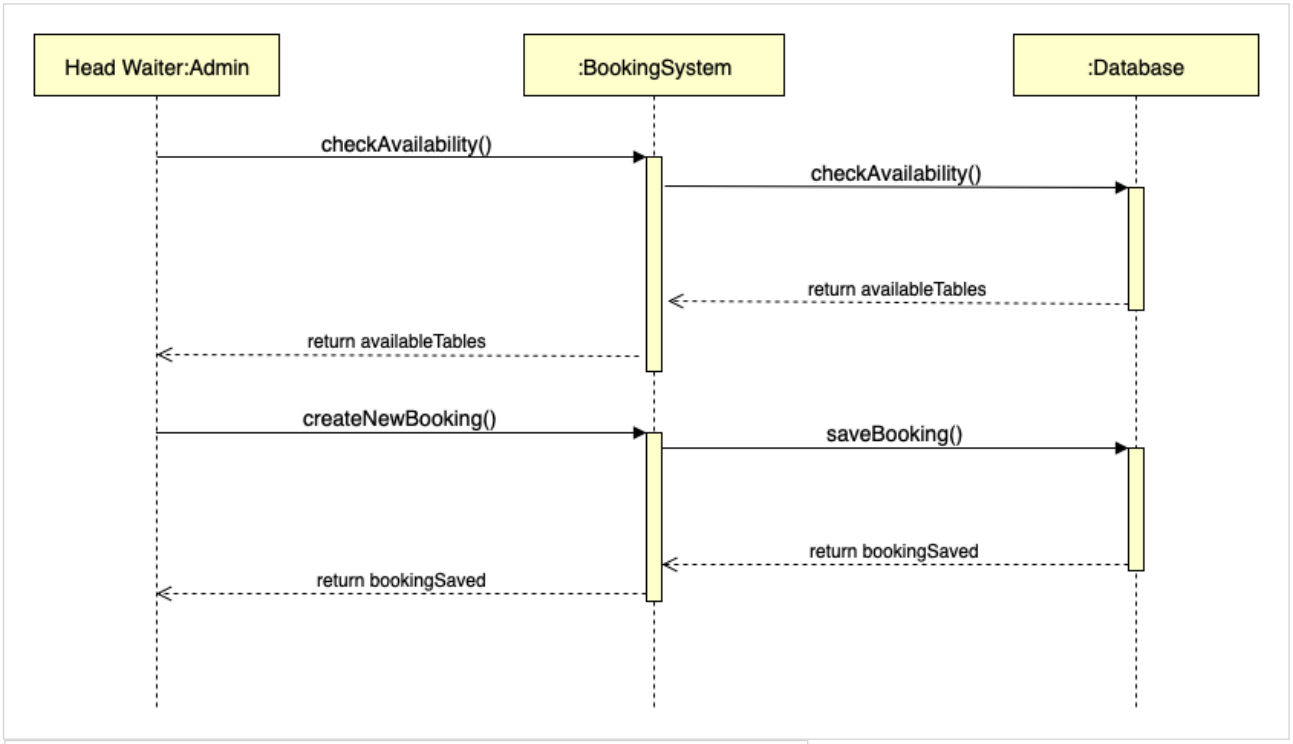
```java
@Test
public void cannotFightEnemyUnlessInRoom() { assertEquals( expected: 100, barbarian.getHealthPoints());

@Test
public void canFightEnemyInRoom(){
    barbarian.enterRoom(enemyRoom);
    barbarian.fight();
    assertEquals( expected: 76, barbarian.getHealthPoints());
    assertEquals( expected: 0, orc.getHealthPoints());
}
```

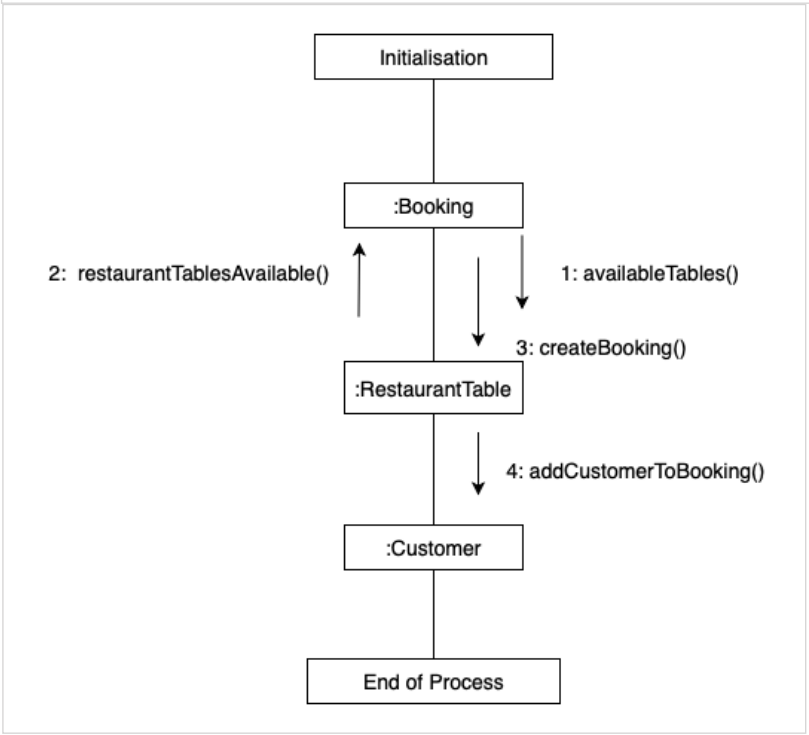
Description here

These two screenshots form part of the same Java project. The main task was to model a fantasy adventure game in which players could have health points and weapons or spells and should be able to get into rooms where they should either collect treasures or to defeat enemies. The reason why I have chosen these two algorithms is mostly because they are a good example representing the main functionality of the game model. Also, they are my most recent example of algorithm, I felt happy to produce due to the challenge Java has presented to me while learning it during the third module of the course.
The example nº 1 is a function that allows players to defend from an enemy's attack/damage. This function defend( ) takes an argument of type int which is the damage of the enemy's attack. Then, I have assigned the result of subtracting the damage to the player's protection to an int type variable called initialDamage and used it to be compared in the next control flow statement. Thus, if the initialDamage is greater than zero (i.e. positive number) will then subtract the amount of damage to the player's health points otherwise it will add the damage to the player's health points when the initialDamage is lesser than zero (i.e. negative number).
The example nº 2 is a function allows players to fight an enemy. In order to let the player fight, a player must be in a room and so the first thing this function does is to check that the room property in the class Player is not null (if it is, the function won't run; if it's not null, it will run). Once the a player is in a room, a while loop will be executed. So, while the enemy's healthPoints are greater than 0, the player will attack (enemy) and defend "from"(enemy's damage) and if the enemy's healthPoints goes below zero then we will set the enemy's healthPoints to zero avoiding negative numbers.
The last screenshot shows two test for the algorithm fight(): one, when player is not in room and second, when player is in a room. So, when a player is in a room the player can fight the enemy and the assertions just display how the player and enemy, both lose healthPoints.

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

Paste Screenshot here



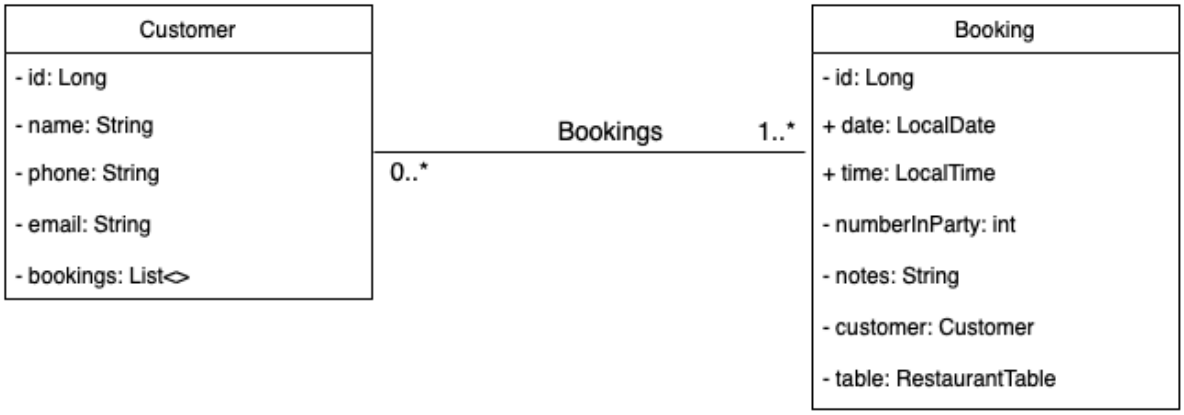


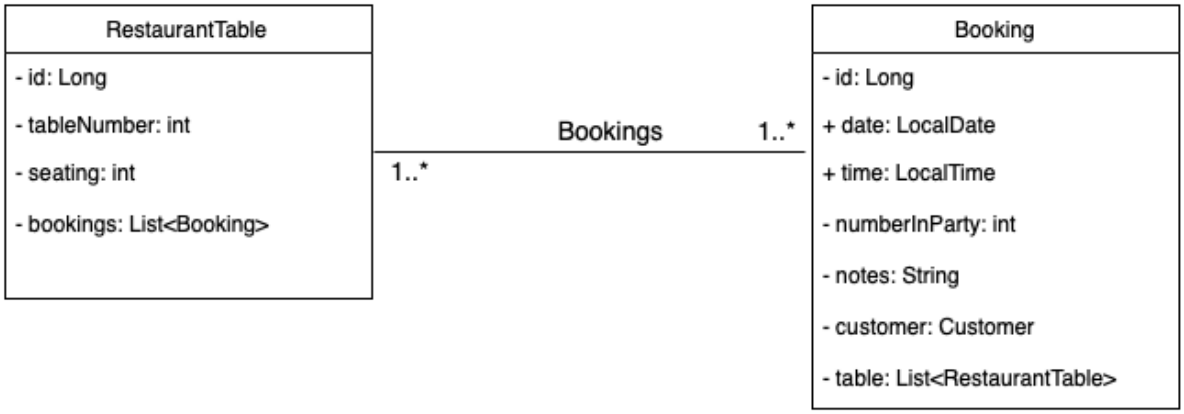
Description here

These are two examples of system integration diagrams: one, sequence diagram and other, collaboration diagram. The first diagram is a sequence diagram that represents the process through which a staff of a restaurant makes a new reservation. The first thing our booking system does is checking the availability of the restaurant tables for a specific date and time. Once, our database has resolved the availability and return the tables available, the admin can then proceed to create a new booking in the system that is saved in the database. Finally, a confirmation of the booking saved is display back to the admin view. The second diagram is a collaboration diagram that represents the process for making a new booking. Again, our booking system first checks for availability and then, continues to create booking and add the customer to the booking just created.

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

Paste Screenshot here





Description here

These are examples of object diagrams that represents the relationships customer and booking and restaurantTable and booking classes for a restaurant booking system. As it can be seen, a Customer has id, name, phone, email and it holds a one-to-many relationship with Booking. A Booking has id, date, time, number in party, notes, a Customer details and a restaurant table. A booking holds a many-to-one relationship with Customer. The second object diagram shows a many-to-many relationship between RestaurantTable and Booking. As it can be observed RestaurantTable holds an arrayList of Booking; likewise, Booking holds another arrayList of RestaurantTable and so they show that bookings can have many tables assigned and restaurant tables can be assigned to many bookings.

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

Paste Screenshot here

BUG/ERROR	SOLUTION	DATE
Uncaught TypeError: Cannot read property 'null' of undefined. This error occurs in GRAB react app because the state in the constructor of the main container Restaurant was set up to 'null'.	The solution for this error was setting up the state in the constructor to [ ] empty array instead of null.	06/06/2019

Access to fetch at 'http://localhost:8080/' from origin 'null' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.	The solution for this error was fixed by creating a SpringDataRestGlobalConfig file to allow our localhost:8080 to send the data from API to localhost:3000. And, modifying the fetch request in our react app:
	headers: {
	"Content-Type": "application/json",
	Accept: "application/json",
	"Access-Control-Allow-Origin": "http://localhost:3000"
	}
		04/06/2019

Description here

This is a report of errors and bugs occurred during our Java group project. Our project was designing a restaurant booking system for staff only. While working on the development of the app (back-end built using Java, Spring and Hibernate and front-end built using JavaScript and React), we constantly encountered with this two main errors. The first one was in our React app when rendering a new component and it was mostly caused by the state setup in the constructor of our main container. The second occurred when we started to make requests from localhost: 3000 to localhost:8080 and it was soon solved by creating a configuration file of CORS and allowing header to access CORS.