

## **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**

<b>Unit</b>	<b>Ref</b>	<b>Evidence</b>
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

Unit	Ref	Evidence
		<p>Description: BusStop class created with empty queue array</p> <pre> 1  class BusStop 2 3     attr_reader :stop_name, :queue 4 5     def initialize(name) 6         @stop_name = name 7         @queue = [] 8     end 9 10 </pre>
		<p>Function add_to_queue takes in a person object and adds it to the queue array</p> <pre> 14 15  def add_to_queue(person) 16      @queue.push(person) 17  end 18 </pre>
		<p>Test to see add_to_queue function works</p> <pre> def test_add_person_to_queue     @bus_stop.add_to_queue(@person)     assert_equal(1, @bus_stop.count_queue) end </pre>
		<p>Test passes and shows person object with name “Hugh” and age 43 inside bus stop’s queue array</p> <pre> # Running:  [#&lt;Person:0x007fefdc81e600 @name="Hugh", @age=43&gt;] .. Finished in 0.000779s, 2567.3941 runs/s, 2567.3941 assertions/s.  2 runs, 2 assertions, 0 failures, 0 errors, 0 skips ➔ specs </pre>

<b>Unit</b>	<b>Ref</b>	<b>Evidence</b>
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

Unit	Ref	Evidence
		<p><b>Description: Pet_shop has an array of pet hashes</b></p> <pre>@pet_shop = {   pets: [     {       name: "Sir Percy",       pet_type: :cat,       breed: "British Shorthair",       price: 500     },     {       name: "Tristan",       pet_type: :dog,       breed: "Basset Hound",       price: 800,     },     {       name: "Merlin",       pet_type: :cat,       breed: "Egyptian Mau",       price: 1500,     }   ],   admin: {     total_cash: 1000,     pets_sold: 0,   },   name: "Camelot of Pets" }</pre>
		<p><b>Function removes pet hash from an array</b></p> <pre>def remove_pet_by_name(pet_shop, pet_name)   pet_to_delete = find_pet_by_name(pet_shop, pet_name)   pet_shop[:pets].delete(pet_to_delete) end</pre>

Test checks that remove\_pet\_by\_name works

```
def test_remove_pet_by_name
  remove_pet_by_name(@pet_shop, "Sir Percy")
  pet = find_pet_by_name(@pet_shop,"Sir Percy")
  assert_nil(pet)
end
```

Test passes

```
# Running:
.

Finished in 0.000884s, 1131.2217 runs/s, 1131.2217 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
→ specs git:(master) ✘
```

### Week 3

<b>Unit</b>	<b>Ref</b>	<b>Evidence</b>
-------------	------------	-----------------

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: Function searches database for vehicle of given id

```
def Vehicle.find(id)
    sql = "SELECT * FROM vehicles WHERE id = $1"
    values = [id]
    result = SqlRunner.run(sql, values)[0]
    return Vehicle.new(result)
end
```

Vehicles in sql database

id	name	category	on_hire	image
1	KITT	screen_stars	false	KITT.jpg
3	Crane	heavy_plant	false	crane.jpg
5	Bulldozer	heavy_plant	true	bulldozer.jpeg
2	Tank	military	true	tank.jpeg
4	Penny Farthing	circus	true	penny_farthing.jpg
6	Velocipede	circus	true	velocipede.jpg
7	Airwolf	screen_stars	false	airwolf.jpg

(7 rows)

Result of function running: Airwolf details displayed in form for editing

[• Home](#)     
 [• Vehicles](#)     
 [• Customers](#)     
 [• Rentals](#)

# AIRWOLF



**Name:**      
 **Category:**      
 **Enter photo filename:**

**On hire?**  Yes  No



Unit	Ref	Evidence
I&T	I.T.4	<p>Demonstrate sorting data in a program. Take screenshots of:</p> <ul style="list-style-type: none"> <li>*Function that sorts data</li> <li>*The result of the function running</li> </ul>
		<p><b>Description: Function gets all vehicles on hire from database and sorts them alphabetically</b></p> <pre> 58 59  #Function to return all vehicles available for rent 60  def Vehicle.available() 61    sql = "SELECT * FROM vehicles WHERE on_hire = \$1 ORDER BY name ASC" 62    values = ["false"] 63    results = SqlRunner.run(sql, values) 64    return results.map{  vehicle  Vehicle.new(vehicle) } 65  end 66 </pre> <p><b>Vehicles in database in order of id</b></p> <pre> ; +-----+-----+-----+-----+   id       name        category   on_hire         image +-----+-----+-----+-----+   1    KITT            screen_stars   false     KITT.jpg   3    Crane            heavy_plant    false     crane.jpg   5    Bulldozer        heavy_plant    true      bulldozer.jpeg   2    Tank              military       true      tank.jpeg   4    Penny Farthing   circus         true      penny_farthing.jpg   6    Velocipede        circus         true      velocipede.jpg   7    Airwolf           screen_stars   false     airwolf.jpg (7 rows) </pre>

Result of function running - Vehicles displayed in app in alphabetical order

- [Home](#)
- [Vehicles](#)
- [Customers](#)
- [Rentals](#)

# VEHICLES

- [available](#)
- [on-hire](#)
- [Add new vehicle](#)

**Categories**

- [screen\\_stars](#)
- [heavy\\_plant](#)
- [military](#)
- [circus](#)



**AIRWOLF**

**Airwolf**

- [Show Airwolf details](#)
- [Rent Airwolf](#)

---



**Bulldozer**

- [Show Bulldozer details](#)

---

Unit	Ref	Evidence
A&D	A.D.1	<p>A Use Case Diagram</p> <p>Description: Use case diagram from rental shop project</p> <p>The diagram illustrates a Use Case Diagram for a rental shop project. It features a stick figure icon labeled "manager" on the left, a central rounded rectangle labeled "use case" containing a list of eight interactions, and a rectangular box labeled "database" on the right. Multiple lines connect the manager to the use case, and from the use case, lines radiate to the database.</p> <pre>graph LR; manager((manager)) --- useCase[use case]; useCase --- database[database];</pre> <ul style="list-style-type: none"><li>check stock</li><li>check rentals</li><li>add vehicle</li><li>create customer</li><li>create rental</li><li>edit customer</li><li>edit rental</li><li>edit vehicle</li></ul>

Unit	Ref	Evidence
A&D	A.D.2	<p>A Class Diagram</p> <p><b>Description: Vehicle, Rental and Customer class diagrams from rental shop project</b></p> <pre> classDiagram     class Vehicle {         id: INT         name: "string"         on-hire: boolean     }     class Rental {         id: int         customer-id: int         vehicle-id: int     }     class Customer {         id: int         first-name: "string"         last-name: "string"     }     Vehicle &lt; -- Rental     </pre>

Unit	Ref	Evidence
A&D	A.D.3	<p>An Object Diagram</p> <p><b>Description: Object diagram showing customer Ian Tough renting a tank</b></p> <pre> objectDiagram     Tank{id: 01, name: "tank", on-hire: yes} --o Rental{01}     Rental{01} --o Customer{03, first-name: "Ian", last-name: "Tough"}     </pre>

Unit	Ref	Evidence
A&D	A.D.4	An Activity Diagram
		<p><b>Description: Activity diagram for Big App Yourself project</b></p> <pre> graph TD     Start(( )) --&gt; UserNavegation[Navigates to Big App Yourself home page]     UserNavegation --&gt; ClicksRegister[Clicks register button]     ClicksRegister --&gt; FillsForm[Fills in registration form]     FillsForm --&gt; ClicksSave[Clicks save]     ClicksSave --&gt; SavesDatabase[Saves user details to database]     SavesDatabase --&gt; ComplimentsUser[Compliments user according to schedule]     ComplimentsUser --&gt; End((( )))     ClicksRegister --&gt; OpenForm[opens registration form]     FillsForm --&gt; OpenForm     ClicksSave --&gt; OpenForm     </pre> <p>The activity diagram illustrates the process of a user registering on the Big App Yourself platform. It is divided into two main vertical sections: 'User' on the left and 'Big App Yourself' on the right. The process begins with the user navigating to the home page. They then click the 'register' button, which triggers the system to open a registration form. The user fills in the form and clicks 'save'. This action saves the user details to the database and concludes with the system complimenting the user according to a scheduled plan. The final step is a red-circled end state.</p>

Unit	Ref	Evidence																					
A&D		<p>Produce an Implementations Constraints plan detailing the following factors:</p> <ul style="list-style-type: none"> <li>*Hardware and software platforms</li> <li>*Performance requirements</li> <li>*Persistent storage and transactions</li> <li>*Usability</li> <li>*Budgets</li> <li>*Time</li> </ul>																					
		<p><b>Description: Implementation and constraints plan for Big App Yourself</b></p> <table border="1"> <thead> <tr> <th>Constraint Category</th><th>Implementation Constraint</th><th>Solution</th></tr> </thead> <tbody> <tr> <td>Hardware and Software Platforms</td><td>Some limitations in date/time functionality supported by web browsers.</td><td>Use more standard date/time functionality supported universally.</td></tr> <tr> <td>Performance Requirements</td><td>Web server is local and not capable of supporting larger user base</td><td>Scaleable web server would be a solution</td></tr> <tr> <td>Persistent Storage and Transactions</td><td>Database is local and not capable of supporting larger user base</td><td>Scaleable database would be a solution</td></tr> <tr> <td>Usability</td><td>Some limitations on mobile device usage, smaller screens</td><td>Implement better mobile functionality with CSS</td></tr> <tr> <td>Budgets</td><td>Scaleable web server and database costs money</td><td>Investment would support scalability.</td></tr> <tr> <td>Time Limitations</td><td>App functionality is limited due to time constraints</td><td>More time spent on developing functionality</td></tr> </tbody> </table>	Constraint Category	Implementation Constraint	Solution	Hardware and Software Platforms	Some limitations in date/time functionality supported by web browsers.	Use more standard date/time functionality supported universally.	Performance Requirements	Web server is local and not capable of supporting larger user base	Scaleable web server would be a solution	Persistent Storage and Transactions	Database is local and not capable of supporting larger user base	Scaleable database would be a solution	Usability	Some limitations on mobile device usage, smaller screens	Implement better mobile functionality with CSS	Budgets	Scaleable web server and database costs money	Investment would support scalability.	Time Limitations	App functionality is limited due to time constraints	More time spent on developing functionality
Constraint Category	Implementation Constraint	Solution																					
Hardware and Software Platforms	Some limitations in date/time functionality supported by web browsers.	Use more standard date/time functionality supported universally.																					
Performance Requirements	Web server is local and not capable of supporting larger user base	Scaleable web server would be a solution																					
Persistent Storage and Transactions	Database is local and not capable of supporting larger user base	Scaleable database would be a solution																					
Usability	Some limitations on mobile device usage, smaller screens	Implement better mobile functionality with CSS																					
Budgets	Scaleable web server and database costs money	Investment would support scalability.																					
Time Limitations	App functionality is limited due to time constraints	More time spent on developing functionality																					

Unit	Ref	Evidence
P	P.5	User Site Map
		<p><b>Description: Site map for rental shop app</b></p> <pre>graph TD; Home[Home] --&gt; Vehicles[Vehicles]; Home --&gt; Rentals[Rentals]; Home --&gt; Customers[Customers]</pre> A site map diagram for a rental shop app. At the top is a box labeled "Home". Three arrows point downwards from "Home" to three separate boxes: "Vehicles" on the left, "Rentals" in the center, and "Customers" on the right. All boxes are white with black outlines.

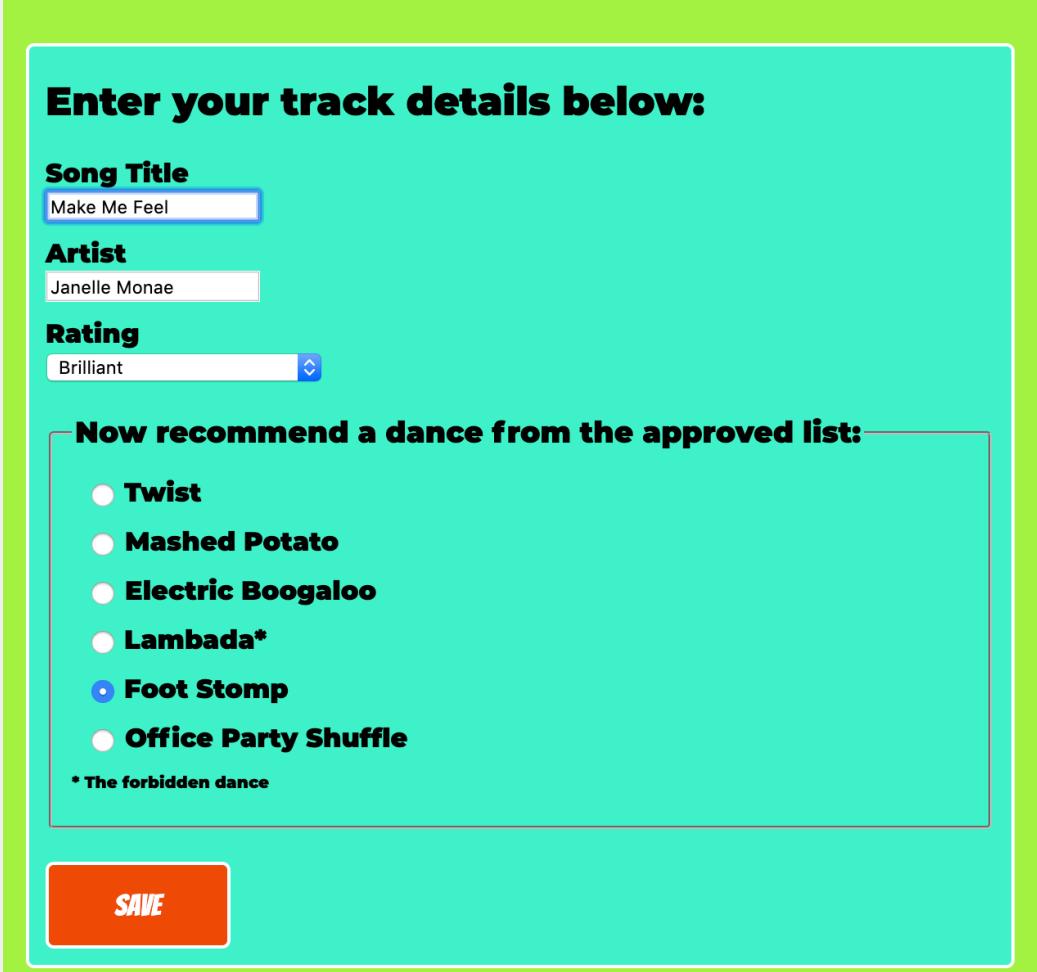
<b>Unit</b>	<b>Ref</b>	<b>Evidence</b>
P	P.6	2 Wireframe Diagrams

Unit	Ref	Evidence
<b>Description: Wireframe for Vehicles view in rental shop project</b>		

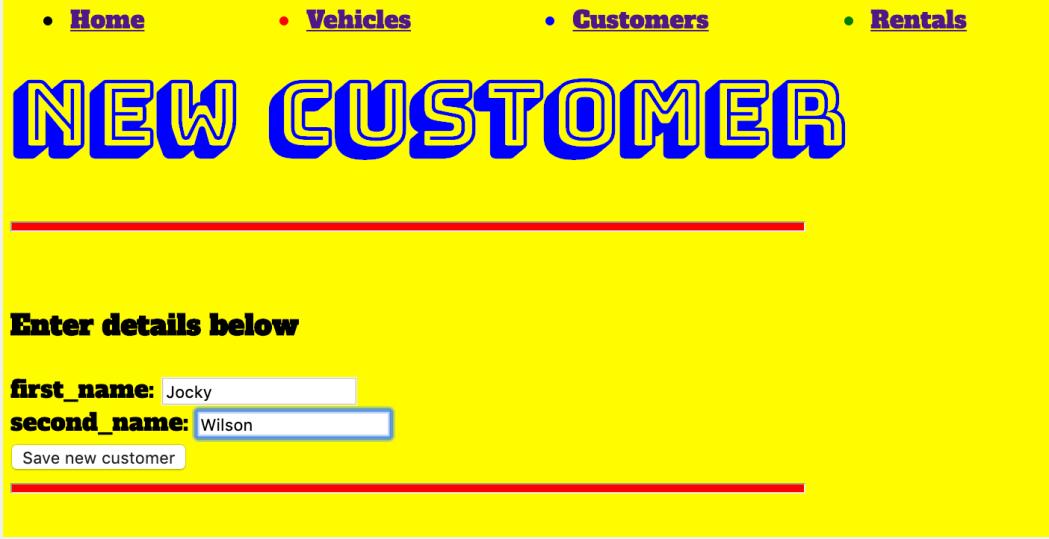
Unit	Ref	Evidence
<b>Wireframe for Rentals view in rental</b>		

Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method
		<p><b>Description: Pseudocode in BusStopTest method</b></p> <pre> class BusStopTest &lt; MiniTest::Test  def setup   @bus_stop = BusStop.new("Duke Street")   @person = Person.new("Hugh", 43) end  def test_get_stop_name   assert_equal("Duke Street", @bus_stop.stop_name) end  # write test for adding person to queue def test_add_person_to_queue   # add person defined in setup to queue   @bus_stop.add_to_queue(@person)   #print out the queue to see if person is in it   p (@bus_stop.queue)   #assert there is now one person in the queue   assert_equal(1, @bus_stop.count_queue) end  end </pre>

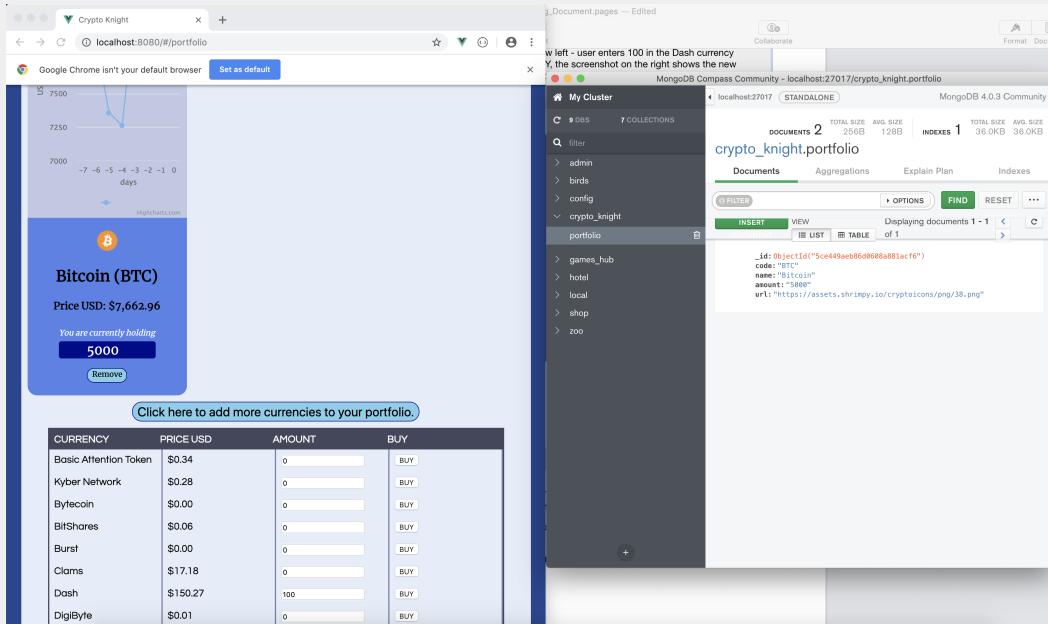
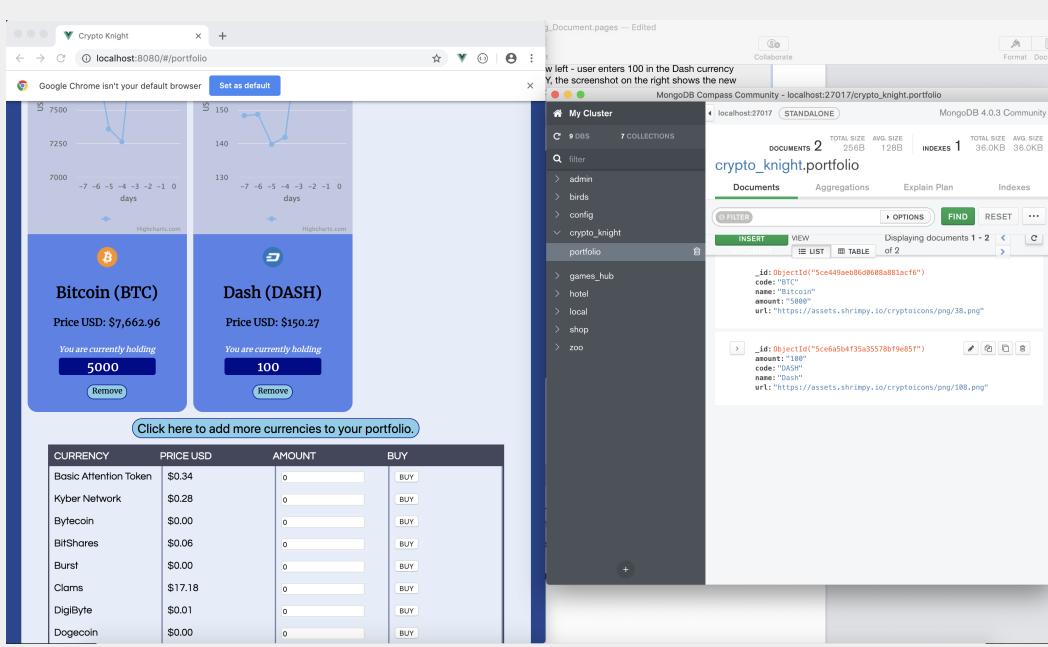
Unit	Ref	Evidence
P	P.13	<p>Show user input being processed according to design requirements. Take a screenshot of:</p> <ul style="list-style-type: none"> <li>* The user inputting something into your program</li> <li>* The user input being saved or used in some way</li> </ul>
		<p>Description: User enters song details and chooses accompanying dance</p>  <p>Enter your track details below:</p> <p><b>Song Title</b> Make Me Feel</p> <p><b>Artist</b> Janelle Monae</p> <p><b>Rating</b> Brilliant</p> <p><b>Now recommend a dance from the approved list:</b></p> <ul style="list-style-type: none"> <li><input type="radio"/> Twist</li> <li><input type="radio"/> Mashed Potato</li> <li><input type="radio"/> Electric Boogaloo</li> <li><input type="radio"/> Lambada*</li> <li><input checked="" type="radio"/> Foot Stomp</li> <li><input type="radio"/> Office Party Shuffle</li> </ul> <p>* The forbidden dance</p> <p><b>SAVE</b></p> <p>Track and dance added to playlist</p>  <h2>Playlist</h2> <ol style="list-style-type: none"> <li>1. Make Me Feel by Janelle Monae (Rated: Brilliant, Recommended dance: The Foot Stomp)</li> </ol> <p><b>DELETE ALL</b></p>

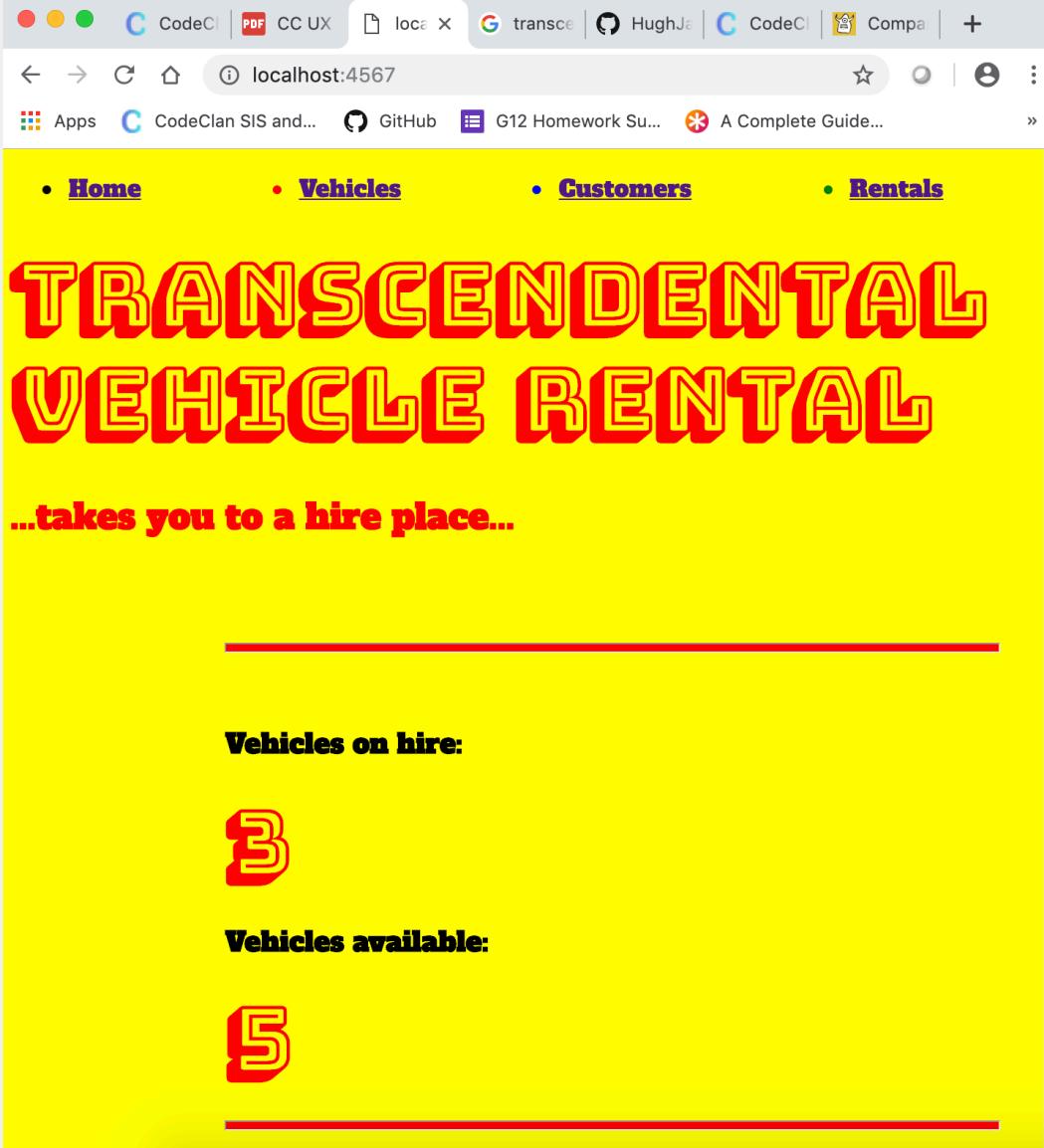


Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved

Unit	Ref	Evidence
		<p>Description: Customer details entered into rental shop app</p> 
		<p>Customer now appears in customer list</p> 
		<p>Customer's details showing in sql database (id 15)</p> <pre>[rental_shop=# SELECT * FROM customers;  id   first_name   second_name ----+-----+-----  1   Lorraine     Kelly  2   Ian           Tough  3   Andy          Murray  4   Francis       McKee  5   Eugene        Kelly  6   Charlie       Reid  7   Mary           Marquis  8   Lorraine     Kelly  9   Ian           Tough 10   Andy          Murray 11   Francis       McKee 12   Eugene        Kelly 13   Charlie       Reid 14   Mary           Marquis 15   Jocky         Wilson (15 rows)</pre>



Unit	Ref	Evidence
P	P.15	<p>Show the correct output of results and feedback to user. Take a screenshot of:</p> <ul style="list-style-type: none"> <li>* The user requesting information or an action to be performed</li> <li>* The user request being processed correctly and demonstrated in the program</li> </ul>
		<p><b>Description: User has entered 100 Dash into form but has not clicked buy. Crypto tracking app and MongoDB show only user's Bitcoin holding.</b></p>  <p>The screenshot displays two side-by-side interfaces. On the left is the 'Crypto Knight' application, which includes a chart for Bitcoin (BTC) price over time, a summary card for Bitcoin holding (Price USD: \$7,662.96, You are currently holding 5000), and a portfolio table. The portfolio table shows various cryptocurrencies with their current price, amount held (all 0), and a 'BUY' button. On the right is the 'MongoDB Compass Community' interface, showing the 'crypto_knight.portfolio' collection with one document inserted. The document details a Bitcoin entry with ID: ObjectId("5ce449ae5b8d0868a881ac1f6"), code: "BTC", name: "Bitcoin", amount: "5000", and url: "https://assets.shrimpy.io/cryptoicons/png/38.png".</p> <p><b>User has now clicked buy, so 100 Dash holding now shows in app and in MongoDB</b></p>  <p>The screenshot shows the same two interfaces after a user has clicked the 'BUY' button for Dash. The app now displays a chart for Dash (DASH) and a portfolio table where the Dash holding has been updated to 100. The MongoDB Compass interface shows two documents in the crypto_knight.portfolio collection, each representing a different cryptocurrency holding (Bitcoin and Dash).</p>

Unit	Ref	Evidence
P	P.11	<p>Take a screenshot of one of your projects where you have worked alone and attach the Github link.</p> <p><b>Description: Screenshot of home page from my rental shop project</b></p> 
		<p><b>Link to GitHub repo</b></p> <p><a href="https://github.com/HughJarvis/Rental_shop">https://github.com/HughJarvis/Rental_shop</a></p>

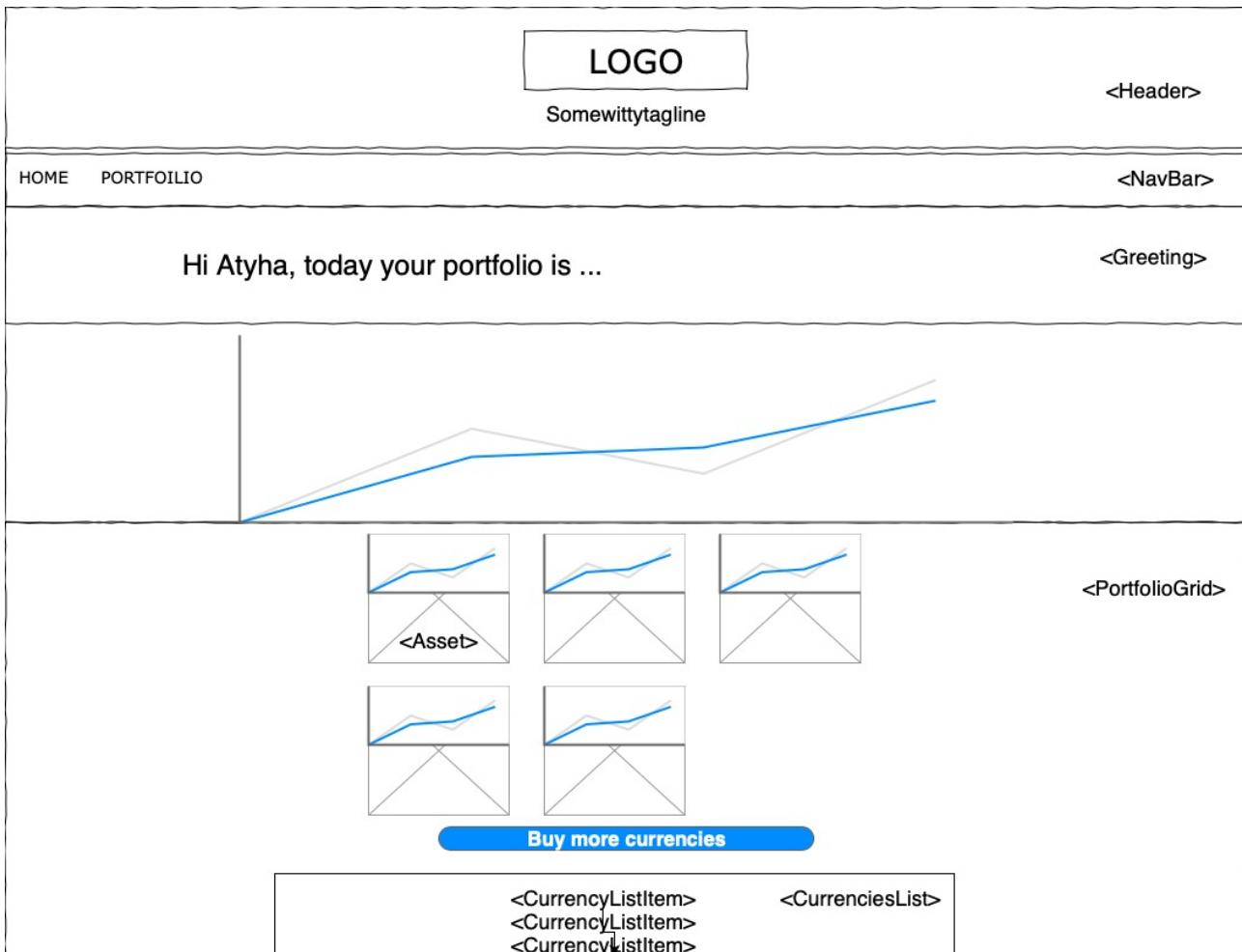
Unit	Ref	Evidence
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running

Unit	Ref	Evidence																		
		<p><b>Fetch method to get info back from Shrimpy currencies API in Crypto Knight portfolio tracker</b></p> <pre>fetchShrimpyTicker: function () {   fetch('https://cors-anywhere.herokuapp.com/https://dev-api.shrimpy.io/v1/exchanges/poloniex/ticker')     .then(res =&gt; res.json())     .then(data =&gt; {       console.log(data);       this.shrimpy = data;     }) }</pre>																		
		<p><b>User adds 10 Litecoin to their portfolio</b></p> <table border="1"> <thead> <tr> <th>CURRENCY</th> <th>PRICE USD</th> <th>AMOUNT</th> <th>BUY</th> </tr> </thead> <tbody> <tr> <td>Litecoin</td> <td>\$118.12</td> <td><input type="text" value="10"/></td> <td><button>BUY</button></td> </tr> <tr> <td>Ardor</td> <td>\$0.11</td> <td><input type="text" value="0"/></td> <td><button>BUY</button></td> </tr> </tbody> </table>	CURRENCY	PRICE USD	AMOUNT	BUY	Litecoin	\$118.12	<input type="text" value="10"/>	<button>BUY</button>	Ardor	\$0.11	<input type="text" value="0"/>	<button>BUY</button>						
CURRENCY	PRICE USD	AMOUNT	BUY																	
Litecoin	\$118.12	<input type="text" value="10"/>	<button>BUY</button>																	
Ardor	\$0.11	<input type="text" value="0"/>	<button>BUY</button>																	
		<p><b>Price data from Poloniex API is rendered as a chart</b></p> <p>The chart displays the price of Litecoin over a 14-day period. The y-axis represents the price in US dollars, ranging from 115 to 135. The x-axis represents the number of days relative to the current price, with values from -7 to 0. A blue line connects circular data points. A tooltip is shown for the data point at day -7, which has a value of \$120.14.</p> <table border="1"> <caption>Estimated Data Points from Chart</caption> <thead> <tr> <th>Day</th> <th>Price (\$)</th> </tr> </thead> <tbody> <tr><td>-7</td><td>120.14</td></tr> <tr><td>-6</td><td>120.14</td></tr> <tr><td>-5</td><td>134.00</td></tr> <tr><td>-4</td><td>122.00</td></tr> <tr><td>-3</td><td>122.00</td></tr> <tr><td>-2</td><td>119.00</td></tr> <tr><td>-1</td><td>122.00</td></tr> <tr><td>0</td><td>120.14</td></tr> </tbody> </table> <p><b>Litecoin (LTC)</b></p> <p>Price USD: \$118.15</p> <p>You are currently holding <b>10</b></p> <p><a href="#">Remove</a></p>	Day	Price (\$)	-7	120.14	-6	120.14	-5	134.00	-4	122.00	-3	122.00	-2	119.00	-1	122.00	0	120.14
Day	Price (\$)																			
-7	120.14																			
-6	120.14																			
-5	134.00																			
-4	122.00																			
-3	122.00																			
-2	119.00																			
-1	122.00																			
0	120.14																			

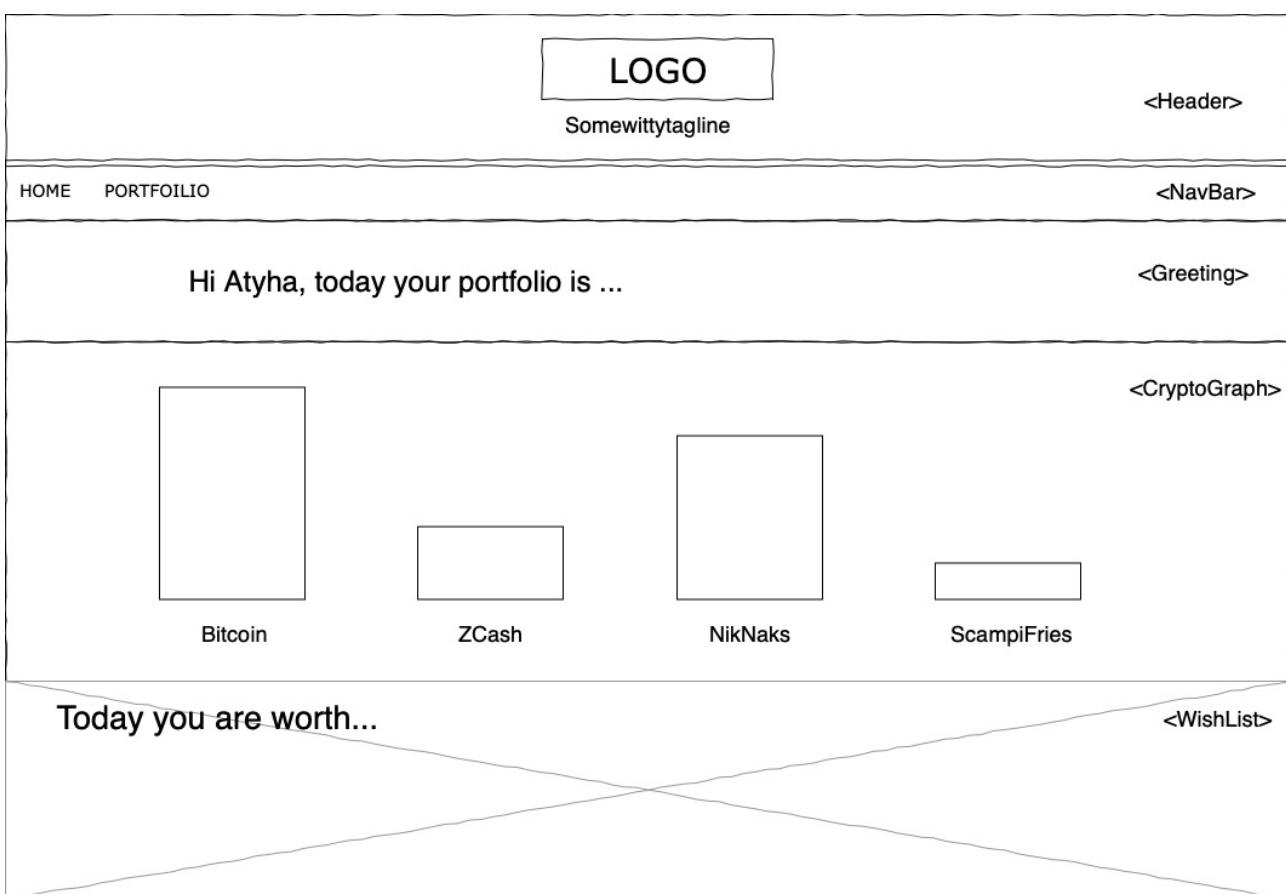
Unit	Ref	Evidence
P	P.1	<p>Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.</p> <p><b>Description:</b> <a href="#"><b>Contributors page from crypto currency tracker group project GitHub repo</b></a></p> <p>The screenshot shows the GitHub Pulse Contributors page for a specific repository. The main chart displays commit activity from May 5, 2019, to May 23, 2019. Three contributors are highlighted: HughJarvis (#1), dshankland (#2), and atyhaarshad (#3). Each contributor has a detailed commit history chart below their name.</p> <ul style="list-style-type: none"> <li><b>HughJarvis (#1):</b> 29 commits, 7,595 ++, 8,018 --.</li> <li><b>dshankland (#2):</b> 26 commits, 29,080 ++, 10,710 --.</li> <li><b>atyhaarshad (#3):</b> 17 commits, 522 ++, 210 --.</li> </ul>

Unit	Ref	Evidence
P	P.12	<p>Take screenshots or photos of your planning and the different stages of development to show changes.</p> <p><b>Description: Notes showing rough wireframes and Vue components.</b></p>

## Wireframe for portfolio performance view

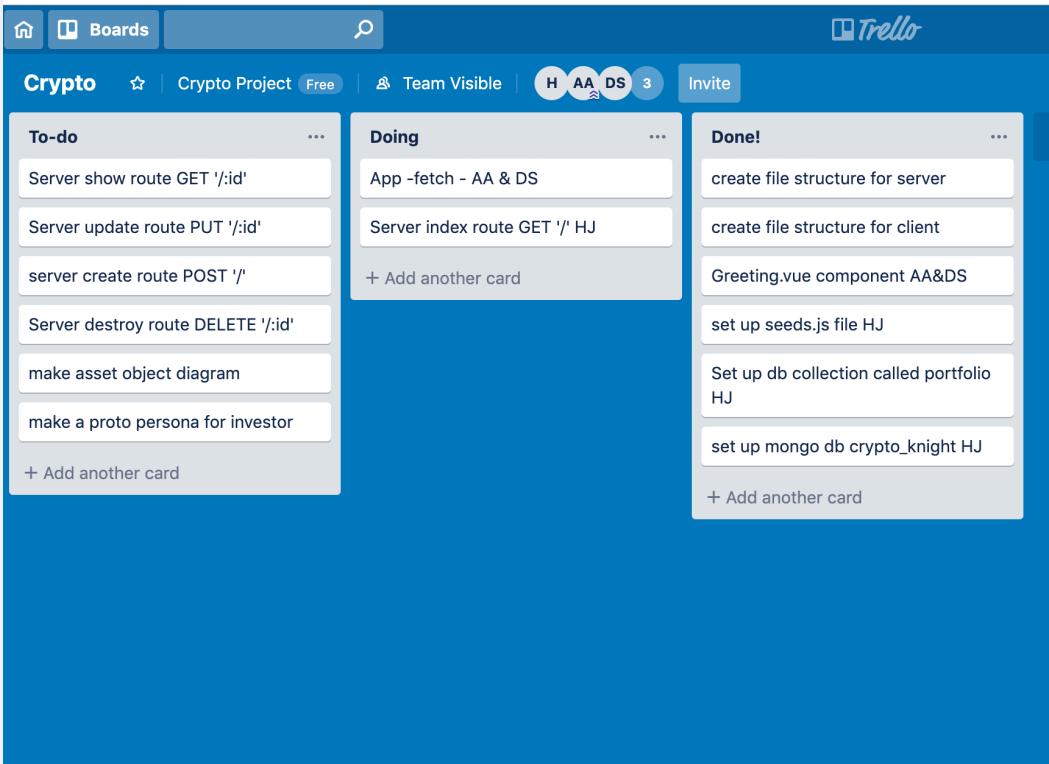


## Wireframe for main portfolio view





Unit	Ref	Evidence
P	P.2	<p>Take a screenshot of the project brief from your group project.</p> <p><b>Description: Project brief from group project</b></p> <div style="border: 1px solid #ccc; padding: 10px;"> <h2>Shares Portfolio Application</h2> <p>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.</p> <h3>MVP</h3> <p>A user should be able to:</p> <ul style="list-style-type: none"> <li>• view total current value.</li> <li>• view individual and total performance trends.</li> <li>• retrieve a list of share prices from an external API and allow the user to add shares to her portfolio.</li> <li>• View a chart of the current values in her portfolio.</li> </ul> <h3>Example Extensions</h3> <ul style="list-style-type: none"> <li>• Speculation based on trends and further financial modelling using projections.</li> </ul> <h3>API, Libraries, Resources</h3> <ul style="list-style-type: none"> <li>• <a href="https://www.alphavantage.co/">https://www.alphavantage.co/</a> (Requires sign up)</li> <li>• <a href="https://www.highcharts.com/">https://www.highcharts.com/</a> HighCharts is an open-source library for rendering responsive charts.</li> </ul> </div>

Unit	Ref	Evidence
P	P.3	<p>Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.</p> <p><b>Description: Trello board from group project</b></p>  <pre> Crypto Project Free   Team Visible   H AA DS 3   Invite  To-do ... Server show route GET '/:id' Server update route PUT '/:id' server create route POST '/' Server destroy route DELETE '/:id' make asset object diagram make a proto persona for investor + Add another card  Doing ... App -fetch - AA &amp; DS Server index route GET '/' HJ + Add another card  Done! ... create file structure for server create file structure for client Greeting.vue component AA&amp;DS set up seeds.js file HJ Set up db collection called portfolio HJ set up mongo db crypto_knight HJ + Add another card </pre>

<b>Unit</b>	<b>Ref</b>	<b>Evidence</b>
P	P.4	Write an acceptance criteria and test plan.
		Acceptance criteria for Crypto Knight portfolio tracker

Acceptance criteria	Test plan	Pass/fail
User can add crypto currency to their portfolio	But function will add currency holding to user's portfolio	Pass
Portfolio updates when user adds new currency	App fetches current price info from APIs	Pass
User can sell currency from their portfolio	Sell function deducts currency from portfolio	Pass
User can check their buying power	Buying power tracker shows how many Ferraris portfolio is worth	Pass

Unit	Ref	Evidence
P	P.7	<p>Produce two system interaction diagrams (sequence and/or collaboration diagrams).</p> <p><b>Description: Sequence diagram for Crypto Knight portfolio tracker</b></p> <pre> sequenceDiagram     participant User     participant CK as Crypto Knight portfolio tracker     participant DB as database     User-&gt;&gt;CK: getPortfolioData()     CK-&gt;&gt;DB: get asset holdings     activate CK     DB--&gt;&gt;CK: asset holdings     CK-&gt;&gt;CK: calculate asset values()     deactivate CK     CK--&gt;&gt;User: Portfolio Data   </pre> <p>This sequence diagram illustrates the interaction between a User, a Crypto Knight portfolio tracker, and a database. The User initiates a request for portfolio data. The portfolio tracker sends a message to the database to retrieve asset holdings. The database returns the asset holdings to the portfolio tracker, which then calculates asset values and returns the portfolio data to the User.</p>
		<p><b>Sequence diagram for Big App Yourself compliment generator</b></p> <pre> sequenceDiagram     participant User     participant CG as Big App Yourself compliment generator     participant DB as database     User-&gt;&gt;CG: save user preferences()     CG-&gt;&gt;DB: post user preferences     activate CG     DB--&gt;&gt;CG: user preferences     CG-&gt;&gt;CG: generate compliment based on preferences     deactivate CG     CG--&gt;&gt;User: compliment user   </pre> <p>This sequence diagram illustrates the interaction between a User, a Big App Yourself compliment generator, and a database. The User saves their user preferences. The compliment generator posts the user preferences to the database. The database returns the user preferences to the compliment generator, which then generates a compliment based on the preferences and returns it to the User.</p>

Unit	Ref	Evidence
P	P.8	<p>Produce two object diagrams.</p> <p><b>Description: Drawer and sweets objects from vending machine</b></p> <pre> classDiagram     class A1 {         &lt;&lt;Drawer&gt;&gt;         DrawerCode: A1         Price: 60         Product: Dairy Milk     }     class DairyMilk {         &lt;&lt;Sweet&gt;&gt;         name: "Dairy Milk"         brand: "Cadbury's"     }     A1 "2" -- "1" DairyMilk   </pre>
		<p><b>Drawer and crisps objects from vending machine</b></p> <pre> classDiagram     class A2 {         &lt;&lt;Drawer&gt;&gt;         DrawerCode: A2         Price: 70         Product: Frazzles     }     class Frazzles {         &lt;&lt;Crisps&gt;&gt;         name: "Frazzles"         brand: "Walkers"     }     A2 "2" -- "1" Frazzles   </pre>

Unit	Ref	Evidence	
P	P.17	Produce a bug tracking report	
		<b>Description: Bug tracking report for Crypto currency project</b>	
Bug/error	Solution	Date	
Bitcoin cash trading symbol doesn't match symbol, so can't add as asset. 10/05/19	Added Map functionality to fetch to allow mapping to portfolio asset.	10/05/19	
router.put doesn't update amount in portfolioGrid (increaseAsset)	Added eventBus handler to increaseAsset to refresh the page.	11/05/19	
Charts not rendering in time.	Tried vue mounted but this didn't work. Used vue watcher, and refactoring of forEach method.	13/05/19	
Unable to match icon data using the currency symbol.	Changed this to trading symbol which meant it could match the icons id.	13/05/19	
calculateHistoricalData method in FetchMethods.js not returning accurate data.	Changed the method to loop around the values returned by the polonium API first, then to loop around the assets in the asset list as this works better.	14/05/19	

Unit	Ref	Evidence
I&T	I.T.7	<p>The use of Polymorphism in a program and what it is doing.</p> <p><b>Description:</b> Abstract class Baddie imposes an attack(player) method on its child classes</p> <pre>public abstract class Baddie {     private String name;     private String weapon;     private int speed;     private int brutality;     private int weaponEffectiveness;      public Baddie(String name, String weapon, int speed, int brutality, int weaponEffectiveness) {         this.name = name;         this.weapon = weapon;         this.speed = speed;         this.brutality = brutality;         this.weaponEffectiveness = weaponEffectiveness;     }      public String getWeapon() { return this.weapon; }     public String getName() { return name; }     public int getSpeed() { return speed; }     public int getWeaponEffectiveness() { return this.weaponEffectiveness; }     public int getBrutality() {         return this.brutality;     }      public abstract void attack(Player player); }</pre> <p>Child class RazorGang has attack(player) method inherited from Baddie class. Also has maim(player), bump(player) and stealBottles(player) imposed by interfaces IMaim, IStealBottles and IBump.</p> <pre>public class RazorGang extends Baddie implements IMaim, IStealBottles, IBump {     public RazorGang(String name, String weapon, int speed, int brutality, int weaponEffectiveness) {         super(name, weapon, speed, brutality, weaponEffectiveness);     }      public void maim(Player player) {         int damage = this.getBrutality() * this.getWeaponEffectiveness() * 10;         player.decreaseHealth(damage);     }      public void bump(Player player) { player.loseGirdLife(); }      public void stealBottles(Player player) { player.loseAllBottles(); }      public void attack(Player player){         this.bump(player);         this.maim(player);         this.stealBottles(player);     } }</pre> <p>Child class Dentist has attack(player) method inherited from Baddie class</p> <pre>public class Dentist extends Baddie {     public Dentist(String name, String weapon, int speed, int brutality, int weaponEffectiveness) {         super(name, weapon, speed, brutality, weaponEffectiveness);     }      public void attack(Player player){         int damage = this.getBrutality() * this.getWeaponEffectiveness() * 10;         player.decreaseHealth(damage);     } }</pre>



Unit	Ref	Evidence
A&D	A.D.5	<p>An Inheritance Diagram</p> <p>Description: FiftyPence, TwentyPence and TenPence classes inherit from Coin class</p> <pre> graph TD     Coin["Coin"] --&gt; Cointype["Cointype : cointype"]     Cointype --&gt; FiftyPence["FiftyPence Cointype.FIFTY"]     Cointype --&gt; TwentyPence["TwentyPence Cointype.TWENTY"]     Cointype --&gt; TenPence["TenPence Cointype.TEN"]   </pre>

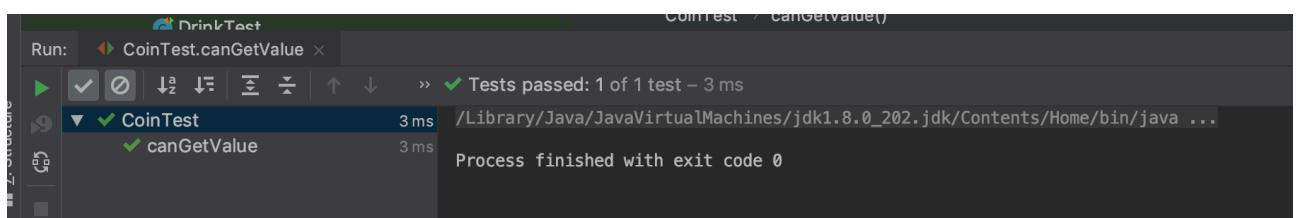
Unit	Ref	Evidence
I&T	I.T.1	<p>The use of Encapsulation in a program and what it is doing.</p> <p>Description: Game class in black jack app has private properties deck, players and dealer, which means they can only be accessed with getters and setters.</p> <pre>public class Game {      private Deck deck;     private ArrayList&lt;Player&gt; players;     private Dealer dealer;      public Game(Dealer dealer, Deck deck){         this.players = new ArrayList&lt;Player&gt;();         this.dealer = dealer;         this.deck = deck;     } }</pre>

Unit	Ref	Evidence
I&T	I.T.2	<p>Take a screenshot of the use of Inheritance in a program. Take screenshots of:</p> <ul style="list-style-type: none"> <li>*A Class</li> <li>*A Class that inherits from the previous class</li> <li>*An Object in the inherited class</li> <li>*A Method that uses the information inherited from another class.</li> </ul>
		<p><b>Description: Coin class</b></p> <pre>package Money; public abstract class Coin {     private CoinType coinType;     public Coin(CoinType coinType) { this.coinType = coinType; }     public CoinType getCoinType() { return coinType; }     public int getCoinValue() { return this.coinType.getValue(); } }</pre> <p><b>FiftyPence class inherits from Coin class</b></p> <pre>package Money; public class FiftyPence extends Coin {     public FiftyPence() { super(CoinType.FIFTY); } }</pre> <p><b>Coin50 is an object of the FiftyPence class</b></p> <pre>coin1 = new Penny(); coin2 = new TwoPence(); coin5 = new FivePence(); coin10 = new TenPence(); coin20 = new TwentyPence(); coin50 = new FiftyPence(); coin100 = new OnePound(); coin200 = new TwoPound();</pre>

## Tests getCoinValue() method inherited from Coin class works for FiftyPence coin

```
@Before  
public void setUp() {  
    coin = new FiftyPence();  
}  
  
@Test  
public void canGetValue() {  
    assertEquals(50, coin.getCoinValue());  
}
```

## Test passes



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.
		<p>Description: Algorithm in method for vending machine to return the correct change. % 100 gives the remainder after dividing the change amount by 100 to determine the number of pound coins in the change. I chose it because it is simple but effective.</p> <pre>int changeLeft = change % 100;</pre> <p>Algorithm for calculating the damage done when a baddy with a given brutality attacks with a weapon of given effectiveness. I chose it because of its elegant efficiency.</p> <pre>int damage = this.getBrutality() * this.getWeaponEffectiveness() * 10;</pre>