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 | |

|  |  |
| --- | --- |
|  | This image above shows an array of hashes stored in a program. Within @pet\_shop there exists the key of pets (marked by the presence of a colon following the word pets). Within this key is an array [ ], and within the array is stored a hash with four key/value pairs. The keys are; name, pet\_type, breed, and price. |
|  | This image above shows a hash being added to existing data using a function.  Lines 128 - 130 show the creation of a function called add\_pet\_to\_stock. Within this function, the .push method is used to push the array the user creates into a preexisting array (pet\_shop).  On lines 132 - 137 the function is called. The new data being stored is a hash and is formatted clearly to show keys (ending with a colon), and values.  Line 139 puts (prints) the full list of pets in the pet shop to the console. |
| A close up of text on a black background  Description automatically generated | This final image above shows the output prior to calling the function (top) and then following the function (bottom). We can see that Barry the snake has been added on the final line. |

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 | |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | The above image shows an empty array called room. Within this room there are three strings - square room, circular room, and rectangular room. The block below the array cycles through each of the strings, nesting each of the arrays into the string beginning “I am in the”. |
| A screenshot of a cell phone  Description automatically generated | The end result appears as below: |

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 | |

|  |  |
| --- | --- |
|  | The image on the left shows my film class template, within which there is a method called find\_specific\_screening. When this method is called the user must put the time they wish to search for as an argument, to be passed through the method as a parameter. |
|  | The image on the right illustrates that there are three screenings for film3. To identify a specific screening, the find\_specific\_screening method can be used. |
|  | The image below shows the product of running that method. We can see that by passing 15:30 as an argument, the method has identified the screening. This has been achieved by running the argument through the SQL statement created within the ruby function above. |

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

|  |  |
| --- | --- |
| A screenshot of text  Description automatically generated | The image above shows my film class template, within this class is a method called customers. This method searches through tables in an SQL database and returns an array containing all of the customers who have purchased a ticket to see the movie the method is called upon. |
| A screenshot of a cell phone  Description automatically generated | By using pry and calling the method on a film created in my code, we can see that the method has successfully linked up with the customer data and has pulled it through. The method achieves this by using sql code within the ruby method ‘customers’, and negotiates through to the customers table through a series of inner joins. The final product is then mapped and placed in an array. |

Week 4

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **A&D** | A.D.1 | A Use Case Diagram | |

|  |  |
| --- | --- |
| A close up of text on a white background  Description automatically generated | The image on the left is a Use Case Diagram showing the relationship between the user and their physical inventory, managed by my Inventory Management System. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **A&D** | A.D.2 | A Class Diagram | |

|  |  |
| --- | --- |
| **A screenshot of a cell phone  Description automatically generated** | The image on the left is a Class Diagram showing the relationship between the classes used in my inventory management project.  The diagram shows a many to many relationship connected via a junction class. The Inventory class has a one to one relationship with the Book class. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **A&D** | A.D.3 | An Object Diagram | |

|  |  |
| --- | --- |
| A picture containing screenshot  Description automatically generated | The image on the left is an Object Diagram showing the relationship between the classes used in my inventory management project complete with example information. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **A&D** | A.D.4 | An Activity Diagram | |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | The image on the left is an Activity Diagram showing the relationship between the user and their inventory when new stock has arrived. |

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

|  |  |  |
| --- | --- | --- |
| Constraint Category | Implementation Constraint | Solution |
| Hardware & Software Platforms | Constrained by the preferred platform/s of the user. | Ensure that there has been a clear discussion with the client as to their preferred platform. |
| Hardware & Software Platforms | Installation also involves knowledge of Sinatra – currently the app is hosted on the users machine, but setting this application up on a server would create complications. | Not known at present. |
| Performance Requirements | Very low performance requirements. Should run on modern and dated machines. Requires a working browser. | Most machines have fully functioning web browsers, so the constraint should not be an issue. |
| Persistent Storage and Transactions | Data is to be stored in an SQL database. Application has no in-built backup software. Regular backups have to be conducted by the user. | None, other than clarity to the user that the database must be updated. |
| Usability | Currently there is little to no documentation to help the reader through use of the program. | Create a more detailed readme and training documentation. |
| Usability | In order to add items to the inventory, the user must first make an author, then a book, then an inventory item. | Create a ‘create’ wizard which guides the user through all three stages. |
| Budgets | Cost of labour spent on the project. Otherwise no other budget constraints. | None. |
| Time Limitations | One week to develop the application. | Focus on functionality and ensure the program meets all the user’s needs. Focus on usability given the user’s inability to read small items on the screen. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **P** | P.5 | User Site Map | |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | The image on the left is a Site Map of my Inventory Management project. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **P** | P.6 | 2 Wireframe Diagrams | |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | Here are three wireframe diagrams showing the main page (top left).  The item description page (top right).  The new author page (bottom centre). |

Week 5

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

|  |  |
| --- | --- |
| **A screenshot of a cell phone  Description automatically generated** | The image on the left shows my pseudocode above the completed Ruby code. |

|  |  |  |  |
| --- | --- | --- | --- |
| **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 | |

|  |  |
| --- | --- |
| **A screenshot of a cell phone  Description automatically generated** | When the user places an author in the system, the system asks for the authors forename and surname. These values are stored separately. |
| **A screenshot of a cell phone  Description automatically generated** | But on the main inventory page, the authors name appears in full, not as separate forename/surname entries. |
|  | The creation of the authors full name is managed by a method in my author.rb file, which takes the forename and surname of the author (entered by the user) and combines the two, so as to create a presentable string. |
| **A screenshot of a cell phone  Description automatically generated** | This method is then accessible in a method in inventory.rb called present\_author\_or\_authors. Which relies on the output from author\_or\_authors (both featured in the image). |
| **A screenshot of a cell phone  Description automatically generated** | Lastly, the method present\_author\_or\_authors can be found within index.erb which presents the main inventory page. The method presents the authors in a strong containing both first and last name. |

|  |  |  |  |
| --- | --- | --- | --- |
| **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 | |

|  |  |
| --- | --- |
| **A screenshot of a cell phone  Description automatically generated** | The image on the far left shows my inventory management program and CLI is positioned directly right of this.  Currently, there are five authors listed in the Author database, which can be seen both in the application as well as within psql. |
| **A screenshot of a cell phone  Description automatically generated** | In the image on the left I am inputting data into the New Author field. Upon hitting submit, the data will be sent to the database and the records will be updated to show John Barth as a new author. |
| **A screenshot of a cell phone  Description automatically generated** | On the left we can see that John Barth is now stored in the sql file. After hitting submit the page has taken us back to the homepage, which does not show authors. If we were to click on view authors on the sidebar, then John Barth would appear at the bottom of the list. |
| **A screenshot of a cell phone  Description automatically generated** | Here we can see that John Barth has indeed been added. |

|  |  |  |  |
| --- | --- | --- | --- |
| **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 | |

|  |  |
| --- | --- |
| **A screenshot of a cell phone  Description automatically generated** | The image on the far left shows my inventory management program and CLI is positioned directly right of this.  I have just started Sinatra which is running in a file named main.rb. |
| **A screenshot of a cell phone  Description automatically generated** | I have clicked on the button ‘more info’ next to Scarlett Thomas, PopCo.  Within Sinatra we can see that there has been a GET request for inventory/1 where 1 is the id page which specifically shows the details page for PopCo by Scarlett Thomas.  The request has been successful, and the details page has loaded. |
| **A screenshot of a cell phone  Description automatically generated** | I have now clicked on the ‘Edit this Entry’ button. In the CLI we can see that there has been a GET request for inventory/1/edit.  The request has been successful, and the details page has loaded. |
| **A screenshot of a cell phone  Description automatically generated** | Finally, I changed some of the values and submitted the form by clicking ‘submit entry’ in the previous page.  In the CLI we can see that a POST request has been sent to inventory/submitted\_edits. I have then been redirected back to inventory/1/edit where we can see that the changes have been saved. |

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

|  |  |
| --- | --- |
| **A screenshot of a cell phone  Description automatically generated** | Image showing the homepage of my project. |
| **A screenshot of a cell phone  Description automatically generated** | Image showing my github page for the project. [Accessible here.](https://github.com/wickertongue/e33_wk4_project__bookshop) |

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

|  |  |
| --- | --- |
| **A close up of text on a white surface  Description automatically generated** | The image on the left shows my original class structure. Author and Book are in a many to many relationship, and Inventory was representative of the junction table between them. |
| **A screenshot of a cell phone  Description automatically generated** | This image show my proto-persona, for whom the project was developed.  A key aspect of this was to make the finished application user friendly, simple to use, and use large features. |
| **A close up of text on a white background  Description automatically generated** | My original wireframes show how I initially intended to create an application that only used two pages – a main page, and a ‘more information’ page.  The intention was that new authors, books and inventory items could be created from these two pages. At this time I had not considered that entry and edit forms may need their own pages. |
| **A screenshot of a cell phone  Description automatically generated** | After creating the classes, database, and connecting the two with save methods employing SQL, I realised that by putting items such as quantity into Inventory I was inadvertently creating duplicate records.  If a book had more than one author, then a second inventory item would be created to connect the second author with the first book, this produced two records in the inventory which was not what I had intended.  I created a new Class Diagram, which can be seen on the left. |

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 | |

|  |  |
| --- | --- |
| **A screen shot of a computer  Description automatically generated** | The image on the left shows the pokeapi loaded into my program in app.vue. The information from the api is then stored under data, specifically as an array in allPokemon. |
|  | The data stored in allPokemon is then passed through to the pokemon-filter and pokemon-detail files. |
|  | In PokemonFilter.vue I have created a search input in HTML which, when data is entered, runs a method which then returns the data to the user. |
| **A screen shot of a social media post  Description automatically generated** | This image shows the search method invoked by entering data into the HTML search bar. An eventBus sends the captured information back to App.vue. |
| **A screenshot of a cell phone  Description automatically generated** | Back in App.vue, an eventBus is coded to listen for ‘selected-pokemon’. It captures the data sent from PokemonFilter and fetches additional data from an API. This data is then stored in selectedPokemonDetails |
| **A screenshot of a cell phone  Description automatically generated** | selectedPokemonDetails is then passed through the HTML as selected-pokemon-details into PokemonDetail. |
| **A screenshot of a cell phone  Description automatically generated** | In PokemonDetail.vue, selectedPokemonDetails is stored as a prop. This data is then used to display the information in the browser through HTML. |
| **A screenshot of a cell phone  Description automatically generated** | In PokemonDetail.vue the HTML is coded to display information about the specific pokemon in a table. |
| **A screenshot of a cell phone  Description automatically generated** | The program as it appears initially in the browser. |
| **A screenshot of a cell phone  Description automatically generated** | The program once a search for ‘char’ has been performed. We can see that the information below the search container has changed to present information about the pokemon the user has searched for. |

Week 8

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

|  |  |
| --- | --- |
| **A screenshot of a social media post  Description automatically generated** | The brief can be found on the left. |

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

|  |  |
| --- | --- |
| **A screenshot of a computer  Description automatically generated** | On the left is an image of the Trello board we used to organise our project, using basic Kanban methods. |
| **A sign above a door  Description automatically generated** | We also used whiteboards extensively, to help construct pages and identify elements we would need to draw from the API. |

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

|  |  |  |
| --- | --- | --- |
| Acceptance Criteria | Expected Result | Pass/Fail |
| A user is able to | The… does… when… | Pass/fail |
| A user is able to calculate their daily recommended intake of calories. | On the calculate calories page, the daily recommended allowance is returned – however, it currently only provides data for men. | Pass |
| A user can return nutritional information about a searched item. | When an item is entered into the search bar, and a result is selected, nutritional information is displayed. | Pass |
| A user can get a set of recipes that suits their lifestyle. | Currently not featured. | Fail |

Week 9

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

|  |  |
| --- | --- |
| **A screenshot of a social media post  Description automatically generated** | The screenshot on the left shows contributions to our group github project. |

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 | |

|  |  |
| --- | --- |
|  | On the left is an image showing a test I had written which tests whether my program can set a player as a dealer. The checkIfDealer function is red because it has not been created yet. |
|  | I then created the setDealer function, and a function for checkIfDealer (not pictured). |
| A screenshot of a cell phone  Description automatically generated | The test fails when run, as I had not actually included the setDealer function in the test narrative. |
|  | I refactored the test so that the setDealer function was included in the narrative. |
| A picture containing screenshot  Description automatically generated | I reran the test, and the test passed. |

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

|  |  |
| --- | --- |
| A close up of a logo  Description automatically generated | In the image on the left, a new Player class is created. The properties are set as private. |
| A close up of a screen  Description automatically generated | In the constructor for this class, the name and healthpoints must be set when a new Player is created, defense is set to a default of 0. |
| A screenshot of a cell phone  Description automatically generated | Three get methods are then created so as to gain access to the properties beyond the scope of the class. Crucially, these three functions can be invoked only by a player (or on any classes that inherit from player).  Specifying the properties as private helps to ensure single responsibility. |

Week 12

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

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | The Guitar Class pictured on the left implements the interface ISell. |
| A screen shot of a social media post  Description automatically generated | In the Shop Class I have created an ArrayList which can contain ISell objects – this means that any Objects created using a Class that extends the ISell interface can be placed inside this ArrayList.  This is beneficial as it allows for the storage of various class types within one ArrayList. Without polymorphism, we would have to createa unique ArrayList for each class (ArrayList<Book>, ArrayList<Guitar>, etc) |
|  | Items are added to the stock ArrayList via the addToStock method, which is located in the Shop Class. |
| A screenshot of a cell phone  Description automatically generated | The image on the left shows the ShopTest Class, including items I have created – Guitars, Books, SetOfPicks. These items have all been created as instances of their class. Book as an instance of the Book Class, Guitar as Guitar Class, and so on. |
|  | In this image, I pass in a book and a guitar using the addToStock method, this will add the book and the guitar to the ISell ArrayList. These items are polymorphic – a guitar can be a guitar, and, because it implements ISell, the guitar can also act as an instance of ISell. |
| A screenshot of a cell phone  Description automatically generated | The test passes, showing that instances of the Book and Guitar classes are polymorphic. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **A&D** | A.D.5 | An Inheritance Diagram | |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | The image on the left shows Guitar, Saxophone and Piano inheriting from Instrument. The three instruments will inherit the properties of Instrument, along with the getters and setters created in the Instrument Class. |

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

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | On the left is an image showing the Instrument Class. The Instrument Class is an abstract class - the reason being that this class should never be instantiated, and that only children of this class should be used to create objects. |
| A screenshot of a cell phone  Description automatically generated | The Guitar Class is a child of the Instrument Class, and consequently inherits the properties and methods of its parent. In the image we can see that when we create a new guitar, we must specify it’s material, colour and type as we would the Instrument Class.  In the constructor, the keyword super indicates the properties that are inherited from the parent.  Note also that I have not had to define the methods previously defined in the parent – these too are inherited, but do not need to be stated (unless the intention is to override them). |
| A screenshot of a cell phone  Description automatically generated | In the GuitarTest Class I have created a guitar object.  In the image we can see three tests which check that we can access and use the methods typed within the Instrument Class, but that are absent from the Guitar Class.  All of these tests have passed (note the green ticks). This means we can use the methods inherited from the Instrument Class on objects created from the Guitar Class. |

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. | |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | This algorithm loops through each player in the players arraylist then, for each player, their hand is counted. If their hand total is larger than the value stored in highest and if the players hand value is less than 22, then the player is assigned the winner and the total of their hand is saved in ‘highest’.  This allows for the system to work out who is the winning player in a game of blackjack. |
|  | In shuffleCards I have imported the java utility ‘collections’ which includes the function ‘shuffle’. Shuffle takes all of the items in ‘cards’ and swaps them with another randomly selected item to create a randomised arraylist. |

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

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **P** | P.8 | Produce two object diagrams. | |

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit** | **Ref** | **Evidence** |  |
| **P** | P.17 | Produce a bug tracking report | |

|  |  |  |
| --- | --- | --- |
| Bug/Error | Solution | Date |
| Tables do not turn red when they are booked. | The tables currently only turn red when clicked, this function should be mapped to read whether the table has data entered into it, and then turn red if true. | Found: 07/11/19  Fixed: TBD |
| Search by date does not function. | To store the output from the user clicking on a date in the calendar within state, then pass this through a fetch to a custom query. The data returned should then be returned to the applicable table. | Found: 07/11/19  Fixed: 07/11/19 |
| Cannot gain access to list of tables within ‘dinnerTables’ endpoint, specifically within bookings, tables. | To create a projection that allows for the tables to be embedded within that endpoint. | Found: 05/11/19  Fixed: 07/11/19 |
| Table data does not render to show the bookings according to the date selected on the header. | The date is stored in state, the issue appears to be that the site is not re-rendering according to the new information. The fix is likely to do with setState not being called correctly | Found: 07/11/19  Fixed: 07/11/19 |
| Zooming in on the homepage breaks the sidebar – the form fields extend beyond the sidebar. | Could be a div or flexbox issue. To fix, recalibrate the divs or alter the flexbox. | Found: 08/11/19  Fixed: TBD |