ENTWA - “Stumps” Cricket Bookshop E-Commerce Project – Adam Thornton UP2018447

1. Introduction

The overall aim of this coursework is to develop an E-Commerce system for the cricket bookshop “Stumps” that can be easily adapted for use by other E-Commerce sites, this will be done by meeting the following objectives:

- Specify the requirements of the system
- Create a user interaction design
- Learn the difference between types of controllers
- Learn how to use Java Server Faces (JSF) and a ‘SessionScoped’ controller to build and manage views
- Learn how to use Enterprise Java Beans (EJB) and the Java Persistence API (JPA) to add information to a database and retrieve information from a database
- Learn how to populate data tables with lists in the controller
- Implement the artifact, documenting implementation decisions as they are made, specifically around the difficult development areas
- Test and debug the artifact, functional tests are carried out on iterations after they have been implemented

2. Design

The single server implementation of JSF was chosen as the implementation architecture because it would allow the easy storage of information in a database, a single server was used because only one database is required for the current system, future work would include using another server to populate the books in the database. According to (deBara & deBara, 2020) keeping web design simple is crucial for designing effective web applications.

2.1 Register Page

In the ‘register’ page, there is an input form to help create a new account, this form collects information on the account being registered using the following fields:

- Role – select one menu
- Given name – input text
- Family name – input text
- Email address – input text
- Password – input secret

The password field was designed as an input secret field instead of an input text field because input text fields show text that has been entered whereas input secret fields do not, this is important because passwords need to be secure. This decision was also made for the password field in the ‘login’ page. A select one menu was used for the ‘role’ field because the account can only be a customer or an administrator, and a select one button field was not used because that would waste space in the view.

There are also 5 buttons:

- Check Role – checks which role is selected and renders the register button for that role
- Register Customer – registers a customer using the information in the form
- Register Administrator – registers an administrator using the information in the form
- Return to home page – navigates to the ‘home’ page
- Log in to account – navigates to the ‘login’ page

However, only the ‘Check Role’, ‘Return to home page’, and ‘Log in to account’ buttons are visible at first. Once the ‘Check Role’ button is clicked the appropriate register button will appear for the role selected, clicking the register button will register the account in the database.

The ‘Check Role’ button was used to verify whether the account being created is a customer or an administrator, because the customer and administrator account details need to be kept in separate tables.

2.2 Login Page

In the ‘Login’ page, there is an input form with two input fields:

- Email Address – input text
- Password – input secret

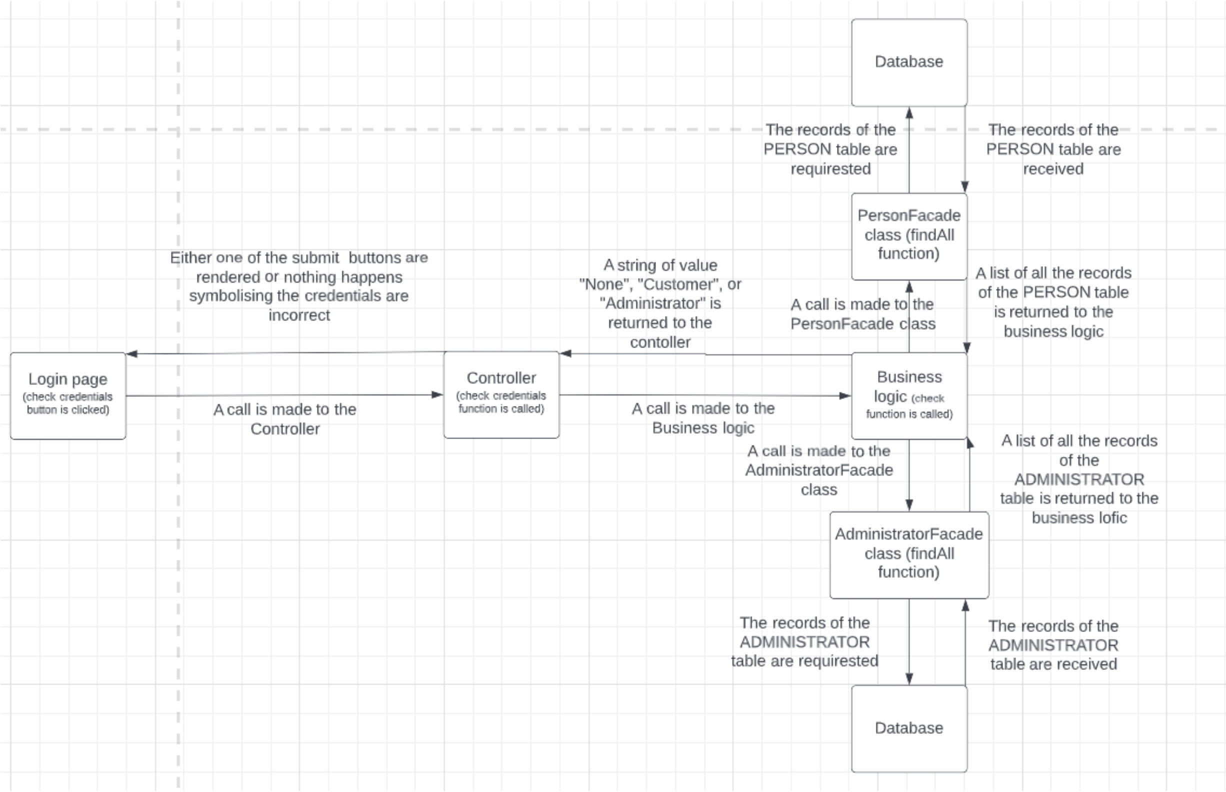
The justification for using an input secret field has already been explained in section 2.1.

The input form also has 5 buttons:

- Return – navigates to the home page
- Check Credentials
- Login as Customer – navigates to the ‘customer’ home page
- Login as admin – navigates to the ‘admin’ home page
- Can’t remember login details – navigates to the ‘forgotLogginDetails’ page

A ‘Check Credentials’ button was created because the system needs to verify that the login details entered are correct, and whether they are that of a customer or an administrator, so the correct login button can be rendered.

The flow of information between components of the artefact during execution of the ‘Check Credentials’ button is demonstrated in the diagram below:


Database
The records of the PERSON table are requirested
The records of the PERSON table are received
PersonFacade class (findAll function)
Either one of the submit buttons are rendered or nothing happens symbolising the credentials are incorrect
A string of value "None", "Customer", or "Administrator" is returned to the contoller
A call is made to the PersonFacade class
A list of all the records of the PERSON table is returned to the business logic
Login page (check credentials button is clicked)
A call is made to the Controller
Controller (check credentials function is called)
A call is made to the Business logic
Business logic (check function is called)
A call is made to the AdministratorFacade class
A list of all the records of the ADMINISTRATOR table is returned to the business lofic
AdministratorFacade class (findAll function)
The records of the ADMINISTRATOR table are requirested
The records of the ADMINISTRATOR table are received
Database

2.3 Previous Orders Page

This page has 3 data tables, the first starts rendered and prints the details of every order the account has made, only the orders the account has made appear because only administrators have the authority to view all the orders in the system for the sake of privacy. The 'Search' button beneath it will hide the first data table and render a second data table that prints the orders made by the account, filtered using the title in the input text field next to the button, and the 'Reset' button will return the data table to its original form. Beneath that there is a 'Cancel Order' button which will cancel the order that has been specified in the input text field next to the button provided the order has not been dispatched. Beneath that there is a 'Search' button that uses the order id provided to print a report of the order in a data table below, clicking 'Hide' will hide the data table. Finally, the 'Return' button navigates to the 'customer' page. When the system shows the filtered data table the unfiltered data table is hidden to save space in the view, this was done to avoid the need for scrolling. Scrolling should be avoided because it will force users to remove at least one hand from the keyboard which will irritate them and reduce efficiency (Post, 2022).

2.4 Browse Page

At the top of the page there is a data table that prints the details of every book stored in the database. Beneath it there are two input forms, the first is for search parameters, it holds the following input text fields:

- Title
- Author

The second form is for search filters, it holds the following input text fields:

- Year of publication
- Price range

These forms were designed because they would allow users to search for a specific title, author, publish year, or price book by clicking the 'Browse' button which renders a second data table beneath and a 'Reset' button which hides this data table. At the bottom of this page there are two input text fields that are labelled 'Book ID' and 'Copies' to ensure users know what data should be entered into these fields, this is called prompting data entry (Smith & Mosier 1986). A 'Submit' button that adds the book(s) to the shopping basket and a 'View Shopping Basket' button that navigates to the 'Shopping Basket' page, the book is was used to determine which book was added because the data tables above include an id for each book and it prevents users from having to type out five or six fields. Finally there are two buttons both labelled 'Return to home menu' that navigate to the 'home' and 'customer' page depending on whether the user is logged in, this allows the same page to be used for logged out browsing as well as logged in because when logged out the button will navigate to the 'home' page and the final input form will be hidden.

2.5 Checkout Page

At the top of this page there are two data tables that print the payment methods and delivery addresses available to the current account. Each data table has an input text field to enter the id of the chosen payment method/delivery address. These tables were filtered because it would be a severe violation of GDPR if users could see and use other users home address and payment method details. Beneath is a third data table showing what is currently in the shopping basket, this is present because sometimes customers might want to double check what they are ordering and finally there are four buttons: 'Return to Home screen', 'Return to Basket', 'Add Payment method/delivery address', and 'Place Order' the first three navigate to the 'customer' page, 'Shopping Basket' page, and the 'Add Payment Method/Delivery Address' page, and the last orders everything that's in the basket.

2.6 All Orders Page

At the top of this page is a data table that prints every order in the system, beneath that is an input text field next to a 'Search' button which renders a data table that shows a report of the order entered into the input text field and 'Hide' which hides this data table. Beneath these buttons are two input text fields that hold the id of an order and the status it is being updated too respectively, next to these input text fields is a 'Change Status' button that will update the specified order to its new status. Below is a 'Return' button that will navigate to the 'admin' home page. An order was selected to have its status updated by entering its id in a text field because the first column in the orders data table is the order's Id and text fields are easy for users to understand.

2.7 All Financial Reports Page

Beneath the headings of this page there is a data table that prints every financial transaction the system has recorded, beneath it is an input text field that holds a transaction Id, when the 'View Transaction Button' next to the input text field is clicked a second data table is rendered which prints a report of the specified transaction, beneath it is a 'Return' button which navigates to the 'admin' home page. The report is generated at the bottom of the current view because creating a new view to only hold a report of a single transaction would waste too much heap space.

2.8 Add Books Page

Beneath the headings of this page there is an input form that holds the following seven input text fields:

- Title
- Author
- Publisher
- Edition
- Year
- Price
- Copies

The footer of this form has two buttons, 'Record' and 'Restock', the 'Record' button will add the book information entered above to the 'BOOKS' table in the database, the 'Restock' button will restock the copies of each book to 50. Beneath is a data table that prints every book in the database this was added to the design because it will inform the administrator of not just which books are currently in the database, but how many copies and it will offer confirmation that the books have been successfully added to the database, finally is a 'Return' button which navigates to the 'admin' home page.

3. Implementation and Testing

To implement the artifact several development tools and data structures were used. For example, to filter lists the Iterator tool was used, by iterating through the list and removing elements that don't contain or equal the filter variable, and array lists were used over maps and sets because they were the easiest to initialise, modify and access.

3.1 Difficult areas of development and how these were overcome

An area of difficulty in the development of the system was placing an order, specifically having the number of copies being ordered removed from the database. To implement this functionality initially meant having a significant amount of business logic in the controller, however that later changed to having the 'placeOrder' button call the 'placeOrder' function in the controller.

```
public void placeOrder() {
        o.setAccountName(accountName);
        o.setStatus("Ordered");
        o.setOrderTime(LocalDateTime.now());
        ss.placeOrder(o);
}
```

Note: 'ss' is the object to the 'StartService' class which is the artifact's business logic.

This function attaches the name associated with the count to the order, assigns the 'Ordered' status and takes note of the time the order was made before calling the 'placeOrder' function in the business logic. To solve the established problem of removing the number of copies ordered from the database, the business logic was split into two functions, the function called from the controller focuses on placing the order, the second focuses on removing the books ordered from the shopping basket and adjusting the copies in the database. The former is the 'placeOrder' function called by the controller.

```
public void placeOrder(Orders o){
        List<Basket> order = retrieveBasket();
        int total = 0;
        double totalCost = 0;
        for(int i = 0; i < order.size(); i++){
            total += order.get(i).getCopies();
        }
        List<String> orderTitles = new ArrayList<>();
        List<String> orderAuthors = new ArrayList<>();
        List<String> orderEditions = new ArrayList<>();
        List<String> orderPrices = new ArrayList<>();
        List<String> orderCopies = new ArrayList<>();
        Basket item;
        List<Basket> items = new ArrayList<>();
        List<Integer> integerCopies = new ArrayList<>();
        for (int i = 0; i < order.size(); i++) {
            item = order.get(i);
            items.add(item);
            orderTitles.add(item.getTitle());
            orderAuthors.add(item.getAuthor());
            orderEditions.add(item.getEdition());
            double indivPrice =
Double.parseDouble(item.getPrice());
            orderPrices.add(item.getPrice());
            int copyNo = item.getCopies();
            double typePrice = indivPrice * copyNo;
            totalCost += typePrice;
            integerCopies.add(copyNo);
            String copyWord = Integer.toString(copyNo);
            orderCopies.add(copyWord);
        }
        o.setTitle(orderTitles.toString());
        o.setAuthor(orderAuthors.toString());
        o.setEdition(orderEditions.toString());
        o.setPrice(orderPrices.toString());
        o.setCopies(orderCopies.toString());
        o.setAllCopies(total);
        o.setTotalPrice(totalCost);

        int index1 = 0;
        for(int i = 0; i < retrievePayment().size(); i++){
            if(o.getPaymentID() ==
retrievePayment().get(i).getId()){
                index1 = i;
                break;
            }
        }

        int index2 = 0;
        for(int i = 0; i < retrieveAddress().size(); i++){
            if(o.getDeliveryID() ==
retrieveAddress().get(i).getId()){
                index2 = i;
                break;
            }
        }
        o.setDeliveryAddress(retrieveAddress().get(index2));
        o.setPayment(retrievePayment().get(index1));
        List<Orders> currentOrders = retrieveOrders();
        if(currentOrders.isEmpty()){
            o.setId(1L);
        } else{
            o.setId(currentOrders.get(currentOrders.size()-
1).getId()+1);
```

```
        }
        of.create(o);
        setOrder(o, items, orderTitles, integerCopies);
    }
```

This function populates the Orders entity object ‘o’ using the Basket entity and other sources. The Orders entity’s foreign keys to the PaymentMethod and Address entities were assigned using the ‘retrieveAddress’ and retrievePayment’ functions which retrieve all the records in the Address and PaymentMethod tables respectively.

```
public List<Address> retrieveAddress() {
        List<Address> add = af.findAll();
        return add;
    }

    public List<PaymentMethod> retrievePayment() {
        List<PaymentMethod> pay = pmf.findAll();
        return pay;
    }
```

The ‘placeOrder’ function then assigns the order an id depending on how many orders have been placed in the system before using ‘of’, the object to the OrdersFacade class, and calling the ‘create’ function which creates a new record in the database’s ORDERS table and populates it with the object ‘o’ before calling the ‘setOrder’ function to reduce the number of copies of each book in the database by the number of copies ordered.

```
public void setOrder(Orders o, List<Basket> items, List<String> 
orderTitles, List<Integer> integerCopies) {
        List<Basket> contents = baf.findAll();
        for (int i = 0; i < contents.size(); i++) {
            Basket item = contents.get(i);
            baf.remove(item);
        }
        List<Books> booksToReduce = retrieveBooks();
        for(int i = 0; i < orderTitles.size(); i++){
            String title = orderTitles.get(i);
            int copies = integerCopies.get(i);
            for(int j = 0; j < booksToReduce.size(); j++){

if(title.equals(booksToReduce.get(j).getTitle())){

                    int remainingCopies = 
booksToReduce.get(j).getCopies();
                    remainingCopies = remainingCopies - copies;
                    Books newBook = booksToReduce.get(j);
                    newBook.setCopies(remainingCopies);
                    bf.edit(newBook);
                }
            }
        }
    }
```

Note: ‘baf’ is the object of the ‘BasketFacade’ class, the 'findAll’ function returns a list of all the records in the table pertaining to the façade class in use. ‘bf’ is the object of the ‘BooksFacade’ class, the ‘edit’ function will edit the table by updating a record with the object passed pertaining to the façade class in use.

This function retrieves a list of all the books in the Basket table and assigns it to a list of type Basket, the function then iterates through this list removing each book ordered from the BASKET table. Next the function iterates through a list of all the titles in the order, in each iteration the function iterates through all the books in the database, if a book title from the order matches a title from the database, the copies of that book in the order are subtracted from the copies in the database and the BOOKS table is edited to show the updated number of copies.

3.2 Filter Searches

Another area of development that proved to be difficult was implementing functionality to initialise the ‘listOfBooks’ list which populates the second data table in the ‘browse’ page so that it represent the list of all books in the database filtered with 2 search parameters and 2 filters. Filtering through the list with a single search parameter and no filters was achieved through the ‘Iterator’ tool to iterate through the list dropping each record that does not contain the ‘title’ filter variable.

The problem with using this approach for multiple search parameters and filters is that iterator if statements don’t work properly if there are multiple arguments in the statement. To overcome this the decision was made that the ‘title’ search parameter took precedence over the ‘author’ search parameter. This means that if neither field is empty then the ‘title’ parameter would be used to filter the list. After the list is filtered by the ‘title’/’author’ parameter, the ‘yearofPublication’ parameter is used, if this parameter is not empty then the Iterator tool is used to iterate through the list and remove records that were published in different years, finally the ‘priceRange’ parameter is used, unless the variable is empty, the iterator tool iterates through the list and every book with price above the upper bound of price ranges or below the lower bound is removed from the list as seen below.

```
public List<Books> findBooks(List<Books> listOfBooks, String
author, String title, String yearOfPublication, String
priceRange) {
        listOfBooks = retrieveBooks();
        for (Iterator<Books> iterator = listOfBooks.iterator();
iterator.hasNext();) {
            if (author.isEmpty()) {
                if
(!iterator.next().getTitle().contains(title)) {
```

```
                    iterator.remove();
                }
            } else {
                if
(!iterator.next().getAuthor().contains(author)) {
                    iterator.remove();
                }
            }
        }
        if(!yearOfPublication.isEmpty())
        {
            for (Iterator<Books> iterator =
listOfBooks.iterator(); iterator.hasNext();) {
                if
(!iterator.next().getYearOfPublication().equals(yearOfPublicati
on)) {
                    iterator.remove();
                }
            }
        }
        if(!priceRange.isEmpty()){
            String[] prices = priceRange.split("-");
            double lowBound = Double.parseDouble(prices[0]);
            double highBound = Double.parseDouble(prices[1]);
            for (Iterator<Books> iterator =
listOfBooks.iterator(); iterator.hasNext();) {
                double price =
Double.parseDouble(iterator.next().getPrice());
                if (price < lowBound || price > highBound) {
                    iterator.remove();
                }
            }
        }
        return listOfBooks;
    }
```

3.3 Check Credentials

There were two methods called from the register page and the login page respectively that checked fields in the form they are part of to verify their contents, in the ‘register’ page the ‘Check Role’ button calls the ‘checkRole’ function in the controller to verify which ‘role’ is selected in the register account form and to render the appropriate register button.

```
public void checkRole() {
        longTerm = password;
        if(name.isEmpty() || surname.isEmpty() ||
emailAddress.isEmpty() || password.isEmpty()){
            customerFlag = false;
            administratorFlag = false;
            registerError = true;
        }
        else if(role.equals("Customer")){
            customerFlag = true;
            administratorFlag = false;
            registerError = false;
        }
        else
        {
            customerFlag = false;
            administratorFlag = true;
            registerError = false;
        }
        password = longTerm;
    }
```

This function will first save the password as long term, then verify the fields are not empty before verifying the value of the ‘role’ variable and assign values to Boolean variables that determine whether the button to register customers or the button to register administrators is rendered in the view.

In the ‘login’ page the ‘Check Credentials’ button calls the ‘checkCredentials’ function in the controller.

```
public void checkCredentials() {
    String type = ss.check(loginEmail, loginPassword);
    switch (type) {
      case "Customer":
        render = true;
        adminRender = false;
        loggedOutBrowser = false;
        break;
      case "Admin":
        render = false;
        adminRender = true;
        loggedOutBrowser = false;
        break;
      default:
        render = false;
        adminRender = false;
        break;
    }
  }
```

The ‘adminRender’ variable will render the admin login button and the ‘render’ variable renders the customer login button, ‘type’ is initialised as the return value of the business logic ‘check’ function call.

```
public String check(String loginEmail, String loginPassword){
        String type = "None";
        for(int i = 0; i < retrieveAccounts().size(); i++){

if(retrieveAccounts().get(i).getEmailAddress().equals(loginEmai
l) &&
retrieveAccounts().get(i).getPassword().equals(loginPassword)){
                type = "Customer";
            }
        }
        for(int i = 0; i < retrieveAdmin().size(); i++){

if(retrieveAdmin().get(i).getUsername().equals(loginEmail) &&
retrieveAdmin().get(i).getPassword().equals(loginPassword)){
                type = "Admin";
            }
        }
        return type;
    }
```

This function checks if the credentials match the credentials of any customers in the database and then check against admin credentials in the database and then returns the ‘type’ variable.

3.4 How the components were tested

A type of black box testing, called functionality testing was carried out on the artifact, functionality testing was carried out because the artifact was created using the Agile development methodology, this methodology includes testing iterations after they have been designed and implemented in the Agile workflow which occurrs inside the iteration stage of the Agile life cycle (*What Is Agile Methodology In Project Management*, n.d), in line with this methodology the artifact iterations were designed, implemented, and tested in sprints. Therefore, functionality testing was carried out on the iterations immediately after they were implemented. The iterations were planned to use the specified requirements, specifically using the high priority specified requirements, this means that as a result the first requirements to serve as the focus of iterations were the priority level 1 requirements, after those requirements were incorporates, priority level 2 requirements were carried out, followed by priority level 3 requirements. Lower priority requirements were carried out later because they weren’t as necessary to the success of the artifact. A few of the test cases include:

- The system can add the account details of registered customers to the database
- The system allows customers to add books to a shopping basket.
- The system records the time orders were placed.

- The system holds the appropriate book data.
- The system allows users to browse while logged out
- Only registers can place orders.

The six tests mentioned previously all passed without issue. However, this is not the case for all test cases. For example, the system fails to record the timestamp they last logged into the system.

4. Summary

In this coursework I have carried out research and applied my own knowledge to establish specific project requirements, then I carried out the user interaction design which outlines how users interact with the system to perform specific functionalities. Then, following the Agile development methodology began designing, implementing, testing and debugging the artifact in iterations, through consistent sprints.

During the design phases of sprints, there were several good design decisions made, for example using a ‘SessionScoped’ backing bean. This meant that calls to the business logic would not have had to occur to add information to the database, and access information from the database, for every request. Another example of a good design decision was adding a ‘Check Credentials’ button to verify that the credentials entered in the form were valid. Further, the technological decision to modify the functionality of the ‘Check Credentials’ button so that the verification occurs in the business logic and that the function also verifies if the credentials are valid for all administrator credentials in the database, not just all customer credentials, to allow users to login as administrators from the ‘login’ page.

The design decision to add the ‘role’ field to the register account form was another effective decision. It allowed administrator accounts to be created, which is important because administrator accounts need to be registered before they can be logged in to perform administrator-based requirements. The technological decision to hide the customer and administrator register buttons until the ‘checkRole’ function called by the ‘Check Role’ button verified which type of account was being registered.

However, there were several bad design decisions made, one of those was deciding to incorporate so many views, making the decision to have, the login and register forms be in different views instead of rendering forms when the login and register buttons were clicked caused a considerable and unnecessary increase in heap space used. Another bad decision made was the technological decision to make many different list variables to populate very similar, if not identical, data tables, this could have been avoided by having multiple data tables print the same list.

With the benefit of hindsight, less view would have been created, with more functionality in each view, for example instead of switching to different views to login and register, the login button could have rendered what is in the ‘login’ page in the ‘home’ page and the same goes for the register page, a reason for this is that it could be potentially save a significant amount of heap space within the artifact. Further, the shopping basket view could have been forgone complete, this is because the shopping basket is printed in the checkout so customers can see what they are ordering as they order it.

References

deBara, D., & deBara, D. (2020, July 16). 11 top tips for outstanding ecommerce website design. *99designs*. https://99designs.com/blog/web-digital/ecommerce-website-design-tips/

Post, S. B. (2022, July 10). *6 Tips for Designing an Effective Data Entry GUI* [Video]. SPK And Associates. https://www.spkaa.com/blog/6-tips-for-designing-an-effective-data-entry-gui

Smith, S. L., & Mosier, J. N. (1986). *Data Entry*. GUIDELINES FOR DESIGNING USER INTERFACE SOFTWARE. Retrieved November 29, 2022, from https://hcibib.org/sam/1.html

What Is Agile Methodology in Project Management? (n.d.). https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/