Online Bookshop with Stock Control

Created by,

*X00179223 -* Raymond McCarthy

*X00167646 -* Taiwo Obadare

*X00162597 -* Vishvajit Sundarrajan

Supervised by,

- Patricia Magee

Contents

[Members Responsibilities 3](#_Toc70325211)

[Responsibilities of team member: Raymond McCarthy – X00179223 3](#_Toc70325212)

[Responsibilities of team member: Taiwo Obadare – X00167646 4](#_Toc70325213)

[Responsibilities of team member: Vishvajit Sundarrajan – X00162597 4](#_Toc70325214)

[Project Introduction 4](#_Toc70325215)

[Project Plan and Requirements 5](#_Toc70325216)

[Project Design 5](#_Toc70325217)

[Overall Goals: 5](#_Toc70325218)

[*Requirements*: 6](#_Toc70325219)

[*High priority:* 6](#_Toc70325220)

[*Medium priority:* 6](#_Toc70325221)

[*Low priority:* 6](#_Toc70325222)

[Class Diagram 7](#_Toc70325223)

[Use Case Diagram 7](#_Toc70325224)

[Use Case Descriptions 8](#_Toc70325225)

[Goals 12](#_Toc70325226)

[Iteration 1 (Prototype) – Delivery date: **5/3/21 – 10pm** 13](#_Toc70325227)

[Iteration 2 (Prototype) – Delivery date: **26/3/21 – 10pm** 13](#_Toc70325228)

[Iteration 3 (Final Iteration) – Delivery date: **26/4/21 – 10pm** 14](#_Toc70325229)

[Assessing Feasibility 14](#_Toc70325230)

[Financial Feasibility 14](#_Toc70325231)

[Technical feasibility 14](#_Toc70325232)

[Resource and time feasibility 14](#_Toc70325233)

[Risk feasibility 14](#_Toc70325234)

[Iteration 1 Report 15](#_Toc70325235)

[Work Completed 15](#_Toc70325236)

[Challenges Faced 15](#_Toc70325237)

[Approach Taken 15](#_Toc70325238)

[Test Cases Used & Results Obtained 15](#_Toc70325239)

[Iteration 2 Report 17](#_Toc70325240)

[Group 16: Book Store Project 17](#_Toc70325241)

[Team member: 17](#_Toc70325242)

[Progress made 17](#_Toc70325243)

[Test Cases 18](#_Toc70325244)

[Team member: 19](#_Toc70325245)

[Progress made. 19](#_Toc70325246)

[Test Cases 20](#_Toc70325247)

[Team member 21](#_Toc70325248)

[Progress made: 21](#_Toc70325249)

[Test Cases 22](#_Toc70325250)

[Iteration 3 Report 22](#_Toc70325251)

[Group 16: Book Store Project 22](#_Toc70325252)

[Team member: 23](#_Toc70325253)

[Progress made 23](#_Toc70325254)

[Test Cases 23](#_Toc70325255)

[Team member 23](#_Toc70325256)

[Progress made: 24](#_Toc70325257)

[Team member: 24](#_Toc70325258)

[Progress made: 24](#_Toc70325259)

[Conclusion 24](#_Toc70325260)

[Achievements 24](#_Toc70325261)

[Limitations 25](#_Toc70325262)

[Possible Future Enhancements 25](#_Toc70325263)

[Declaration of Academic Integrity 25](#_Toc70325264)

# Members Responsibilities

## Responsibilities of team member: Raymond McCarthy – X00179223

* Sign/up/in/out
* Custom user models
* Custom groups
* Cart application
* Refund functionality
* Password change functionality
* Django admin actions
* Manager dashboard
* Category/book slug
* Sales reports & charts
* Initial navbar with category links
* Footer
* CSS
* Models' unit tests
* Selenium User Acceptance Tests
* Planning & requirements documentation
  + Project Design
  + Requirements
  + Use Case Descriptions (split)
  + Goals
  + Assessing Feasibility
* Iteration 2 report template
* Iteration 3 report template

## 

## Responsibilities of team member: Taiwo Obadare – X00167646

* Voucher CRUD manager
* Website Design/Logo Creation
* Discount Alert
* JavaScript with listed responsibility.
* Wish List
* User Profile
* Books By Same Category.
* Review Functionality
* Blog functionality
* Average star ratings
* Top Star rated + Cheapest Book.
* Books database population
* Manager view Authentication
* Book CRUD
* URLS test

## 

## Responsibilities of team member: Vishvajit Sundarrajan – X00162597

* Voucher creation backend
* Email confirmation
* Order app
* Order history tracking
* Order thanks page
* Order history information
* Search Functionality
* Pagination Functionality
* Voucher database population
* Overall Styling
* Bug fixing
* Views Test
* Use Case Diagram
* CSS

# Project Introduction

For the second-year project our team consisting of Raymond McCarthy, Taiwo Obadare, and Vishvajit Sundarrajan, have created an electronic commerce website with the primary goal of selling goods for profit.

The type of e-commerce store we have chosen to create is a bookstore with stock control capabilities using Python, HTML, CSS, JS and the Django Framework. We have used Microsoft Azure DevOps for project management and version control.

You can visit the website at [taiwo.pythonanywhere.com](http://taiwo.pythonanywhere.com/)

For demonstration purposes, you can login to the website with these credentials:

* Manager account
  + Username: manager2
  + Password: easypass
* Customer account
  + Username: newuser10
  + Password: easypass

# Project Plan and Requirements

## Project Design

### Overall Goals:

Our end goal is to have a fully functional, easy to navigate user friendly website with all the functionality and features necessary to operate as a secure e-commerce store.

We will design database models to store information about our customers, products, and orders. Products will be uniquely identifiable with the use of UUID modelling. We will develop administration account types that are authorized to view, add, and update database information.

The website will have the ability for our customers to sign up, sign in and sign out. Our website will have a clear and organized visual user interface with the use of images to present our products, a navigation bar to traverse the website and drop-down menu links to access account features.

We will include pages for customer reviews as well as a contact us page with relevant information, these pages will use Django forms to allow customers to submit information.

Books will be easily located with a search feature and separated into relevant categories. Products can be viewed by category, product detail, and all products.

We will make use of the Django session framework to store cart information for our users. For credit card processing we will implement a Stripe payment facility. After making payment our customers will be redirected to a thank you page and can view their order details. We will also implement the ability to request a refund from a previous order.

We will make use of the slug field to make sure our products are easily found by search engines. And implement a voucher feature for sales and offers.

Our website will be well tested using software quality and assurance testing techniques for error handling and bug detection.

And finally, when requirements have been met, we will launch the website on a domain server.

## 

### Requirements:

### *High priority:*

* Web site structure.
* Web page design.
* Sign up/log in/log out.
* Database models.
  + Customers
  + Products
  + Orders
  + UUID
* Shopping cart functionality
  + Add items to cart.
  + Remove items from cart.
  + Empty cart.
  + Finalize order.
* Website deployment.
* Make payment.
* Django admin for performing add/update/delete.
* Django session framework to manage shopping cart.
* Stripe API functionality.

### *Medium priority:*

* Graphics and images.
* Review page.
* Paginator.
* Search.
* Navbar links.
* Contact us page.
* Returns.

### *Low priority:*

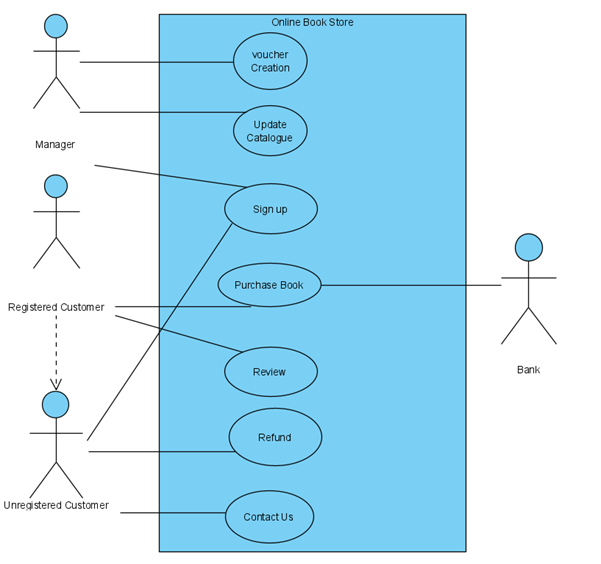
* Slug.
* Voucher

## Class Diagram

Diagram

Description automatically generated

## Use Case Diagram



## Use Case Descriptions

|  |  |
| --- | --- |
| **Use Case:** | Customer Sign up |
| **Actor (s):** | Unregistered customer |
| **Goal:** | Create an account for an individual customer, that stores their information for future logins and purchases |
| **Overview:** | The customer is registered and logged in. The customer is now present in the customer database. |
| **Pre-Condition:** | The customer must be unregistered and not logged into an account. |
| **Post Condition:** | The customer is registered and logged in. The customer is now present in the customer database. |
| **Successful Scenario:** | 1. Customer will land on homepage and be met with a ‘Sign Up’ link. 2. Customer clicks link and is redirected to sign up page. 3. Customer enters their chosen username, email address and password. 4. Customer confirms correct password. 5. Customer clicks ‘Sign Up’ submit button. 6. Customer is redirected to success page. 7. Customer is now registered. |
| **Alternative Scenario (s)** | 1. The customer’s details are unsuccessful due to already being registered. |

|  |  |
| --- | --- |
| **Use Case:** | Registered customer purchases book |
| **Actor (s):** | Registered customer |
| **Goal:** | Registered customer successfully purchases book |
| **Overview:** | The payment has been successful, and the customer is issued an order number. |
| **Pre-Condition:** | The customer must be registered. |
| **Post Condition:** | The customer has made a successful book purchase. |
| **Successful Scenario:** | 1. Customer will land on homepage and select log in, if not already logged in. 2. Customer enters username and password on log in page and clicks submit. 3. Customer is redirected to success page. 4. Customer searches for book. 5. Customer adds book to cart. 6. User navigates to cart and chooses pay now option. 7. Stripe API opens and customer is directed to enter payment details. 8. Customer enters credit card information and clicks submit. 9. Card payment is authenticated by stripe. 10. Payment is successful and order is logged. |
| **Alternative Scenario (s)** | 1. The customer’s payment is unsuccessful, and the ordered is not approved. |

|  |  |
| --- | --- |
| **Use Case:** | Unregistered customer purchases book |
| **Actor (s):** | Unregistered customer |
| **Goal:** | Unregistered customer successfully purchases book |
| **Overview:** | The payment has been successful, and the customer is issued an order number. |
| **Pre-Condition:** | N/A |
| **Post Condition:** | Customer has made a successful book purchase |
| **Successful Scenario:** | 1. Customer searches for book. 2. Customer adds book to cart. 3. Customer navigates to cart and chooses ‘pay now’ option. 4. Stripe API opens and customer is directed to enter payment details. 5. Customer enters credit card information and clicks submit. 6. Card payment is authenticated by stripe. 7. Payment is successful and order is logged. |
| **Alternative Scenario (s)** | 1. The customer’s payment is unsuccessful, and the ordered is not approved. |

|  |  |
| --- | --- |
| **Use Case:** | Manager updates catalogue |
| **Actor (s):** | Manager |
| **Goal:** | Manager adds a new book, deletes an old book, or updates a books details on the database. |
| **Overview:** | The new book is now visible for customers, the old book is no longer visible, or the book details and description have been updated. |
| **Pre-Condition:** | Manager is logged in with permissions. |
| **Post Condition:** | Database has updated book catalogue. |
| **Successful Scenario:** | 1. Manager logs in with manager permissions. 2. Proceeds to GUI version of the Django database admin. 3. Manager clicks, add, update, or edit book. 4. Manager enters new information and images, or manager removes old book from database. |
| **Alternative Scenario (s)** | 1. The manager cancels changes and logs out. |

|  |  |
| --- | --- |
| **Use Case:** | Manager adds voucher codes |
| **Actor (s):** | Manager |
| **Goal:** | Add voucher codes which will reduce price when entered at checkout. |
| **Overview:** | Voucher is stored in database for use. |
| **Pre-Condition:** | Manager is logged in with permissions. |
| **Post Condition:** | Customers can now enter voucher code at checkout and receive a discounted price. |
| **Successful Scenario:** | 1. Manager logs in with manager permissions. 2. Proceeds to GUI version of the Django database admin. 3. Manager clicks vouchers and adds a voucher code, discount amount and date of validity. 4. Clicks save. |
| **Alternative Scenario (s)** | 1. Code is no longer valid and total price remains unchanged. |

|  |  |
| --- | --- |
| **Use Case:** | Registered Customer reviews |
| **Actor (s):** | Registered Customer |
| **Goal:** | Registered Customer successfully leave a review |
| **Overview:** | Customer will land on homepage and select log in, if not already logged in. Customer will click the review page and if they want to give feedback with the services provided, the Customer can leave a review on the review page. |
| **Pre-Condition:** | The Customer must be registered. |
| **Post Condition:** | The Customer has made a successful review. |
| **Successful Scenario:** | 1. Customer logs in 2. Selects review page 3. Clicks the text box 4. Leaves a review 5. Exits page |
| **Alternative Scenario (s)** | 1. The Customer exceeds the text box character limit 2. Unable to leave a review |

|  |  |
| --- | --- |
| **Use Case:** | Customer uses contact us |
| **Actor (s):** | Customer |
| **Goal:** | Customer successfully contacts us |
| **Overview:** | Customer will land on homepage and select log in, if not already logged in. Customer will click the Contact us page. Customer will input info necessary to contact the online bookstore. |
| **Pre-Condition:** | The Customer must be on the website |
| **Post Condition:** | The Customer has successfully contacted the online bookstore |
| **Successful Scenario:** | 1. Customer goes into the website 2. Customer clicks the contact us page 3. Inputs the required fields data 4. Contacts the online bookstore 5. Exits page |
| **Alternative Scenario (s)** | 1. The customer does not have the required data 2. The customer is unable to contact the online webstore |

|  |  |
| --- | --- |
| **Use Case:** | Customer uses refund |
| **Actor (s):** | Customer |
| **Goal:** | Customer successfully refunds the selected product |
| **Overview:** | Customer will request a refund. Constraint will be checked to match user to product purchased. |
| **Pre-Condition:** | The Customer must be on the website |
| **Post Condition:** | The Customer has successfully refunded product. |
| **Successful Scenario:** | 1. Customer goes into the website 2. Customer clicks the refund button 3. Inputs the required fields data 4. Customer receives payment equivalent to items purchased |
| **Alternative Scenario (s)** | 1. The customer did not purchase items |

## Goals

### Iteration 1 (Prototype) – Delivery date: **5/3/21 – 10pm**

* We will meet for our scrum session on Microsoft teams to discuss the iteration and determine our approach for completing tasks on **16/2/21 at 9am**.
* Tasks set out:
* Develop a working system with core functionality.
  + Initialize and fork repository.
  + Create models.
    - Users
    - Products
    - Admin
  + Develop views, templates, and URLs.
  + Develop add, update, and delete functionality by Admin.
  + Design templates.
    - Base.html
    - Navbar.html
    - Products pages
    - Sign up/in/out.
    - Styling
* Paginator & Search functionality.
* Cart application.
* Design a series of tests to verify that the system is working correctly.
* Document any issues or problems encountered.
* Write a 700-word iteration reporting work completed.

### Iteration 2 (Prototype) – Delivery date: **26/3/21 – 10pm**

* By the time we have reached iteration two we plan to have a well-built foundation with basic functionality to build upon.
* We will meet on Microsoft teams for our scrum session to discuss the approach for our next iteration on **9/3/21 at 9am**.
* Tasks set out:
  + We will now implement additional functionality such as:
    - Strip payment API.
    - Orders app.
    - Thank you, page.
    - Contact us page.
    - Sign up, sign in & sign out.
    - Voucher’s functionality.
    - Many to many mapping
      * This will allow us to add the same product to multiple categories, e.g., ‘Crime’ & ‘Thriller’.
    - Incorporating the Django ‘Slug’ field.
  + More testing for our newly implemented code.
  + Removal of any residual bugs from previous tests.
  + Submit a 700-word report on work completed.

### Iteration 3 (Final Iteration) – Delivery date: **26/4/21 – 10pm**

* We will meet on Microsoft teams on **13/4/21 at 9am** for our scrum session to discuss our approach for the final iteration**.**
* At this stage of the project, we will have completed all coding and testing of the functionality of our website and be focusing on testing and making minor adjustments to improve overall functionality and appearance where necessary.
* We will prepare our final iteration documentation and our final prototype for review before **10pm** on the **26th of April 2021.**

## Assessing Feasibility

### Financial Feasibility

This website will be created as an amateur project without labor costs. The tools we have decided to use in developing our project are open source, and the hosting service we have chosen is free of charge. So, our current expenses for the project total are €0.

### Technical feasibility

The tools which we intend to use are:

* HTML5
* CSS
* JavaScript
* Python
* Django

The three members of our team all have relevant experience and knowledge in all the listed technologies and our technical skills have been determined to be sufficient based on our work history.

### Resource and time feasibility

The resources required for this project are:

* Programming devices (laptops/computers)
* Hosting space (freely available)
* Programming tools (open source)
* Programmers

We have concluded that there will be no issue in having access to all the necessary resources we need to produce our final product.

The time we have been allocated for each iteration of the project is 3 weeks which we have deemed perfectly feasible based on previous projects.

### Risk feasibility

Expert understanding of our chosen technologies is not evident amongst the team, although confident, we may assume likelihood of running into time consuming issues while trying to implement functionality into our project with the use of multiple coding languages.

It is our understanding from personal experience that the time that has been allocated to complete our project should be sufficient to allow for coding issues to arise and be dealt with accordingly, using online tutorials and forums such as stackoverflow.org, brainstorming amongst each-other and conferring with our supervisor.

# Iteration 1 Report

## Work Completed

Shop app, static files, search app, account app with Custom User model, views, URLs, templates, Category, Book, and Author model. Individual work assigned is represented in the commits.

## Challenges Faced

We faced a challenge with the model as we named the model wrongly, so we had to change it. We wanted to implement a many-to-many relationship with the category and books but after doing so we ran into issues with the get\_absolute\_url so implemented the one-to-many relationship. Also, the books were not displaying, and it was fixed by changing the for loop. And when entering the individual book page, it gave an error which is now resolved. An issue was encountered when creating the custom user model as to how permissions would be implemented. We were able to solve this problem by implementing roles in Django admin and assigning roles to our staff users which will allow them authorisation to perform certain tasks. Another issue found out was that when attempting to display the book through the search bar was no book was being displayed. The reason for this was that that the search was looking for a specific name in the views which was different from that of the name in the models. The issue was resolved by changing the view functions searching name to match that in the previously created model.

## Approach Taken

The Book shop contains Books that are based on distinct categories. So, a category model was created and a book model. Furthermore, the author of the book is also stored as a model. We have also provided a facility to allow a manager user. The search functionality is also implemented in the search app to provide search functionality

## Test Cases Used & Results Obtained

We tested the URLs, models, and views and tested the CRUD functionality via django admin. We tested to see if the relevant URL path was linked to the view and URL combo we wanted. AllBookCat, books\_by\_category and book\_detail. These passed. The views we tested were the searchResult function for this we reversed the URL and checked the template used as well as the response status code this view succeeded. We attempted to test the view for book\_detail however we received an error stating that there was no reverse match for the book\_detail view. Also, for the allBookCat we received an error stating that there was no image file associated with the allBookCat. We hope to debug this in by the end of the next iteration. To test the model's class designs we first initiated an instance of each model to ensure creation was successful, we tested that referential integrity was successful by implementing foreign keys in the tests. Once the objects were initiated, we asserted that the object was created with the same variables we initiated it with by using functions such as assertEquals() and assertTrue().

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case Matrix** | | | | | |
| **Test Case No** | **Test Case Name** | **Input** | **Expected Outcome** | **Actual Outcome** | **Result (Pass/Fail)** |
| 01 | Add Book | Input IABN: TEST, TITLE:TEST, SYNOPSIS: TEST, PRICE: 1234, START RATING: 3, IMAGE:IMAGE, STOCK:4, PUB DATE:TODAY, NUM PAGES 1, PUBLISHER: TEST, AUTHOR: TOM, CATEGORY: KIDS | The book “TEST” was  added successfully. | The book “TEST” was added successfully. | pass |
| 02 | Delete Book | Delete django admin | Successfully deleted 1 book. | Successfully deleted 1 book. | Pass |
| 03 | Update Book | Input Start rating: 5 | The book “TEST” was  changed successfully. | The book “TEST” was changed successfully. | pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 04 | Add Category | Name: TEST, DESCRIPTION:TEST, image:IMAGE, POPULARITY: 3 | The category “TEST” was added successfully. | The category “TEST” was added successfully. | pass |
| 05 | Update Category | Input Popularity 5 | The category “TEST” was changed successfully. | The category “TEST” was changed successfully. | pass |
| 06 | Delete Category | Delete via django admin | Successfully deleted 1 category. | Successfully deleted 1 category. | pass |
| 07 | Add Author | Input First name: TEST, Last name: TEST, Age: 23, About: TEST | The author “TEST” was added successfully. | The author “TEST” was added successfully. | pass |
| 08 | Delete Author | Delete via django admin | Successfully deleted 1 author. | Successfully deleted 1 author. | pass |
| 09 | Update Author | Input age 42 | The author “TEST” was changed successfully. | The author “TEST” was changed successfully. | pass |

# Iteration 2 Report

### Group 16: Book Store Project

*Raymond McCarthy, Taiwo Obadare, Vishvajit Sundarrajan.*

*Supervisor: Patricia Magee.*

Team member: **Raymond McCarthy**

### Progress made

**Cart application:** Development of cart application and template to display cart content, with view functions to allow a user to add books to cart, remove a single book from cart, and remove all items from the cart. And models to store cart content information.

**Manager/customer roles:** Use of Django roles, groups, and permissions to develop manager and customer user types. Manager was given high-level permissions, whilst customer was given limited permissions.

**Sign up/in/out functionality:** Development of templates that display forms for users to sign up and sign in. View functions that allow users to sign up, sign in and sign out and store their details into models in the database.

**'Contact Us' application:** Development of a contact template with a submission form and a ‘thank you/confirmation’ template which renders upon successful submission. A view function to ‘post’ entries and a model to store the data entered.

**Refund functionality:** Development of templates with a form to submit a refund application and success page. Updated order model to include a reference code, delivery status, refund status. View function to take form data and update order object. Model to store refund records.

**Slug:** Updated book and category models to include a slug field with autoslug. Updated all templates and functions that use a get method or dot operator on books or categories, to get objects by slug field. Updated all uuid URLs to slug URLs.

**Password change form:** Developed a view function to change password, and template to change the style of the Django admin password\_change form to fit the front-end styling.

### Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case Matrix** | | | | | |
| **Test Case No** | **Test Case Name** | **Input** | **Expected Outcome** | **Actual Outcome** | **Result (Pass/Fail)** |
| 01 | Test\_blog\_create\_success | Create blog object, test model fields present | Blog record created and initialised with fields | Blog record created and initialised with fields | Pass |
| 02 | test\_cart\_create\_success | Create cart object, test model fields present | Cart record created and initialised with fields | Cart record created and initialised with fields | Pass |
| 03 | Test\_cart\_item\_create\_success | Create cart item object, test model is present | Cart item record created and initialised with fields | Cart item record created and initialised with fields | Pass |
| 04 | test\_contact\_create\_success | Create contact object, test model is present | Contact record created and initialised with fields | Contact record created and initialised with fields | Pass |
| 05 | test\_order\_create\_success | Create order object, test model fields present | Order record created and initialised with fields | Order record created and initialised with fields | Pass |
| 06 | test\_order\_item\_create\_success | Create order item object, test model fields present | Order item record created and initialised with fields | Order item record created and initialised with fields | Pass |
| 07 | test\_book\_is\_assigned\_slug\_on\_creation | Create book object, test slug field is assigned on creation | Book is created and automatically assigned a slug field | Book is created and automatically assigned a slug field | Pass |
| 08 | test\_category\_is\_assigned\_slug\_on\_creation | Create category object, test slug field is assigned on creation | Category is created and automatically assigned a slug field | Category is created and automatically assigned a slug field | Pass |

Team member: **Taiwo Obadare**

### Progress made.

**User Review functionality:** Authenticated user can write a review, which is displayed with the date posted and username. Was queried with newest review displayed first on the page. JavaScript was used to show number of reviews displayed currently on page and to hide the review form at first and display it on user click. Furthermore

**Blog Functionality:** App allows a user to write a blog about particular book, displays book at the end of blog body. Reviews by writer of blog displayed at the right and a link to the relevant book user has reviewed in the past. Was queried in the view to only allow the first 5 reviews from the author

**Wish List Functionality:** This feature is created by a model, that has the user, the book etc. The book\_detail.html page has a link that when clicked calls on the add\_wishList view wich accepts the book slug on the link and adds to the Wishlist database by current user. It tells user already added through django message framework, else if it does not exist in the get query it creates the object and saves the object to the database displaying the success message. User can view their Wishlist and delete/add to cart.

**Stripe/PayPal payment:** Payment functionality using stripe and PayPal, which displays the relevant price of the book(s). And allows a user complete purchase.

**Website Styling:** Improved overall website design.

**Manager Authentication/CRUD Front End:** Allows for the manger to access the add, view, delete, edit functionality only. And updates the book in the database and add all fields except image when updating and adding book.

**\*Only first testcase URL from each app will be written.**

### Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case Matrix (URLS)** | | | | | |
| **Test Case No** | **Test Case Name** | **Input** | **Expected Outcome** | **Actual Outcome** | **Result (Pass/Fail)** |
| 01 | Test blog list URL is resolved with function blogView. | Reverse of blog list url and resolve to get function | BlogView  The relevant view corresponding to the URL | BlogView  The relevant view corresponding to the URL | Pass |
| 02 | Test add\_cart URL is resolved with function add\_cart | Reverse of add\_cart url and resolve to get function | add\_cart  The relevant view corresponding to the URL | add\_cart  The relevant view corresponding to the URL | Pass |
| 03 | Test contact\_url  URL is resolved with function contactView | Reverse of contact\_url and resolve to get function | contactView  The relevant view corresponding to the URL | ContactView  The relevant view corresponding to the URL | Pass |
| 04 | Test topRatedBooks URL is resolved with function topRatedBooks | Reverse of topRatedBooks url and resolve to get function | topRatedBooks  The relevant view corresponding to the URL | TopRatedBooks  The relevant view corresponding to the URL | Pass |
| 05 | Test apply\_url  is resolved with function voucher\_apply | Reverse of apply url and resolve to get function | voucher\_apply  The relevant view corresponding to the URL | voucher\_apply  The relevant view corresponding to the URL | Pass |
| 6 | Test apply\_url  signup\_url  is resolved with function signupView | Test signup\_url  is resolved with function signupView | signupView  The relevant view corresponding to the URL | signupView  evant view corresponding to the URL | Pass |
| 7 | Test Thanks\_url  is resolved with function voucher\_apply | Test  Thanks\_url  is resolved with function voucher\_apply  thanks | thanks  The relevant view corresponding to the URL | thanks  The relevant view corresponding to the URL | Pass |

Team member: **Vishvajit Sundarrajan**

### Progress made:

**Thanks Page:** Developed a thank you template which upon creation of a successful order displays a confirmation of the user's purchase gives them order id as well as displaying message of thanks to the customer.

**Order Functionality:**  This functionality stores the user's personal information in the order model. Then calculates the total costs of the user’s purchases based on the number of the items in the user’s cart. The user’s information is then displayed with relevant corresponding payment information of the user at checkout.

**Order history page:**  Users who have made purchases have the ability to view all purchases made on this page. When the users click on a view order link it will display all the information of the previously made purchases associated with their accounts.

**Vouchers:** Implemented voucher functionality which allows users with manager privilege to create

discount vouchers. The which users can use before payment to reduce overall price of order items.

**Website Styling:**  Improved the website layout.

**Search Functionality:** Implemented the search functionality of the website in which the user can search a book based on category or author name and is taken to the relevant page for the book requested.

### Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case Matrix** | | | | | |
| **Test Case No** | **Test Case Name** | **Input** | **Expected Outcome** | **Actual Outcome** | **Result (Pass/Fail)** |
| 01 | test\_cheap\_books | Cheap books url | Status code of cheap book page received | Status code of page received | Pass |
| 02 | test\_topRatedBooks | Top rated books url | Status code of top rated book page received | Status code of cheap book page received | Pass |
| 03 | test\_book\_detail | Book detail url | Status code of top rated book page received | Status code of top rated book page received | Pass |
| 04 | Test\_add\_to\_wishList | Wish List url | Status code of top rated book page received | Status code of top rated book page received | Pass |
| 05 | Test\_delete \_f |  |  |  |  |

# Iteration 3 Report

### Group 16: Book Store Project

*Raymond McCarthy, Taiwo Obadare, Vishvajit Sundarrajan.*

*Supervisor: Patricia Magee.*

Team member: **Raymond McCarthy**

### Progress made

**Manager dashboard:** Development of a dashboard which holds data that may be beneficial for a manger to review. It includes a main page with information about sales, such as total sales for the year, month and day which are derived from the order model. Charts displaying sales by month and book stock levels derived from book model queries.

There are links to pages which display user accounts information, orders details, crud functionality to update books which was created by another team member in the previous iteration and based on a suggestion by our supervisor I moved this functionality from the main site to the manager dashboard and changed the styling for more of a stock control perspective.

There is also frontend voucher CRUD within the dashboard which was implemented by my teammate.

**User Acceptance Tests:** I used selenium IDE to develop user acceptance tests which will be detailed below. The **.side** file can be found in the documents folder in the git repository.

### Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **00001** | |
| **Test case name:** | **Add/remove item to wish list** | |
| **Summary: A registered user should be able to login and add an item to their wish list then remove it.**  **Precondition: The user must be registered with username and password.**  **Assumption: A supported browser is being used.** | | |
| **Test steps:** | | **Expected Result:** |
| Open taiwo.pythonanywhere.com | | Bookspace homepage launches successfully |
| Click sign in link | | Sign in page loads |
| Select username field and enter the username of registered user (newuser10) | |  |
| Select the password field and enter the password of the registered user (easypass) | |  |
| Click the submit button | | User is logged in |
| Click on book in catalogue | | Book detail page opens |
| Click add to wish list button | | Book is added to wish list |
| Verify text “added to wish list” | | Test is present |
| Click wish list link | | Wish list page opens |
| Verify text “Book title” | | Book title is present in wish list page |
| Click delete button | | Book is removed from wish list |

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **00002** | |
| **Test case name:** | **Add/remove book to cart** | |
| **Summary: A registered user should be able to add a book to their cart and then remove it** | | |
| **Test steps:** | | **Expected Result:** |
| Open taiwo.pythonanywhere.com | | Bookspace homepage launches successfully |
| Click sign in link | | Sign in page loads |
| Select username field and enter the username of registered user (newuser10) | |  |
| Select the password field and enter the password of the registered user (easypass) | |  |
| Click the submit button | | User is logged in |
| Click on book in catalogue | | Book detail page opens |
| Click add to cart button | | Book is added to cart |
| Click remove button | | Book is removed from cart |
| Verify text “Your shopping cart is empty” present | | Text is present |

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **00003** | |
| **Test case name:** | Search book | |
| **Summary: User can search for a book and matching results will be displayed** | | |
| **Test steps:** | | **Expected Result:** |
| Open taiwo.pythonanywhere.com | | Bookspace homepage launches successfully |
| Click search box | |  |
| Enter book to search for | |  |
| Press enter | | Book matching query is displayed |
| Verify text of book title | | Text present |

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **00004** | |
| **Test case name:** | Sign in/out | |
| **Summary: A registered user should be able to login successfully and log out successfully** | | |
| **Test steps:** | | **Expected Result:** |
| Open taiwo.pythonanywhere.com | | Bookspace homepage launches successfully |
| Click sign in link | | Sign in page loads successfully |
| Click username box and enter username (newuser10) | |  |
| Click password box and enter password (easypass) | |  |
| Click submit | | User is now logged in |
| Verify username present on page | | Username present |
| Click sign out | | User is now signed out |

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **00005** | |
| **Test case name:** | Submit contact form | |
| **Summary: A user should be able to successfully submit a contact form entry and be notified of its success** | | |
| **Test steps:** | | **Expected Result:** |
| Open taiwo.pythonanywhere.com | | Bookspace homepage launches successfully |
| Click contact link | | Contact page opens |
| Click first name box | |  |
| Enter first name “fname” | |  |
| Click last name box | |  |
| Enter last name “lname” | |  |
| Click email box | |  |
| Enter email “fname@email.com” | |  |
| Click subject box | |  |
| Enter subject “selenium text” | |  |
| Click submit | | Form submission successful page opens |
| Verify text “thanks for contacting us” | | Text is present |

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **00006** | |
| **Test case name:** | Login manager/view dashboard | |
| **Summary: A manager account should be able to login and navigate to the manager dashboard page**  **Prerequisite: User account must be assigned to manager group** | | |
| **Test steps:** | | **Expected Result:** |
| Open taiwo.pythonanywhere.com | | Bookspace homepage launches successfully |
| Click sign in link | | Sign in page loads successfully |
| Click username box and enter username (manager2) | |  |
| Click password box and enter password (easypass) | |  |
| Click submit | | User is now logged in |
| Verify ‘DASHBOARD’ link present | | Link is present |
| Click dashboard link | | Dashboard page opens |
| Verify “total sales all time” element present | | Element is present |

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **00007** | |
| **Test case name:** | Manager add/remove voucher | |
| **Summary: A manager account should be able to create a new voucher via the manager dashboard and delete the created voucher**  **Prerequisite: User must be registered and assigned to manager group** | | |
| **Test steps:** | | **Expected Result:** |
| Open taiwo.pythonanywhere.com | | Bookspace homepage launches successfully |
| Click sign in | | Sign in page opens |
| Click username box and enter “manager2” | |  |
| Click password box and enter “easypass” | |  |
| Click submit | | User signed in successfully |
| Click dashboard link | | Dashboard page opens |
| Click vouchers link | | Voucher front end crud page opens |
| Click add voucher button | | Add voucher page opens |
| Click code box and enter “TEST” | |  |
| Click valid from box and enter “1/1/11” | |  |
| Click valid to box and enter “2/2/32” | |  |
| Click amount box and enter “5” | |  |
| Click active tick box | |  |
| Submit | | Voucher is created and active, redirects to voucher list page |
| Click delete on newly created voucher | | Are you sure ‘confirm’ button loads |
| Click confirm button | | Redirects to voucher list page, voucher has been deleted |

Team member: **Vishvajit Sundarrajan**

### Progress made:

**Email Confirmation:** Developed email confirmation system for authenticated users if logged in with a valid email to receive an automatically created email confirmation of their purchase thanking them for the use of the website.

**Search filters:**  Updated the search functionality to a wider choice of search functionality based on a greater range of choices. Now the users can search the books based on the category,publisher and iban number on top of the already created basic searching based on the authors first and last name.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case Matrix (Views)** | | | | | |
| **Test Case No** | **Test Case Name** | **Input** | **Expected Outcome** | **Actual Outcome** | **Result (Pass/Fail)** |
| 01 | Test profileView url gives a reverse and has a status code | Reverse of profile view | profileView  Relevant status code | profileViewRelevant  status code | Pass |
| 02 | Test blogDetail url gives a reverse and has a status code | Reverse of blogDetail | blogDetail Relevant status code | blogDetail Relevant status code | Pass |
| 03 | Test add\_cart url gives a reverse and has a status code | Reverse of add\_cart | add\_cart Relevant status code | add\_cart Relevant status code  URL | Pass |
| 04 | Test ContactView url gives a reverse and has a status code | Reverse of ContactView | ContactView Relevant status code | ContactView Relevant status code | Pass |
| 05 | Test manager\_dashboard url gives a reverse and has a status code | Reverse of manager\_dashboard | manager\_dashboard  Relevant status code | manager\_dashboard Relevant status code | Pass |
| 06 | Test refund url gives a reverse and has a status code | Reverse of refund | refund Relevant status code | refund  Relevant status code | Pass |
| 07 | Test SearchResult url gives a reverse and has a status code | Reverse of SearchResult | SearchResult Relevant status code | SearchResult  Relevant status code | Pass |
| 08 | Test cheapBook surl gives a reverse and has a status code | Reverse of cheapBooks | cheapBook Relevant status code | cheapBook  Relevant status code | Pass |
| 09 | Test voucher\_apply gives a reverse and has a status code | Reverse of  voucher\_apply | voucher\_apply  Relevant status code | voucher\_apply  Relevant status code | Pass |

Team member: Taiwo Obadare

### Progress made:

**Manager Discount Alert:** Functionality that allows manager to announce discount code on site. It is displayed also in the individual book page.

**Deploy:** Deployed app using python anywhere <https://taiwo.pythonanywhere.com>

**Redesign:** Redesigned shopping cart, manager dashboard now is full screen. Hover effect on navbar effect. Profile image inside navbar. Manager badge in navbar when logged in as manager. Underlined hover effect in navbar. Hover effect on book hover.

**Voucher CRUD front end:** Create, Delete, Update and View voucher in manager dashboard.

**User Profile:** User has profile. User has profile image wish can be changed here. And the user profile also has a name and last name and colour. The reviews, wish lists, orders and blogs are displayed here. The refund/Password reset functionality was implemented by my teammate.

**Blog Create:** Before it was just manager that was supposed to add a blog, but it was changed now to all users that have a profile.

**Star-rating Functionality:** A logged in user can add a star rating for a book in the review. And it is added and averaged out into the books star rating field. And the average star rating is displayed beside the book including half stars when appropriately between .5 and 1.

**Books By Category:** Under the individual book you can see books in the same category displayed under.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case Matrix (URLS)** | | | | | |
| **Test Case No** | **Test Case Name** | **Input** | **Expected Outcome** | **Actual Outcome** | **Result (Pass/Fail)** |
| 01 | Test profile\_view URL is resolved woth function profileView | Reverse of profile\_view urland resolve and get function | ProfileView  The relevant view corresponding to the URL | ProfileView  The relevant view corresponding to the URL | Pass |
| 02 | Test dashboard\_view  URL is resolved with function dashboardView | Reverse of dashboard\_view URL and resolve and get function | dashboardView  The relevant view corresponding to the URL | dashboardView  The relevant view corresponding to the URL | Pass |
| 03 | Test add\_blog  URL is resolved with function addBlog | Reverse of add\_blog  URL and resolve and get function | addBlog  The relevant view corresponding to the URL | addBlog  The relevant view corresponding to the URL | Pass |
| 04 | Test user\_list  URL is resolved with function userListView | Reverse of userListView  URL and resolve and get function | Reverse of user\_list  URL and resolve and get function | Reverse of user\_list  URL and resolve and get function | Pass |
| 05 | Test reviewList  URL is resolved with function reviewList | Reverse of reviewList  URL and resolve and get function | Reverse of reviewList  URL and resolve and get function | Reverse of reviewList  URL and resolve and get function | Pass |
| 06 | Test voucher\_list  URL is resolved with function voucherListView | Reverse of voucher\_list  URL and resolve and get function | Reverse of voucher\_list  URL and resolve and get function | Reverse of voucher\_list  URL and resolve and get function | Pass |

# References

<https://docs.djangoproject.com/>

<https://stackoverflow.com/>

<https://www.chartjs.org/>

<https://django-autoslug.readthedocs.io/en/latest/fields.html>

# Conclusion

## Achievements

In conclusion we are very proud of the achievements made thought the project. We developed great teamwork skills and communication especially in the covid-19 pandemic situation in which we couldn't meet at university. We gained important time managements skills had very intense time frames for each iteration and each member helped contribute to the project significantly.

The contributions together accumulated to an eCommerce website called Bookspace we are very proud of that has lots of important functionality that we originally planned for in our class and use case diagrams at the start. When we researched and improved upon the project, we created additional functionalities we needed and overall, we are incredibly satisfied with our project.

## Limitations

Some limitations we encountered in the process of developing this website included those caused by Covid-19, which made it impossible to meet, strategize, and plan in person so most of our communication was completed via text messaging and it can be difficult using this medium as topics and conversation can be sporadic and sparse.

Another limitation we believed to hold us back was the workload from other modules in our degree programme, which kept us under pressure. In a regular working environment, you might have a lesser workload and be able to focus on just a few tasks at a time all of which relevant to the project, I believe this would keep you sharp in the relevant skills, whereas in a university programme, you are required to work on many assignments with short deadlines, whilst also preparing for quizzes and exams in non-related topics. This can often make it difficult to pick up where you left off.

## Possible Future Enhancements

With more time to work on this project we could potentially have familiarised ourselves better with JavaScript and built some more interactive UI elements. If we had time in the future to improve this project, we would like to make the website more responsive and friendly to multiple device types.

# Declaration of Academic Integrity

Student Compliance with Academic Integrity

All students are expected to complete their courses in compliance with University regulations and standards. No student shall engage in any activity that involves attempting to receive a grade by means other than honest effort, for example:

1. No student shall complete, in part or in total, any examination or assignment for another person.
2. No student shall knowingly allow any examination or assignment to be completed, in part or in total, for himself or herself by another person.
3. No student shall plagiarize or copy the work of another and submit it as his or her own work.
4. No student shall employ aids excluded by the instructor in undertaking course work.
5. No student shall knowingly procure, provide, or accept any materials that contain questions or answers to any examination or assignment to be given at a subsequent time.
6. No student shall procure or accept assignments from any other student from current or prior classes of this course.
7. No student shall provide their assignments, in part or in total, to any other student in current or future classes of this course.
8. No student shall submit substantially the same material in more than one course without prior authorization.
9. No student shall alter graded assignments or examinations and then resubmit them for regrading.
10. All programming code and documentation submitted for evaluation or existing inside the students computer accounts must be the students original work or material specifically authorized by the instructor.
11. Collaborating with other students to develop, complete or correct course work is limited to activities explicitly authorized by the instructor.
12. For all group assignments, each member of the group is responsible for the academic integrity of the entire submission.

**N.B to be filled out by each member of the team.**

By including my name in the form below, I declare that I understand and will abide by the University Regulations and Policies covering Academic Integrity. I accept that each member of the group is responsible for the academic integrity of the entire submission. I will retain a copy of this agreement for future reference.

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name (1):** | **Raymond McCarthy** | **Student No.:** | **X00179223** |
| **Student Name (2):** | **Vishvajit Sundarrajan** | **Student No.:** | **X00162597** |
| **Student Name (3):** | **Taiwo Obadare** | **Student No.:** | **X00167646** |
| **Student Name (4):** |  | **Student No.:** |  |
| **Module title:** | **Year 2 Project** | | |
| **Programme Title:** | **BSc. in Computing with Software Development** | | |
| **Date: DD/MM/YYYY** | **26/04/2021** |