Folio

An Ebook Retailer

Zana Ahmad Hugh Alun-Jones Thomas Akers Emmanuel Adedeji

Contents

List of Tables

System Specification

Folio is a website for users to buy and download e-books. Users have the option to search for an e-book using an author name or a title of a book. The homepage will present the user with various books that are randomly chosen. There will also be a browse page to look for a book via publisher, date of publication, author or genre. The project itself consists of 3 systems. Please see the relevant subsections to find out more.

Other pages that would be included within the website include the login and register pages where the user will be asked to enter their details to either register or login. Once this is completed, they can buy a book they would like which then gets added to the Order System for recording

who has bought what. Folio consists of four systems that are allocated between four people. More details of each system are shown below under each subsection.

At the backend, we have access to the database so we will directly access data and do all the relevant actions such as addition, deletion, filtration and modification, listing and validation. A form will be designed to take in input for example for adding a new book entry.

Pre-defined SQL queries will be designed to instantly output relevant data for ease of use. For example, a query will be made to return all the users in the database or all books that are bought by a user. The result will return a list.

User Management System

This system is run by Zana who will be creating the following functionality:

Please note that the abreviation UMS is used to repersent the 'User Management System'

As mentioned before, this system will consist of two parts; a backend facing windows form application and a frontend website.

It will carry out functions such as:

- **Adding a user:** for the backend facing form, an admin can add a user by a form that has been provided for them that consists of the same fields as the frontend page will. It will ask for information such as first name, last name, date of birth, email and telephone number.
- **Editing a user:** at the backend, the admin will be able to search for a user and then bring up all the editable fields that were provided and modify them as they please as long as it can be validated. On the frontend, the user can do the same thing but only for their own account.
- **Deleting a user:** the admin can search for a user and completely delete the record. At the frontend the user can choose to delete their own record.
- **Listing users or a single user:** on the backend, an admin can choose to see all the users or search for specific ones. Once this has been done, they can then view the details of that user. A user on the frontend can see their own details on their profile page.
- **Finding users:** an admin can find another user at the backend. This will not be possible as of yet for the frontend. Although it may be an extra feature that could be useful if enough time was allocated.
- **Filtering users:** an admin can use the windows form at the backend to filter out only the users they require to see. For example, an admin can ask the database to retrieve data from a user that is born on April 8^{th} and that they have bought 32 books.
- **Validating users:** any information should be validated that are going into the database. For example, a user should be at least at minimum age range, possibly at 16 which ensures a user legally can make a purchase. Validation is going to be done at the backend of the application that talks to the database. However, the frontend user will still be notified if something goes wrong such as too few characters for a password.

It is worth mentioning that the windows form will not have a login form for admins. We expect that the machine will be kept secure by providing a password for their machine.

User Schema

Below is the schema for a user. On the register page they would have to provide the following details.

1. First name
2. Last name
3. Date of birth

4. Email
5. Telephone number

Then these data will be sent to the server to be validated. An email also will be sent to the user to verify their information which is essential if they want to buy an e-book so that we know that they will receive an email for the receipt and the pdf of the desired book.

Table 1: User Schema Table

Attribute	Type (`datatype`)	Key
user_id	String `varchar(25)`	Private Key
user_fullname	String `varchar(50)`	
user_password	String `varchar(50)`	
user_dob	Date `date`	
user_email	String `varchar(255)`	
user_tel	String `varchar(13)`	
user_numof_books_bought	Integer `int`	
user_is_email_verified	Boolean `bit`	

With every purchase, we will increment the `user_numof_books_bought` field to keep record of how many books each individual has bought. This may come in handy to see how popular the website as a whole is.

When a user is authenticated whether through the frontend or backend, the rest of the application is handled by the other systems.

Usecases

Table 2: (UMS Usecase) Admin Add User

Usecase	Usecase Name
Usecase Name	Admin Add User
Usecase Description	The admin adds a user from the backend application.
Usecase Author	Zana
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	Admin will have the option on the GUI to add a new user. Once clicked on the button, a form will be presented and the admin will be able to input the relevant data.
Usecase Alternate Pathways(s)	N/a
Usecase Exception Pathway(s)	Database refuses to connect. In this case, admin or user cannot access the relevant data.

Table 3: (UMS Usecase) Admin Edit User

Usecase	Usecase Name
Usecase Name	Admin Edit User
Usecase Description	The admin edits an existing user information.
Usecase Author	Zana
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	Admin will first have to find a user. Once selected, a button will be available to press and access the data of the selected user. A form, similar to add user will be present. Only difference is, it will most likely already be populated.
Usecase Alternate Pathways(s)	N/a
Usecase Exception Pathway(s)	User does not exist anymore. Since the user can delete the entry, the admin may not able to find the user. In cases where a user was deleted during the edit phase by the admin, when an admin submits, the user may not exist. An error should be presented rather than readding the user.

Table 4: (UMS Usecase) User Edit User

Usecase	Usecase Name
Usecase Name	User Edit User
Usecase Description	The user edits their own user information.
Usecase Author	Zana
Usecase Actor	User
Usecase Location	Frontend
Usecase Primary Pathway	The user will see an edit button on their page. Once clicked, they will be redirected to an edit form containing their information. From there they have options to edit whatever data they want. Also a delete button (Please see User Delete User usecase).
Usecase Alternate Pathways(s)	Users should not be able to access data on other users. If some pages are not limited, they will have access to other peoples data.
Usecase Exception Pathway(s)	Admin may have deleted a users account so a user cannot login anymore so they cannot access any of their data to see or edit. Admin may also have changed the users details which could cause login issues.

Table 5: (UMS Usecase) Admin Update User

Usecase	Usecase Name
Usecase Name	Admin Update User
Usecase Description	Admin updates a users data
Usecase Author	Zana
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	After edits have been made, admin will click the update button to confirm changes.
Usecase Alternate Pathways(s)	N/a
Usecase Exception Pathway(s)	The forms will have some validation so that they information comply with the input specification. In this case, errors will be shown and if data is sensitive, they will be required to type again.

Table 6: (UMS Usecase) User Update User

Usecase	Usecase Name
Usecase Name	User Update User
Usecase Description	A user updates their own information
Usecase Author	Zana
Usecase Actor	User
Usecase Location	Frontend
Usecase Primary Pathway	After they have filled in the relevant fields, an update link will be provided to redirect them to a success page.
Usecase Alternate Pathways(s)	A direct request to the update link should redirect to the user's edit page.
Usecase Exception Pathway(s)	User may have been deleted by the admin at the time of a user updates. This case, the user should be logged out with a message.

Table 7: (UMS Usecase) Admin Delete User

Usecase	Usecase Name
Usecase Name	Admin Delete User
Usecase Description	An admin deletes a specified user.
Usecase Author	Zana

Usecase	Usecase Name
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	Admin, on the edit page will be presented with a delete button. This way, no accidental deletions are made. A message will also appear before a deletion is confirmed.
Usecase Alternate Pathways(s)	N/a
Usecase Exception Pathway(s)	The database connection may be lost. In this case, a message should be shown to the admin and the deletion to fail.

Table 8: (UMS Usecase) User Delete User

Usecase	Usecase Name
Usecase Name	User Delete User
Usecase Description	A user deletes their account
Usecase Author	Zana
Usecase Actor	User
Usecase Location	Frontend
Usecase Primary Pathway	User, on the edit page, will be given a red delete button which will show a popup to confirm deletion. This can only be done if they type in their password.
Usecase Alternate Pathways(s)	N/a
Usecase Exception Pathway(s)	The system might throw a null pointer exception if the user was already deleted by the admin. Instead, they should be redirected to a register page and with a relevant message to tell the user their account was deleted.

Table 9: (UMS Usecase) List User

Usecase	Usecase Name
Usecase Name	List User
Usecase Description	Admin lists the users
Usecase Author	Zana
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	Admin will be presented with all the users once they click “see users” on the backend homepage application

Usecase	Usecase Name
Usecase Alternate Pathways(s)	N/a
Usecase Exception Pathway(s)	Once the button is clicked, it is possible that the database contains no users. In this case, a message should tell the admins that there are no users.

Table 10: (UMS Usecase) Find User

Usecase	Usecase Name
Usecase Name	Find User
Usecase Description	Admin finds a user based on the information the users gave during registration. Please see the user schema table for more details on the available fields.
Usecase Author	Zana
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	User will be given options on the "see users" page on the backend. Once they are in that part of the application, they will be listed with all the users and a few fields such as inputs to search for users. To find a specific user, we can use the id.
Usecase Alternate Pathways(s)	The same user may match if the other input fields were filled in. They admin has to clear the form to ensure that the system only takes the id as a search query.
Usecase Exception Pathway(s)	A search query might show no results due to no matching user in the database. In this case, a message would be useful to show that there was no return search result..

Table 11: (UMS Usecase) Filter User

Usecase	Usecase Name
Usecase Name	Filter User
Usecase Description	Admin searches through all users, finding only the ones that match a specific query.
Usecase Author	Zana
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	Forms will be provided which will be used as a search query. They will be on the same page as the listview for users. If any inputs match a user, they will be displayed in the listview.
Usecase Alternate Pathways(s)	N/a

Usecase	Usecase Name
Usecase Exception Pathway(s)	A search query might show no results due to no matching **users** in the database. In this case, a message would be useful to show that there was no return search result.

Table 12: (UMS Usecase) Admin Validate User

Usecase	Usecase Name
Usecase Name	Admin Validate User
Usecase Descriptions	When a new user is being added or existing one is being edited on the backend, on submit, the data gets checked before any more action is taken.
Usecase Author	Zana
Usecase Actor	Admin
Usecase Location	Backend
Usecase Primary Pathway	On the "add user" or "edit user" forms, when submit is pressed, the request data is validated and made sure everything should be both compatible with the database and correct before it is sent to the database. If it is not, then the admin will get messages to help with resolving the issue or issues.
Usecase Alternate Pathways(s)	N/a
Usecase Exception Pathway(s)	If no data is sent, the admin should be able to get the relevant messages back as well as an option to cancel the form so that if they change their mind, they can just exit safely without changing any data.

Table 13: (UMS Usecase) User Validate User

Usecase	Usecase Name
Usecase Name	User Validate User
Usecase Description	The user, when they register or they edit their information will have their request data to be validated and checked before the information is submitted to the database similar to Admin Validate User usecase except this will be available for user login too.
Usecase Author	Zana
Usecase Actor	User
Usecase Location	Frontend
Usecase Primary Pathway	There are 3 primary paths. When a user tries to login, data will be sent to be checked for malicious intent and check if the users email is an email format.When a new user is registering, it will do the same thing as above, except it will do it for all the fields. So if telephone number is not a in the right format, validation for it will fail.When a user tries to update their information, they will have to go through the same validation as above.

Usecase Usecase Name
Usecase N/a
Alter-
nate
Pathways(s)
Usecase N/a (besides the normal expected error from the failed validation)
Ex-
cep-
tion
Pathway(s)

Book Management System

This system is run by Thomas who will be creating the following:

The work I have been allocated is the creation of the book management system, this system contains all information regarding all books including their title, price and type. All the data in this table will be in the backend of the system and will only be used by the front end for fetching the required information. This record will allow for adding, editing, deleting, listing and filtering.

- **Adding:** This will be for adding the information on the book to be used as a listing later. Examples of the data added would be `book_title`, `book_price` and `book_type`.
- **Editing:** This will be used for editing existing records. This would be used in instances where the price or book title would have to be changed.
- **Deleting:** An instance of this would be if a book is no longer available on our site and would apply to all variables in the table on that book.
- **Listing:** This will be used to show the books and their information within the record, this will be used mainly within the search engine but also within the general layout of the website when a selection of books will be shown to the user.
- **Filtering:** This will be used mainly through the backend system for system admins to find specific books for editing, deleting or adding.

My table will be heavily linked with the Publisher and Author Table as the books would need to be linked to their individual author/publisher. This will also link inn with the front end of the website for the search engine.

Book Schema

Attribute	Type (`DataType`)	Key
book_id	Integer `int`	Primary Key
author_id	Integer `int`	Foreign Key
book_title	String `varchar(50)`	
book_price	Float `float`	
book_type	String `varchar(15)`	
book_genre	String `varchar(15)`	
book_type	String `varchar(15)`	
book_pubdate	Date `date`	
book_first_ed	Boolean `bit`	

The `book_id` variable will be changing constantly when a new book is added automatically allowing for easy organization of the Book Management System and will help all backend and front end systems in finding a specific book.

Author & Publisher System

This system will be run by Hugh, who will be creating the following functionality:

- **Frontend author/publisher tables:** these pages will consist of information stored about publishers and authors. The user will be able to access data on an author, or a publisher, which will be displayed using the respective `AuthorID` or `PublisherID`. On an author page there will be a short description of the author along with a date of birth and a list of all other books related to them. The publisher page will have a similar design.
- **Adding an author/publisher:** authors and publisher will be added using the backend form. This form will consist of the same fields that the front end provides. Information included on an author will be name, date of birth, whether or not they are deceased and a short description of them. Publisher name and website will be asked for.
- **Editing an author/publisher:** admins will be able to find author and publisher records and edit them as needed. Users will not have this functionality
- **Deleting an author/publisher:** an admin will be able to search and find an author or publisher record and delete it. Users will not have this functionality.
- **Listing an authors/publishers or a single author/publisher:** admins will be able to list all records of authors or publishers or a specific one at the backend, including fields hidden from users. Users will be able to find information available to them by using the search functionality.
- **Filtering authors/publishers:** admins will be able to filter author/publisher records at the backend to find records they wish to see. For example, they may wish to only filter via deceased authors. Users will be able to use this functionality on the front end.

Author & Publisher Schema

Below is a table schema for both the author and publisher tables. Additions to these tables will be made in the backend.

Table 15: Author Schema Table

Attribute	Type (`DataType`)	Key
author_id	Integer `int`	Primary Key
book_id	String `varchar(255)`	Foreign Key
author_name	String `varchar(50)`	
author_dob	Date `date`	
author_isalive	Boolean `bit`	

Table 16: Publisher Schema Table

Attribute	Type (`DataType`)	Key
publisher_id	Integer `int`	Primary Key
author_id	Integer `int`	Foreign Key
book_id	Integer `int`	Foreign Key
publisher_name	String `varchar(50)`	
publisher_datefounded	Date `date`	
publisher_website	String `varchar(255)`	

Order & Record System

This system will be run by Emmanuel, creating the following functionality:

1. Frontend user order/record table `My Orders`. This page will consist of pulled previous-order information related to the particular `UserID` attempting to access the order records and displays this for each user. The information will be accessed from the larger order-record table which holds the orders of all customers. Each order will have a download button beside it, allowing the user to download the particular book related to the `order_id`.
2. Backend order addition, deletion, filtration and modification, listing and validation.
3. Frontend user order confirmation. This page will provide each user with an order-number (`order_id`) and itinerary confirming the details of the book they have purchased and redirection link to the `My Orders` page where they can download their book.

Order & Record Schema

On the table below, the template for each user order on the `My Orders` is shown.

On the user order page, each customer will have the opportunity to enter an integer between 1-5 representing their level of satisfaction of the order. Helping gauge the popularity of each book, particularly in identifying the sites best sellers.

Table 17: Order & Record Schema Table

Attribute	Type (`DataType`)	Key
order_id	Integer `int`	Primary Key
user_id	String `varchar(25)`	Foreign Key
book_id	String `varchar(255)`	Foreign Key
order_date_of_purchase	Date `date`	
order_is_claimed	Boolean `bit`	
order_satisfaction	Integer `int`	

Simultaneous to every additional order in the system, there will be an increment in the `order_id` value.

There is a `one to zero or many` and `one to one (mandatory)` relationship between `user_id` and `order_id`.