Venkatesh Srinivasan

Cognizant technology solution  India

Domain driven design solution

Contents

[Domain – The problem to solve 2](#_Toc142811513)

[ubiquitious Language – Glosarry for all communications 2](#_Toc142811514)

[Sub domains – sub Probelem to solve 3](#_Toc142811515)

[Item catalogue 3](#_Toc142811516)

[Bounded Context 3](#_Toc142811517)

[AUthors bounded context 4](#_Toc142811518)

[user rating bound context 4](#_Toc142811519)

[Payment gateway integration 5](#_Toc142811520)

[User registration 5](#_Toc142811521)

[user Registration bounded context 5](#_Toc142811522)

[Authentication 6](#_Toc142811523)

[App Notification 6](#_Toc142811524)

[Web notification 6](#_Toc142811525)

[Log Analytics 6](#_Toc142811526)

[Pushtaka Application 7](#_Toc142811527)

[Reader Bounded context 7](#_Toc142811528)

[Library bounded context 7](#_Toc142811529)

[Annotation Bounded Context 8](#_Toc142811530)

[Reader preferences Bounded Context 8](#_Toc142811531)

[Shopping cart – Sub domain 8](#_Toc142811532)

[Discount Bounded Context 9](#_Toc142811533)

[Shopping cart bounded context 10](#_Toc142811534)

[Favorites Sub domain 10](#_Toc142811535)

[Favorites Bound context 11](#_Toc142811536)

[NON-FUNCTIONAL sub domain 12](#_Toc142811537)

[Repository Details 12](#_Toc142811538)

Domain driven design solution

# Domain – The problem to solve

The domain under consideration is an e-shopping application that already existing and is a monolith. On analysis of the *existing application* has the following functionality

1. Item catalogue – list of books and audio eBooks and audio books
2. Payment gateway integration
3. User registration
4. Authentication using user id and password.
5. App notification
6. Web notification
7. Log Analytics
8. Pustaka Mobile application reader
9. Shopping cart

On further discussion with the user experience of Pustaka mobile application, he wants the following feature enhanced.

1. Resume reading from where last left termed as “Auto bookmark.”
2. Resume hearing for audio books
3. Ability to store “user markings” like “comments and highlights” in the application.

On further discussion with the backend application user

1. Need for handling burst load cost effectively.
2. **Author biography** to be added.
3. **User rating** displayed and allowed user to rate book.
4. **Email Reminder** on pending shopping cart items
5. **Interest expressed (favorites)** tracker. That is sometimes the user might say he is interested in a book but may not buy it because of cost. So, when the cost becomes lesser, we need to let the user know about it. Also, if there are any special discount sales they need to inform.
6. **Multi factor authentication.**

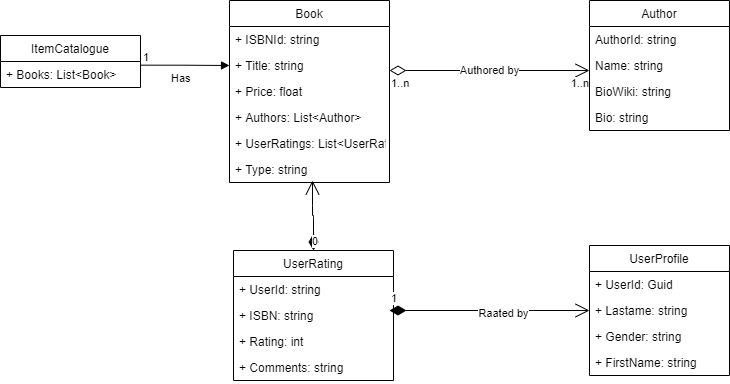
# ubiquitious Language – Glosarry for all communications

|  |  |
| --- | --- |
| **Ubiquitous Terms** | **Explanation** |
| Catalogue of Items | These refers to audio book sand eBooks that can be purchased by the app |
| Pustaka App | The mobile app |
| Shopping kart | The shopping car to which the user adds his purchases before check out |
| Payments | The payments made by user using payment gateway |
| User profile | The profile of registered user like email, gender, date of birth, preferences, navigation preferences |
| Web app | The browser-based app that has similar functionality as that of Pustaka app |
| Authentication | The process of validating the user |
| Web notification | Those email that are sent out of any business process |
| App Notification | Mobile app notification |
| Auto book mar | To store the last read book or audio book and resume from there when launched next time |
| User markings | The highlights or comments that are added by user to his copy of book in Pustaka reader |
| Reader | The Pustaka reader interface both web and mobile |
| Author Bio | The Biography of the author that needs to be entered manually or screen scrapped from external site. |
| User rating | The comments the user write about the book accompanied by star rating |
| Loyalty points | Price discounts given on purchase of user associated with business for long time or have purchased lots of items or Beta copy of book |
| Analytics | Insights from logs |
| Favorites aka Interest Expressed | These are books that the user wanted to buy but have not bought for quite some time |
| Books | Refers to both Audio and eBooks |
| Multi factor authentication | Typical multi factor app with additional sign on steps to avoid false impersonation |

# Sub domains – sub Probelem to solve

## Item catalogue

Item catalogue refers to the list of items (Books) that the user can purchase. It is an aggregate type. The various bounded contexts in that and their relationships are shown below.



### Bounded Context

Book is bounded context. It refers to both audio and video books. It also has an aggregate of Authors. This itself qualifies for a micro service.

#### Micro service – Book

The following are the end points and functionality in Book Micro service

|  |  |
| --- | --- |
| End point | functionality |
| AddBooks | Adds a book |
| EditBook | The title and price of the book alone can be edited |
| Removebook | Removes the book |
| Get | Gets all books |
| GetBookById | Get book by ID |

The swagger json is attached here with which gives the



### AUthors bounded context

The next bounded context that we are considering is Authors. It is composed ( a sub type of aggregation as per UML terminology) in books bounded context. That is a Author has to be associated with a book to get its meaning and books cannot exist without Authors.

#### Microservices Author

|  |  |
| --- | --- |
| End point | functionality |
| AddAuthors | Adds a Author |
| UpdateBio | Updates the Bio or biowiki of the author |
| DeleteAuthor | Removes the Author |
| GetAllAuthors | Gets all authors |
| GetById | Get Author by ID |

To provide the layout of end point and parameters the swagger.json is attached to here with



### user rating bound context

User rating as o to many compositions’ relationship with books. Here the user can only rate the book once. In case the user rates the book more than once, the latest rating will be considered.

#### Microservices user rating

|  |  |
| --- | --- |
| End point | functionality |
| GetAllRatings | Get All Ratings |
| GetById | Get the rating by user and ISBN id |
| RateBook | Rate a book by ISBN ID |

To provide the layout of end point and parameters the swagger.json is attached to here with



## Payment gateway integration

The payment gateway integration is an important part of the problem domain. It helps to process the payment and returns whether the payment was successful or not. Though it has important functionality, it is a **supporting domain** and can be integrated with SaaS platforms.

## User registration

The user registration holds the user profile for the site. It is a **core domain** to solve. It has an association relationship with the authentication service. The claim (typically a JWT) provides the user first name, last name, mobile, email address. Additional information captured is used to drive user specific targeted campaigns or user group targeted campaigns like discounts and suggestions based on age group. Example, a particular age group and ethnicity would prefer religious books from a famous Guru or evangelizer. They can also help to generate meaningful analytics on preferences. In the below diagram the Authentication is not elaborated but the user profile details are shown. The Authentication sub domain is shown just for relationship purposes only.

A diagram of a data flow

Description automatically generated

### user Registration bounded context

This is a single bounded context and has a simple uses association relationship with authentication.

#### MIcroservices User registration

|  |  |
| --- | --- |
| End point | functionality |
| GetAllUsers | Get All users |
| GetUserByID | Get all users by id |
| AddUser | Add User Id |
| EditUser | Edit user |
| DeleteUser | Delete the user |

The layout of microservices can be seen attached swagger JSON



## Authentication

Authentication is a **supporting** domain for this domain. Though legacy functionality has handled it in a custom way, the recommended and secured way of handling this is use the out of the box services provided by Hyper scalers like AWS or Azure. The advantages of using such a service are as below.

1. No need to store/ manage passwords.
2. Support for Multifactor authentication
3. Sign in using various domains like Google, Facebook, and live id.
4. Single sign on using claim-based authentication in case of Microservice based distributed architecture.
5. Expiration based authentication tickets.

Since the service is hosted in AWS, AWS cognition is recommended.

## App Notification

App notification is one of the desired functionalities in this problem domain. These are notifications that are set to the reader app. It is well known that this this capability exists in most of the hyper scalers. The ideal way to do this feature is to integrate with existing services provided by the hyper scalers like AWS or Azure. Since this application is hosted in AWS. The best way to achieve this is to Use the AWS SNS service Available out-of-the-box. This is a **supporting** domain.

## Web notification

Web notifications refers to those notifications that are sent via e-mail. This could be anything like confirming your e-mail or it could be something like a promotion for books or it could be something related to the items pending in the shopping cart. This is a **supporting** domain. The best way to do this is to integrate with an AWS e-mail service which is provided out-of-the-box in AWS.

## Log Analytics

Log Analytics is a **generic** domain. It is used to analyze the logs emitted by the applications of various microservices. Hence out of the broad out-of-the-box products like a AWS cloud watch can be used to achieve the same. Alerts are added to inform the support staff in case of critical events.

## Pushtaka Application

The Pustaka application is a sub domain that is a core domain that needs to be built. The overall bounded context is depicted in the diagram below.

A screenshot of a computer

Description automatically generated

### Reader Bounded context

The reader is the Pustaka reader. It is available in two forms one is the mobile reader as well as the web reader. For the audio and Video Blazored Audio and video component and for eBook Blazor reader can be used. The reader should do the following.

1. Metaphor for reading/hearing the book.
2. Adding comment to pages or gearing position
3. Auto bookmark. Resume from last read/heard position.
4. Add bookmarks by users.
5. Managing highlights in eBook

### Library bounded context

After the user purchases the books, a copy of the book is stored in Blob storage under the user directory. When the users use the Pustaka reader, the list of books bought by the user is shown. Then when users choose the URI is returned. This URI is used by the reader to stream the audio or PDF reader.

#### Microservices Library

|  |  |
| --- | --- |
| End point | functionality |
| GetAllPurchasedBooks | Get All purchased books |
| GetBookURI | Get the book URI based on ISBN id. Used by reader |
| AddToLibrary | Add books to library. Called by shopping cart |

### Annotation Bounded Context

This is a **core** domain that needs to be solved. Users when reading or hearing the book will write comments, perform highlights or book mark a page or audio segment. Apart from that there should be the option for users to resume reading or hearing from where they left. All these are possibly made possible by annotation. The Reader will be using this service to persist these aspects and load them on page-to-page basis in case of eBook or in case of audio stream load it at start.

#### Microservices

|  |  |
| --- | --- |
| End point | functionality |
| GetAnnotationByBookId | Get all the annotation for books. Mostly for use by Audio books. |
| GetAnnotationByBookPage | Page wise book annotations |
| UpsertAnnotation | Insert or update annotations |
| DeleteAnnotation | Remove annotation |

### Reader preferences Bounded Context

This is a **core** domain that needs to be solved. These are certain preferences used by the reader. The preferences include setting like Night mode etc.: -

#### microservices

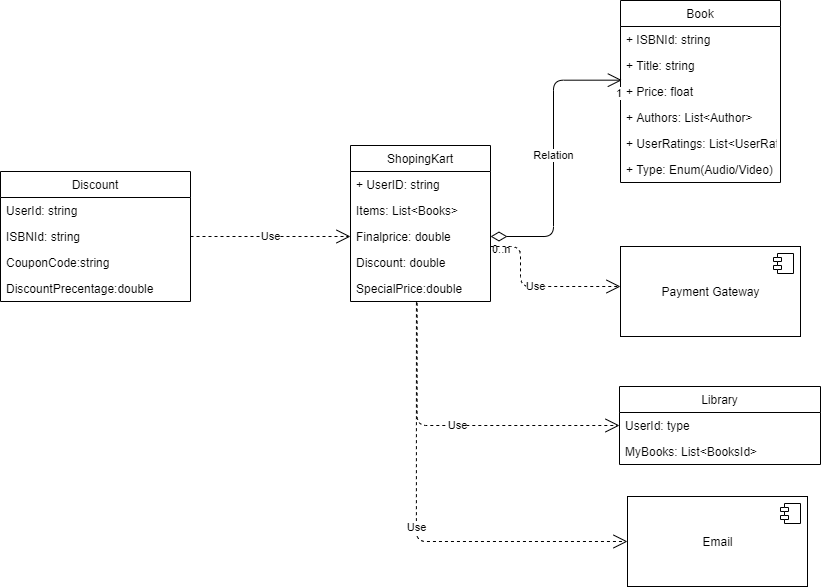
|  |  |
| --- | --- |
| End point | functionality |
| Get | Gets the preferences for reader. In special preferences is found then default preferences is returned |
| Update | Update the preference by user |

## Shopping cart – Sub domain

Shipping is another core domain that we need to solve. The users can keep shopping for the books from the item catalogue. Once that the complete the users can proceed to check out. At that time any discount codes that are shared to the user are applied. Also, if there are any special prices for a new book because of the new introduction then that is also added as a special price. After the pricing is arrived on clicking the verify and pay, the shopping Kart domain re-directs to integrated Payment gateway which is an external generic domain. If the payment is successful, shopping kart invokes Library domain, that will get copies of the book that the user wanted and store them in S3.

If the user has left anything unattended in the cart, a reminder is sent to him every 10 days for a month. After that it will be removed from the cart.

Below is the layout in the form of class diagram.



### Discount Bounded Context

This bounded context is more for maintaining and applying the discount for Shopping cart. There are end points for adding, editing, or removing the discount coupons which are used by Admin. The Get portions are used by the shopping kart. There are two ways it happens.

1. There are multiple discount coupons the user has, and he will provide the code.
2. The second is there could be goodwill discounts or special pricing for items present in the item catalogue. Those are applied to the item of finalizing the price.

#### Microservice Discount

|  |  |
| --- | --- |
| End point | functionality |
| GetByBook | Gets specific discount by book |
| GetByUser | Gets the discount at user level for loyally points or surprise offer |
| GetByPurchase | Get user by items in the cart |
| Add | Add a discount coupon |
| Delete | Deletes the discount code |
| Edit | Edit discount code |

### Shopping cart bounded context

Shopping cart is a core bounded context. As when the user browses the item catalogue, they would need to add the item to shopping cart. Once the shopping is complete, the user checks out the cart. In this process the final price by updating the discount has arrived at. Then payment gateway is invoked. There could be multiple payment gateways. Once the payment is successfully completed the shopped items are added to the library.

#### Microservices Shopping cart

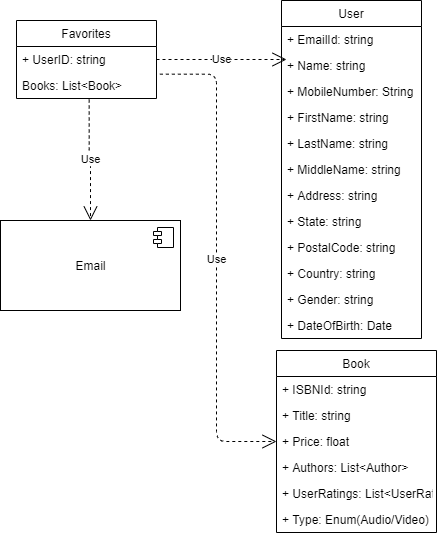
|  |  |
| --- | --- |
| End point | functionality |
| GetByUserId | Gets the shopping cart for a user |
| AddItems | Add Item to cart |
| GetByPurchase | Get items in the cart |
| EditCart | Modify items in cart |
| RemoveItem | Remove single item from cart |
| EmptyCart | Remove all items from cart |
| FinalPrice | Computes the final price |
| DoPayment | Invokes the payment gateway and receive response |
| AddToLibrary | Add items to library |

The swagger json for the same is attached herewith.



## Favorites Sub domain

Favorites is another **core** domain that needs to be handled. Users sometimes “express interest” over some books without buying it. Up to 50 such interests can be tracked. These are treated as user favorites. Whenever there is a new release in that book/upgrade or sale these are communicated to users continuously. Sometimes old books would be archived. This is true especially with books in the technology category. So, when old technology gets outdated the sales drop and they get removed. If a user has had that book in his favorites, then it should be communicated to him before archiving. The diagrammatic representation of the same is given below.



### Favorites Bound context

Favorites itself is a self-contained bounded context. The input for the preferences comes from the item catalogue. Once the user wishes to track the book it is added to his preferences.

#### Microservices Favorites

|  |  |
| --- | --- |
| End point | functionality |
| GetByUserId | Gets the favorite for a user |
| GetByBookId | Get all the favorites for a book |
| Entice | For a user and for a book send some targeted updates |
| AddToFavorites | Add a favorite |
| RemoveItem | Removes from Favorite |

To provide the layout of microservices the swagger json is provided below. 

## NON-FUNCTIONAL sub domain

There are a few nonfunctional domains that are required. These are generally catered by standard service available by the cloud provider or patterns that solve a technical problem. These are not visible to the user.

1. Cron Jobs: These are some periodic jobs that does altering, archival and emailing of promotions. Not much technical design will be required except the link to content that needs to be emailed, alerted, or archived.
2. Anti-corruption Layer: - This is a technical piece for de-composing monoliths to micro services. There is a no general design but the pattern emphasis that the existing bigger application is not broken because of carving out the micro service from out. These are achieved using Adapter design pattern.
3. Event driven components: - In order make the architecture scalable event driven architecture will be implemented. This comes in two parts. One is the event management component like RabbitMQ, Kafka etc., the other part is the event producers and consumers. While the event producers and consumers will be coded as part of micro services as part of the communication strategy, the event management sub domain itself can be a generic domain. Any Queue s/w based on the need can be chosen. For the shopping cart, it is better to use Amazon SQS. Instead of directly binding the producers and consumers directly to cloud providers’ queueing platform it is suggested that intermediaries like Dapper.IO be used. This will avoid strong binding to one cloud provider there by making the application more portable across cloud providers.

# Repository Details

Source code repo :- [VenkateshSrini/DomainDrivenDesign (github.com)](https://github.com/VenkateshSrini/DomainDrivenDesign)

Design documentation link:- [DomainDrivenDesign/DDD-Solution.docx at main · VenkateshSrini/DomainDrivenDesign (github.com)](https://github.com/VenkateshSrini/DomainDrivenDesign/blob/main/DDD-Solution.docx)