IIHT

DIGITAL BOOKS

Table of Contents

[1. Problem Statement 3](#__RefHeading___Toc639_3426528727)

[2. Wireframes 4](#__RefHeading___Toc641_3426528727)

[3. Application architecture 5](#__RefHeading___Toc643_3426528727)

[Possible rest clients 5](#__RefHeading___Toc675_3426528727)

[4. Cloud Architecture 6](#__RefHeading___Toc645_3426528727)

[5. Tool Chain 7](#__RefHeading___Toc647_3426528727)

[6. Business Requirements 8](#__RefHeading___Toc649_3426528727)

[7. Proposed Rest Endpoints 9](#__RefHeading___Toc651_3426528727)

[8. Key Rubrics (Expected Deliverables) 10](#__RefHeading___Toc653_3426528727)

[A. As an application developer: 10](#__RefHeading___Toc655_3426528727)

[B. Debugging & Troubleshooting 10](#__RefHeading___Toc657_3426528727)

[C. Code Quality/Optimizations 10](#__RefHeading___Toc659_3426528727)

[9. Platform 11](#__RefHeading___Toc661_3426528727)

[10. Methodology: Agile 12](#__RefHeading___Toc663_3426528727)

# 1. Problem Statement

Build **D****igitalBooks** app which takes traditional books a step further, combining text with visual and audio elements to make authors' publications truly multimodal. Authors can write down their thoughts and assemble a collection of original or curated content ranging from photos, drawings, and images to audio and video clips -- in some cases, even animated text.

And while **D****igital Book** app can magically use images of autobiographical presentations or fantastical tales which are certainly options. It is also important to think beyond personal narratives to how authors might share the "stories/experiences" of their learning on any topic. And beyond author presentations and publications, plenty of students, teachers, doctors, engineers can jump on board, to create dynamic books and presentations that serve as instructional tools.

Build digital book app. Below are the different roles, which need to be supported by this software system.

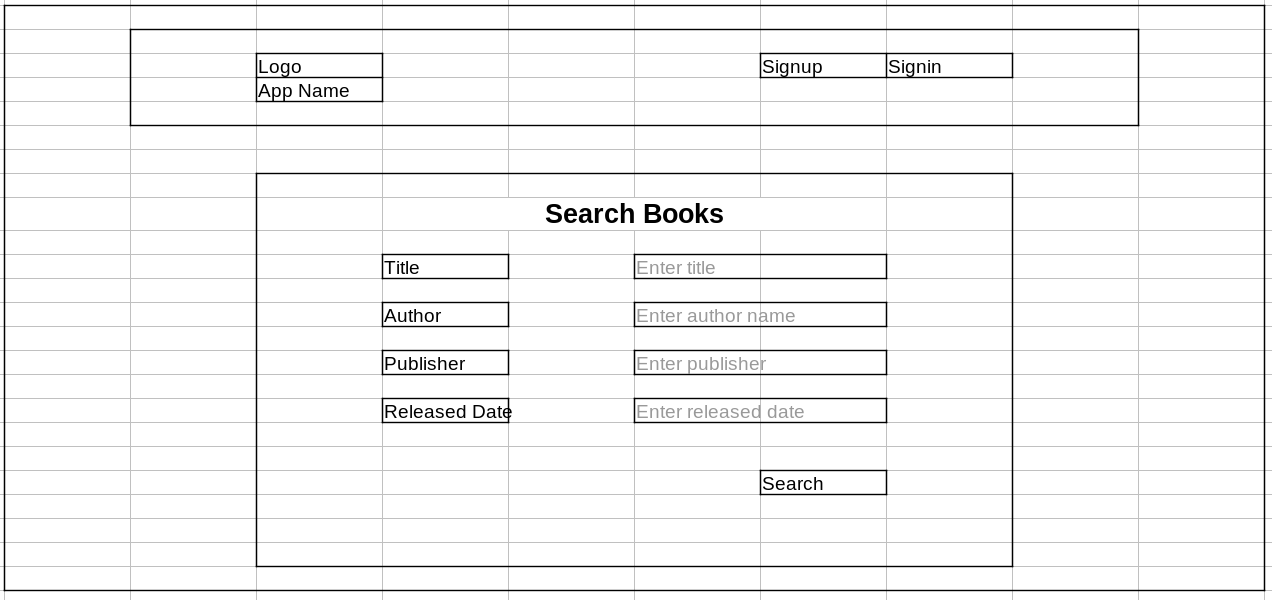
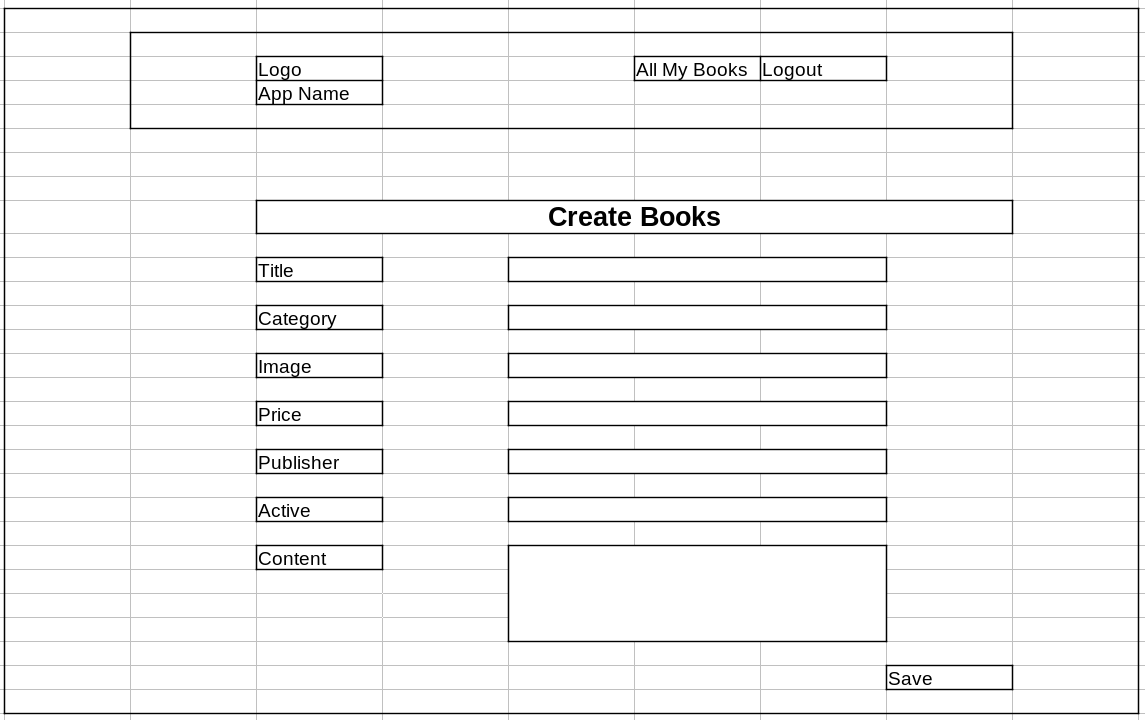
1. Author
2. Reader

Microservices: AuthorService, ReaderService, BookService → ignore table mapping

The scope includes developing the application using tool chain mentioned below.

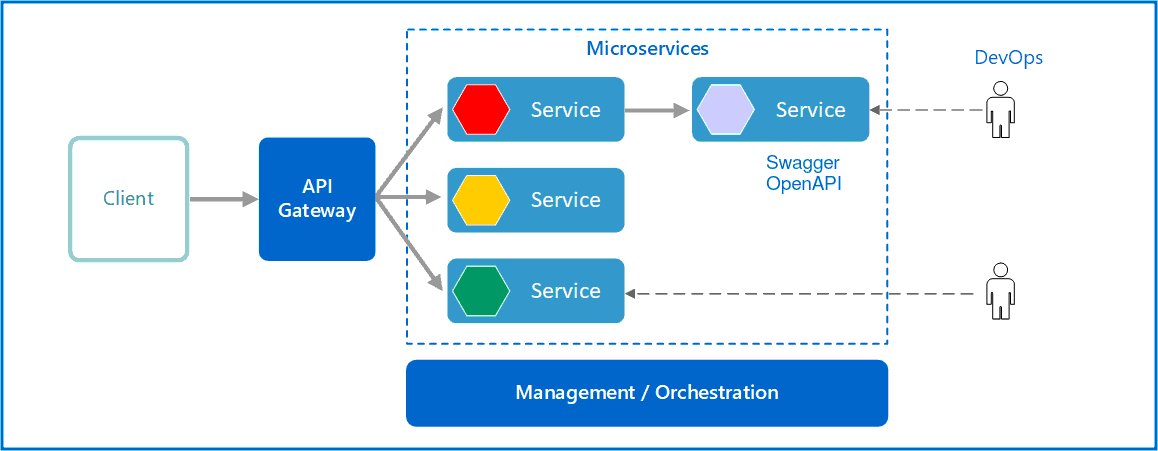
# 2. Wireframes

UI needs improvisation and modification as per given use case.



# 3. Application architecture

Compute and Integration/Presentation/Networking and Content Delivery



## Possible rest clients

We will use below **clients** for our microservice applications

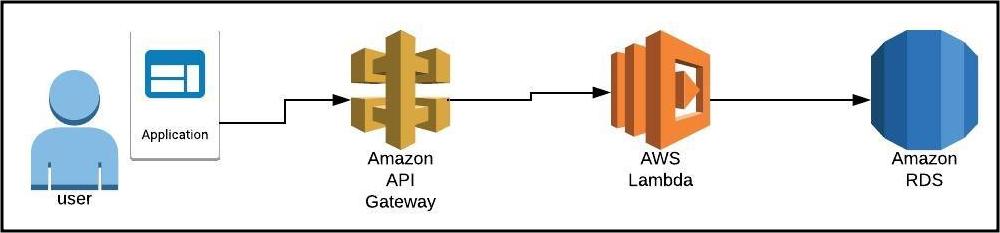
* Postman
* Postwoman (hoppscotch.io)
* Swagger
* Rest Client - plugin
* Angular app
* Java application

Any client from below list can consume our microservice (we will not use them):

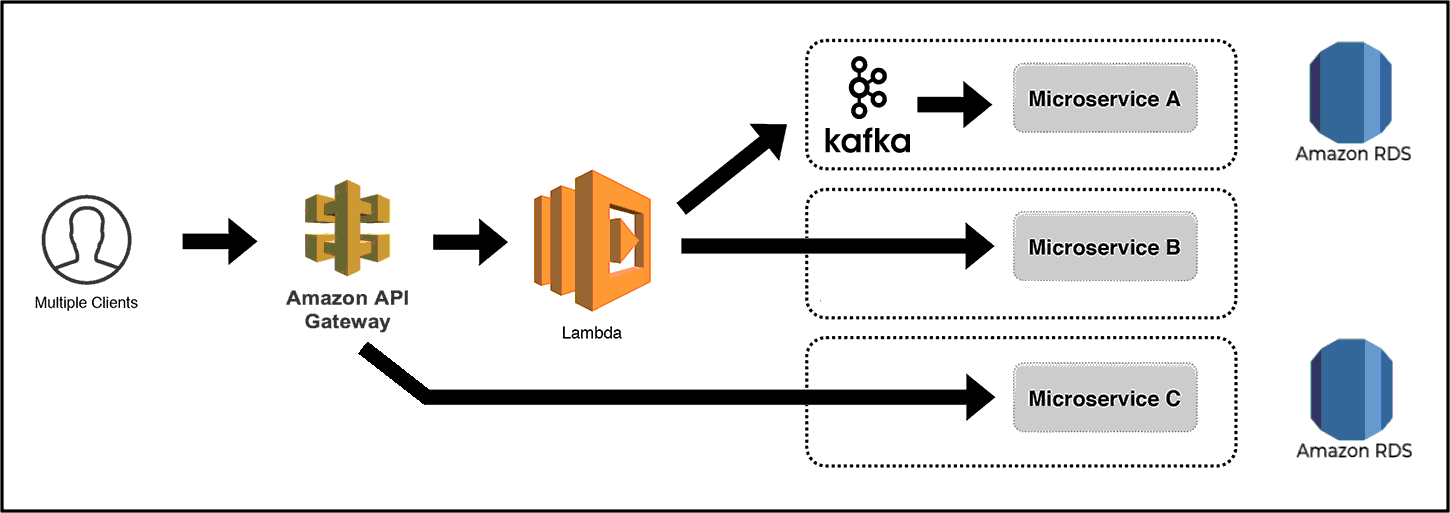
* React app
* Jmeter
* Android app
* Ios app
* .Net application
* python application
* many more…

# 4. Cloud Architecture

1. Api gateway → Lambda → RDS



1. Api gateway → Lambda → Kafka → EC2 → RDS
2. Api gateway → Lambda → EC2 → RDS
3. Api gateway → Lambda → EC2
4. Api gateway → EC2 → RDS



# 5. Tool Chain

|  |  |  |
| --- | --- | --- |
| **Competency** | **Skill** | **Skill Detail** |
| Engineering Mindset | Networking and Content Delivery |  |
|  | DevOps |  |
|  | Secure Coding | Veracode |
|  | Code Quality | Sonar |
| Programming Languages | Application Language | Java |
| Products & Frameworks | Presentation | Angular |
|  |  | Material/Bootstrap, rxjs, ngrx/store |
|  |  | Javascript/Typescript |
|  | Networking and Content Delivery | Zuul (SpringCloud) |
|  | Security and Identity | OpenIAM |
|  | Compute & Integration | Spring Boot |
|  |  | Kafka |
|  | Database & Storage | MySQL |
|  | Governance & Tooling | Git |
|  |  | Junit |
|  |  | Mockito |
| Engineering Quality |  |  |
| Platform | Cloud Tools | AWS EC2 |
|  |  | RDS-MySQL/Aurora |
|  |  | AWS Lambda |
|  |  | AWS API Gateway |
|  |  | AWS ELB(Elastic Load Balancer) |
|  |  | AWS CloudWatch |

# **6. Business Requirements**

As an application developer, develop microservices with below guidelines:

|  |  |  |
| --- | --- | --- |
| User  Story # | User Story Name | User Story |
| US\_01 | Reader Mode | 1. Reader should not create account. 2. Reader can search for books based on **title, category, author, price** 3. Each search result need to display book logo, title, author, publisher, price, published date, category. 4. From search results, reader should be able to select a specific book and go ahead to complete purchase/subscription by providing **Name and Email ID** 5. On successful purchase, unique **payment\_id/subscription\_id** need to be generated. **Payment/Subscription should be taken as a guest user only.** 6. Reader should be able to download invoice. 7. Reader can read only purchased/subscribed books. 8. Reader should be able to  * view history of all previous payments/subscriptions * view invoice using emailId or payment\_id/subscription\_id * cancel subscription within 24 hrs of payment/subscriptionis done |
| US\_02 | Author Mode | 1. Author shall be able to create account. 2. Author shall be able to login/logout. 3. There can be pre-defined username/password for Author. 4. Author shall be able to add/edit book. For example    1. **logo**: image    2. **title**: Ironman is back    3. **category**: comic    4. **price**: 24    5. **author**: {current user name}    6. **publisher**: Moon publisher    7. **published date**: 22/04/2022    8. **chapters/content**    9. **active:** true 5. Author shall be able to block and unblock an book. When book is blocked    1. Books will not be shown in Search results.    2. Readers (who have purchased book) should get notification about the unavailability of book. |

# **7. Proposed Rest Endpoints**

If you think rest endpoints need improvisation and modification as per given use case, you can make necessary changes.

|  |  |  |
| --- | --- | --- |
| **GET** | /api/v1/digitalbooks/books/search?category&author&price&publisher | reader can search books |
| **POST** | /api/v1/digitalbooks/books/buy  payload: {bookId, reader {name, email}} | reader can buy book |
| **GET** | /api/v1/digitalbooks/readers/{emailId}/books | reader can find all purchased books |
| **GET** | /api/v1/digitalbooks/readers/{emailId}/books/{bookId} | reader can read book |
| **POST** | /api/v1/digitalbooks/readers/{emailId}/books?pid | reader can find purchased book by paymentId |
| **POST** | /api/v1/digitalbooks/readers/{emailId}/books/{bookId}/refund | reader can ask for refund within 24 hrs of payment |
| **POST** | /api/v1/digitalbooks/author/signup | author can create account |
| **POST** | /api/v1/digitalbooks/author/login | author can login |
| **POST** | /api/v1/digitalbooks/author/{authorId}/books | author can create book |
| **PUT** | /api/v1/digitalbooks/author/{authorId}/books/{bookId} | author can edit/block/unblock hist book |

# **8. Key Rubrics (Expected Deliverables)**

## A. As an application developer:

* + 1. Develop the application as a microservice architecture.
    2. Ensure package Structure for project is like **com.****digitalbooks.\*** with proper naming conventions for package and beans.
    3. Use **application.properties** or **application.yaml** file to maintain all spring boot config.
    4. Implemented the package structure - Controller, Interface, Service, DAO, Testing, Validation, Security etc
    5. Implementation as follows:
       1. Use Domain Driven Design to implement distributed architecture
       2. Follow Single Data Store per microservice practice
       3. Document REST endpoints with OpenAPI/ Swagger
       4. Add CQRS pattern for Event Sourcing using Kafka for saving and retrieving book details, using Kafka (producer & consumer) topics
       5. Expose all rest Endpoints using a common API Gateway

## B. Debugging & Troubleshooting

* + 1. Generate bug report & error logs - Report must be linked with final deliverables which should also suggest the resolution for the encountered bugs and errors.

## C. Code Quality/Optimizations

* + 1. Associates should write clean code that is readable
    2. Associate should use the Code Analyzer (PMD/SonarQube) to ensure code quality and standard code style.

# 9. Platform

* + Use EC2 to deploy application on cloud.
  + Use AWS RDS service as a database for the Application.
  + Use AWS Lambda and DB to build a backend process for handling requests for Flight booking App.
  + Use Amazon API Gateway to expose the Lambda functions built in the previous step to be accessible on public internet.
  + Use AWS ELB(Network Load Balancer) to configure the load balancing of the instances

**Note** : Minimum 2APIs (UI+Backend) to be hosted in cloud

# 10. Methodology: Agile

* + As an application developer, use project management tool along to update progress as you start implementing solution.
  + As an application developer specify the estimation and planning as a part of Agile process.
  + As an application developer, the scope of discussion with mentor is limited to:
    1. Q/A
    2. New Ideas, New feature implementations and estimation.
    3. Any development related challenges
    4. Skill Gaps
    5. Any other pointers key to UI/UX and Middleware Development