Book Alley

Software Architecture Document

Version 2.2

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 24/11/2023 | 1.0 | Introduction, architectural goals and constraints, use-case model | Book Alley Team |
| 28/11/2023 | 1.1 | Logical view | Book Alley Team |
| 2/12/2023 | 1.2 | Review | Book Alley Team |
| 6/12/2023 | 2.0 | Add deployment diagram | Tran Minh Hai Uyen |
| 12/12/2023 | 2.1 | Section 5 and Add Folder structure | Tran Minh Hai Uyen |
| 13/12/2023 | 2.2 | Fix the class diagrams (Section 4) and add description for Section 6 | Book Alley Team |

Table of contents

[**1. INTRODUCTION 3**](#_heading=h.9f0rc46ej9l8)

[1.1. Purpose 3](#_heading=h.dhq1n75dnirl)

[1.2. Scope 3](#_heading=h.vbss6oqho83e)

[1.3. Definitions, Acronyms and Abbreviations 3](#_heading=h.brbw4ju5186m)

[1.4. References 3](#_heading=h.via5dehtf5gi)

[1.5. Overview 3](#_heading=h.eeszghfjblzf)

[**2. ARCHITECTURAL GOALS AND CONSTRAINTS 4**](#_heading=h.sw9le6u4tr1w)

[2.1. Architectural goals: 4](#_heading=h.jwx3pumirafc)

[2.2. Architectural constraints: 4](#_heading=h.p73d9n9z90ty)

[**3. USE-CASE MODEL 5**](#_heading=h.ep6kbn1cng3g)

[**4. LOGICAL VIEW 6**](#_heading=h.mqwhawv7j9yt)

[4.1. Component: Customer 7](#_heading=h.3mow6wleud3d)

[4.2. Component: Seller 10](#_heading=h.gbhu5z42m1yy)

[4.3. Component: Admin 11](#_heading=h.yd1d8pf73ytl)

[**5. DEPLOYMENT 11**](#_heading=h.e6dveulhn1ks)

[**6. IMPLEMENTATION VIEW 11**](#_heading=h.ftn6lwx0goz7)

# INTRODUCTION

## Purpose

This document outlines the software architecture for Book Alley project - a website selling books, utilizing the Model-View-Controller (MVC) pattern. The primary objective of this architecture is to establish a robust, scalable, and maintainable foundation for the website.

## Scope

This architecture encompasses the overall design of the website, including the separation of concerns, component interactions, and data flow. It covers the application logic, user interface, and database management aspects.

## Definitions, Acronyms and Abbreviations

| **NO** | **Acronyms/Abbreviation** | **Definition** |
| --- | --- | --- |
| **1** | **MVC** | Model-View-Controller A design pattern that separates an application into three main components: Model, View, Controller. |
| **2** | **UML** | Unified Modeling Language  A standardized modeling language that helps software developers: Visualize, Construct, Document, Specify, Analyze, Design, Implement. |
| **3** | **SAD** | Software Architecture Document  A document provides a comprehensive overview of the architecture of the software system. |
| **4** | **RBAC** | Role-based access control  An approach to restricting system access to authorized users, and to implementing mandatory access control or discretionary access control. |
| **5** | **UI** | User Interface |
| **6** | **UX** | User Experience |
| **7** | **API** | Application Programming Interface |

## References

* Software Architecture Document - Course Registration System, Version 1.0, 2001, Wylie College IT.
* Use-case specification document.
* Project 6 MVC Class Diagram, Ravindu Jayarathne - <https://creately.com/diagram/example/htz4hikx3/project-6-mvc-class-diagram>

## Overview

The MVC pattern achieves separation of concerns and promotes maintainability. The model handles data manipulation, the view handles data presentation, and the controller coordinates their interactions. This separation simplifies development, testing, and future updates.

# ARCHITECTURAL GOALS AND CONSTRAINTS

## Architectural goals:

**Technical Platform:** The Book Alley project will be deployed onto 1 server

**Security**: The website must keep user’s information confidential such as password, email, phone number, ID, etc.

* The application must implement basic security behaviors:
  + - Authentication: Login using at least an email and a password
    - Authorization: according to their roles, users must be granted or not to perform some specific actions
* For internet access, the following requirements are mandatory
  + - Confidentiality: sensitive data must be encrypted (credit card payments)
    - Data integrity : Data sent across the network cannot be modified by a tier
    - Auditing: Every sensitive action can be logged

**Access Control**: Employ strict access controls to limit user access based on roles and permissions to prevent unauthorized data access.

**Scalability**: Design the system to scale horizontally or vertically to accommodate an increase in users without sacrificing performance.

**Responsive design**: Ensure the website is responsive and accessible across various screen sizes and devices

**Caching strategies**: Implement effective caching mechanisms to improve page loading times and reduce server load.

## Architectural constraints:

* Requires internet access to use the website
* Limited number of concurrent users
* Platform Dependency: The system must run on existing hardware or use a specific software stack.
* Time: The system must be developed within a specific timeframe.
* Understanding and documenting constraints is crucial, as they directly impact design decisions and trade-offs that architects and developers must make.

# USE-CASE MODEL

# 

# LOGICAL VIEW

The flow of this website includes these steps: User will send request to frontend,router will process, when front create an api call to backend, backend will process this request from models, then return the data as json and send this json to view on frontend.

We use MVC as the model for developing this website, and we use these technologíes including ReactJS and Redux Toolkit in the Front-end; Nodejs, Express and MongoDB is used the Back-end (MERN stack)

In Front-end:

* We use Redux Toolkit: a batteries-included toolset for efficient Redux development, which helps manage, update application state and ensure consistency of state.
* Services: to handle data communication between Front-end and Back-end, we need modules to manage API calls, handle asynchronous.
* UI components: ReactJS used for building reusable UI components which improve readability, which helps to maintain larger apps..

In Back-end:

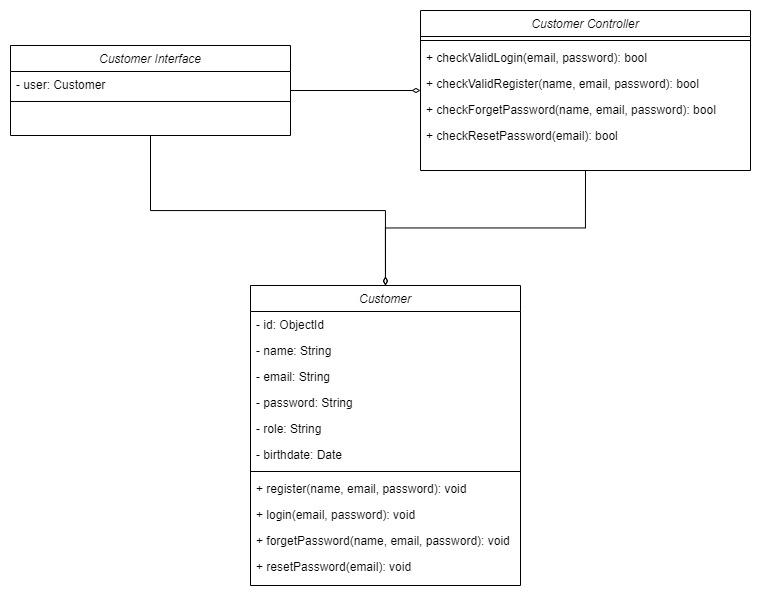
* We use NodeJS as the runtime environment and Express as the web application framework to build RESTful APIs for the front-end to use.
* Custom middleware is used for access control, cors (cross-origin sharing), etc.
* For storing user data, we use MongoDB as our main database, and the mongoose library as the driver.
* For storing images, we use Firebase Storage and save the image’s url in the database, it acts like a CDN (Content Delivery Network) for our web app.

The logical view of BookAlley includes 3 main Components: Customer, Seller, Admin

## Component: Customer

The customer Component shows how the customer communicates with the System.

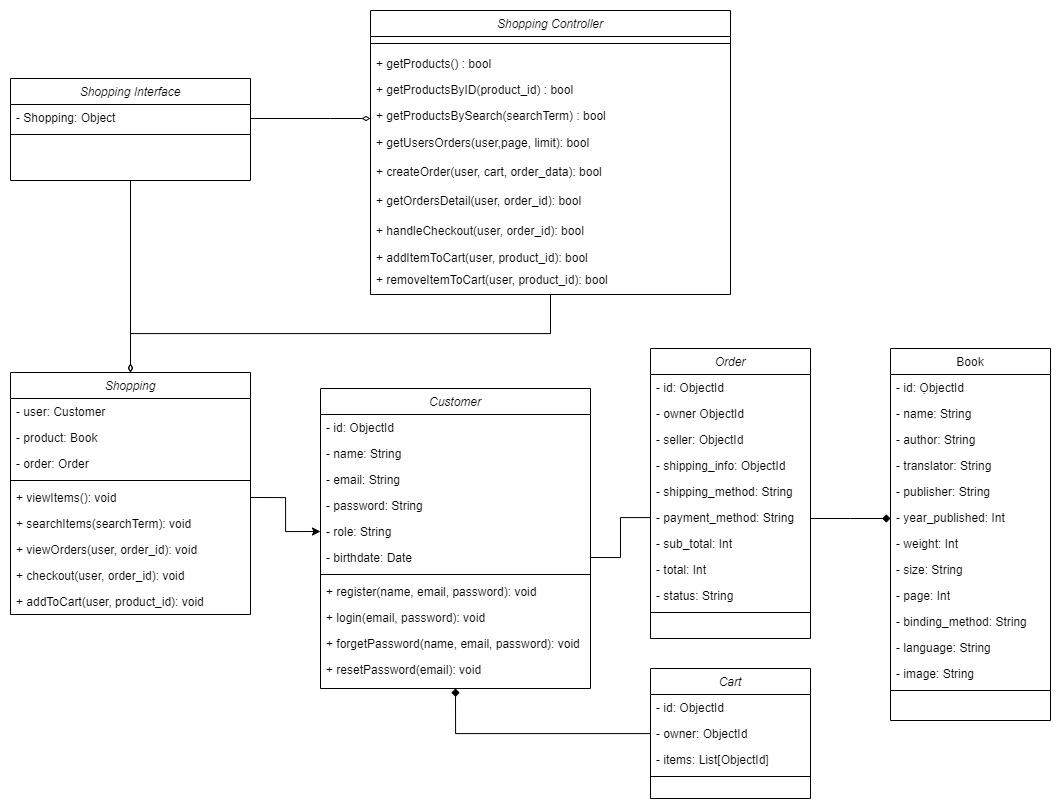
**4.1.1 Component: Authentication**

****

In this Component, there are 4 methods: login, register, forgot password and reset password.

* **Login:** check if the user can access the website. It will call checkValidLogin to check whether the account is valid or not.
* **Register:** check if the user can create a new account. It will call checkValidRegister to check whether a new account has been created successfully.
* **Forgot Password:** check if the user can receive email to create a new password. It will call checkForgetPassword to check whether the user has received the email. Includes a text field for user to enter the email to recover the account
* **Reset Password:** check if the user’s password has been reset. It will call checkForgetPassword to check whether the. Include the field for user to reset the password

**4.1.2 Component: Shopping**

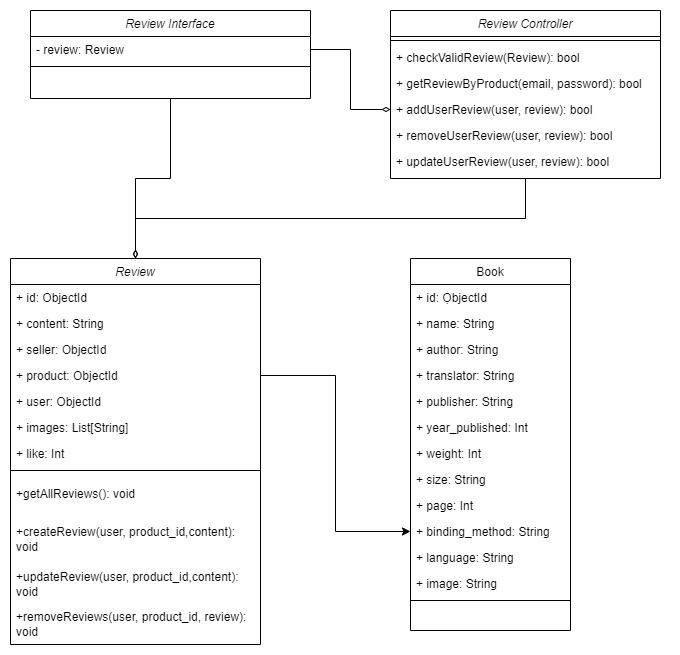
****

The shopping Component has 5 methods: View Item, Search Item, Checkout, View Orders and Add to Cart

* **View Item:** allow the user to view a product or multiple products. It will call getProductsByID and getProducts to check whether the products are valid or not.
* **Search Item:** allows the seller to add a product or multiple products. It will call addProduct to allow the seller to add an item
* **View Orders:** allows the seller to remove a product or multiple products. It will call removeProduct to check whether the products exist ịn the shop or not
* **Checkout:** allows the user to choose the items in the cart to move to checkout and payment methods. This will call the handleChẹckout to check if the user wants to checkout and do payment methods, and removeItemToCart if there are any items chosen
* **Add to cart:** allow the seller to add and remove products to cart. It will call addItemToCart and removeItemToCart to handle add to cart.

**4.1.3 Component: Reviews**

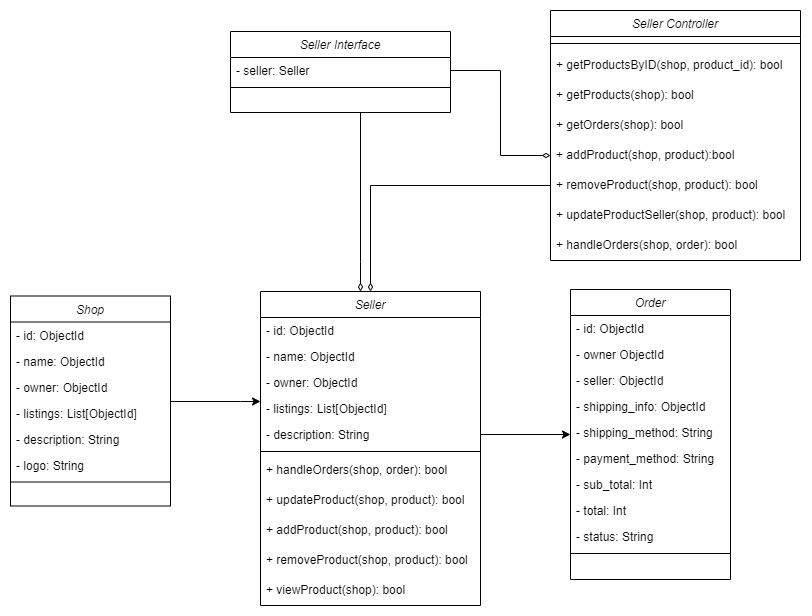
The reviews Component shows how the customer interacts with reviews with the System.

****

Model in review Component includes:

* **Display all reviews:** when users view a product in detail, they can see all reviews of products. It will call getReviewByProduct in the controller to get all product data.
* **Add a review:** after checked out orders, users can add a review. Controller will check if a review is valid or not, then the review will be added to the database.
* **Remove reviews:** users can remove their reviews.
* **Update reviews:** users can update their reviews. Controller will check if a review is valid or not, then the review will be updated to the database.

## Component: Seller

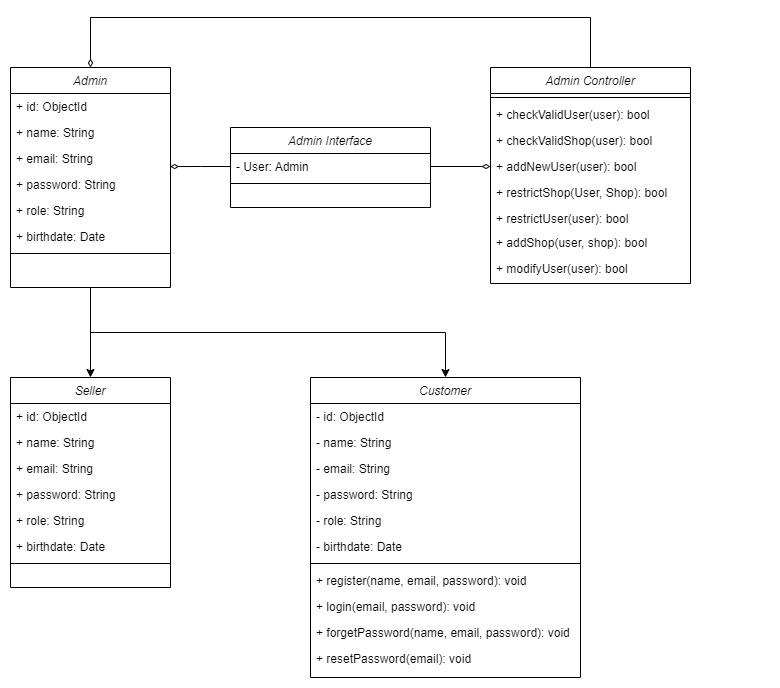
****

The seller Component shows how the customer communicates with the System.

The Seller Component has 5 methods: View Product, Add Product, Remove Product, Update Product and Handle Orders

* **View Product:** allows the seller to view a product or multiple products. It will call getProductsByID and getProducts to check whether the products are valid or not.
* **Add Product:** allows the seller to add a product or multiple products. It will call addProduct to allow the seller to add an item
* **Remove Product:** allows the seller to remove a product or multiple products. It will call removeProduct to check whether the products exist ịn the shop or not
* **Update Product:** allows the seller to remove a product or multiple products. It will call updateProductSeller to check whether the seller want to edit an item the shop or not
* **Handle Orders:** allows the seller to handle orders. It will call handleOrders to check whether the seller want to edit an order or not

## Component: Admin

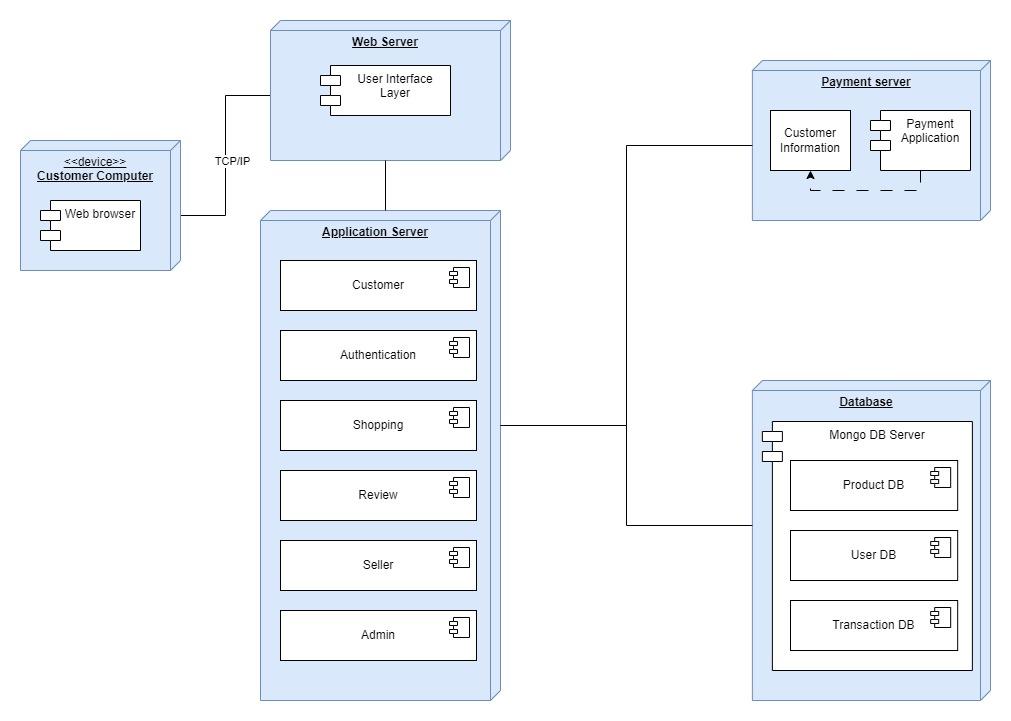
****

The admin Component shows how the admin communicates with the System.

The Admin Component has 5 interfaces: Add User,Add Shop, Restrict User,Restrict Shop, Modify User

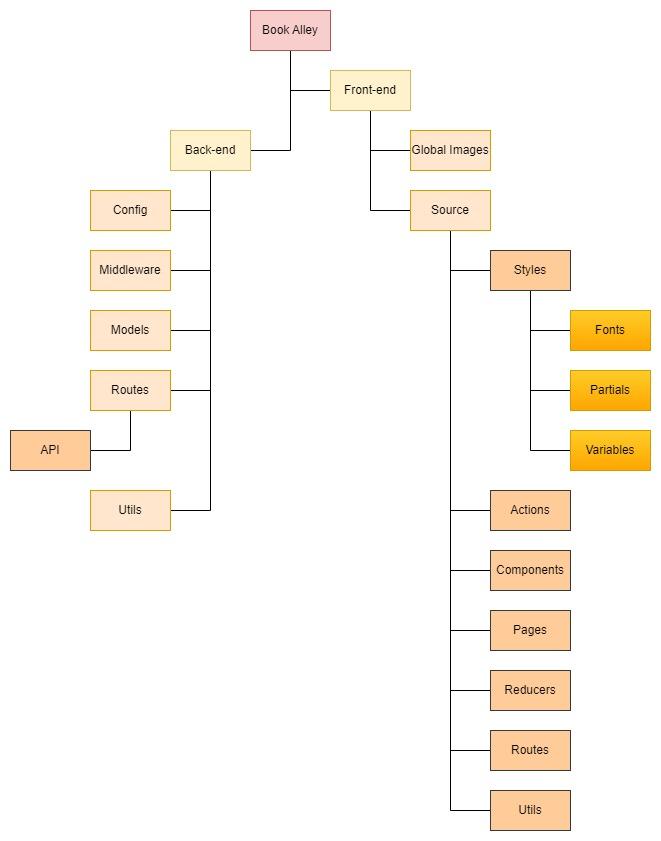
* **Add User:** allows the admin to create a new user. It will call addNewUser to create user and check if this user is valid or not by calling checkValidUser
* **Add Shop:** allows the admin to create a new shop. It will call addNewUser and check if this user is valid and this shop is valid or not by calling checkValidShop
* **Restrict User:** allows the admin to restrict a user. It will call restrictUser and check if this user is valid or not by calling checkValidUser
* **Restrict Shop:** allows the admin to restrict a shop. It will call restrictUser and check if this user is valid and this shop is valid or not by calling checkValidShop
* **Modify User:** allows the admin to modify a user. It will call modifyUser and check if this user is valid or not by calling checkValidUser

# DEPLOYMENT



* The components are deployed on the following nodes:
  + Customer computer: This node is where the user's web browser is running.
  + Web server node: This node is responsible for showing the interfaces and other static resources to clients.
  + Application server node: This node is responsible for running the application's business logic. Application consists of the components at *Section 4*:
    - Authentication component
    - Customer component
    - Shopping component
    - Review component
    - Seller component
    - Admin component
  + Database server node: The database server stores the following data:
    - Customer information: This data includes the customer's name, address, email address, and other contact information.
    - Product information: This data includes the product name, description, price, and other product information.
    - Transaction information: This data includes the customer's purchase history.
  + Payment server node: This node is responsible for directing the users to Payment services to purchase the orders

# IMPLEMENTATION VIEW



* BookAlley/Frontend: include code for the front end
* BookAlley/Frontend/GlobalImages: include all images of the project (Ex: logo, icon, …)
* BookAlley/Frontend/Source:include all source code of the project
* BookAlley/Frontend/Source/Styles: folder serves style including Fonts, Partials and Variables.
* BookAlley/Frontend/Source/Actions: include that happened in the application, based on Redux Toolkit structure. This folder is controller in all components (Section 4)
* BookAlley/Frontend/Source/Components: include some components to help build pages (Ex: Header, Footer, …)
* BookAlley/Frontend/Source/Pages: include all pages in the website (Ex: home page, product page, …). This folder is interface in all components (Section 4).
* BookAlley/Frontend/Source/Reducers: include all reducers, based on Redux Toolkit
* BookAlley/Frontend/Source/Routes: direct users to specific pages
* BookAlley/Frontend/Source/Utils: include some partial components
* BookAlley/Backend: include code for the backend
* BookAlley/Backend/Config: include configuration for database, cloud storage,…
* BookAlley/Backend/Middlewares: contains middlewares for authentication, file upload, error handling and role checking.
* BookAlley/Backend/Models: contains information schema model for mongodb.
* BookAlley/Backend/Routes/API: contains routers and controllers for API.
* BookAlley/Backend/Utils: contains utility functions (file upload, sending email,...)