<Alpha-SE>

<EBook4U>
Software Architecture Document

Version <1.2>

Revision History

Date	Version	Description	Author
24/11/2022	1.1	Create Software Architecture Document	Vu Hoang Phuc
14/12/2022	1.2	Revise Software Architecture Document	Vu Hoang Phuc

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This Software Architecture Document provides an architectural overview of the Reading book web. The Website System is being developed by Alpha-SE to support reading books online.
This Document has been generated directly from the Reading Book Analysis & Design Model implemented in Alpha-SE. The majority of the sections have been extracted from the Thai Model using SoDA and the Software Architecture Document template.

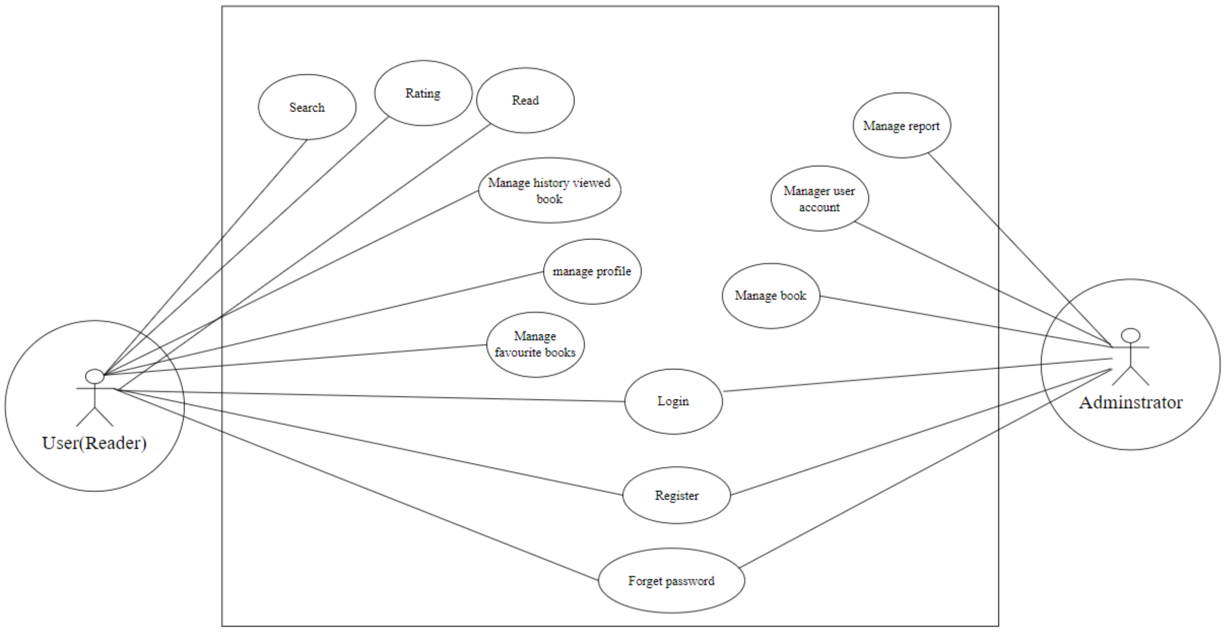
1.3 Acronym list:

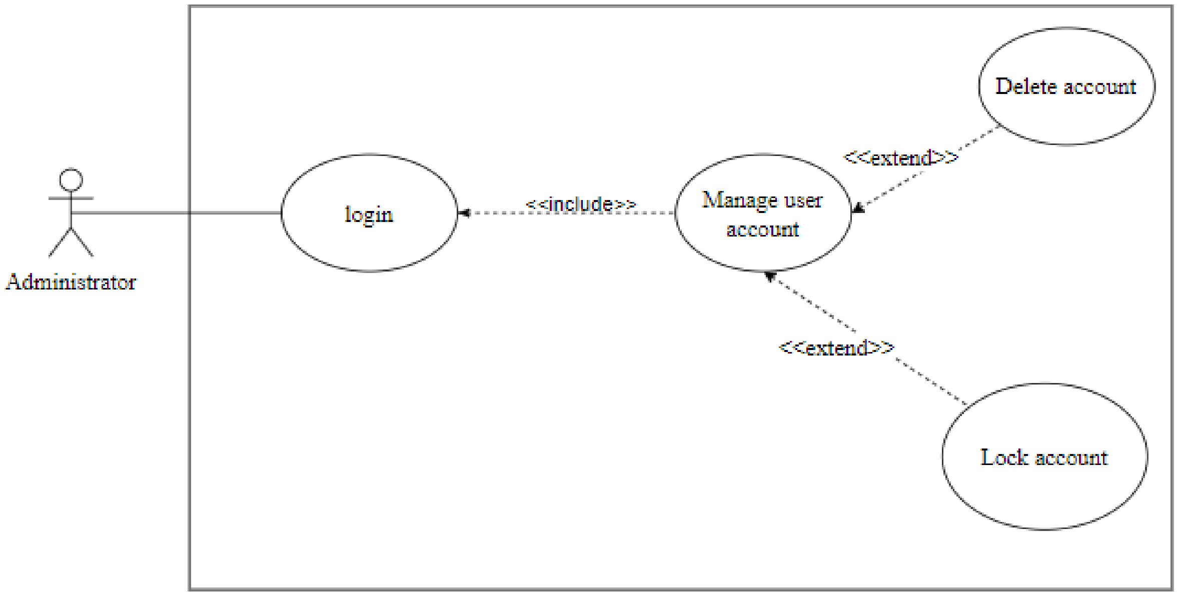
- SAD: Software Architecture Document.
- REST: Representational State Transfer, web API featuring a state-less client-server infrastructure.
- API: Application Programming Interface, a protocol used as an interface to allow communication between different components.
- MJPEG: Motion JPEG, a video format in which each frame is compressed as a JPEG image.
- CSS: Cascading-Style Sheets, document that describes the appearance of web pages.
- JSON: JavaScript Object Notation, a text-based standard for human-readable data exchange.
- MVC: Model-View-Controller, a software architecture pattern that separates the physical way to store data, the business logic and the appearance to the user.

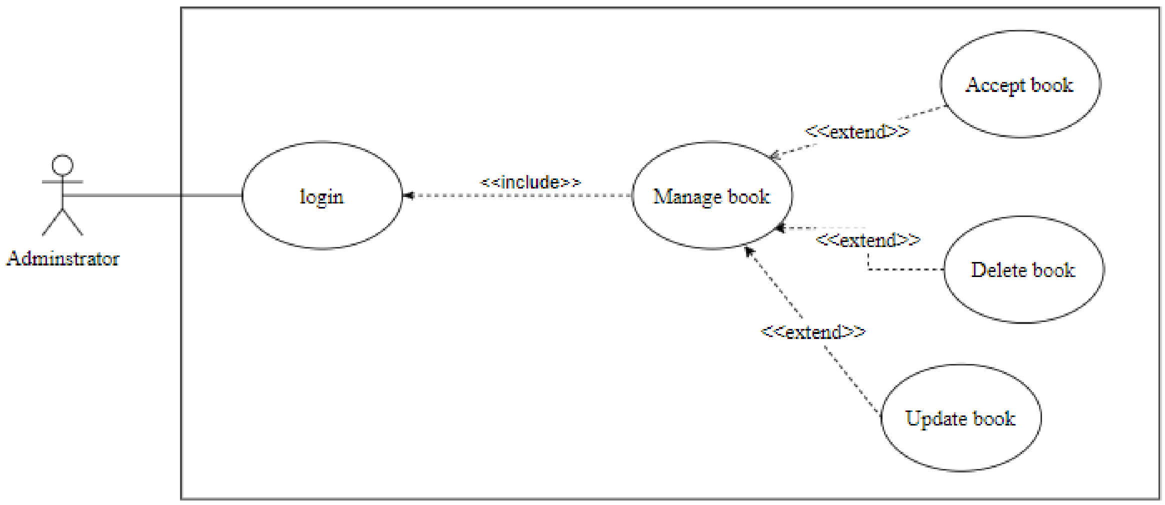
2. Architectural Goals and Constraints

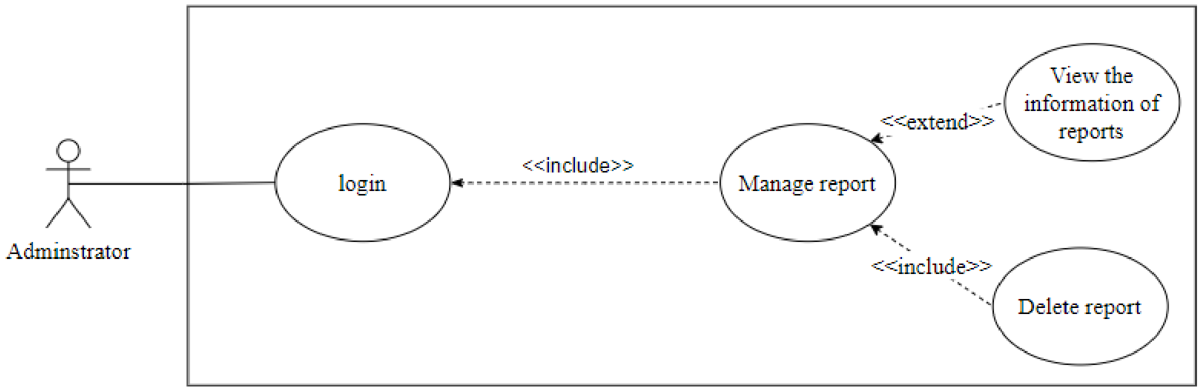
- Security for the account user
    + The password of users will be encrypted and stored in the database. Besides, ensuring the admin of the database doesn't know what password is
    + When the user logins, if the user types the password incorrectly more than five times, the website will lock the account and force the user to input the code to log in again
- To limit the leaking information of users outside, we will apply some method to control access to system because the data of the user will be stored in a cloud database
- Ensure the system still runs if it has an error by using try catch to catch the error
- Maintain system continuously
- Legal aspect, not existing any book or comment which has content 18+, violence,…
    + When book is posted, admin will check it and if it is valid, book will be accepted to display in website
- Loading well when a large number of people access the website at the same time to ensure the website does not lag. This enhances the user experience
- In term of front-end website will be coded by ReactJS, in term of back-end website will be coded by NodeJS and ExpressJS and in term of database website will use mongodb to store data.
- Environment of the application is web.
- Loading well when a large amount of people access to website at the same time.

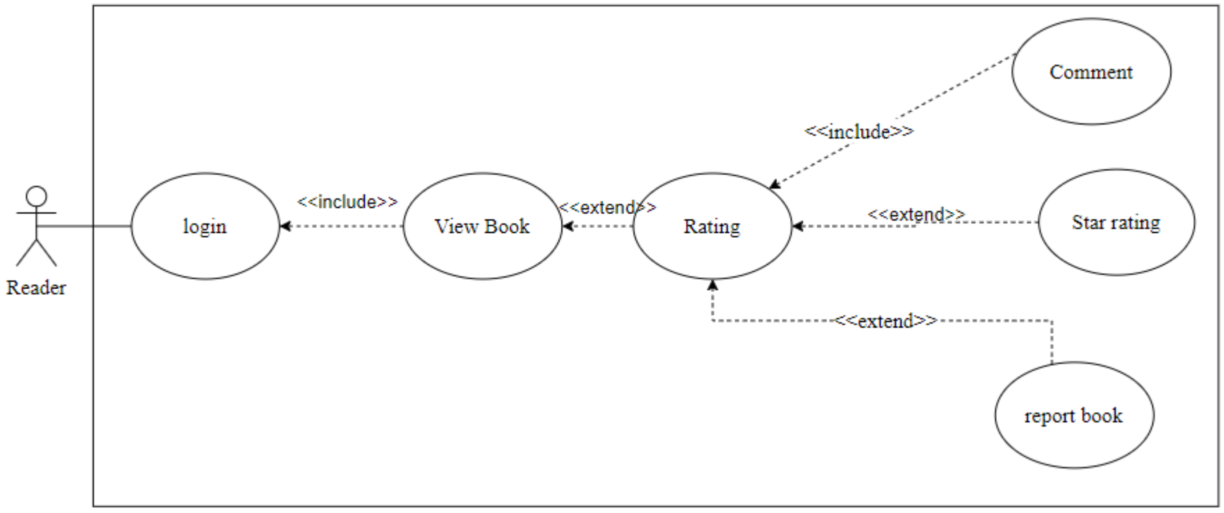
3. Use-Case Model

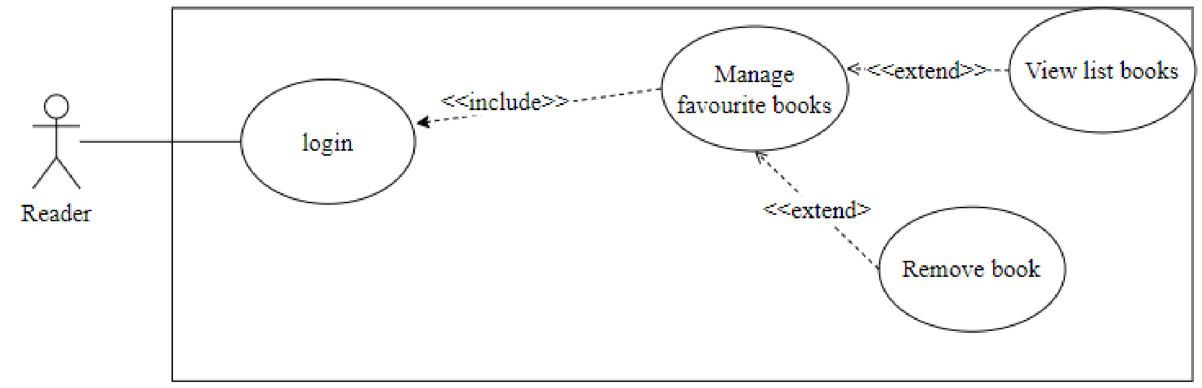

Search
Rating
Read
Manage history viewed book
manage profile
Manage favourite books
Login
Register
Forget password
Manage report
Manager user account
Manage book
User(Reader)
Adminstrator

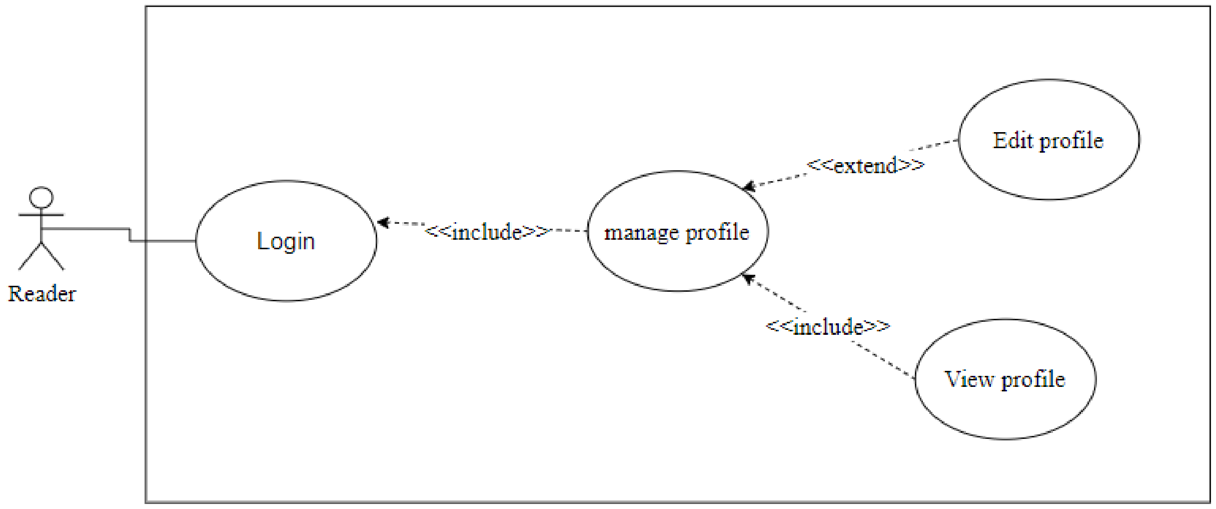

Administrator
login
<<include>>
Manage user account
<<extend>>
Delete account
<<extend>>
Lock account

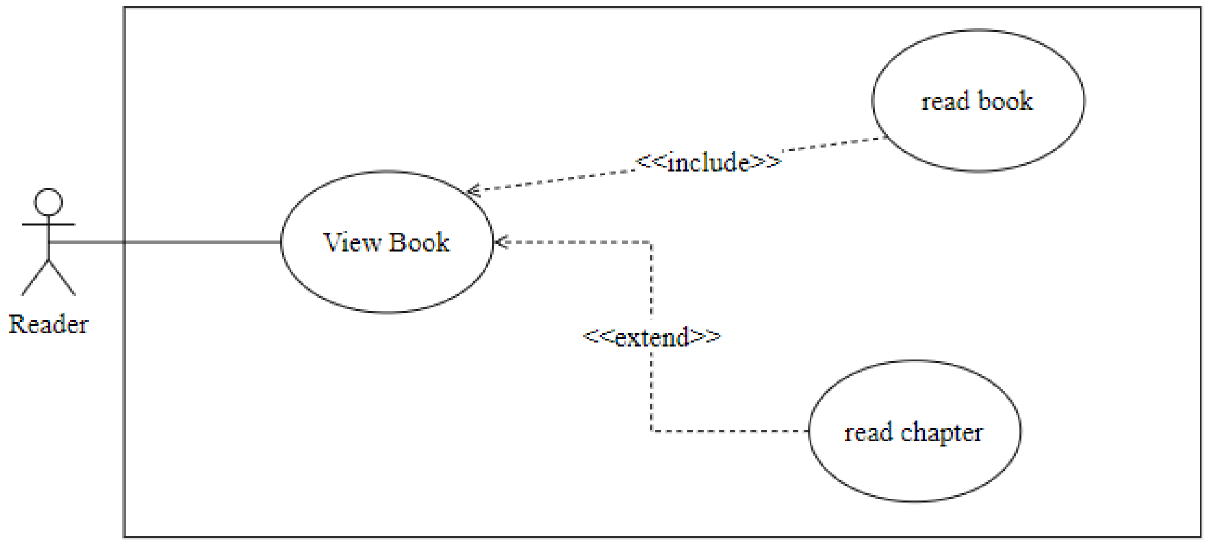

Adminstrator
login
<<include>>
Manage book
<<extend>>
Accept book
<<extend>>
Delete book
<<extend>>
Update book

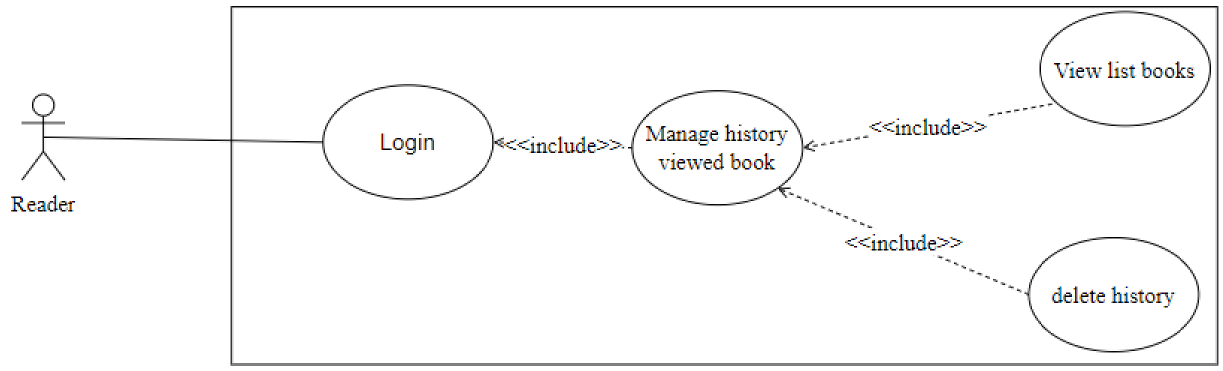

Adminstrator
login
<<include>>
Manage report
<<extend>>
View the information of reports
<<include>>
Delete report

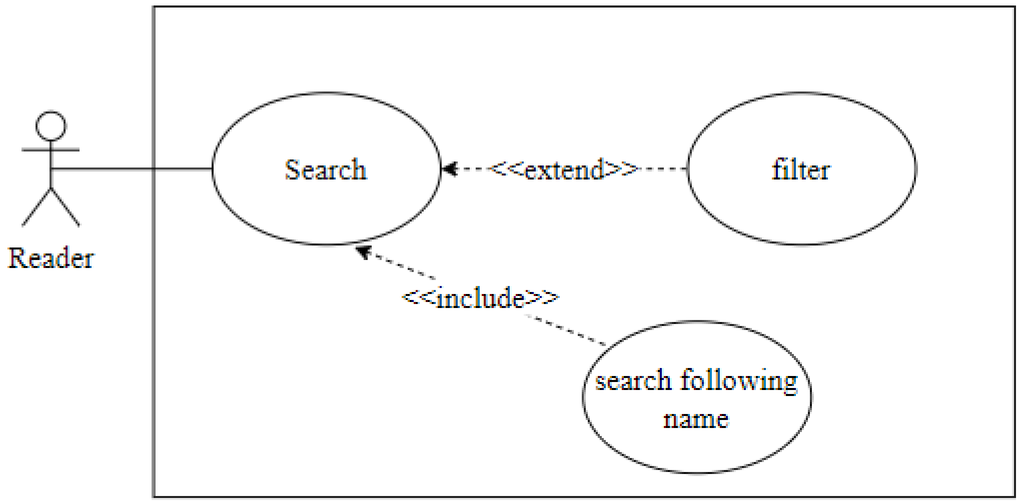

Reader
login
<<include>>
View Book
<<extend>>
Rating
<<include>>
Comment
<<extend>>
Star rating
<<extend>>
report book

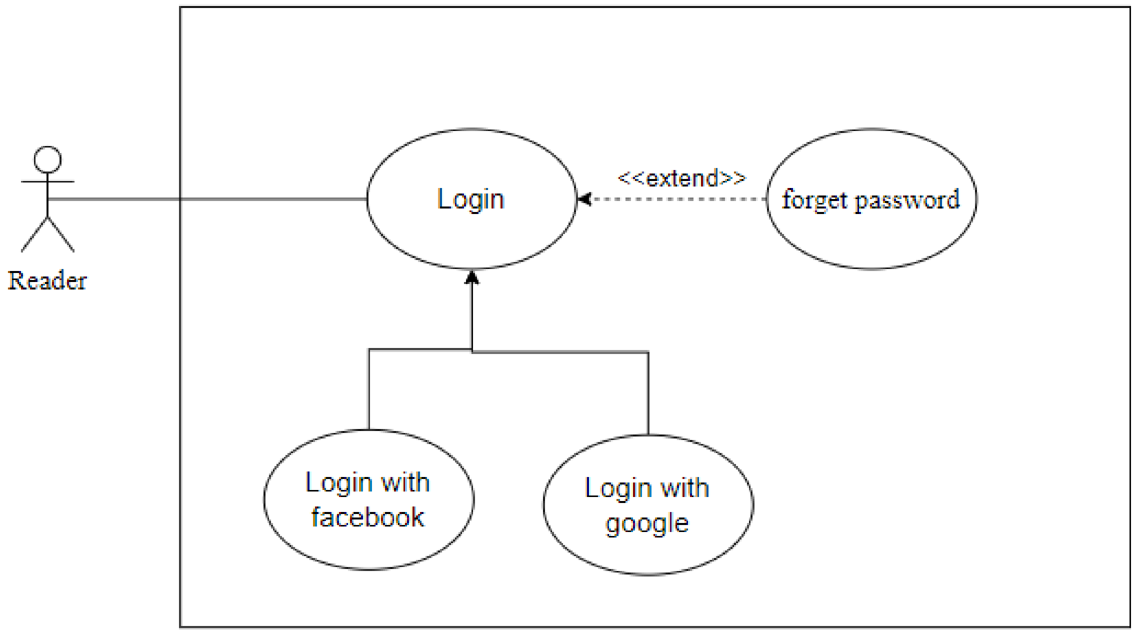

Reader
login
<<include>>
Manage favourite books
<<extend>>
View list books
<<extend>
Remove book


Reader
Login
<<include>>
manage profile
<<extend>>
Edit profile
<<include>>
View profile

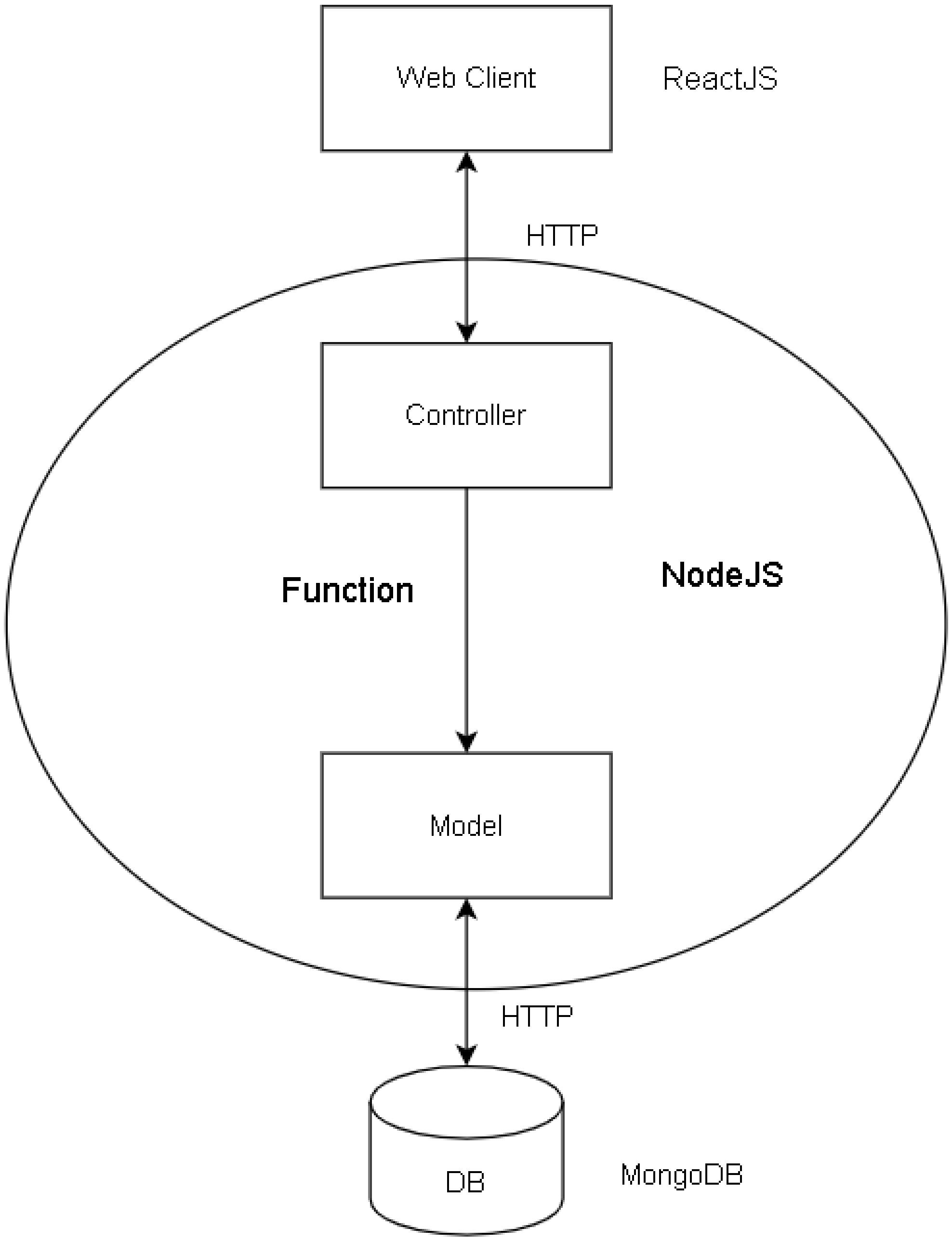

Reader
View Book
<<include>>
read book
<<extend>>
read chapter


Reader
Login
<<include>>
Manage history viewed book
<<include>>
View list books
<<include>>
delete history


Reader
Search
<<extend>>
filter
<<include>>
search following name


Reader
Login
<<extend>>
forget password
Login with facebook
Login with google

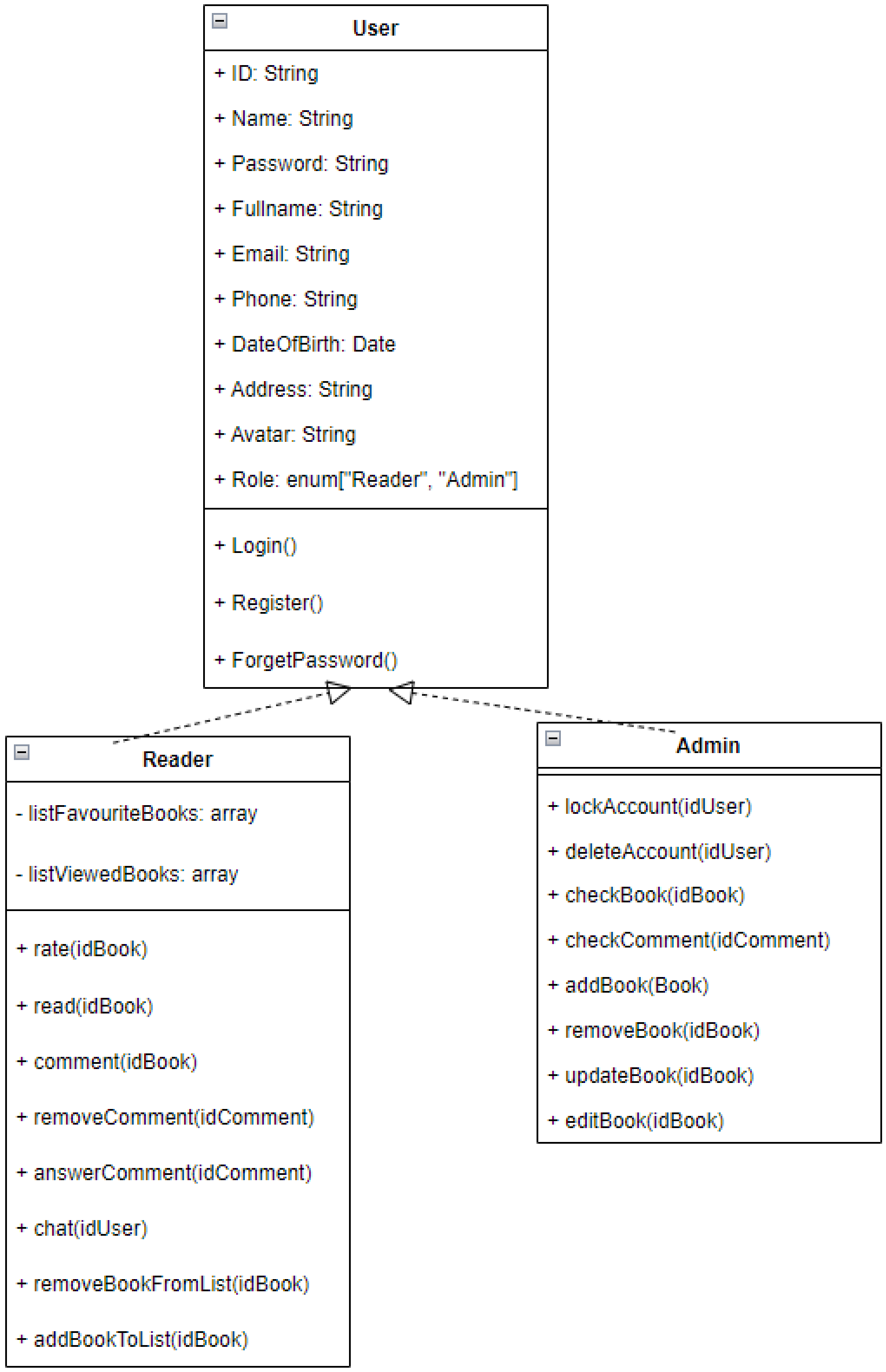
4. Logical View



4.1 Client

+ Language: ReactJS
- Display the interface of main website for reading books, user detail page(including list of favorite books, information about the user,...), displaying books,...

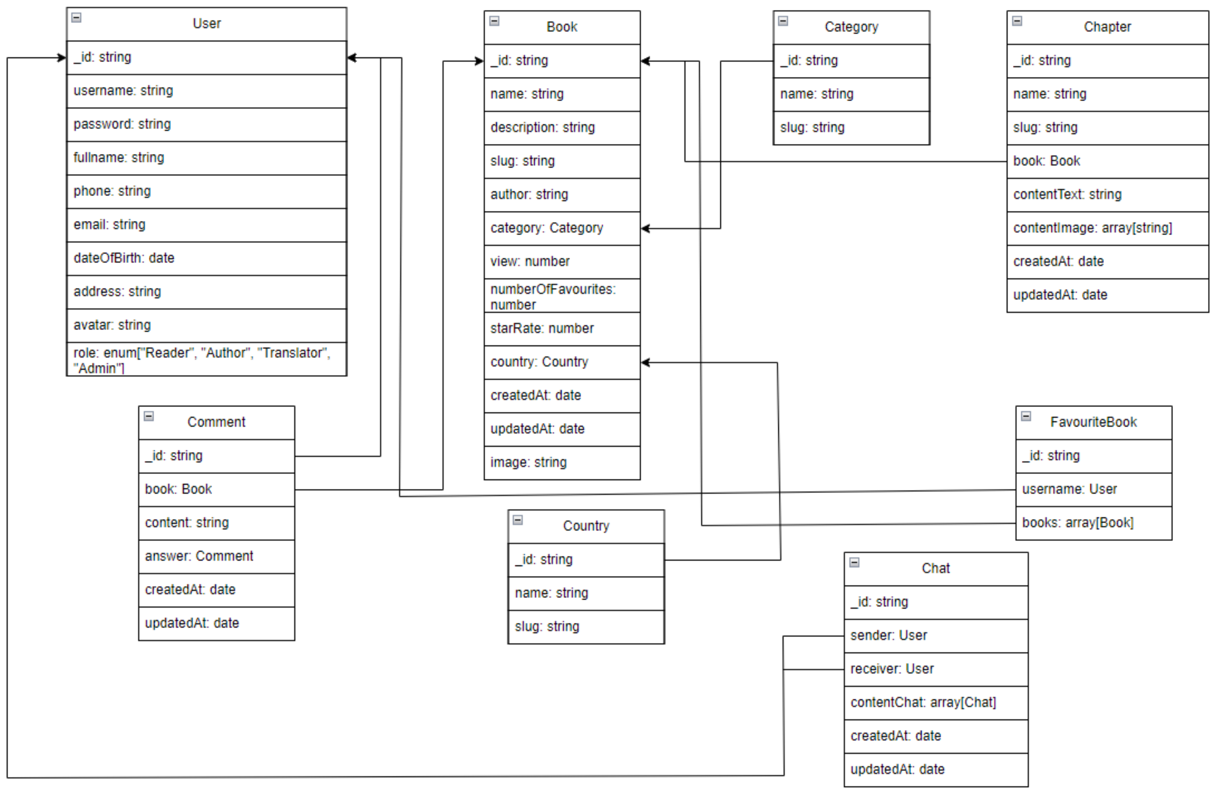
- Request api to server in order to offer users required information.



User
+ ID: String
+ Name: String
+ Password: String
+ Fullname: String
+ Email: String
+ Phone: String
+ DateOfBirth: Date
+ Address: String
+ Avatar: String
+ Role: enum["Reader", "Admin"]
+ Login()
+ Register()
+ ForgetPassword()
Reader
- listFavouriteBooks: array
- listViewedBooks: array
+ rate(idBook)
+ read(idBook)
+ comment(idBook)
+ removeComment(idComment)
+ answerComment(idComment)
+ chat(idUser)
+ removeBookFromList(idBook)
+ addBookToList(idBook)
Admin
+ lockAccount(idUser)
+ deleteAccount(idUser)
+ checkBook(idBook)
+ checkComment(idComment)
+ addBook(Book)
+ removeBook(idBook)
+ updateBook(idBook)
+ editBook(idBook)

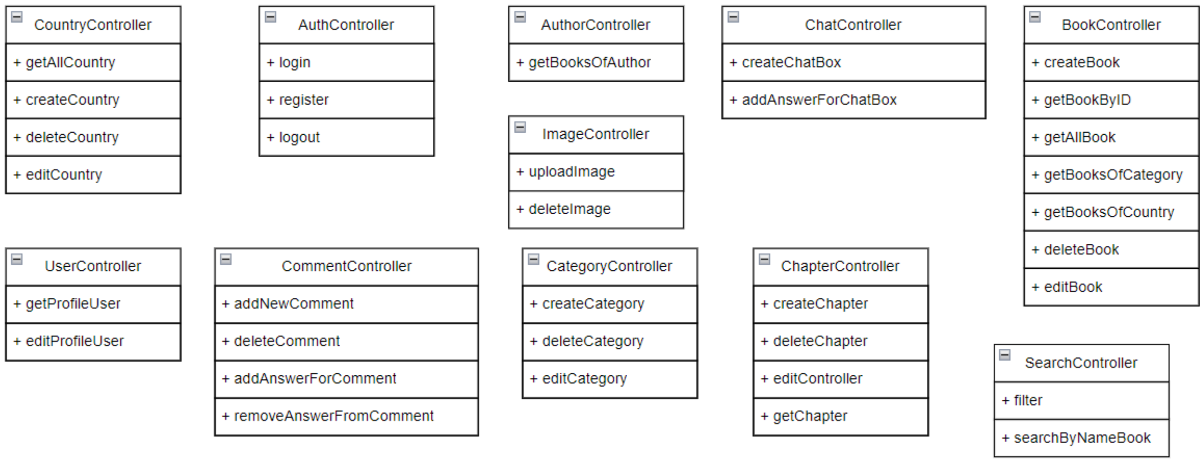
4.2 Server

- **Language server:** Nodejs
- **Model:** storing data and logic processing calculations to solve problems that software cares about (business logic). The Component Model is usually displayed in Domain Model form. A model where to initialize objects for collections and init collection. The Model will interact with Database directly. In our server, The Model defines every field for every collection in Database.



+ Description: Model connect to database through config in Back-End server

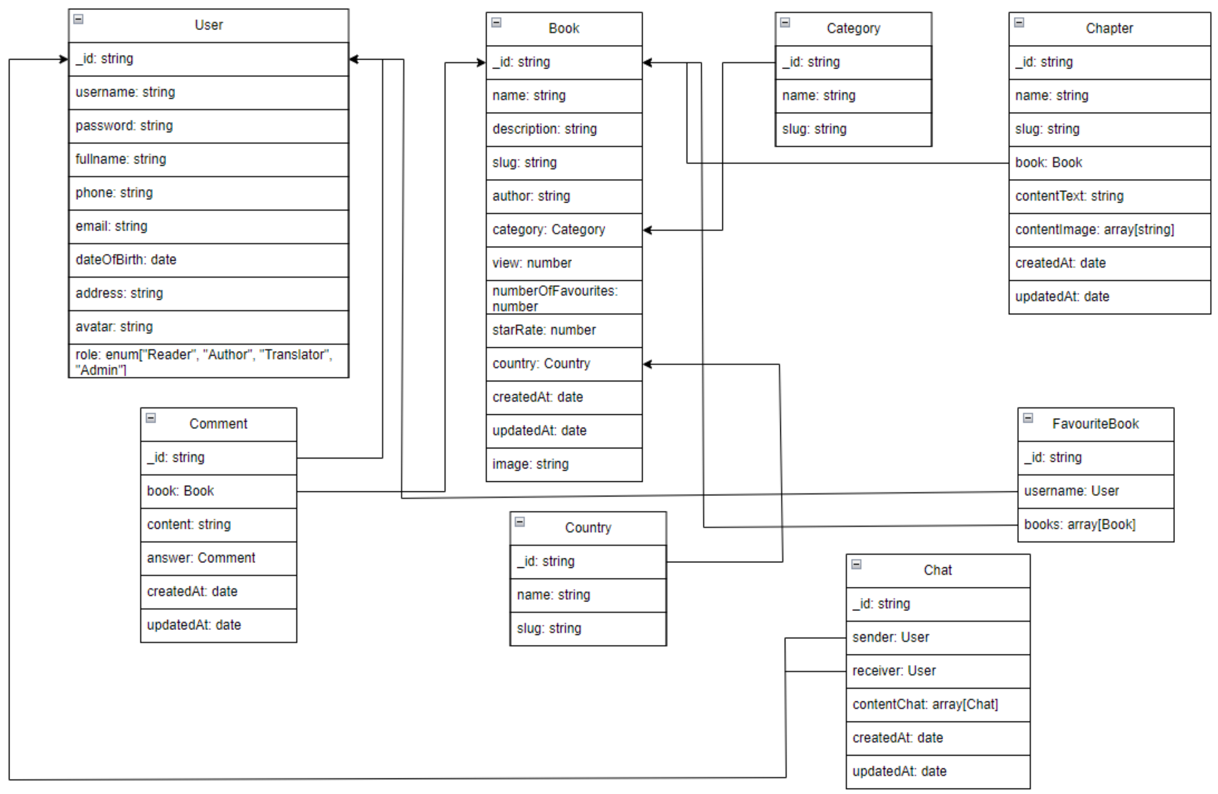
- **Controller:** is the component that takes responsibility for handling request API from Front-End/User to server and data posted by Front-End/User and client Controller is a bridge between users and application in the back-end. Accessing the Model's Data to get, put, delete and post data.



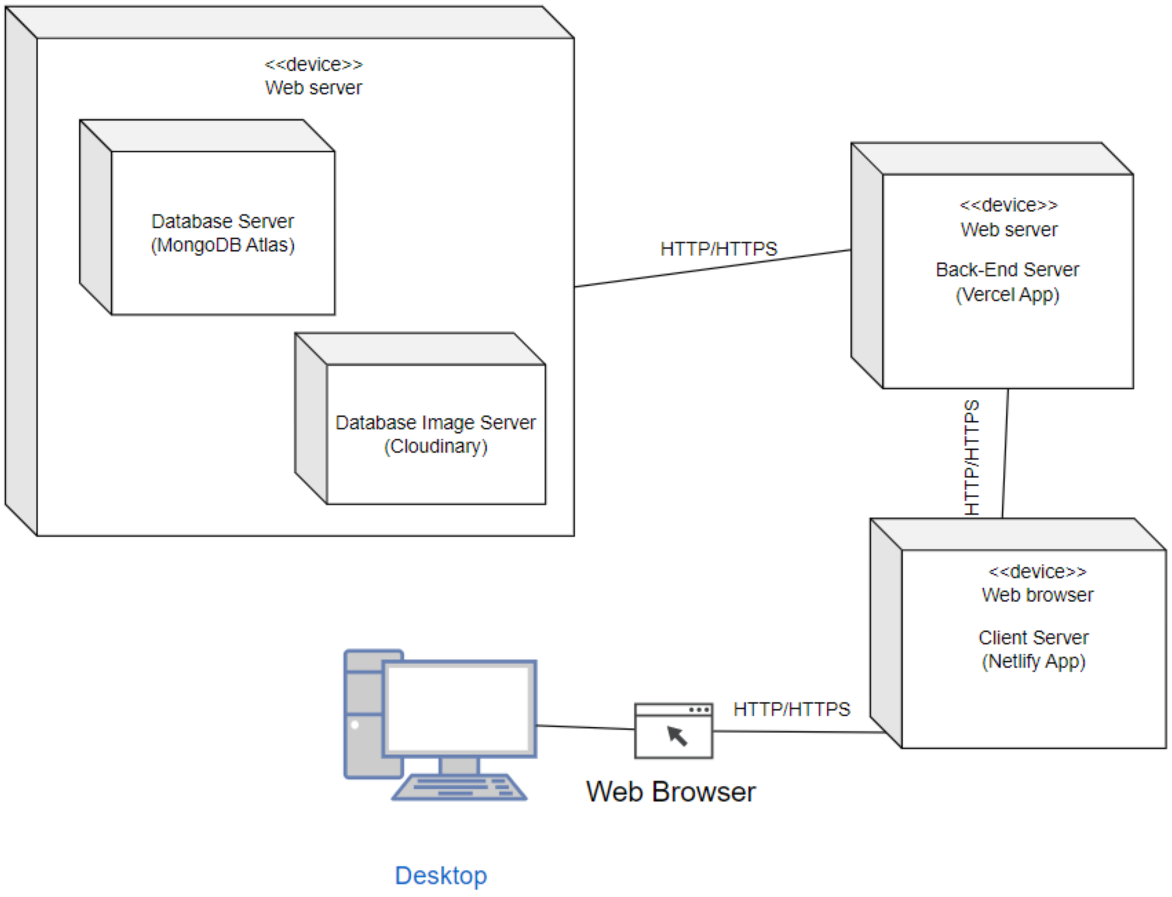
+ Description: Controller use model to interact with database

4.3 Database

- Database: MongoDB version cloud
- Data is stored in collection/document form.
- In MongoDB, there are 7 collections and every collection has many documents which has data
- 7 collections include:
  + users
  + books
  + categories
  + chapters
  + comments
  + countries
  + chats
  + favoriteBooks


User
_id: string
username: string
password: string
fullname: string
phone: string
email: string
dateOfBirth: date
address: string
avatar: string
role: enum["Reader", "Author", "Translator", "Admin"]
Book
_id: string
name: string
description: string
slug: string
author: string
category: Category
view: number
numberOfFavourites: number
starRate: number
country: Country
createdAt: date
updatedAt: date
image: string
Category
_id: string
name: string
slug: string
Chapter
_id: string
name: string
slug: string
book: Book
contentText: string
contentImage: array[string]
createdAt: date
updatedAt: date
Comment
_id: string
book: Book
content: string
answer: Comment
createdAt: date
updatedAt: date
Country
_id: string
name: string
slug: string
FavouriteBook
_id: string
username: User
books: array[Book]
Chat
_id: string
sender: User
receiver: User
contentChat: array[Chat]
createdAt: date
updatedAt: date

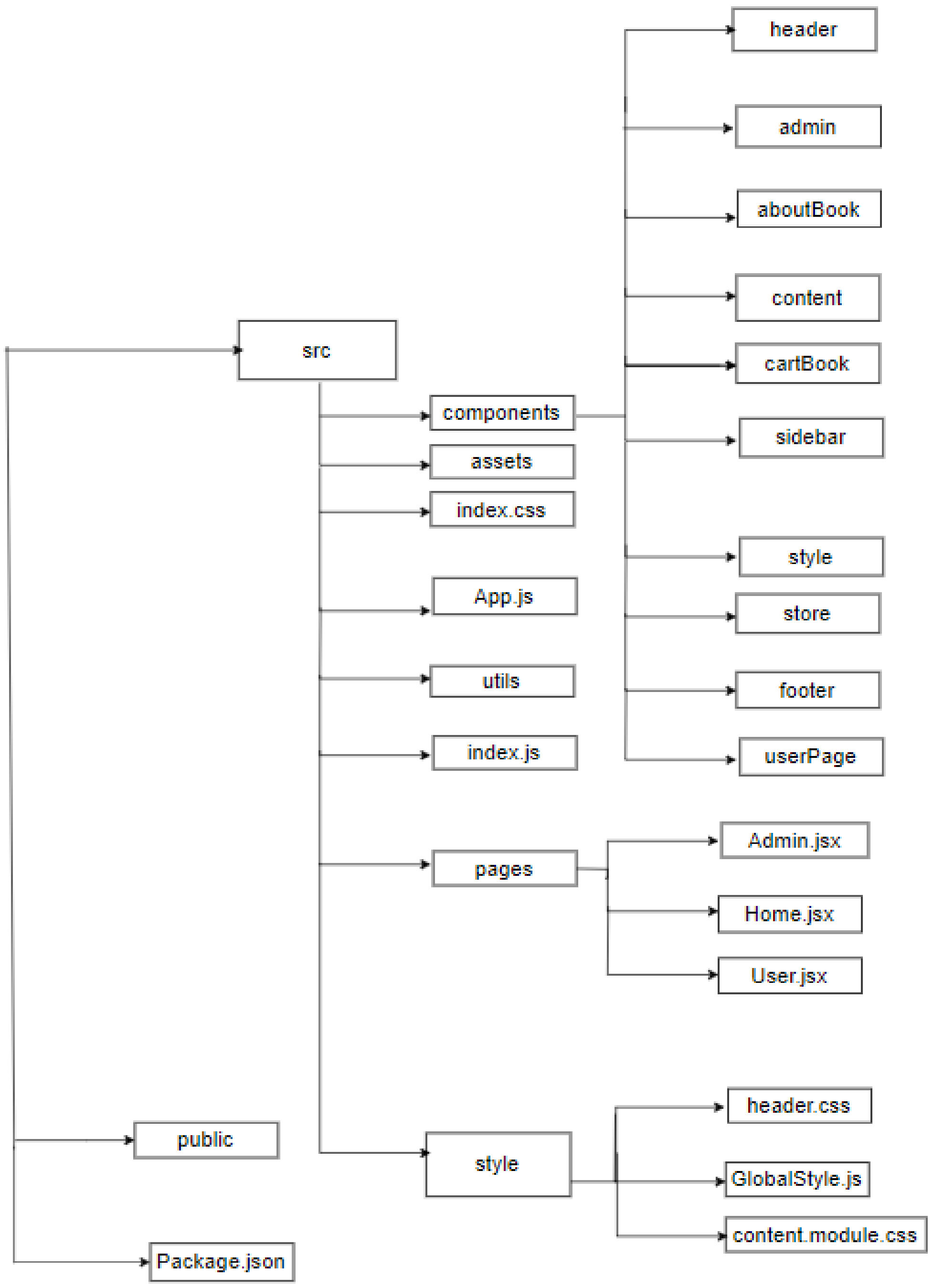
5. Deployment



- Node database server: we use 2 cloud databases to store data.
  - MongoDB Atlas: It is a cloud database free 500MB using. In MongoDB Atlas, we store all collections and data of the website.
  - Cloudinary: It is a cloud database free. We use it to store images for book, chapter and avatar users.
- Node Back-End Server: Vercel App is free and supports deployment of app server node js + express. The Back-End server will include the model, controller, and config connected to MongoDB Atlas, Cloudinary. Vercel App will supply us with one host to run the Back-End app. Front-End can interact with this host to get, post, put, and delete data and our server can get, handle and return API.
- Node Front-End Server: Netlify App is free and supports deployment of Front-End app ReactJS. The app will supply us with one host to run the Front-End app. Users can access this host to use our web.

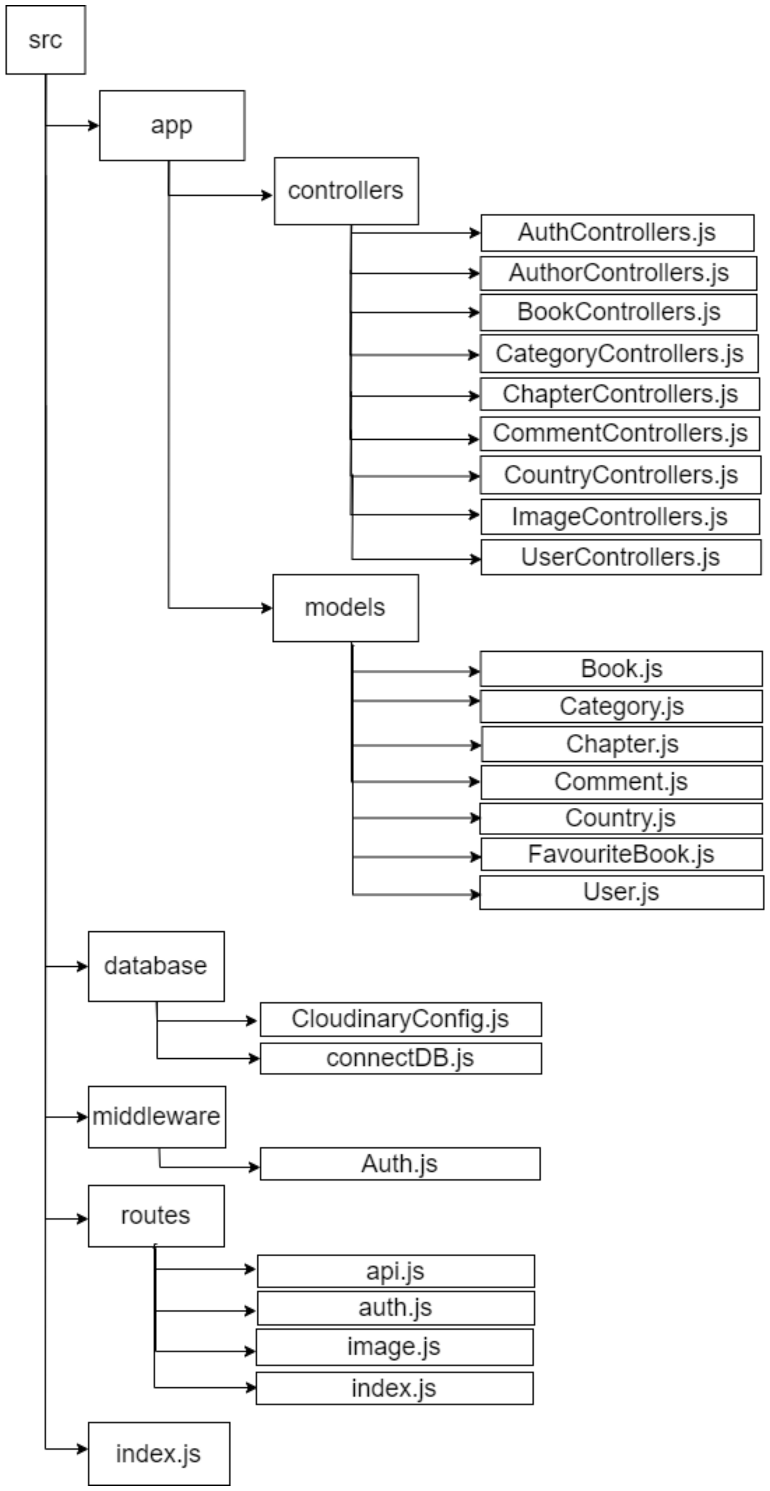
6. Implementation View

Front-End



- We'll split the UI into components contained in the components directory, which makes code reuse easy.
- The page folder contains parent components containing child components, which are interface pages
- Utils contains the necessary tools for the code
- Asset contains resources such as images,video,...

Back-End



- Controllers folder: In this folder, we create, and set up many functions described in sections 4-4.2.
- Models folder: in this folder, every js file is a collection in the database MongoDB and in there, we define the fields of a collection described in sections 4-4.1

- Database folder: in this folder, we config to connect with the database MongoDB atlas and Cloudinary
- Middleware folder: in this folder, we create many function middleware to handle between router and controller such as check login, ...
- Routes folder: in this folder, we create a prefix URL to match with every controller
- index.js file: this is a server file, and in there, we config the server. This file is used for starting the server