Comic World

Software Architecture Document

Version 2.0

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 03/08/2023 | 1.0 | Write part 1, 2, 3, 4 | Ngô Quốc Quý  Phạm Phú Toàn |
| 04/08/2023 | 1.1 | Write part 4.1, 4.2,4.3,4.6 | Nguyễn Trần An Hòa |
| 04/08/2023 | 1.2 | Write part 4.7, 4.8 | Tô Khánh Linh |
| 04/08/2023 | 1.3 | Write part 4.9 | Nguyễn Thuận Phát |
| 19/8/2023 | 2.0 | Write part 5,6 | Nguyễn Trần An Hòa |

Table of Contents

[**1. Introduction 4**](#_heading=h.kx24v3rick2j)

[**2. Architectural Goals and Constraints 4**](#_heading=h.30j0zll)

[**3. Use-Case Model 5**](#_heading=h.1fob9te)

[**4. Logical View 6**](#_heading=h.2et92p0)

[**- 4.1 Component: Index Route (Server) 7**](#_heading=h.tyjcwt)

[**- 4.2 Component: UserRoute (Server): 7**](#_heading=h.z4wvx5mr5tdg)

[**- 4.3 Component: ComicsRoute (Server) 8**](#_heading=h.a1buz7150yy3)

[**- 4.4 Component: ChapterRoute (Server) 9**](#_heading=h.9z225hivyt4)

[**- 4.5 Component: GroupRoute (Server) 9**](#_heading=h.2y6n9nv4y85t)

[**- 4.6 Component: Database 11**](#_heading=h.775ycrpoeppt)

[**- 4.7 Component: Controller ( Client ) 12**](#_heading=h.1ylajpcinvh2)

[**- 4.8 Component: Controller ( Server ) 13**](#_heading=h.dgi1q6vah74s)

[**- 4.9 Component: GUI ( Client ) 13**](#_heading=h.ixd34ts3l7k2)

[**5. Deployment 14**](#_heading=h.3dy6vkm)

[**6. Implementation View 15**](#_heading=h.1t3h5sf)

Software Architecture Document

# Introduction

This document has a purpose of providing a detailed architecture design of the Comic World System by focusing on the model of the program and all its packages. These attributes were chosen based on their importance in the design and construction of the web application.

The scope of incorporating Comic Web elements within the realm of software architecture encompasses a comprehensive exploration of the design, integration, benefits, challenges, and potential applications of this innovative approach. This scope will guide the development and understanding of Comic Web within software systems.

An innovative software design approach that blends the narrative and visual elements of comic books with interactivity and accessibility of web applications, creating engaging and immersive user experiences.

Term list:

* Stakeholder: any person involved or affected, directly or indirectly, by this product.
* Javascript: web-browser interpreted programming language for enhancing websites in a dynamic way.
* ReactJS: React is a free and open source JavaScript front-end library for building user interfaces based on individual UI components.
* Express.js: is a back-end web application framework for building RESTful APIs with Node.js
* Node.js: is a JavaScript runtime built on top of Chrome's V8 JavaScript engine

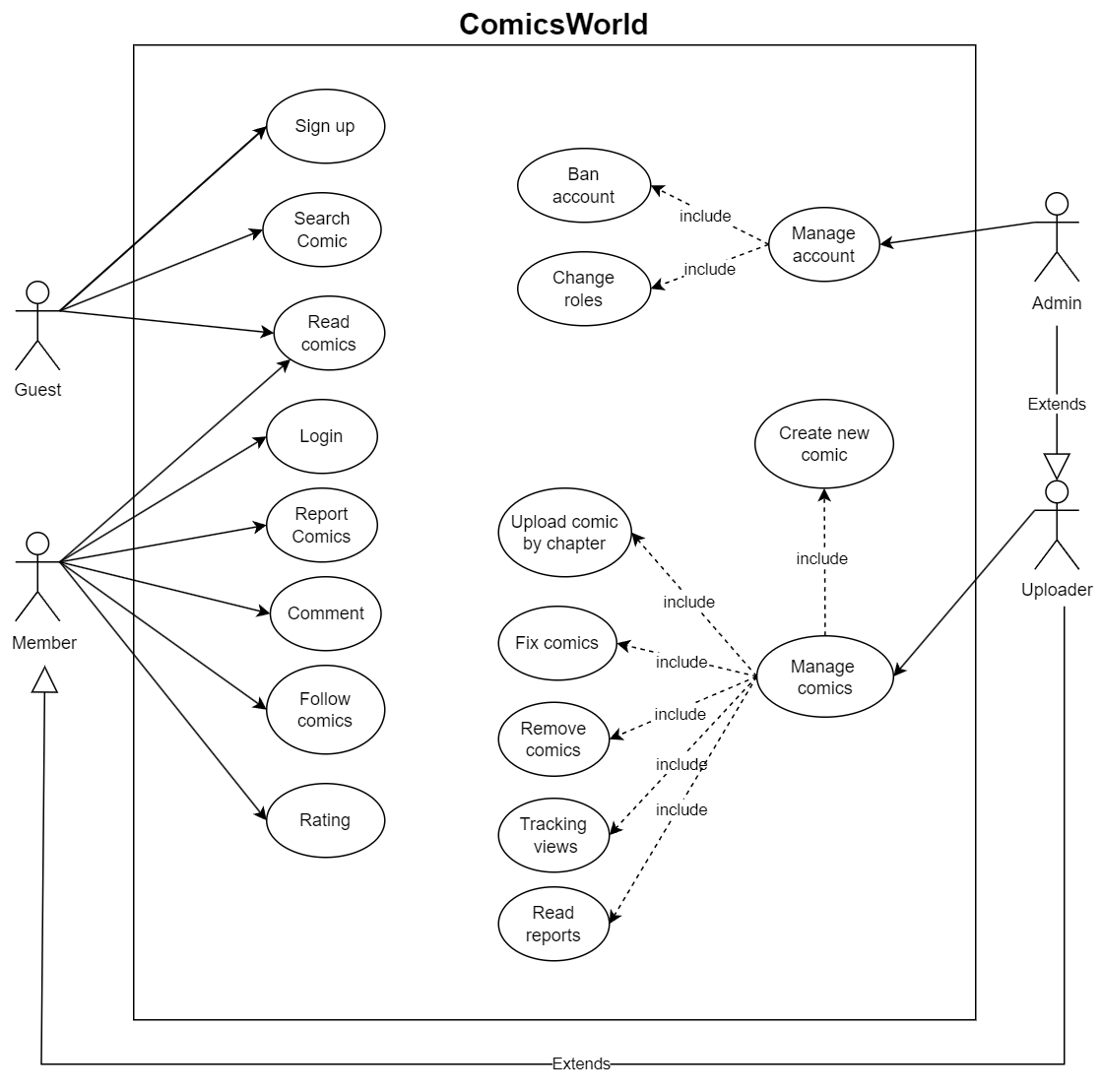
Acronym list:

* UI: User Interface
* API: Application Programming Interface

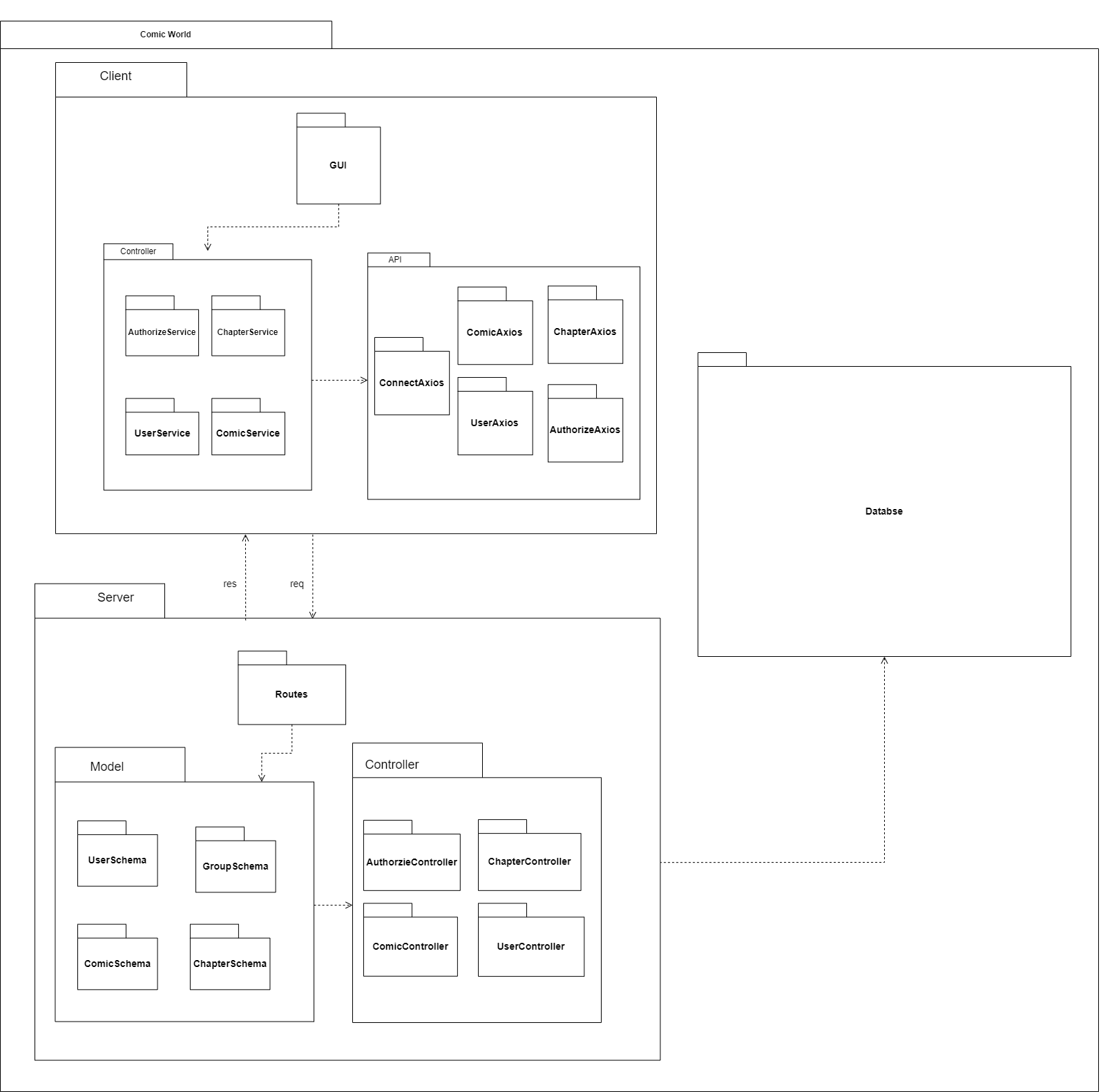
# Architectural Goals and Constraints

* Application environment: Web.
* Programming language and platform: Javascript (Node.js, ReactJS, Express.js), HTML, CSS.
* Database environment: MongoDB (this is NoSQL Database, for large scale and real-time applications).
* Security: password must be encrypted before storing in the database.
* Copyright requirement: Authors must make sure that the posted comics are their own product.
* Responsive requirement: laptop, pc, smartphone, tablet…

# Use-Case Model

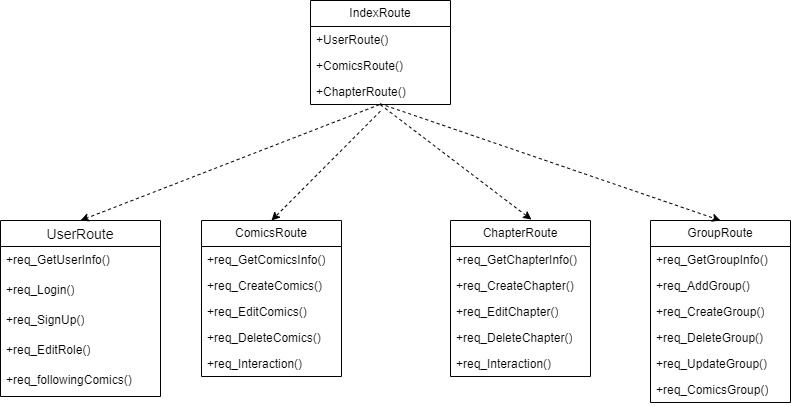


# Logical View



* **Client:** is a website developed in NodeJS, the component which displays the UI for the user, receiving requests from the user then pushing requests to server handle.
* **Service:** User services which are displayed by UI, providing any comic service for users.
* **GUI:** This displays the services in UI format which are easy for users to interact with the website.Components keep their own state, and between components also provide interfaces (props) to communicate with each other.
* **Controller:**Are payloads of information that send data from your application to your store. If any state change is necessary the change required will be dispatched through the actions.
* **Model:** Calling API to server.
* **Server:** The component that receives the request and handles it which is sent by the client, developed on the Express.js environment. The server acts as a RESTful API server, providing an interface to receive requests by client, process and send responses back to the client.
* **Routes:** The component that receives the request from users then passes it to the controller in the server handle.
* **Controller:** The component that validates data gets from request then handles it then calls to service to perform corresponding tasks, and send response for the client in final.
* **Service:** are for managing the data, the component which interacts directly with the database to query data and receives responses from the database and sends them back to service.
* **Database:** Using MongoDB

## 4.1 Component: Index Route (Server)

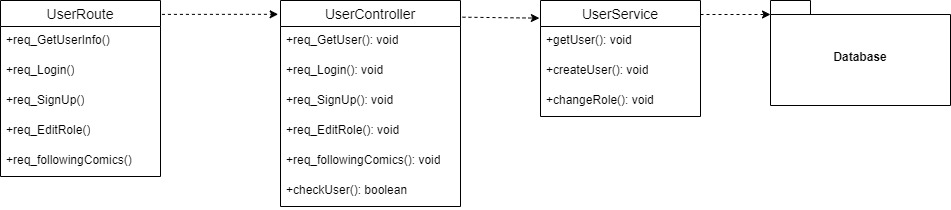


* Depending on the requests, the index router will send requests to the child routers that match that request
* Ex:

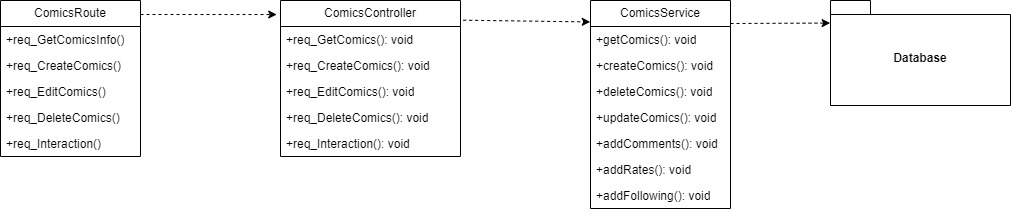
- “/user/…” will go to UserRouter

- “/comics/…” will go to ComicsRouter

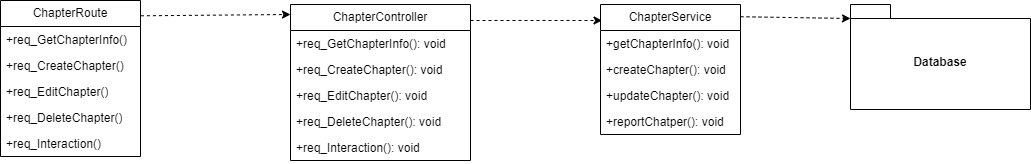
## 4.2 Component: UserRoute (Server):

* Details:
  + When a user logins, the system will call req\_Login() in UserRoute, this will call the req\_Login() in UserController. UserController will send username and password to checkUser() then compare with the one from getUser() in UserService. If it is the same, it will return true, otherwise it will send back the error message.
  + When user choose to SignUp() in UserRoute, this will call the req\_Login() in UserController, which will call the createUser() in the UserService, before submitting the information to the database, username will be sent to the checkUser() in UserController, to check if the username has been created before. If it has not been, then creating a User account continues.
  + When the admin wants to editRole in UserRoute, this will call the req\_EditRole() in UserController, which will call the changeRole() in the UserService.
  + When the user wants to know their following Comics, req\_followingComics() in UserRoute will call the req\_followingComics in UserController, which then gets the list of the comics in the getUser() in UserService.

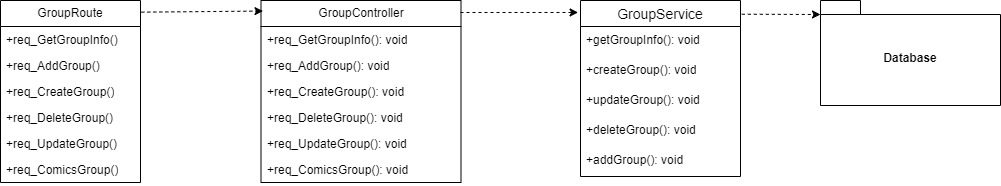
## 4.3 Component: ComicsRoute (Server)

* Details:
  + When a user clicks on the comics, the system will call the req\_GetComicsInfo() in the ComicsRoute. ComicsRoute then calls the req\_GetComics() in the ComicsController , which will call the getComics in the ComicsService.
  + When an uploader wants to create a new Comics, the system will call the req\_CreateComics() in the ComicsRoute. This will call the req\_CreateComics() in the ComicsController and then createComics() in ComicsService will be called to add new comics to the database.
  + When an uploader wants to edit a comic's information, the system will call the req\_EditComics() in the ComicsRoute. This will call the req\_EditComics() in the ComicsController .It will call the getComics() in ComicsService to get the comics information. After editing the information, the updateComics() will be called to update the new information to the database.
  + When an uploader wants to delete comics, the system will call the req\_DeleteComics() in the ComicsRoute. This will call the req\_DeleteComics() in the ComicsController. Then it will call the deleteComics() to remove it from the database.
  + When a user wants to interact with the comics, the system will call the req\_Interaction() in the ComicsRoute. This will call the req\_Interaction() in the ComicsController. In ComicsService, if it is rating, it will call the addRates(); if it is commenting, it will call the addComments(); if it is following comics, it will call the the addFollowing().

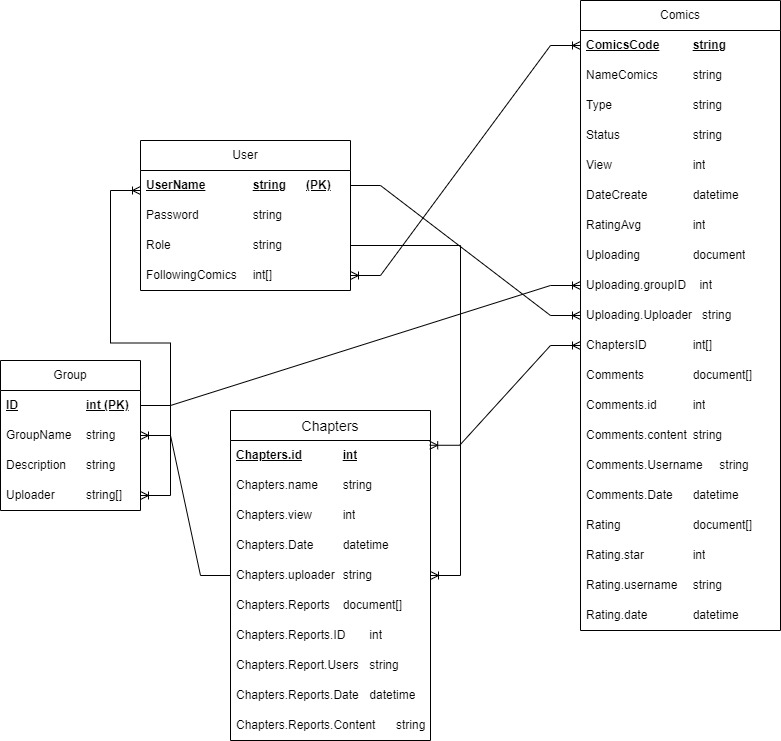
## 4.4 Component: ChapterRoute (Server)

* Details:
  + When a user clicks on a chapter, the system will call the req\_GetChapterInfo() in the ChapterRoute. ChapterRoute then calls the req\_GetChapterInfo() in the ComicsController, which will call the getChapterInfor() in the ChapterService.
  + When an uploader wants to create a new chapter, the system will call the req\_CreateChapter() in the ChapterRoute. This will call the req\_CreateChapter() in the ChapterController and then createChapter() in ChapterService will be called to add a new chapter to the database.
  + When an uploader wants to edit a chapter’s information, the system will call the req\_EditChapter))\_ in the ChapterRoute. This will call the req\_EditChapter() in the ChapterController. It will call the GetChapterInfo() in ChapterService to get the chapter information. After editing the information, the updateChapter() will be called to update the new information to the database.
  + If an uploader wants to delete a chapter, the system will call the req\_DeleteChapter() in the ChapterRoute. This will call the req\_DeleteChapters() in the ChapterController. Then it will call the deleteChapter() to remove it from the database.
  + When a user wants to interact with the chapter, the system will call the req\_Interaction() in the ChapterRoute. This will call the req\_Interaction() in the ChapterController. ChapterController will call the reportChapter() from the ChapterService.

## 4.5 Component: GroupRoute (Server)

* Details:
  + When a user wants to know the information about the group information, the system will call the req\_GetGroupInfo() in the GroupRoute. GroupRoute calls the req\_GetGroupInfo() in the GroupController . This then calls the getGroupInfo() in the GroupService.
  + When an admin creates a group, the system will call the req\_CreateGroup() in the GroupRoute. This will call the req\_CreateGroup() in the GroupController and the GroupController will call the createGroup() in the GroupService. New information will be assigned to the database.
  + When an admin wants to delete a group, the system will call the req\_DeleteGroup() in the GroupRoute. This will call the req\_DeleteGRoup() in the GroupController and the GroupController will call the deleteGroup().
  + When an uploader edits the group’s information, the system will call the req\_UpdateGroup() in the GroupRoute. Then req\_UpdateGroup() in the GroupController will be called and the GroupService returns the getGroupInfo(). After editing, the updateGroup() will be called in the GroupService. New information will be updated to the database.
  + When a user wants to see the comics that group has been created, the system will call the req\_ComicsGroup() in the GroupRoute. This will call the req\_ComicsGroup() in the GroupController and the GroupController will call the getGroupInfo() to return the name of comics.
  + When an admin assigns an uploader to a group, the system will call the req\_AddGroup() in the GroupRoute. GroupRoute then calls the req\_AddGroup() in the GroupController, then the addGroup() in the GroupService will be called. New information will be assigned to the database

## 4.6 Component: Database

****

Each class stores information in the database.

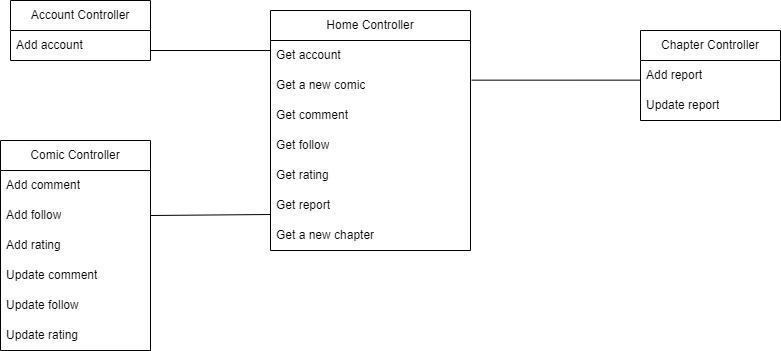
Class user: stores the information of the members (included the uploader and admin). In this class, username is the unique one and this class is the one to many relationship and many to many relationship with comics class, many to many relationship with the group class, one to many relationship with the chapters class.

Class Comics stores the information of the comics including some information which is related to the other class , which has been listed above and many to many relationship with chapters class, one to many with group class.

Class chapters stores the information about the chapters which belongs to one’s comics and also stores the relevant information like reports. Also, this class has the one to many relationship with the group class.

Class group stores the information about the group who registers to upload comics on the website. Its relationships with the other class have been mentioned above.

## 4.7 Component: Controller ( Client )

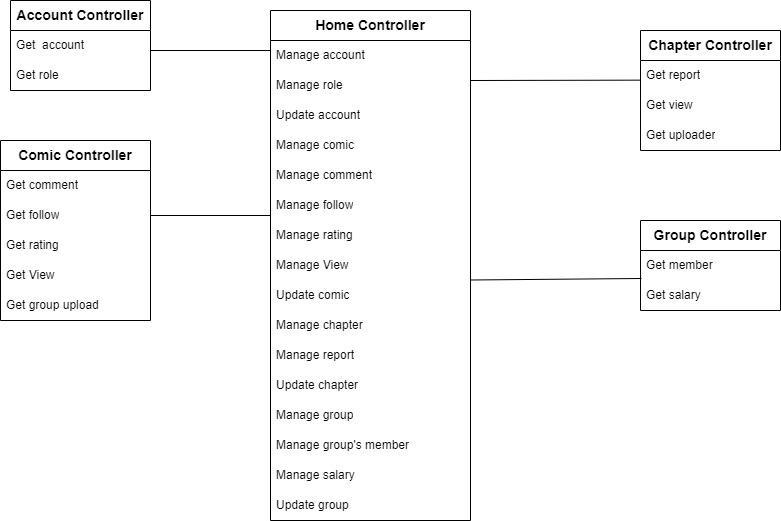
****

* Home controller: The source class is responsible for accessing the guest controllers when requested by the

user to navigate

* Account controller: Users can create accounts, update their accounts.
* Comic controller: Users can add and edit the rating, follow and comment in the comic.
* Chapter controller: Users can add and edit the report in the comic’s chapter

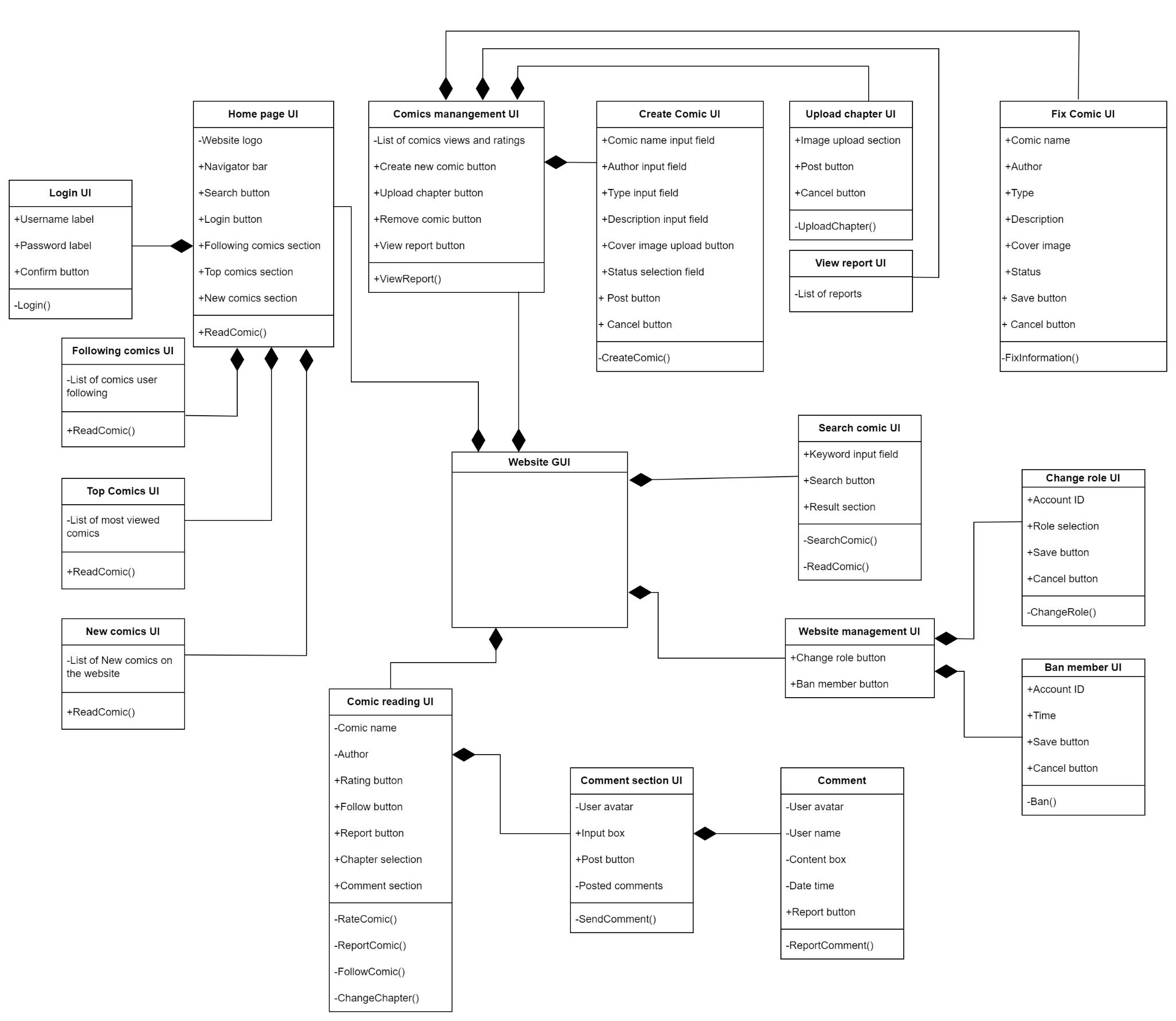
## 4.8 Component: Controller ( Server )

****

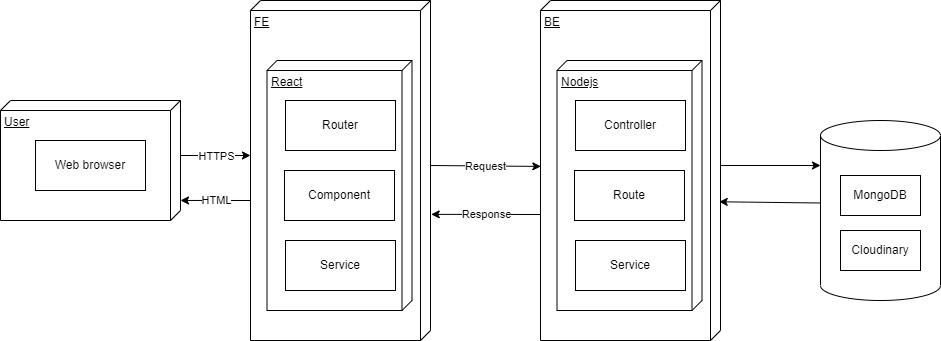
* Home controller: The source class is responsible for accessing the server controllers when requested by the server to navigate
* Account controller: Server can get user’s account, role
* Comic controller: Server can get comic’s comment, follow, rating, view and group which upload that comic
* Chapter controller: Server can get the report and the uploader of that chapter.
* Group controller: Server can get member and salary of a group.

## 4.9 Component: GUI ( Client )

* Class Home page UI: this page will show up when users access the website. Comics are displayed in the home page body, users click on the comic to read it. Comics are grouped into some sections such as top comics( the most viewed comics), new comics(new comics on the website),... make it easier for users to choose what to read.
* Class Comic management UI: uploaders go to this page to manage their posted comics. Some operations can be made by uploaders such as creating comics, removing comics, editing the information of the comics, uploading new chapters, tracking views and reports.
* Class Website management UI: website managers go to this page to manage the website. They can ban members and change the roles of members.
* Class Comic reading UI: when users click on a comic, the website will direct them to this page to read that comic. They can also follow, comment, rate and report the comic there.
* Class Search comic UI: users will be directed to this page when they click the search button on the header. When they type the keywords and click send, the website will search for it in the database and return the results. Users can click on the results to read them

****

# Deployment



* User uses a web browser to access the website and calls HTTPS to the front-end.
* Front-end sends requests and data through the back-end. Back-end receives the request then accesses the database server to get the data. Data will be saved in the database MongoDB, and the images or files will be saved on the Cloudinary. After receiving the data, Back-end sends back the response to the FE (including the messages and the data). FE receives the response , handles it and shows it to the screen to the user.

# Implementation View

