<Monesa>

Software Architecture Document

Version <2.2>

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 22/07/2022 | 2.0 | Basic description of pa4 | Team |
| 26/07/2022 | 2.1 | make clips | Team |
| 29/7/2022 | 2.2 | Format pa4 | Team |

Table of Contents

[**Introduction**](#_heading=h.gjdgxs) **3**

[**Architectural Goals and Constraints**](#_heading=h.30j0zll) **3**

[**Use-Case Model**](#_heading=h.1fob9te) **4**

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

[Component: AccountController](#_heading=h.tyjcwt) 6

[Component: AuthenticationController](#_heading=h.uo7rns63gugm) 7

[Component: ServiceController](#_heading=h.492618vbhi96) 7

[Component: WorkspaceController](#_heading=h.igal9luvi97i) 8

[Component: ManagementController](#_heading=h.poudulyoa8vl) 8

[Component: ExpenditureController](#_heading=h.eogrkd4cu62g) 9

[Component: ExpenditureModel](#_heading=h.j23xdtkbozdh) 10

[Component AccountModel](#_heading=h.6fm3w0w4ftfd) 11

[Component: DashboardView](#_heading=h.387l5rx04ve5) 12

[Component: AuthView](#_heading=h.urfxnqlurfaf) 12

[Component: FriendView](#_heading=h.84gnej3rywm5) 12

[Component: AccountView](#_heading=h.p6i1zpombe5b) 13

[Component: ServerManagementViews](#_heading=h.hm21rvzbl5sn) 13

[Component: WorkspaceManagementView](#_heading=h.7qttx2cvnrjl) 13

[Component: UserManagementView](#_heading=h.4tg1fsmx3aez) 14

[Component: WorkspaceView](#_heading=h.7jxho0ert2up) 14

[**Deployment**](#_heading=h.3dy6vkm) **15**

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

Software Architecture Document

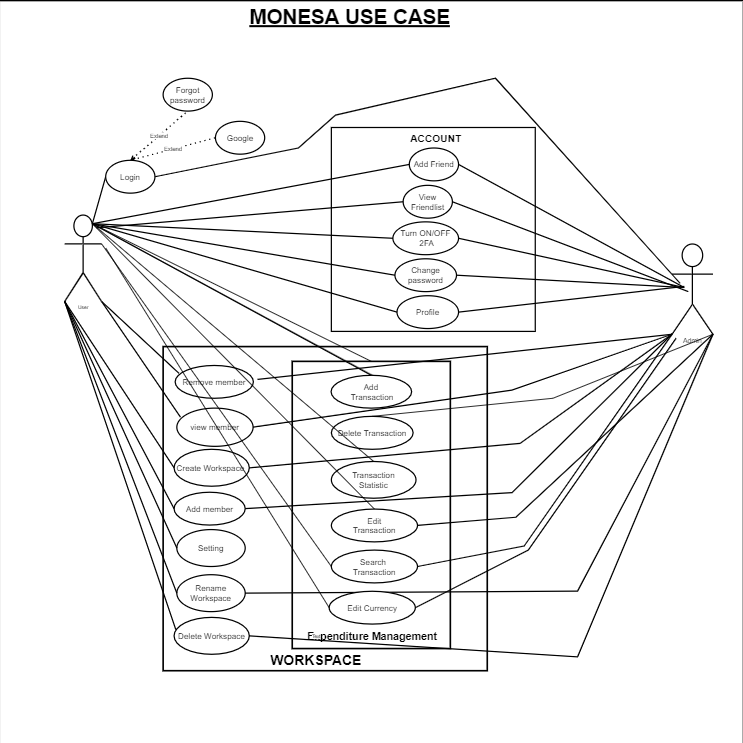
# Introduction

* Monesa is built based on MVC architecture. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Architecture Document.

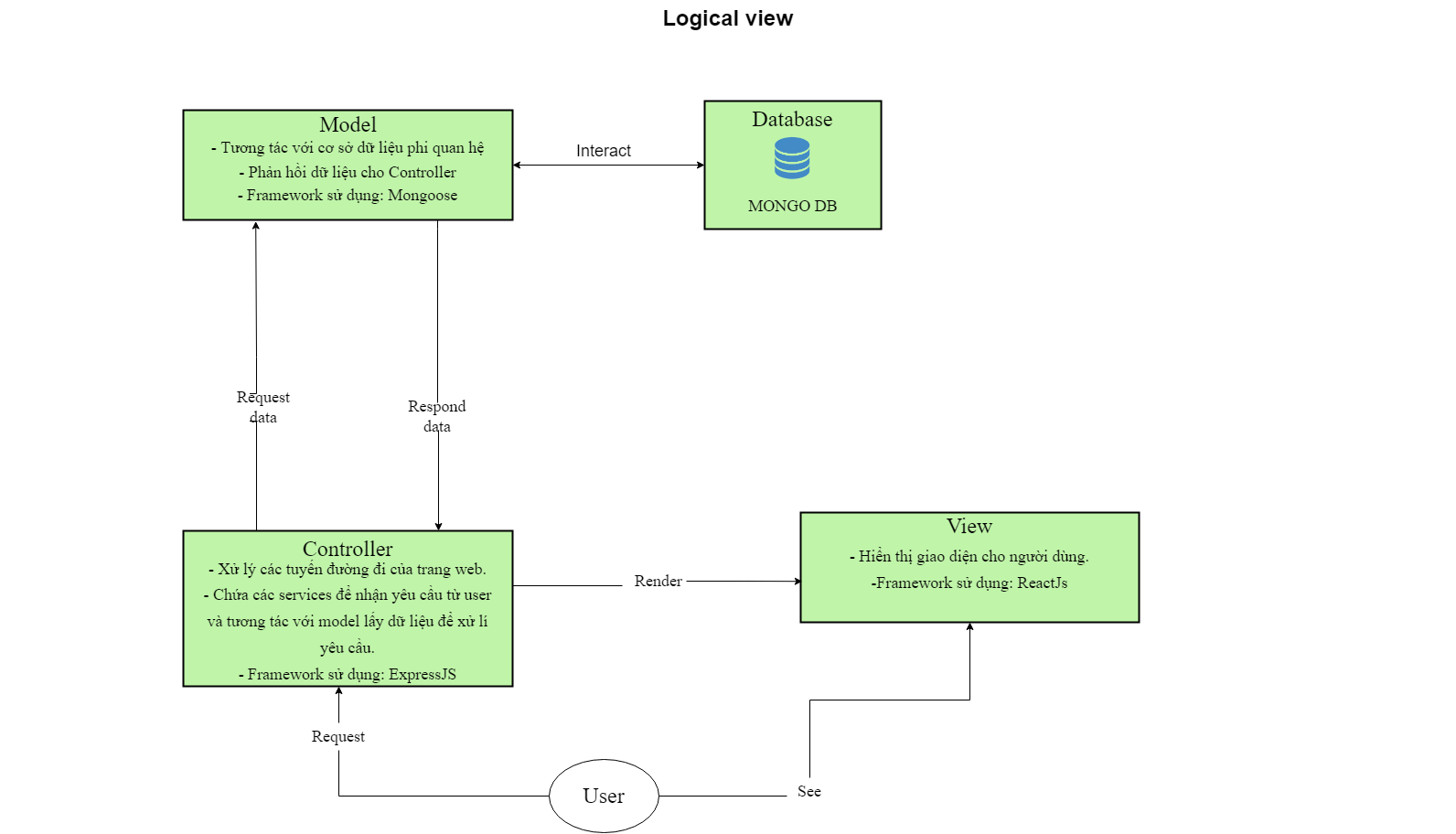
# Architectural Goals and Constraints

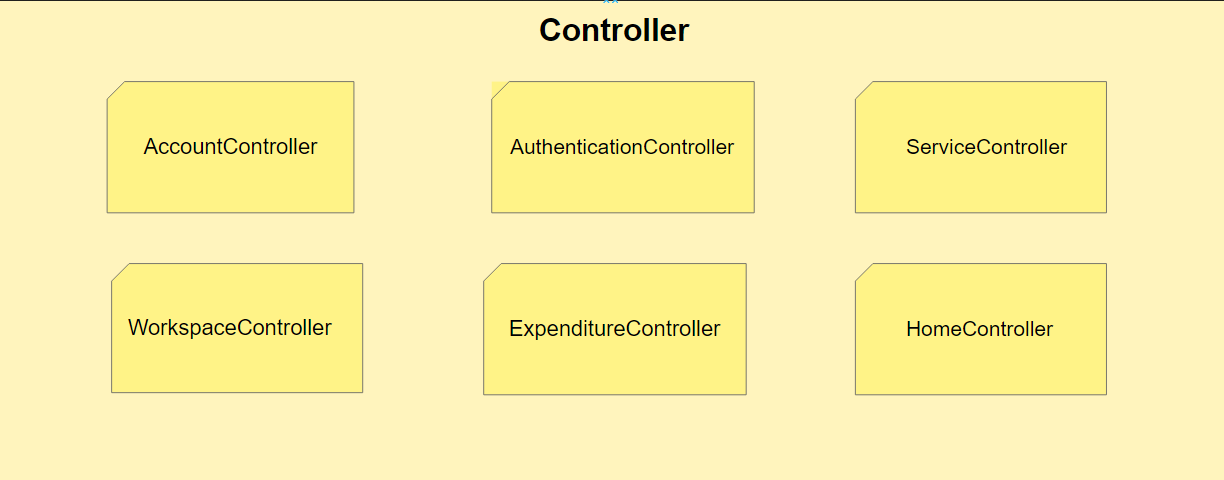
* Server uptime 99.99%
* Support all browser
* Friendly UI, Easy to use
* Data secure 100%
* Time response should not exceed 30s (Final target: Under 15s)
* Each function must not contain over 1 bug

# Use-Case Model

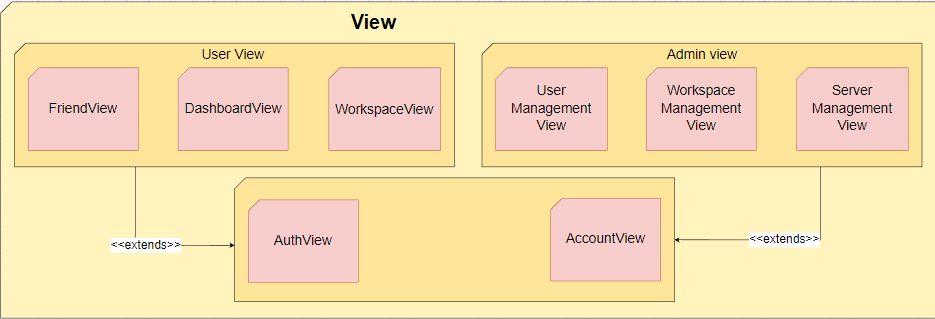


# Logical View

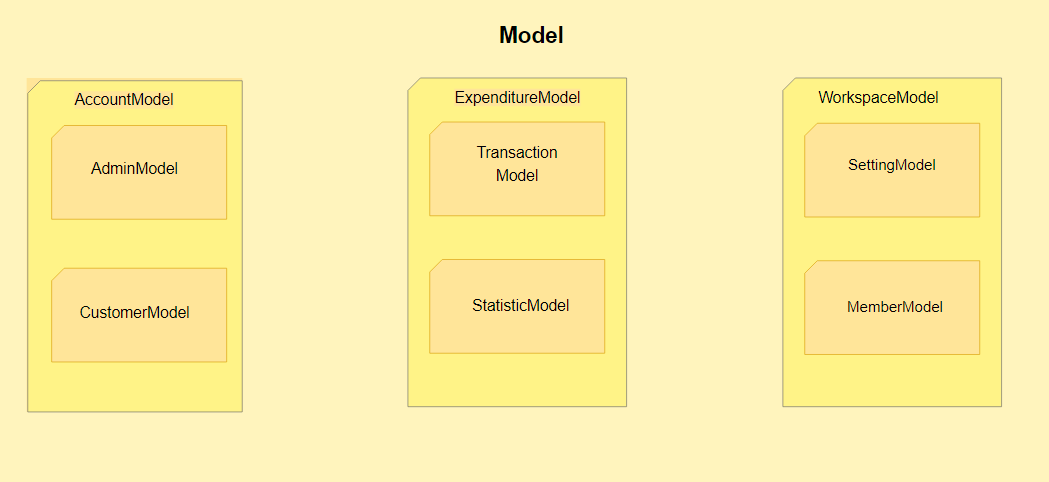




* Controller: routing and offering services for handling the request from the user by view then responding valid data to the user. We are using ExpressJS for routing and handling requests.



* View: displaying web layout to the user and being responsible for interacting with the user. We are using ReactJS to create the appearance of our website.

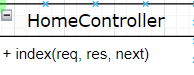


* Model: offering data models, interacting with databases and returning data to controllers. We are using a NoSQL database for modelling, Mongoose framework for interacting with MongoDB.

## Component: AccountController

Detail: redirecting to different routes.

Class diagram:



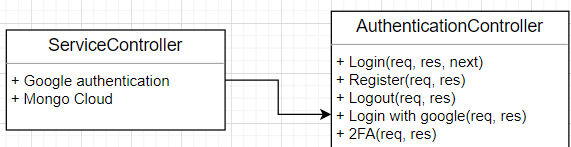
Function detail:

* index(req, res, next): responsible for redirecting to other controllers.

## Component: AuthenticationController

Detail: handling login, register, logout requests from the user.

Class diagram:



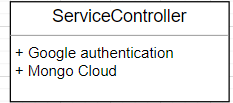
Function details:

* Login(req, res , next): allowing end-users for signing in with their existing accounts.
* Registes(req, res): allowing end-users for creating new accounts.
* Logout(req, res): allowing end-users for signing out from their accounts.
* Login with google(req, res): allowing end-users for signing in to the website using their existing accounts and creating new accounts if there are no records.
* 2FA(req, res): increasing security of account by registering through second factor authority.

## Component: ServiceController

Detail: handling services for third-party applications.

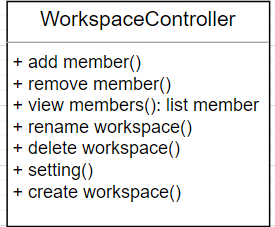
Class diagram:



Function details:

* Google authentication(): handling signing up with google service and export login object to the website.
* Mongo Cloud(): modelling data and interacting with cloud storage of MongoDB.

## Component: WorkspaceController



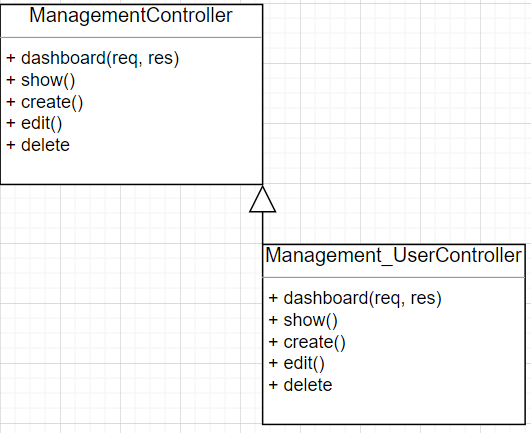
Detail: Workspace is a place for storing information about transactions. A workspace may contain several members.

* add member(): adding member to workspace by username.
* remove member(): removing existing members from workspace.
* view members(): exporting list of members in workspace.
* rename workspace(): renaming the workspace.
* create workspace(): creating another workspace.
* delete workspace(): deleting an existing workspace.
* setting(): modifying preset of workspace.

## Component: ManagementController

Detail: Handling requests from admin-user, returning special services for admin.

Class diagram



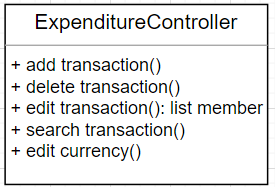
Details:

* ManagementController: a parent class, handling request from admin-user:
  + dashboard(req, res): exporting layout to admin-user to interact with.
  + show(req, res): handling requests of user information.
  + create(req, res): creating records.
  + edit(req, res): editing records.
  + delete(req, res): deleting records.
* Class Management\_UserController: derived class for handling requests of managing users.

## Component: ExpenditureController

Detail: Interacting with users, data and action like add, delete,…edit their personal data

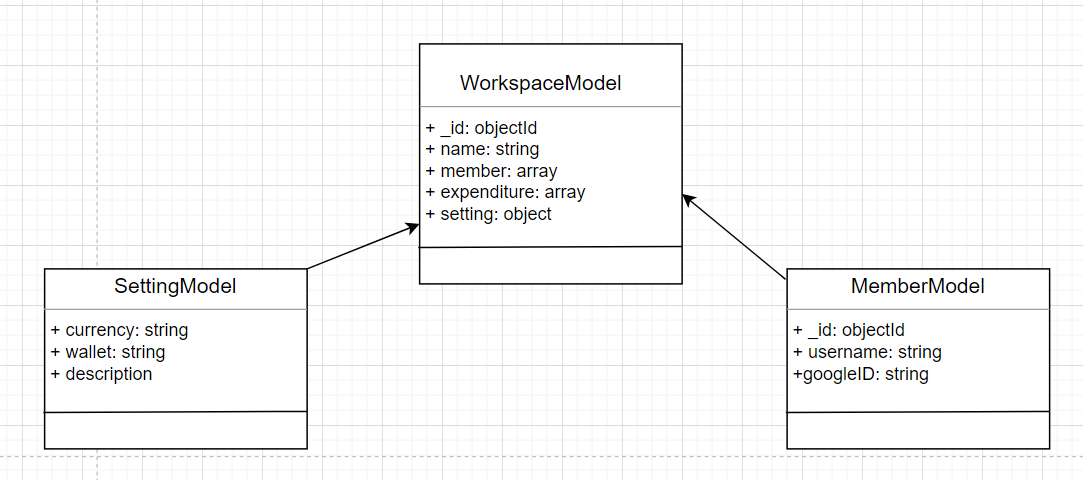
Class diagram:



Detail:

* Class này dùng các phương thức add, delete, edit, search các dữ liệu mà họ đang có với cơ sở dữ liệu thông qua model.
* add transaction(): adding new record of transaction to user workspace.
* delete transaction(): deleting existing record of transaction in user workspace.
* edit transaction():modify existing record of transaction in user workspace.
* search transaction(): searching existing record of transaction in user workspace.
* edit currency(): modifying currency
  1. **Component: WorkspaceModel**

Detail: Storing information of the user about transactions, setting, …

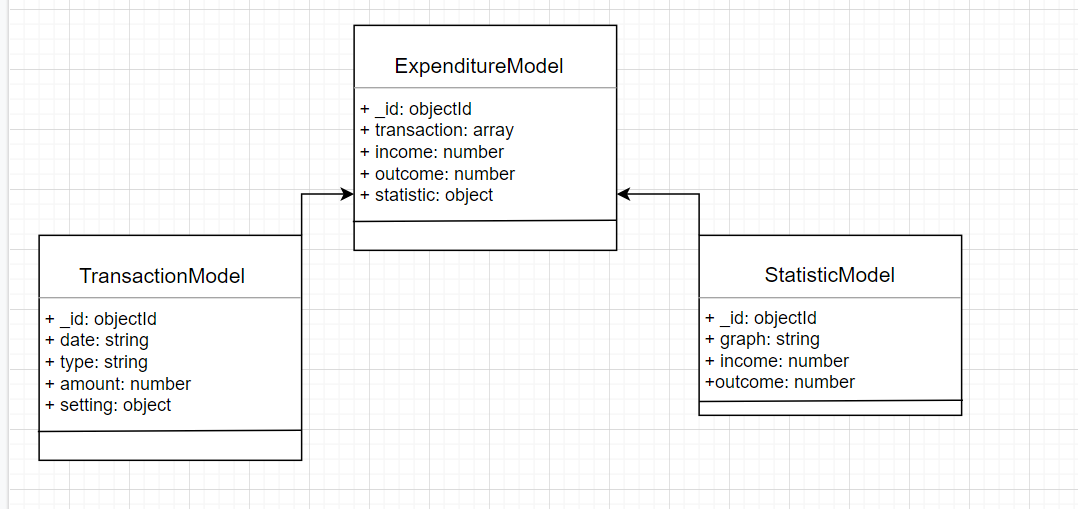


Detail:

* Class WorkspaceModel:Class for storing records of the user. Two derived classes for storing settings of the user.

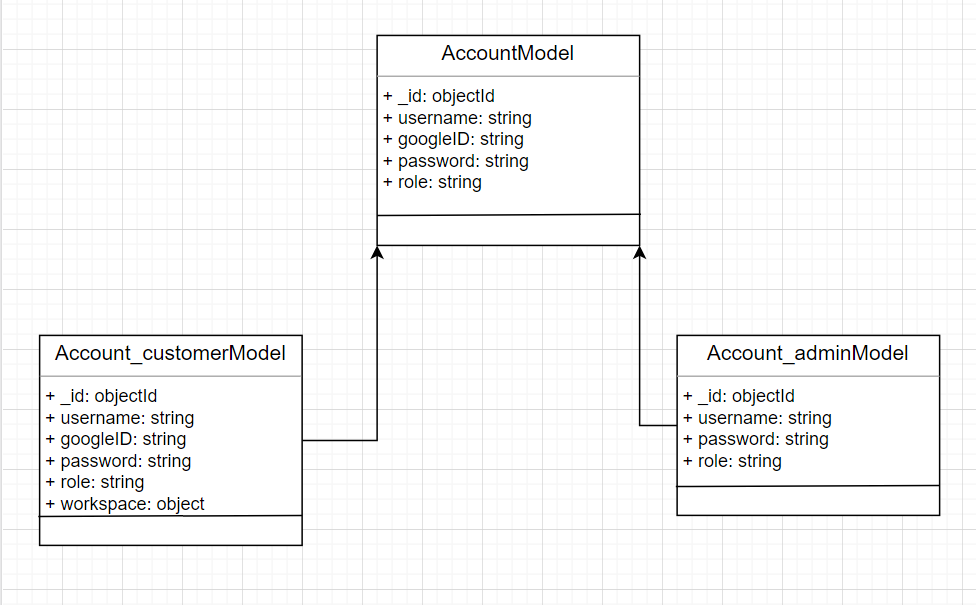
## Component: ExpenditureModel

Detail: Expenditure model is used for storing records of user’s transactions. Transaction model is used for storing information about transactions. Statistic models are used for visualising data.

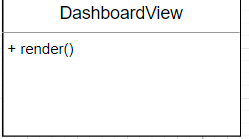


## Component AccountModel

Detail: Model for storing specific account information.

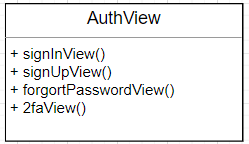


## Component: DashboardView



Detail: rendering display to users.

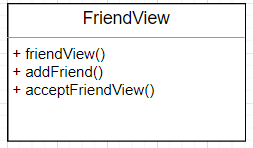
## Component: AuthView



Detail:

* signInView(): rendering sign in form.
* signUpView(): rendering sign up form.
* forgotPasswordView(): rendering forgot password form.
* 2FAView(): rendering 2FA form.

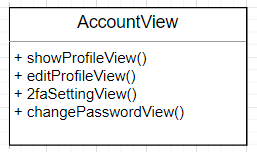
## Component: FriendView



Detail:

* friendView(): rendering friendlist.
* addFriend(): rendering add friend form.
* acceptFriendView(): rendering accept friend form.

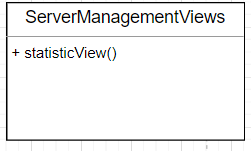
## Component: AccountView



Detail:

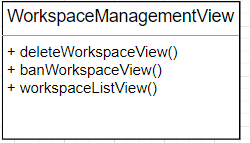
* showProfileView(): rendering user’s profile.
* edit Profile View(): rendering edit profile form.
* 2FASettingView(): rendering 2FA setting form.
* changePasswordView(): rendering change password form.

## Component: ServerManagementViews



Detail: rendering visualised data using graphs to users.

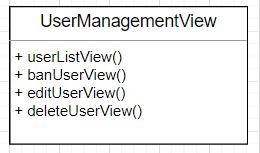
## Component: WorkspaceManagementView



Detail:

* deleteWorkspaceView(): rendering delete workspace form.
* banWorkspaceView(): rendering ban workspace form
* Workspace ListView(): rendering workspace List form

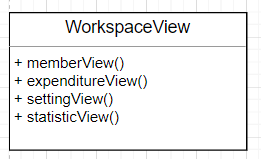
## Component: UserManagementView



Detail:

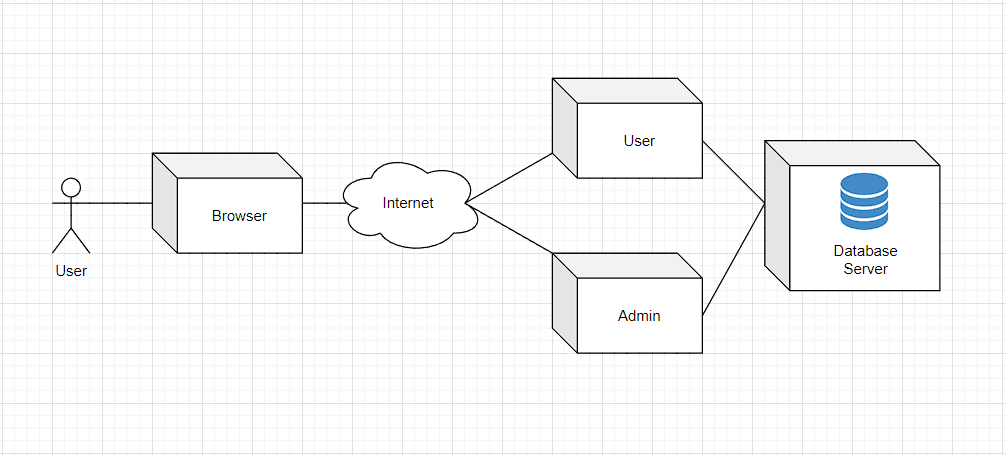
* userListView(): rendering user list form.
* banUserView(): rendering ban user form.
* editUserView(): rendering edit user form.
* deleteUserView(): rendering delete user form.

## Component: WorkspaceView



* memberView(): rendering member form.
* expenditureView(): rendering expenditure form.
* settingView(): rendering setting from.
* statiticView(): rendering of a statistical graph.

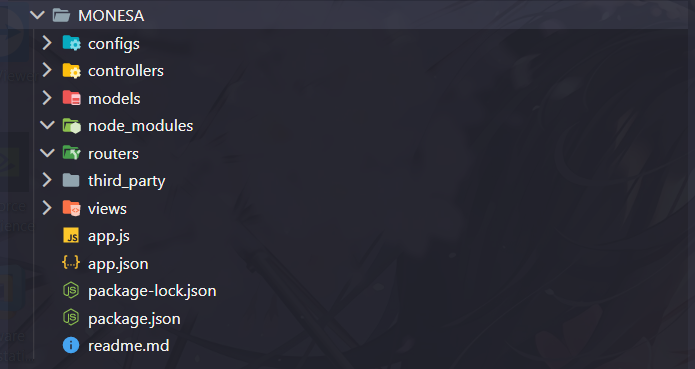
# Deployment

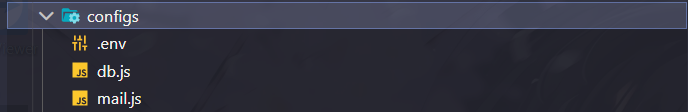
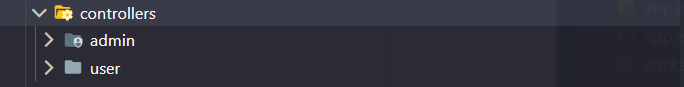
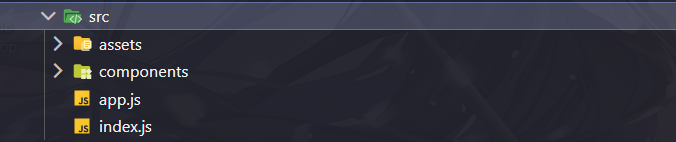
**

* Browser: End-users use browsing to get access to the website using the internet and send requests to the server.
* User: User controllers handle requests from the user, routing and providing services and interacting with the database (If there is CRUD operation).
* Admin:Admin controller handles requests from the admin, routing and providing services and interacting with the database (If there is CRUD operation).
* Database server: Storing users information and querying data.

# Implementation View

**Root folder structure:**



* **Config folder**: Including files (sub folders) used to configure database or mailer, etc.
  + **
  + File db.js stores information for connecting the database. File mail.js stores information for mailing.
* **Controllers folder**: Including files (sub folders) for handling requests from Views and interacting with Models for querying data and responding to Views and rendering to users.
  + Folder Admin and User are specified to handle requests from users with different roles.
  + **
* **Models folder**: Including files (sub folders) for defining schemas and models which are used to store user’s information and querying data.
  + File setting.js is used to create a model for website settings. File user.js is used to create a model for saving information of the user which are username, password, workspace, etc. File workspace.js is used to create a model for workspace containing information of workspace name, owner, transactions, etc.
  + **
* **Node\_modules folder**: Including packages of code for specific purposes which will help reduce effort in coding.
* **Routers folder**: include files (sub folders) for routing, bring user to right page to handle request from user and respond
* **Third\_party folder**: including file for functions with general purposes.
  + **
* **View folder**: Include files (sub folders) for creating website layout, being responsible for rendering the appearance of the website to the user to interact with.
  + There are:
  + **
    - **Build folder**: Including release app of React.
    - **Thư mục public**: Including logo, index.html.
    - **Thư mục src**: Including relevant files and folders related to front end work.
      * **
        + **Thư mục assets**: Including folder CSS, SCSS, IMG, ...
        + **File app.js**: Including class App.
        + **File index.js**: Using to define main router for Reactjs
        + **Thư mục components**:

**

**Thư mục admin**: Including admin files, folders

**Thư mục index**: Including homepage files, folders

**Thư mục user**: Including user files, folders

* **File app.js**: là file dùng để chứa các đoạn mã liên kết với các thư mục trên để chạy trang web.
* **File app.json**: Using to save pm2 config (Coming soon).
* **File package-lock.json**: Auto initialised when using npm init.
* **File package.json**: Storing information of project: creator, scripts, modules.
* **File readme.md**: use to note information of project.