GROUP 10

TUS
Software Architecture Document

Version 1.2

Revision History

Date	Version	Description	Author
20/11/2023	1.0	Work on Introduction, Architectural Goals and Constraints, Use-case Model	Ly Nhat Hao Huynh Son Ha Trinh Xuan Bach
25/11/2023	1.1	Work on Logical View	Ly Nhat Hao Tran Hoang Duy Nguyen Quang Thai
01/12/2023	1.2	Review all parts	All members

Table of Contents

1. Introduction **4**
1.1. Purpose 4
1.2. Scope 4
1.3. Definitions, Acronyms, Abbreviations 4
1.4. Reference 4
1.5. Overview 4
2. Architectural Goals and Constraints **5**
3. Use-Case Model **6**
4. Logical View: **7**
4.1. Package: Manage User (User panel) 9
4.2. Package: Manage Course (User panel) 10
4.3. Package: Manage Request (User panel) 11
4.4. Package: Manage FavList (User panel) 12
4.5. Package: Manage Upload (User panel) 13
4.6. Package: Manage Checkout (User panel) 14
4.7. Package: Manage Category (User panel) 15
4.8. Package: Manage User (Admin Panel) 17
4.9. Package: Manage Course (Admin Panel) 18
4.10. Package: Manage Report User (Admin Panel) 19
4.11. Package: Manage Ban User (Admin Panel) 20
4.12. Package: Manage Course (Tutor Panel) 21
4.13. Package: Manage Order (Tutor Panel) 22
5. Deployment **23**
6. Implementation View **23**

Software Architecture Document

1. Introduction

This document presents a comprehensive outline and elucidates the entirety of the Process Specification Tool (PST)'s architecture. It delineates the process by which an online user can generate and manage software development process definitions, while also delving into the underlying architecture of the tool.

The document offers a broad depiction of the architecture's objectives, encompassing the supported use cases and the chosen architectural styles and components that effectively fulfill these use cases. By establishing this framework, it becomes possible to formulate the design criteria and create detailed technical and domain standard documents that define the standards in depth.

1.1. Purpose

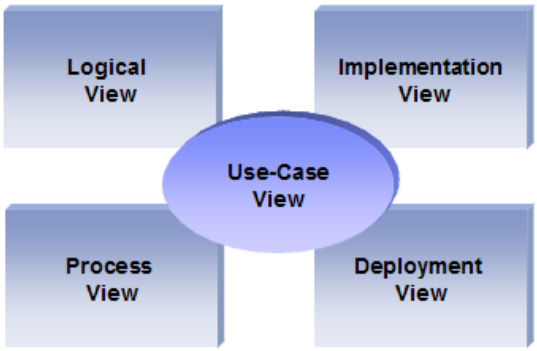
The primary objective of a Software Architecture Document (SAD) is to deliver a lucid and succinct portrayal of the architecture of a software system to all involved parties, comprising developers, testers, project managers, and customers. Through the documentation of the architecture, the Software Architecture Document (SAD) guarantees a unified comprehension among all participants in the development process regarding the system, its constituent elements, and their interconnectedness.

1.2. Scope

The Software Architecture Document (SAD) offers a thorough and encompassing insight into **TUS** - A high-quality tutor search website. It employs multiple architectural views to portray diverse facets of the system. The document aims to capture and effectively communicate the pivotal architectural decisions that have been carefully incorporated into the system's design.

1.3. Definitions, Acronyms, Abbreviations

- SAD - Software Architecture Document
- API - Application Programming Interface
- UI - User Interface
- DBMS - Database Management System
- MVC - Model-View-Controller
- SOA - Service-Oriented Architecture
- REST - Representational State Transfer
- AWS - Amazon Web Services
- JS – JavaScript



1.4. Reference

None

1.5. Overview

After summarizing the architectural representation, goals and constraints, this document describes the system using several architectural views (Use Case Model, Logical View, Deployment, Implementation View, and data).

2. Architectural Goals and Constraints

- **Server side**
  All communication with clients will comply with public HTTP communication protocol standards.
  TUS will be hosted by Docker on the Azure Linux Server platform.
  Storage is based on AWS S3.
  Database
  - MySQL will be hosted by Docker on Azure Linux Server.
  - MongoDB will be hosted by Atlas Cloud.
- **Client Side**
  - Admins and Users are required to use a modern web browser, the latest version of Google Chrome or Safari.

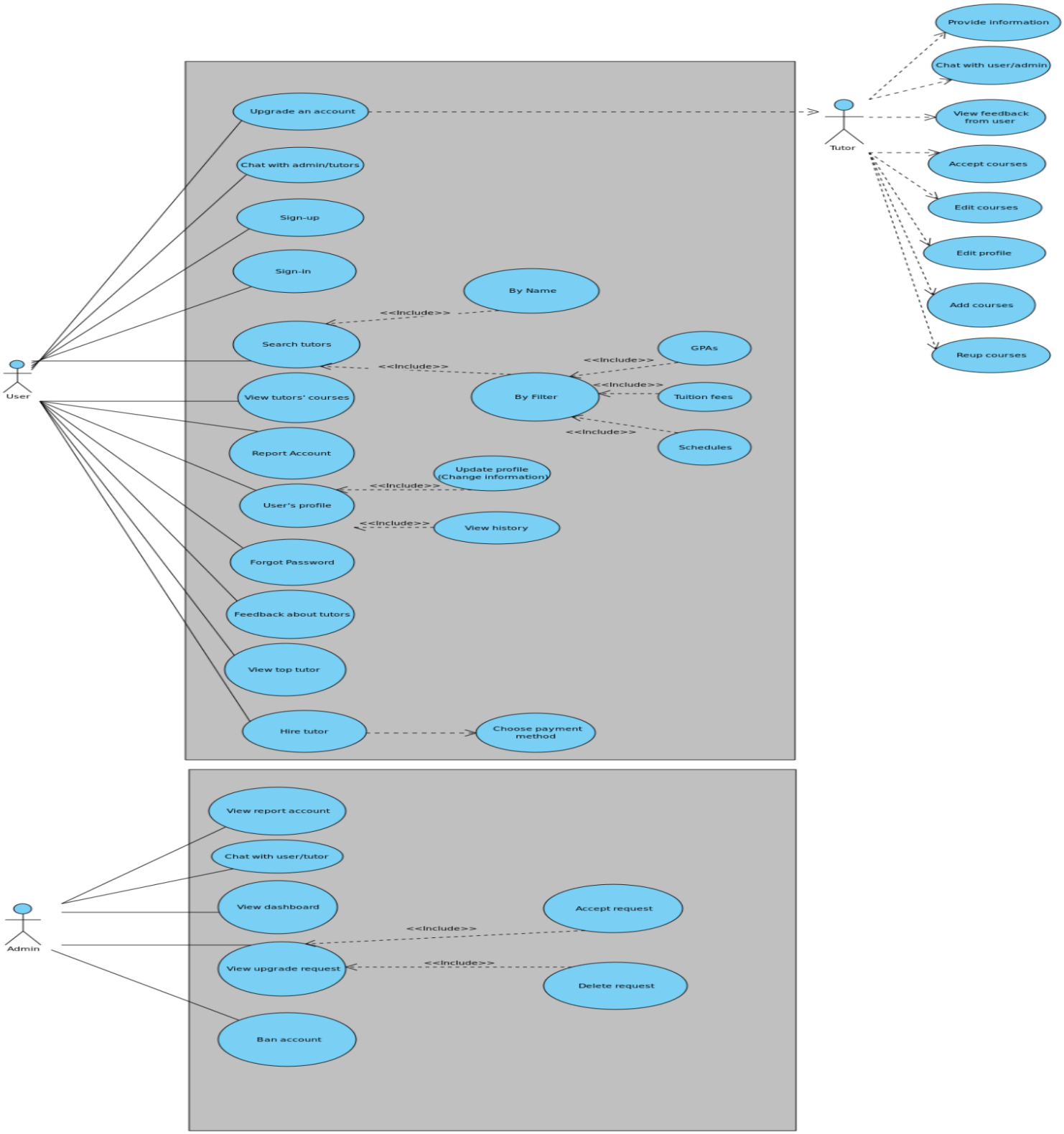
- **Environment**
  Website

- **Programming languages and frameworks**
  NodeJS, MongoDB, CSS, HTML, ReactJS, JavaScript, …

- **Security**
  Passwords must be encrypted before storing in the database.
  Only Admin can remove and ban user accounts.

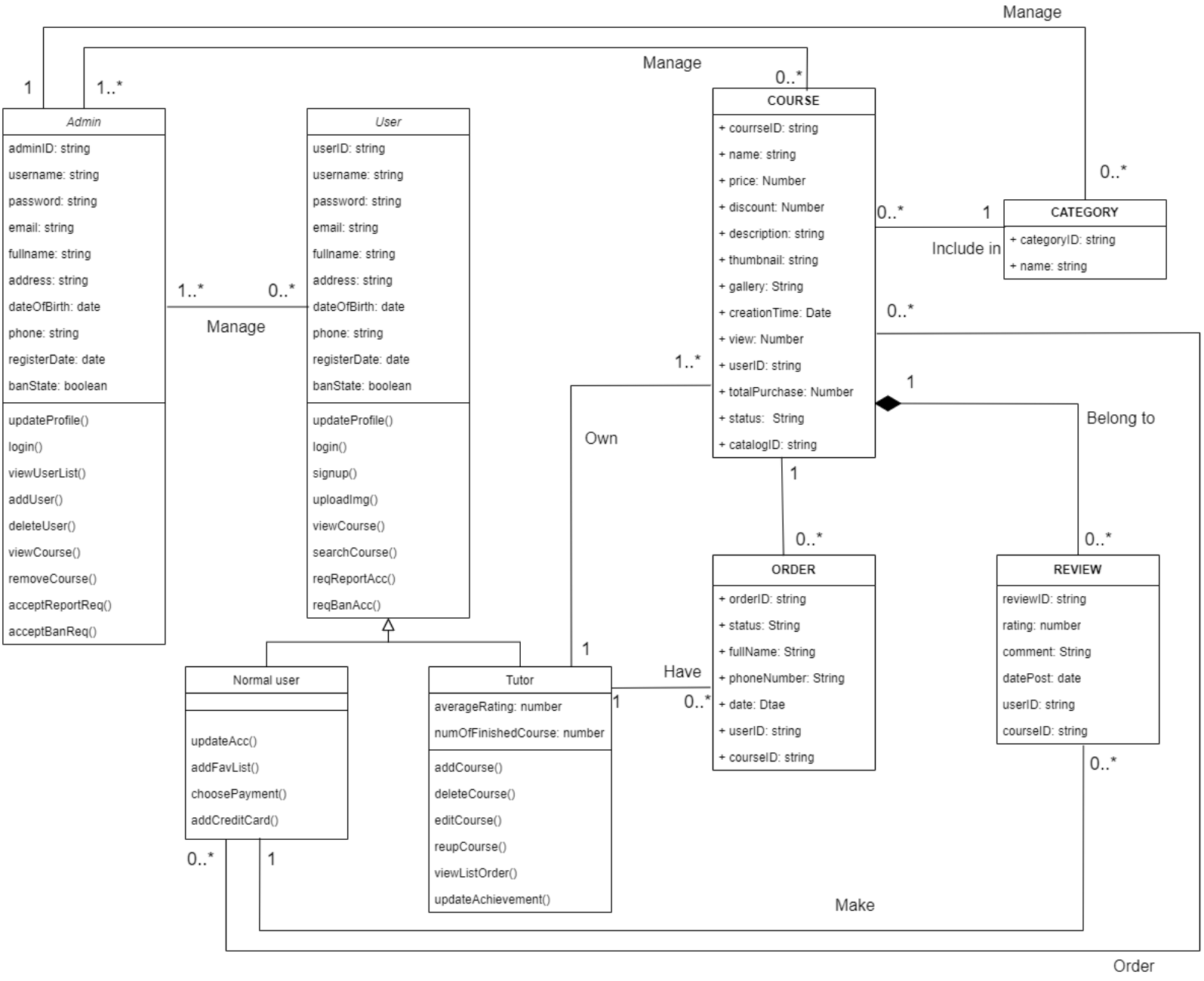
- **Performance**
  It is anticipated that the system should respond to any request well 0.00001s
  In addition, upload / download times can depend on data size which in turn depends on user input.

3. Use-Case Model

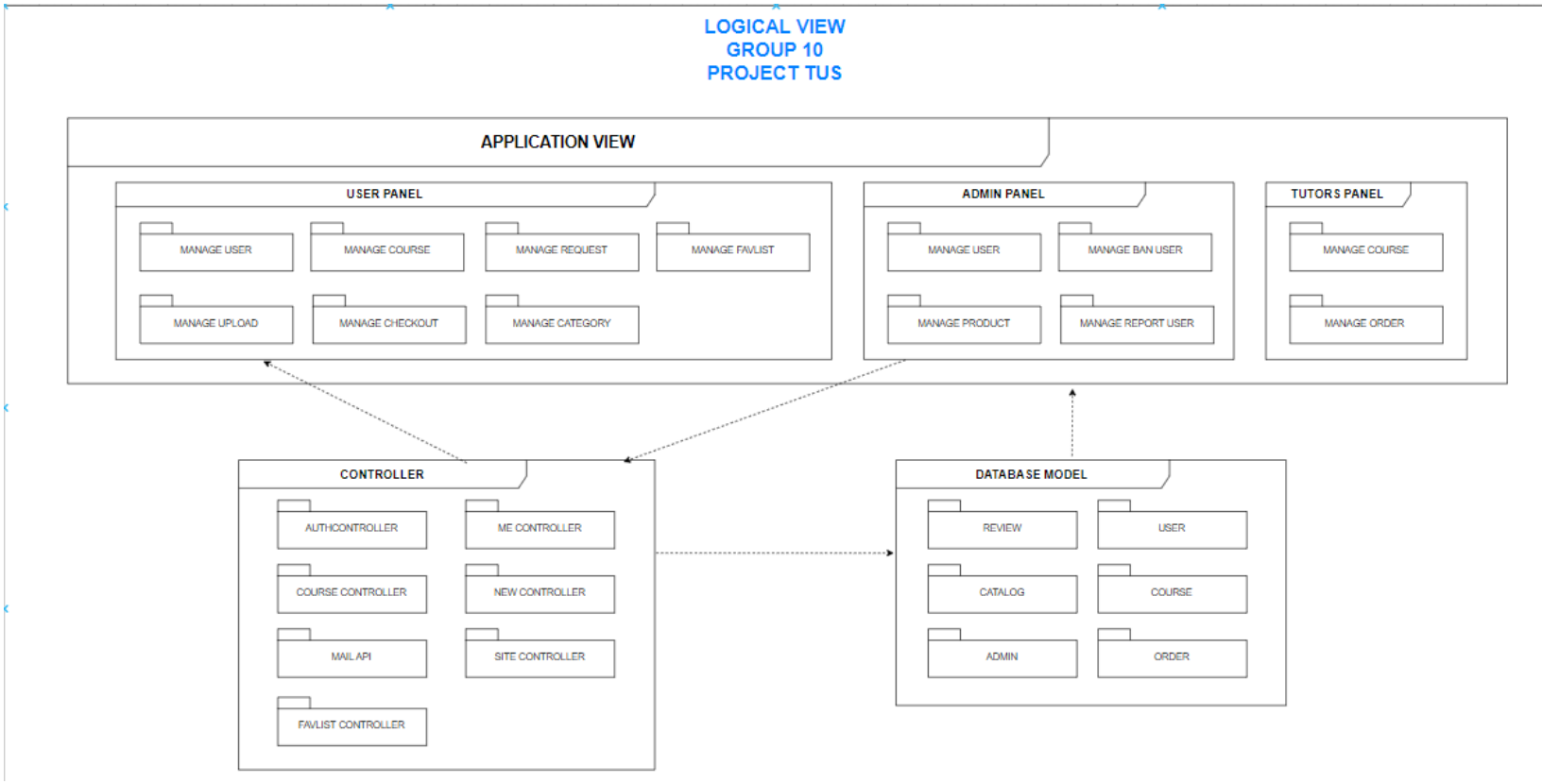


4. Logical View:

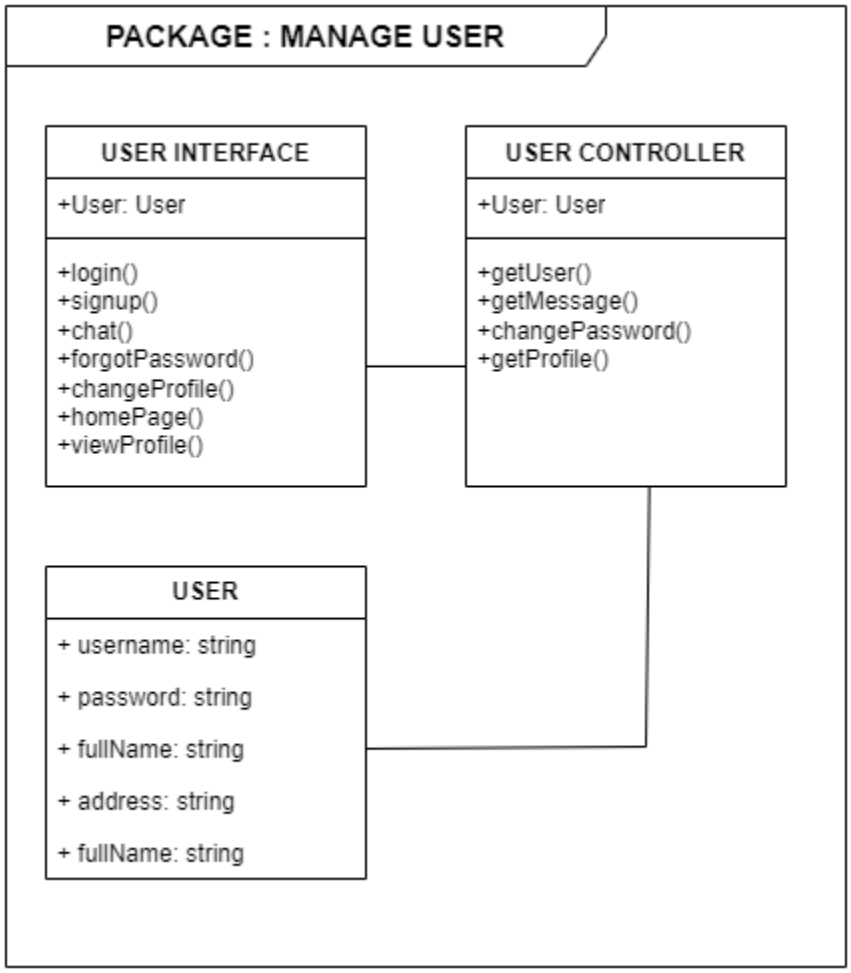
Class diagram:



Logical view:

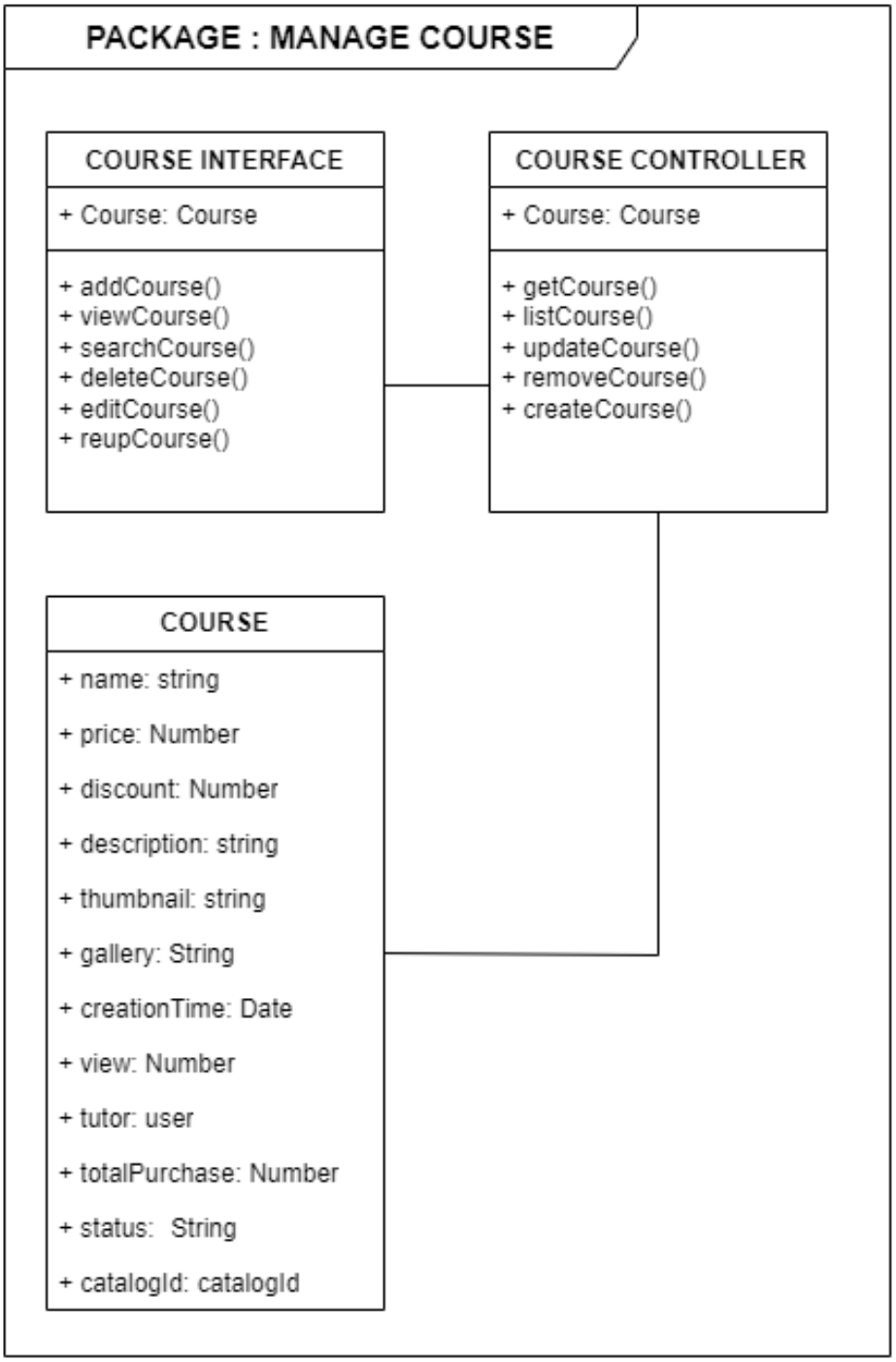


4.1. Package: Manage User (User panel)



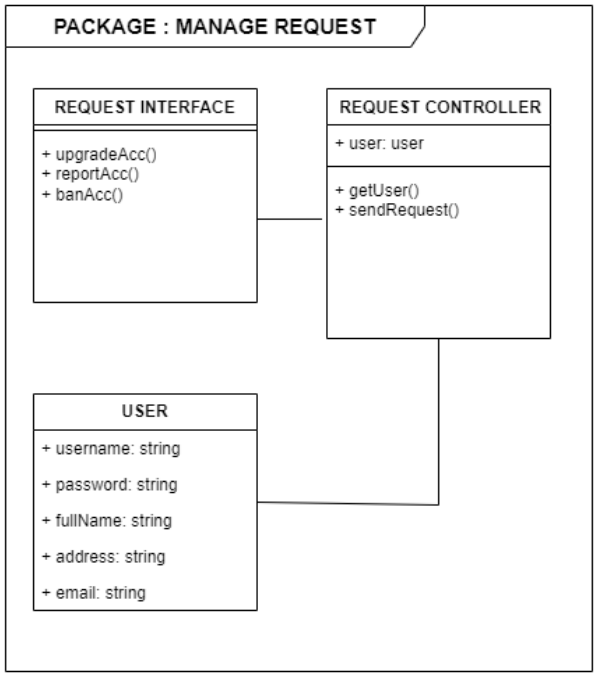
- **Class User Interface** is used to manage all user's functions from Interface.
    - Operations in this class
        - login(): sign in to system.
        - signup(): register as new user.
        - chat(): ask questions to Admin/Moderator/Seller
        - forgotPassword() : request to access to account by re-set password from email.
        - changeProfile() : update profile information.
        - viewProfile(): view our profile or seller's profile.
        - Homepage(): access to home page after access successfully to account.
- **Class User** is used to store user's information.
- **Class User Controller** is used by system controller - manage requests which are sent from Client side.
    - Operations in this class
        - getUser(): get user's information from database.
        - getMessage(): send/receive message from others
        - changePassword(): update password from users.
        - getProfile(): get information of users.

4.2. Package: Manage Course (User panel)



- **Class Product Interface** is used to manage all user's functions about the product from Interface.
    - Operations in this class
        - viewCourse(): to view a Course information.
        - searchCourse(): to find a Course by name, category,…
- **Class Course** is used to store Course's information.
- **Class CourseController** is used by system controller to manage requests which are sent from Client side on the product.
    - Operations in this class
        - getCourse): to get date of a Course (for viewProduct() of user).
        - listCourse(): to get data of a list of Course (for viewProduct() of user).

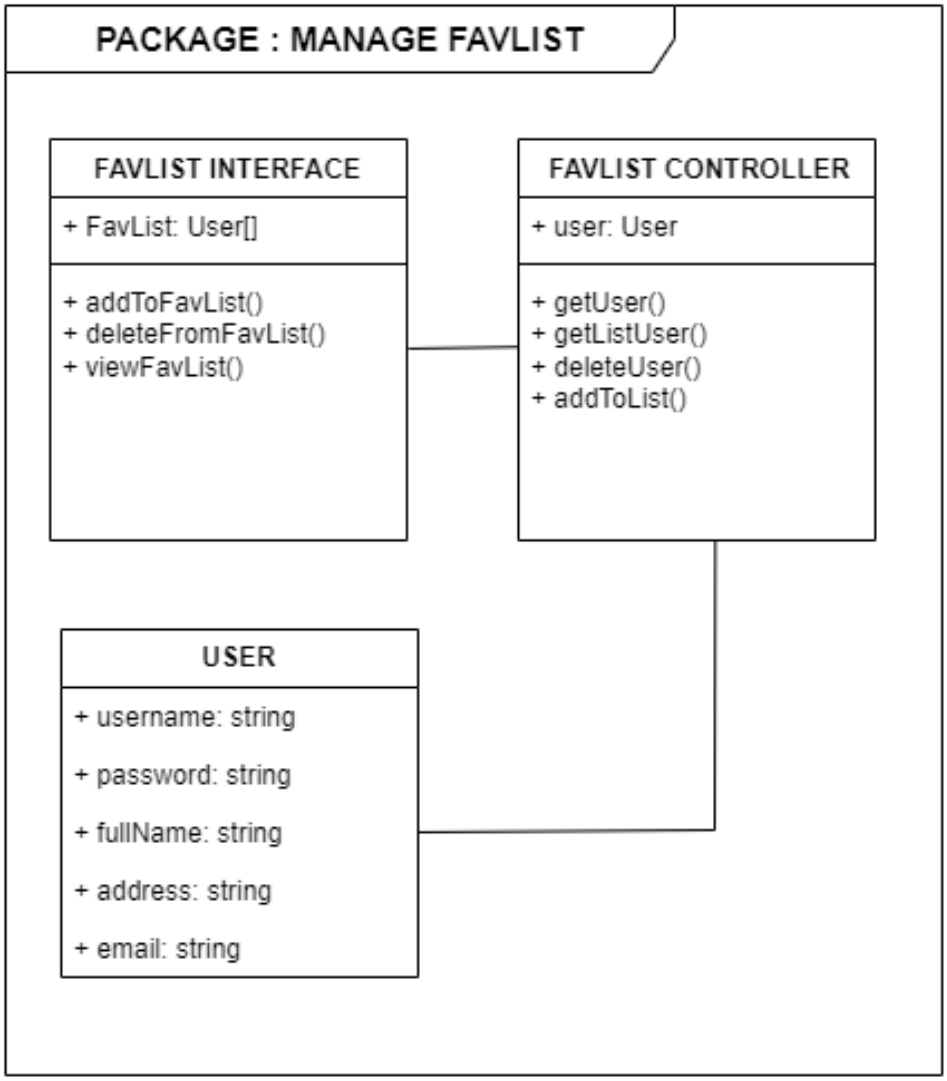
4.3. Package: Manage Request (User panel)



- **Class Request Interface** is used to manage all user's functions about the request from Interface.
    - Operations in this class
        - upgradeAcc(): to send request to upgrade account to open store for selling.
        - reportAcc(): to send request to report an illegal account.
        - banAcc(): to send request to ban an illegal account.
- **Class User** is used to store user's information.
- **Class Request Controller** is used by system controller to manage requests which is sent from Client side on the request.

- Operations in this class
  - getUser(): to get user information.
  - sendRequest(): to send request to report/ban account to Website for Admin/Moderator to handle.

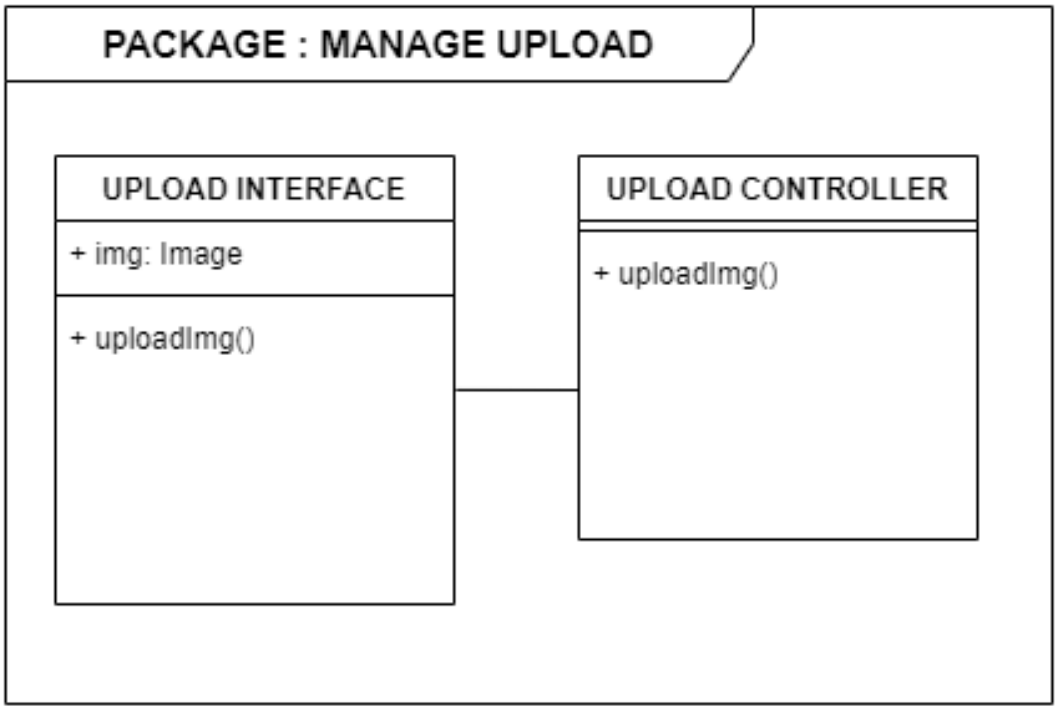
4.4. Package: Manage FavList (User panel)



- **Class favList Interface** is used to manage all user's functions about the favList from Interface.
  - Operations in this class
    - addToFavList(): to add a product to the favList.
    - viewFavList(): to view all products stored in the favList.
    - deleteFromFavList(): to delete a product from favList.
- **Class User** is used to store product's information.
- **Class FavList Controller** is used by the system controller to manage requests which are sent from Client side on the FavList.
  - Operations in this class
    - getUser(): to get information about a User.

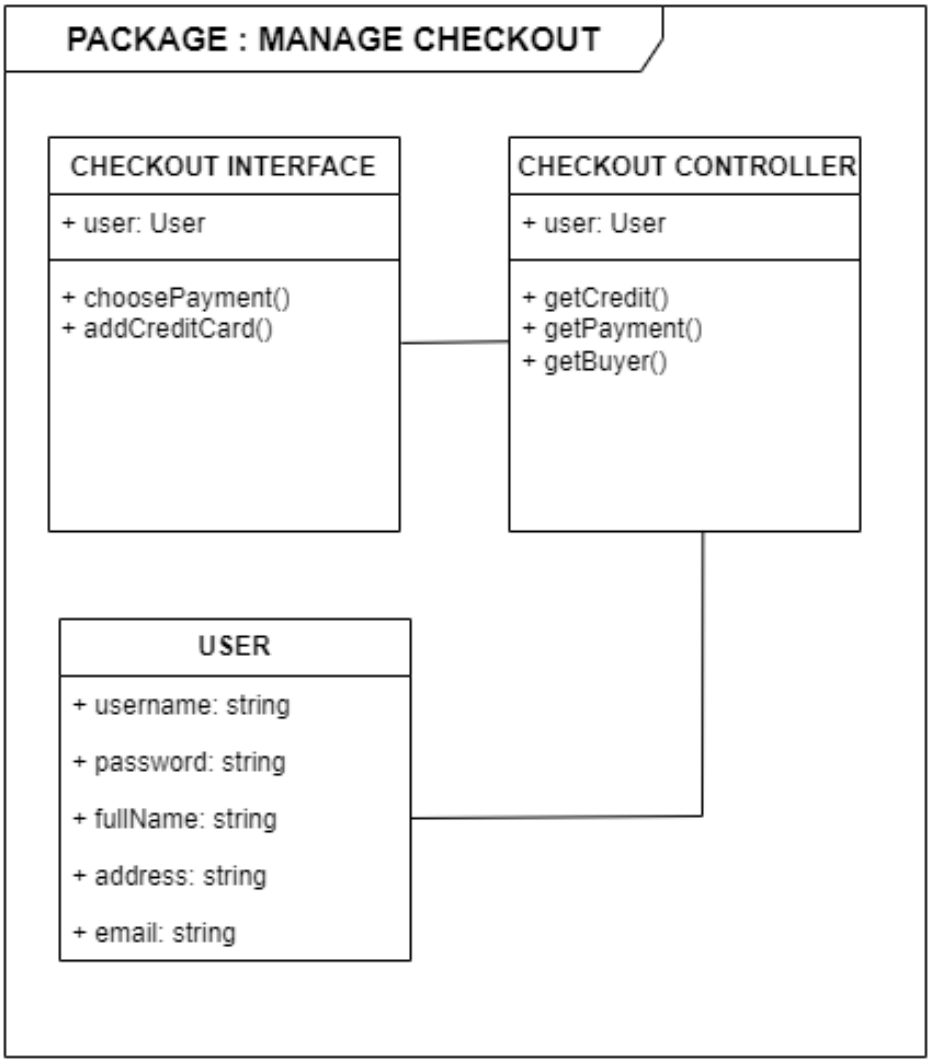
- getListUser(): to get data of a list of User on the FavList.
- addToList(): to add a User to the FavList.
- deleteUser(): to remove User information from FavList.

4.5. Package: Manage Upload (User panel)



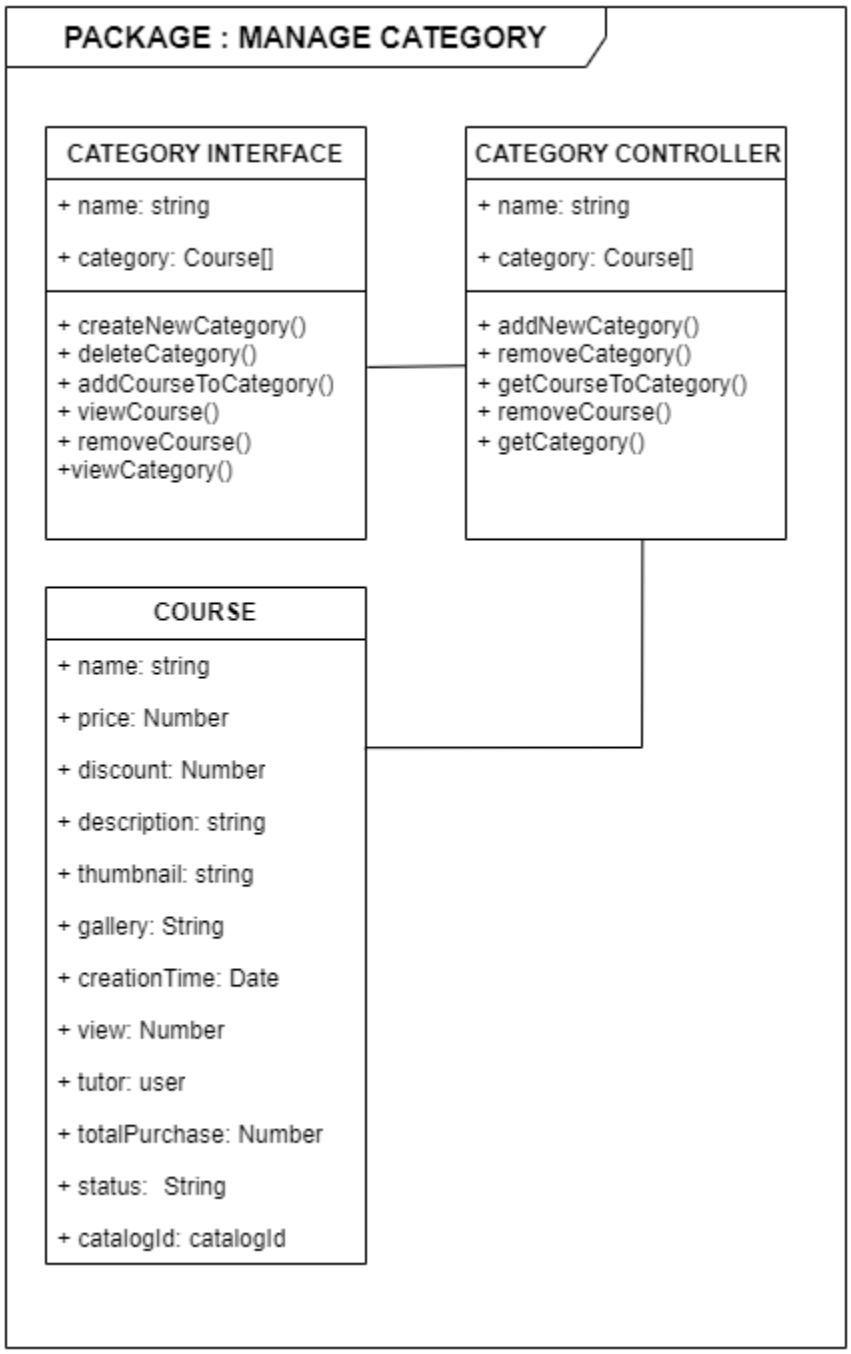
- **Class Upload Interface** is to manage Image which is used to represent product from user interface.
- **Class Upload controller** to manage request of uploading image from user.

4.6. Package: Manage Checkout (User panel)



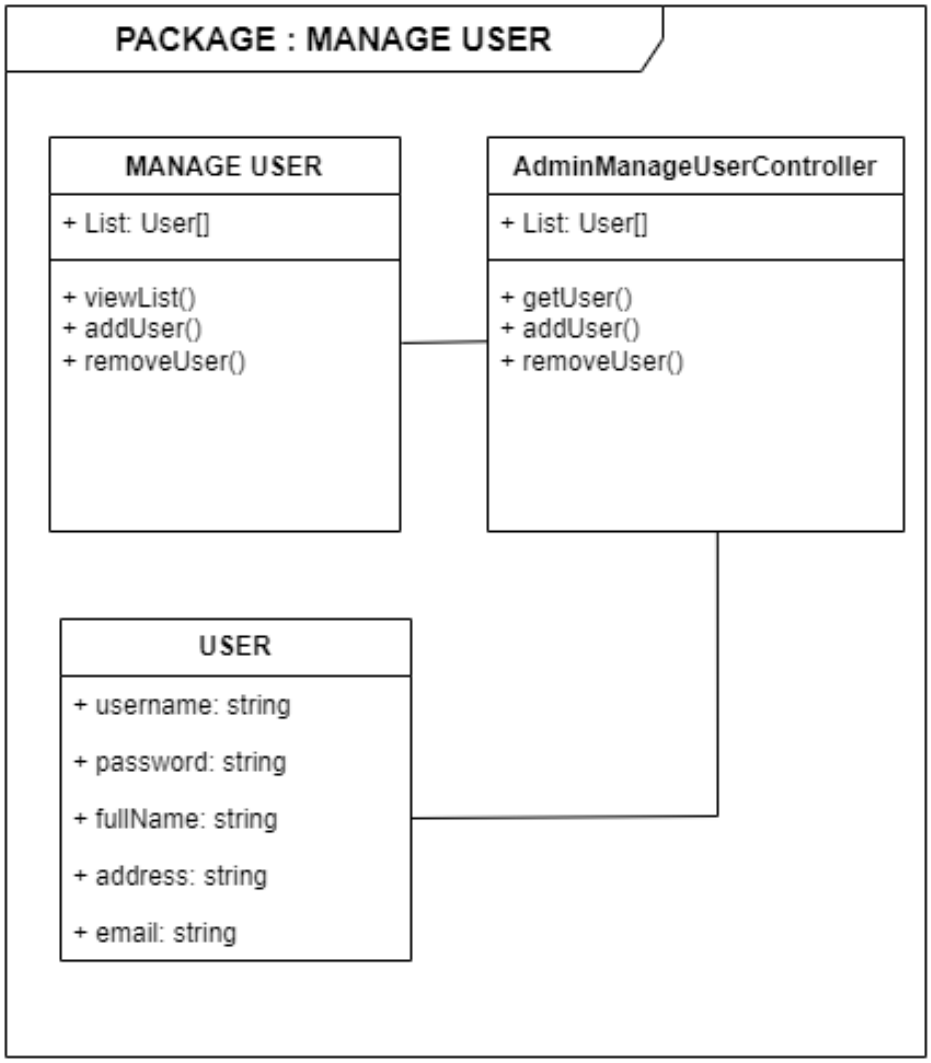
- **Class Checkout Interface** is used to manage all user's functions about the checkout from Interface.
  - Operations in this class
    - choosePayment(): to choose method which is used to make payment.
    - addCreditCard(): to add credit for payment.
- **Class User** is used to store user's information.
- **Class Checkout Controller** is used by system controller to manage requests which are sent from Client side on the checkout.
  - Operations in this class
    - getCredit(): to get credit card for payment method.
    - getPayment(): to confirm payment successfully or not.
    - getBuyer(): to get bookerr information and store order with them.

4.7. Package: Manage Category (User panel)



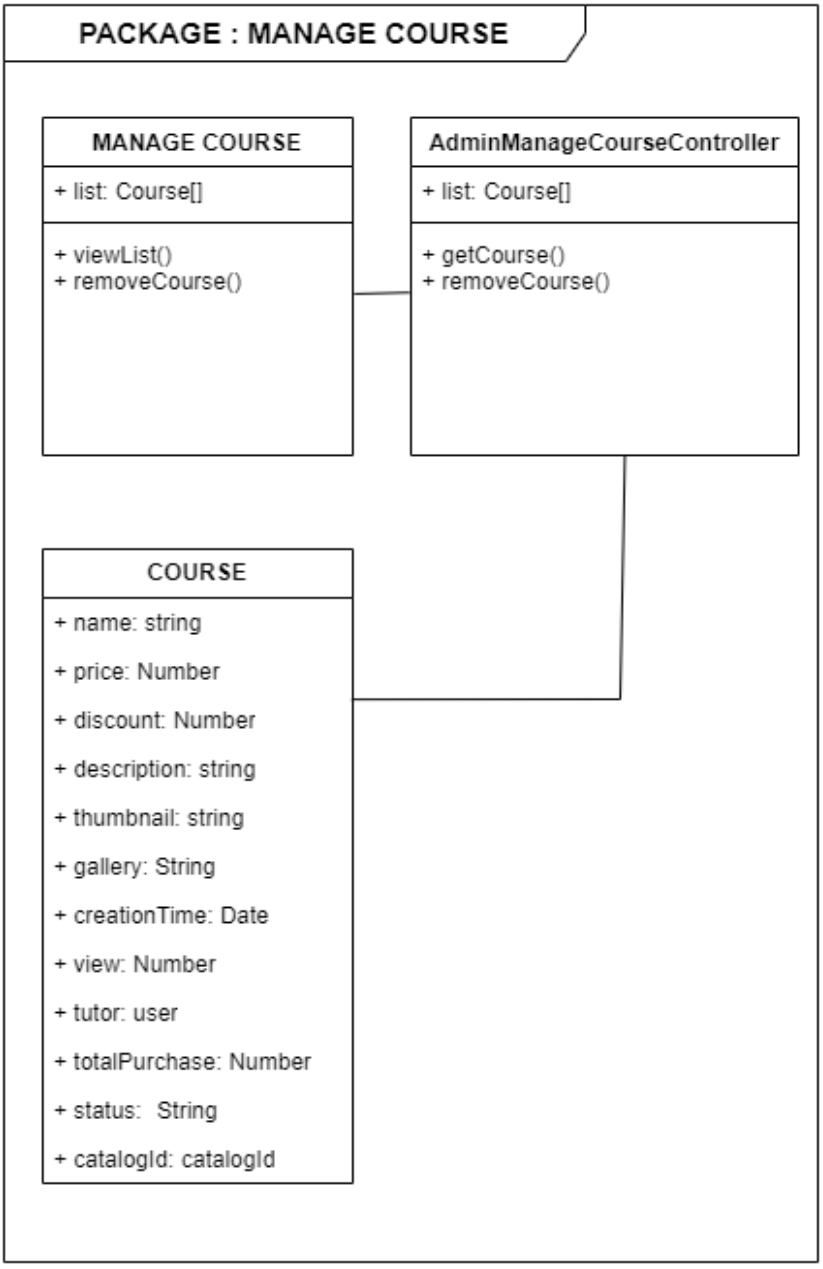
- **Class Category Interface** is used to manage all user's functions about the package from Interface.
    - Operations in this class
        - createNewCategory(): to create a new category of course..
        - deleteCategory(): to delete an existing category.
        - addCourseToCategory(): to add an existing/new course to an existing category.
        - viewCourse(): to view product information in a category.
        - removeCourse(): to remove a course from category.
        - viewCategory(): to view all courses in a category.
- **Class Course** is used to store course's information.
- **Class Category Controller** is used by system controller to manage requests which are sent from Client side on the category.
    - Operations in this class
        - addNewCategory(): to add a new category information to DB.
        - removeCategory(): to remove an existing category information from DB.
        - getCourseToCategory(): to add product information to an existing category.
        - removeCourse(): to remove product information from an existing category.
        - getCategory(): to get all course information of a category.

4.8. Package: Manage User (Admin Panel)



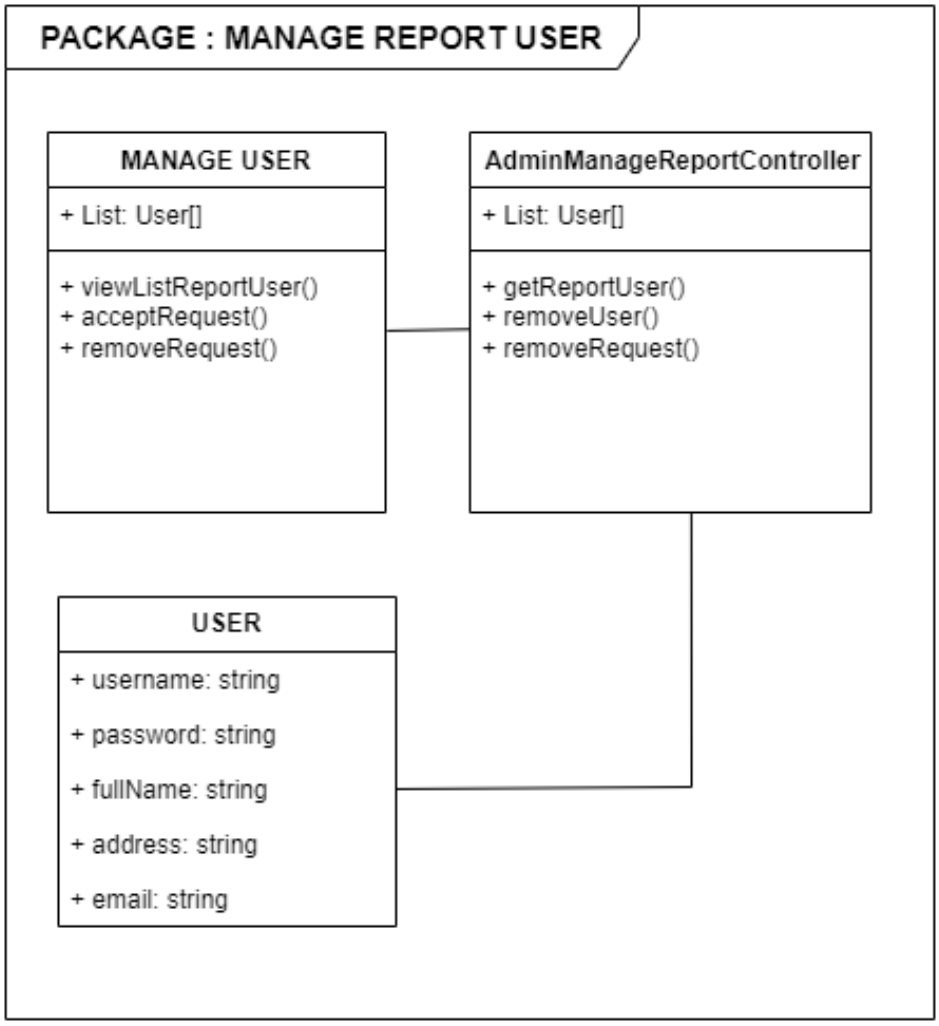
- **Class Manage User** is used to manage all user from
    - Operations in this class
        - viewList(): Admin view all users that valid in the system.
        - addUser(): Admin add user to the system.
        - removeUser(): If account is no longer valid or violate rules, Admin remove those.
- **Class User** is used to store user's information.
- **Class AdminManageUserController** is used by system controller to manage requests from Client side.
    - Operations in this class
        - getUser(): get user's information from database.
        - addUser(): to add new user information to database.
        - removeUser() : to user information from database.

4.9. Package: Manage Course (Admin Panel)



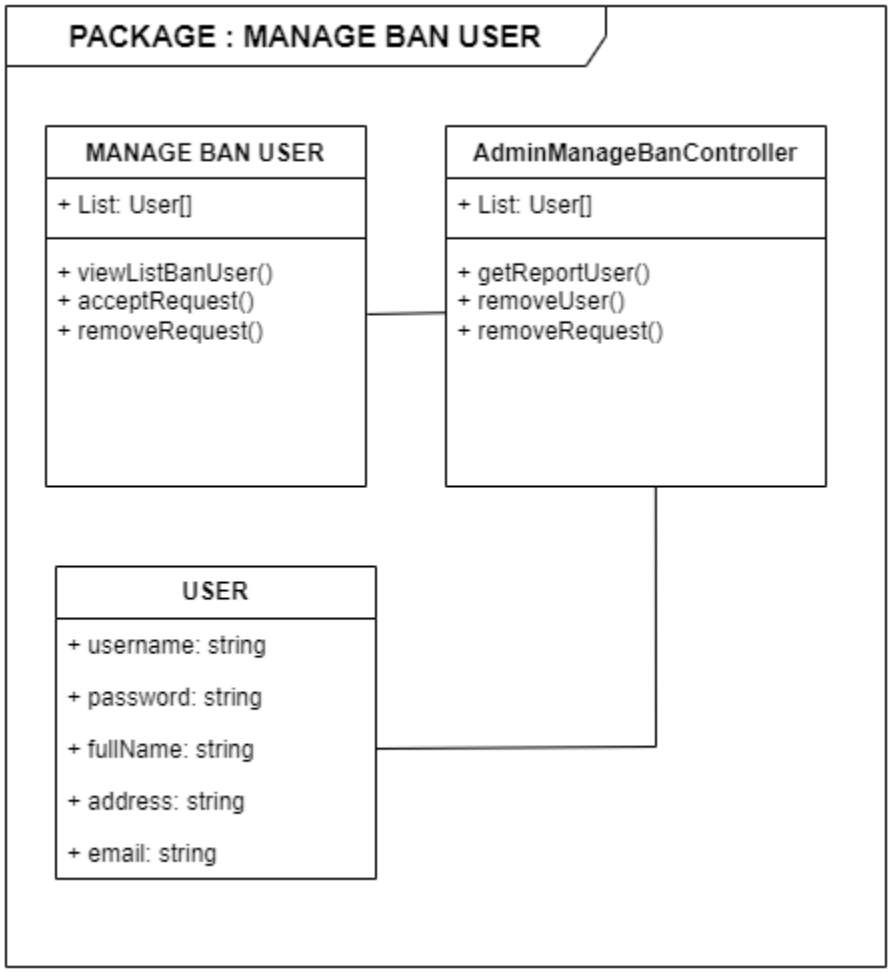
- **Class Manage Course** is used to manage all Course in the system from Admin side.
    - Operations in this class
        - viewList(): view all valid Course.
        - removeCourse(): remove illegal Course from database.
- **Class Course** is used to manage Course information.
- **Class AdminManageCourseController** is used to handle all requests from Admin side.
    - Operations in this class
        - getCourse(): get Course information from database.
        - removeCourse(): remove Course from database.

4.10. Package: Manage Report User (Admin Panel)



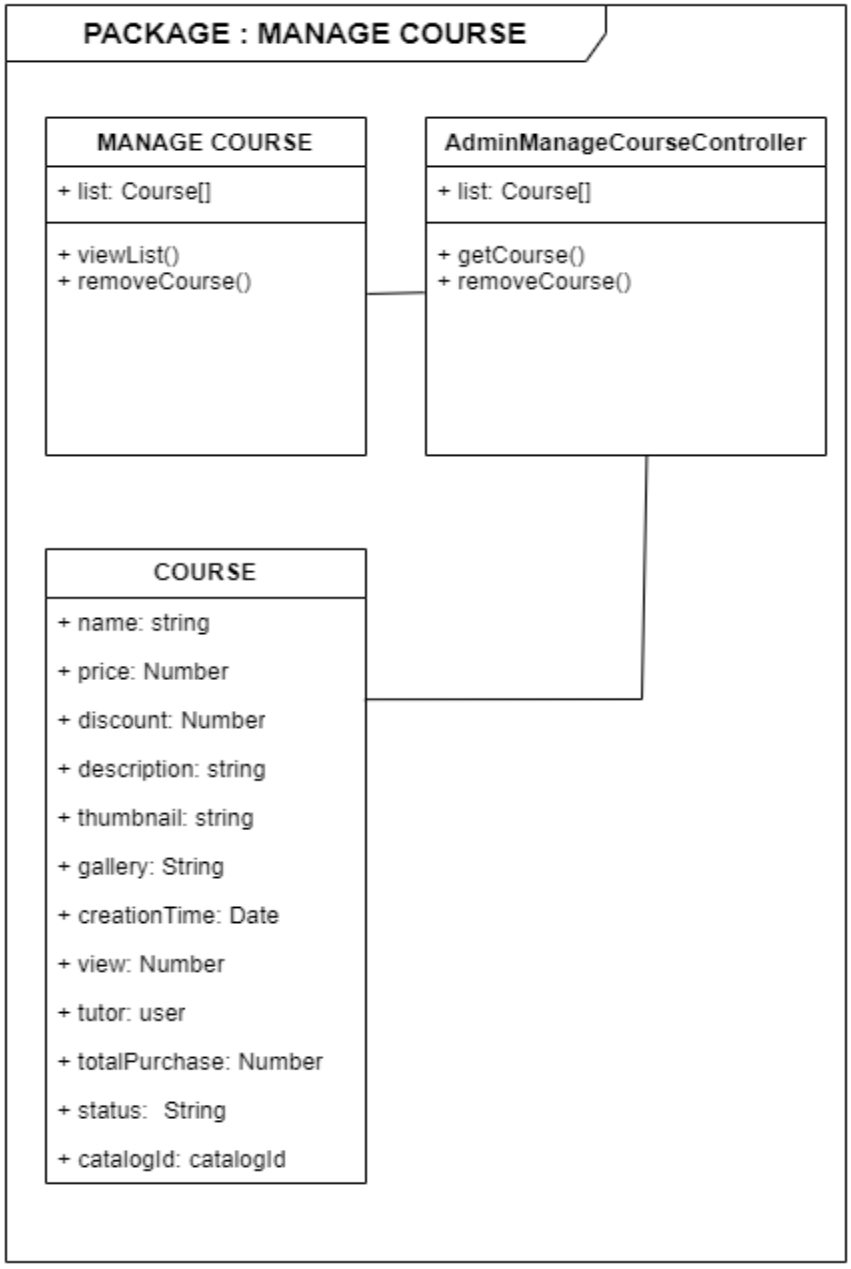
- **Class Manage Report User** is used to manage all report users in the system from Admin side.
    - Operations in this class
        - viewListReportUser(): view all users that reported.
        - acceptRequest(): accept the request of reporting that account.
        - removeRequest(): remove the request of reporting that account.
- **Class User** is used to manage user information.
- **Class AdminManageReportController** is used to handle all requests from Admin side.
    - Operations in this class
        - getReportUser(): get report user information from database.
        - removeUser(): remove user information from database if acceptRequest() is requested.
        - removeRequest(): remove request from database if removeRequest() is requested

4.11. Package: Manage Ban User (Admin Panel)



- **Class Manage Ban User** is used to manage all ban users in the system from Admin side.
    - Operations in this class
        - viewListBanUser(): view all users that reported.
        - acceptRequest(): accept the request to ban that account.
        - removeRequest(): remove the request to ban that account.
- **Class User** is used to manage user information.
- **Class AdminManageBanController** is used to handle all requests from Admin side.
    - Operations in this class
        - getReportUser(): get banned user information from database.
        - removeUser(): remove user information from database if acceptRequest() is requested.
        - removeRequest(): remove request from database if removeRequest() is requested

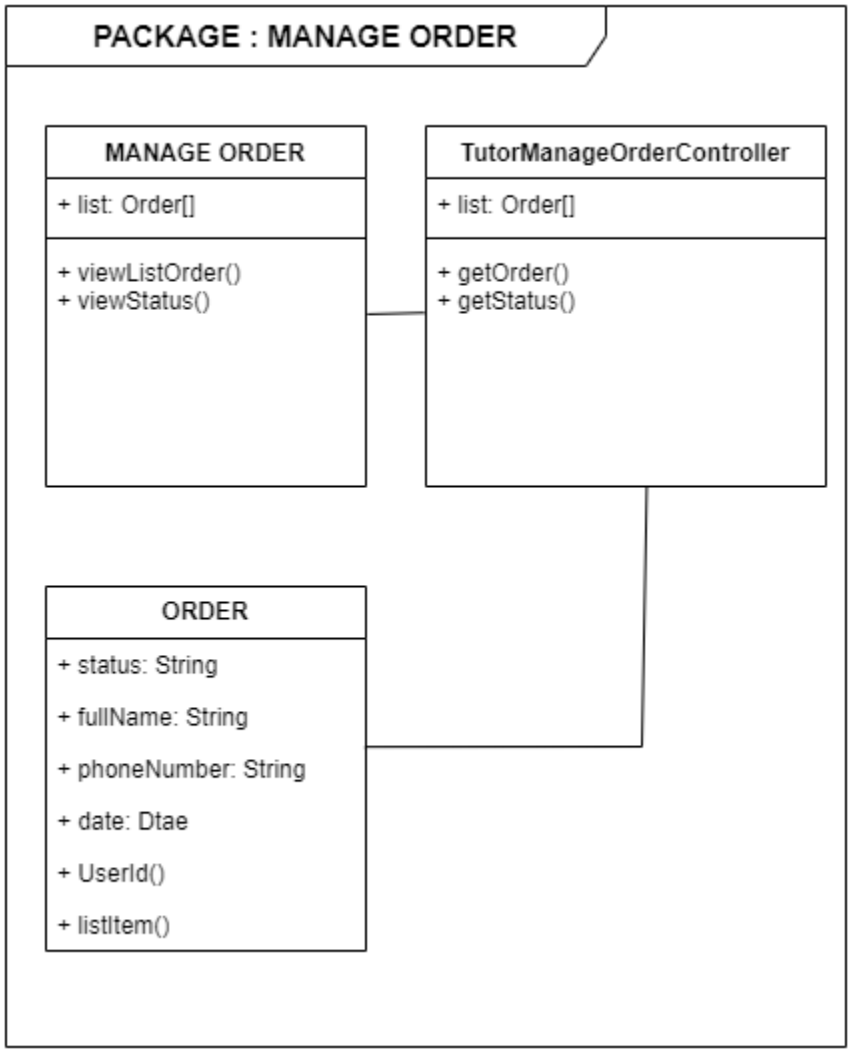
4.12. Package: Manage Course (Tutor Panel)



- **Class Manage Course** is used to manage all products in the system from the Tutor side.
  - Operations in this class
    - addCourse(): to add a new Course to the store.
    - viewCourse(): to view a Course information.
    - searchCourse(): to find a Course by name, category,…
    - deteleCourse(): to remove a Course from store.
    - editCourse(): to change Course information.
    - reupCourse(): to reup a existing Course

- **Class Course** is used to manage Course information.
- **Class TutorManageCourseController** is used to handle all requests from the Tutor side.
    - Operations in this class
        - getCourse(): get product information from database.
        - removeCourse(): remove product from database.

4.13. Package: Manage Order (Tutor Panel)



- **Class Manage Order** is used to manage all products in the system from Tutor side.
    - Operations in this class
        - viewListOrder(): view all orders.
- **Class Order** is used to manage order information.
- **Class TutorManageOrderController** is used to handle all requests from Tutor side.
    - Operations in this class
        - getOrder(): get Course information from database.
        - getStatus(): get order status

5. Deployment

6. Implementation View