Group01

Pic2Model
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
21/11/2024	1.0	Draft the software architecture document (SAD) by filling in all the sections in the document using the provided template. The document will provide and explain key components and how these components are organized to form the architecture. We will try to detail all the sections in the document with all the available information we have at this time.	All group members

Table of Contents

Software Architecture Document

1. Introduction

The introduction of the **Software Architecture Document** for project Pic2Model provides an overview of the entire **Software Architecture Document**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **Software Architecture Document**.

1.1 Purpose

The purpose of this document is to outline the architectural framework of the Pic2Model project. It aims to provide a comprehensive blueprint for the design and implementation of the system, ensuring that all stakeholders have a clear understanding of the system's structure and functionality.

1.2 Scope

This document covers all aspects of the software architecture of the Pic2Model project. It includes detailed descriptions of the system components, their interactions, and the technologies used. The scope is limited to the architectural design and does not include detailed implementation or code-level specifics.

1.3 Definitions

- **API (Application Programming Interface)**: A set of rules and tools for building software and applications. It defines how different software components should interact and communicate with each other. APIs allow developers to use predefined functions to interact with the operating system, other applications, or services without having to write the code from scratch.
- **DB (Database)**: A structured collection of data stored and accessed electronically. Databases are used to efficiently store, retrieve, and manage large amounts of information. They can be relational (SQL) or non-relational (NoSQL).
- **UI (User Interface)**: The part of a software application that interacts with the user. It includes all the visual elements, such as buttons, menus, and forms, that users use to interact with the software. The goal of UI design is to create an interface that is easy to use and provides good user experience.
- **UX (User Experience)**: The overall experience a user has when interacting with a product or service. It encompasses all aspects of the user's interaction, including usability, accessibility, and the pleasure of use. Good UX design aims to create products that are efficient, enjoyable, and easy to use.
- **HTTP (Hypertext Transfer Protocol)**: A protocol used for transferring hypertext requests and information on the internet. HTTP defines how messages are formatted and transmitted, and how web servers and browsers should respond to various commands.
- **HTTPS (Hypertext Transfer Protocol Secure)**: An extension of HTTP that includes encryption using SSL/TLS to secure data transferred between the web server and the browser. HTTPS ensures that data remains private and secure during transmission, protecting it from interception and tampering.
- **JSON (JavaScript Object Notation)**: A lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. JSON is often used to transmit data between a server and a web application as an alternative to XML.
- **CRUD (Create, Read, Update, Delete)**: A set of basic operations that can be performed on a database or a data store. These operations are the foundation of most database interactions:
  - **Create**: Adding new data.
  - **Read**: Retrieving existing data.
  - **Update**: Modifying existing data.
  - **Delete**: Removing data.
- **MVC (Model-View-Controller)**: A design pattern used for developing web applications. It separates an application into three interconnected components:
  - **Model**: Represents the application's data and business logic.
  - **View**: Represents the user interface and displays the data to the user.
  - **Controller**: Handles user input and interactions, updating the model and view as necessary.

1.4 Abbreviations

Common acronyms used in this document include:

Abbreviation	Definition
API	Application Programming Interface
DB	Database
UI	User Interface
UX	User experience
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
CRUD	Create, Read, Update, Delete
MVC	Model – View - Controller

1.5 References

- Tiến C. L. V. (2024, September 10). MVC là gì? Ứng dụng của mô hình MVC trong lập trình. Vietnix. https://vietnix.vn/tim-hieu-mo-hinh-mvc-la-gi/
- Tuan Thanh HO (2020) CS300-CSC13002-FIT-HCM. Youtube. https://www.youtube.com/playlist?list=PL3Bp9JDvkAra8rRrUPOpfKdKvJ6-okqaw

1.6 Overview

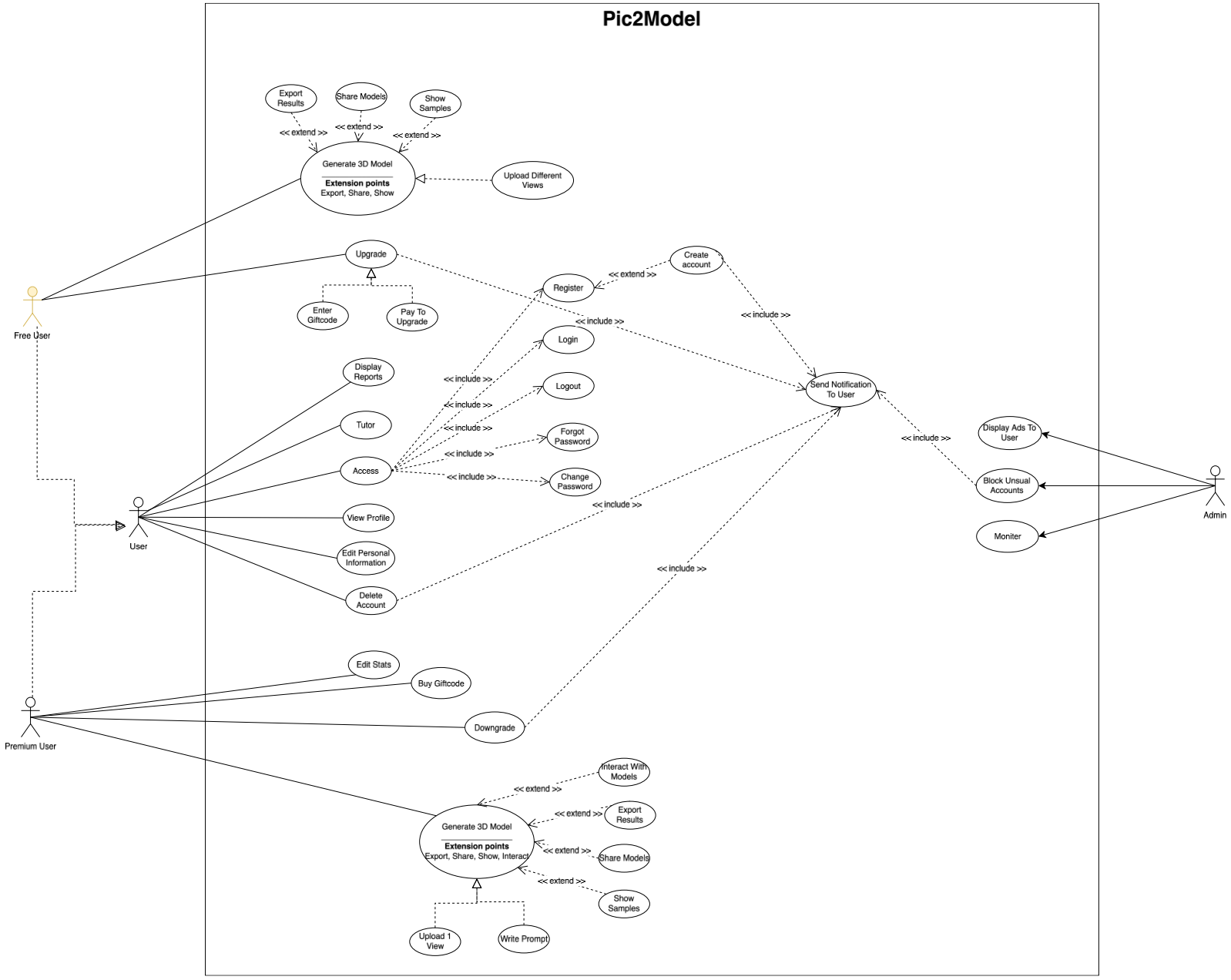
This document provides a comprehensive overview of the Pic2Model project's software architecture. It includes descriptions of the system's major components, their interactions, and the guiding principles behind the architectural decisions. The overview serves as a roadmap for stakeholders to understand the overall structure and design approach of the system.

2. Architectural Goals and Constraints

- **Architectural Goals:**
  - **Scalability**: The system can handle increased user loads smoothly, even as the user base grows.
  - **Security**: Protect user data, especially for uploaded images and 3D models, ensuring data privacy and compliance with regulations.
  - **Maintainability**: The architecture must support modularity and clear separation of concerns to facilitate debugging, updates, and future expansions.
  - **Performance**: Deliver fast and efficient 3D model processing with minimal waiting times. Aim for a maximum model generation time of under 60 seconds for standard photo sets.
  - **User Experience (UX)**: Create an intuitive and user-friendly interface that's easy for anyone to navigate, even without technical expertise.
  - **Reliability**: Provide a stable and reliable platform with minimal downtime, aiming for 99.9% uptime.
- **Architectural Constraints:**
  - **Technology Stack**: The backend will use Node.js and MongoDB for server-side processing and database management. The frontend will utilize React for building the user interface.
  - **Platform Support**: Ensure the app works well across all popular browsers and devices to reach a broad audience.
  - **Data Privacy Compliance**: The system must adhere to relevant data protection regulations (e.g., GDPR).

- **Resource Constraints**: The architecture must optimize resource usage due to potential limitations in server capacity and bandwidth.
- **Testing**: The system architecture must facilitate automated testing for both frontend and backend components to ensure reliability.
- **Integration**: The architecture must allow for seamless integration with external APIs for features like speech-to-text conversion and image recognition.

3. Use-Case Model



4. Logical View

A. Client-Server Architecture

Clients initiate interactions with servers, which offer a range of services. Clients call upon these services whenever necessary and wait for the responses to their requests. The client interface is responsible for displaying data and making minor updates, while the server takes care of data management. The client-server pattern facilitates modifiability and reusability by extracting common services, making it easier to modify them in one place. Additionally, this pattern enhances scalability and availability by centralizing resource and server control.

B. Model-View-Controller (MVC) Pattern

The MVC pattern aims to decouple user interface functionality from application functionality. It achieves this by dividing the application functionality into three distinct components:

- Models, responsible for handling application data.
- Views, responsible for displaying data to the user and facilitating user interactions.
- Controllers, acting as intermediaries between the model and view components, managing state changes.

By implementing MVC, usability is improved as it enables the separate design and implementation of the user interface from the core application logic.

i. Client

- The client acts as a component responsible for requesting services from a server component. Clients are equipped with ports that define the services they need. In the Coop Evaluation System context, the client refers to a web browser utilized to access the system. The client communicates with the server through RESTful web services, utilizing HTTP requests.
- The client's role is to offer users a graphical interface through which they can interact with the system.
- It consists of HTML, CSS, JavaScript, and other associated files that execute within the user's preferred web browser. Additionally, my project incorporates frameworks like Bootstrap for layout and ReactJS. The rationale behind choosing these technologies lies in their utility, large development community, and other advantages they bring to the project.

ii. Controller

- The controller encapsulates all business-related logic and manages incoming requests. In many instances, it responds by invoking a method on the model. The view and model are interconnected via a notification mechanism, ensuring that any changes resulting from this action are automatically updated in the view. As a RESTful API server, the server serves as an interface to receive requests from the client and invoke corresponding services.

iii. Model

- The component encompasses all the application's business logic, including the logic required for its construction, whether through database queries or other approaches.
- The model incorporates services responsible for preparing data and sending it back to the Controller.

iv. Database

- The database is responsible for storing, accessing, and updating data, among many other functionalities. It plays a vital role in managing security and recovery services within the data management system, ensuring the enforcement of constraints in this underlying system.

v. External API

- An open or external API is specifically designed to be accessible not only by a broader user base, including web developers, admins, and owners but also by developers within the organization that published the API. Additionally, external developers who wish to use the interface can easily subscribe to it. This openness allows for seamless integration and utilization of the API by both internal and external parties.

C. Technologies

i. Client

- React with TypeScript (ReactTS) is a declarative, efficient, and versatile JavaScript library for creating user interfaces. It empowers developers to construct intricate UIs by composing small, isolated code units known as "components". When building websites, ReactTS works seamlessly alongside HTML, CSS, and JavaScript, and in our project, we utilize Tailwind CSS for layout.
- Several factors drove the decision to choose React with TypeScript:
  - React with TypeScript is among the most widely adopted JavaScript libraries, boasting a robust foundation and an active developer community.
  - Leveraging components in ReactTS enables efficient development and enhances code organization.
  - ReactTS significantly contributes to faster app and page load times. It also simplifies maintenance and debugging, ensuring a smoother development experience.
- Vite's lightning-fast development server and native ES module support synergize perfectly with React and TypeScript. Developers enjoy near-instantaneous hot module replacement and faster refresh times, while TypeScript's static typing ensures robust code and early bug detection.

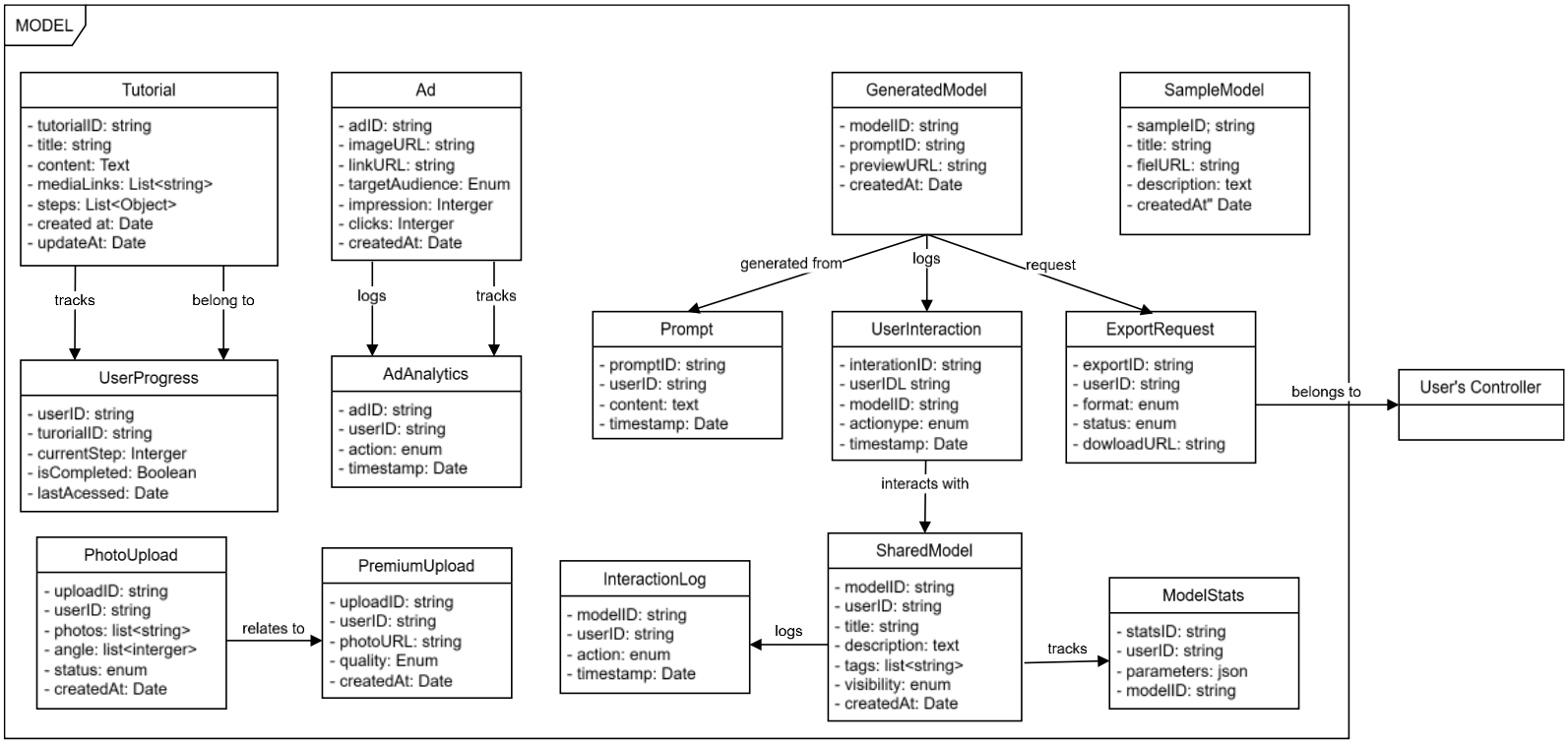
ii. Server

- The server acts as a component that accepts and manages requests transmitted by the client, developed within the Node.js environment. Node.js is a cross-platform open-source runtime environment and library that allows web applications to run independently of the client's browser. Furthermore, we utilize the ExpressJS framework, which is a feature-rich Node.js web application framework, offering extensive capabilities for constructing web and mobile applications.
- Reason for choosing ExpressJS:
  - The server functions as a module that receives and handles requests sent by the client, created using the Node.js environment. Node.js is a cross-platform open-source runtime environment and library that enables web applications to operate outside the client's browser.
  - Additionally, we employ the ExpressJS framework, a robust Node.js web application framework that provides a wide range of features for building web and mobile applications.

iii. Database

- **MongoDB** is a popular NoSQL database known for its flexibility and scalability. It stores data in JSON-like documents, making it ideal for handling unstructured or semi-structured data. Some advantages of MongoDB include:
  - **Flexible Data Model**: MongoDB's document-based model allows developers to store data in flexible and schema-less JSON-like documents. This flexibility enables easy handling of unstructured or semi-structured data, making it well-suited for dynamic and evolving application requirements.
  - **Scalability**: MongoDB excels in scalability, allowing data distribution across multiple servers through sharding. As data volume grows, MongoDB can seamlessly handle increased traffic and ensure high performance and responsiveness.
  - **Rich Query Language**: MongoDB provides a powerful and expressive query language, enabling developers to perform complex queries and manipulations on the stored data. This versatility supports a wide range of use cases and enhances data retrieval efficiency.
  - **High Performance**: MongoDB is designed for efficiency, offering efficient indexing and optimized storage. This results in excellent read-and-write performance, making it ideal for high-throughput applications and real-time data processing.

4.1 Component: Model



a. Class: Tutorial

Attributes:

- **tutorialID** (String): Unique identifier for the tutorial.
- **title** (String): The name or title of the tutorial.
- **content** (Text): Detailed information or instructions for the tutorial.
- **mediaLinks** (List<String>): URLs to any media (images, videos, etc.) associated with the tutorial.
- **steps** (List<Object>): A list of steps or actions that make up the tutorial.
- **createdAt** (Date): Timestamp when the tutorial was created.
- **updatedAt** (Date): Timestamp when the tutorial was last updated.

Methods:

- **addStep**(Object step): Adds a new step to the tutorial.
- **updateContent**(Text newContent): Updates the tutorial content.
- **getMediaLinks**(): Returns all media links associated with the tutorial.

b. Class: UserProgress

Attributes:

- **userID** (String): Identifier of the user associated with the progress.
- **tutorialID** (String): Identifier of the tutorial being tracked.
- **currentStep** (Integer): The current step the user is on in the tutorial.
- **isCompleted** (Boolean): Indicates if the user has completed the tutorial.
- **lastAccessed** (Date): Timestamp of the last time the user accessed the tutorial.

Methods:

- **updateStep**(Integer step): Updates the current step for the user.
- **markComplete**(): Marks the tutorial as completed for the user.
- **getProgress**(): Retrieves the current progress details.

c. Class: Ad

Attributes:

- **adID** (String): Unique identifier for the advertisement.
- **imageUrl** (String): URL of the ad's image.
- **linkUrl** (String): URL where the ad redirects users.
- **targetAudience** (Enum): Defines the audience type (e.g., age group, region, etc.).
- **impressions** (Integer): The number of times the ad was displayed.
- **clicks** (Integer): The number of times the ad was clicked.
- **createdAt** (Date): Timestamp when the ad was created.

Methods:

- **incrementImpressions**(): Increases the impressions count.
- **incrementClicks**(): Increases the clicks count.
- **getCTR**(): Calculates the click-through rate of the ad.

d. Class: AdAnalytics

Attributes:

- **adID** (String): Identifier of the related advertisement.
- **userID** (String): Identifier of the user who interacted with the ad.
- **action** (Enum): Type of user interaction (e.g., click, view).
- **timestamp** (Date): Timestamp when the interaction occurred.

Methods:

- **logAction**(Enum action, Date timestamp): Logs a specific action performed on the ad.
- **getAnalytics**(): Retrieves analytics for the given ad.

e. Class: SharedModel

Attributes:

- **modelID** (String): Unique identifier for the shared model.
- **userID** (String): Identifier of the user sharing the model.
- **title** (String): The title of the shared model.
- **description** (Text): A brief description of the model.
- **tags** (List<String>): Tags or keywords associated with the model.
- **visibility** (Enum): Defines whether the model is public, private, or restricted.
- **createdAt** (Date): Timestamp when the model was shared.

Methods:

- **addTag**(String tag): Adds a new tag to the model.
- **changeVisibility**(Enum newVisibility): Updates the visibility setting of the model.
- **getDetails**(): Retrieves all details about the shared model.

f. Class: InteractionLog

Attributes:

- **modelID** (String): Identifier of the related model.
- **userID** (String): Identifier of the user interacting with the model.
- **action** (Enum): Type of interaction (e.g., view, like, share).
- **timestamp** (Date): Timestamp when the interaction occurred.

Methods:

- **logInteraction**(String modelID, String userID, Enum action): Logs user interaction for the model.
- **getLog**(): Retrieves the interaction log for the model.

g. Class: PhotoUpload

Attributes:

- **uploadID** (String): Unique identifier for the photo upload.

- **userID** (String): Identifier of the user uploading photos.
- **photos** (List<String>): List of URLs of the uploaded photos.
- **angles** (List<Integer>): Angles of the photos in degrees.
- **status** (Enum): Status of the upload (e.g., pending, completed, failed).
- **createdAt** (Date): Timestamp when the photos were uploaded.

Methods:

- **addPhoto**(String photoUrl, Integer angle): Adds a new photo with its angle.
- **updateStatus**(Enum status): Updates the status of the upload.
- **getUploadDetails**(): Retrieves details of the photo upload.

h. Class: PremiumUpload

Attributes:

- **uploadID** (String): Identifier of the premium upload.
- **userID** (String): Identifier of the user uploading the photo.
- **photoUrl** (String): URL of the premium photo.
- **quality** (Enum): Quality of the photo (e.g., high, medium, low).
- **createdAt** (Date): Timestamp when the upload was made.

Methods:

- **upgradeQuality**(Enum newQuality): Updates the quality of the uploaded photo.
- **getPremiumDetails**(): Retrieves details of the premium upload.

i. Class: Prompt

Attributes:

- **promptID** (String): Unique identifier for the prompt.
- **userID** (String): Identifier of the user creating the prompt.
- **content** (Text): The text content of the prompt.
- **timestamp** (Date): Timestamp when the prompt was created.

Methods:

- **updateContent**(String newContent): Updates the prompt content.
- **getPromptDetails**(): Retrieves the prompt details.

j. Class: GeneratedModel

Attributes:

- **modelID** (String): Unique identifier of the generated model.
- **promptID** (String): Identifier of the related prompt.
- **previewUrl** (String): URL of the generated model preview.
- **createdAt** (Date): Timestamp when the model was generated.

Methods:

- **generatePreview**(): Generates a preview for the model.
- **getModelDetails**(): Retrieves details of the generated model.

k. Class: ModelStats

Attributes:

- **statsID** (String): Identifier for the stats entry.
- **userID** (String): Identifier of the user the stats belong to.
- **parameters** (JSON): A set of parameters used for the model.
- **modelID** (String): Identifier of the related model.

Methods:

- **updateParameters**(JSON newParams): Updates the parameters of the stats.
- **getStats**(): Retrieves the statistics.

l. Class: SampleModel

Attributes:

- **sampleID** (String): Unique identifier of the sample model.
- **title** (String): Title of the sample.
- **fileUrl** (String): URL of the sample file.
- **description** (Text): Description of the sample model.
- **createdAt** (Date): Timestamp when the sample was created.

Methods:

- **updateSampleDetails**(String title, String description): Updates details of the sample.
- **getSample**(): Retrieves the sample details.

m. Class: UserInteraction

Attributes:

- **interactionID** (String): Unique identifier of the user interaction.
- **userID** (String): Identifier of the user interacting.
- **modelID** (String): Identifier of the model being interacted with.
- **actionType** (Enum): Type of action (e.g., like, share, comment).
- **timestamp** (Date): Timestamp when the action occurred.

Methods:

- **logInteraction**(Enum actionType, String modelID): Logs user interaction.
- **getInteractions**(): Retrieves a list of interactions.

n. Class: ExportRequest

Attributes:

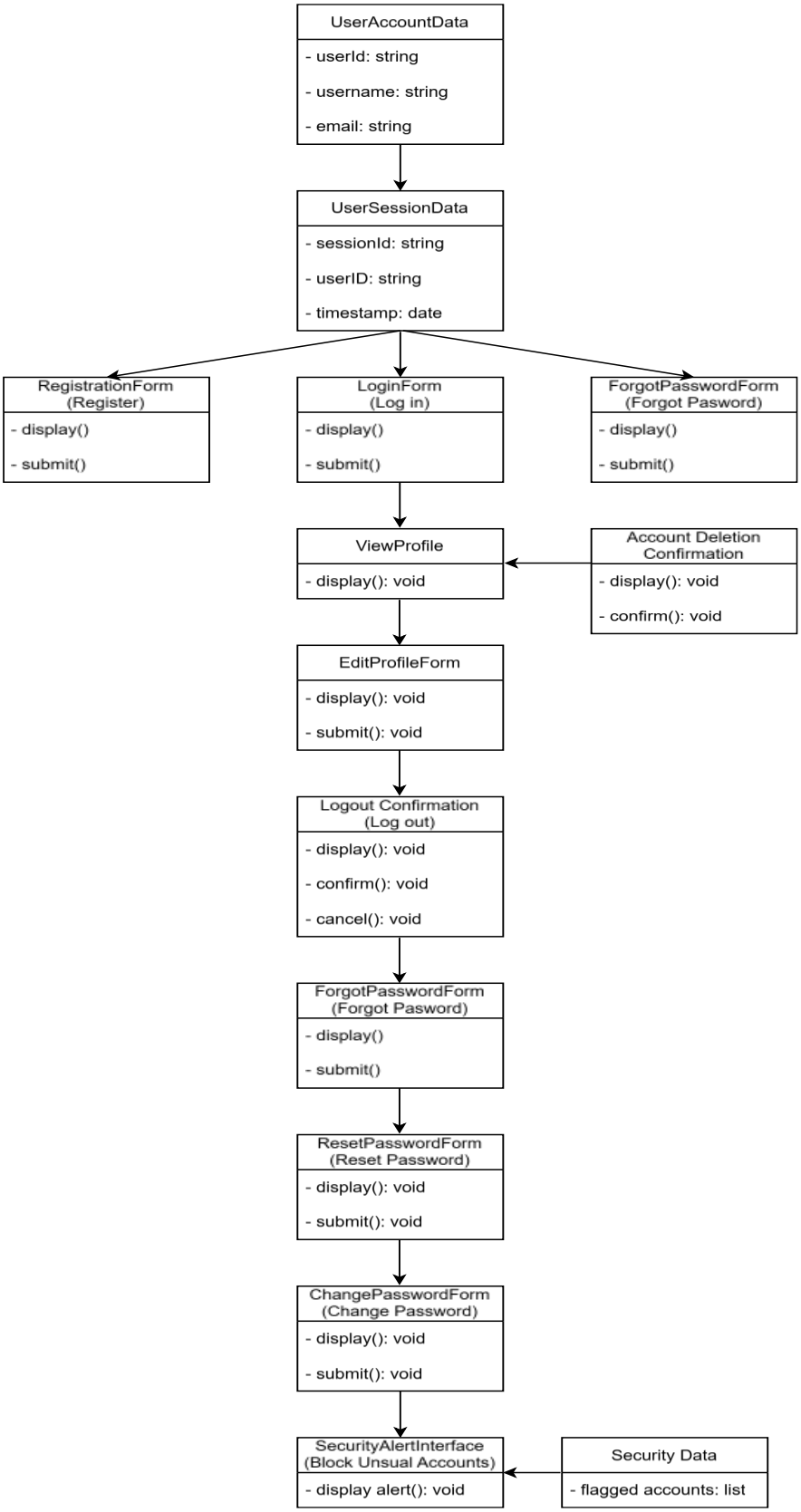
- **exportID** (String): Unique identifier of the export request.
- **userID** (String): Identifier of the user making the request.
- **format** (Enum): Format of the exported file (e.g., OBJ, STL, FBX).
- **status** (Enum): Status of the request (e.g., pending, completed, failed).
- **downloadUrl** (String): URL for downloading the export file.

Methods:

- **updateStatus**(Enum status): Updates the status of the export request.
- **getExportDetails**(): Retrieves details of the export request.

4.2 Component: View

a. User Account and Session Management:

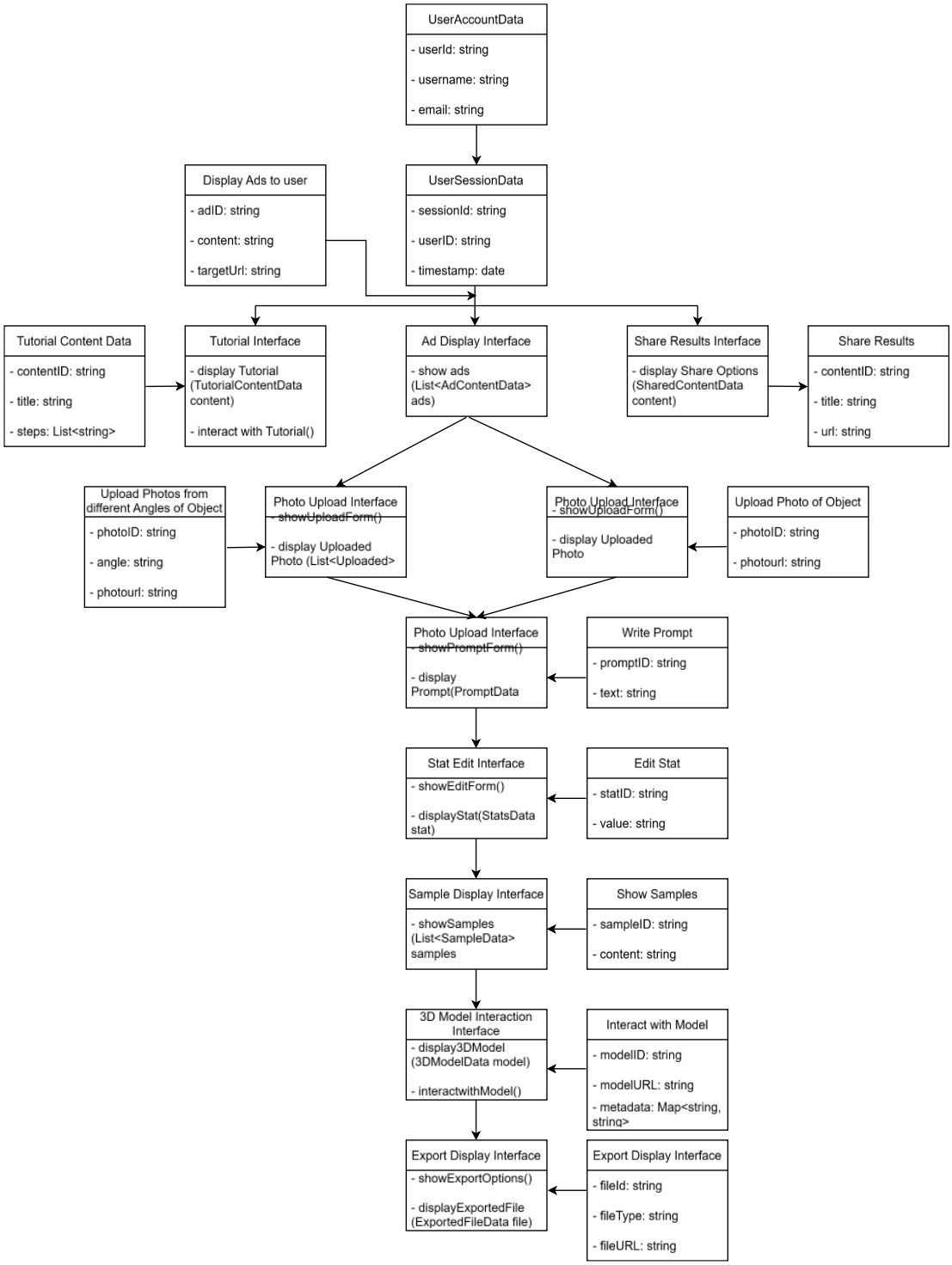

UserAccountData
- userId: string
- username: string
- email: string
UserSessionData
- sessionId: string
- userID: string
- timestamp: date
RegistrationForm (Register)
- display()
- submit()
LoginForm (Log in)
- display()
- submit()
ForgotPasswordForm (Forgot Pasword)
- display()
- submit()
ViewProfile
- display(): void
Account Deletion Confirmation
- display(): void
- confirm(): void
EditProfileForm
- display(): void
- submit(): void
Logout Confirmation (Log out)
- display(): void
- confirm(): void
- cancel(): void
ForgotPasswordForm (Forgot Pasword)
- display()
- submit()
ResetPasswordForm (Reset Password)
- display(): void
- submit(): void
ChangePasswordForm (Change Password)
- display(): void
- submit(): void
SecurityAlertInterface (Block Unsual Accounts)
- display alert(): void
Security Data
- flagged accounts: list

This level focuses on managing user accounts, including registration, authentication, and session tracking.

- **User Registration**: This involves creating a new user account by collecting user information such as username, email, and password. The system validates this information and stores it securely.
  - **View:** Registration Form
  - **Method:**
    - **display()**: Shows the registration form to the user.
    - **submit()**: Collects and submits the user's registration data.
  - **Flow:**
    - User fills in registration information.
    - The system validates and creates a new account.
    - User receives a registration success confirmation.
- **Login**: This process verifies user credentials (username and password) to grant access to the system. If the credentials match, a session is initiated for the user.
  - **View**: LoginForm
  - **Methods**:
    - **display()**: Shows the login form.
    - **submit()**: Collects and submits the login information.
  - **Flow**:
    - User enters login credentials.
    - The system verifies the credentials.
    - User is successfully logged in and a session starts.
- **Forgot Password**: In case users forget their passwords, this functionality helps them reset it by verifying their email and sending a reset link.
  - **View**: ForgotPasswordForm
  - **Methods**:
    - **display()**: Shows the forgot password form.
    - **submit()**: Collects and submits the email for password reset.
  - **Flow**:
    - User enters email.
    - The system sends a password reset link via email.
- **Reset Password**: Allows users to set a new password using the reset link received via email.
  - **View**: ResetPasswordForm
  - **Methods**:
    - **display()**: Shows the reset password form.
    - **submit()**: Collects and submits the new password.
  - **Flow**:
    - User enters a new password.
    - The system updates the password.
    - User can log in with the new password.
- **Change Password**: Enables users to change their existing password to a new one after verifying their current password.
  - **View**: ChangePasswordForm
  - **Methods**:
    - **display()**: Shows the change password form.
    - **submit()**: Collects and submits the current and new password.
  - **Flow**:
    - User enters the current and new password.
    - The system verifies and updates the password.
    - User receives confirmation of the password change.
- **Delete Account**: Provides the option for users to permanently delete their accounts from the system.
  - **View**: AccountDeletionConfirmation
  - **Methods**:

        - **display()**: Shows the account deletion confirmation prompt.
        - **confirm()**: Confirms account deletion.
    - **Flow**:
        - User requests account deletion.
        - The system asks for confirmation.
        - Account is deleted upon confirmation.
- **Block Unusual Accounts**: Detects and flags suspicious activities or accounts to maintain security.
    - **View**: SecurityAlertsInterface
    - **Methods**:
        - **displayAlert()**: Shows an alert when unusual activity is detected.
    - **Flow**:
        - The system detects unusual activity.
        - An alert is displayed to the admin.
- **View Profile**: Allows users to view their profile information.
    - **View**: ProfilePage
    - **Methods**:
        - **display()**: Shows the user's profile information.
    - **Flow**:
        - User accesses their profile page.
        - Profile information is displayed.
- **Edit Personal Information**: Users can update their personal details such as name, email, and address.
    - **View**: ProfileEditForm
    - **Methods**:
        - **display()**: Shows the profile edit form.
        - **submit()**: Collects and submits updated information.
    - **Flow**:
        - User edits personal information.
        - The system updates the information.
        - Updated information is displayed.
- **Log out**: Ends the user's session and logs them out of the system.
    - **View**: LogoutConfirmation
    - **Methods**:
        - **display()**: Shows the logout confirmation prompt.
        - **confirm()**: Confirms logout.
        - **cancel()**: Cancels the logout process.
    - **Flow**:
        - User requests to log out.
        - The system asks for confirmation.
        - User is logged out upon confirmation.

b. Content Interaction and Management:



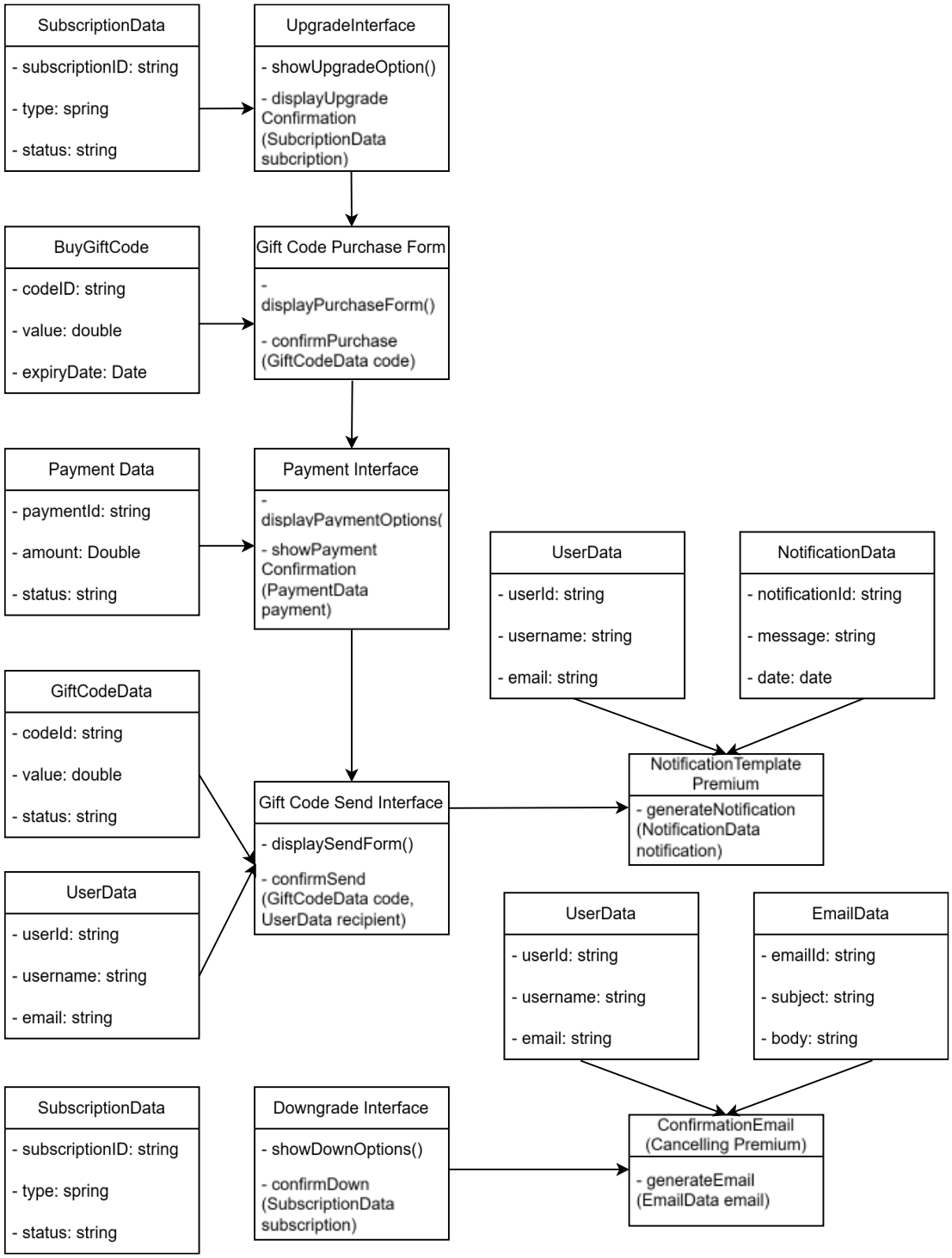
This level involves various interactions users can have with the content within the system, including tutorials, ads, sharing results, and more.

- **Tutor**: Provides users with tutorials or guides to help them understand the system's features or perform specific tasks.
  - **View**: Tutorial Interface
  - **Methods**:

        - **displayTutorial(content: TutorialContentData)**: Displays tutorial content.
        - **interactWithTutorial()**: Allows the user to interact with the tutorial.
    - **Flow**:
        - User accesses the tutorial.
        - Tutorial content is displayed.
        - User interacts with the tutorial.
- **Display Ads To User**: Shows advertisements to users based on their profile or activity within the system.
    - **View**: Ad Display Interface
    - **Methods**:
        - **showAds(ads: List<AdContentData>)**: Displays advertisements.
    - **Flow**:
        - User visits a webpage with ads.
        - Advertisements are displayed based on user profile or activity.
- **Share Results**: Allows users to share their results or content with others through various means.
    - **View**: Share Results Interface
    - **Methods**:
        - **displayShareOptions(content: SharedContentData)**: Displays sharing options.
    - **Flow**:
        - User wants to share results.
        - The system displays sharing options.
        - User selects and shares the results.
- **Upload Photos From Different Angles Of Object**: Lets users upload multiple photos of an object from different angles for purposes like creating 3D models.
    - **View**: Photo Upload Interface
    - **Methods**:
        - **showUploadForm()**: Shows the multi-photo upload form.
        - **displayUploadedPhotos(photos: List<UploadedPhotosData>)**: Displays the uploaded photos.
    - **Flow**:
        - User selects to upload multiple photos.
        - The system shows the upload form.
        - User uploads photos from different angles.
        - The system processes and displays the photos.
- **Upload Photo Of Object**: Enables users to upload a single photo of an object.
    - **View**: Photo Upload Interface
    - **Methods**:
        - **showUploadForm()**: Shows the photo upload form.
        - **displayUploadedPhoto(photo: UploadedPhotoData)**: Displays the uploaded photo.
    - **Flow**:
        - User selects to upload a photo.
        - The system shows the upload form.
        - User uploads the photo.
        - The system processes and displays the photo.
- **Write Prompt**: Users can write prompts or commands for the system to execute certain actions or generate content.
    - **View**: Prompt Input Form
    - **Methods**:
        - **showPromptForm()**: Shows the prompt input form.
        - **displayPrompt(prompt: PromptData)**: Displays the prompt.
    - **Flow**:
        - User writes a prompt.
        - The system displays the prompt.

        - The prompt is applied to content generation.
- **Edit Stats**: Allows users to edit and manage statistical data.
    - **View**: Stats Edit Form
    - **Methods**:
        - **showEditForm()**: Shows the stats edit form.
        - **displayStat(stat: StatsData)**: Displays the stat.
    - **Flow**:
        - User edits stats.
        - The system updates and displays the new stats.
- **Show Samples**: Displays sample content or models to users.
    - **View**: Sample Display Interface
    - **Methods**:
        - **showSamples(samples: List<SampleData>)**: Displays sample content.
    - **Flow**:
        - User accesses samples.
        - The system displays samples.
        - User interacts with the samples.
- **Interact With Model**: Provides an interface for users to interact with 3D models, allowing manipulation and inspection of the models.
    - **View**: 3D Model Interaction Interface
    - **Methods**:
        - **display3DModel(model: 3DModelData**): Displays the 3D model.
        - **interactWithModel()**: Allows interaction with the 3D model.
    - **Flow**:
        - User accesses a 3D model.
        - The model is displayed.
        - User interacts with the model.

c. Subscription, Transaction and Notification Management:



This level handles user subscriptions, financial transactions, and notifications within the system.

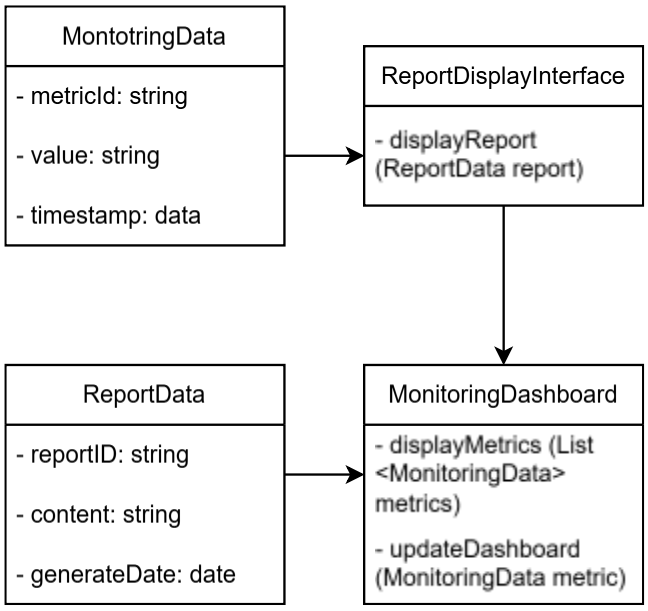
- **Upgrade**: Enables users to upgrade their subscription plans to access additional features or services.
  - **View**: Upgrade Interface
  - **Methods**:
    - **showUpgradeOptions()**: Shows upgrade options.
    - **displayUpgradeConfirmation(subscription: SubscriptionData)**: Displays upgrade confirmation.

    - **Flow**:
        - User selects upgrade options.
        - The system processes the upgrade.
        - User receives confirmation of successful upgrade.
- **Buy Gift Code**: Allows users to purchase gift codes for themselves or others.
    - **View**: Gift Code Purchase Form
    - **Methods**:
        - **displayPurchaseForm()**: Shows the gift code purchase form.
        - **confirmPurchase(code: GiftCodeData)**: Confirms gift code purchase.
    - **Flow**:
        - User selects to buy a gift code.
        - The system processes the transaction.
        - User receives the gift code.
- **Pay**: Facilitates payment processing for various services or products within the system.
    - **View**: Payment Interface
    - **Methods**:
        - **displayPaymentOptions()**: Shows payment options.
        - **showPaymentConfirmation(payment: PaymentData)**: Displays payment confirmation.
    - **Flow**:
        - User initiates a payment.
        - The system processes the transaction.
        - Payment data is recorded.
        - User receives a payment confirmation.
- **Send Gift Code**: Users can send purchased gift codes to other users.
    - **View**: Gift Code Send Interface
    - **Methods**:
        - **displaySendForm()**: Shows the send gift code form.
        - **confirmSend(code: GiftCodeData, recipient: UserData)**: Confirms sending the gift code.
    - **Flow**:
        - User selects the option to send a gift code.
        - User enters recipient details and gift code.
        - The system processes the input.
        - The gift code data is updated to reflect the transfer.
        - The recipient receives the gift code through the selected method.
- **Downgrade**: Allows users to downgrade their subscription plans.
    - **View**: Downgrade Interface
    - **Methods**:
        - **showDowngradeOptions()**: Shows downgrade options.
        - **confirmDowngrade(subscription: SubscriptionData)**: Confirms downgrade.
    - **Flow**:
        - User selects the downgrade option.
        - The system processes the request.
        - Subscription data is updated to reflect the downgrade.
        - Downgrade confirmation is displayed.
- **Sends Notification When User Registers for Premium**: Sends notifications to users when they register for premium services.
    - **View**: Notification Template
    - **Methods**:
        - **generateNotificationTemplate(notification: NotificationData)**: Generates the notification template.
    - **Flow**:
        - User registers for the Premium service.

- Registration controller updates the user status to Premium in the user data.
- Notification controller triggers the creation of a notification.
- Notification data is updated to include the new notification.
- The notification is sent using the notification template.
- User receives a notification about the Premium registration.

- **Sends Confirmation Email for Canceling Premium**: Sends a confirmation email to users when they cancel their premium subscription.
  - **View**: Confirmation Email Template
  - **Methods**:
    - **generateEmailTemplate(email: EmailData)**: Generates the confirmation email template.
  - **Flow**:
    - User requests to cancel Premium subscription.
    - Subscription controller processes the cancellation request.
    - User status is updated in the user data.
    - Email controller generates a confirmation email.
    - Email data is updated with the new email.
    - The confirmation email is sent using the confirmation email template.
    - User receives a confirmation email for the cancellation.

d. Monitoring:



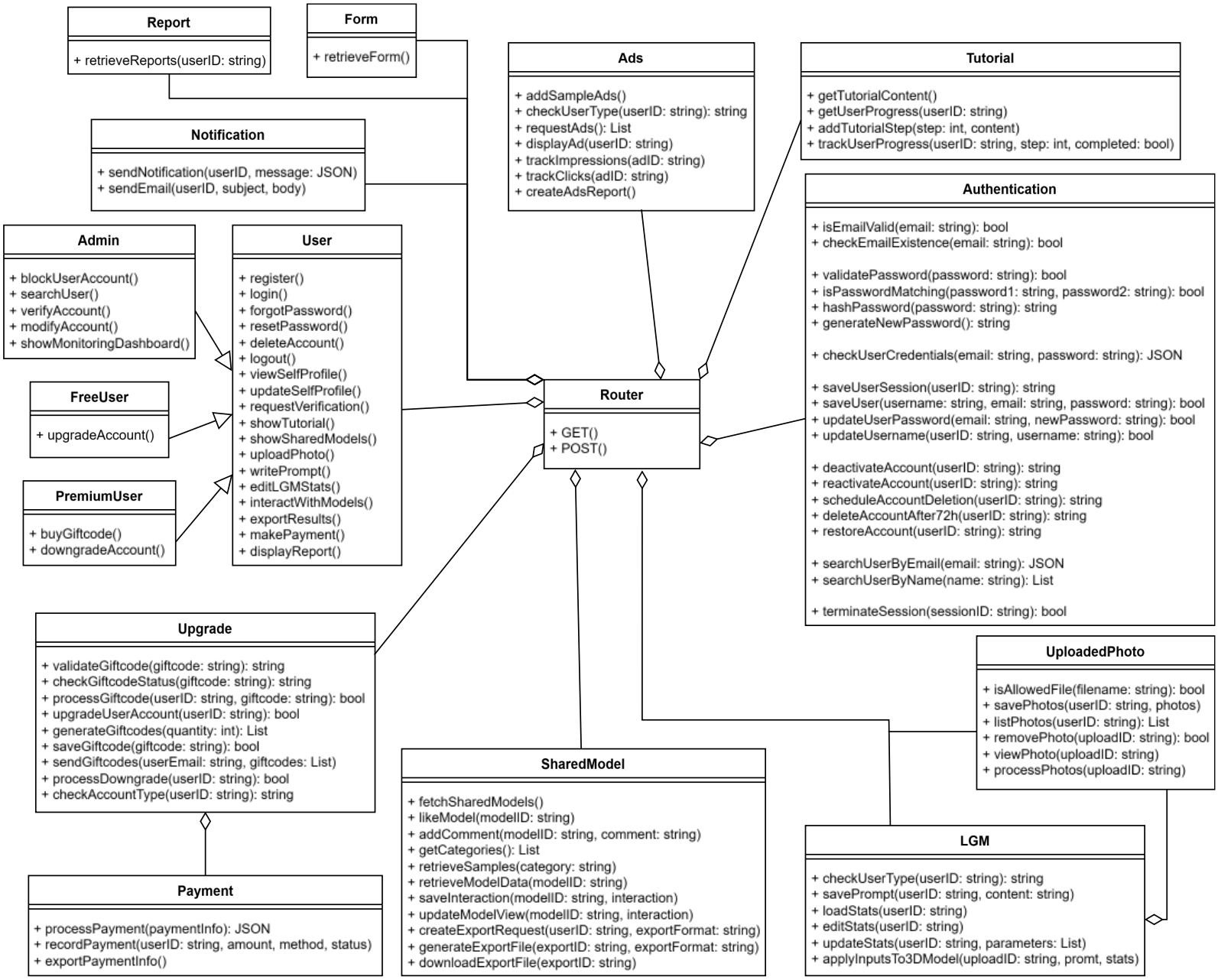
This level focuses on monitoring the system's performance, security, and user activities to ensure smooth operation.

- **Monitor**: Continuously collects data on system performance, security, and user activities. This data is analyzed to identify trends or issues.
  - **View**: Monitoring Dashboard
  - **Methods**:
    - **displayMetrics(metrics: List<MonitoringData>)**: Displays monitoring metrics.
    - **updateDashboard(metric: MonitoringData)**: Updates the dashboard with new data.
  - **Flow**:
    - The system continuously collects monitoring data.
    - Monitoring controller processes and analyzes the collected data.
    - Monitoring data is updated with the latest information.
    - The monitoring dashboard displays the processed data in real-time.

    - Admins or users view the monitoring data on the dashboard.
- **Display Reports**: Generates and displays reports based on the monitored data, providing insights into system performance, user behavior, and other metrics.
    - **View**: Report Display Interface
    - **Methods**:
        - **displayReport(report: ReportData)**: Displays the report.
    - **Flow**:
        - User requests a report.
        - Report controller retrieves relevant data.
        - Report data is compiled and formatted.
        - Report display interface is generated.
        - User views the report on the report display interface.

4.3 Component: Controller

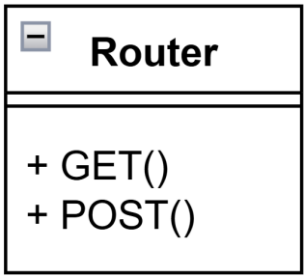
4.3.1 Class diagram overview



4.3.2 Description of key classes

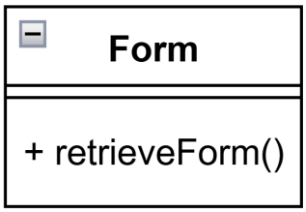
The controller's responsibility is to pull, modify, and provide data to the user. Essentially, the controller is the link between the view and model. Through getter and setter functions, the controller pulls data from the model and initializes the views. If there are any updates from the views, it modifies the data with a setter function.

a. Class: Router



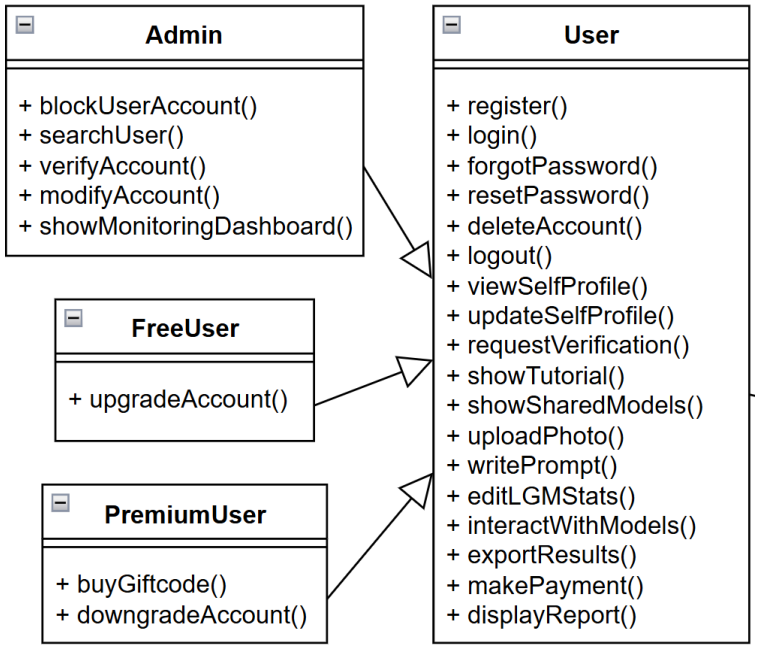
- The "Router" forwards the supported requests (and any information encoded in request URLs) to the appropriate controller functions. The controller functions get the requested data from the models, create an HTML page displaying the data, and return it to the user to view in the browser.
- After receiving a request from the client, the route will navigate to the corresponding controller. After that, call the services in the model and receive data by model and view.

b. Class: Form



- Controller is the component that directly interacts with the user, and to do this, the website needs to provide users with intuitive, easy-to-interact interfaces. And one of those interfaces is the form for users to enter data.
- The controller will send requests to the remaining two components, the model and the view, based on each user request to get the necessary forms and display them to the user.
- This is an important and quite complex component in the system because it needs to rely on current requests and user interactions with the system to be able to retrieve appropriate forms (interfaces) and display them to the user.

c. Class: User



The "User" class will call the services that the system provides to users. For different types of users, when using this system, they will have different permissions and services. But in general, users can use basic services such as:

- **register()**: Allows users to register a new account on the system. At this time, the system will create a registration form for users to enter their personal information such as: email, password, name, etc.
- **login()**: Allows users to log in to a previously registered account. At this time, the system will create a login form for users to enter their login information such as: email, password.
- **forgotPassword()**: Allows users to retrieve their password when they forget their password. At this time, the system will send an email containing a password change link to the user.
- **resetPassword()**: Allows users to proactively change their password. This function can only be used when the user is logged in to his account.
- **deleteAccount()**: Allows the user to delete his account from the system. At this time, the user's account will be temporarily disabled and will be deleted from the system after a certain period of time. However, the user can cancel the deletion of his account at any time before the account is deleted from the system.
- **logout()**: Allows the user to log out of his account. At this time, the system will delete the user's login information and session.
- **viewSelfProfile()**: Allows the user to view his personal information.
- **updateSelfProfile()**: After viewing his personal information, the user can update his personal information to match the current time.
- **showTutorial()**: For new users, the system provides some basic instructions on how to use the system. This function helps users quickly become familiar with the system. Users can proactively access this function at any time.
- **showSharedModels()**: Allows users to view 3D models that other users have shared on the system.
- **uploadPhoto()**: Allows users to upload one or more images to the system. This image will be the input data source for creating 3D models.
- **writePrompt()**: Allows users to write a paragraph describing the 3D model that the user wants to create. This paragraph will help users describe more clearly the 3D model that the user wants to create.
- **editLGMStats()**: Allows users to edit the parameters of the AI model to help create 3D models from photos and promt.
- **interactWithModels()**: Allows users to perform some basic interactions with 3D models such as: rotate, zoom in, zoom out, move, etc.
- **exportResults()**: Allows the user to export the 3D model results that the user has created as a .ply file, a 4D photo of the 3D model, or a 360-degree video around the 3D model.
- **makePayment()**: Allows the user to make payments for services such as registering for a premium account, buying gift codes, etc.
- **displayReport()**: Allows the user to view reports of the 3D models that the user has previously created, as well as other activities that the user has performed on the system.

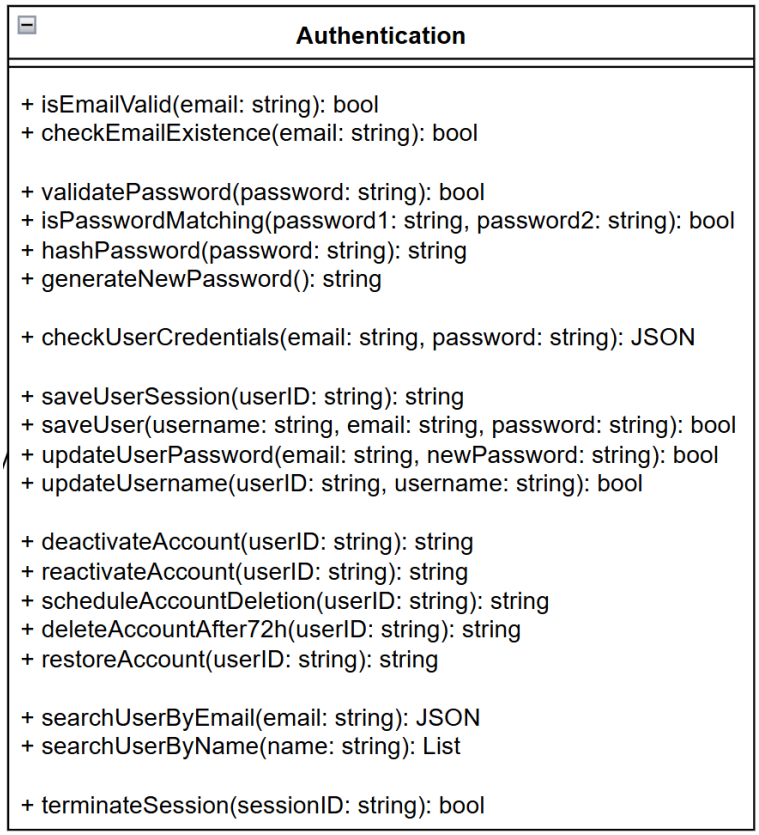
On the other hand, different types of users will have separate services that the system provides for them. For example:

- With "User" being "Admin":
  - **blockUserAccount()**: Allows the "Admin" to block the account of another "User" that they believe violates the system's regulations. The blocked user will need to confirm via email and accept some terms to be able to use his account again.
  - **searchUser()**: Allows "Admin" to search for information of another "User" on the system.
  - **verifyAccount()**: Allows "Admin" to authenticate the account of another "User". Authenticated accounts will have an authentication icon next to the user's name.
  - **modifyAccount()**: Allows "Admin" to actively edit the personal information of another "User".
  - **showMonitoringDashboard()**: Allows "Admin" to view statistics about system activity, number of users, number of 3D models created, etc.
- With "User" being "FreeUser":

- **upgradeAccount()**: Because "FreeUser" can only use some basic services of the system and will be limited in the number of 3D models they can create every 24 hours. Therefore, "FreeUser" can upgrade their account to "PremiumUser" to use all the services provided by the system.

- With "User" as "PremiumUser":
  - **buyGiftcode()**: "PremiumUser" can buy giftcode to give to others. This giftcode will help the recipient to use the system's premium services without having to pay for a certain period.
  - **downgradeAccount()**: "PremiumUser" can downgrade their account to "FreeUser" if they no longer want to use premium services.

d. Class: Authentication

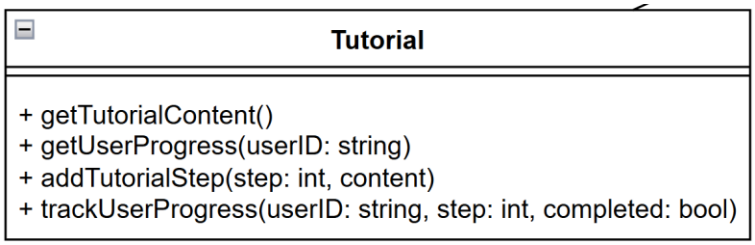


The "Authentication" class represents a group of methods that help authenticate users. These methods will be used in API endpoints to authenticate users before they can access system resources. At the same time, it also provides some methods to manipulate the database to store information about users.
Specifically, this class provides the following methods:

- **isEmailValid()**: This method will check whether an email address is valid or not through regular expression (regex). This check will be based on some basic rules such as: the email address must contain the @ character, must have at least one character before and after the @ character, must have at least one period after the @ character, and this period must be at the end of the email address.
- **checkEmailExistence()**: When an email is authenticated as valid, this method will check whether this email already exists in the database.
  - In case the user registers a new account, if the email already exists, the system will notify an error and ask the user to enter another email address.
  - In case the user logs in, if the email does not exist, the system will notify an error and ask the user to re-enter the email.

- **validatePassword()**: This method will check the validity of the password based on some rules such as: the password must contain at least 8 characters, including at least one uppercase letter, one lowercase letter, one number and one special character (for example: !@#$%^&*).
- **isPasswordMatching()**: When the user chooses the function to register a new account or change the old password, the user is forced to enter the (new) password twice. This method will check whether the two passwords the user enters match or not.
- **hashPassword()**: To increase the security of user information, the password will be encrypted before being saved to the database. This method will encrypt the password based on a password encryption algorithm such as bcrypt.
- **generateNewPassword()**: This method will generate a random password for the user. This password will be used in case the user forgets the old password and requests the system to generate a new password.
- **checkUserCredentials()**: This method will check if the user's login information (including email and password) is correct. If the login information is incorrect, the system will notify an error and ask the user to re-enter the login information.
- **saveUserSession()**: When the user successfully logs in, the system will save the user's login session information to the database. This method will perform the storage of this information.
- **saveUser()**: After the user registers a new account, this method will be called to save the user's information to the database. This information includes email, encrypted password, username, etc.
- **updateUserPassword()**: When the user wants to change the password, this method will be called to update the new password to the database.
- **updateUsername()**: When the user wants to change the display name, this method will be called to update the new display name to the database. The new display name will be used to display on the user interface.
- **deactivateAccount()**: When the user wants to temporarily disable the account, this method will be called to do this. The disabled account will not be able to access the system until the account is reactivated.
- **reactivateAccount()**: When the user wants to reactivate a disabled account, this method will be called to do this. The account after being reactivated will be able to access the system normally.
- **scheduleAccountDeletion()**: When the user wants to permanently delete the account, this method will be called to schedule the account deletion. The account will be deleted from the database after a certain period (e.g. 30 days). During this time, the user can cancel the deletion of the account.
- **deleteAccountAfter72h()**: This method will delete the user account from the database after 72 hours from when the account was marked for deletion. This gives the user time to cancel the deletion of the account if they change their mind.
- **restoreAccount()**: When the user cancels the deletion of the account, this method will be called to restore the account that was marked for deletion. The restored account will be able to access the system normally.
- **searchUserByEmail()**: This method will search for user information based on the email address. Since each user's email is unique, if a matching email is found, the method will return the user's details such as email, username, etc. If not found, the method will return an error message.
- **searchUserByName()**: This method will search for user information based on username. Since usernames are not unique, the method will return to a list of users with matching names. This list will include detailed information about each user such as email, username, etc.
- **terminateSession()**: When a user logs out of the system, this method will be called to terminate the user's session. The session information will be deleted from the database.

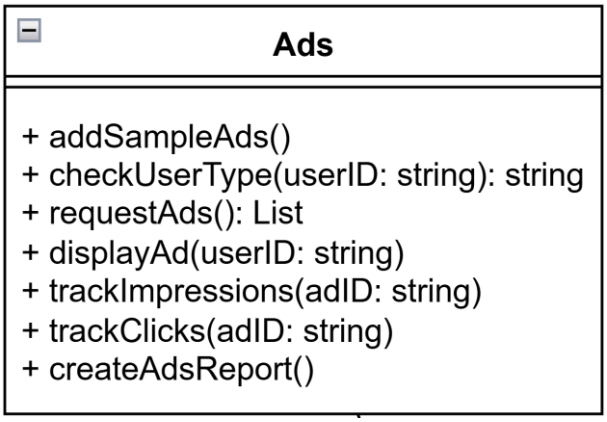
e. Class: Tutorial



The "Tutorial" class includes methods that help the system display tutorial information to users. At the same time, it also helps the system track user responses to improve the quality of the tutorial. This is extremely important to increasingly improve the friendliness and ease of use of the system.
More specifically, the "Tutorial" class includes the following methods:

- **getTutorialContent()**: This method helps the system query the database to get the entire tutorial content on how to use the system for users. However, the system will not display all this content but only display a part of the content needed for users through specific tutorial steps.
- **getUserProgress()**: This method helps the system query the database to get information about the user's learning progress. This helps the system display tutorial information that is appropriate to the user's learning progress. At the same time, it also serves as a utility function for switching between tutorial steps.
- **addTutorialStep()**: When the user clicks the "Next" or "Back" button on the tutorial interface, the system will call this method to display the next tutorial step or return to the previous tutorial step.
- **trackUserProgress()**: This method helps the system track and save the user's learning progress. This helps programmers analyze data to improve the quality of the tutorial.

f. Class: Ads

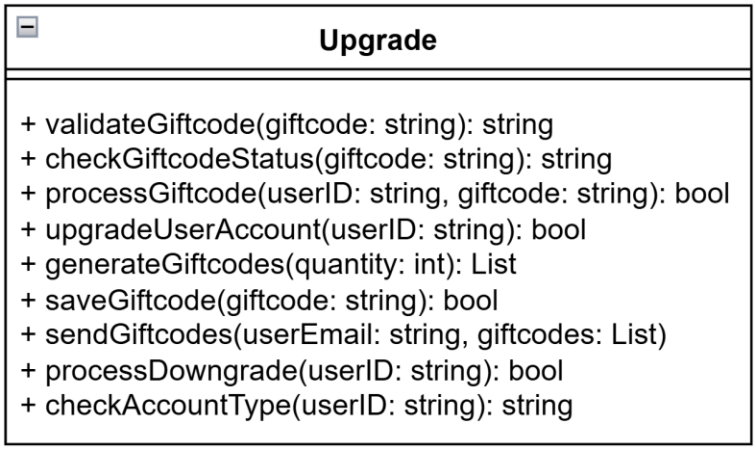


The "Ads" class will include services related to displaying advertisements from third parties on the website. For their advertisements to be displayed on the website, they will need to pay us a fee and commit that the contents of those advertisements do not violate any of our laws or policies. This is an additional source of income to help us maintain the operation of the website.
More specifically, this class will include the following methods:

- **addSampleAds()**: This method will add some sample ads to the website so that advertisers can preview how the ads will appear on our website. It acts as a prototype for the actual ads. Through this, advertisers can request us to edit the way the ads appear to suit their needs.
- **checkUserType()**: As mentioned before, only "FreeUsers" can see the ads appearing on the website. This method checks if the current user is a "FreeUser". If so, the ad will be shown, otherwise it will not.
    - Showing ads is not a good experience for the user, but it is a way for us to earn more revenue. But we will try to minimize the impact of ads on the user experience by showing ads appropriately and not bothering them too much.

- **requestAds()**: Retrieve data from the database to get information about the ads to display. This data will be stored in a list and returned to the display ads method.
- **displayAd()**: This method will display ads on the website. It will get information from the stored ads list and display them on the website. To turn off this ad, the user can click the "Turn off ads" button.
- **trackImpressions()**: For each time an ad is displayed, we will record an impression. This helps us know how many times the ad has been displayed and thus evaluate the effectiveness of that ad.
- **trackClicks()**: If a user clicks on an ad, we will record a click. This helps us know how many times the user is interested in that ad and thereby evaluate the effectiveness of that ad.
- **createAdsReport()**: This method will create a report on the effectiveness of the ad. This report will include information such as the number of impressions, the number of clicks, the click-through rate, the revenue from the ad, etc. This report will help us evaluate the effectiveness of the ad display and thereby make appropriate adjustments.

g. Class: Upgrade



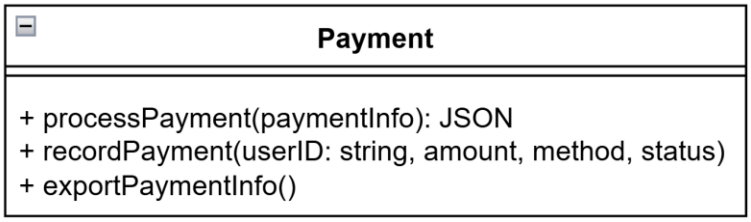
The "Upgrade" class will include methods to perform the upgrade of the user's account. Every time a user creates a new account, their account will default to a free account ("FreeUser"). By upgrading the account to "PremiumUser", the user will not be limited to the number of 3D model creations per day and will have a better user experience without being shown ads.

More specifically, the "Upgrade" class will include the following methods:

- **validateGiftcode()**: This method will check whether the code entered by the user is valid or not. A valid code will be determined by several criteria such as: the number of characters in the code, the starting characters of the code, etc.
- **checkGiftcodeStatus()**: This method will check whether the code entered by the user has been used or not. If the code has been used, the user will not be able to use that code to upgrade the account. In other words, each code can only be used once.
- **processGiftcode()**: This method will upgrade the user's account from "FreeUser" to "PremiumUser" by using the code. After upgrading the account, the code will be marked as used and cannot be used again.
- **upgradeUserAccount()**: This method will update the user's account information after they have upgraded their account from "FreeUser" to "PremiumUser". The user's account information will be updated in the system database.
- **generateGiftcodes()**: With "PremiumUser", they will be able to pre-order a number of codes to give to their friends, helping them upgrade their accounts. Users will be able to choose the number of codes they want to buy, and the system will generate random codes for them.
- **saveGiftcode()**: This method will store information about the code in the system database, including information such as: code, status (used or not), creation date, expiration date, etc.
- **sendGiftcodes()**: This method will send giftcodes that "PremiumUser" has purchased to their email address. All codes will be sent via email as a list, making it easy for users to share with friends.

- **processDowngrade()**: This is a feature only for "PremiumUser". This method will downgrade the user's account from "PremiumUser" to "FreeUser" and update their account information in the database. After downgrading the account, the user will no longer enjoy the benefits of "PremiumUser".
- **checkAccountType()**: Since the "downgrade" feature is only available for "PremiumUser", this method will check if the current user's account is "FreeUser" or "PremiumUser". If their account is "FreeUser", they will not be able to use the "downgrade" feature.

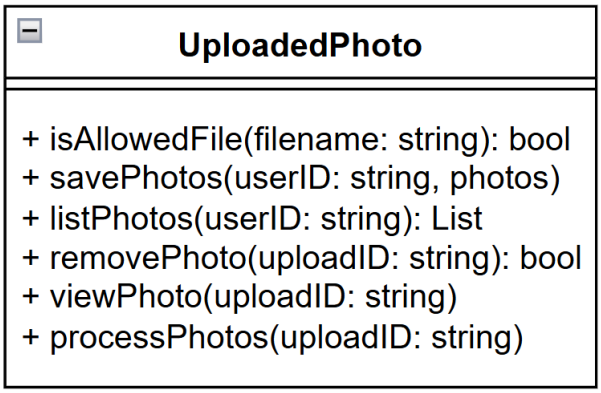
h. Class: Payment



The "Payment" class will provide payment-related functions, including:

- **processPayment()**: This method will enter the necessary information related to the user's transaction and make the payment. This payment process is quite complicated because we need to link with different payment gateways such as: PayPal, Stripe, Credit Card, etc. and need to comply with the security regulations of each payment gateway. If the payment process is successful, this method will return a JSON object containing information about the transaction, otherwise it will return an error message.
- **recordPayment()**: In case the payment process is successful, this method will be called to save information about the transaction to the database. Information about the transaction is very important to prove the validity of the transaction and help users to check their transaction information in the future.
- **exportPaymentInfo()**: This method will be called to export information about the transaction when requested by the user. This information will be exported as a PDF file or a web page so that the user can save or print it for future use.

i. Class: UploadedPhoto

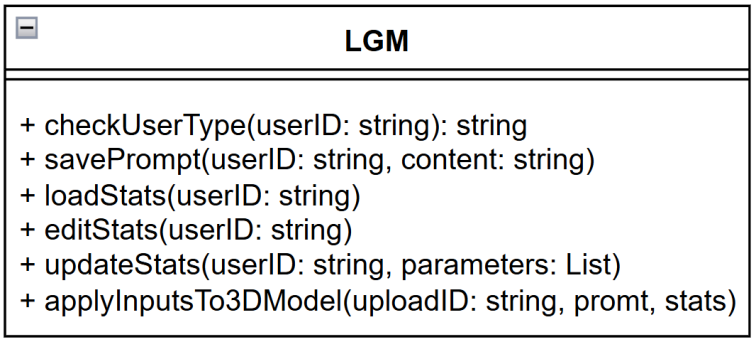


The "UploadedPhoto" class will contain methods to handle image files uploaded by users to the website. Specifically:

- **isAllowedFile()**: This method will check whether the uploaded image file is a valid image file or not. A valid image file is an image file with a standard format such as .jpg, .jpeg, .png, etc. This method will return true if the image file is valid and false if it is not valid.
- **savePhotos()**: This method will save the uploaded image files to the server storage directory. This method will return an array containing the relative paths of the stored image files. Users can use this path to display images on the website.
- **listPhotos()**: This method will return a list containing the image files previously uploaded by the user. Each image file will be represented by an object with properties such as file name, path, size, etc.

- **removePhoto()**: This method allows the user to delete a previously uploaded image file. In case the image file does not exist or cannot be deleted, this method will return false. Otherwise, if the image file has been successfully deleted, this method will return true.
- **viewPhoto()**: This method allows the user to view a previously uploaded image file through the browser.

j. Class: LGM

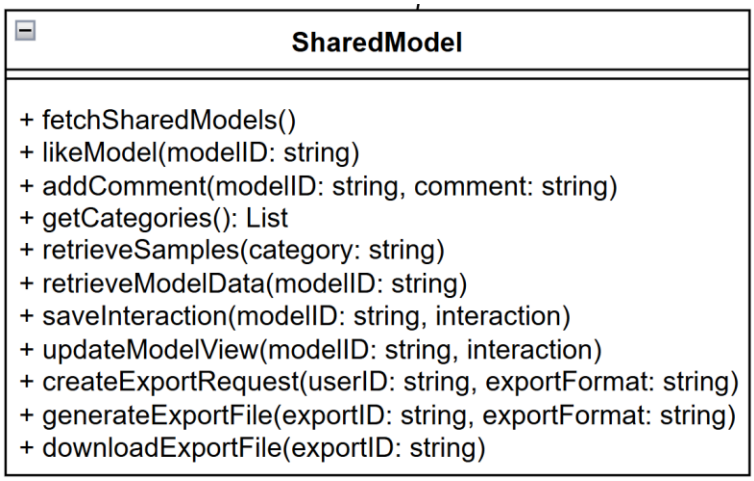


The phrase "LGM: Large Multi-View Gaussian Model for High-Resolution 3D Content Creation" is the name of an advanced AI model that can help users create a representation of a 3D model in the form of Gaussian Splat (.ply). The strength of this model is that its input data is quite diverse, Input can be only text, only image, or both image and text. If users are not satisfied with generated results, they can fine-tune the model by changing its parameters.

We will build a class "LGM" to do this with the following methods:

- **checkUserType()**: This method will check the user type. If the user is a "FreeUser", they will be limited in the number of times they can use the AI model within 24 hours. If the user is a "PremiumUser", they will not be limited in the number of times they can use the AI model.
- **savePrompt()**: This method will save the promt that the user has entered to prepare for creating a 3D model. There will be 2 types of promt here:
  - "promt": This promt will contain detailed description information about the 3D model that the user wants to create.
  - "negative promt": This promt will contain descriptive information, which are phrases that the user does not want their 3D model to have. By using promt and negative promt, the AI model will create a 3D model that meets the user's requirements.
- **loadStats()**: This method will load the default parameters of the AI model and display them through the web interface for the user. Through this, the user can customize these parameters to create the best 3D model.
- **editStats()**: This method will allow the user to edit the parameters of the AI model to create the best 3D model. Each parameter has a certain range of values, users can only choose values within that range.
  - However, users should learn carefully about these parameters before editing to understand their impact on the result.
- **updateStats()**: This method will update the parameters that the user has edited into the AI model. After updating, the AI model can use these new parameters to create the best 3D model.
- **applyInputsTo3DModel()**: This method will use all the information that the user has entered (uploaded image, promt, negative promt, edited parameters) to create a 3D model. We will call the API of the AI model that the author group has published on the Huggingface website to do this. The process of creating a 3D model may take a certain amount of time, during which the website will switch to the loading interface to notify the user.
  - Here we will use 2 different APIs. The first one is an API to help generate .ply file from image and promt. Second API will help convert Gaussian Splat (.ply) to Mesh (.glb). All intermediate and results will be stored on the server and can be downloaded by the user after the 3D modeling process is complete.

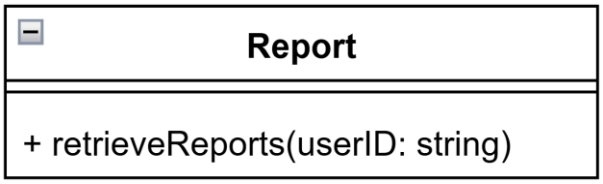
k. Class: SharedModel



The "SharedModel" class will contain methods that allow users to view and manipulate 3D models that have been shared by users on the system. These methods include:

- **fetchSharedModels()**: This method will access the database to retrieve data of 3D models that have been shared on the system.
- **likeModel()**: This method will add a like to the 3D model that the user selects. At the same time, it will update the number of likes of that 3D model in the database.
- **addComment()**: This method will add a comment to the 3D model that the user selects. At the same time, it will update the list of comments of that 3D model in the database.
- **getCategories()**: This method will access the database to retrieve a list of 3D model categories that the user can choose to view.
- **retrieveSamples()**: Based on the "category" selected by the user, this method will access the database to retrieve a list of sample 3D models belonging to that category.
- **retrieveModelData()**: This method will access the database to retrieve detailed data of a 3D model requested by the user.
- **saveInteraction()**: This method will perform the task of saving the actions that the user performs with the 3D model. These actions include rotating, zooming in, zooming out, and moving the 3D model.
- **updateModelView()**: Based on the actions that the user performs, this method will update the view of the 3D model on the user interface.
- **createExportRequest()**: This method will create a request to export the 3D model selected by the user. This request will be sent to the system for processing.
- **generateExportFile()**: This method will generate the 3D model export file as a 3D file or image file depending on the user's request.
- **downloadExportFile()**: This method will download the 3D model export file to the user's computer. At the moment, this download is completely free for both free and paid users.

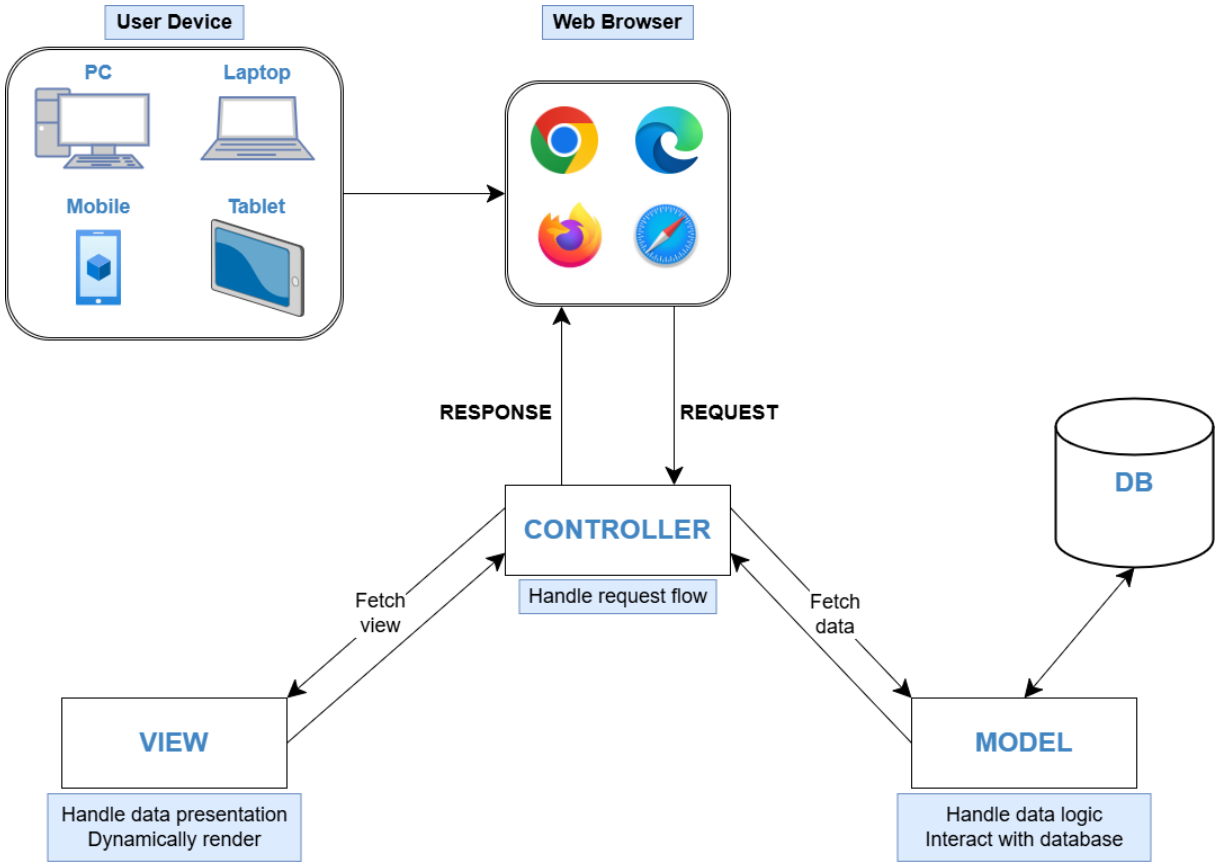
l. Class: Report



- The "Report" class will contain methods to help users generate reports based on their activity data on the website.
- To do this, the system will use each user's unique identifier, access the database to get the necessary data, and then use the supporting libraries to generate reports in PDF or Excel file format.

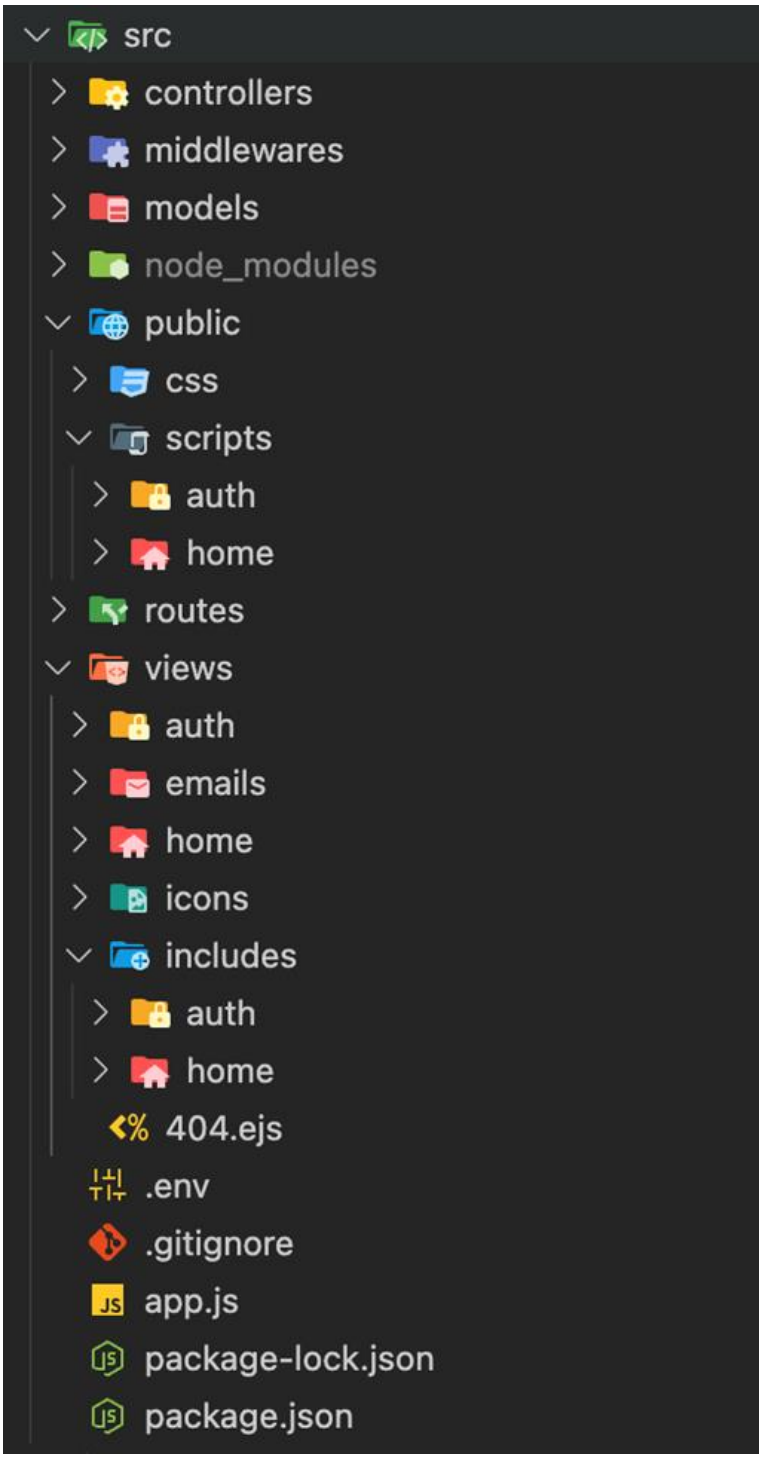
- Generating these reports will help users easily track and evaluate their activities on the website, thereby being able to adjust their behavior to achieve the highest efficiency.
- In the future, the system will be further improved to support users in generating reports according to their specific requirements, thereby helping them have the most comprehensive and detailed view of their activities on the website.

5. Deployment



- We aim our users can access the web application on any device (Laptop, PCs, Mobile, Tablet, etc.), any browser (Google Chrome, Microsoft Edge, Firefox, Safari, etc.) and compatible operating systems like Windows, MacOS, Linux.
- To cover this, we need to create responsiveness from any scale of screen and use libraries that support mostly browsers. But we will prioritize optimal development on Laptops and PCs first.

6. Implementation View



- **Controllers**: handle incoming requests from the user, interact with model to retrieve or manipulate data, and return the appropriate view (response) to the user. Controllers help separate the application's logic.
- **Middlewares**: authenticate user session, error handling, etc.
- **Models**:
  - **User model**: basic information of a user (name, email, password, role, etc.)
- **Public**: contains CSS and JavaScript files, responsible for the styling and interactivities of the webpage

- **Routes**: handle HTTP requests. Each route specifies the path and HTTP method (GET, POST, PUT, DELETE, etc.) that the application should response to, and it links to the corresponding controller function.
- **Views**: responsible for rendering the user interface. Views typically represent the presentation layer of an application, displaying dynamic content to the user. They are generated by template engine (in our application EJS).
  - **Includes**: stores the layout templates
- **App.js**: is the main entry point of the application, where the app is set up, middleware is applied, routes are defined, and the server started