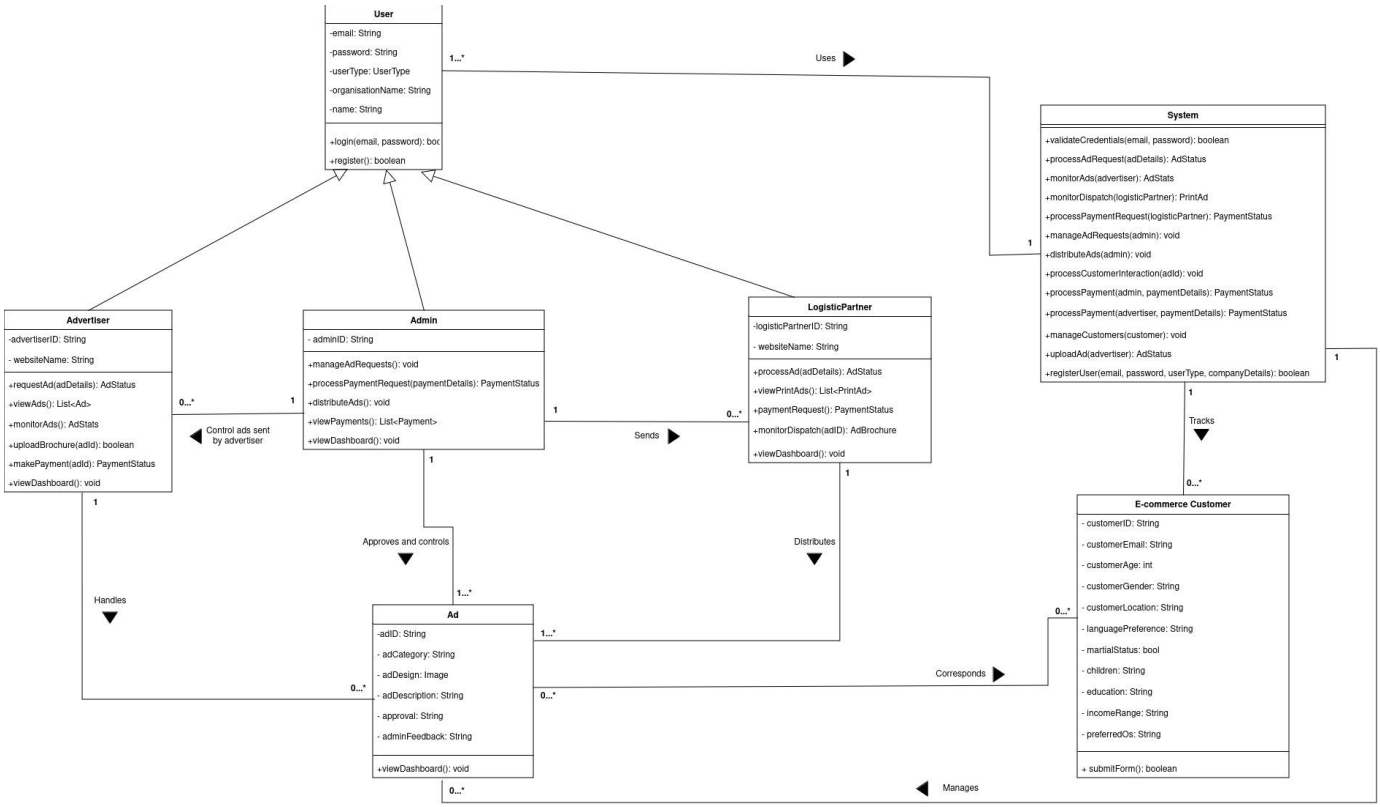
Product Design

Team 36

Sai Praneeth Bommana - 2022101097
Sai Sudhan Kunapareddy - 2022101052
Chanaganti Venkata Karthikeya - 2022101058
Kriti Madumadukala - 2022101069
Kavuri Vivek Hruday – 2022114012

Design Model



[Link for the uml diagram in draw.io]

User	**Class State:** This class holds information of all registered users such as – email, password, userType, organisationName, name. **Class behaviour:** • login() : boolean – Authenticates a user and logs in the user • register() : boolean – Registers a new user

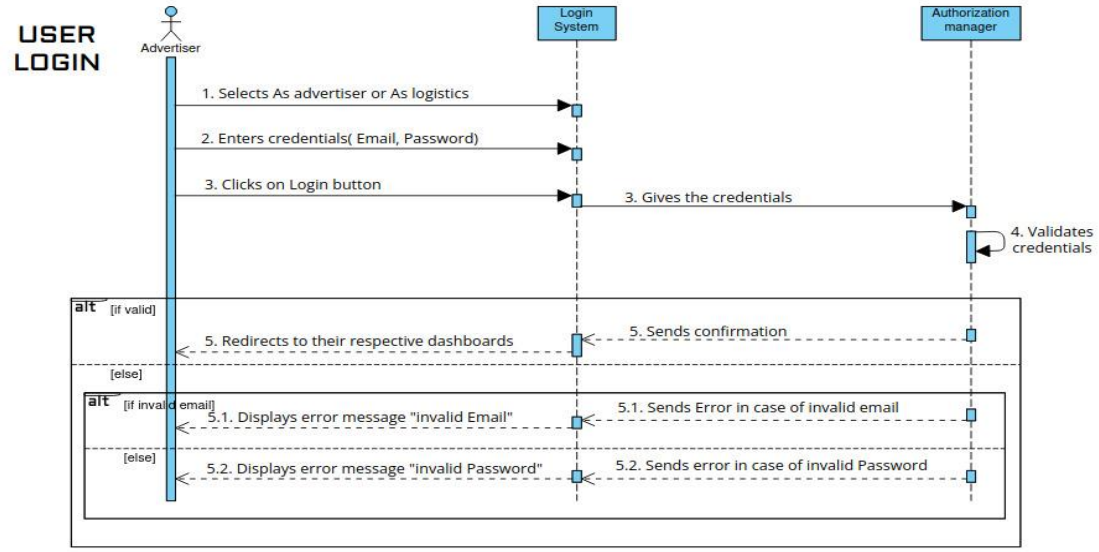
Admin	**Class state:** This class is a subclass of the class “User”. This holds all the information which “User” class holds. Along with it, it carries adminID, which is unique for each admin. **Class behavior:** Since, this is a subclass of “User” class, it exhibits all the methods exhibited by the “User” class, along with it, • manageAdRequests(): void - Manages incoming ad requests. • paymentRequestProcess(paymentDetails) - PaymentStatus: Processes payment requests. • distributeAds(): void - Manages the distribution of accepted ads. • viewPayments(): List <Payment> - Retrieves a list of payment details.
Advertiser	**Class state:** This class is a subclass of the class “User”. This holds all the information which “User” class holds. Along with it, it carries advertiserID, which is unique for each advertiser, website, which is the website that advertiser gives. **Class behavior:** Since, this is a subclass of “User” class, it exhibits all the methods exhibited by the “User” class, along with it, • RequestAd(adDetails): AdStatus - Requests to publish an ad with specified details. • viewAds(): List <Ad> - Retrieves a list of ads associated with the advertiser. • uploadBrochure(adId): boolean - Uploads a brochure for a specific ad. • paymentRequestProcess(paymentDetails): PaymentStatus - Initiates a payment for a specific ad. • adsMonitoring(): AdStats - Monitors ad interactions and statistics.
Logistic Partner	**Class state:** This class is a subclass of the class “User”. This holds all the information which “User” class holds. Along with it, it carries advertiserID, which is unique for each advertiser, website, which is the website that advertiser gives. **Class behavior:** Since, this is a subclass of “User” class, it exhibits all the methods exhibited by the “User” class, along with it, • processAd(adDetails): AdStatus - Processes ads for printing. • viewPrintAds(): List<PrintAd> - Retrieves a list of ads to be printed. • paymentRequest(): PaymentStatus - Requests payment for logistic services. • AdsDispatchMonitor(adID): AdBrochure - Monitors the dispatch of printed ads.

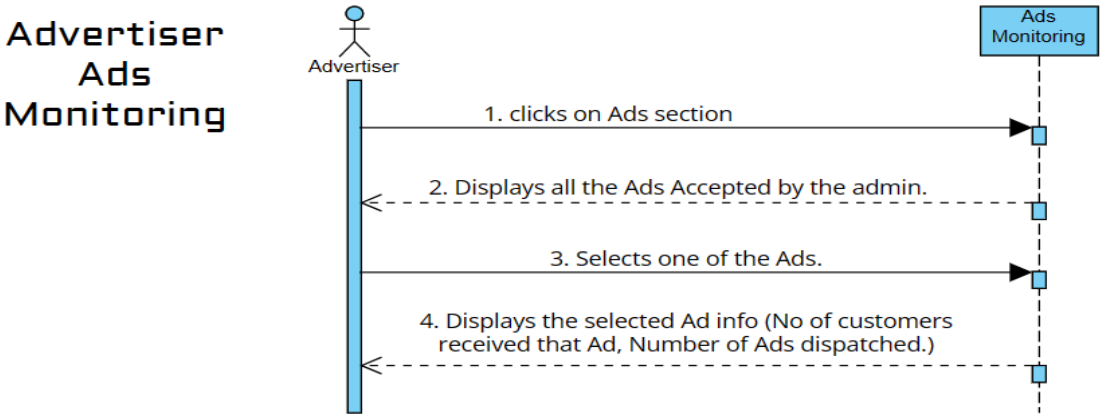
Ad	**Class state:** This class holds the information of advertisements, such as – • adID – It is a unique attribute to every Ad. • adCategory – This gives the category of the Ad to which it belongs, ex: Electronics, Home needs, etc... • adDesign – This contains the image of actual Ad. • adDescription – This is the description, that would be given by advertiser. **Class behavior:** This class does not exhibit explicit methods, but the other classes' methods are used to change the data in this class.
System	**Class state:** This class does not explicitly hold any information but is used for interactions between different classes. **Class behavior:** The following methods are implemented by this class: • validateCredentials(email, password): boolean – This method takes in the email and password input by a “User” and returns true if they match with the database records, else returns false. • processAdRequest(adDetails): AdStatus – This method takes in the details of an Ad and returns the status of the Ad (0 – Awaiting review, 1 – Approved, 2- Declined) • monitorAds(advertiser): AdStats – This method takes in the advertiser and returns the stats of the Ads of that advertiser. • monitorDispatch(logisticPartner): PrintAd – This method takes in the logistic partner and prints all the Ads that are needed to be dispatched by that logistic partner. • processPaymentRequest(logisticPartne): PaymentStatus – This takes in a logistic partner and returns the payment status of the admin, to that respective logistic partner. • manageAdRequests(admin): void – This updates the databse with the updated data of approved/declined status of the advertisement. • distributeAds(admin): void – This distributes advertisements that need to be distributed among the logistic partners. • processCustomerInteraction(adId): void – This increases the count of the number of interactions of that particular adID, when the customer scans the QR code on the brochure. • processPayment(admin, paymentDetails): PaymentStatus – This processes the payment that the admin makes, given the admin and payment details. • processPayment(advertiser, paymentDetails): PaymentStatus - This processes the payment that the advertiser makes, given the admin and payment details.

	• manageCustomers(customer): void – This manages the data of the customers. • uploadAd(advertiser): AdStatus – This uploads an accepted ad. • registerUser(email, password, userType, companyDetails): boolean – This method registers a new user.
E-commerce Customer	**Class state:** • customerEmail – Email ID of the customer • customerAge – Aget of the customer • customerGender – Gender of the customer • customerLocation – Location of the customer • languagePreference – Language preferred by the customer • maritalStaus – Customer's marital status • children – Number of children of the customer • education – Education of the customer • incomeRange – Income range of the customer • preferredOS – OS preferred by the customer **Class behavior:** The following method is implemented by this class: • + submitForm(): Boolean – This submits the form filled by the customer and displays a message as submitted if successfully submitted.

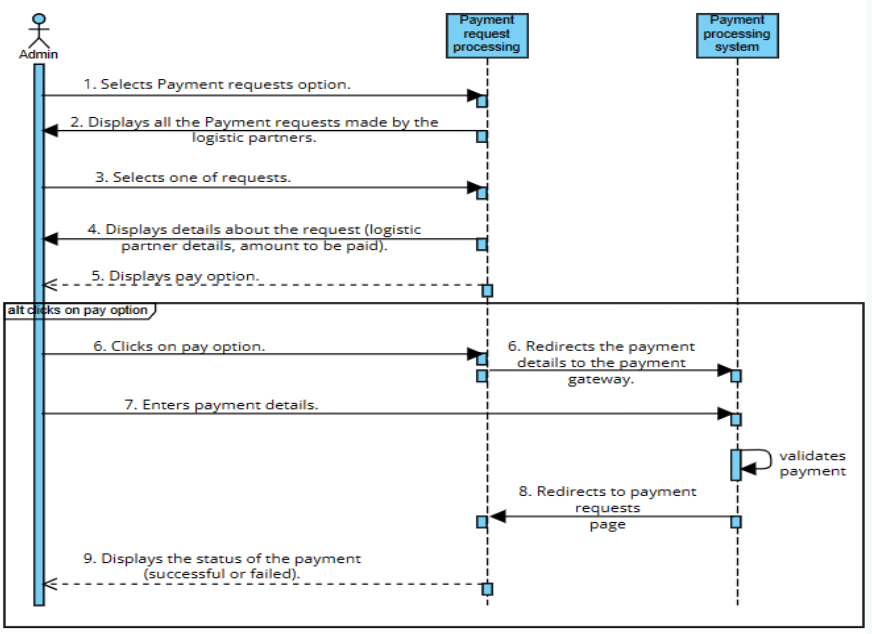
.

Sequence Diagram(s)

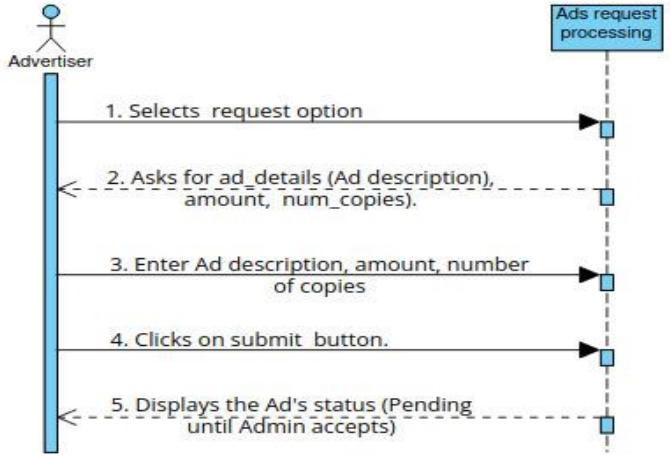




processing payment requests



Advertiser Ad requests



Design Rationale

Images storage in database vs file-system

For this initial version, we have implemented the storage of images of ad brochures inside the same document as ads in the database itself for easier accessibility. There were long waiting times / response times for each HTTP API request to the server as it is also transferring binary encoded data of the image present in the database.

Cons of using database

- Extra Computation for encoding and decoding the binary data
- Large size in HTTP payload (might be transferring megabytes of binary data)
- Double bandwidth usage since we are using a Cloud Database (MongoDB Atlas)
- Other data from the ad will be blocked from reaching the client, waiting for the whole payload to be received.

Whereas by using file-system, images are stored in the same place where the server is hosted. This way, we can just store the image paths (paths in system) in the database. This enables the users to directly access images from the public folder instead of sending them over the same HTTP payload.
So, this is better compared to the previous implementation

Categorical Mapping

The initial plan is to have one category for each ad, but users can have multiple categories which they are interested in. Users will be mapped with ads according to the advertisement category. But this is essentially narrowing down the range of users who are being exposed to that specific ad, and it is very possible that potential customers are being missed out. Because generally, an ad can have multiple categories.

And for customers, instead of giving equal value to all their categories, we can put their categories in a priority order based on their responses and interactions.

Extending Form Builder

Initially, there is this feature called, “Form Builder” which allows Admin to add new questions to the existing form. Further implementation would include some algorithm / an ML algorithm which generates questions and adds to the form.

Considering this, we thought of extending this functionality to also remove or edit questions from the form. This will allow more flexibility for the admin or the algorithm in managing the questions in a better way.

Ad Publishing Process

Initially, we thought of a two-step publishing process for ad requests. The implementation, follows that,

1. Advertiser uploads ad details, waits for approval.
2. Admin approves / rejects. If approved, a unique link is generated (for the QR to be printed).
3. System sends this unique link to the advertiser
4. Advertiser create QR with the link, and uploads the final brochure
5. The Ad is published.

This flow will require double interactions from the advertiser to publish an ad. This might become inconvenient, or inefficient in terms of time, resources.
So, in the new flow, we will put an extra operation at the end of Logistics Partners. For each ad, along with the print design, we will also require them to print the QR on the back-side of the brochure. With this flow, we will be extracting the QR complexity from Advertiser’s perspective, and putting an overhead at the end of Logistics Partner. This flow has its pros and cons. There is a trade-off in terms of user experience, and overhead at Logistics Partner. But atleast the flow will be smooth compared to the previous one. This must be discussed with the client and finalized later.