“METAGRAM”

Project submitted in the partial fulfillment for the award of degree for

Master of Computer Application

Submitted By

Mukesh Sharma(2270205)

Under the Guidance of:

Dr. Rabindra K Barik

Associate Professor, SCA

KIIT, Bhubaneswar

School of Computer Applications

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Bhubaneswar, Odisha

April 2024

CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled **Metagram** submitted to **School of Computer Application, KIIT University** in partial fulfillment of the requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS (MCA) ,** is an authentic and original work carried out by **Mr. Mukesh Sharma** Roll no. 2270205 and registration no. 22281888522 and **Mr**. **Sriman Sandip Dash** Roll no. 2270366 and Registration no. 22294788651 under my guidance.

The matter embodied in this project is genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfillment of the requirements of any course of study.

…….. .....…….

Signature of the Student: Signature of the Guide:

Date: …….. Date: …….

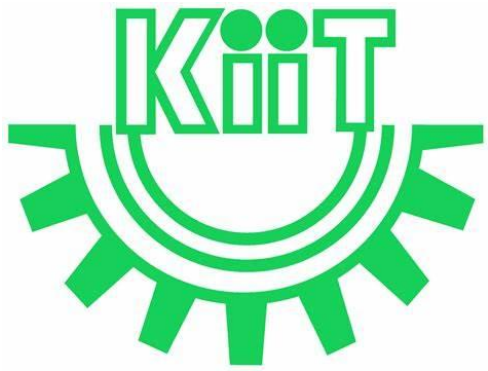
Name:

Designation:

KALINGA INSTITUTE OF INDUSTRIAL

TECHNOLOGY

DEPARTMENT OF COMPUTER APPLICATIONS



CERTIFICATE

This is to certify that the project entitled **'METAGRAM'** submitted in partial fulfillment of the requirement of the degree of Master's of Computer Applications is a result of the bonfires work carried out by **Mukesh Sharma bearing roll no. 2270205 and Sriman Sandip Dash bearing roll no. 2270366,** is an authentic and original work. During the academic year 2022-2024

.....
Signature

(Internal Examiner)

Date.....

.....
Signature

(External Examiner)

Date.....

Director,
Dr. Veena Goswami

DECLARATION

I, **Mukesh Sharma** , roll no **2270205** do hereby declare that the project report entitled **Metagram** submitted to **School of Computer Application, KIIT University, Bhubaneswar** for the award of the degree of **MASTER OF COMPUTER APPLICATION (MCA) ,** is an authentic and original work carried out by group from 20th Jan 2024 to 25th Mar 2024 at **School of Computer Application, KIIT University** under the guidance of **Dr. Rabindra K Barik**.

.....

Signature of the student

Date.....

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. I would like to extend my sincere thanks to all of them. I am highly indebted to Dr. Rabindra K. Barik and Dr. Veena Goswami for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project. I would like to express my gratitude towards my parents & friends for their kind co-operation and encouragement which help me in completion of this project.

ABSTRACT

Metagram presents a novel social media platform designed to foster connections and facilitate content sharing. Built with React and Typescript for a robust frontend, Metagram leverages Tailwind CSS for a modern and responsive design. Utilizing ShadCn's and Appwrite design expertise, the platform boasts an intuitive and user-friendly interface with robust database.

Core functionalities include user authentication through registration and login. Users can curate their profiles and leverage CRUD operations (Create, Read, Update, Delete) for their posts. Interaction features encompass saving and liking posts, alongside the ability to follow other users and build networks. Infinite scrolling ensures a seamless feed exploration experience, while a comprehensive search function empowers users to locate specific content and connect with individuals. Zod integration streamlines React form validation, maintaining data integrity. Appwrite, a backend-as-a-service platform, provides secure user authentication and database storage for user information and posts. This document outlines Metagram's functionalities, technology stack, architecture, and potential avenues for future development.

Table of Contents

Figure Content

Introduction to Metagram: A Social Media Platform

Metagram is a basic but dynamic and engaging social media platform designed to connect users, foster meaningful interactions, and provide a seamless experience for sharing content. Whether you're an avid content creator, or simply looking to stay connected with friends and family, Metagram has you covered.

Metagram ensures secure access through robust user authentication. Your account is protected, and you can confidently explore the platform. Users can create and manage their posts effortlessly. Share your thoughts and photos with your followers. Bookmark or save posts you love or express appreciation by liking them. Grow your network by following friends, influencers, or anyone whose content resonates with you. Say goodbye to pagination! Metagram features infinite scrolling, allowing you to seamlessly explore an endless stream of content without interruption. Easily find posts and users using our intuitive search feature. Whether you're looking for specific content or discovering new profiles, Metagram's search capabilities have you covered. Edit your profile to reflect your personality. Upload a profile picture, write a bio, and showcase your interests. Your profile is your digital identity on Metagram.

I've leveraged React with Type-Script to build a responsive and interactive user interface. Reacts component-based architecture ensures flexibility and scalability. Tailwind CSS provides a utility-first approach to styling, allowing us to create consistent and visually appealing designs. ShadCn adds depth and elegance to our UI components, enhancing the overall user experience. Zod ensures data integrity and validates user inputs on the front-end. I've chosen Appwrite, a powerful backend-as-a-service platform, for managing user accounts, data storage, and authentication. Appwrite simplifies back-end development, allowing us to focus on delivering features to our users. Metagram is all about making connections, fostering creativity, and empowering you to share your story with the world.

System Analysis

❖ Identification for Need:

- **Existing System requirement:** Current situation is that peoples are dependent on Social media platform for sharing contents like blogs, pictures, thoughts and etc. So I have create a platform where people can connect with one another and share anything.

- **User-friendly technology:** In today's digital age, people want a user-friendly and accessible platform to use without any hassle.

❖ Feasibility Study:

A feasibility study is a critical initial step documented in project planning. It's essentially an assessment of a proposed project's viability, investigating various factors to determine if it's worth pursuing. This analysis helps decision-makers weigh the pros and cons before committing resources.

● Technical Feasibility:

The key aspects to consider when evaluating the technical feasibility of your social media website like Metagram. React with Typescript Used for building the user interface and handling user interactions. Typescript ensures type safety and improves code maintainability. UI design expertise is reflected in the user-friendly and intuitive interface. Backend-as-a-Service (BaaS) solutions like Appwrite simplify development by providing pre-built functionalities for user authentication, database storage, and APIs. Ensure secure user registration, login, and password storage with strong encryption mechanisms. Optimize website code and leverage caching mechanisms to ensure fast page load times across different devices and network conditions.

By carefully considering these technical feasibility aspects, we can ensure the social media website has a solid foundation for growth and success. It's also important to stay updated on the latest advancements in web technologies and adjust your approach accordingly.

● Economical Feasibility:

Economic feasibility assesses the financial viability of Metagram, determining if it's a profitable venture.

Factor in ongoing expenses like server maintenance, database fees, content moderation (if outsourced), and marketing campaigns though I'm not marketing now so its almost cost free or with minimal cost expenditure. A hybrid approach with a free basic tier and a premium tier with additional functionalities can attract users and generate revenue from those willing to pay for more once deployed to market. Calculate the point at which Metagram's revenue equals its total costs (break-even point). This will provide an estimate of how many users you need to acquire and monetize to become profitable. Consider the long-term growth potential of Metagram and how user base expansion might translate to increased revenue. Develop a long-term plan for maintaining Metagram's financial health and adapting to evolving market trends.

By conducting a thorough economic feasibility analysis, we can make informed decisions about resource allocation, revenue generation strategies, and the overall financial viability of Metagram.

● Operational Feasibility:

Operational feasibility assesses how well your social media platform, Metagram, integrates with your existing infrastructure, team, and business processes to function smoothly.

I'm the only developer working with the chosen technology stack (React, Typescript, Tailwind, Appwrite). Server infrastructure have the capacity to handle the anticipated user base and high traffic volume. The ongoing maintenance and optimization is not needed for my chosen database solution because Appwrite provides 99.98% uptime. The availability of development resources and potential dependencies to ensure a smooth development process. Allocate sufficient time for thorough testing, bug fixing, and deployment procedures. Plan for ongoing bug fixes, security updates, and feature enhancements to keep Metagram functioning optimally. Monitor key performance indicators (KPIs) like website uptime, response times, and user engagement to identify areas for improvement.

By addressing these operational feasibility factors, we can ensure Metagram is not just technically possible but also practical to implement, manage, and maintain within your existing resources and business processes

❖ Software Requirements Specification (SRS)

➢ 1. Introduction

The purpose of this SRS document is to outline the requirements for the user account management system of a social media application (Metagram). The system aims to handle user registration, validation, and profile management.

➢ 2. System Overview

The social media application allows users to create accounts, log in, and manage their profiles. It also includes an administrative interface for system maintenance.

➢ 3. Functional Requirements

1. User Registration (1.0 Social Media Application)

I. Users provide account information (e.g., username, email, password).
II. The system validates and verifies the account.

2.Authentication (2.0 Authenticate User Account)

I. Users log in using their credentials.
II. The system authenticates user accounts.

3.Profile Management (3.0 Social Media Application)

I. Users can edit their profiles (e.g., update profile picture, change bio).
II. Users can post content (e.g., text, images, videos).
III. Users can manage their uploads (e.g., delete posts, edit their post).

4.System Administration (4.0 Update System Information)

I. System administrators manage the database.

II. Admins can perform maintenance tasks (e.g., backup, restore).

➢ 4. Non-Functional Requirements

1.Security

I. User data must be securely stored and transmitted.

II. Authentication mechanisms should prevent unauthorized access.

2. Performance

I. The system should handle concurrent user requests efficiently.

II. Response times should be reasonable.

3.Usability

I. The user interface should be intuitive and user-friendly.

II. Error messages should be clear and helpful.

4.Scalability

I. The system should accommodate future growth (e.g., increasing user base).

5.Reliability

I. The system should be available and reliable.

II. Backup and recovery procedures should be in place.

➢ Data Requirements

I. **User Account Database**: Stores user profiles, authentication data, and user-generated content.

➢ Assumptions and Constraints

1.Assumptions:

I. Users have basic knowledge of using social media applications.

II. The system will be developed using modern web technologies.

2.Constraints:

I. Development must adhere to a specific budget and timeline.

➢ 7. Dependencies

I. The social media application relies on external services (e.g., email providers for account validation).

Conclusion:

This Software Requirements Specification (SRS) document has outlined the functional and non-functional requirements for the user account management system of a social media application. It covers functionalities like user registration, authentication, profile management, and basic administrative features.

This SRS serves as a foundation for further development activities. Stakeholders, including developers, designers, and system administrators, can utilize this document to understand the system's purpose, functionalities, and limitations. The document also highlights technical considerations, data requirements, and dependencies for a holistic understanding of the project.

❖ Software Engineering Paradigm Applied:

The software engineering paradigm which is also referred to as a software process model or software development life cycle (sdlc) model is the development strategy that encompasses the process of identifiable stages that a software product undergoes during its lifetime that is a description of software in a systematic and disciplined manner. The life cycle mode used to develop this project is agile model.

● Agile Model :

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release. Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer. And every iteration in performed after 2 weeks of span.

- **Iteration 1 :**

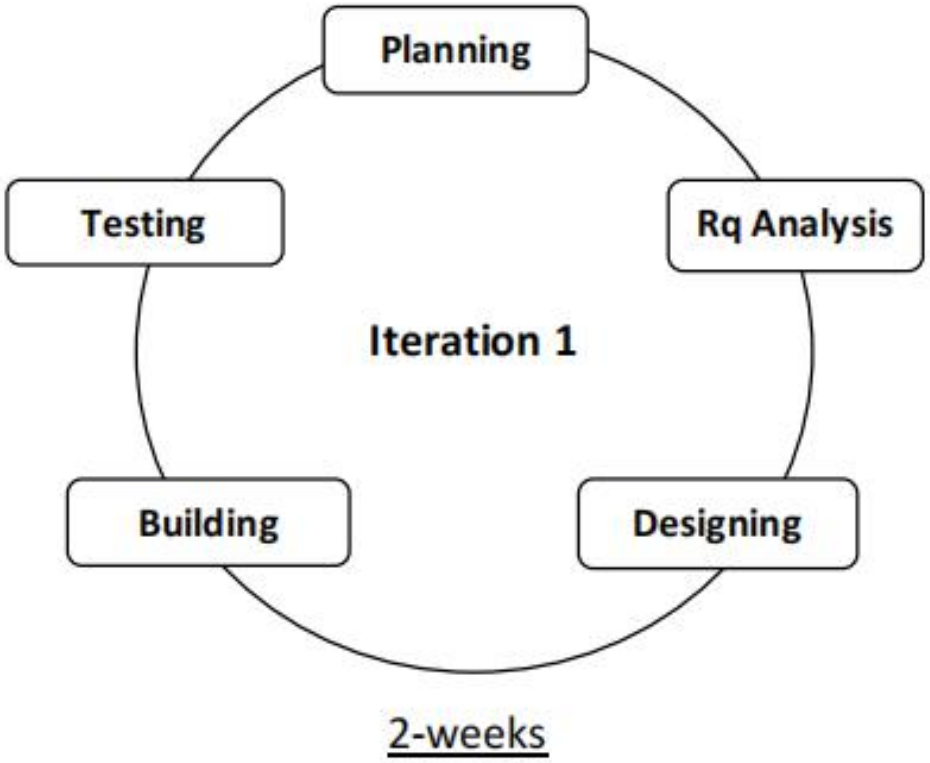


Figure 1.0 (2 week of iteration of Agile model)

- **Bug fixed:**
  - Fixed Some UI Alignment Issue in Registration Page and Scroll bar.
  - Added Default Image in login and sign-up page.

- **New Functions:**
  - Integrated Appwrite for Registration and validation purpose.

- **Iteration 2 :**

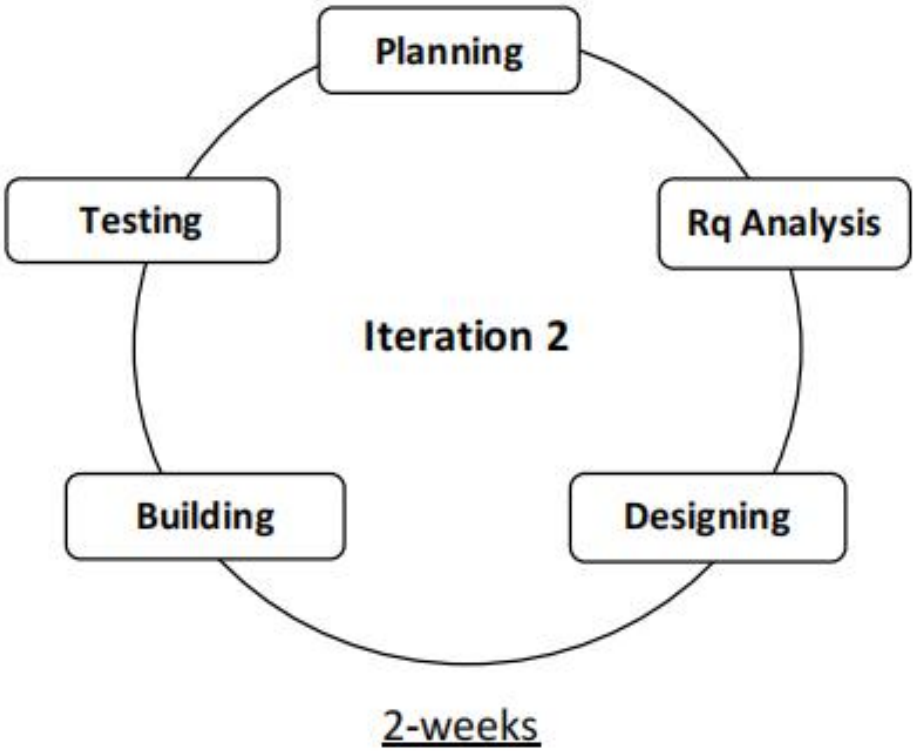


Figure 1.1 (4 weeks of iteration of Agile Model)

- **Bug fixed:**
- Fixed UI Alignment Issue in home for website logo .
- Fixed routing for login to home page.

- **New Functions:**
- Functional Log out button was added.
- Added other required pages and setup their routing.

- **Iteration 3 :**

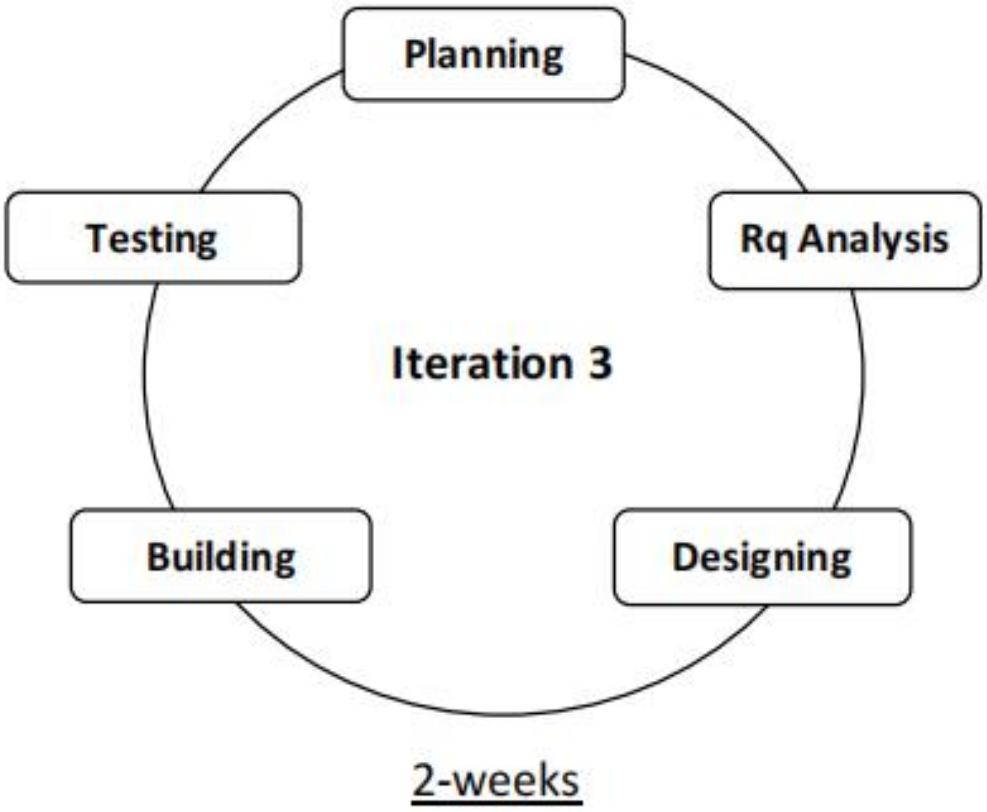


Figure 1.2 (6 weeks of iteration of Agile model)

- **Bug fixed:**
  - Fixed UI Alignment Issue in home page for post card .
  - Fixed coloring issue in left navigation bar.

- **New Functions:**
  - Created upload or drop image function in create post page.
  - Build add location function for adding location to posts.
  - Make all the fields in create post page mandatory.
  - Added Search bar in Search page and Explore page.

- **Iteration 4 :**

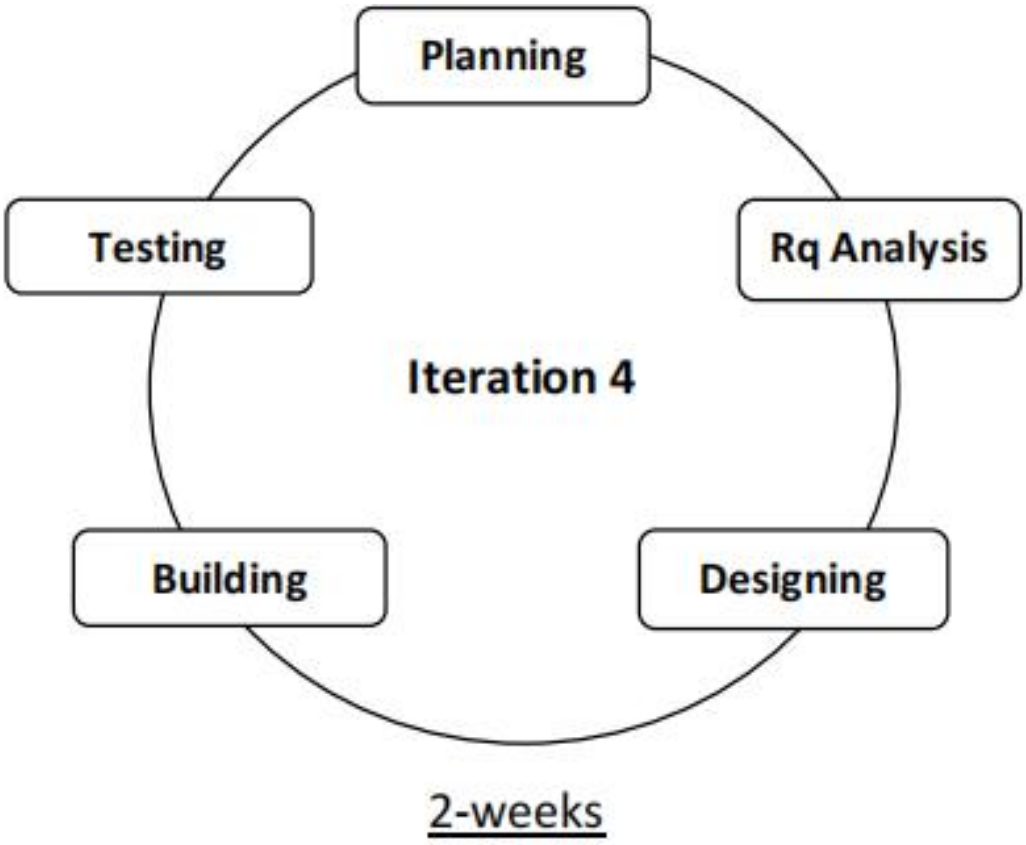


Figure 1.3 (8 weeks of iteration of Agile model)

- **Bug fixed:**
- Fixed saved and unsaved function issue in Appwrite.
- Updated Appwrite for CRUD operations.
- Fixed issue in searching results in Search post page.

- **New Functions:**
- Created a post card component for multiple use.
- Added a Top creator right nav-bar in home page and use post card for listing user profiles.
- Added User profile(name, username and picture) on the left navigation bar.
- Added Search bar in Explore page.

- **Iteration 5 :**

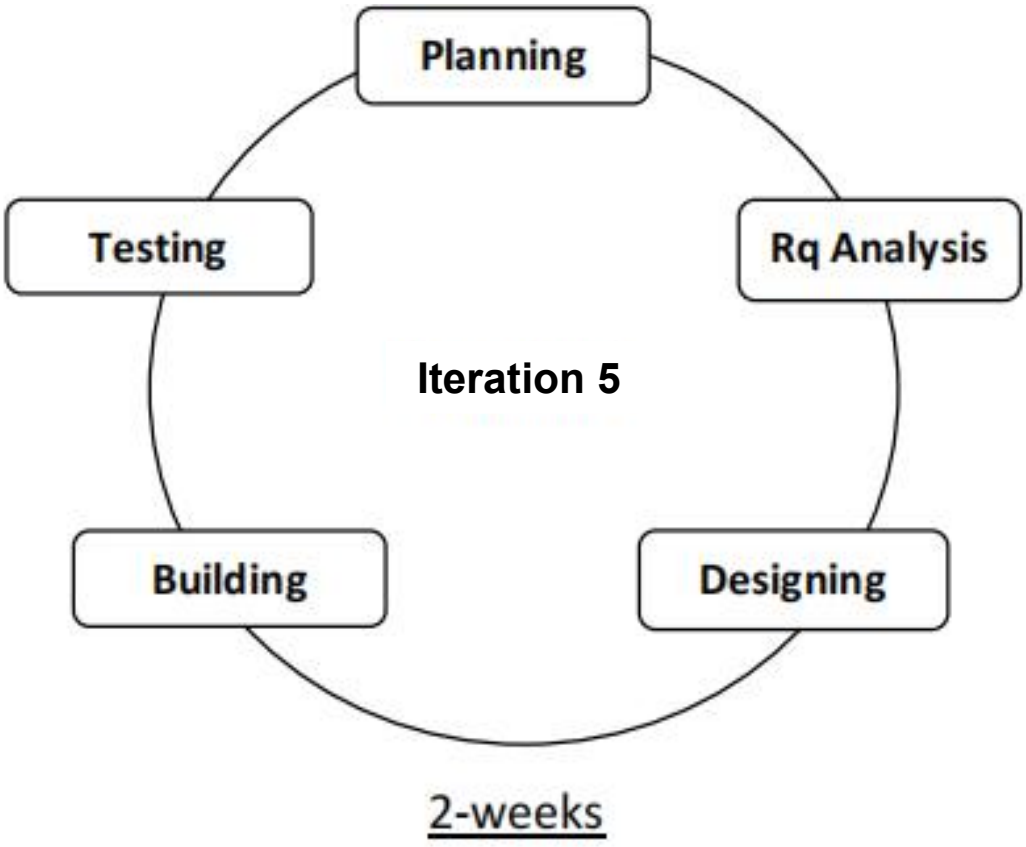


Figure 1.4 (10 weeks of iteration of Agile model)

- **Bug fixed:**
- Fixed issue with profile picture container on the home page for image fit.
- Fixed Edit profile button on user's profile page.
- Updated some functions in Appwrite for media storage.

- **New Functions:**
- Added saved and liked posts by user in the profile page.
- Added Infinite Scrolling functionality.
- Added default posts, followers and following in profile page.
- Added Follow button in profile page.

- **Iteration 6 :**

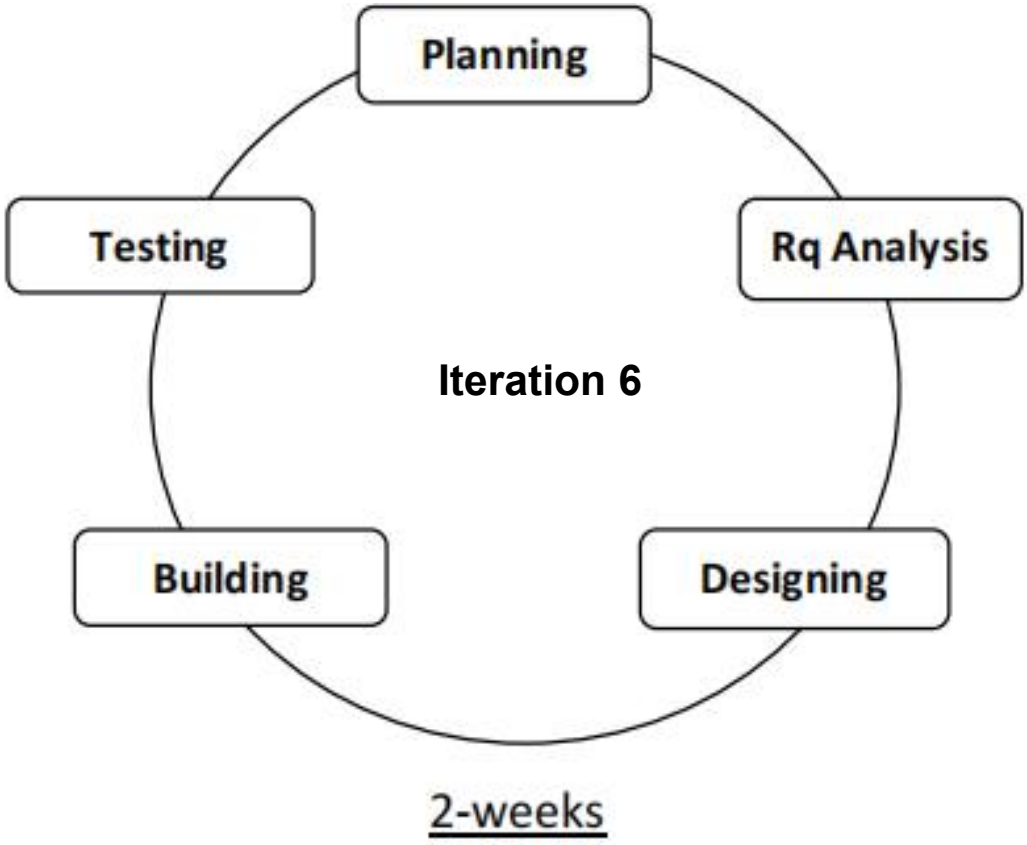


Figure 1.5 (12weeks of iteration of Agile model)

- **Bug fixed:**
  - Fixed path routing issue for post image.
  - Fixed data query for infinite scrolling.
  - Fixed UI alignment issue for small devices.
  - Fixed timing issue for individual posts.

- **New Functions:**
  - Added home page path in the logo.
  - Added bio / about in the edit profile page.

For every iteration for this project is developed under the following steps:

- ***Planning:***

This phase is the very first step of system development life cycle. The objective and goal of system is determined, and the system requirement are considered. An estimate of resources, such as software requirement, hardware requirement that along with the system are proposed. Once the current process requirement is done, then it will process to the next phase which is requirement analysis phase.

- ***Requirement Analysis:***

The goal of this phase is to understand the exact requirements of the customer and to document them properly (SRS).

- ***Design:***

The goal of this phase is to transform the requirement specification into a structure that is suitable for implementation in some programming language.

- ***System Building Phase:***

In this phase will start coding for the system. React with Typescript and Appwrite is used for front and back-end respectively. The system will be used for interacting, sharing contents and making communities on the basis on the outcome from system design.

- ***System Testing Phase:***

In this phase, the full testing toward the develop system are performed. This is to ensure that the system has meet the requirements and objectives. Besides that, it is also to ensure that the system is free from error and bug.

❖ Data Flow Diagram(DFD):

Data flow diagram is graphical representation of flow of data in an information system. In a DFD, processes, data stores, and data flows are depicted to illustrate how data moves between different components of the system. In Metagram, the User register and add personal detail to sign up, to develop its levels. DFD's depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

• *Types of DFD*

Data Flow Diagrams are either Logical or Physical.

➢ *Logical DFD:*

This type of DFD concentrates on the system processes, and flow of data in the system. For example, in a banking software system, how data is moved between different entities.

➢ *Physical DFD:*

This type of DFD shows how the data flow is actually implemented in the system. It is more specific and closer to the implementation.

• *DFD Components*

DFD can represent source, destination, storage and flow of data using the following set of components-

Figure 2.0 (DFD Components)

➢ *Entities:*

Entities are source and destination of information data. Entities are represented by a rectangle with their respective names

➢ *Process:*

Activities and action taken on the data are represented by circle or round-edged rectangles.

➢ *Data Storage:*

There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open sided rectangle with only one side missing.

➢ *Data Flow:*

Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

• *Context level DFD*

The context level DFD is showing the overall process which can be seen in a system, so that the new authorized user can easily and flawlessly interact with the **Metagram** system and operating all the process in this system.

This interaction has shown in the context level DFD below:

◆ **Level-0 DFD :**

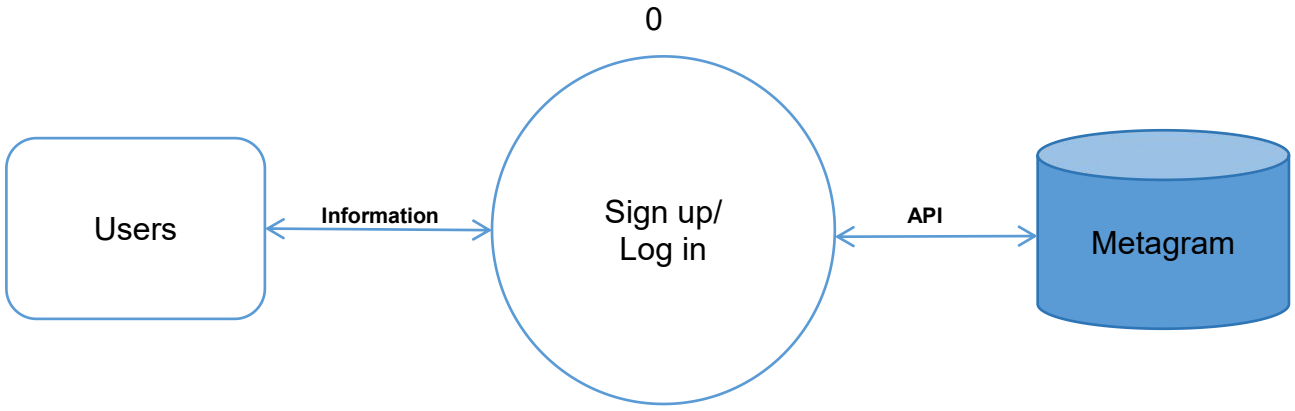


Figure 2.1 (DFD Level-0 of Metagram)

◆ **Level-1 DFD :**

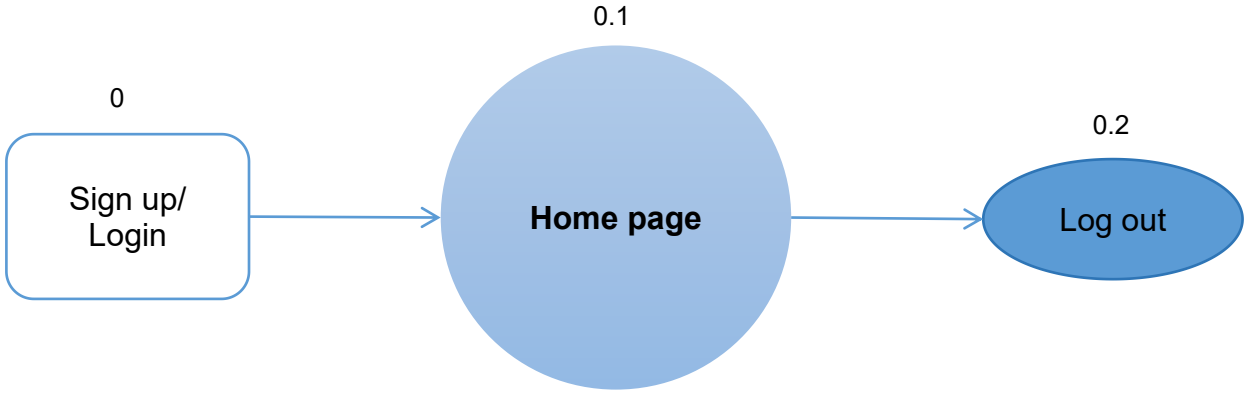


Figure 2.2 (DFD Level-1 of Metagram)

◆ **Level-2 DFD：**

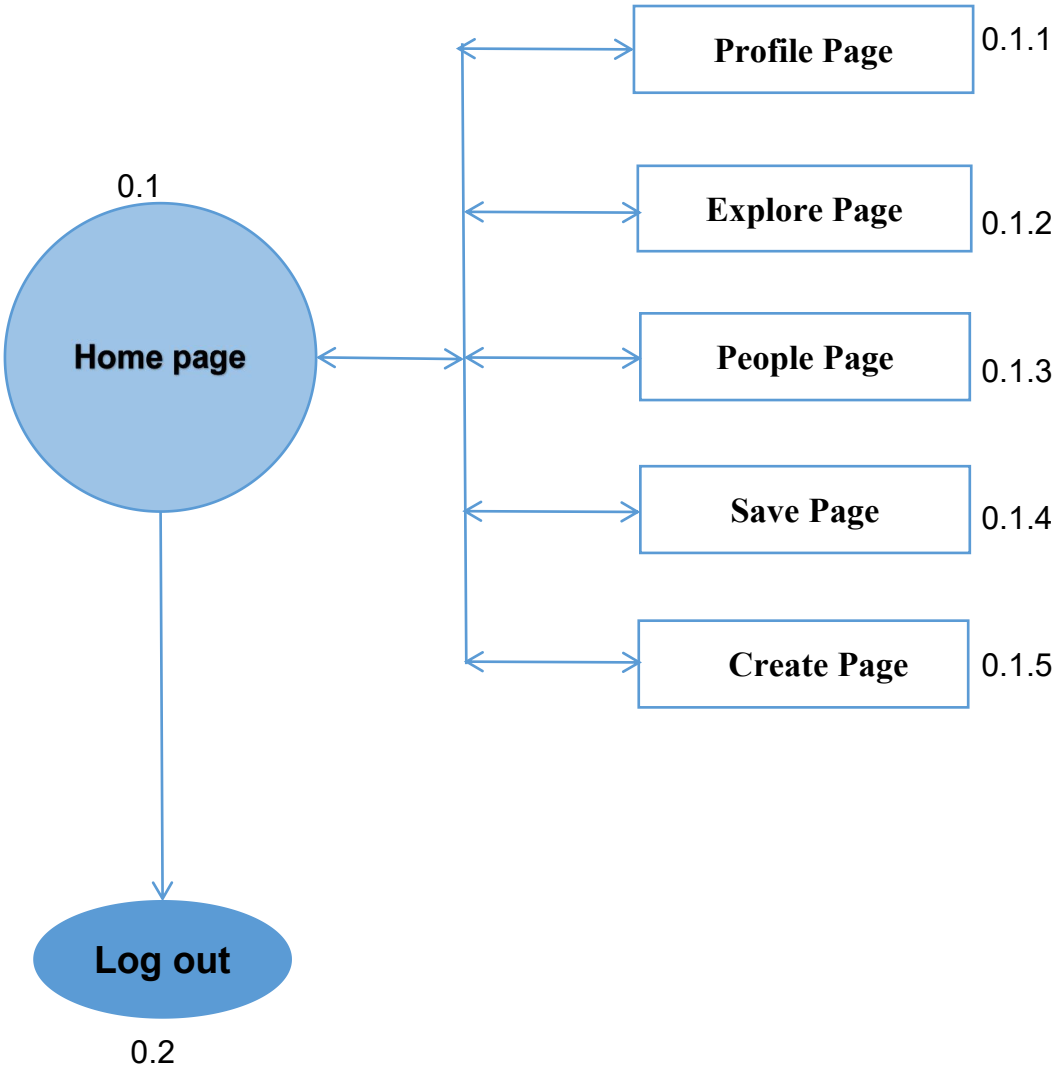


Figure 2.3 (DFD Level-2 of Metagram)

❖ Activity Diagram:

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

- ***Symbols*:**

I. Starting point:

A small filled circle followed by an arrow represents the initial action state or the start point for any activity diagram.

Start point/ Initial state

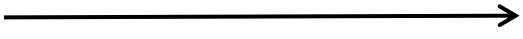
II. Activity or Action State:

An action state represents the non-interruption action of objects.

Activity

III. Action Flow:

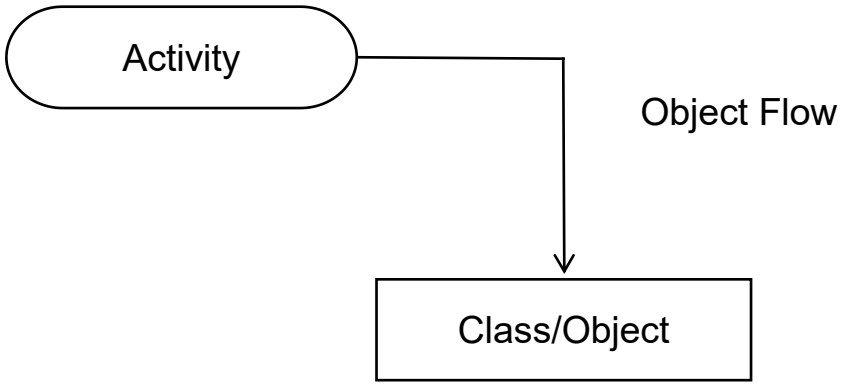
Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrow line.



Action Flow

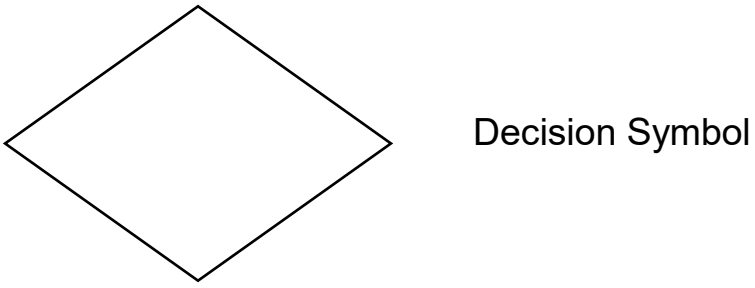
IV. Object Flow:

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



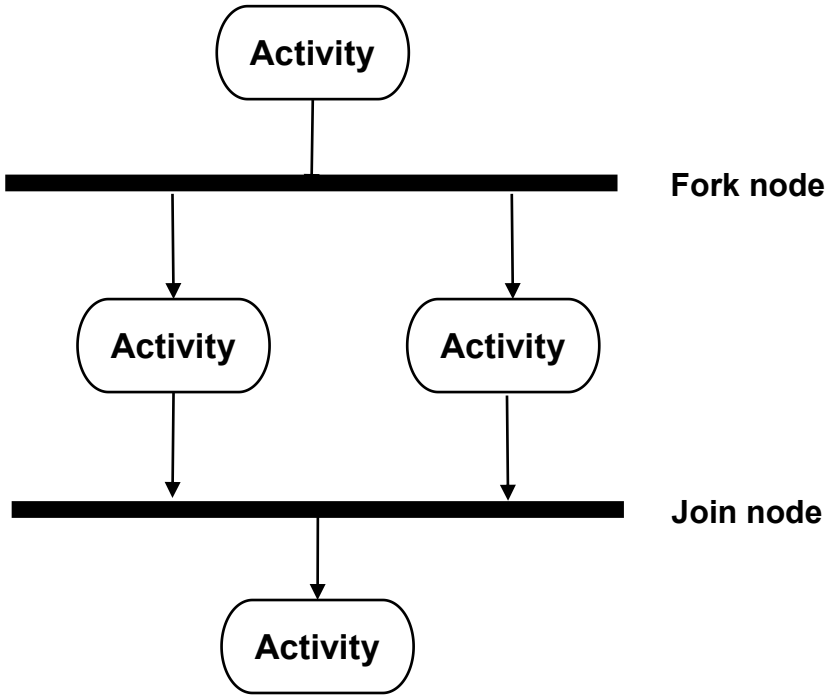
V. Decision and Branching:

A diamond represents a decision with alternate paths. When an activity requires a decision prior to moving on to the next activity, add a diamond between the two activities. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



VI. Synchronization:

A fork node is used to split a single incoming flow into multiple concurrent flows. It is represented as a straight, slightly thicker line in an activity diagram. A join node joins multiple concurrent flows back into a single outgoing flow. A fork and join mode used together are often referred to as synchronization.



Activity Diagram of METAGRAM:

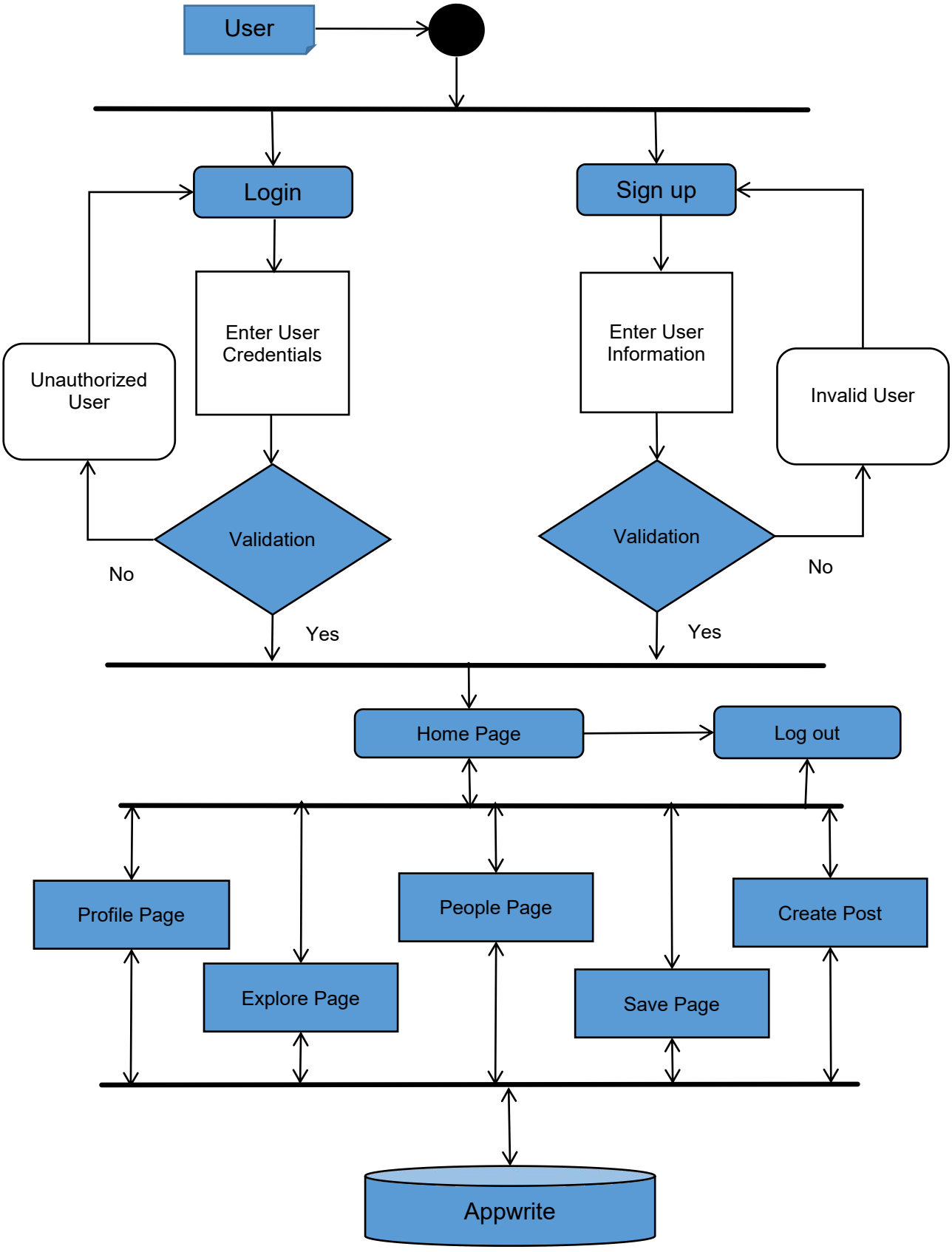


Figure 3.0 (Activity Diagram of Metagram)

❖ Use Case Diagram:

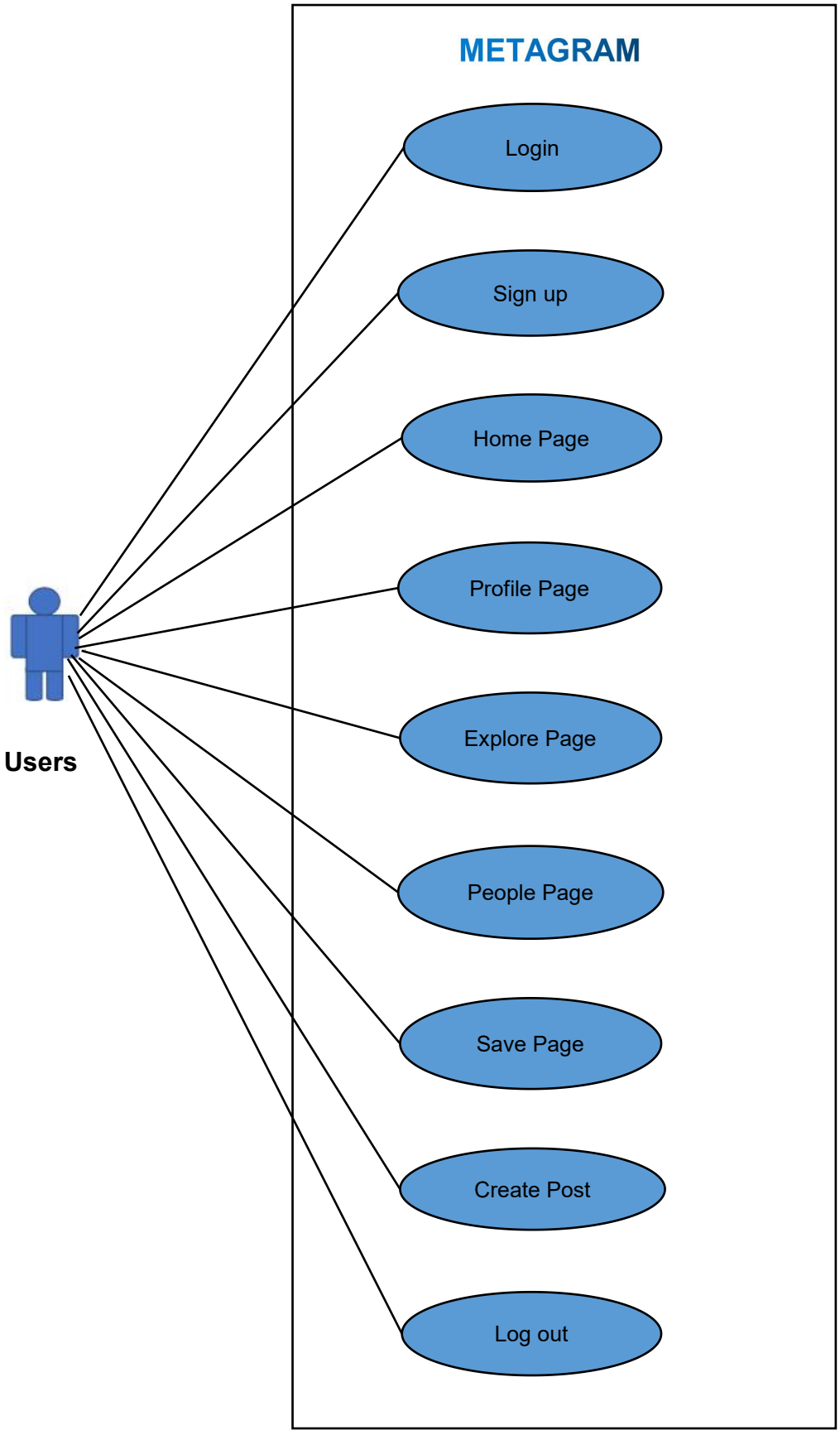


Figure 4.0 (Use Case Diagram of Metagram)

❖ ER Diagram:

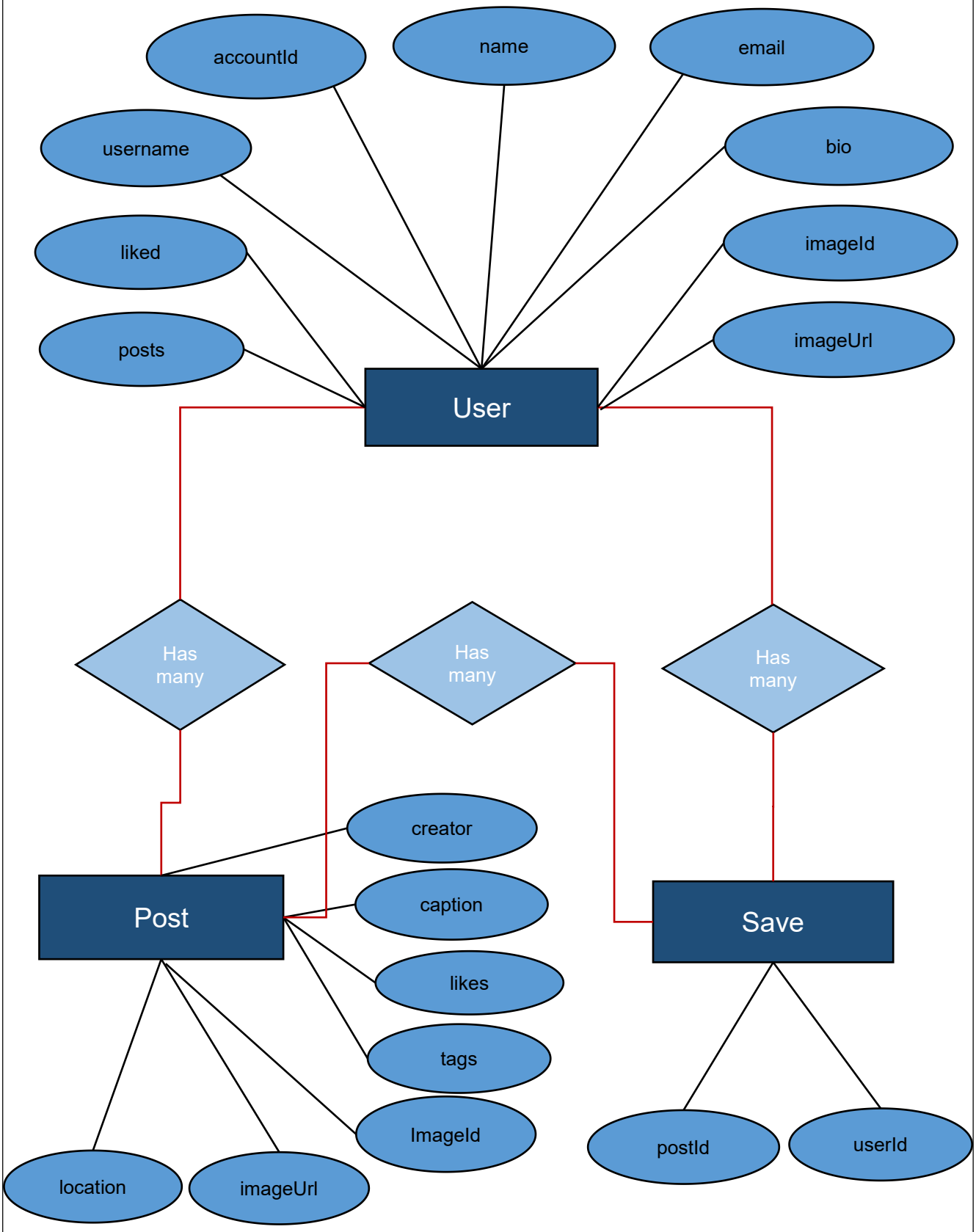


Figure 5.0 (ER Diagram of Metagram)

System Design

❖ Modularization Detail

Structured design partitions the program into small and independent modules. These are organized in top-down manner with the details shown in bottom.

Thus, structured design uses an approach called Modularization or decomposition to minimize the complexity and to manage the problem by subdividing it into smaller segments.

- **Login:**
  Login is the process by which an individual gain access to the system by identifying and authenticating themselves. Every user's has their own credentials to access their account. And once the user is logged in to the system then their user id is retrieved form the database and stored in a session. Once the user log out, their session is destroyed.
  Password and user-id(email) taken by Login page and it's used for security, after encrypting then the password is stored in Appwrite's database connection service. Database validates the given username and password and if its correct then it redirects the user to home page and if the credential does not matches then if gives an error message i.e wrong user-id(email) or password .

- **Sign up:**
  This module is use for registering new users for this system and register themselves as an authenticate user. This module also checks if the user is already existing, if the user is already there, then it will not accept re registration of the users. While adding new user the system will check that the particular username is not taken again, if so then give "**already exist"** message to the user. Once the user signs up with all valid information it will directly redirect to Home Page.

- **Home page:**

  Once sign in or sign up the user will be redirected to the home page. The home page again consist public posts or feeds with infinite scrolling feature for user to scroll endlessly without getting bored, and Five other modules or component that are profile page, explore page, people page, save page, create post page and another key feature is Top creator on the right side of home page and hidden in small screen these are the components which an authorized user can see and use.

- **Profile Page:**

  The page provides a basic social media profile with limited information about User. Such as profile picture which user can upload and change as per their needs, name, username, bio, statistics of post, followers and following. There are two tabs ‘posts’ and ‘liked post’ , these tabs allow user to see there posts and the posts they have liked. And an edit button where user can edit or change profile picture, name and bio.

- **Explore Page:**

  The page provides a basic social media like interface to search any content or posts. Here in this page user can search or explore existing posts, as well as like and save posts. User can also see post details with just one click and more related post.

- **People Page:**

  The page provides a basic social media website like interface to search new community or existing peoples in the Metagram. Here in this page Appwrite(back-end) allows user to search others users based on their name or username seamlessly and one user can follow others just by tapping follow button on others profile.

- **Save Page:**

  The page provides a basic social media website like interface where posts are saved by users. This page gives an easy access to users to look their saved posts and also to see post details seamlessly with the help of Appwrite's database.

- **Create Post Page:**

  The page provides a basic social media website like interface where user create posts and upload them to feeds.Text "Create Post" which signifies the purpose of the page. A text box labeled "caption" where you can write about post. Below the caption box, there's an option to "Add Photos" where user can upload images to your post. Drag and drop functionality is available or you can upload from your device.There's a section to "Add Location" to your post. A section to "Add Tags" to your post with text suggestions are provided like "Live, Love, Laugh". "Cancel" button allows user to discard the post. "Create Post" button allows user to publish the post to the feeds.

  Overall, this web page allows user to create a new social media post with text, images, location and tags.

Along with all this components or modules we also have **log out and Top creator** modules. Where logout function simply ends the session for the users and top creator is a display bar where top creator are listed on the daily basis with the limit of 15 creator in a list.

❖ User Interface and Design

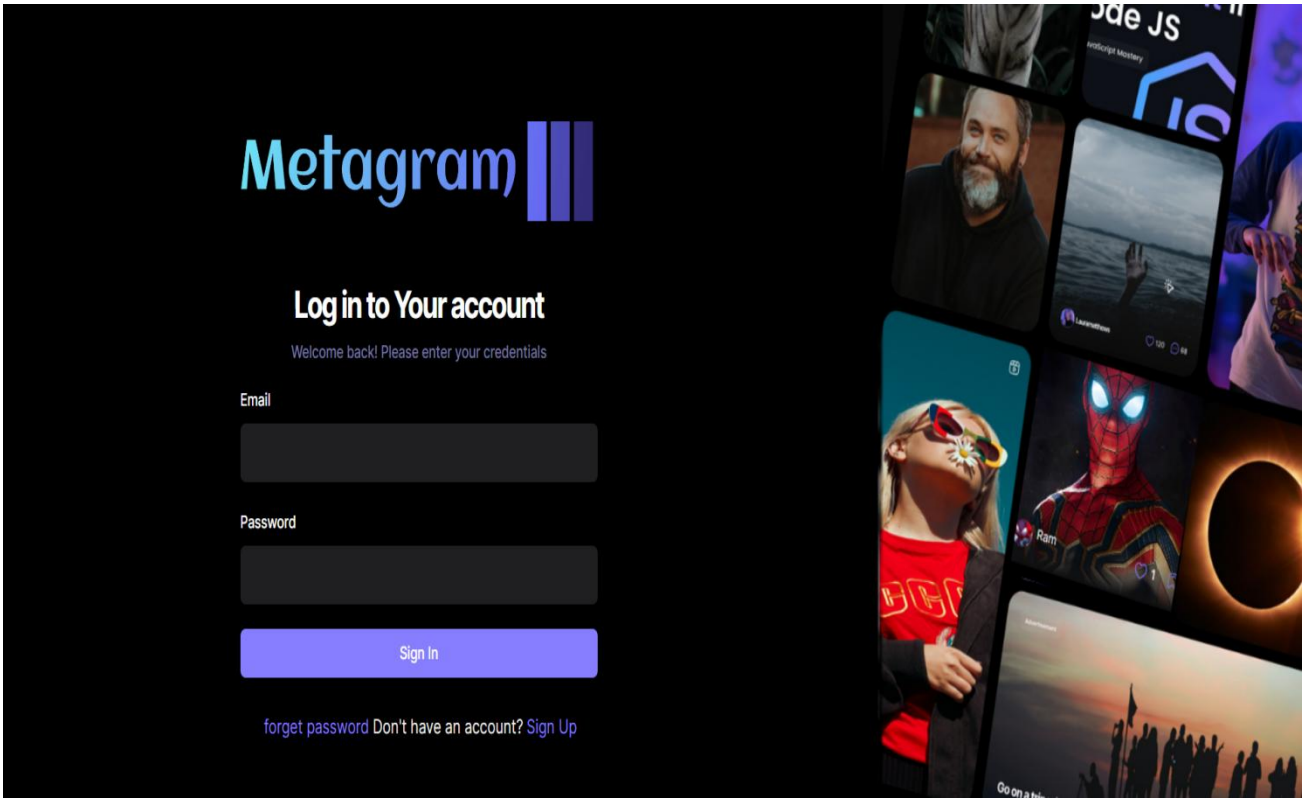


Figure 6.0 (Image of Log in /Sign in page)

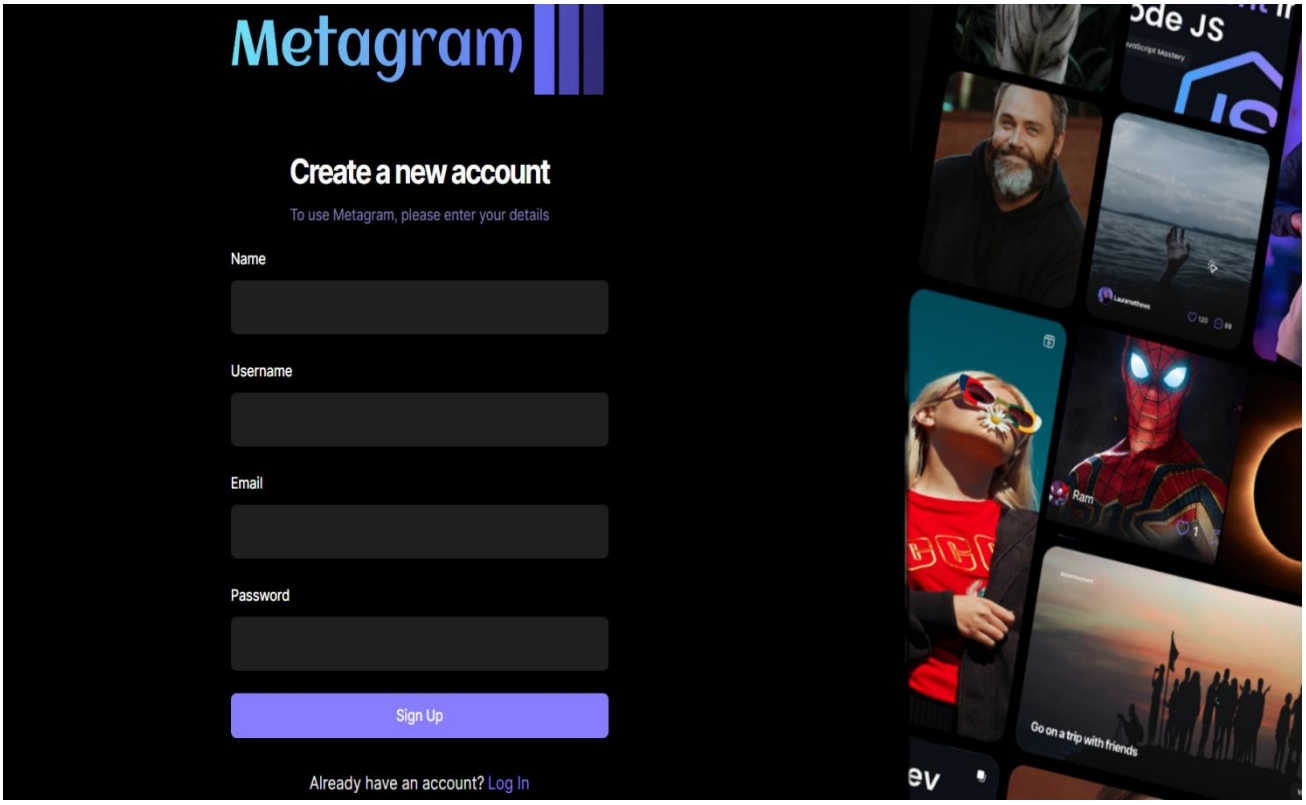


Figure 6.1 (Image of Sign up page)

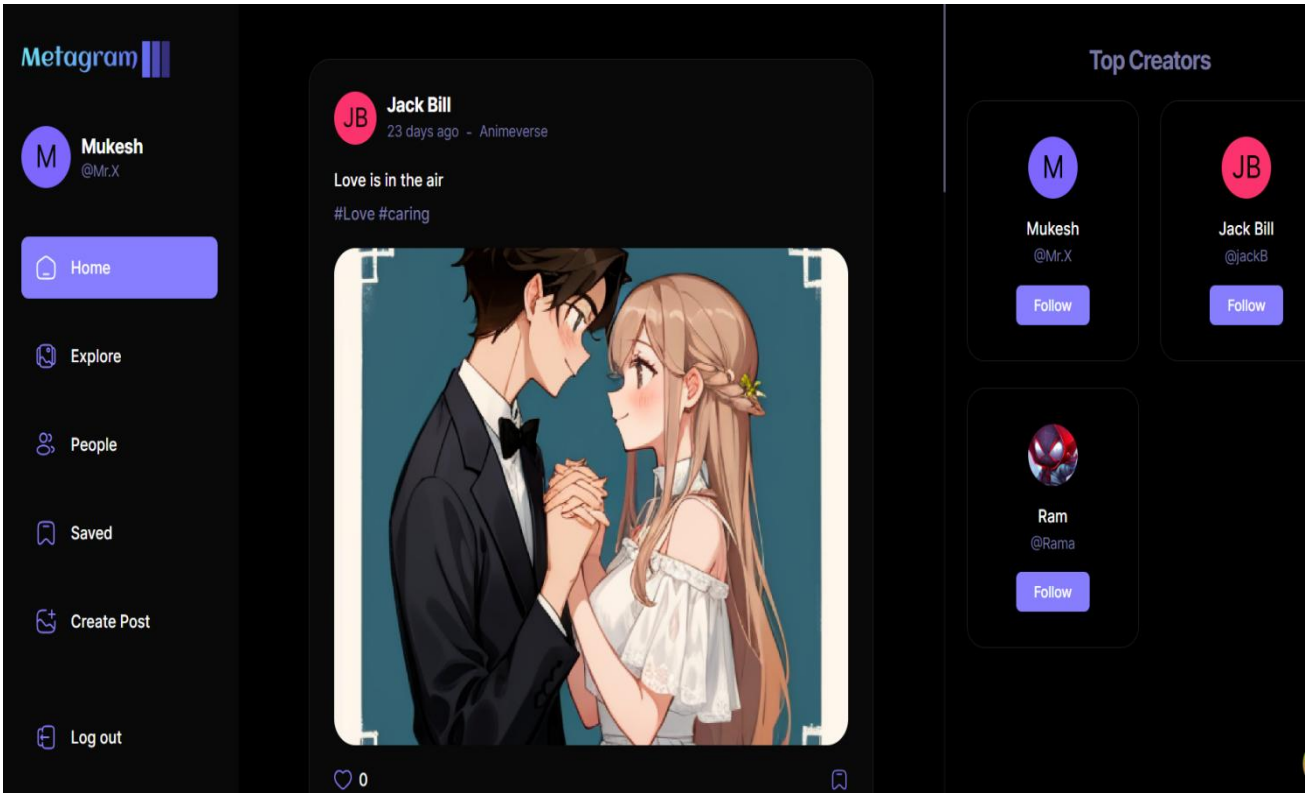


Figure 6.2 (Image of Home page)

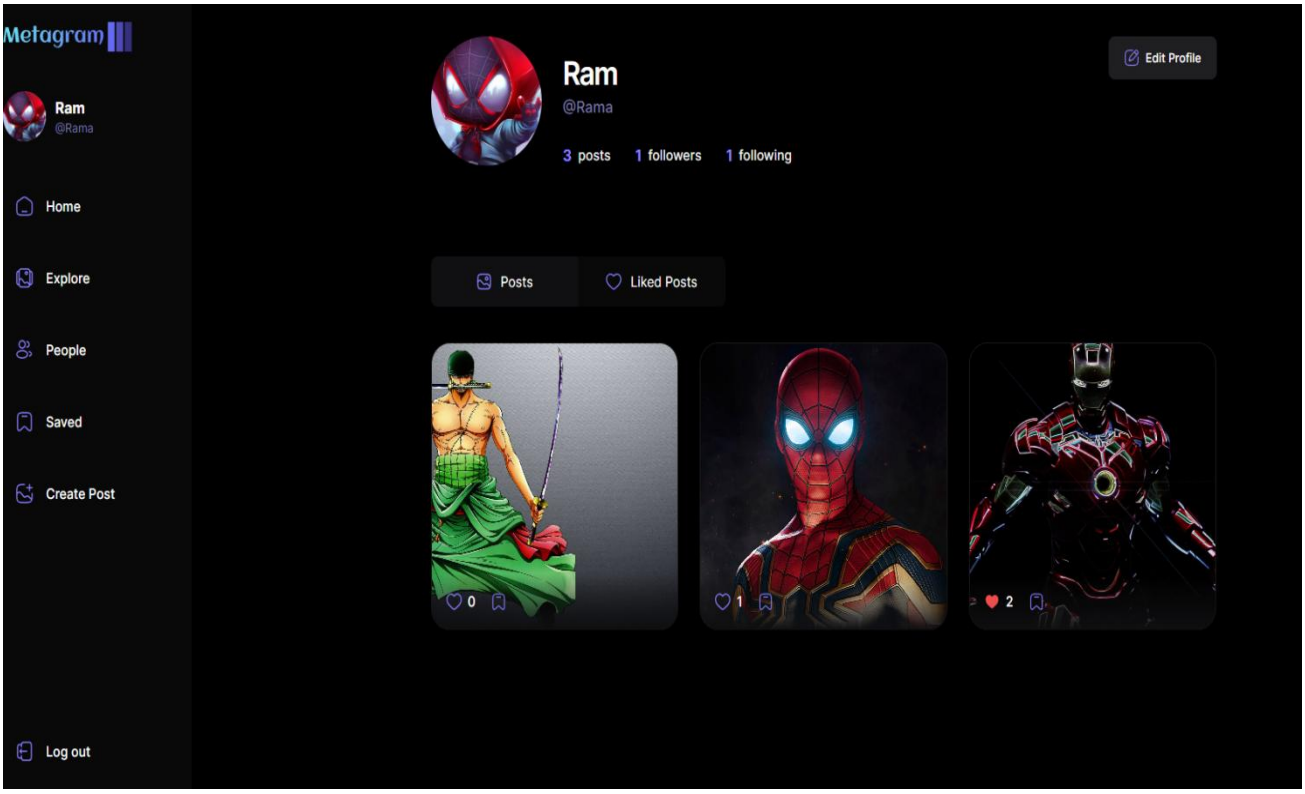


Figure 6.3 (Image of Profile Page)

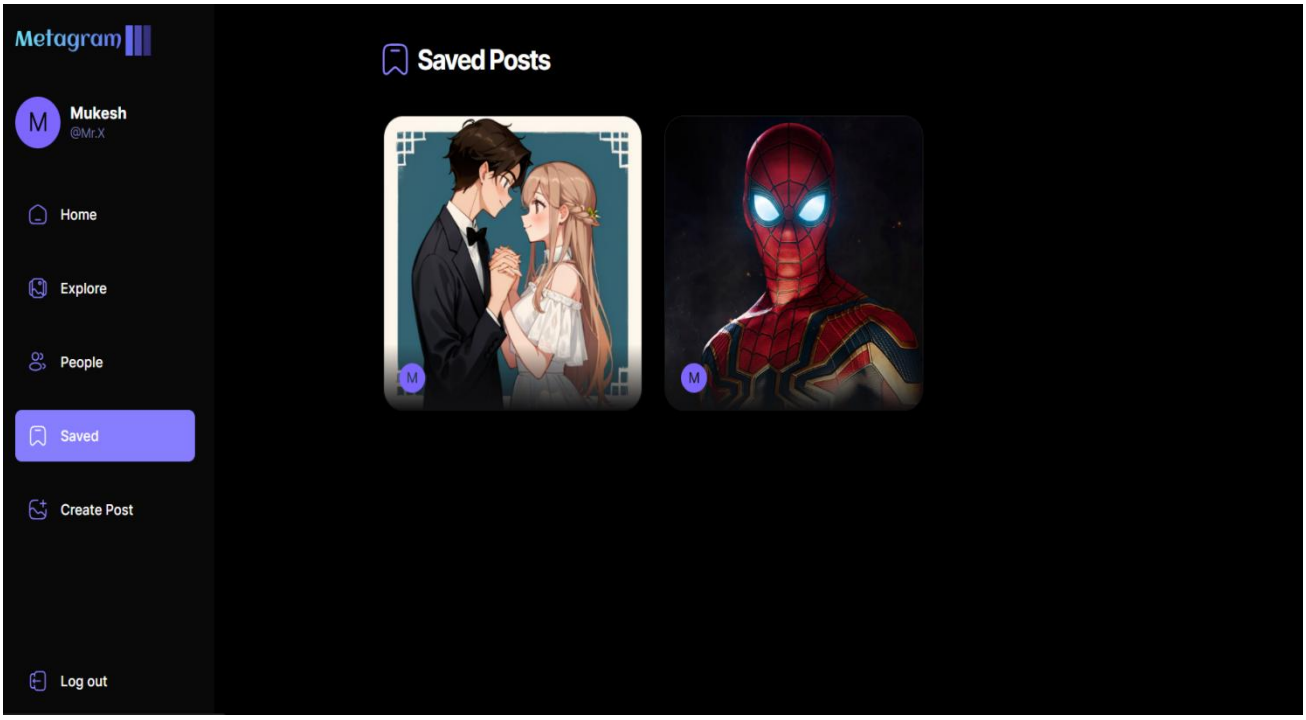


Figure 6.4 (Image of Save post Page)

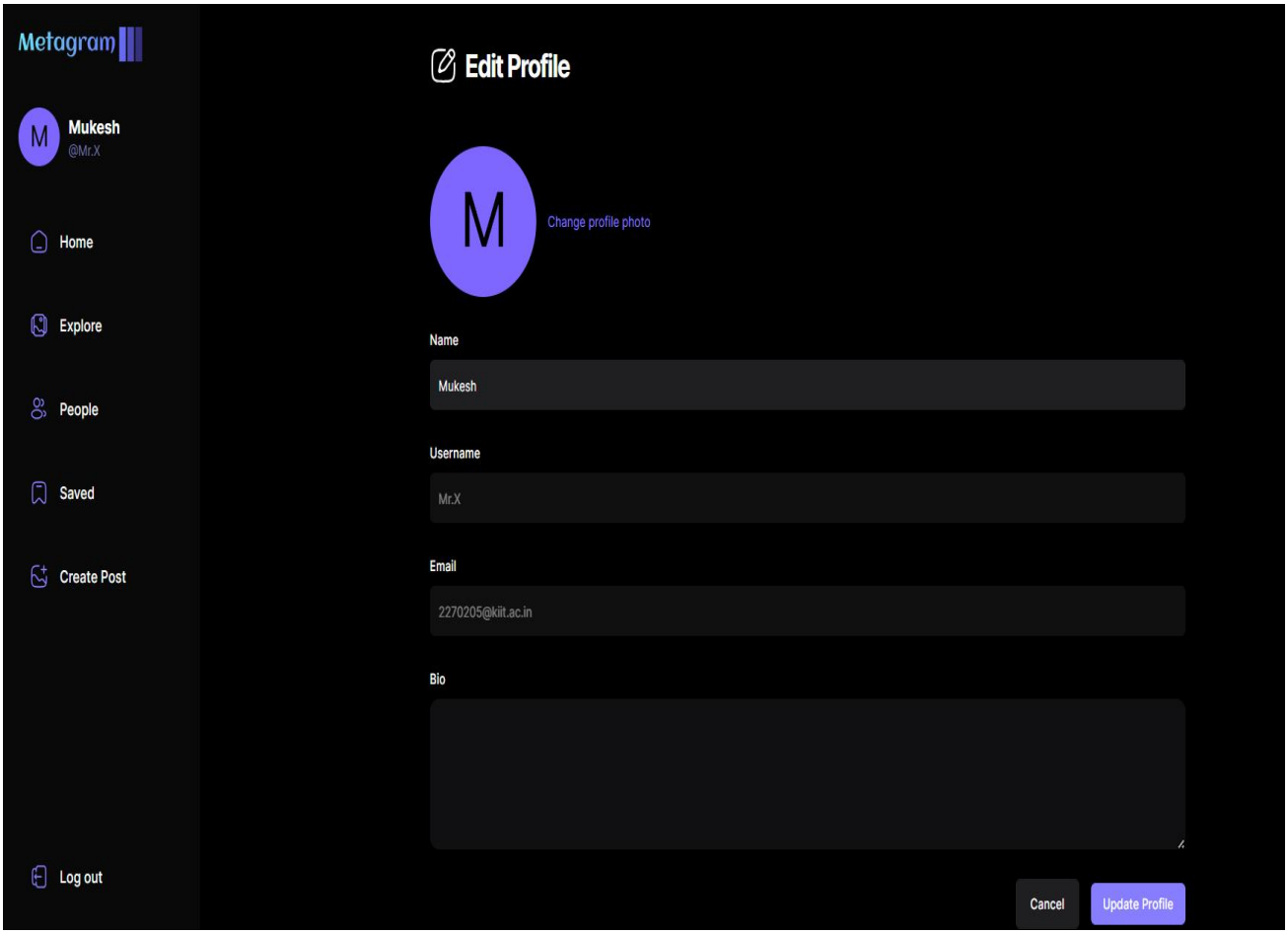


Figure 6.5 (Image of Edit Profile page)

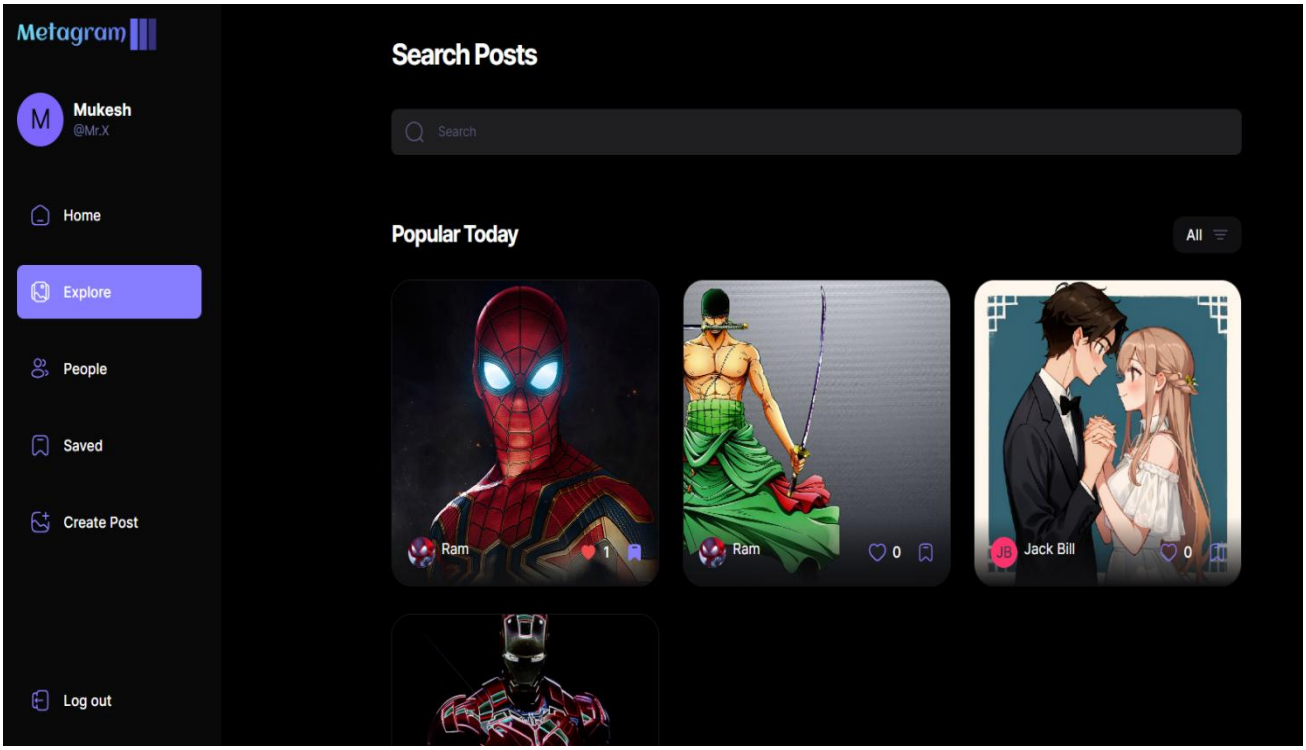


Figure 6.6 (Image of Explore page)

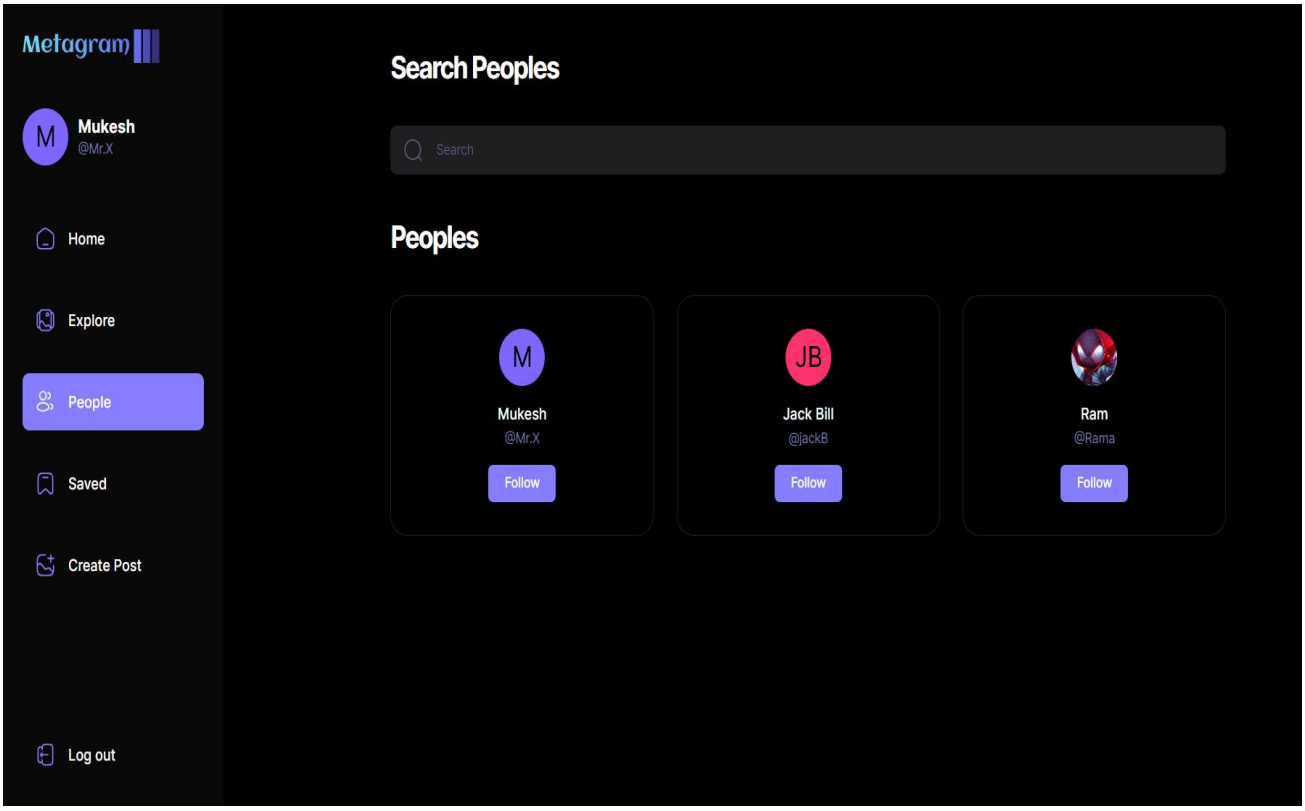


Figure 6.7 (Image of Search People Page)

- **Create Post Page:**

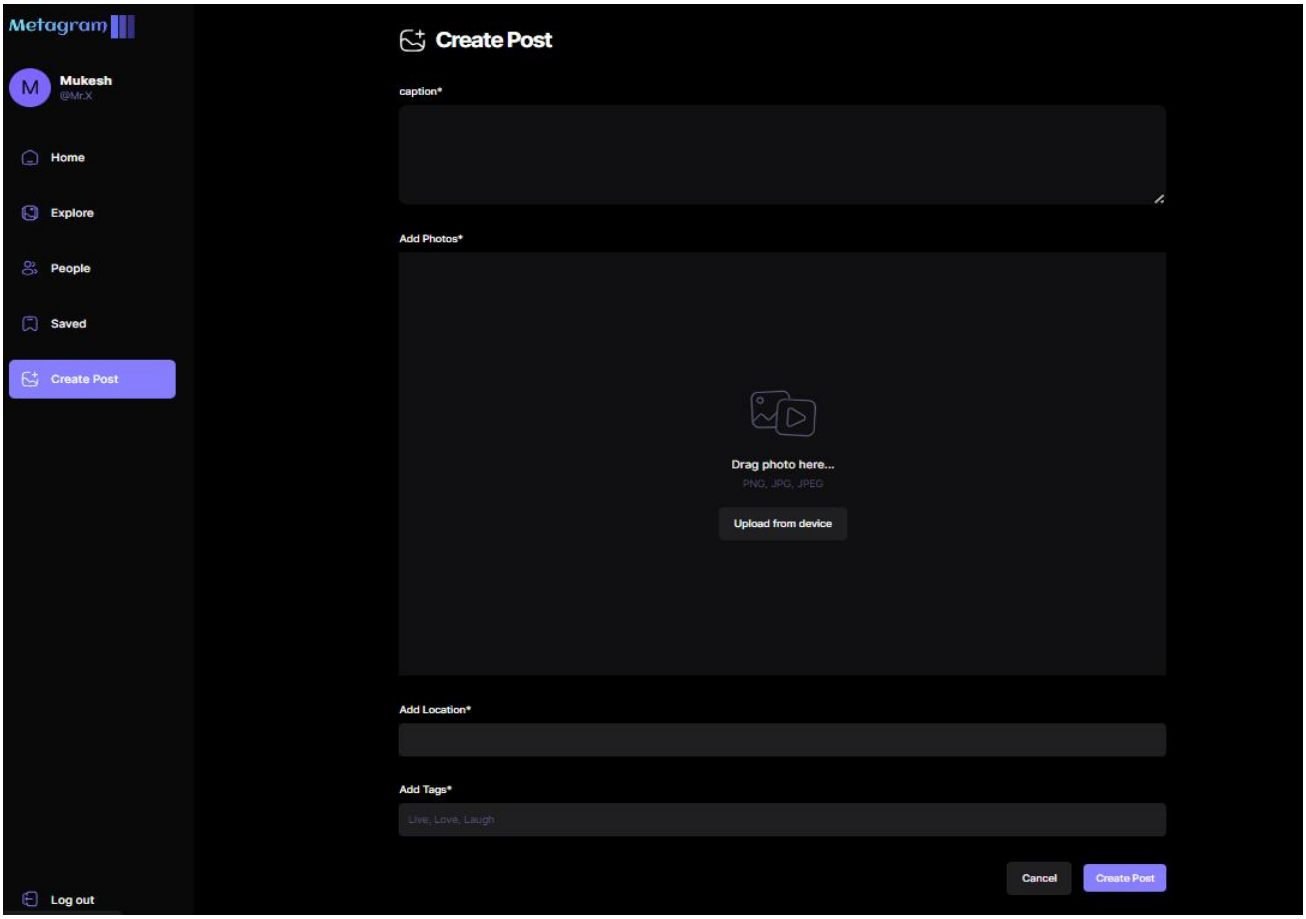


Figure 6.8 (Image of Creating post page)

Coding

❖ Standardization of the coding /Code Efficiency:

While coding standardization that are followed are:

I. Placing comments in the code:

Well description of packages and functions are describe while coding in comments “//” and “{/* */}”.

src > App.tsx > ...

```tsx
// Import necessary components for routing and styling
import { Routes, Route } from 'react-router-dom';
import './globals.css'; // Import global styles

// Import components for various pages
import {
  AllUsers,
  CreatePost,
  EditPost,
  Explore,
  Home,
  PostDetails,
  Profile,
  Saved,
  UpdateProfile,
} from './_root/Pages';

// Import components for authentication
import SigninForm from './_auth/forms/SigninForm';
import SignupForm from './_auth/forms/SignupForm';
import AuthLayout from './_auth/AuthLayout';
import RootLAyout from './_root/RootLAyout';

// Import Toaster component for notifications
import { Toaster } from "@/components/ui/toaster";
import ForgetPassword from './_root/Pages/ForgetPassword';

// Main App component that handles routing
const App = () => {
  return (
    <main className="flex h-screen">
      <Routes>
        {/* Public Routes (accessible without authentication) */}
        <Route element={<AuthLayout />}>
          {/* Route for sign-in form */}
          <Route path="/sign-in" element={<SigninForm />} />
```

Figure 7.0 (Screenshot of codes with proper comments)

II. **Page or Components Creation in the code:**

All components or pages are placed in separate folder for better clean coding readability and management.

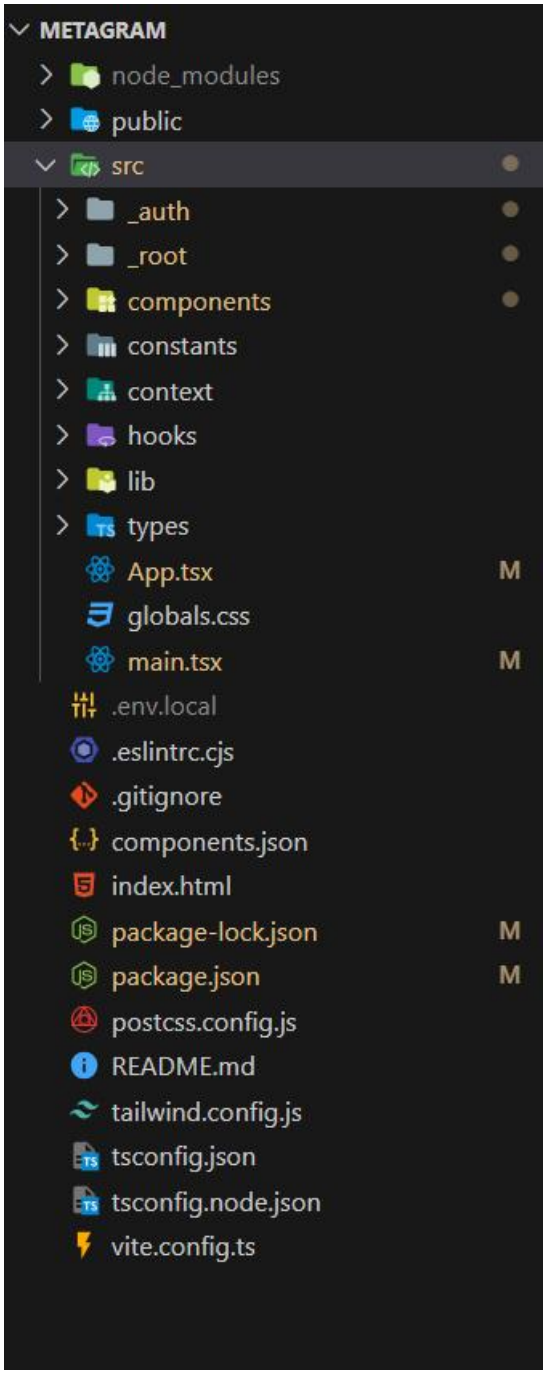


Figure 7.1 (Screenshot of File or page creation in a proper format)

III. **Common naming conventions across the code:**

Same coding conventions is followed to name the variables, functions etc. Across the code. This makes the code uniform. Also, meaningful names to variables are maintained which enhances the readability of the code. Common naming conventions are camel case and underscore. In camel case, the first letter of each word is capitalized except the first word, while in underscore naming convention you place an underscore between each word.

Figure 7.2(Screenshot of common naming convention used for variables)

Figure 7.3 (Screenshot of common naming convention used for method)

.

IV. **Avoid too much deep nesting:**

Too much deep nesting makes the code difficult to understand so it must be avoided. While developing News App, too much of nesting was avoided for code maintainability kept in mind.

❖ Error handling:

Error handling refers to the response and recovery procedures from error conditions present in a software application. In other words, it is the process comprised of anticipation, detection and resolution of application errors, programming errors or communication errors. Error handling helps in maintaining the normal flow of program execution.

In this Metagram website when searched something it will not get any result and get the error code from Appwrite(database) then it is handled by the code to show result "No Result Found" message instead for showing nothing which make bad user experience.

```
// SearchResults component that displays search results or informs about no results found
const SearchResults = ({ isSearchFetching, searchedPosts }: SearchResultProps) => {
  // Check if search is currently fetching data
  if (isSearchFetching) {
    // Display a loader component while fetching data
    return <Loader />;
  } else if (searchedPosts && searchedPosts.documents.length > 0) {
    // If search results exist and have documents, display them
    return <GridPostList posts={searchedPosts.documents} />;
  } else {
    // If no search results found, display a message
    return (
      <p className="text-light-4 mt-10 text-center w-full">No results found</p>
    );
  }
};
```

Figure 7.4 (Screenshot of Error Handling in code)

❖ Parameters/Props passing:

When we pass value to function by giving the value in function call statement as a parameter/Props.

In this React Application all the API Field are passed using Props and Props are received by component which make this application easy to modify without going thorough all the Codes.

Here in this code post and action are the Props to the PostForm function.

```
const PostForm = ({ post, action }:PostFormProps) => {

    const { mutateAsync: createPost, isPending: isLoadingCreate } =
    useCreatePost();
    const { mutateAsync: updatePost, isPending: isLoadingUpdate } =
    useUpdatePost();
    const { user } = useUserContext();
    const { toast } = useToast();
    const navigate = useNavigate();

     // 1. Define your form.
  const form = useForm<z.infer<typeof postValidation>>({
    resolver: zodResolver(postValidation),
    defaultValues: {
      caption: post ? post.caption: "",
      file:[],
      location: post ? post.location:"",
      tags: post ? post.tags.join(',') : "",
    },
  })
```

Figure 7.5 (Sreenshot of code using props/parameters)

❖ Validation checks:

Validation and verification are two ways to check that the data entered into a computer is correct.

● Format check:

Format check is a validation check, which checks the data is in the right format or not. While this React Application was developed every text field has sets of validation where the format check was also taken care of.

```
<form onSubmit={form.handleSubmit(onSubmit)}
className="flex flex-col gap-5 w-full mt-4">
<FormField
    control={form.control}
    name="email"
    render={({ field }) => (
    <FormItem>
      <FormLabel>Email</FormLabel>
      <FormControl>
        <Input type="text" className= "shad-input" {...field} />
      </FormControl>
      <FormMessage />
    </FormItem>
  )}
/>
<FormField
    control={form.control}
    name="password"
    render={({ field }) => (
    <FormItem>
      <FormLabel>Password</FormLabel>
      <FormControl>
        <Input type="password" className= "shad-input" {...field} />
      </FormControl>
      <FormMessage />
    </FormItem>
  )}
/>
```

Figure 7.6 (Screenshot of valid form format in code)

Testing

Web and Mobile (PWA) applications testing is a procedure to test these applications usability, functional and consistency glitches.

❖ Functional Testing Test Cases:

The functional testing of Metagram normally consists in the areas of testing user interactions as well as testing the database. The various factors which are relevant in functional testing done in Metagram are:

I. The required mandatory fields are working as required.
II. The mandatory fields are displayed in the screen in a distinctive way than the non-mandatory fields.
III. Application is works as per as requirement whenever the application starts/stops.
IV. The device is able to perform required multitasking requirements whenever it is necessary to do so.
V. The navigation between relevant components in the application are as per the requirement.
VI. The installed application enables other applications to perform satisfactorily, and it does not eat into the memory of the other applications.
VII. The installation of the application is done without any downloading because website is using Device default Web Browser to run, Which make the app light weight and responsive.
VIII. The application performs according to the requirement in all versions of Mobile that is 3G, 4G and 5G.
IX. The session only ends when user log out from Metagram.
X. Infinite scroll feature is tested fine.
XI. The system is properly connected with Appwrite database and it stores the details of users in well manner.

XII. Verified and tested the system accepts valid inputs with relevant errors messages.

❖ Performance Testing Test Case:

The performance testing of Metagram system normally consists in the areas of testing the performance of the existing system. This type of testing's fundamental objective is to ensure that the system performs acceptably under certain performance requirements.

The various factors which are relevant in this performance testing done in Metagram are:

I. The system performs as per the requirement under different load conditions is working well.

II. Performance testing tools and environments are set up for load generation.

III. Record the baseline response time of key system functionalities (e.g., user login, routing, image loading) under normal load conditions.

IV. The response time of the system is as per as the requirements.

V. Gradually increase the load to reach the desired stress levels on the system.

VI. Continuously monitor the response time of critical system endpoints (e.g., login page, image loading, posting content) during the load test.

VII. As the Application is build using React Framework so it can handle tons of traffic because once the code is send to client device through server and then all the processing is done in client side.

VIII. Measure the average response time, maximum response time, and latency for each endpoint.

IX. Monitor the system's behaviour and response time as the stress levels increase.

X. Evaluate the system's ability to scale horizontally or vertically to handle increased load.

XI. The minimum requirement of this application to run in optimal performance with 2GB of RAM and 1.2GHz processor can run the website smoothly on any device.

XII. Add or remove resources (e.g., CPU, memory) dynamically and observe the impact on system performance.

XIII. The system should demonstrate scalability by efficiently handling increased load and adapting to changes in resource availability.

XIV. The system should maintain acceptable response times and performance metrics under normal load conditions.

XV. Performance test results, including response time metrics and scalability analysis, are documented and analyzed.

XVI. Any performance bottlenecks or areas for optimization are identified for further investigation and improvement.

❖ USABILITY TESTING TEST CASE:

The usability testing of Metagram system normally consists in the areas of testing the basic and advanced usability of the existing system. The main objective is to ensure that we end up having an easy-to-use, intuitive and similar to industry-accepted interfaces which are widely used.

The various factors which are relevant in this usability testing done in Metagram are:

I. The buttons and the information box have the required size to look decent.

II. Test environment is set up with representative users and testing devices.

III. The buttons are placed in the same section of the screen and spread equally geometrical to avoid confusion to the end users.

IV. The icons are natural and consistent with the system.

V. Task users with common actions, such as logging in, registering, and accessing attendance records.

VI. Observe users as they navigate through the system and complete assigned tasks.

VII. Evaluate users ability to understand the purpose and functionality of each UI element.

VIII. Solicit feedback on the clarity of labels, icons, and visual cues used in the UI.

IX. Introduce deliberate errors or invalid inputs during task execution (e.g., incorrect login credentials).

X. Assess users' ability to identify and recover from errors.

XI. Measure the effectiveness of error messages and prompts in guiding users to resolve issues.

XII. Evaluate the consistency of UI elements and interactions across different screens and modules.

XIII. Ensure that common UI patterns, such as navigation and action buttons, are consistent throughout the website.

XIV. Identify any inconsistencies or deviations from established UI design guidelines.

XV. The navigation are not overloaded because it has to be interactive and user friendly.

XVI. The text is kept simple and clear to be visible to the users.

XVII. The font size is big enough to be readable and not too big or too small.

XVIII. The system items are always synchronized according to the user actions.

XIX. All strings are converted into appropriate languages whenever a language translation facility is available.

❖ Security Testing Test Cases:

The fundamental objective of security testing is to ensure that the application's data and networking security requirements are met as per guidelines.

The following are the most crucial areas of checking that's has been done are:

I. The website is not permitting an attacker to access sensitive content or functionality without proper authentication.

II. The website has a strong password protection system and it does not permit an attacker to obtain, change or recover another user's password all because of Appwrite's security.

III. It is maintained with dynamic dependencies and take measures to prevent any attacker for accessing these vulnerabilities.

IV. To protect the application and the network from the denial of service attacks.

V. Data storage and data validation requirements are maintained.

VI. Metagram is well protecting against malicious client side injections.

VII. It is well protecting against malicious runtime injections.

VIII. Regular audits will be done for data protection analysis.

IX. Different data streams is analyzed and preventing any vulnerabilities from these.

❖ Compatibility Testing Test Cases:

Compatibility testing on all devices is performed to ensure that since devices have different size, resolution, screen, version and hardware so the application should be tested across all the devices to ensure that the application works as desired.

The following are the most prominent areas for compatibility testing.

I. The user Interface of the application is as per the screen size of the device, no text/control is partially invisible or inaccessible.

II. The text is readable for all users for the application.

III. The website is running in all sorts of devices.

IV. The website is minimized or suspended on the event of a call and then whenever the call stops the application is resumed.

System Security measures

These Measures apply to anyone who have accesses, uses, or controls this website. Appwrite being a Backend-as-a-Service (BaaS) platform, would likely implement a variety of system security measures to protect user data and applications built on their platform. Here's what we can explore based on publicly available information (since access to their specific implementation details might be limited).

❖ **Database/data security:**

Database security testing is done to find if there is any loop holes in security mechanisms and about finding the vulnerabilities or weaknesses of database system.

The main target of database security testing is to find out vulnerabilities in a system and to determine whether its data and resources are protected from potential intruders. Security testing defines a way to identify potential vulnerabilities effectively, when performed regularly.

- **Unauthorized Access to data:**

User Authentication: Appwrite most likely uses a secure method for user authentication, such as passwords with strong hashing algorithms or multi-factor authentication (MFA) for added security. User Roles and Permissions: They might offer a system for defining user roles and assigning granular permissions to control access to specific functionalities and data within the platform.

- **Data Security:**

Data Encryption: Appwrite likely encrypts data at rest (stored data) and in transit (data being transferred) to protect it from unauthorized access. Secure Storage: They might utilize secure storage solutions to store user data, potentially leveraging cloud providers' robust data security infrastructure.

- **Database Security:**

The specific database management system used by Appwrite (e.g., PostgreSQL, MongoDB) likely has its own built-in security features. These might include user authentication, authorization, and activity auditing.

- **Network Security:**

Firewalls: Appwrite likely employs firewalls to filter incoming and outgoing traffic, blocking unauthorized access attempts. Intrusion Detection/Prevention Systems (IDS/IPS): They might have systems in place to monitor network activity for suspicious behavior and prevent potential intrusions.

- **Security Software:**

They might utilize anti-virus, anti-malware, and vulnerability scanning tools to proactively identify and address potential security risks within their infrastructure.

- **Security Policies:**

Appwrite likely has documented security policies outlining their security practices, user responsibilities, and incident response procedures.

- **Data Encryption:**

At Rest: Appwrite likely encrypts data stored in their databases using strong encryption algorithms like AES-256. This ensures that even if an attacker gains access to the database, the data would be unreadable without the decryption key.

In Transit: Data transferred between users and Appwrite's servers, such as during uploads or downloads, should also be encrypted using protocols like TLS/SSL. This protects data from eavesdropping or tampering during transmission.

- **Additional Considerations:**

Compliance: Appwrite might adhere to relevant data privacy regulations like GDPR or CCPA, depending on their target audience and data storage location.

Penetration Testing: Regular penetration testing by security experts can help identify and address vulnerabilities in their platform.

Transparency: Ideally, Appwrite should provide clear documentation about their security practices to build trust with developers using their platform.

Future Scope

Let's explore the exciting future scopes for this basic social media website built using React, Typescript, Tailwind CSS for styling and Appwrite as the backend-as-a-service (BaaS) platform. With the foundational functionalities , here are some potential areas for expansion and enhancement:

1. Enhanced User Experience:

- **Richer User Profiles**: Allow users to add more details to their profiles, such as interests, location, and a cover photo.
- **Notifications**: Implement real-time notifications for likes, comments, and new followers.

2. Content Features:

- **Comments and Replies**: Enable users to comment on posts and reply to comments.
- **Mentions**: Allow users to mention others in their posts or comments.
- **Polls and Surveys**: Add interactive elements like polls or surveys to engage users.
- **Reels and Story:** Add interactive UI for sharing story and reels with all relevant functionalities.

3. Advanced Post Management:

- **Scheduled Posts**: Let users schedule posts for future publishing.
- **Drafts**: Allow users to save drafts of their posts.
- **Editing History**: Keep track of post edits and display revision history.

4. **Privacy and Security**:

- **Private Accounts**: Allow users to set their profiles as private, requiring approval for followers.
- **Two-Factor Authentication (2FA)**: Enhance security by implementing 2FA during login.
- **Content Moderation**: Integrate tools to detect and handle inappropriate content.
- **Sign up or Login Methods:** Allowing user to sign up or log in effortlessly for easy registration or login with Third-party-authentication methods like google.

5. **Analytics and Insights**:

- **User Analytic's**: Provide users with insights into their post reach, engagement, and follower growth.
- **Popular Hashtags and Trends**: Display trending hashtags and topics.

6. **Monetization**:

- **Ads**: Explore options for targeted ads based on user interests.
- **Premium Features**: Offer premium subscriptions with additional features (e.g., ad-free experience, analytics).

7. **Community Building**:

- **Groups or Communities**: Allow users to create or join interest-based groups.
- **Events**: Enable users to create and promote events within the community.

8. **Geo-location Features**:

- **Check-Ins**: Let users check in at specific locations.
- **Local Recommendations**: Suggest nearby places or events.

9. **Integration with Other Platforms**:

- **Cross-Platform Sharing**: Enable users to share their Metagram posts on other social media platforms.
- **APIs for Third-Party Apps**: Provide APIs for developers to build apps that integrate with Metagram.

10. **Internationalization (i18n)**:

- **Localization**: Support multiple languages to cater to a global audience.

Further development:

The Development of this website does not end here. There are many more feature are going to add in future. Like many above feature are going to be added in future development. Metagram will be more optimized which will make the website more responsive and smooth. Security updates as any new vulnerability introduce. The scalability, performance optimization, and maintaining a responsive user experience are critical as the user base grows. Regularly gather user feedback and iterate on features to keep Metagram relevant and engaging.

Assumption

An assumption is something that is believed to be true. It's an event that you can expect to happen during a project. However, that certainty isn't supported by factual proof, it comes from experience. Just like dependencies and constraints, assumptions are events that are outside of the project manager's and team's control. But unlike constraints, which put restrictions on a project and can pose a danger to its successful completion, assumptions open possibilities for it and make it possible for the project to finish successfully. This is how you can differentiate assumptions from constraints and dependencies.

- **Operating System:**

I. Windows 7, 8.1, 10, 11
II. Mac-OS 10, 11, 12
III. Minimum version assumed Android 5.0 and iOS 9.

- **Hardware configuration:**

I. Minimum hardware configuration assumed to be of 2 GB RAM, 1.2 GHz duel core processor.

- **Device screen size:**

I. Assumption of the device screen size is range from 5 inch to 32 inch.

- **Network Connectivity:**

Metagram relies on stable network connectivity to facilitate communication between client devices and the central server for real time usage.

- **User :**

Metagram assumes that user has the basic understanding of using mobile, laptop and Computers . And some knowledge about using social media websites.

These assumptions underpin the design, development, and deployment of Metagram, providing a framework for its functionality, usability, and effectiveness in addressing attendance management needs within organizations

Glossary

- **React :**

React is a popular JavaScript library for building user interfaces (UIs). It was developed by Facebook and is widely used for creating dynamic, interactive web applications.

Component-Based React follows a component-based architecture. You build your UI by composing reusable components that encapsulate their own logic and rendering. Virtual DOM React uses a virtual representation of the DOM (Document Object Model) to efficiently update only the necessary parts of the actual DOM when data changes. This improves performance and minimizes unnecessary re-renders. Declarative Syntax React encourages a declarative approach to UI development. You describe what you want the UI to look like, and React takes care of updating the actual DOM. JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code within your JavaScript files. It makes React components more readable and expressive.

React and Vite offer a powerful combination for building modern, performant, and scalable web applications. With its focus on developer experience and performance optimization, this duo can empower you to create exceptional user experiences.

- **React TSX:**

React and Typescript are two powerful technologies that can be used together to build robust and maintainable web applications. TypeScript is a Superset of JavaScript that adds optional static typing. Improves code maintainability and helps catch errors early in the development process. Provides features like interfaces, classes, and generics for better code organization and type safety. TypeScript helps define the types of variables, functions, and props used in your React components. This catches potential type errors during development, preventing runtime issues. Explicitly defined types make code

easier to understand for both you and other developers. Features like auto completion and type checking in IDEs (Integrated Development Environments) can significantly improve development speed and efficiency. React and Typescript both have large and active communities, with plenty of resources and libraries available. Using React with Typescript provides a powerful combination for building modern web applications. It leverages the flexibility of React for component-based development while adding the benefits of static typing from Typescript, leading to more maintainable, robust, and developer-friendly code.

- **Components and Props:**

Components split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components. We can find a detailed component API reference here. Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

- **Hooks:**

React Hooks are a relatively new addition to React (introduced in version 16.8) that allow you to "hook into" React features like state and life-cycle methods from functional components. Prior to Hooks, class components were the primary way to manage state and side effects in React applications. Hooks offer a more concise and functional alternative.
React Hooks are a powerful addition to the React ecosystem. They provide a clean and functional way to manage state, side effects, and other functionalities in React applications, leading to more maintainable and well-structured code.

- **React Router Dom:**

React Router DOM is a popular library for building single-page applications (SPAs) with React. It allows you to manage navigation and routing within your React application. React Router DOM defines routes as components. This allows for a more declarative and reusable approach to routing compared to traditional configuration-based routing. It maps URLs to specific React components, ensuring the appropriate component is rendered based on the current URL. You can use components like Link and NavLink to define navigation links within your React components. These components handle browser history updates and trigger component re-renders when clicked. React Router DOM supports nested routes, allowing you to create complex application structures with hierarchical relationships between component. You can use the useNavigate Hook (or older history object) to programmatically navigate to different routes within your application using JavaScript code.
React Router DOM is a valuable tool for building dynamic and user-friendly SPAs with React. It simplifies navigation management and provides a structured way to handle routing logic within your React application.

- **ShadCn:**

ShadCn appears to be a developer focused on creating open-source UI component libraries and potentially building front-end applications. The website provides a React-based UI component library called ShadCn UI. It offers a collection of beautifully designed, accessible, and customizable components that can be easily integrated into your React applications to streamline the development process. By using pre-built and customizable components, developers can significantly speed up the development process compared to building everything from scratch. ShadCn UI enforces a consistent design language throughout your application, resulting in a more polished and professional user experience. Using reusable components makes your code more maintainable and easier to update in the future.

ShadCn UI appears to be a promising UI component library for React developers. It offers a comprehensive set of accessible and customizable components that can significantly improve the development workflow and the quality of your React applications.

- **Appwrite:**

Appwrite (https://appwrite.io/) is a self-hosted, open-source Backend-as-a-Service (BaaS) platform. It provides developers with a set of tools and functionalities to build secure and scalable web, mobile, and backend applications without having to manage the underlying infrastructure. Provides a NoSQL database for storing and managing application data. Allows storing and managing various file types (images, videos, documents) with features like uploads, downloads, and access control. Enables developers to create serverless functions using various programming languages to extend Appwrite's functionalities and implement custom logic. Offers features for real-time communication between users and applications. Appwrite eliminates the need to build and manage backend infrastructure, allowing developers to focus on building application features. The platform can handle growing application needs without requiring significant infrastructure changes. Appwrite takes security seriously, offering features like data encryption, access control, and secure storage practices. Being open-source allows for customization and self-hosting, giving developers more control over their data. Appwrite supports various programming languages and frameworks, enabling developers to use their preferred tools.
Appwrite offers a powerful and versatile BaaS platform for developers seeking a secure, scalable, and self-hosted solution for building backend functionalities in their web, mobile, or other applications.

- **Zod:**

Zod for Robust Validation in React Applications. It is a powerful TypeScript-first schema validation library that integrates seamlessly with React to streamline form validation and data handling.Zod leverages TypeScript's type system to define detailed schemas for your data. This ensures type safety throughout your application, catching potential data validation issues early in the development process. Explicit data types within Zod schemas enhance code readability and maintainability. IDEs can provide autocompletion and type checking, making development more efficient.Zod performs runtime validation of user input, ensuring data adheres to the defined schema before it's used in your application. This prevents unexpected errors and protects your application from invalid data.Zod schemas can be easily shared between your React frontend and backend written in various languages. This ensures consistent data validation across your entire application stackZod works particularly well with popular React form libraries like React Hook Form and Formik. These libraries handle form state management and user interactions, while Zod focuses on data validation.

Overall, Zod is a valuable tool for building robust and type-safe React applications. By leveraging its expressive syntax and integration with popular form libraries, developers can achieve efficient and reliable data validation, leading to a more secure and user-friendly application experience.

- **Tailwind Css:**

Tailwind CSS is a utility-first CSS framework that offers a unique approach to styling web applications. Unlike traditional CSS frameworks that provide pre-built components and classes, Tailwind focuses on providing a collection of low-level utility classes that can be combined to achieve any desired design. Tailwind provides a comprehensive set of utility classes that target various aspects of an element's style, such as its width, height, padding, margin, color, font, and more. These classes have descriptive names that clearly convey their purpose (e.g., w-1/2 for half width, text-red-500 for red color).

You can configure the exact set of utility classes you want to include in your project and even define custom classes based on your specific needs. Tailwind supports responsive design out of the box. Responsive variants of utility classes allow you to easily control styles for different screen sizes (e.g., md:text-center for centering text on medium screens and above). Since styles are directly applied to components using classes, CSS code stays organized and easier to maintain, especially for larger projects. Tailwind's utility-first approach offers a high degree of flexibility. Developers have granular control over the styles applied to each element. By only including the utility classes you actually use, Tailwind can help reduce the overall size of your CSS bundle, improving website loading performance.

Overall, Tailwind CSS is a powerful and versatile tool for building modern web applications. Its utility-first approach offers flexibility, maintainability, and rapid development capabilities. However, it's essential to consider the learning curve and potential for verbosity before adopting Tailwind for your project.

- **Microsoft Windows:**

Microsoft Windows, commonly referred to as Windows, is a group of several proprietary graphical operating system families, all of which are developed and marketed by Microsoft. Each family caters to a certain sector of the computing industry. Active Microsoft Windows families include Windows NT and Windows IoT; these may encompass subfamilies, (e.g. Windows Server or Windows Embedded Compact) (Windows CE). Defunct Microsoft Windows families include Windows 9x, Windows Mobile and Windows Phone.

- **Android:**

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance, with the main contributor and commercial marketer being Google.

Bibliography

- https://reactjs.org/
- https://v5.reactrouter.com/web/guides/quick-start
- https://nodejs.org/en/docs/
- https://fontawesome.com/
- https://stackoverflow.com/
- https://www.youtube.com/
- https://tailwindcss.com/docs/installation/play-cdn
- https://ui.shadcn.com/
- https://appwrite.io/
- https://www.totaltypescript.com/tutorials/zod
- https://app.logo.com/editor/ideas
- https://www.svgrepo.com/