# SOFTWARE DESIGN DOCUMENT

# PROJECT TITLE - FLYSHARE

## 1. INTRODUCTION

This comprehensive software design document presents a detailed overview of the architectural design, system functionalities, and components integral to the FlyShare platform. It encompasses a comprehensive analysis of the system architecture, database structure, and component-level designs, providing a blueprint for the successful implementation and deployment of the Flyshare application.

FlyShare aims to streamline the travel experience by leveraging modern technologies and collaborative efforts among passengers. Through secure user interactions, efficient luggage sharing mechanisms, and a simplified-user interface, FlyShare endeavors to create a seamless and cost-effective solution for managing luggage while fostering connections and cooperation among travelers.

The subsequent sections delve into the architectural intricacies, system functionalities, and detailed component-level designs, providing a thorough understanding of the FlyShare system's Structure and operational capabilities.

## 2. SYSTEM OVERVIEW

FlyShare is designed to offer a platform where airline passengers can connect with co-passengers to share luggage space. The main functionalities include:

**User Authentication and Registration:**

- ✓ Users can sign up for an account or log in securely.
- ✓ Authentication mechanisms ensure data security and user privacy.

**Dashboard and Navigation:**

- ✓ Upon logging in, users are directed to a dashboard interface.
- ✓ Navigation options include "Get Help" and "Post Help" sections for seamless interaction.

**Get Help Section:**

- ✓ Users can view existing posts shared by fellow passengers.
- ✓ Filters based on departure city, destination city, and departure date enable efficient browsing.
- ✓ Users can engage in communication with posters through a chat functionality.

**Post Help Section:**

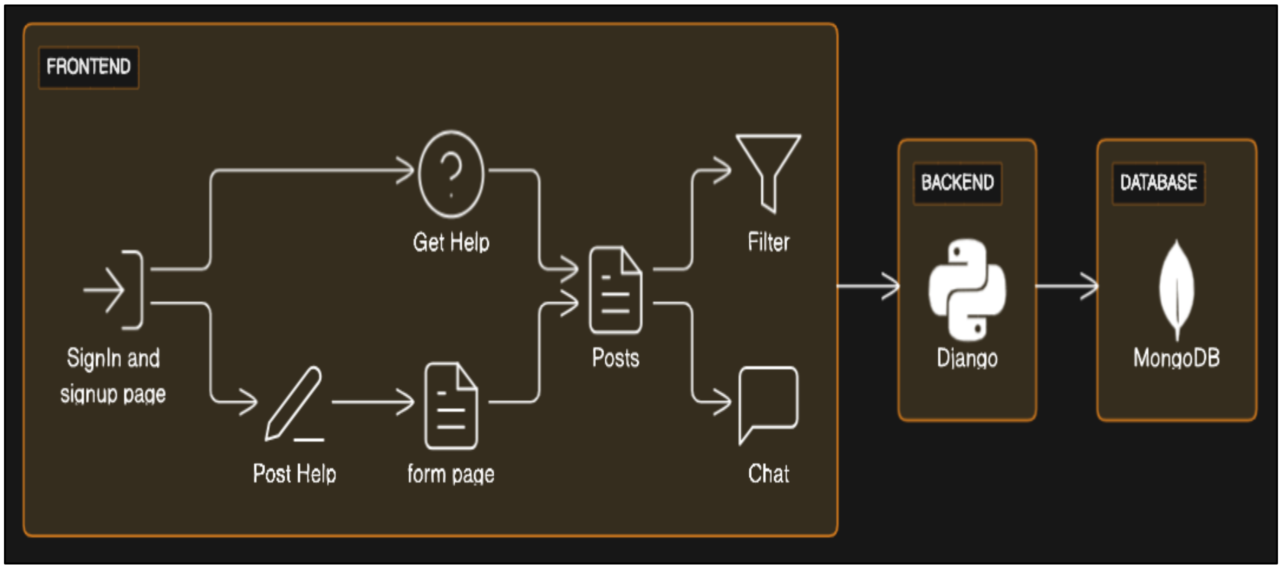
- ✓ Users can create new posts by providing essential details such as passenger name, departure date, PNR number, flight number, departure city, destination city, and luggage description including free space in kgs.
- ✓ Upon submission, a confirmation message is displayed, and the post is added to the available listings.

**Communication Interface:**

- ✓ Real-time chat functionality allows users to communicate securely within the platform.
- ✓ Image upload capabilities facilitate sharing visuals of luggage for better understanding.

## 3. SYSTEM ARCHITECTURE

The system architecture of FlyShare adopts a modern, scalable, and robust design. It leverages Django for backend logic, MongoDB for flexible data storage, HTML/CSS/JavaScript for the frontend, AWS for cloud-based hosting, Docker for containerization, Kubernetes for orchestration, Jenkins for CI/CD, and Postman for API testing.



### 3.1 COMPONENTS:

**Backend -** Django**:**

- ➢ **Purpose:** Handles server logic, API endpoints, and interaction with MongoDB.
- ➢ **Features:** Utilizes Django's powerful framework for rapid development, including ORM for seamless database interaction and user authentication.

**Frontend** - HTML/CSS/JavaScript:

- ➢ **Purpose:** Renders the user interface and enhances user interactions.
- ➢ **Features:** Combines HTML for structuring, CSS for styling, and JavaScript for interactive functionalities, ensuring an engaging user experience.

**Database -** MongoDB:

- ➢ **Purpose:** Stores application data with a flexible schema.
- ➢ **Features:** MongoDB's NoSQL structure allows dynamic data handling and scalability, suitable for accommodating various data types within the application.

## 3.2 INTERACTION FLOW:

### 1. Client Interaction:

Frontend (HTML/CSS/JavaScript) interacts with the user, facilitating user input and displaying information.

### 2. Server-side Processing:

Django handles the business logic, processes user requests, and interacts with MongoDB for data retrieval and manipulation.

### 3. Database Interaction:

MongoDB stores and manages application data, providing a flexible and scalable data storage solution.

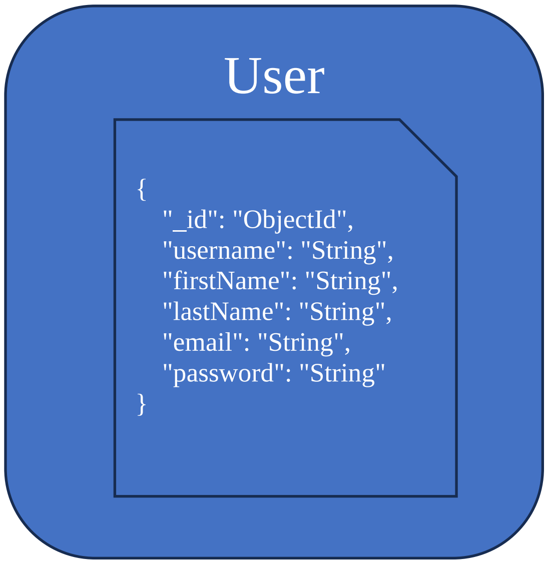
### 4. Deployment & Scaling:

AWS hosts the application, while Docker/Kubernetes ensures containerized deployment, scaling, and management for high availability and reliability.

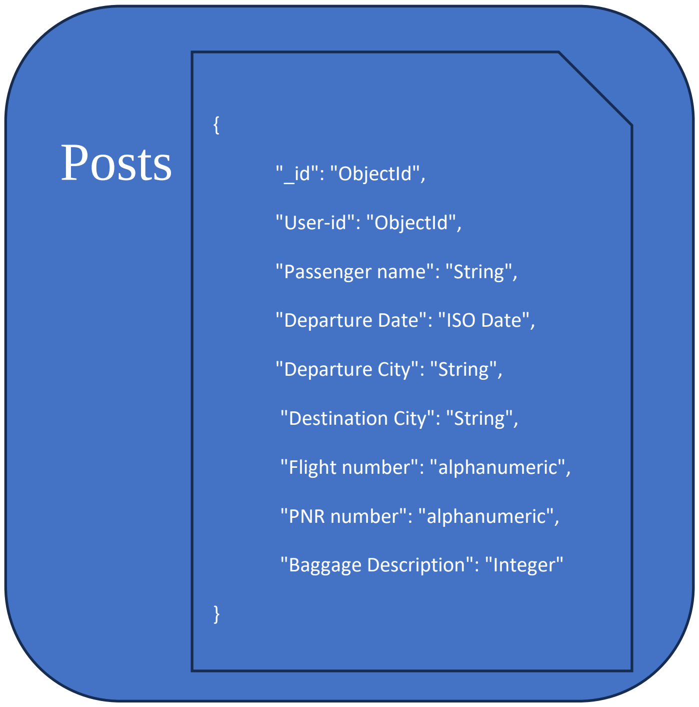
### 5. Automation & Testing:

Jenkins automates the CI/CD pipeline, ensuring smooth deployment, while Postman aids in comprehensive API testing, ensuring functionality and reliability.

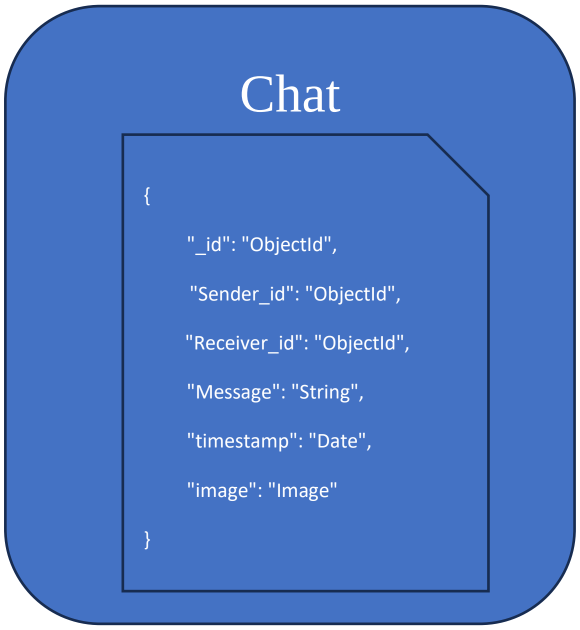
## 4.DATABASE DESIGN


User

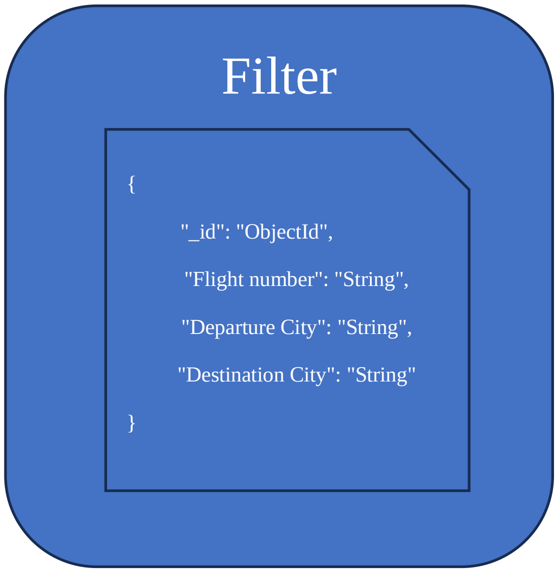
```
{
   "_id": "ObjectId",
   "username": "String",
   "firstName": "String",
   "lastName": "String",
   "email": "String",
   "password": "String"
}
```


Posts

```
{
        "_id": "ObjectId",
        "User-id": "ObjectId",
        "Passenger name": "String",
        "Departure Date": "ISO Date",
        "Departure City": "String",
         "Destination City": "String",
         "Flight number": "alphanumeric",
         "PNR number": "alphanumeric",
         "Baggage Description": "Integer"
}
```


Chat

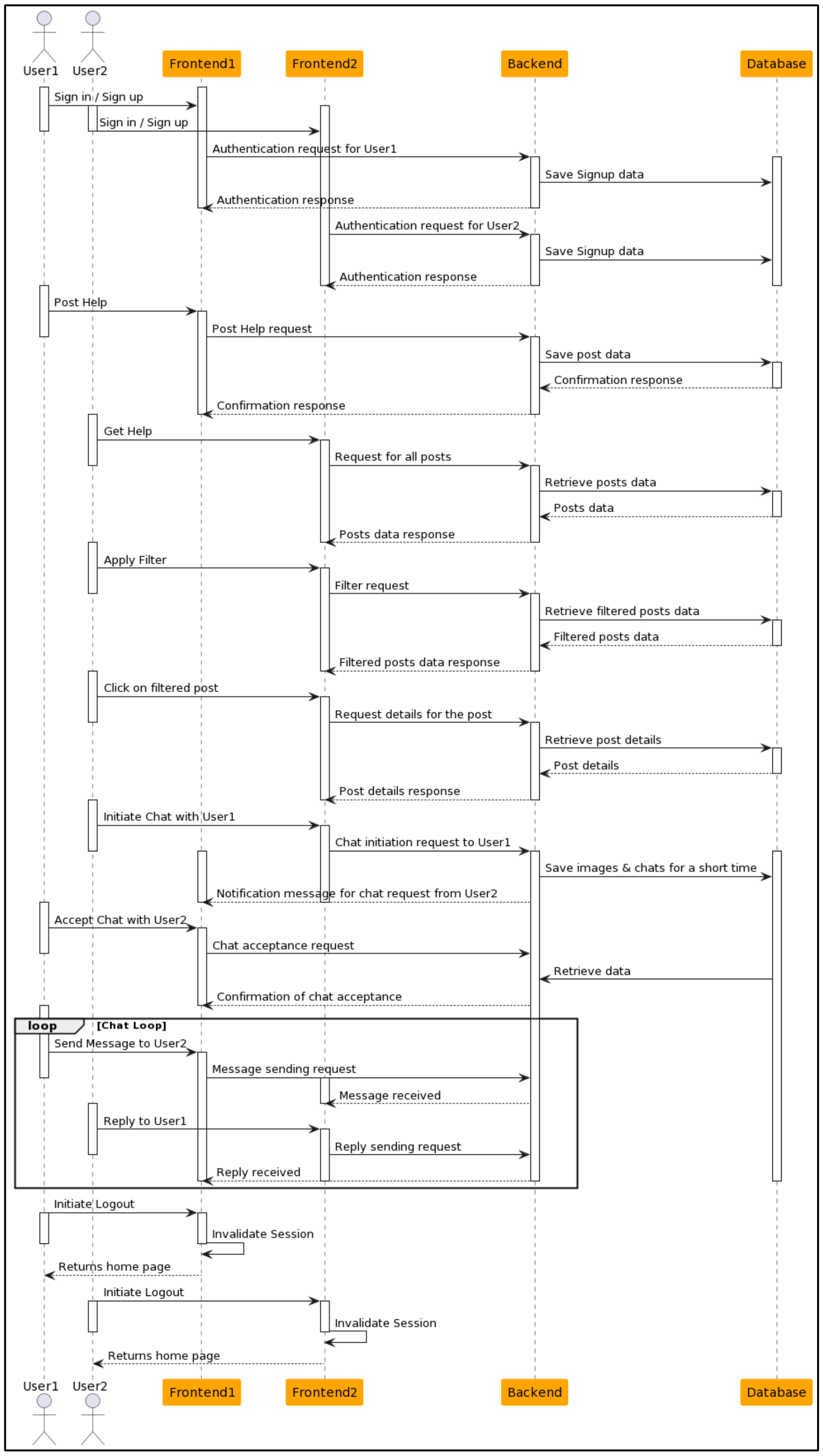
```
{
      "_id": "ObjectId",
       "Sender_id": "ObjectId",
      "Receiver_id": "ObjectId",
      "Message": "String",
      "timestamp": "Date",
      "image": "Image"
}
```


Filter

```
{
      "_id": "ObjectId",
       "Flight number": "String",
      "Departure City": "String",
     "Destination City": "String"
}
```

# 5. COMPONENT LEVEL DESIGN

## 5.1 SEQUENCE DIAGRAM



## Sign-in/Sign-up Process:

- ✓ User1 and User2 initiate the sign-in or sign-up process via their respective frontend interfaces (Frontend1 and Frontend2).

- ✓ The frontends (Frontend1 and Frontend2) send authentication requests to the Backend.
- ✓ The Backend authenticates the users with the Database and sends back authentication responses to the frontends.

**Post Help and View All Posts:**

- ✓ User1 posts help or gets help using Frontend1, and the Frontend1 sends a request to the Backend to save post data in the Database.
- ✓ User2 views all available posts through Frontend2, prompting a request to the Backend for retrieving posts data.
- ✓ The Backend retrieves posts data from the Database and forwards it to Frontend2 as a response.

**Filtering Posts:**

- ✓ User2 applies a filter on posts via Frontend2, triggering a request to the Backend for filtered posts data.
- ✓ The Backend retrieves filtered posts data from the Database and sends it back as a response to Frontend2.

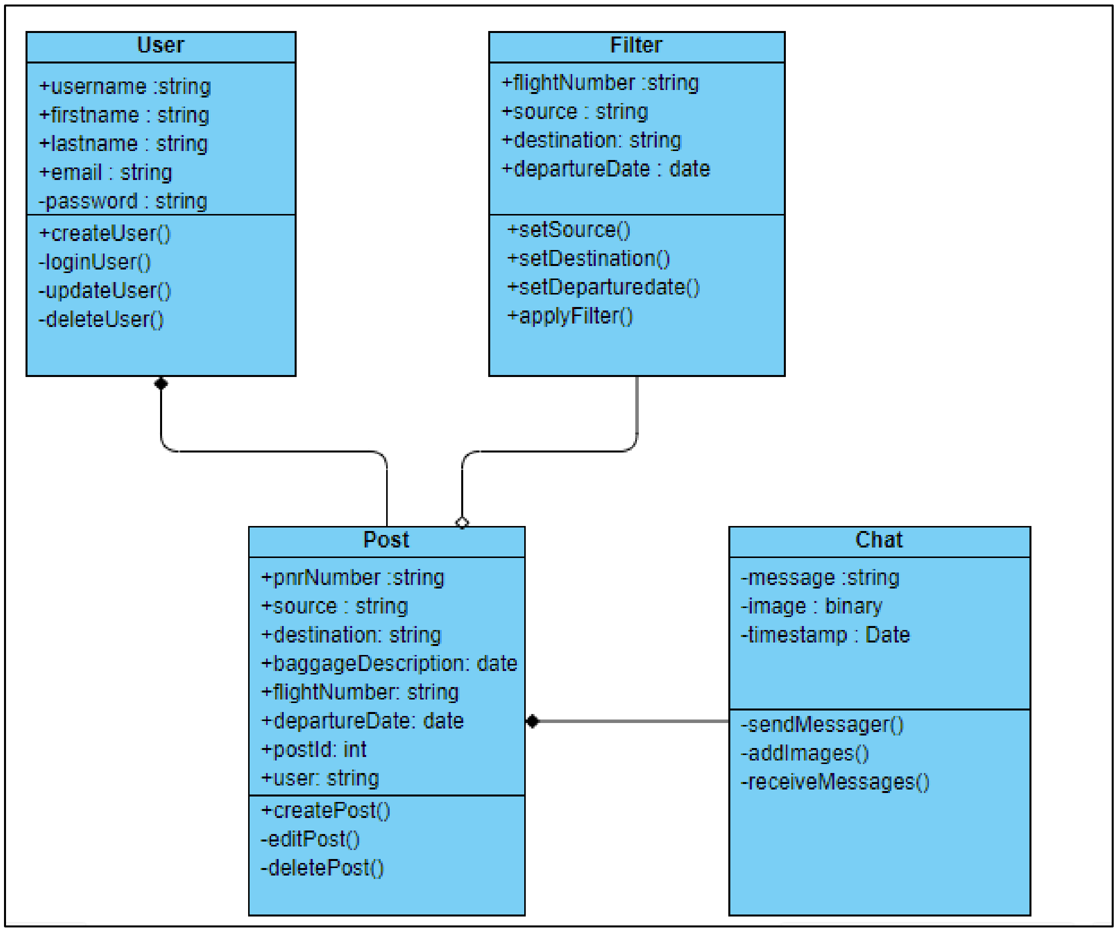
**Clicking on a Filtered Post:**

- ✓ User2 clicks on a post from the filtered posts list via Frontend2, resulting in a request for post details to the Backend.
- ✓ The Backend retrieves specific post details from the Database and sends them as a response to Frontend2.

**Initiating and Accepting Chat:**

- ✓ User1 initiates a chat with User2 through Frontend1, prompting a chat initiation request to the Backend.
- ✓ The Backend sends a notification message to User2 through Frontend2 regarding the chat request from User1.
- ✓ User2 accepts the chat request through Frontend2, triggering a chat acceptance request to the Backend.
- ✓ The Backend confirms chat acceptance and communication between User1 and User2 continues.

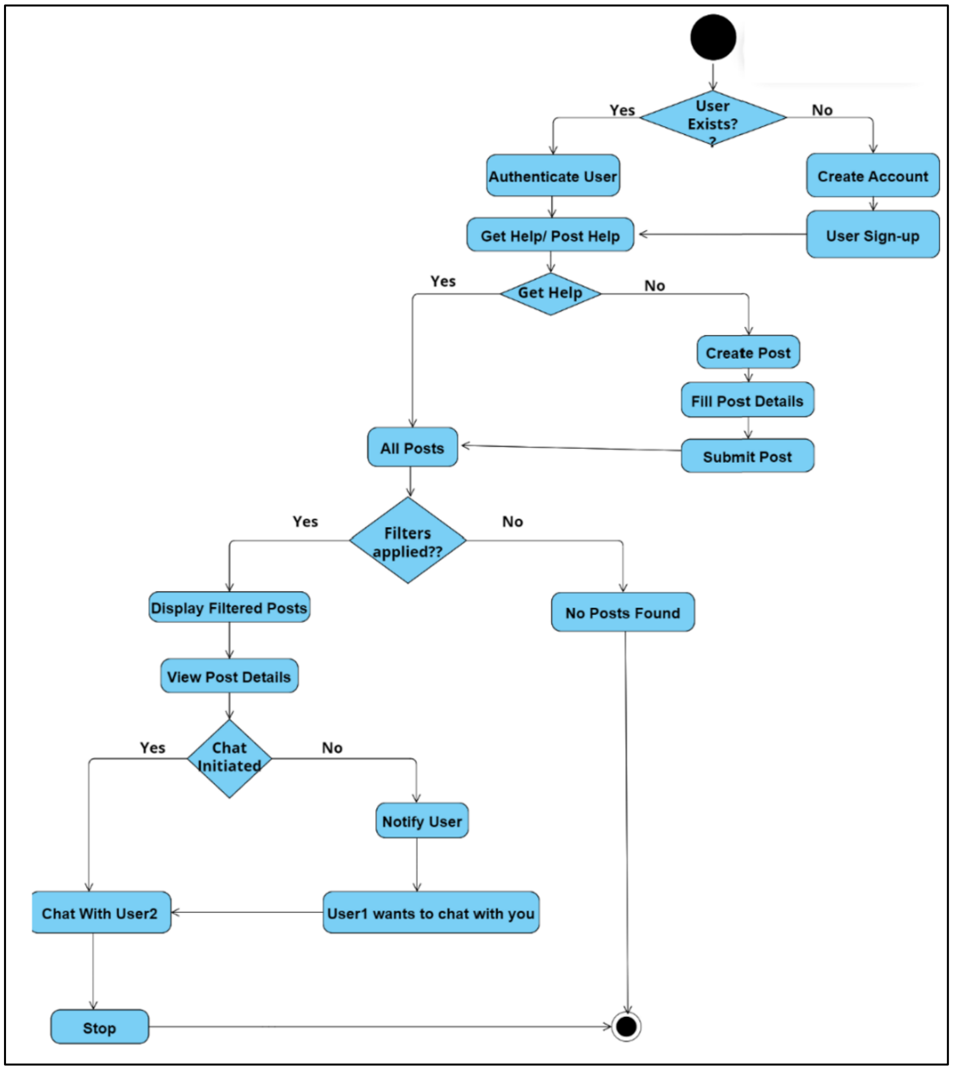
## 5.2 CLASS DIAGRAM



The class diagram delineates the structural blueprint of a system designed for facilitating interactions among users during travel. It comprises several essential classes: User, Post, Filter, Chat, and MongoDB. The User class encapsulates attributes like username, firstname, lastname, email, and password, representing users within the system, along with associated methods for user management such as createUser, loginUser, updateUser, and deleteUser. The Post class signifies the posts created by users during travel, holding attributes like postId, departureDate, pnrNumber, flight Number, source, destination, and baggage Description. This class establishes a relationship where a single user can create multiple posts, enabling a "one-to-many" connection.

The Filter class introduces attributes such as flight number, source, destination, and departureDate to filter posts. This class correlates to the Post class, allowing multiple posts to correspond to a filter. The Chat class incorporates an array of participants, likely representing users engaged in specific chat sessions, enabling communication between posts. This class relates to the Post class, illustrating that multiple chat interactions can be associated with a single post. Lastly, the MongoDB class represents database functionalities responsible for persisting and retrieving user and post-related data within the system.

## 5.3 ACTIVITY DIAGRAM



This activity diagram delineates the sequence of actions within a travel-oriented web application, starting with user interaction and progressing through various functionalities. It begins by checking if a user exists; if affirmative, the system proceeds with authenticating the user. Upon successful authentication, users are directed to the primary functionalities, namely "Get Help / Post Help." Within this branch, if a user opts to seek assistance ("Get Help"), the system allows filtering of posts. If the user applies a filter, the system displays filtered posts.

Then, users can view the details of specific posts, and if a chat is initiated, they can engage in conversation ("Chat with Co-passenger"). If no chat is initiated, a notification is sent to prompt engagement: "User1 wants to chat with you." Conversely, if the user chooses to provide help ("Post Help"), they can create a post, fill in post details, and submit the post for others to view or interact with.