**Class Diagrams**

1. **SignIn:**

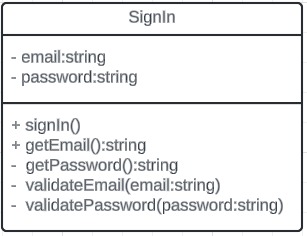
The **SignIn** class manages the login process for users by storing their **email** and **password** and providing methods for validating and retrieving these values.

**Attributes:**

* **email** (String): Stores the user's email address.
* **password** (String): Stores the user's password.

**Functions:**

* **getEmail()**: Returns the user's email address.
* **getPassword()**: Returns the user's password.
* **validateEmail()**: Validates if the email address is in the correct format (e.g., contains "@" and a domain).
* **validatePassword()**: Checks if the password meets the required security criteria (e.g., length and complexity).
* **signIn()**: Handles the authentication process by validating both the email and password, then verifying if they match a registered user's credentials.



1. **SignUp:**

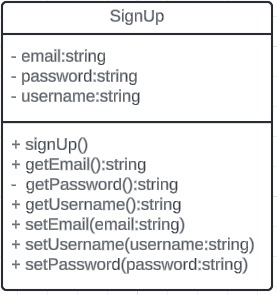
The **SignUp** class manages the registration process for new users by storing their **username**, **email**, and **password**. It provides methods to set, retrieve, and validate user information during the signup process.

**Attributes:**

* **username** (String): Stores the user's chosen username.
* **email** (String): Stores the user's email address.
* **password** (String): Stores the user's password.

**Functions:**

* **getUsername()**: Returns the user's username.
* **getEmail()**: Returns the user's email address.
* **getPassword()**: Returns the user's password.
* **setUsername(username: String)**: Sets the user's username.
* **setEmail(email: String)**: Sets the user's email address.
* **setPassword(password: String)**: Sets the user's password.
* **signup()**: Handles the signup process by validating the provided username, email, and password, then registering the new user in the system.



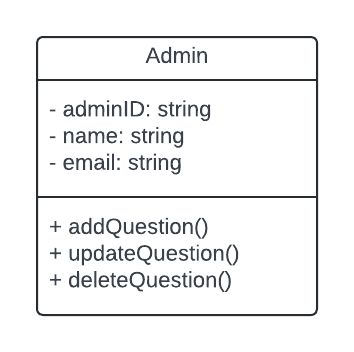
1. **Admin:**

The **Admin** class represents the administrator of the quiz system. It holds attributes like **adminID**, **name**, and **email** to identify the admin. The class provides methods to manage quiz questions, including adding, updating, and deleting questions.

**Attributes:**

* **adminID** (String): Stores the admin’s ID.
* **email** (String): Stores the admin’s email address.
* **name** (String): Stores the admin’s name.

**Functions:**

* **addQuestion()**: Adds a new question to the quiz database for the users to answer.
* **updateQuestion()**: Updates an existing question in the quiz database based on its ID.
* **deleteQuestion():** Removes a question from the quiz database using its ID.

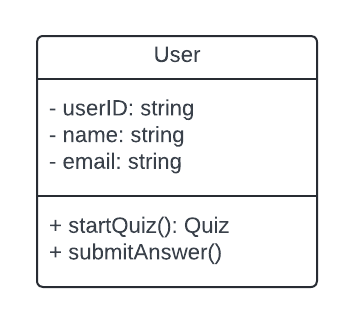
1. **User:**

The **User** class represents the participants who take quizzes. Attributes such as **userID**, **name**, and **email** are used to store user information. The class includes methods to start a quiz and submit answers after completion.

**Attributes:**

* **userID** (String): Stores the user’s ID.
* **email** (String): Stores the user’s email address.
* **name** (String): Stores the user’s name.

**Functions:**

* **startQuiz()**: Initiates a new quiz session with randomly selected questions.
* **submitAnswer()**: Submits the user's answers to the quiz for evaluation.

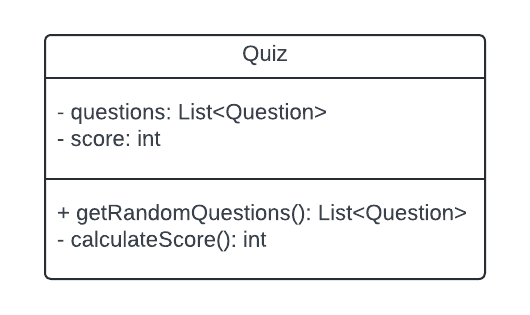
1. **Quiz:**

The **Quiz** class is responsible for managing a quiz session. It contains a list of questions and keeps track of the user's score. Methods are provided to retrieve random questions for the quiz and calculate the score based on the user’s answers.

**Attributes:**

* **questions** (List<Question>): A list of questions included in the quiz.
* **score** (Integer): The total score calculated based on the user's answers.

**Functions:**

* **getRandomQuestions()**: Retrieves a set of randomly selected questions for the quiz.
* **calculateScore()**: Calculates the quiz score based on the user's submitted answers.

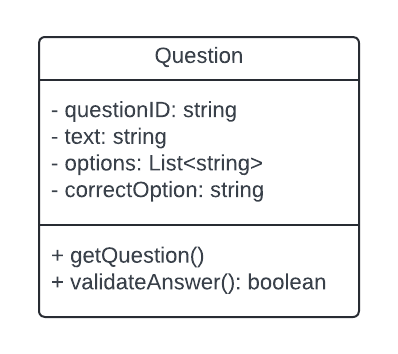
1. **Questions:**

The **Question** class defines individual quiz questions. Attributes like **questionID**, **text**, **options**, and **correctOption** are used to store the question details. It includes methods to present a question and validate the user's selected answer.

**Attributes:**

* **questionID** (String): A unique identifier for each question.
* **text** (String): The content of the quiz question.
* **options** (List<String>): A list of possible answer choices for the question.
* **correctOption** (String): A list of possible answer choices for the question. The correct answer for the question.

**Functions:**

* **getQuestion()**: Retrieves the question's text and options for display.
* **validateAnswer()**: Checks if the user's selected answer is correct.

### **BlogPage**

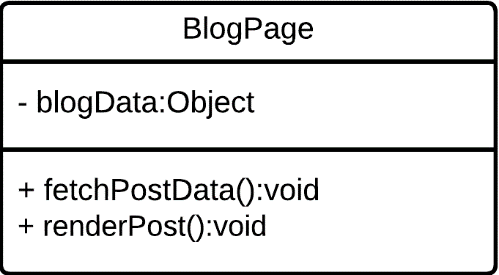
The **BlogPage** class is responsible for handling the display of blog data. It fetches the required data and renders posts for the users.

**Attributes:**

* **blogData (Object)**: Stores the data needed for rendering blog posts on the page.

**Functions:**

* **fetchPostData()**: Retrieves the post data from the backend (likely from MongoDB or Firebase) to be displayed on the page.
* **renderPost()**: Renders the fetched post data on the blog page for the users to view.



### **Post**

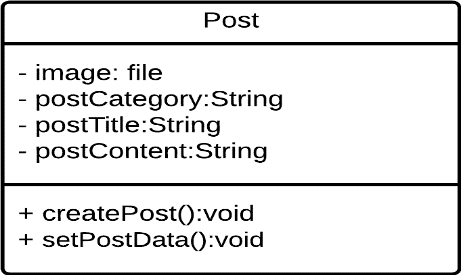
The **Post** class defines the structure of an individual post. It holds attributes related to the blog post like the image, category, title, and content.

**Attributes:**

* **image (file)**: The image associated with the post.
* **postCategory (String)**: The category of the blog post.
* **postTitle (String)**: The title of the blog post.
* **postContent (String)**: The content of the blog post.

**Functions:**

* **createPost()**: Creates a new post with the given details.
* **setPostData()**: Sets or updates the post data with new information like title, category, content, or image.



### **Database**

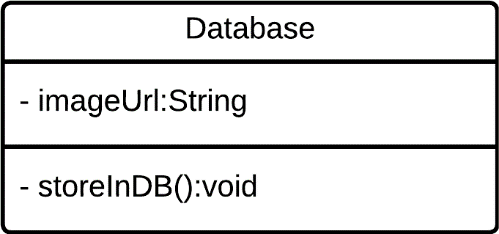
The **Database** class manages the storage of the image URLs linked to the blog posts.

**Attributes:**

* **imageUrl (String)**: The URL of the image associated with the blog post, stored in Firebase or another storage service.

**Functions:**

* **storeInDB()**: Stores the image URL or other post data in the database.



### **MongoDb**

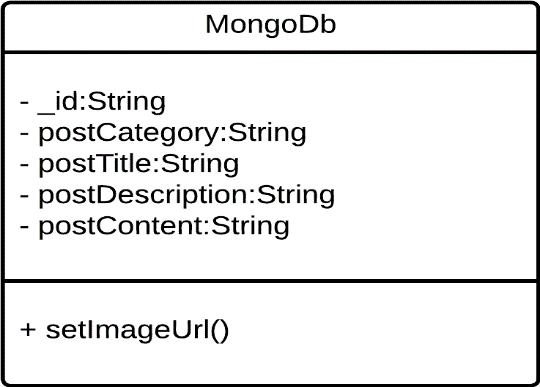
The **MongoDb** class is responsible for storing post metadata (category, title, description, and content) into a MongoDB database.

**Attributes:**

* **\_id (String)**: A unique identifier for each post in the MongoDB database.
* **postCategory (String)**: The category of the post.
* **postTitle (String)**: The title of the post.
* **postDescription (String)**: The description or summary of the post.
* **postContent (String)**: The full content of the post.

**Functions:**

* **setImageUrl()**: Updates the image URL in MongoDB, linking it to the respective post.



### **Firebase**

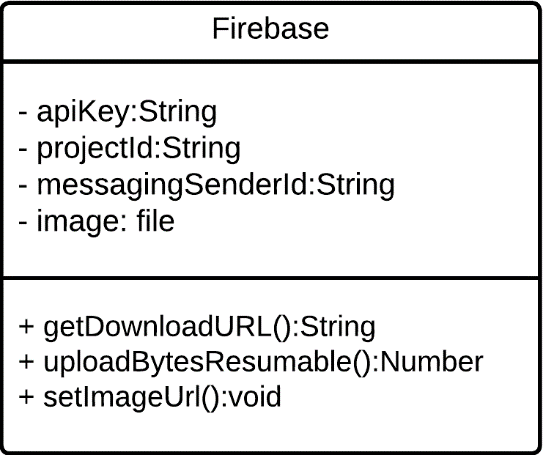
The **Firebase** class is responsible for handling Firebase-specific tasks such as storing images and retrieving download URLs.

**Attributes:**

* **apiKey (String)**: The API key for accessing Firebase services.
* **projectId (String)**: The unique identifier for the Firebase project.
* **messagingSenderId (String)**: The identifier used for Firebase Cloud Messaging.
* **image (file)**: The image file that is uploaded to Firebase storage.

**Functions:**

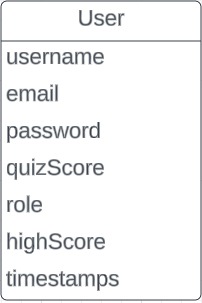
* **getDownloadURL()**: Retrieves the download URL of an uploaded image from Firebase.
* **uploadBytesResumable()**: Uploads the image to Firebase storage with resumable upload support.
* **setImageUrl()**: Sets or updates the image URL in Firebase, possibly for linking it to a database.



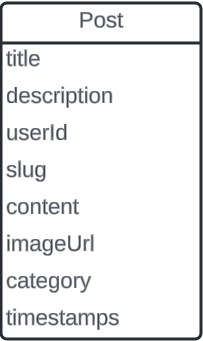
**Database diagrams:**

We use a non-relational database (**MongoDB**) to manage our data. The system includes three key schemas:

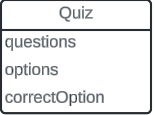
1. **User Schema**: Stores user information such as username, email, password, quiz score, role and high score.



1. **Post Schema**: Manages blog posts with details like title, description, user ID, slug, content, image URL, category, timestamps.



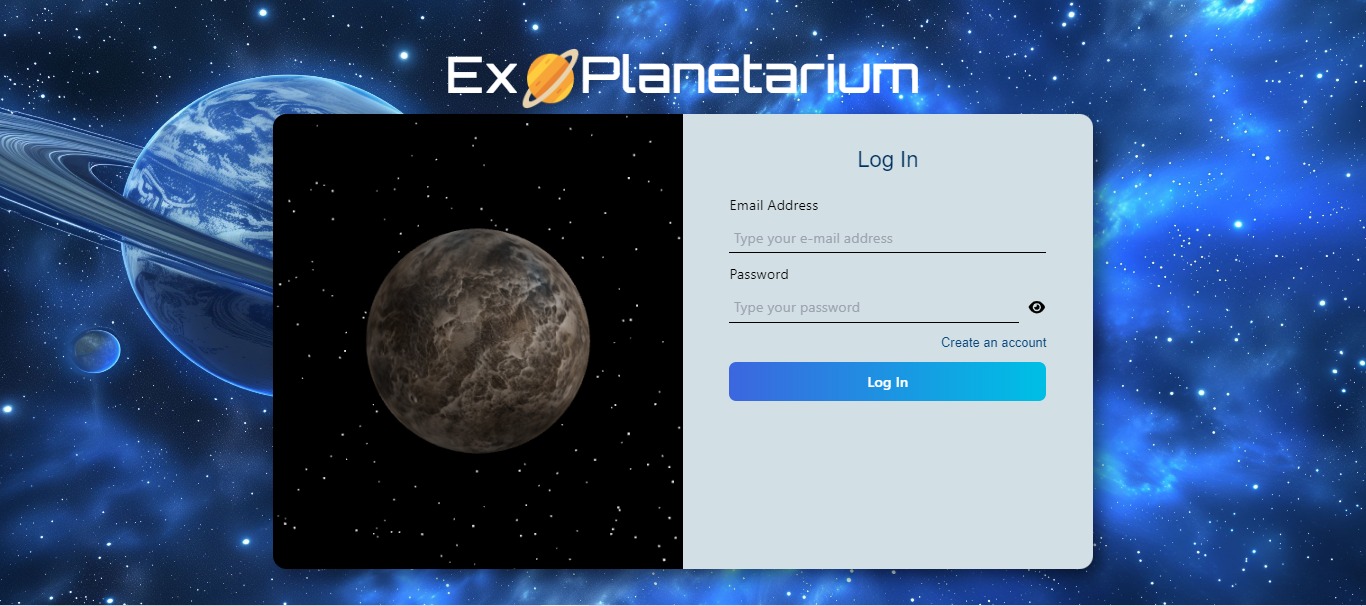
1. **Quiz Schema**: Defines quiz questions, options, and correct answers.



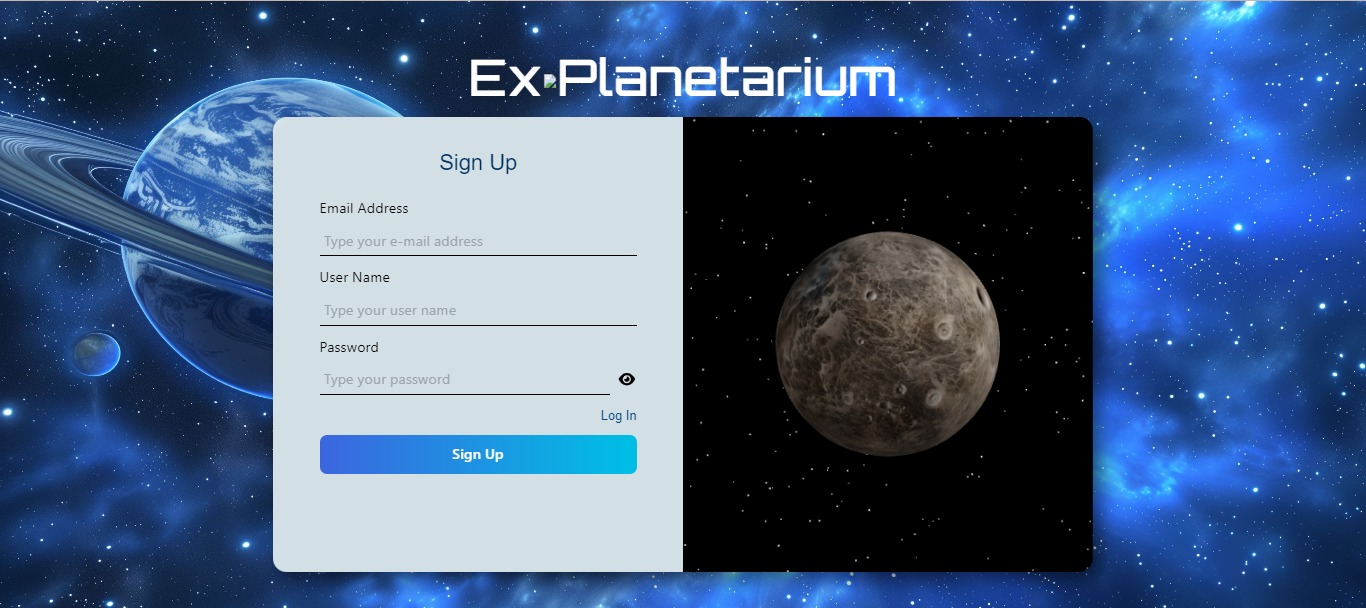
These schemas enable a flexible structure for storing and accessing data efficiently in MongoDB.

**User Interfaces:**

* 1. **Login:**



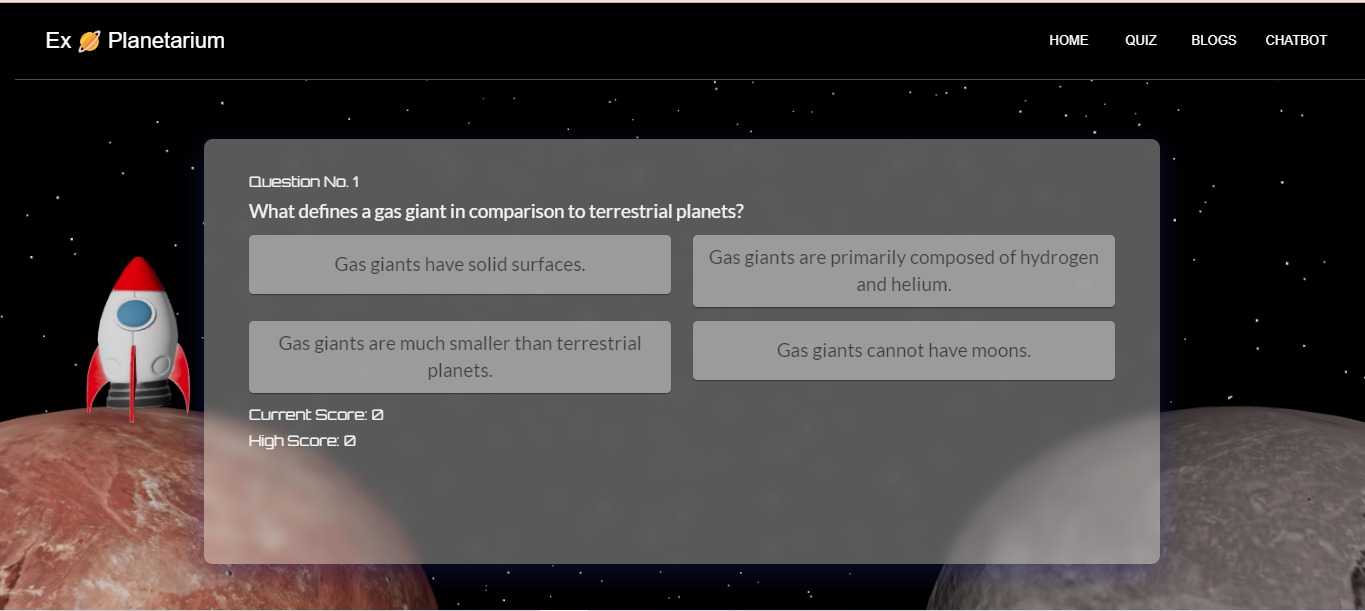
* 1. **SignUp**



* 1. **Home**



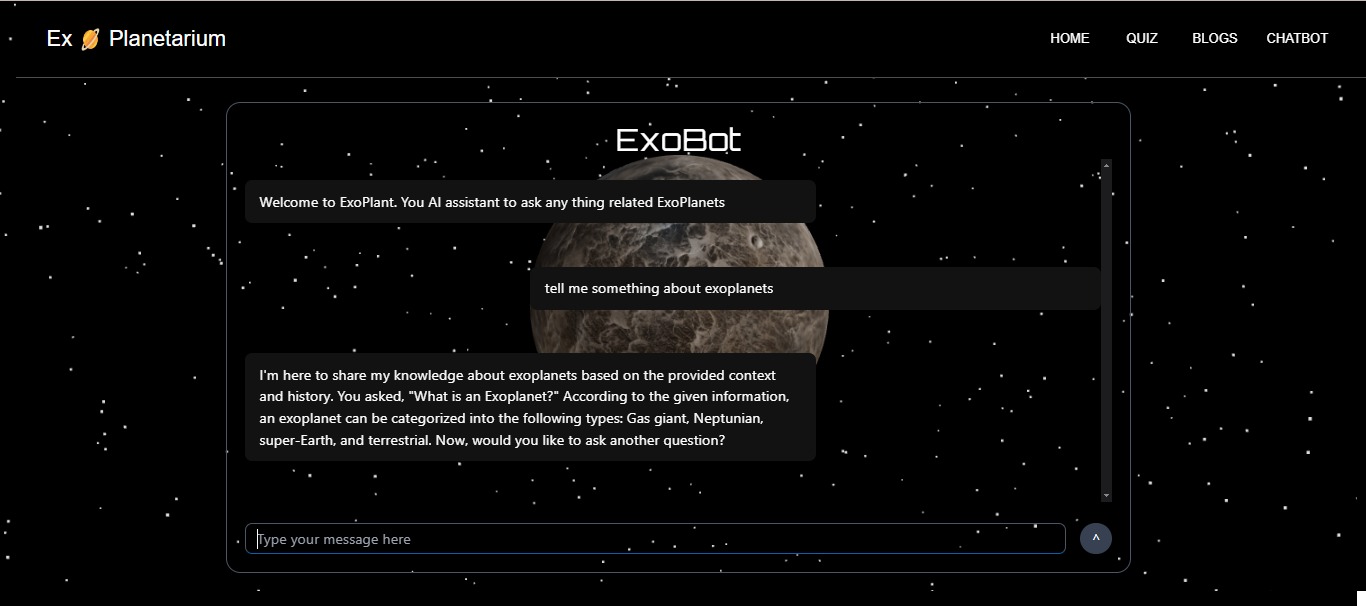
* 1. **Quiz**



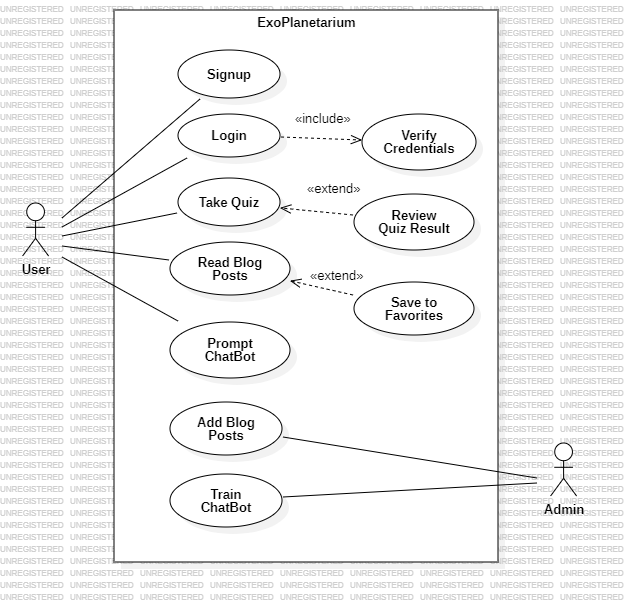
* 1. **Blogs/Articles**



* 1. **ExoBot (Chatbot)**



**Usecase Diagram:**



**Actors:**

 **User:** A general user of the ExoPlanetarium platform who interacts with various features such as quizzes, blog posts, and the chatbot.

 **Admin:** A privileged user responsible for managing content, training the chatbot, and overseeing other administrative tasks.

1. **Signup:**

* A user can create an account in the ExoPlanetarium system.

1. **Login:**

* A user logs into their account.
* **Includes:**
  + **Verify Credentials:** The system checks if the user's login details are correct.

1. **Take Quiz:**

* Users can participate in a quiz related to the ExoPlanetarium content.
* **Extends:**
  + **Review Quiz Result:** After completing the quiz, the system allows users to review their results.

1. **Read Blog Posts:**

* Users can access and read blog posts within the platform.
* **Extends:**
  + **Save to Favorites:** Users have the option to save specific blog posts to their favorites list.

1. **Prompt ChatBot:**

* Users can interact with a chatbot to get responses related to ExoPlanetarium content or topics.

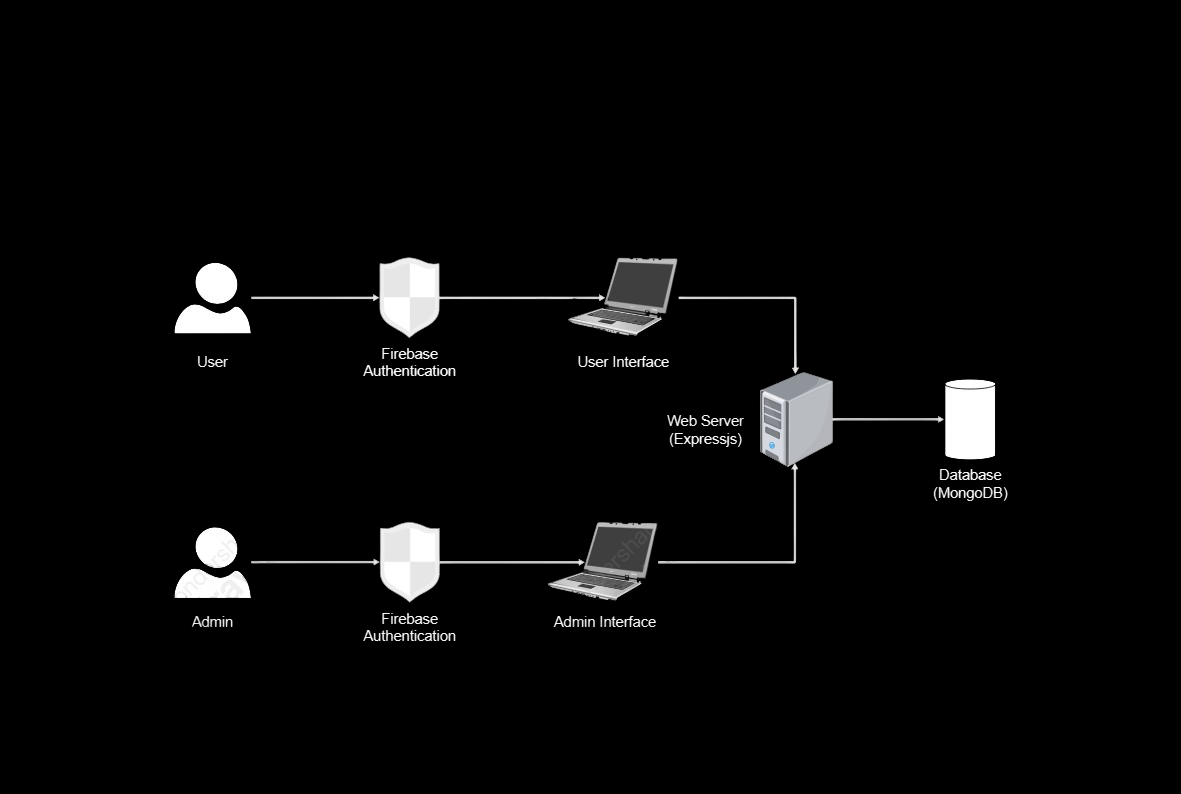
1. **Add Blog Posts:**

* This feature allows admin to add new blog posts related to the Exoplanets, in to the system.

1. **Train ChatBot:**

* Admin can help train the chatbot, to enhance its responses or data.

**Architecture Diagram:**



**User and Admin:**

* These represent the two primary roles in ExoPlanetarium.

**Firebase Authentication:**

* This service handles user authentication and authorization, ensuring secure access to the application.

**User Interface:**

* Provides information about exoplanets, interactive quizzes, blog posts, and a chatbot designed for ExoPlanet conversations. Users can take quizzes, read blog posts, and engage in conversations with the Exobot (chatbot).

**Admin Interface:**

* Allows administrators to add and manage blog posts, ensuring that only authorized personnel can create or modify content.

**Web Server (Express.js):**

* This server acts as the intermediary between the user and admin interfaces and the database. It handles incoming requests, processes them, and interacts with the database to retrieve or store data related to exoplanet quizzes, blog posts, and Exobot (chatbot) interactions.

**Database (MongoDB):**

* Stores exoplanet and user data, including information about exoplanets, user quiz scores, blog posts and user credentials in an encrypted format.