

Project Architecture

Author: Phuong Nguyen

I. Create User, Login

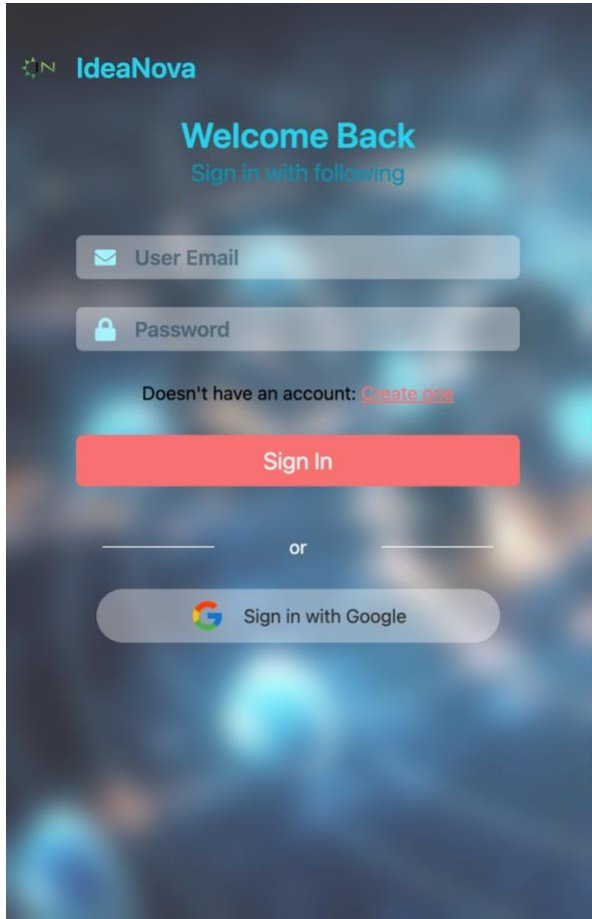


Figure 1: main screen for login

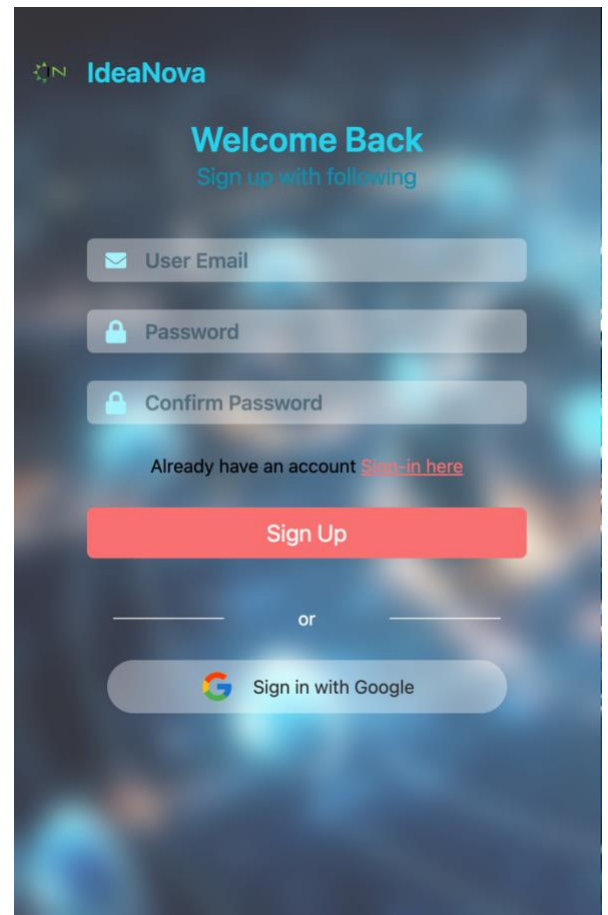


Figure 2: Sign up screen

The app offers two methods to log in:

1. Creating a New Account:

- Click the “Create one” button, as shown in the image above.
- You will be redirected to the account creation page (Figure 2), where you can register using your email and a password of your choice.

- Once you have filled out the required fields, click the Sign Up button.
- After successfully creating an account, you will be redirected to the main page.

2. Logging in with Google:

- Click the Sign in with Google button.
- A new tab will open, prompting you to select your Google email account (Figure 3).
- After selecting your email account, you will be redirected to the main page.

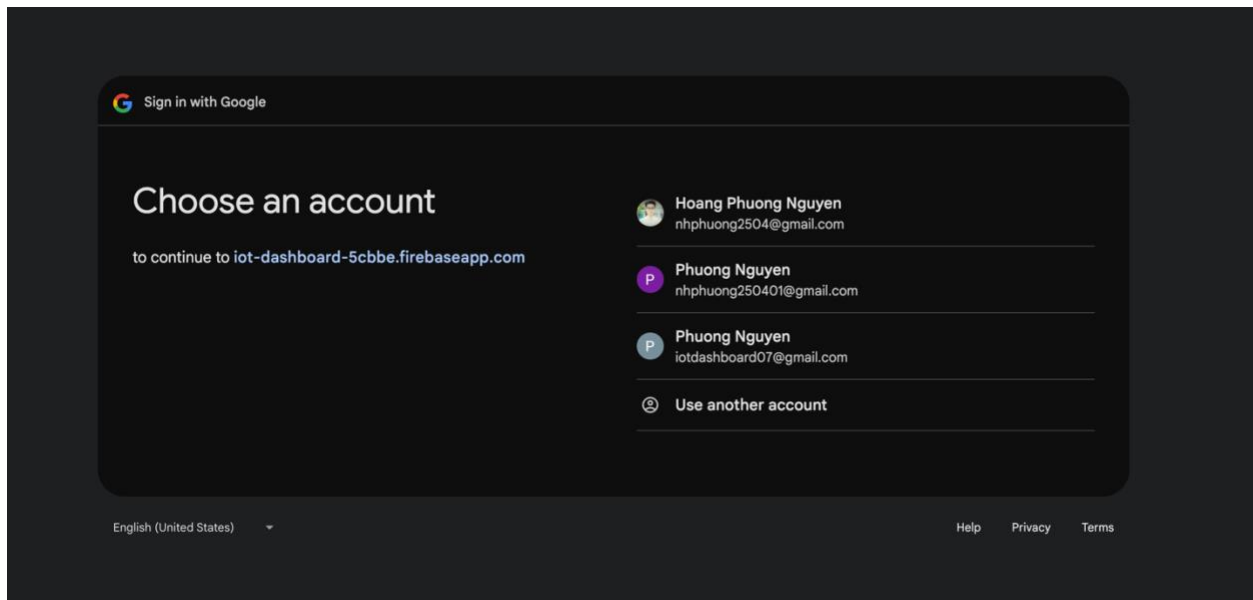


Figure 3: Sign in with Google method

II. Gmail Account for the Project

A dedicated Gmail account has been created for use with Google Firebase and MongoDB for this project:

- **Email:** iotdashboard07@gmail.com

- **Password:** Onetwothree123

However, you are welcome to create a new Google Firebase account or MongoDB account for your company if preferred. Below are step-by-step instructions for setting them up:

1. How to Set Up and Use Google Firebase:

Step 1: Access Firebase

- Go to [Google Firebase](#).

Step 2: Sign In

- Sign in with your Gmail account.

Step 3: Open the Firebase Console

- Click the **"Go to console"** button located next to your avatar.

Step 4: Create a Project

- Click **"Create a project"**.
- Enter a name for your project and click **"Create project"**. (This process may take a few moments.)

Step 5: Set Up Firestore Database

- From the left-hand menu, select the **"Build"** dropdown and choose **"Firestore Database"**.
- Click **"Create database"**.
 - A setup tab will appear where you can specify the database name and location. Click **"Next"**.
 - Select the **"Start in production mode"** option and proceed.

- You can repeat similar steps for "**Storage**" or "**Realtime Database**", if these features are relevant to your project.

Step 6: Set Up Authentication

- From the "**Build**" dropdown, choose "**Authentication**" and click "**Get started**".
- Enable the "**Email/Password**" and "**Google**" options as sign-in providers.

Step 7: Create and Link a Web App

- Go to the "**Project Overview**" section and select "**Project settings**".
- Under the "General" tab, create an app to link with your Firebase project.
 - Choose the **Web app icon** (`</>`) and name your app.
 - Register the app and complete the setup process.
 - You will receive your Firebase configuration details.

Note: I saved the Firebase configuration details in an .env file inside the client folder. You can reuse or replace them depending on your needs.

2. How to Set Up and Use MongoDB:

Step 1: Access MongoDB

- Go to [MongoDB](#).

Step 2: Log In and Create a Cluster

- Sign in using your preferred account.
- During setup, you may be asked to answer a few questions or choose a plan. Select options that suit your needs.
- Once completed, click "**Create Cluster**".

Step 3: Create a User

- Under the **"Quickstart"** option, create a new database user.

Step 4: Configure Network Access

- Navigate to **"Network Access"**.
- Add an IP Address to grant permission for database access from that specific IP.

Step 5: Retrieve Connection String

- Go to the **"Database"** section and choose **"Connect"**, then select **"Connect your application"**.
- MongoDB will generate a **connection string** for your app.
 - Copy this string and replace it in the .env file inside the server folder as needed.

Step 6: Store User Information

- When users sign up or log in using Google, their information will be saved in MongoDB.
 - You can view these details in the **"Database"** → **"Collections"** section of your MongoDB dashboard.

III. How to Use the Admin Option

Header Information

- The header of the application retrieves and displays information from your account, including:
 - **Name**
 - **Avatar**

- **Role** (e.g., Member or Admin)

Default Role

- By default, every new user is assigned the role of **"Member"**.
- Members do **not** have access to the **"Management"** option, which is only visible to users with the **"Admin"** role.

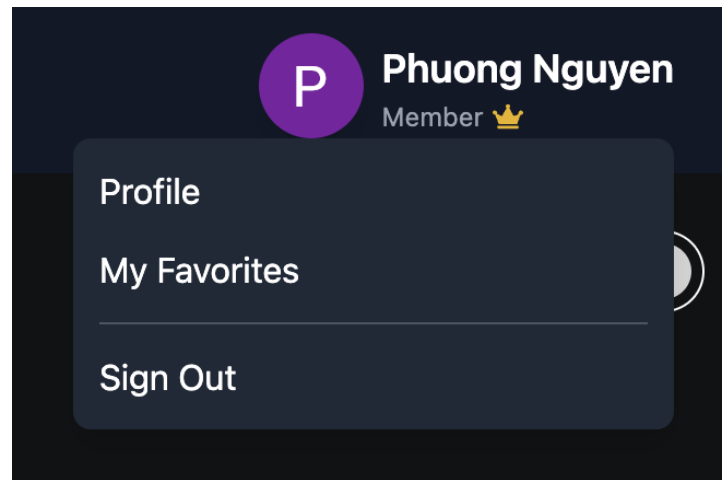
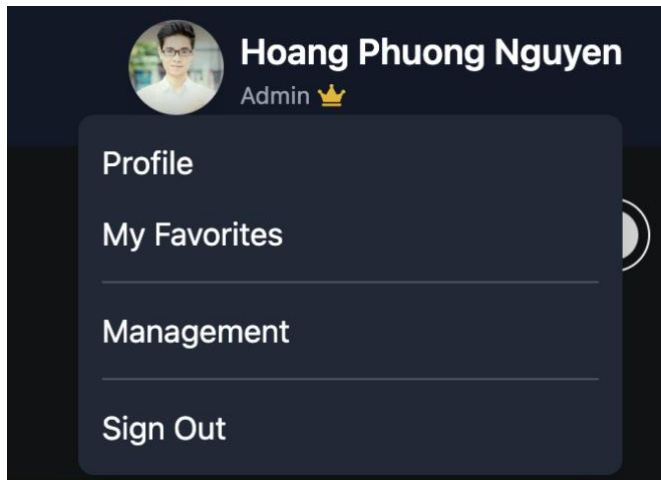


Figure 4: Different between roles

Management Panel (Admin Access Only)

Once logged in as an admin, you can access the **"Management"** option. Here's what you can do:

View User Information

- The Management panel displays a list of all accounts with the following details:
 - **Image:** Profile picture of the user.
 - **Name:** The user's display name.

- **Email:** The registered email address.
- **Verified:** Indicates whether the account is email-verified.
- **Created Time:** The date and time the account was created.
- **Role:** Displays whether the user is an **Admin** or a **Member**. (Figure 5)

Change Roles

- As an admin, you can change the role of other users:
 - To change a user's role, click the **purple button** located next to their current role.
 - For example, you can switch a user's role from **Member to Admin** or from **Admin to Member**. (Figure 6)

Delete Accounts

- You can delete other user accounts by clicking the **trash icon** located at the beginning of the user's row. (Figure 7)

Important Notes:

- You cannot delete or change the role of your own account.
- These restrictions ensure that an admin cannot accidentally or maliciously remove their own admin privileges.

Total : 7





Image	Name	Email	Verified	Created	Role
	Hoang Phuong Nguyen	nhphuong2504@gmail.com	True	October 15th 2024, 7:28:26 pm	admin
	Phuong Nguyen	nhphuong250401@gmail.com	True	October 15th 2024, 8:50:36 pm	member Admin
	Lucas Ericson	lericson@ideanovatech.com	True	October 25th 2024, 5:21:40 pm	admin Member
	Phuong Nguyen	iotdashboard07@gmail.com	True	October 25th 2024, 5:44:47 pm	admin Member

Figure 5: Management main page

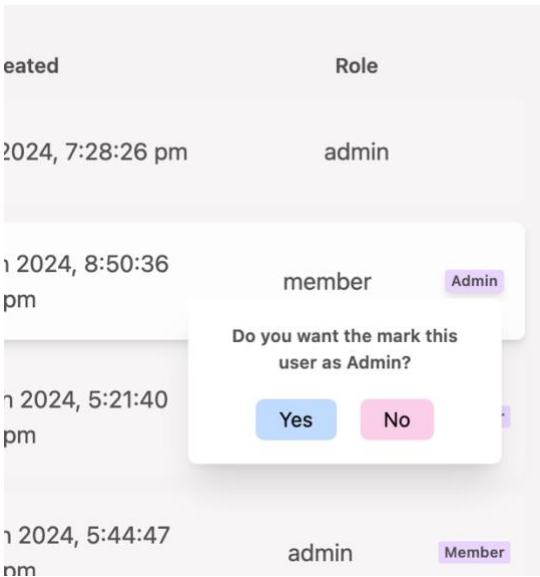


Figure 6: Option to change user roles

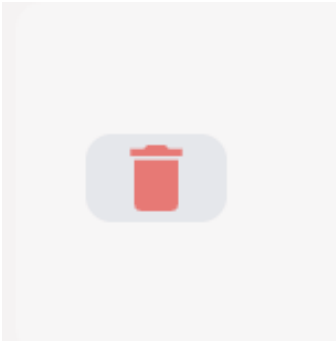


Figure 7: Icon to delete user

IV. Backend Overview

The backend is organized into key categories for better structure and maintainability:

1. Core Components

- **app.js:**
 - The main entry point for the backend application.
 - Configures middleware, sets up routing, and initializes the server.

- **routes/:**
 - Contains the route definitions for various API endpoints.
 - Each route handles specific functionality, like user authentication, data retrieval, or management actions.
- **models/:**
 - Defines the database schemas and models for managing data in MongoDB.
 - Handles interaction with the database to perform CRUD operations.
- **config/database/:**
 - Contains configuration files for database connections and external services.
 - Includes files for setting up MongoDB and AWS S3.

2. AWS Bucket Integration

- All tasks related to managing AWS S3 buckets are handled in **config/database/s3bucket.js**.
 - This includes operations such as uploading, downloading, and managing files in the S3 bucket.

General Endpoints

1. GET /api/airlines

Retrieves a list of all airlines from the S3 status bucket.

2. GET /api/fleets

Fetches fleets for a specified airline.

Query Parameters:

- **airline:** Name of the airline.

3. **GET /api/subfleets**

Fetches subfleets for a specified airline and fleet.

Query Parameters:

- airline: Name of the airline.
- fleet: Name of the fleet.

4. **GET /api/tailids**

Retrieves tail IDs for a specified airline, fleet, and subfleet.

Query Parameters:

- airline: Name of the airline.
- fleet: Name of the fleet.
- subfleet: Name of the subfleet.

5. **GET /api/campaign**

Lists all files in a specific tail ID directory.

Query Parameters:

- airline, fleet, subfleet, tailID: Hierarchical identifiers.

6. **GET /api/progress**

Retrieves progress data for a specified tail ID.

Query Parameters:

- airline, fleet, subfleet, tailID: Hierarchical identifiers.

S3 Utility Functions

Functions for AWS S3 Operations

1. checkS3Connection

- **Purpose:** Verifies connectivity to the specified S3 buckets and checks for objects under predefined prefixes.
- **How it works:**
 - Calls checkS3Bucket for two specific buckets: the Status bucket (for monitoring progress) and the Media bucket (for content files).
 - Logs the number of objects found under each prefix.
- **Usage:** Ensures the application can access and interact with S3 before serving requests.
- **Key Dependencies:** ListObjectsV2Command

2. listFilesInBucket(bucketName, prefix = "", suffix = "")

- **Purpose:** Lists all files or directories in a given bucket, filtered by an optional prefix or suffix.
- **How it works:**
 - Sends a ListObjectsV2Command request to the S3 bucket.
 - Filters files based on the provided suffix (e.g., .progress for progress files or .tgz for media files).
 - If Delimiter: '/' is specified, it lists directories instead of files.
- **Usage:** Used for listing airlines, fleets, subfleets, or retrieving .progress or media files.
- **Key Dependencies:** ListObjectsV2Command

3. listAllAirlines()

- **Purpose:** Retrieves all unique airline directories from the Status bucket.
- **How it works:**
 - Calls `listFilesInBucket` with an empty prefix and delimiter `/`.
 - Extracts airline names by removing trailing slashes from the directory names.
- **Usage:** Provides the top-level structure for the airline hierarchy.

4. `listAllFleets(airline)`

- **Purpose:** Fetches all fleets under a specific airline in the Status bucket.
- **How it works:**
 - Calls `listFilesInBucket` with the prefix `{airline}/` to limit results to the specified airline's directory.
 - Extracts fleet names from the resulting directory structure.
- **Usage:** Used to retrieve fleet data for an airline.

5. `listAllSubfleets(airline, fleet)`

- **Purpose:** Retrieves all subfleets for a given airline and fleet, along with aggregated progress and status information.
- **How it works:**
 - Calls `listFilesInBucket` with the prefix `{airline}/{fleet}/`.
 - For each subfleet directory:
 - Retrieves all tail IDs using `listAllTailIDs`.
 - Fetches progress data for each tail ID using `getProgressData`.
 - Aggregates data such as the total number of tail IDs, completed tail IDs, in-progress tail IDs, and overall completion percentage.

- Returns subfleet data including progress metrics.
- **Usage:** Provides detailed subfleet-level progress and status information.

6. listAllTailIDs(airline, fleet, subfleet)

- **Purpose:** Fetches all unique tail IDs for a specific subfleet.
- **How it works:**
 - Calls listFilesInBucket with the prefix {airline}/{fleet}/{subfleet}/.
 - Extracts tail IDs from the directory structure by splitting the path.
- **Usage:** Provides identifiers for aircraft under a specific subfleet.

7. listAllFilesInTailID(airline, fleet, subfleet, tailID)

- **Purpose:** Lists all files in a specified tail ID directory.
- **How it works:**
 - Calls listFilesInBucket with the prefix {airline}/{fleet}/{subfleet}/{tailID}/.
 - Returns file metadata including key, size, and last modified date.
- **Usage:** Retrieves all files stored for a specific aircraft in the Status bucket.

8. getProgressData(airline, fleet, subfleet, tailID)

- **Purpose:** Calculates the download progress for content files associated with a specific tail ID.
- **How it works:**
 - Fetches .progress files for the specified tail ID using listFilesInDirectory.
 - Reads each .progress file to determine the number of bytes downloaded using readFileFromBucket.
 - Fetches corresponding media files from a fixed path in the Media bucket.

- Matches .progress files with media files to calculate download percentages and statuses (e.g., Completed or In progress).
- Aggregates and returns progress data for each content item.
- **Usage:** Displays real-time progress for content downloads on the dashboard.

9. readFileFromBucket(bucketName, key)

- **Purpose:** Reads the content of a specific file from an S3 bucket.
- **How it works:**
 - Sends a GetObjectCommand request to fetch the file.
 - Converts the file stream into a string using the streamToString helper function.
 - Parses the content (e.g., bytes downloaded) into the appropriate format.
- **Usage:** Used to read .progress files for determining download progress.

10. calculateProgress(progressFiles, mediaFiles)

- **Purpose:** Matches .progress files with their corresponding media files to compute download progress.
- **How it works:**
 - Iterates through the list of .progress files.
 - Matches each .progress file with a corresponding media file by comparing content names.
 - Reads the number of bytes downloaded from the .progress file and compares it to the media file's size.

- Computes the progress percentage and determines the status (Completed or In progress).
- **Usage:** Used by getProgressData to aggregate progress metrics for a tail ID.