Green IT Hackathon - Food Waste Management

Comprehensive Guide to Setting Up the Backend of the Application

Objective

The backend is the backbone of the application, handling data processing, storage, authentication, and communication between the mobile and web apps. It should be designed in a manner such that it is scalable, secure, and efficient to handle potentially large volumes of data and user interactions. This guideline will assist in setting up, configuring, and understanding the technologies required for the backend portion of this hackathon.

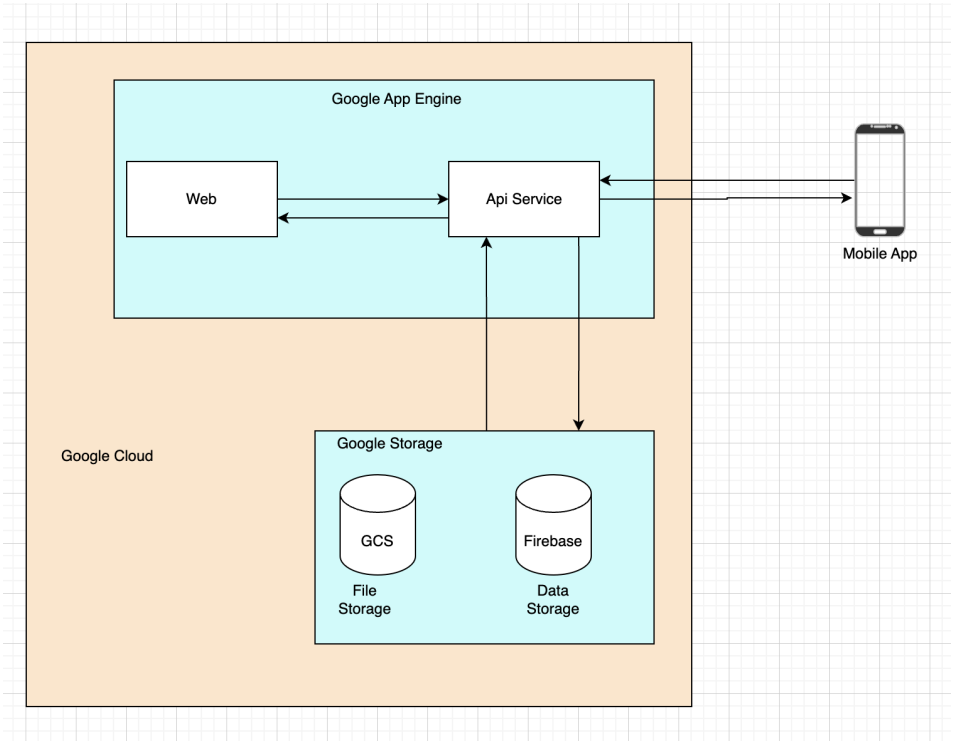
Components of the Backend

Component	Description
API Service	Exposes endpoints for the mobile and web apps to interact with the backend.
Database	Stores user data, food waste records, recipes, tips, and other relevant information.
Authentication	Manages user registration, login, and authorization.
Data Processing	Handles business logic, data validation, and processing.

Cloud Provider	Ensures scalability, reliability, and provides additional services like serverless computing, storage, and security.

Technology Stack

The diagram below shows an overview of the cloud infrastructure. We will be using Google cloud and the two main services will be Google App Engine for deploying the web frontend and the API and Google Storage for handling data. Firebase will be used as the main database and GCS will be used for saving photos, pdfs, videos and other types of files.



Cloud Provider (Google Cloud)

We will be using Google Cloud because it has excellent machine learning and data analytics services, competitive pricing and a slightly easier learning curve than AWS and AZURE for beginners.

Google also offers $300 USD credits to use on any of their services for newly created accounts. It has been verified with Google support that this offer will still be valid in August, in Malaysia.

API Frameworks

In the table below, we provide a list of popular frameworks that can be used to build the API service. You are free to use any framework of your choice however we recommend using express.js (node.js) for its simplicity, speed, and extensive middleware options. Our tutorial will be focused on express.js.

Framework	Description
Express.JS (Node.JS)	Lightweight, fast, large community, easy to get started
Django (Python)	Batteries included framework, built in admin panel, ORM
SpringBoot (Java)	Enterprise grade, robust ecosystem, strong security

Authentication

We will be using **Firebase** authentication. It is easy to implement and supports various authentication methods. The tutorial will show how to integrate Firebase Authentication into an Express.js backend.

Tutorial

Google Cloud Account Set Up

Create a New Project

1. Go to the Google Cloud Console and sign in.
2. Click on "Select a project" at the top and then click on "New Project."
3. Choose a unique project name and select the desired organization if applicable.
4. Once the project is created, note down the Project ID for future reference.
5. Link this project to an appropriate Billing Account.

Enable Required APIs

1. Navigate to the APIs & Services section in the Google Cloud Console.
2. Click on Library in the left sidebar.
3. Search for Cloud Storage and enable the Cloud Storage API.
4. Search for App Engine Admin and enable the App Engine Admin API.

Set Up Cloud Storage

1. Go to the Cloud Storage section in the Google Cloud Console.

2. Click on Create Bucket to make a new bucket.
3. Choose a unique name and select the desired region.
4. Leave other settings as default and create the bucket.

Install GCP CLI

1. Navigate to the root directory of your Node app in the terminal.
2. Install the Google Cloud SDK following the instructions at Google Cloud SDK Installation Guide.
3. Authenticate the Google Cloud SDK by running **'gcloud auth login'** and following the on-screen instructions.
4. Set the project ID by running **'gcloud config set project PROJECT_ID'**, replacing **'PROJECT_ID'** with your actual Project ID.

Set-Up Service Account

1. Go to the Google Cloud Console.
2. Navigate to IAM & Admin → Service Accounts.
3. Click on Create Service Account.
4. Provide an appropriate name and description for the service account. For instance, use github-ci-cd as it will be utilized for Github CI/CD.
5. Assign the following roles:
    - App Engine Admin
    - Cloud Build Service Account
    - Service Account User
    - Storage Object Admin
6. Click the three dots and select Manage keys.
7. Click on Add Key → Create New Key.
8. Choose the JSON key type and securely download it. Remember, this key grants access to Google Cloud resources, so keep it safe.

Create API Service

Install Node JS

We recommend using NVM (Node Version Manager). It makes it easy for managing multiple Node.js versions on the same machine. It works across macOS, Linux, and Windows (with nvm-windows).

Mac and Linux

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
```

```
source ~/.nvm/nvm.sh
nvm install node
```

Windows

Step 1: Download **'nvm-setup.zip'** from the nvm-windows releases page.
Step 2: Extract the contents and run the nvm-setup.exe installer.
Step 3: Install Node.js:

```
nvm install latest
nvm use latest
```

Verify Installation (for all machine types)

```
node -v
npm -v
```

Create Express.js NodeJS Backend

Initialize a new nodejs project

Create a folder for your project - for this tutorial we will call it waste-backend and initialize a new node application.

```
mkdir waste-backend
cd waste-backend
npm init -y
```

Install Initial Dependencies

Install the basic dependencies to get started with your node js project

```
npm install express axios cors body-parser
```

- **Express.js**, or simply **Express**, is a back end web application framework for building RESTful APIs with Node.js
- **Axios** is a node js client that helps you make http requests. You will be using this to send information to and from the frontend
- **Cors** is It is used to enable cross-origin requests in web applications. This is essential for APIs that are accessed from a different domain than the one serving the client application.

- **Body Parser** It is used to parse JSON, raw, text, and URL-encoded form data submitted in HTTP requests. This is essential for handling form submissions and JSON payloads in your APIs.

Version Control

Using GitHub

Create GItHub account or use an existing one.

Initialize Git in API Service

```
git remote add origin https://<GITHUB_URL>/waste-backend.git
git branch -M main
git push -u origin main
```

Please remember to replace **<GITHUB_URL>** with your actual url

Integrating Firebase Authentication WIth Our API Service

Install Firebase Admin SDK

Install the firebase-admin plugin from npm

```
npm install firebase-admin
```

Initialize Firebase Admin SDK

Create a file named auth.js in your project's root directory and enter the code below:

```
const admin = require('firebase-admin');
const serviceAccount = require('./path/to/serviceAccountKey.json');

admin.initializeApp({
    credential: admin.credential.cert(serviceAccount),
});

async function verifyToken(token) {
    try {
```

```
        const decodedToken = await admin.auth().verifyIdToken(token);
        return decodedToken;
    } catch (error) {
        throw new Error('Authentication failed');
    }
}

module.exports = { verifyToken };
```

Creating Express.js Server with Authentication Endpoints

Create a file named index.js in your projects root directory and enter the code below:

```
const express = require('express');
const { verifyToken } = require('./auth');
const admin = require('firebase-admin');

const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.json());

// Endpoint for user registration
app.post('/register', async (req, res) => {
    const { email, password } = req.body;
    try {
        const user = await admin.auth().createUser({
           email,
           password,
        });
        res.status(201).send(user);
    } catch (error) {
        res.status(400).send(error.message);
    }
});

// Endpoint for user login (generates custom token)
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
```

```
    try {
        const user = await admin.auth().getUserByEmail(email);
        // Normally, you would verify the password here
        // Since Firebase Admin SDK does not handle password
verification, this is just for demonstration
       const token = await admin.auth().createCustomToken(user.uid);
       res.status(200).send({ token });
    } catch (error) {
        res.status(400).send(error.message);
    }
});

// Middleware to protect routes
app.use(async (req, res, next) => {
    const token = req.headers.authorization?.split(' ')[1];
    if (token) {
        try {
            req.user = await verifyToken(token);
            next();
        } catch (error) {
            res.status(401).send('Unauthorized');
        }
    } else {
        res.status(401).send('Unauthorized');
    }
});

// Protected route example
app.get('/profile', (req, res) => {
    res.send(`Hello ${req.user.email}`);
});

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

This will allow you to create endpoints that can be used by the frontend to send or access data. The following URLs can be used to access the different endpoints.

- ❖ Additional resources on how to use firebase authentication with your backend and frontend can be found in the *Additional Resources* section.

Deploying API Service to Google Cloud

Create app.yaml

To deploy the app we need to create a configuration file in our project's root directory. This file has to be a yaml file as per Google's recommendation. For the purpose of this tutorial we will create a file called app.yaml in our root directory and add the settings and routing for Google App Engine.

1. The app.yaml file configures settings and routing for Google App Engine applications.
2. Keep the app.yaml in your project's root directory alongside your source code.

```
# [START app_yaml]

runtime: nodejs20

service: node-express-api

# [END app_yaml]
```

Explanation of Configuration:

3. runtime: Specifies the runtime environment (e.g., nodejs20)
4. service: Defines the service name, typically a project-specific prefix or subdomain.

Deploy your App Locally

Deploy your Node app by executing the command in root directory of your project

```
gcloud app deploy
```

Additional Resources

Title	Link

How to deploy Node JS to Google Cloud	https://dev.to/rushi-patel/deploy-node-js-project-to-google-app-engine-with-github-actions-cicd-a-complete-guide-3od9
Google Cloud tutorial	Cloud Computing, Hosting Services, and APIs | Google Cloud
Python FastAPI tutorial	https://realpython.com/fastapi-python-web-apis/
Integrating cloud SQL into your Node JS app	GitHub - GoogleCloudPlatform/cloud-sql-nodejs-connector: A JavaScript library for connecting securely to your Cloud SQL instances
Getting started with NodeJS, Google Cloud, Firebase and Cloud Storage	https://cloud.google.com/nodejs/getting-started

Partners

- Lappeenranta-Lahti University of Technology
- Sunway University