JIMMA INSTITUTE OF TECHNOLOGY

DEPARTMENT OF SOFTWARE ENGINEERING

FACULTY OF COMPUTING AND INFORMATICS

Elective I- WEB Service Assignment-2

GROUP MEMBERS	ID
1. Jida Guta	RU1644/14
2. Gemeda Tamiru	RU0227/14
3. Sintayehu Bikila	RU0207/14
4. Sena Kebede	RU0191/14
5. Meron Mulu	RU0477/14
6. Bekalu Endirias	RU2501/13

Submitted to: Mr Dessalew Y.

November, 2025

Table of Contents

1. Abstract 1
2. Introduction 2
3. Implementation Steps 2
    3.1 Web Service Integration 2
    3.2 API Authentication and Authorization 3
    3.3 Data Transformation & Normalization 4
    3.4 Error Handling and Logging 4
4. Tools and Technologies Used 5
5. Results and Observations 6
6. Conclusion 7
7. References 7
8. Submission Requirements 8

1. Abstract

This report presents the complete analysis and implementation of a News Aggregation System that integrates external web services to fetch real-time news updates. The system retrieves data from three distinct RESTful APIs NewsAPI.org, The Guardian Open Platform, and NewsData.io each providing news articles in JSON format. To ensure secure interactions, the application employs API key authentication, proper request configuration, and structured error handling. Data from each API is then transformed and normalized to provide a unified and user-friendly reading experience. This document explains the implementation steps, tools used, results obtained, and conclusions drawn during the project.

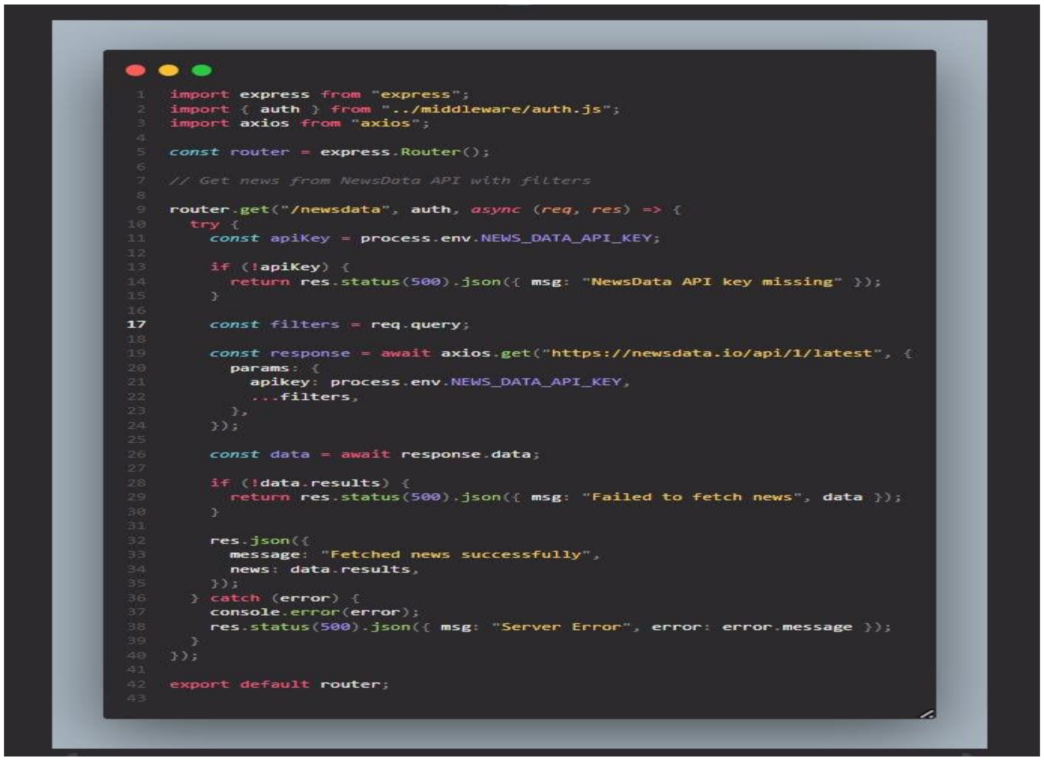
2. Introduction

This report presents the implementation of a web service integration project focused on securely consuming external APIs. The assignment demonstrates how a client application retrieves data from an external RESTful API, applies authentication, performs data transformation, and implements robust error handling and logging. The primary objective is to understand practical techniques for integrating web services and ensuring secure and reliable communication.

3. Implementation Steps

3.1 Web Service Integration

The application integrates three widely used news APIs to gather headlines and articles: NewsAPI.org, The Guardian API, and NewsData.io. Each service provides structured JSON responses containing article titles, descriptions, publication dates, and other related metadata. A dedicated configuration class (ApiConfig) manages the base URLs, endpoints, and API keys for all integrated services. The application sends HTTP GET requests using these URLs, retrieves the data, and forwards it to the processing layer.

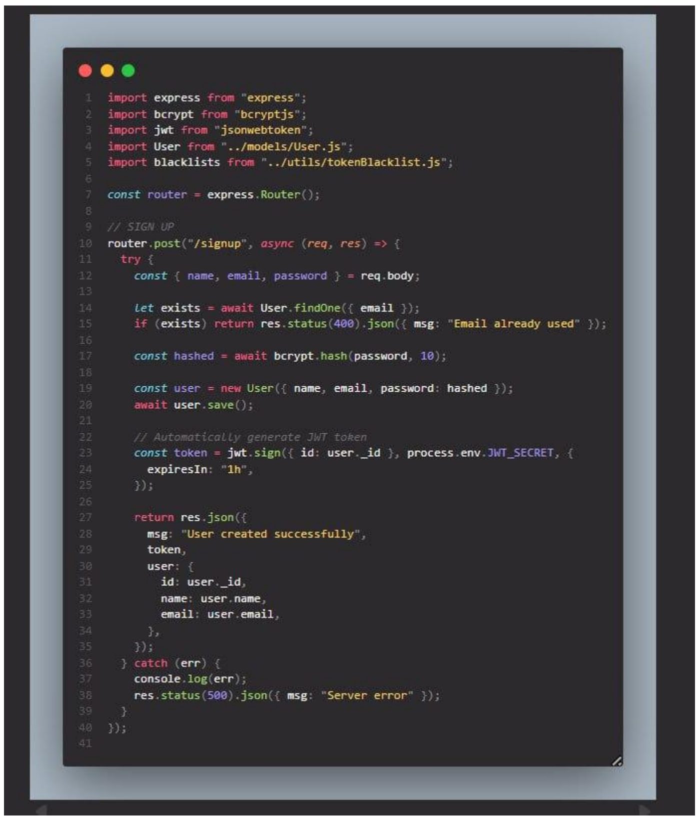


```javascript
import express from "express";
import { auth } from "../middleware/auth.js";
import axios from "axios";

const router = express.Router();

// Get news from NewsData API with filters

router.get("/newsdata", auth, async (req, res) => {
  try {
    const apiKey = process.env.NEWS_DATA_API_KEY;

    if (!apiKey) {
      return res.status(500).json({ msg: "NewsData API key missing" });
    }

    const filters = req.query;

    const response = await axios.get("https://newsdata.io/api/1/latest", {
      params: {
        apikey: process.env.NEWS_DATA_API_KEY,
        ...filters,
      },
    });

    const data = await response.data;

    if (!data.results) {
      return res.status(500).json({ msg: "Failed to fetch news", data });
    }

    res.json({
      message: "Fetched news successfully",
      news: data.results,
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ msg: "Server Error", error: error.message });
  }
});

export default router;
```

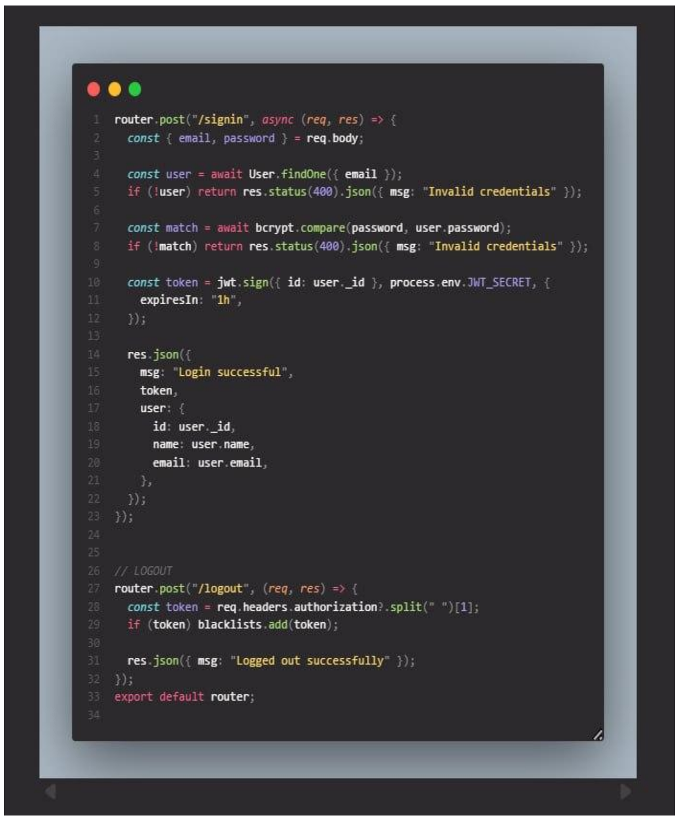
3.2 API Authentication and Authorization

All the APIs integrated in this project require API key authentication to authorize data access. However, each API expects the key to be provided in a slightly different way. NewsAPI.org expects the key to be sent through request headers, where the key is attached using the X-Api-Key field. In contrast, The Guardian API and the NewsData.io API expect the key to be added as a query parameter attached to the URL.

Before making any requests, the system verifies the validity of the API keys using Boolean checks available in the ApiConfig class. This prevents unauthorized or malformed requests from being executed. If any key is missing, invalid, or too short, the application disables access to that specific API and logs an appropriate warning message.



```javascript
import express from "express";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
import User from "../models/User.js";
import blacklists from "../utils/tokenBlacklist.js";

const router = express.Router();

// SIGN UP
router.post("/signup", async (req, res) => {
  try {
    const { name, email, password } = req.body;

    let exists = await User.findOne({ email });
    if (exists) return res.status(400).json({ msg: "Email already used" });

    const hashed = await bcrypt.hash(password, 10);

    const user = new User({ name, email, password: hashed });
    await user.save();

    // Automatically generate JWT token
    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
      expiresIn: "1h",
    });

    return res.json({
      msg: "User created successfully",
      token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
      },
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({ msg: "Server error" });
  }
});
```



```javascript
router.post("/signin", async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({ email });
  if (!user) return res.status(400).json({ msg: "Invalid credentials" });

  const match = await bcrypt.compare(password, user.password);
  if (!match) return res.status(400).json({ msg: "Invalid credentials" });

  const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
    expiresIn: "1h",
  });

  res.json({
    msg: "Login successful",
    token,
    user: {
      id: user._id,
      name: user.name,
      email: user.email,
    },
  });
});


// LOGOUT
router.post("/logout", (req, res) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (token) blacklists.add(token);

  res.json({ msg: "Logged out successfully" });
});
export default router;
```

3.3 Data Transformation & Normalization

Each of the three APIs returns data in a different JSON structure. To present a consistent user experience, the application parses each response and extracts the essential fields, such as the title, description, publishing time, URL, and the article source. After this, the extracted information is transformed into a unified article model, ensuring users see a clean and standardized list regardless of which API provided the data.

The transformed articles are then displayed in the Flutter UI. Each news article appears as a card or list item containing the headline, summary, time of publication, and a link to read the full article.

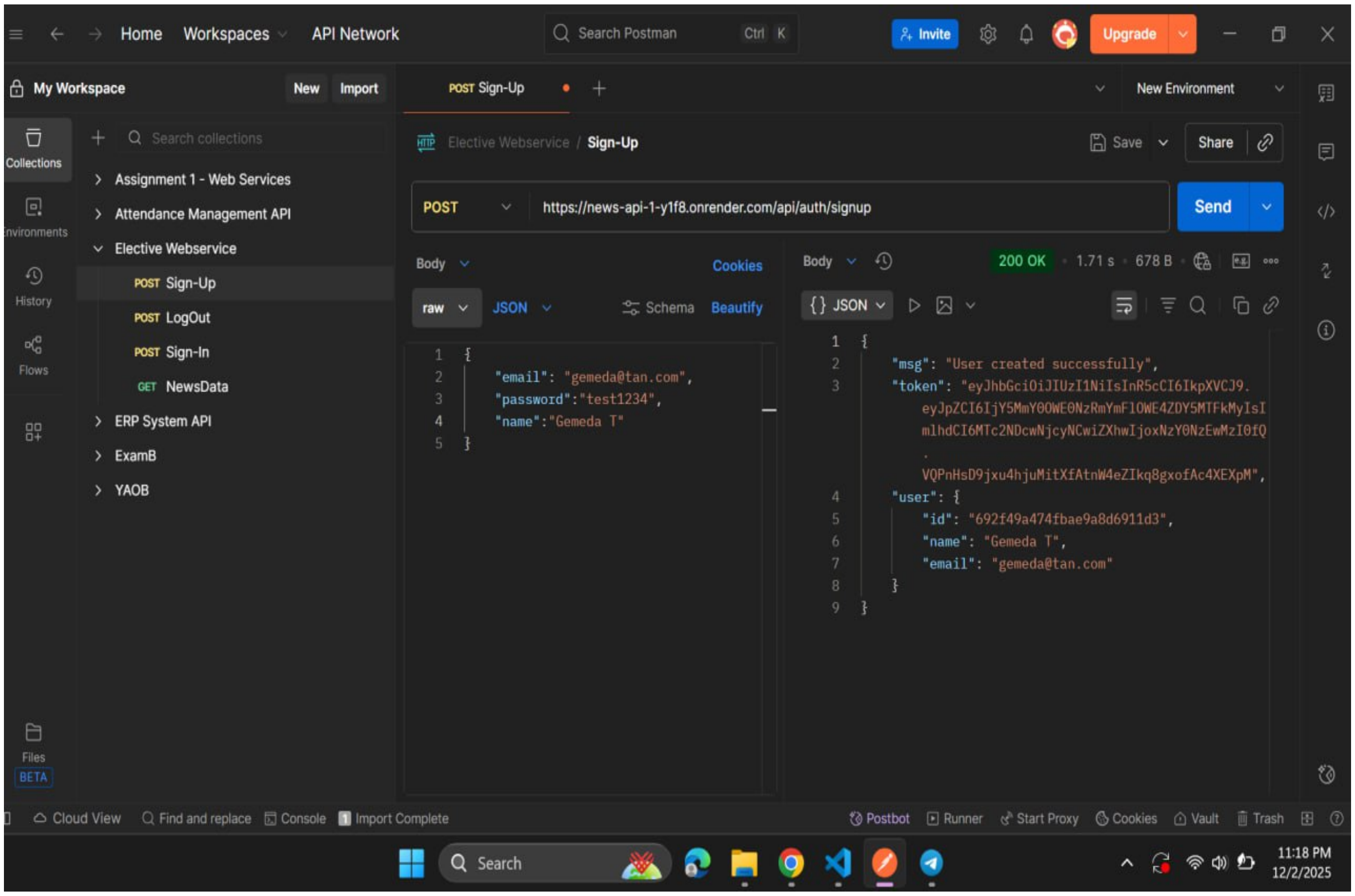
3.4 Error Handling and Logging

A solid error-handling system is implemented to manage issues that may occur during API communication. These issues include invalid API keys, network interruptions, request timeouts, rate limiting, and unexpected server responses such as malformed JSON. The application uses try catch blocks to capture these errors and provides descriptive log messages to help developers identify the source of problems.

All errors and warnings are recorded with time stamps, key previews, and status codes to make debugging easier. Request timeouts are also managed through predefined values in the `ApiConfig` class, ensuring that the application does not hang when network conditions are poor

4. Tools and Technologies Used

The project makes use of several tools and technologies to implement the required functionality. Flutter and Dart are used to build the interface and handle the logic, while packages such as Dio or the HTTP client handle API communication. Postman and URL were used during development to test endpoints and verify the format of the returned JSON. Git and GitHub were used for version control, while Android Studio or Visual Studio Code served as the development environment.

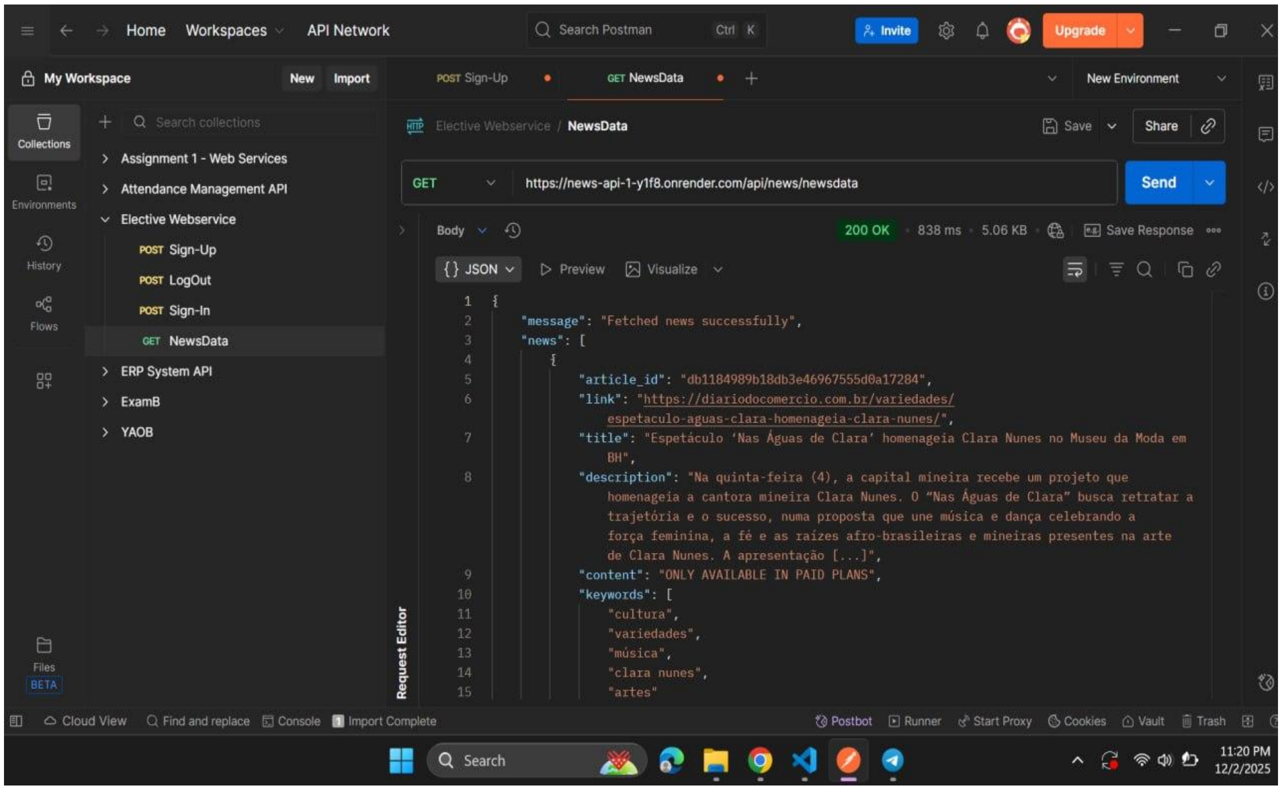


POST https://news-api-1-y1f8.onrender.com/api/auth/signup

```
{
    "email": "gemeda@tan.com",
    "password":"test1234",
    "name":"Gemeda T"
}
```

200 OK · 1.71 s · 678 B

```
{
    "msg": "User created successfully",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6IjY5MmY0OWE0NzRmYmFlOWE4ZDY5MTFkMyIsI
        mlhdCI6MTc2NDcwNjcyNCwiZXhwIjoxNzY0NzEwMzI0fQ
        .
        VQPnHsD9jxu4hjuMitXfAtnW4eZIkq8gxofAc4XEXpM",
    "user": {
        "id": "692f49a474fbae9a8d6911d3",
        "name": "Gemeda T",
        "email": "gemeda@tan.com"
    }
}
```

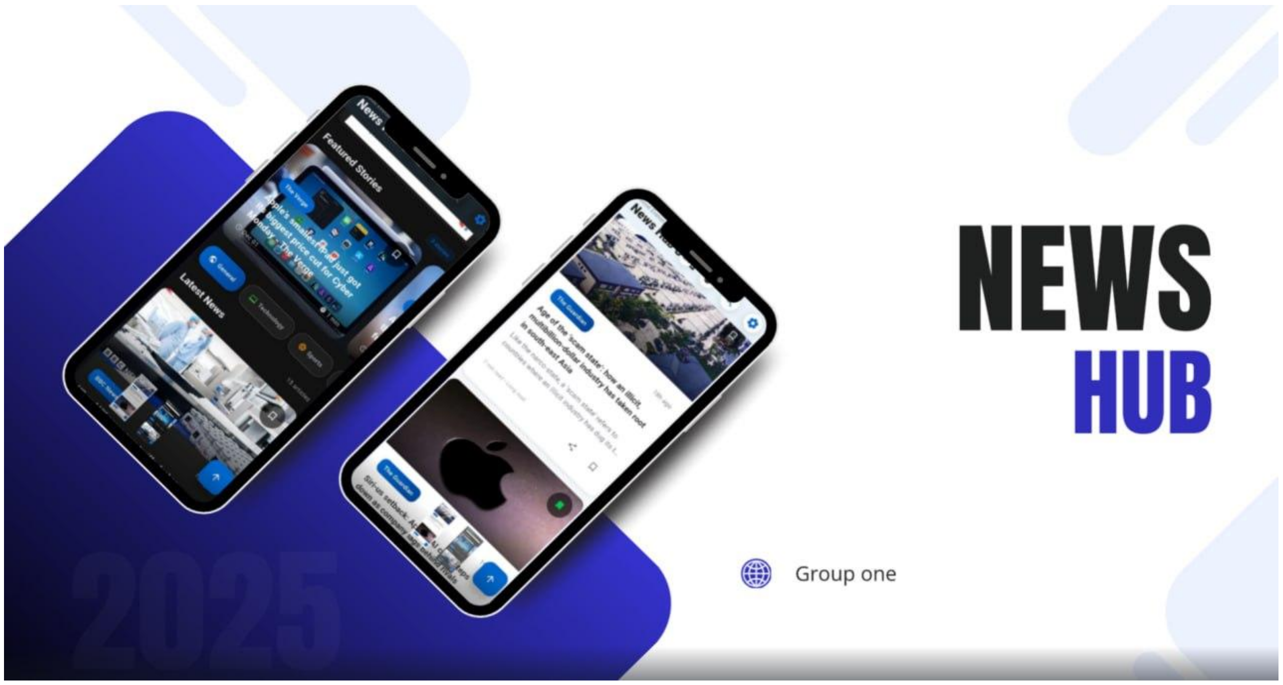


```json
{
    "message": "Fetched news successfully",
    "news": [
        {
            "article_id": "db1184989b18db3e46967555d0a17284",
            "link": "https://diariodocomercio.com.br/variedades/
                espetaculo-aguas-clara-homenageia-clara-nunes/",
            "title": "Espetáculo 'Nas Águas de Clara' homenageia Clara Nunes no Museu da Moda em
                BH",
            "description": "Na quinta-feira (4), a capital mineira recebe um projeto que
                homenageia a cantora mineira Clara Nunes. O “Nas Águas de Clara” busca retratar a
                trajetória e o sucesso, numa proposta que une música e dança celebrando a
                força feminina, a fé e as raízes afro-brasileiras e mineiras presentes na arte
                de Clara Nunes. A apresentação [...]",
            "content": "ONLY AVAILABLE IN PAID PLANS",
            "keywords": [
                "cultura",
                "variedades",
                "música",
                "clara nunes",
                "artes"
```

5. Results and Observations

The final application successfully fetched and displayed news from all three integrated APIs. NewsAPI.org provided breaking headlines, while The Guardian API returned detailed metadata about each article. NewsData.io proved useful for global coverage by aggregating articles from multiple sources. Authentication behaved as expected, with valid keys granting access and invalid keys triggering error logs.

After data was fetched, the transformation and normalization process ensured a consistent and readable article format in the user interface. The application remained stable even under unstable network conditions, thanks to the robust error-handling system. Rate limit restrictions were logged properly, allowing the developer to understand the request limits of each API.



6. Conclusion

This project demonstrated how a modern application can effectively integrate multiple web services to enrich user experience. By combining three external APIs, the application provided broader news coverage and improved reliability. The assignment strengthened key skills such as secure API communication, JSON parsing, error handling, and API integration. These skills are essential for building scalable applications that rely on cloud-based services.

7. References

1. **NewsAPI.** (n.d.). *NewsAPI Documentation*. Retrieved from https://newsapi.org
2. **The Guardian.** (n.d.). *Open Platform Documentation*. Retrieved from https://open-platform.theguardian.com
3. **NewsData.io.** (n.d.). *NewsData API Documentation*. Retrieved from https://newsdata.io
4. **Dart/Flutter Team.** (n.d.). *Dart HTTP & Dio Package Documentation*. Retrieved from the official Dart and pub.dev documentation pages.
5. **GitHub.** (n.d.). *GitHub Documentation*. Retrieved from the official GitHub Docs.

8. Submission Requirements

The source code for this project has been uploaded to GitHub with clear commit messages that describe each change made during development. The PDF version of this report will be included in the final submission, along with screenshots of API responses and application output.