Distributed Systems Assignment 1 Report **Kemal Yılmaz 20239189**

Motivation of Design Decisions:

The design of the API prioritises simplicity, readability, and scalability. The choice of Flask and Flask-RESTful frameworks aligns with the need for building RESTful APIs in Python. The structure of the code follows best practices, separating concerns into different classes and methods, enhancing maintainability and readability.

The decision to use argparse for taking API keys as command-line arguments ensures flexibility and security. This way, sensitive information like API keys isn't hard-coded within the codebase. Additionally, the OOP approach for defining API endpoints makes the code modular and easier to extend in the future.

RESTfulness of the API:

The API adheres to RESTful principles by utilising HTTP methods appropriately and maintaining statelessness. Each endpoint represents a resource, and the interactions with these resources are handled through standard HTTP verbs (GET, POST, DELETE). Uniform resource identifiers (URIs) are meaningful and hierarchical, contributing to the clarity and organisation of the API.

For instance, endpoints like `/countries`, `/country/<country_name>`, `/weather/current/<country_name>`, and `/favourites` represent distinct resources and perform specific actions related to those resources. The API responses are consistent and include appropriate HTTP status codes, enhancing the overall RESTfulness.

Efficiency Considerations:

In the development of this API, utility and readability were prioritised over efficiency. While efficiency is important, the emphasis was placed on creating a user-friendly and understandable codebase. This decision was reflected in two key areas:

Abundance of Try-Except Blocks:
The code includes numerous try-except blocks to handle exceptions and errors gracefully. While this approach ensures robustness and fault tolerance, it may introduce some overhead in terms of performance. However, the primary goal was to prioritise reliability and prevent unexpected crashes or disruptions in service. Each try-except block is carefully crafted to provide informative error messages and guide users in troubleshooting issues, contributing to the overall usability of the API.

Extra Data Manipulation for Readability:
Additional data manipulation steps were incorporated throughout the code to enhance readability and comprehension. For example, the extraction of common names from the response of the REST Countries API increases the clarity of country listings for users. Similarly, the modification of response data in the CountryDetails endpoint improves the presentation of country details in a more structured and understandable format. While these

manipulations may add some computational overhead, they significantly improve the user experience by making the API responses more intuitive and informative.

By prioritising utility and readability, the API aims to provide a seamless and enjoyable experience for developers and end-users alike. While efficiency remains a consideration, it is balanced against the overarching goals of accessibility and ease of use. Future optimizations can be explored to enhance performance without compromising the clarity and usability of the API.

Handling Faulty Requests:

The API includes error handling mechanisms to deal with faulty requests:

1. **Exception Handling:** Faulty requests or unexpected errors are caught and appropriately handled using try-except blocks. Responses sent relevant HTTP status code 500 for internal server errors.
2. **Request Validation:** The API uses reqparse to validate incoming requests and ensure that required parameters are provided. This prevents potential errors due to missing or incorrect input data.

For example, in the `WeatherForecast` endpoint, the API expects the `days` parameter to be provided with the request, ensuring that clients cannot make faulty requests without specifying the number of days for the forecast.

Time Spent on the Assignment:

Assignment took me around 20 hours to complete. I spent the first 12 hours creating the api and testing it throughout the way. After that another 5 hours were spent for testing and optimising for the best performance. Last 3 hours were spent on creating the necessary documentation and reports for the assignment.

Design Considerations Not Implemented:

While the current design fulfills the basic requirements of the assignment, there is an additional feature that could be considered for future iterations:

1. **Duplicate Checking in Favourite List**:

Implementing a mechanism to check for duplicates when adding a country to the favourites list. This would prevent redundancy and ensure data integrity within the database. Duplicate checking could be based on country names or unique identifiers associated with each country.

This feature was not implemented in the current version due to time constraints and the focus on core functionality. However, it represents valuable additions for enhancing security, scalability, and usability in real-world scenarios.