Software Design Document

for

Uncover the Rubik's Cube

Project-Team 2

Chandana Chenchula Nandan
Shanthakumar Sivakumar
Xinyu Yang
Yuan Gao
Yuchen Zheng

Boston University - Software Engineering

Date - 31 October 2023

Table of Contents

1. Overview of "Uncover the Rubik's Cube" Project

1.1 Scope:

The "Uncover the Rubik's Cube" project aims to provide a comprehensive and user-friendly platform for solving and simulating the Rubik's Cube puzzle. This project's scope encompasses various components, including a 3D interface for interactive cube manipulation, an efficient solving algorithm, and features for user guidance and convenience. The project also addresses the persistence of user progress and customization options, making it a versatile tool for both novice and experienced Rubik's Cube enthusiasts.

1.2 Purpose:

The primary purpose of this project is to enhance the user experience of solving the Rubik's Cube by making the entire process more intuitive and convenient. Key purposes include:

1. Optimized Solution Steps: Providing users with the most concise and optimal solution steps to solve the Rubik's Cube efficiently.

2. Integration of Memory Functionality: Implementing a memory function within the application, allowing users to retain their current Rubik's Cube recovery progress, even when incomplete.

3. Intuitive User Interface: Creating a visually engaging and user-friendly design to enhance user interaction and navigation throughout the Rubik's Cube solving process.

4. Enhanced User Guidance: Implementing interactive tutorials and hints within the application to provide users with valuable insights and improve their problem-solving skills and strategies.

5. Expanded Platform Compatibility: Expanding the application's compatibility across various platforms, ensuring that users can enjoy the Rubik's Cube-solving experience seamlessly, whether on web browsers,mobile devices, or desktop applications.

1.3 Target Audience:

1. Rubik's Cube Enthusiasts: This project caters to individuals passionate about solving the Rubik's Cube, whether they are beginners or advanced solvers looking for optimized solutions.

2. Educational Institutions: Teachers and students interested in teaching or learning about algorithms, patterns, and strategies for solving the Rubik's Cube.

3. Casual Gamers: Gamers who enjoy solving puzzles and are looking for an interactive and entertaining experience.

4. General Puzzle Enthusiasts: Anyone interested in logic puzzles and challenges that require problem-solving skills.

5. Online Community: Enthusiasts who are part of an online community dedicated to Rubik's Cube solving and are seeking a digital tool to assist in their hobby.

The project aims to provide a valuable and engaging experience for this diverse audience, offering an intuitive platform to learn, practice, and enjoy solving the Rubik's Cube.

2.Functional description

The application offers a comprehensive set of features to facilitate Rubik's Cube restoration for users.

I. Rubik’s Cube Simulation: Users can interact with a 3D simulation of a Rubik’s Cube, rotating specific layers through intuitive mouse controls. Highlighting specific cubes and using arrow buttons enables precise layer rotation.
II. Hint Function: When users are unsure about the next step, the hint function provides guidance, suggesting specific layer rotations to progress toward

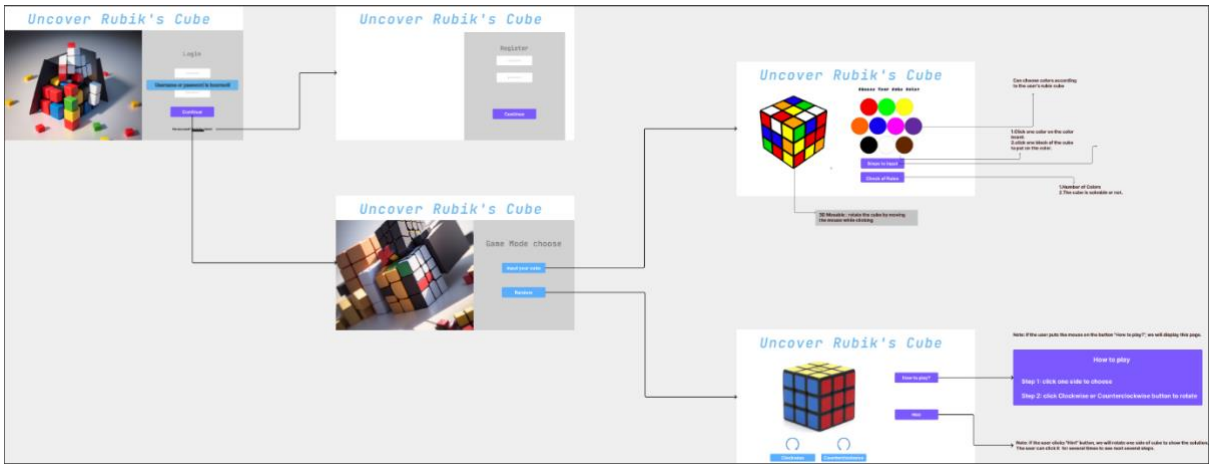
Rubik’s Cube recovery. The application automatically executes the suggested moves, guiding users efficiently.

III. Reset Function: Users can instantly reset the Rubik’s Cube to its initial state, allowing them to restart the recovery process with a single click.

IV. Randomly Scramble: The application generates random Rubik’s Cube configurations, eliminating the need for manual scrambling. Each click on the "randomly scramble" button produces a new, unique initial state for users to solve.

V. User History: Users can save their current Rubik’s Cube state. Upon returning to the application, the saved state is restored, enabling seamless continuation of the recovery process.

3. User interface

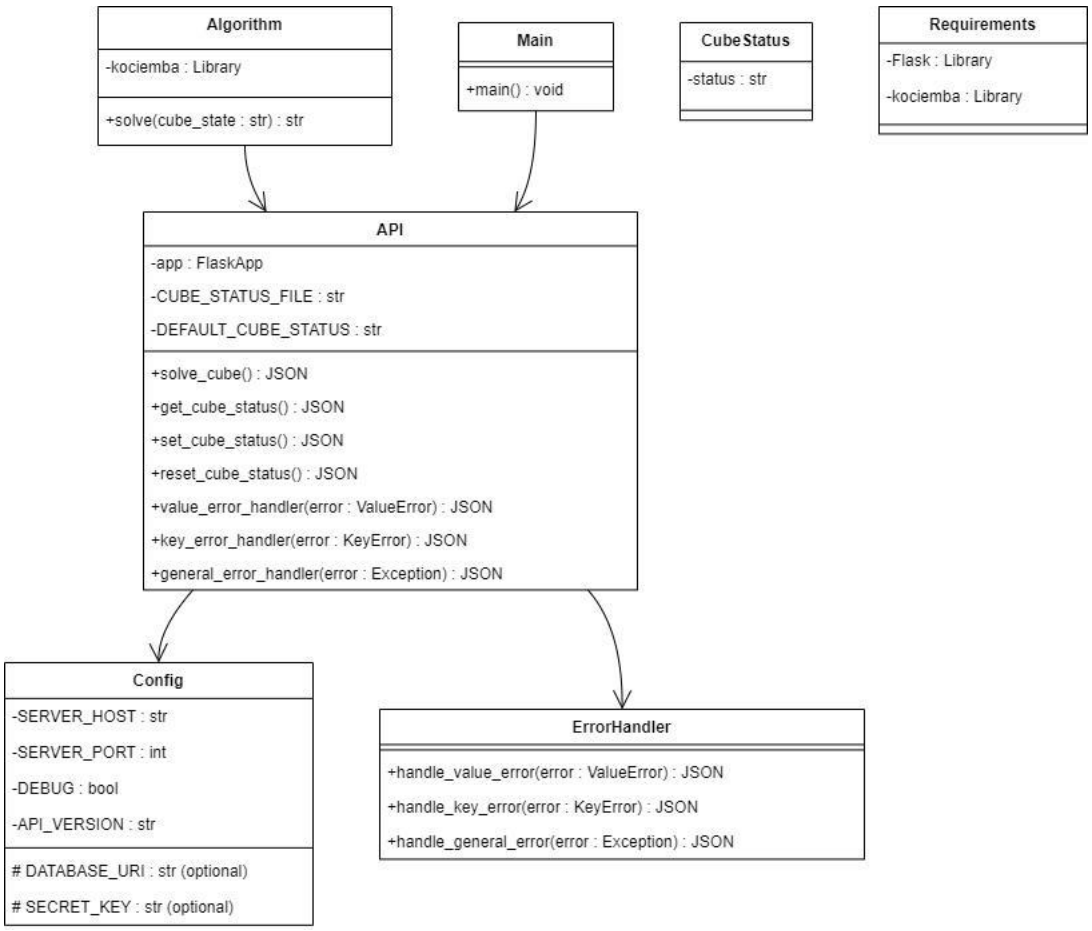


I. Home Page: The central focus is a 3D Rubik’s Cube simulation. The top menu bar includes the project logo and essential options. A "Play Game" button on the right side of the Rubik’s Cube grants access to the game page.

II. Game Page: Features a central interactive Rubik’s Cube. Functional buttons for hint, reset, and random scramble are located in the upper right corner, with arrow buttons for layer rotation placed in the lower right corner.

III. Rules & Regulations Page: Contains information about the application's guidelines and usage regulations.
IV. Login Page: Allows users to log into their accounts.
V. Register Page: Enables new users to create accounts.
VI. Progress Tracking Page: Provides insights into users' recovery progress.

4. UML Diagram



Algorithm: This class contains the Kociemba library and a function called solve that returns a solution based on the provided Rubik's Cube state.

API: Contains methods for handling API requests. This class is associated with the Algorithm class because it uses the solve method.

Config: This class represents the configuration settings for your application.

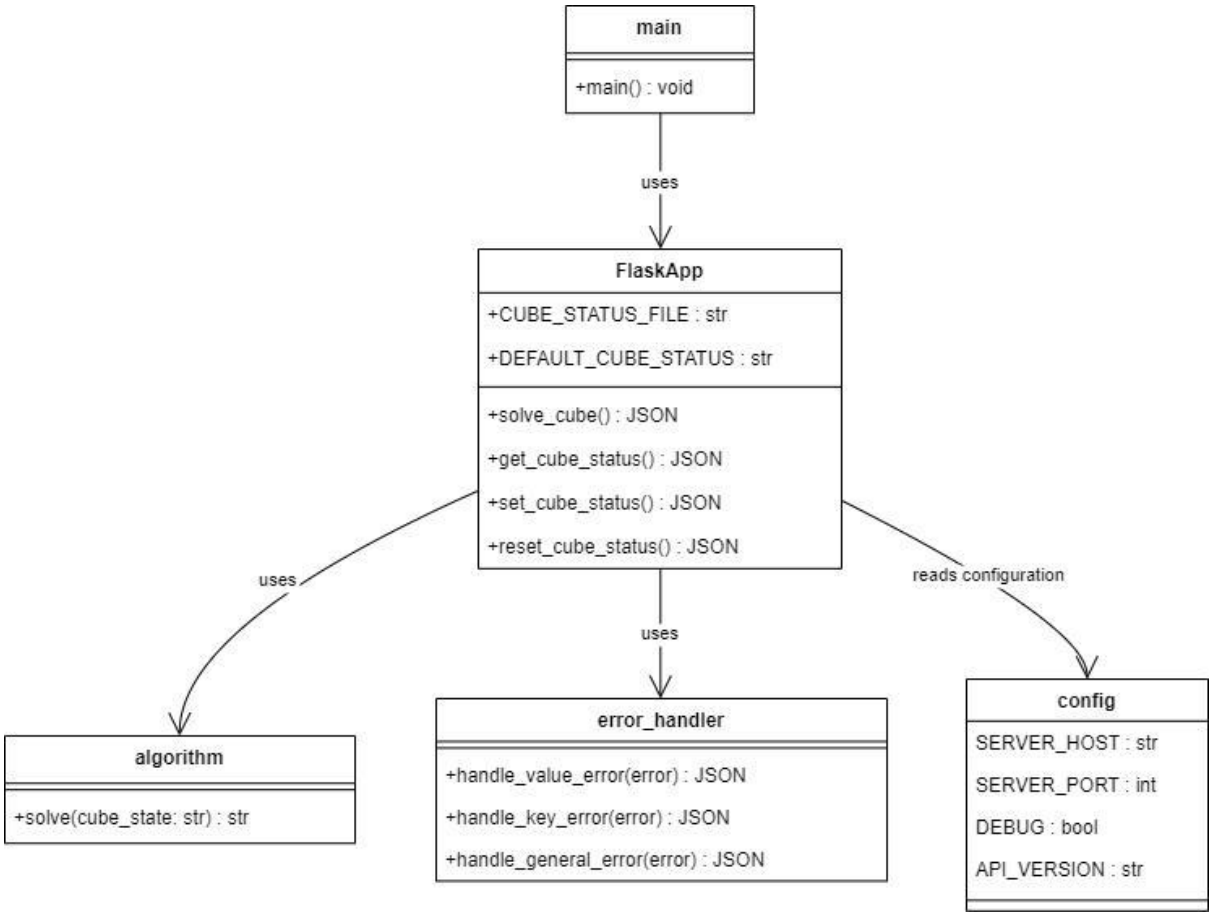
ErrorHandler: This is a class that contains error-handling functions to generate appropriate responses for different error types.

Main: This class is the entry point of the program.

Cube Status: Indicates the status of the Rubik's Cube. It is a text file, but in the UML diagram, we treat it as a class to represent its functionality.

Requirements: Represents the program's dependencies.

5. Class diagram



Algorithm Class:

- Responsibility: The Algorithm class is responsible for solving the Rubik's Cube based on the provided cube state.
- Methods:
    - solve(cube_state: str): Accepts the current Rubik's Cube state as input and returns the optimal solution steps as a string.

FlaskApp Class:

Responsibility: The FlaskApp class handles the web application logic, serving as the backend for user interactions.

Attributes:

- CUBE_STATUS_FILE: str: Represents the file path where the Rubik's Cube status is stored.
- DEFAULT_CUBE_STATUS: str: Represents the default Rubik's Cube status.

Methods:

- solve_cube(): JSON: Solves the Rubik's Cube and returns the solution in JSON format.
- get_cube_status(): JSON: Retrieves the current Rubik's Cube status in JSON format.
- set_cube_status(): JSON: Sets the Rubik's Cube status and returns a confirmation in JSON format.
- reset_cube_status(): JSON: Resets the Rubik's Cube status to default and returns a confirmation in JSON format.

Error_Handler Class:

- Responsibility: The ErrorHandler class handles various types of errors that can occur during the application's operation.
- Methods:
  - handle_value_error(error): JSON: Handles value errors and returns an appropriate JSON response.
  - handle_key_error(error): JSON: Handles key errors and returns an appropriate JSON response.
  - handle_general_error(error): JSON: Handles general errors and returns an appropriate JSON response.

Config Class:

- Responsibility: The Config class stores configuration settings for the application.
- Attributes:

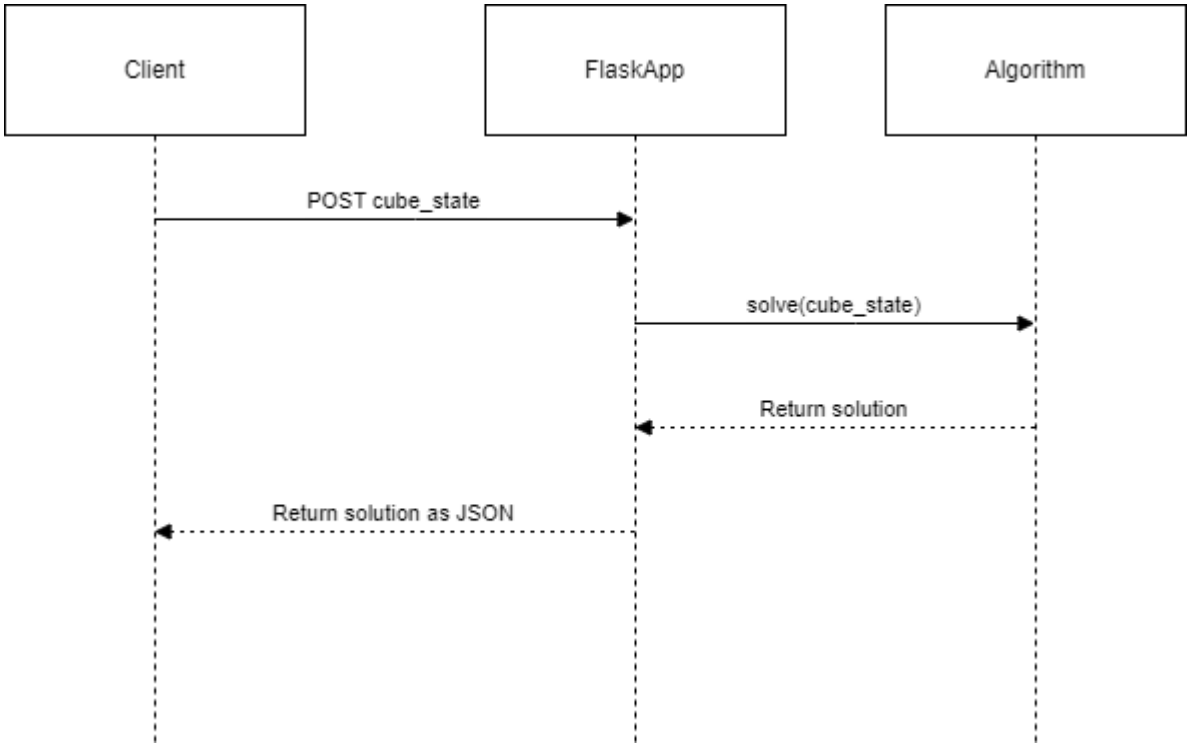
- SERVER_HOST: str: Represents the host address where the server listens for requests.
- SERVER_PORT: int: Represents the port number for communication.
- DEBUG: bool: Indicates whether the server is in debug mode.
- API_VERSION: str: Denotes the version of the API used by clients for compatibility.

Interactions:

- Main -> Uses -> FlaskApp:
  - The Main class uses the functionalities provided by the FlaskApp class to handle user interactions and orchestrate the Rubik's Cube solving process.
- FlaskApp -> Uses -> Algorithm:
  - The FlaskApp class utilizes the Algorithm class to solve the Rubik's Cube based on the user's input.
- FlaskApp -> Uses -> ErrorHandler:
  - The FlaskApp class employs the ErrorHandler class to handle errors and generate appropriate JSON responses for different error scenarios.
- FlaskApp -> Reads Configuration -> Config:
  - The FlaskApp class reads configuration settings from the Config class, ensuring the application operates with the specified server configurations.

6.Designing functions and classes

6.1 SOLVE CUBE STRUCTURE



User Interaction:

1. User Action:
    - The user interacts with the web or mobile application interface, making a POST request to the designated endpoint on FlaskApp, containing the current status of the Rubik's Cube in the request payload. This request is typically initiated when the user requests a solution for the Rubik's Cube puzzle.

FlaskApp Processing:

2. FlaskApp Receives the Request:
    - FlaskApp, running on the server, receives the incoming POST request from the user. It parses the request payload to extract the current status of the Rubik's Cube.
3. FlaskApp Calls the Algorithm Module:

- Upon receiving the Rubik's Cube state, FlaskApp invokes the solve method within the Algorithm Module. This method is responsible for solving the Rubik's Cube and returns the optimal solution steps as a string.

4. Algorithm Module Solves the Rubik's Cube:
    - Within the Algorithm Module, complex solving algorithms analyze the provided Rubik's Cube state. The algorithm explores various combinations and sequences of moves to determine the most efficient solution. This process involves mathematical and logical computations tailored to Rubik's Cube solving techniques.
5. FlaskApp Receives the Solution:
    - After the Algorithm Module successfully solves the Rubik's Cube, it returns the solution steps back to FlaskApp. The solution is typically a series of moves represented as a string (e.g., "R F' U R2 B...").
6. FlaskApp Prepares the Response:
    - FlaskApp constructs a JSON response containing the Rubik's Cube solution received from the Algorithm Module. The response includes the solution steps, status codes, and potentially additional metadata, such as execution time or algorithm version used.

Server Response:

7. FlaskApp Sends JSON Response to User:
    - FlaskApp sends the prepared JSON response back to the user as an HTTP response. The user's client application receives this response and processes the JSON data, presenting the Rubik's Cube solution to the user on the interface.

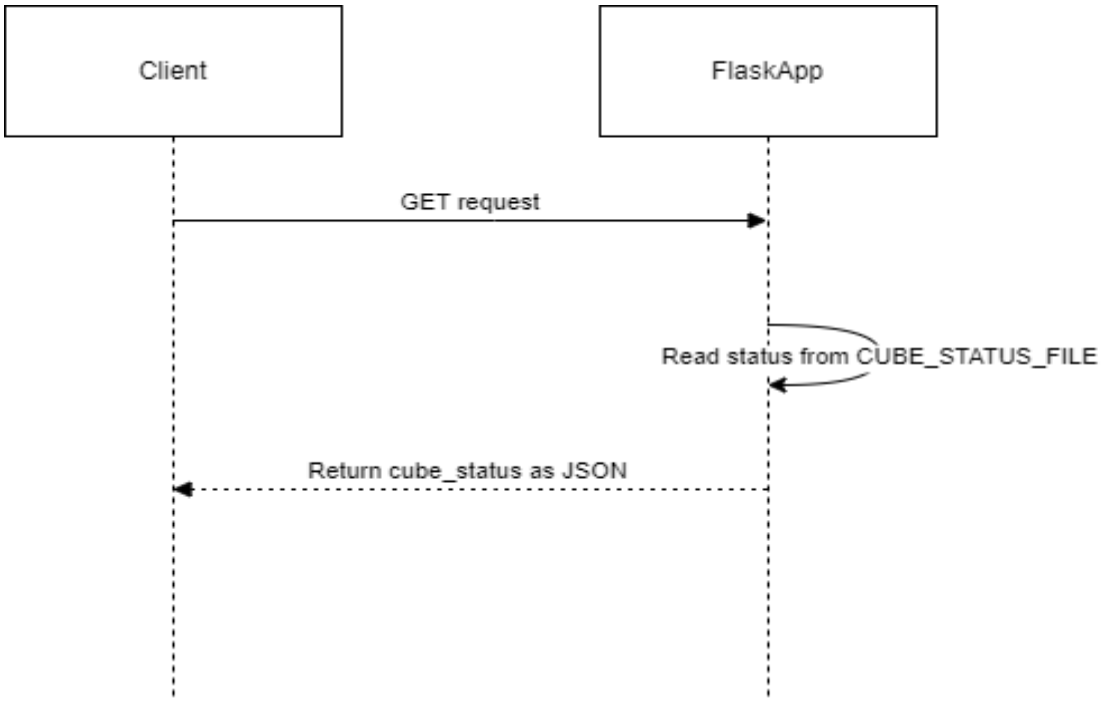
Technical Considerations:

- Error Handling:
    - FlaskApp includes robust error handling mechanisms. If the Algorithm Module encounters errors during solving (e.g., invalid cube state), appropriate error codes and messages are included in the JSON response to inform the user of the issue.
- Asynchronous Processing

- Depending on server load and response time requirements, FlaskApp can implement asynchronous processing. This allows FlaskApp to handle multiple requests concurrently without blocking, enhancing the application's responsiveness.

- Security Measures:
  - FlaskApp implements security measures such as input validation and sanitization to prevent malicious input, ensuring the integrity and security of the Rubik's Cube state data.

6.2 GET CUBE STATUS



User Interaction:

1. User Action:
   - The user interacts with the web or mobile application interface, making a GET request to the designated endpoint on FlaskApp. This request is sent to inquire about the current status of the Rubik's Cube.

FlaskApp Processing:

2. FlaskApp Receives the Request:
    - FlaskApp, running on the server, receives the incoming GET request from the user.
3. FlaskApp Reads the Cube Status:
    - FlaskApp accesses the CUBE_STATUS_FILE located in the file system. This file contains the serialized status of the Rubik's Cube. FlaskApp reads the content of this file to obtain the current state of the Rubik's Cube.
4. FlaskApp Prepares the Response:
    - FlaskApp constructs a JSON response containing the Rubik's Cube status read from the CUBE_STATUS_FILE. The status data may include the positions of individual cubelets, colors, or any other relevant information required to represent the current state of the Rubik's Cube.
5. FlaskApp Sends JSON Response to User:
    - FlaskApp sends the prepared JSON response back to the user as an HTTP response. The user's client application receives this response and processes the JSON data, displaying the current status of the Rubik's Cube on the user interface.

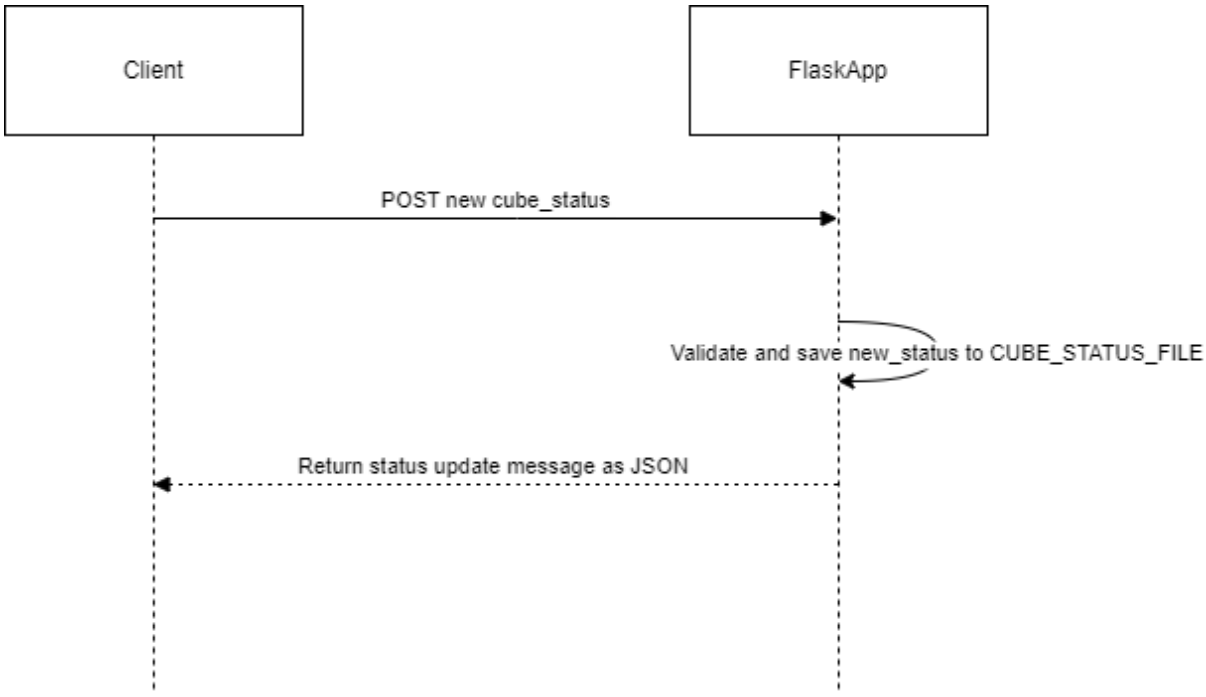
Technical Considerations:

- CUBE_STATUS_FILE Management:
    - FlaskApp handles file I/O operations carefully, ensuring proper file permissions and error handling. It verifies the existence of the CUBE_STATUS_FILE before attempting to read from it, preventing potential issues if the file is missing or inaccessible.
- Response Formatting:
    - FlaskApp formats the response data into a well-structured JSON format, ensuring consistency and ease of processing for the client application. The JSON response typically includes keys representing cubelet positions, colors, or other relevant attributes, along with their corresponding values.
- Caching

- FlaskApp may implement caching mechanisms to optimize performance. If the Rubik's Cube status does not change frequently, FlaskApp can cache the status data, reducing the need for repeated file reads and improving response time.

- Error Handling:
  - FlaskApp includes error handling mechanisms to manage scenarios where the CUBE_STATUS_FILE cannot be accessed or if there are issues with the file's content. Appropriate error codes and messages are included in the JSON response to inform the user of any problems encountered while retrieving the Rubik's Cube status.

6.3 SET CUBE STATUS



User Interaction:

1. User Action:
   - The user interacts with the web or mobile application, initiating a POST request to a specific endpoint on FlaskApp. This request contains the new Rubik's Cube state, signifying an intention to update the cube's status.

FlaskApp Processing:

2. FlaskApp Receives the Request:

    - FlaskApp, running on the server, receives the incoming POST request from the user. The request payload includes the new Rubik's Cube state to be stored.

FlaskApp Writes the New Cube Status:

- FlaskApp performs the following steps to update the Rubik's Cube status:

It opens the CUBE_STATUS_FILE in write mode.

FlaskApp writes the new Rubik's Cube state to the file. This state data may include cubelet positions, colors, or any relevant attributes.Upon successful writing, FlaskApp closes the file, ensuring data integrity.

3. FlaskApp Sends Success Confirmation:

    - FlaskApp prepares a JSON response indicating the successful update of the Rubik's Cube status. This response is sent back to the user as an HTTP response, confirming the status update.

Technical Considerations:

- File I/O and Error Handling:

    - FlaskApp performs file I/O operations securely, with proper error handling. It checks for any issues during file opening, writing, and closing, providing informative error messages in case of any problems.

- Data Validation:

    - Before writing the new Rubik's Cube state to the CUBE_STATUS_FILE, FlaskApp may validate the incoming data to ensure it adheres to the expected format and contains valid cube state information. Invalid data is rejected and an error response is sent to the user.

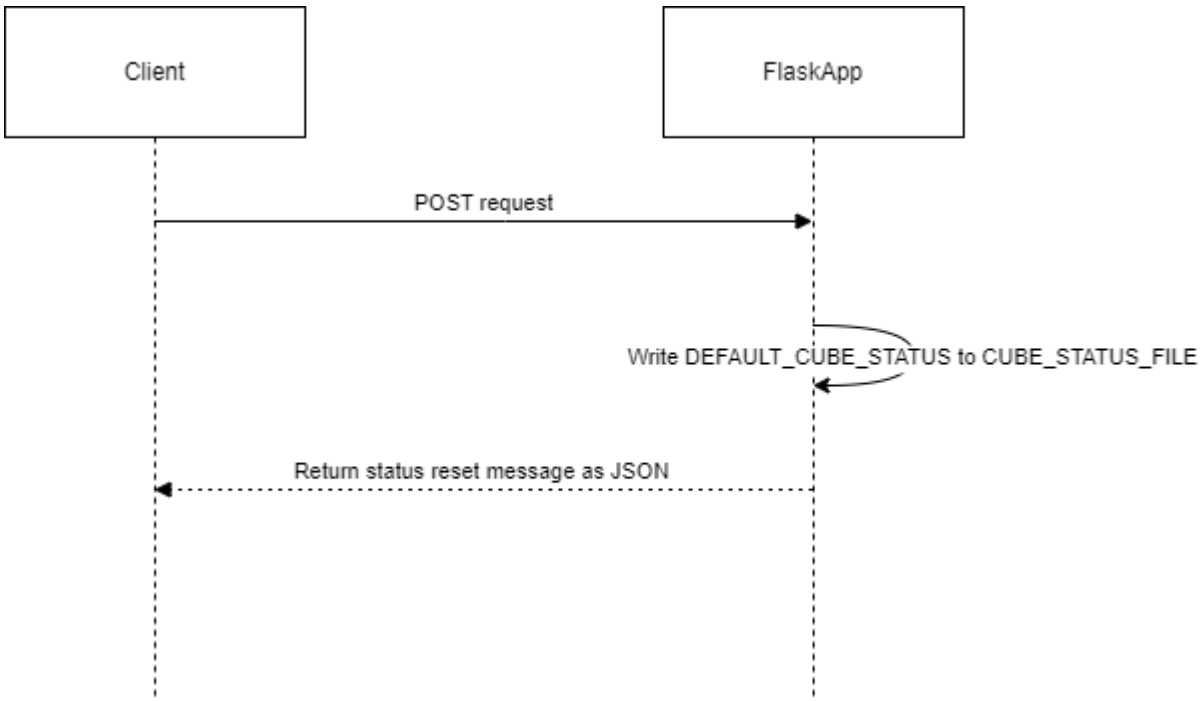
- Concurrency Handling :

    - FlaskApp may implement concurrency control mechanisms to manage simultaneous write requests. This prevents data corruption and ensures the integrity of the stored Rubik's Cube status.

- Response Formatting:

- FlaskApp formats the response into a JSON message that signifies the successful update of the Rubik's Cube status. This response may include a confirmation message along with a success status code.

6.4 RESET CUBE STATUS



User Interaction:

1. User Action:
    - The user interacts with the application, sending a POST request to FlaskApp to reset the Rubik's Cube state.

FlaskApp Processing:

2. FlaskApp Receives the Reset Request:
    - FlaskApp, upon receiving the POST request, identifies it as a reset request and proceeds with the reset process.
3. FlaskApp Resets the Rubik's Cube State:
    - FlaskApp accesses the CUBE_STATUS_FILE in the file system.

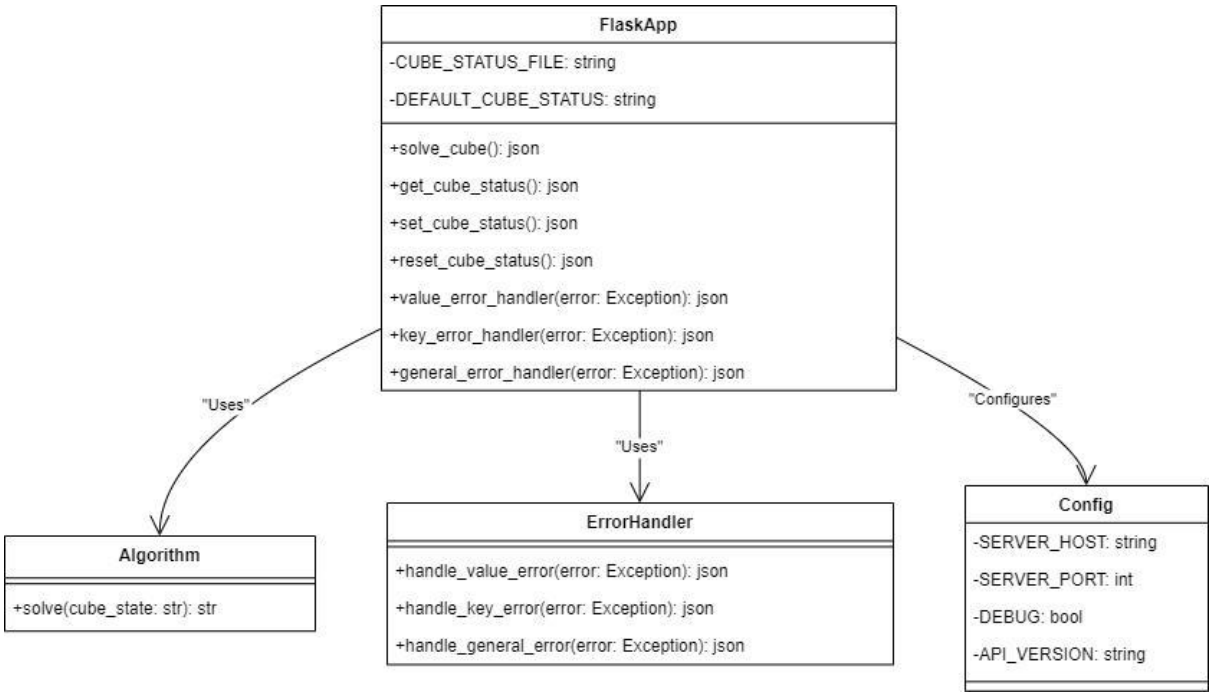
- It writes the default Rubik's Cube state, representing an unsolved configuration, to the file.
- Once the write operation is successful, FlaskApp closes the file, confirming the reset operation.

4. FlaskApp Sends Success Confirmation:
    - FlaskApp constructs a JSON response containing a success message confirming the reset operation.
    - The response includes a success status code (e.g., 200 OK) along with a detailed success message indicating that the Rubik's Cube state has been reset to its default configuration.
5. FlaskApp Responds to the User:
    - FlaskApp sends the JSON response back to the user's client application, acknowledging the successful reset of the Rubik's Cube state.

Technical Considerations:

- File Locking
    - FlaskApp may implement file-locking mechanisms to prevent concurrent write operations, ensuring data consistency during the reset process.
- Atomic Operations:
    - FlaskApp performs the reset operation as an atomic operation, ensuring that the file is updated in a single, cohesive step to maintain data integrity.
- Response Format:
    - The JSON response generated by FlaskApp adheres to a standardized format, containing appropriate HTTP status codes and clear, human-readable messages, ensuring consistency in communication with the client application.
- Logging
    - FlaskApp may log the reset operations, including timestamps and user identifiers, for auditing purposes or troubleshooting potential issues.

7. Designing classes:



I. Algorithm Class

Methods:

solve(cube_state: str): str

Description: This method takes a string representing the current state of the Rubik's Cube as input and returns a string representing the solution steps to restore the Rubik's Cube to its solved state. The algorithm employed within this class efficiently solves the Rubik's Cube, ensuring the minimum number of steps required for restoration.

II. FlaskApp Class

Attributes:

CUBE_STATUS_FILE: string

Description: A constant string representing the file where the current status of the Rubik's Cube is stored.

DEFAULT_CUBE_STATUS: string

Description: A constant string representing the default state of the Rubik's Cube.

Methods:

solve_cube: json

Description: Invokes the solve method from the Algorithm class, passing the current Rubik's Cube state, and returns the solution steps in JSON format.

get_cube_status(): json

Description: Reads and retrieves the current status of the Rubik's Cube from the designated file and returns it in JSON format.

set_cube_status(): json

Description: Accepts a JSON object representing the Rubik's Cube state, updates the status file, and returns a JSON response indicating success.

reset_cube_status(): json

Description: Resets the Rubik's Cube status to the default state, updates the status file, and returns a JSON response confirming the reset.

value_error_handler(error: Exception): json

Description: Handles exceptions related to value errors and returns an appropriate JSON error response.

key_error_handler(error: Exception): json

Description: Handles exceptions related to key errors and returns an appropriate JSON error response.

general_error_handler(error: Exception): json

Description: Handles general exceptions and returns an appropriate JSON error response.

III. ErrorHandler Class

Methods:

handle_value_error(error: Exception): json

Description: Handles value errors, typically arising from incorrect input data, and returns a JSON error response indicating the specific issue.

handle_key_error(error: Exception): json

Description: Handles key errors, which occur when a dictionary key is not found, and returns a JSON error response specifying the missing key.

handle_general_error(error: Exception): json

Description: Handles general errors that do not fall into the specific categories above, providing a generic JSON error response.

IV. Config Class

Attributes:

SERVER_HOST: string

Description: A string representing the host address for the Flask server.

SERVER_PORT: int

Description: An integer representing the port number on which the Flask server listens.

DEBUG: bool

Description: A boolean flag indicating whether the application is in debug mode.

API_VERSION: string

Description: A string representing the version of the API used in the application.

V. Usage Relationships

- Flask App -> Uses -> Algorithm Class:

The Flask application utilizes the Algorithm class to solve the Rubik's Cube based on the user's input and generate the appropriate solution steps.

- Flask App -> Uses -> ErrorHandler Class:

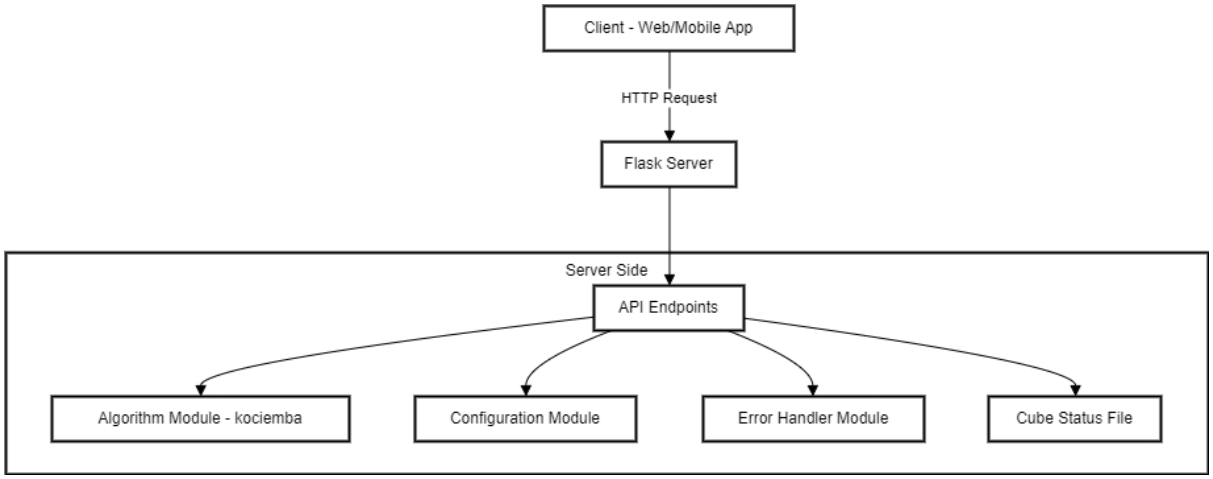
The Flask application employs the ErrorHandler class to handle various types of exceptions that might occur during runtime, ensuring graceful error handling and meaningful error responses to the users.

- Flask App -> Configures -> Config Class:

The Flask application is configured with the settings provided by the Config class, including server host, port, debug mode, and API version. These configurations ensure the proper functioning and behavior of the Flask server.

This design facilitates a clear separation of concerns, ensuring that the Rubik's Cube solving algorithm, error handling, and server configuration are encapsulated within their respective classes, promoting modularity and maintainability of the application.

8. Client Server Architecture



Client:

The client represents the web or mobile application, acting as the user interface for interacting with the Rubik's Cube solving application. It initiates requests to the Flask server through API endpoints for operations such as solving the Rubik's Cube, retrieving the current cube status, updating the cube status, and resetting the cube to its default state.

Flask Server:

The Flask server serves as the backend application, handling requests from clients and orchestrating the Rubik's Cube solving process. It processes incoming requests, delegates tasks to various server-side components, and sends back appropriate responses to clients.

Server-Side Components:

API Endpoints:

API Endpoints are specific URLs the Flask server exposes to handle distinct client requests. These include:

- solve_cube: Receives the current Rubik's Cube state as input and returns the optimal solution steps to solve the cube.
- get_cube_status: Retrieves and returns the current state of the Rubik's Cube to the client.
- set_cube_status: Accepts the client's new Rubik's Cube state and updates the cube status.
- reset_cube_status: Resets the Rubik's Cube to its default state.

Algorithm Module:

The Algorithm Module contains the Rubik's Cube-solving algorithm. When the solve cube endpoint is called, this module processes the current cube state using sophisticated algorithms to find the shortest sequence of moves required to solve the Rubik's Cube. It ensures efficiency and accuracy in solving the cube puzzle.

Config Module:

The Config Module handles server configuration settings, including:

- SERVER_HOST: Specifies the host address on which the Flask server listens for incoming requests.
- SERVER_PORT: Defines the port number through which the server communicates with clients.
- DEBUG: A Boolean flag indicating whether the server is in debug mode, providing detailed error messages during development.
- API_VERSION: Denotes the version of the API used by clients for compatibility and version control.

ErrorHandler Module:

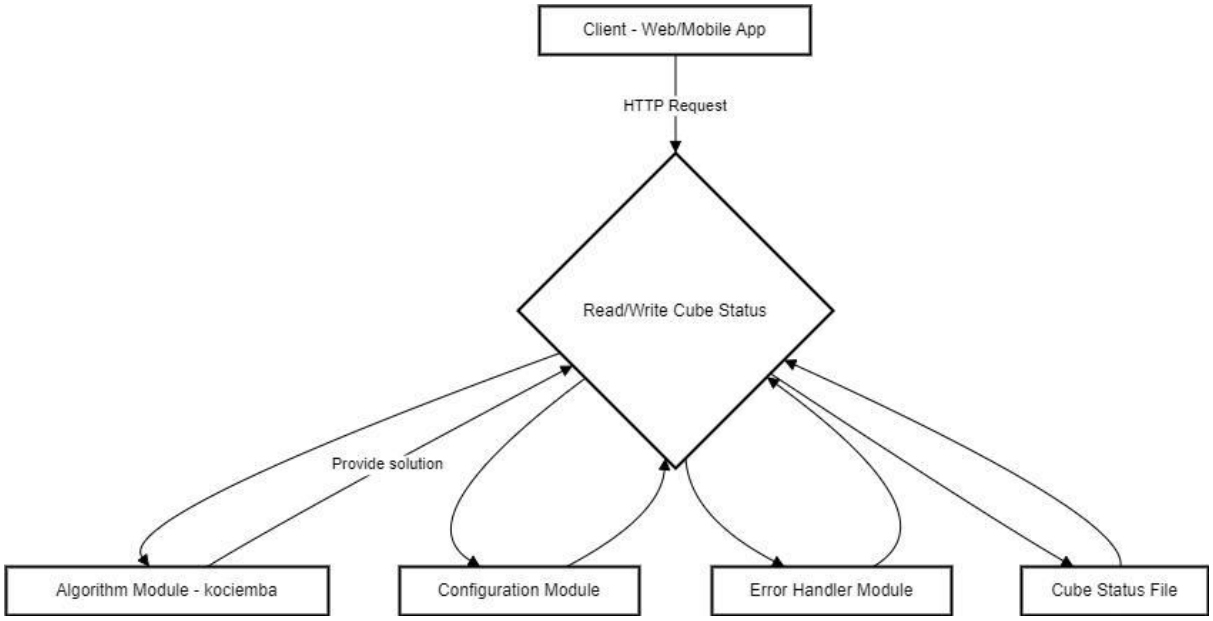
The ErrorHandler Module deals with different types of errors that may occur during server-side operations. It provides tailored error handling for value, key, and general errors exceptions. Proper error responses are generated to ensure clients receive meaningful feedback in case of erroneous requests.

CubeStatusFile:

The CubeStatusFile represents the file where the current state of the Rubik's Cube is stored persistently. It serves as the data storage mechanism, allowing the server

to maintain the Rubik's Cube's state across different client interactions. The server reads from and writes to this file to retrieve and update the cube status as needed.

9. Sequence Diagram



The image shows a detailed flowchart that represents a client's process, specifically a web or mobile app, interacting with a server. The client sends an HTTP request which is first received by a component named "Read/Write Cube Status". This component provides a solution which is then passed onto an "Algorithm Module - Kociemba", responsible for further processing. The outcome of the algorithm module is sent to a "Configuration Module" for adjustments and then directed to an "Error Handler Module". This module is tasked with handling any potential errors that may occur. Finally, the data is sent to a "Cube Status File". The structure of the flowchart suggests a logical and systematic procedure for reading and writing cube statuses.

10. Goal and Milestones

10.1 Main Objective:

Our primary objective is to enhance the user experience of solving the Rubik's Cube by making the entire process more intuitive and convenient. Achieving this objective involves the following key milestones:

1. Optimized Solution Steps:
    - Milestone: The program has reached a significant milestone by providing users with the most concise and optimal solution steps to solve the Rubik's Cube. Users are guided with the minimum number of steps required to successfully solve the puzzle, ensuring efficiency in their solving endeavors.
2. Integration of Memory Functionality:
    - Milestone: We have successfully implemented a memory function within the application. This feature allows users to retain their current Rubik's Cube recovery progress, even if they haven't completed the solving process. When users return to the application, they can seamlessly continue their recovery journey from where they left off. Additionally, for users who prefer a fresh start, the program offers the convenience of resetting the Rubik's Cube memory with a single click.

10.2 Achieved Progress:

- Concise Solution Steps: Users now receive an optimal and streamlined set of instructions for Rubik's Cube solution, minimizing the complexity and maximizing efficiency in their solving attempts.
- Persistent User Progress: Users can confidently step away from their solving process, secure in the knowledge that their progress is preserved. The application's memory function ensures a seamless experience, allowing users to resume solving tasks whenever they engage with the Rubik's Cube challenge again.

10.3 Upcoming Milestones:

- Intuitive User Interface: Our next milestone involves refining the user interface to make it even more intuitive. We aim to create a visually engaging and user-friendly design, enhancing user interaction and navigation throughout the Rubik's Cube solving process.
- Enhanced User Guidance: Our next focus is implementing interactive tutorials and hints within the application. These guides will provide users with valuable insights, helping them improve their problem-solving skills and strategies.
- Expanded Platform Compatibility: We plan to expand the application's compatibility across various platforms, ensuring that users can enjoy the

Rubik's Cube-solving experience seamlessly, whether on web browsers, mobile devices, or desktop applications.

11. Prioritization

Primary Focus:

Intuitive 3D Interface and User Interaction:

- Priority: Highest
- Description: Developing a visually intuitive 3D Rubik's Cube interface is paramount. Users should effortlessly manipulate the virtual Cube, while the application provides clear recovery steps in an engaging and user-friendly manner.

Secondary Focus:

User-Generated Cube Data Import:

- Priority: High
- Description: Allowing users to import their own Rubik's Cube configurations is essential. The import process must be straightforward and user-friendly. Additionally, the application should intelligently store incomplete Cube states for users to resume later.

Optional Enhancements:

Simplified Data Import (e.g., Photo Input):

- Priority: Medium
- Description: Implementing features like photo input can further streamline data import, enhancing user experience. While not essential, these options would significantly improve user convenience.

Enhanced User Controls (e.g., Keyboard Support):

- Priority: Medium
- Description: Providing alternative control methods, such as keyboard support, can augment user interaction. This flexibility

enhances accessibility and usability, offering users varied ways to operate the virtual Rubik's Cube.

Interactive Tutorials:

- Priority: Medium
- Description: Crafting interactive tutorials, especially graphical ones, empowers users to learn Rubik's Cube solving methods effectively. These tutorials should be engaging, informative, and progressively guide users through solving techniques.

User Accounts and Personalization (Login System):

- Priority: Low
- Description: Introducing a login system allows for personalized experiences. Users can maintain independent Rubik's Cubes, providing a sense of ownership. However, this feature is less crucial due to the application's standalone nature.

Additional Small Features (e.g., Timing, Random Cube Generation, Step Recording):

- Priority: Low
- Description: Implementing minor features, such as solving timing, random Cube generation for practice, and step recording, can enhance user engagement. While not essential, these additions contribute to the overall user experience.

12. Current and proposed solutions

Current Implementation:

- Backend Technology: The backend is developed using Python, incorporating open-source packages like Flask and kociemba. These packages provide efficient solutions for Rubik's Cube recovery, ensuring optimal performance.
- Platform Compatibility: The application runs seamlessly on the Browser and website, eliminating the need for additional tools. Users can access the program through the Browser and Website

Proposed Enhancements:

- 3D Visualization and User Interaction: Enhance the 3D Rubik's Cube interface, ensuring smooth user interactions and responsive controls. Utilize advanced libraries to create an immersive experience, making Cube manipulation intuitive and enjoyable.
- Data Import Optimization: Streamline the user-generated data import process, exploring options like photo input or intuitive drag-and-drop functionality. Ensure robust error handling and validation to handle various input formats.
- Interactive Tutorials: Develop engaging graphical tutorials, guiding users through Rubik's Cube solving methods. Use animations and interactive elements to simplify complex algorithms, making learning accessible to users of all skill levels.
- Optional Features Implementation: Consider implementing optional features based on user feedback. Prioritize enhancements that align with user preferences, ensuring a customized and enjoyable experience for all users.