This documentation provides an overview of the code's architecture, functionality, and usage of libraries.

**High-Level Architecture of the System:**

**Client-Server-Database Architecture:** This application follows a typical client-server-database architecture, where the client is the web browser, the server is the Flask application, and the database is used for storing user data, folders, and flashcards.

**Client:** The web browser is used as the client to interact with the Flask server.

**Server:** The Flask server handles HTTP requests and serves web pages using Jinja2 templates. It also communicates with the database to perform CRUD (Create, Read, Update, Delete) operations.

**Database:** The SQLAlchemy library is used to interact with the database, which stores user information, user-generated folders, and flashcards.

**Project Structure:**

**Project Files and Folders Description:**

1. **static:** This directory contains static files, such as images.

2. **templates:** This directory contains HTML templates for rendering web pages. Templates such as `base.html`, `dashboard.html`, `folder.html`, `login.html`, etc., are used by the Flask application to generate web pages.

3. **main.py:** This is the main Python application file that contains the Flask application and all the routes, view functions, and configurations for the web application. It is the entry point for running the Flask app.

4. **generate\_flashcards.py:** This Python file contains functions to generate flashcards based on text input.

5. **models.py:** This file contains the database models and the SQLAlchemy setup. It defines the structure of the database tables and their relationships.

6. **utilities.py:** contains utility functions or helper functions that are used throughout the Flask web application.

7. **mydb.db:** This is an SQLite database file. It's used by the application to store data, such as user information, folders, flashcards, etc. The tables and schema are defined in `models.py`, and the database is created based on that schema.

**Code Elements and What Each Section Does:**

**Imports:** The code starts by importing necessary libraries and modules.

**Flask Configuration:**

* An instance of the Flask application is created and configured.
* Secret key is set for session management.
* LoginManager is initialized for user authentication.

**User Model:** SQLAlchemy models for **User**, **Folder**, and **Flashcard** are defined, which represent the database tables.

**Database Configuration:**

SQLAlchemy is used to set up a database engine and create database tables.

A session is created for database interactions.

**User Registration, Login, and Logout Routes:** These routes handle user registration, login, and logout, including form validation and user authentication.

**Dashboard Route:** Requires login and displays the user's dashboard with folders and user-specific data.

**Folder and Flashcard Routes:** These routes allow users to create, view, edit, and delete folders and flashcards. Some of these routes include form processing and permission checks.

**Generate Flashcards Routes:** Routes for generating flashcards from text input.

**Error Handlers:** Handlers for 404 (Not Found) and 500 (Internal Server Error) errors.

**Main Execution:** The application runs when executed.

**Field Names and Descriptions:**

**User Model:**

* id: User ID (Primary Key)
* username: User's username
* password: Hashed password
* points: User's points

**Folder Model:**

* id: Folder ID (Primary Key)
* name: Folder name
* color: Folder color
* user\_id: Foreign key to the User table
* full\_access: Flag indicating if the user has full access to modify/delete the folder.
* latest\_score: The latest score related to the folder.

**Flashcard Model:**

* id: Flashcard ID (Primary Key)
* question: Flashcard's question
* answer: Flashcard's answer
* attempts: Number of attempts made for the flashcard.
* folder\_id: Foreign key to the Folder table

**Connections between the models and the elements of the project:**

**User:**

* A user can be associated with multiple folders (1 to many relationship).
* A user can have user points, which can be updated based on exam results.

**Folder:**

* Each folder belongs to one user (many to one relationship).
* Each folder can contain multiple flashcards (1 to many relationship).
* Folders can have attributes such as a name, color, and a latest\_score.

**Flashcard:**

* Each flashcard belongs to one folder (many to one relationship).
* Flashcards have attributes like question, answer, and attempts.

**UserPoints:**

* User points are associated with a user and can be updated based on exam results.

**Exam:**

* An exam result may include a grade (percentage) that can be associated with a folder or other attributes.

**Libraries Used and Their Purpose:**

**Flask:** Used for creating the web application and handling HTTP requests and responses.

**Flask-Login:** Provides user authentication and session management.

**SQLAlchemy:** Used for database interactions and creating database models.

**datetime:** Used for handling date and time operations.

**openai:** Enables interaction with the OpenAI GPT-3 model to generate questions and answers from text.

**os:** Provides access to the operating system for environment variable access.

**time:** Used for adding time-related delays for rate limiting.

**werkzeug.security:** Used for password hashing and verification.

**Jinja2:** Templating engine for rendering HTML pages.

**Outsourced Code:**

The code includes calls to the OpenAI GPT-3 model to generate flashcards from text. The logic for generating questions and answers from text and interacting with the OpenAI API is contained in the generate\_questions\_and\_answers and related functions.