HIVE Application Documentation

Overview

HIVE is a robust, web-based productivity and communication platform developed using Flask, a lightweight Python web framework. Designed to cater to individuals, teams, and communities, HIVE provides a centralized hub for users to register, log in, and access a feature-rich dashboard. The platform enables users to manage personal notes, upload and share files, engage in real-time chats with other users, and view received messages—all within a visually engaging, modern interface. HIVE leverages animated gradient text, dynamic video backgrounds, and a responsive design powered by Tailwind CSS and Bootstrap, ensuring an immersive user experience.

The application is structured to be extensible, scalable, and user-friendly, making it ideal for productivity, collaboration, and communication needs. It combines backend functionality with real-time features, offering a seamless experience across desktops and mobile devices. HIVE is available in two versions: one with HTML-based routes for a traditional web interface and another with API-based routes for programmatic access, catering to different use cases and integration needs.

Purpose

HIVE's primary goal is to serve as an all-in-one solution for productivity and collaboration. By integrating note-taking, file sharing, and real-time communication, HIVE addresses the needs of users seeking a single platform to organize tasks, share resources, and connect with others efficiently. Whether for personal use, remote teams, or community engagement, HIVE provides a cozy, intuitive environment to foster productivity and interaction. The HTML version offers a user-friendly web interface, while the API version enables developers to integrate HIVE's functionality into other applications or services.

Key Features

HIVE offers a wide array of features to enhance productivity and communication, available in both HTML and API versions with slight differences in interaction:

1. **User Authentication and Security**
    - Users can register with a unique username and password, log in, and securely log out.
    - Authentication is managed using Flask-Login and Flask-WTF for the HTML version, and session-based authentication for the API version, with user data securely stored in a SQLite database (users.db).

- Passwords are stored in plain text (as seen in the code), but implementing hashing (e.g., using werkzeug.security for generate_password_hash and check_password_hash) is recommended for production use.

2. **Dashboard Experience**

   - In the HTML version, the dashboard (dashboard.html) serves as the central hub with a visually striking welcome section, forms for notes and files, and real-time chat/messages.
   - In the API version, dashboard functionality is accessed programmatically via endpoints like /add_note, /view_note, and /view_messages (not directly shown but implied), returning JSON responses for integration into custom UIs or applications.

3. Notes Management

   - **Both versions allow creating, editing, and deleting personal notes**.
   - **HTML:** Managed through dashboard.html, edit_note.html, and Flask routes.
   - **API:** Handled via /add_note (POST), /edit_note/note_id (PUT), /view_note/note_id (GET), and /delete_note/note_id (DELETE), returning JSON responses.

4. **File Sharing and Management**

   - **Users can upload, view, and delete files in both versions.**
   - **HTML:** Managed through forms and links in dashboard.html.
   - **API:** Handled via /upload (POST) for uploads, /uploads/filename (GET) for viewing, and implied DELETE for deletion (not shown but possible via extension).

5. **Real-Time Chat System**

   - In the HTML version, real-time chat is available via chat.html using Socket.IO for instant messaging.
   - The API version does not include real-time chat but could be extended with WebSocket endpoints or periodic polling for messages, accessed via /view_messages (implied) or custom endpoints.

6. **Messages Management**

    - **Both versions allow viewing received messages with sender details and timestamps.**
    - **HTML:** Displayed in dashboard.html and view_messages.html with real-time updates via Socket.IO.
    - **API:** Could be accessed via a /view_messages endpoint (not shown but implied), returning JSON data for integration.

7. **Visual and Interactive Design**

    - **HTML version:** Features a modern, dark-themed UI with Tailwind CSS for most templates and Bootstrap for edit_note.html, register.html, and view_messages.html, plus animated gradients and video backgrounds.
    - **API version:** Focuses on JSON responses, leaving UI design to the client application integrating the API.

Libraries and Modules Used

The HIVE application relies on several Python libraries and modules, each serving a specific purpose to enable its functionality. Below is a detailed explanation of each:

- **Flask**

    - **Description:** Flask is a micro web framework written in Python, designed to be lightweight and flexible, allowing developers to build web applications quickly with minimal dependencies.
    - **Why Used:** Flask is the core framework for both HIVE versions, providing the structure for routing, request handling, and session management. It defines routes for HTML templates in the HTML version and RESTful API endpoints in the API version, rendering HTML or returning JSON responses.

- **Flask-SQLAlchemy**

    - **Description:** Flask-SQLAlchemy is an extension for Flask that integrates SQLAlchemy, a powerful SQL toolkit and Object-Relational Mapping (ORM) library, to interact with databases seamlessly.
    - **Why Used:** In both HIVE versions, Flask-SQLAlchemy manages the SQLite database (users.db), defining models (User, Note, Message) to store and retrieve user data, notes, and messages. It simplifies database operations like CRUD for both HTML forms and API requests.

- **Flask-WTF**
    - **Description:** Flask-WTF is an extension for Flask that integrates WTForms, a flexible forms validation and rendering library, to handle web forms securely with CSRF protection.
    - **Why Used:** In the HTML version, Flask-WTF creates and validates forms like RegisterForm, LoginForm, NoteForm, and UploadForm for user input. In the API version, while forms are not used directly, the validation logic (e.g., length checks) is adapted into API request validation, ensuring consistent data integrity.

- **Werkzeug**
    - **Description:** Werkzeug is a WSGI (Web Server Gateway Interface) utility library for Python, providing tools for request and response handling, URL routing, and security features.
    - **Why Used:** In both versions, Werkzeug's secure_filename function sanitizes filenames during file uploads, preventing security vulnerabilities like path traversal attacks. It ensures uploaded files have safe names before being saved to the uploads directory.

- **Datetime**
    - **Description**: datetime is a built-in Python module for handling dates and times, providing classes to manipulate and format date-time objects.
    - **Why Used:** HIVE uses datetime.utcnow to timestamp messages in the Message model, allowing users to see when messages were sent in both HTML displays and API responses, adding context and organization.

- **Functools**
    - **Description:** functools is a built-in Python module offering higher-order functions and operations on callable objects, such as decorators.
    - **Why Used:** In the HTML version, functools.wraps is used in the login_required (not shown in API but implied) and handle_errors decorators in app.py to preserve the metadata of decorated functions, ensuring proper function naming and documentation while adding authentication logic. This could be adapted for API middleware if needed.
- **Os**
    - **Description:** os is a built-in Python module for interacting with the operating system, providing functions to handle files, directories, and system paths.
    - **Why Used:** In both versions, os is used to create and manage the uploads directory (os.makedirs), list uploaded files (os.listdir), and delete files (os.remove). It ensures the application can handle file operations safely and

cross-platform for both HTML and API interactions.

- **Tailwind CSS**
    - **Description:** Tailwind CSS is a utility-first CSS framework delivered via CDN, allowing developers to build custom designs quickly by applying low-level utility classes.
    - **Why Used:** Used exclusively in the HTML version in templates like index.html, login.html, dashboard.html, and chat.html to create a responsive, modern, and dark-themed UI. It simplifies styling with classes like bg-gray-900, text-white, and rounded-lg, enhancing development speed and consistency for the web interface.

- **Bootstrap**
    - **Description:** Bootstrap is a popular CSS framework (delivered via CDN) that provides pre-designed components, grids, and responsive layouts for building web interfaces.
    - **Why Used:** Used exclusively in the HTML version in edit_note.html, register.html, and view_messages.html to create clean, minimalistic, and responsive designs with components like form-control, btn-primary, and list-group-item. It ensures a consistent look for these templates while leveraging Bootstrap's grid system.

These libraries and modules work together to create a fully functional, secure, and interactive web application and API, addressing HIVE's requirements for web development, database management, form handling, file management, and styling in the HTML version, and RESTful API functionality in the API version.

Technical Details

- **Framework:** Flask, a micro web framework for Python, providing simplicity and flexibility.
- **Database:** SQLite (users.db), managed via Flask-SQLAlchemy, storing user credentials, notes, and messages in three models (User, Note, Message).
- **Frontend (HTML Version):**
    - HTML for structure.
    - Tailwind CSS (via CDN) for most templates, offering utility-first styling for rapid development and responsiveness.
    - Bootstrap (via CDN) for edit_note.html, register.html, and view_messages.html, providing a consistent, grid-based layout.
    - JavaScript with Socket.IO (not shown in API but implied for real-time features) for real-time chat functionality.

- **API (API Version)**: JSON-based RESTful endpoints, returning data for integration into custom clients or applications, without a predefined UI.
- **Real-Time Communication:** Socket.IO (used in HTML version) enables real-time message broadcasting, with rooms based on usernames for targeted delivery; the API version could extend this with WebSocket endpoints or polling.
- **Dependencies:**
    - **Flask:** Core web framework.
    - **Flask-SQLAlchemy:** ORM for database integration.
    - **Flask-WTF:** Form handling and validation (used in HTML, adapted for API validation).
    - **Werkzeug:** Utilities for file handling and security.
    - **Tailwind CSS and Bootstrap:** Styling frameworks (via CDN, HTML version only).
- **Configuration:**
    - **app.secret_key**: A unique key for session management and CSRF protection.
    - **SQLALCHEMY_DATABASE_URI:** Points to the SQLite database (users.db).
    - **UPLOAD_FOLDER:** Specifies the directory (uploads) for file uploads.

How It Works

1. **HTML Routes:** The HTML version of HIVE provides a traditional web interface for users to interact with the application directly through a browser. Below are the key routes and their functionalities:

- **/ (GET/POST):** Renders index.html for registration or processes registration form submission using RegisterForm, redirecting to /login on success.
- **/login (GET/POST):** Renders login.html for login or processes login form submission using LoginForm, redirecting to /dashboard on success.
- **/dashboard (GET/POST):** Displays dashboard.html with notes, files, and messages for logged-in users, handling note additions and file uploads.
- **/delete_note/int:note_id:** Deletes a specific note for the logged-in user, redirecting to /dashboard.
- **/upload (POST):** Handles file uploads, saving to uploads and redirecting to /dashboard.
- **/uploads/filename:** Serves uploaded files from the uploads directory.
- **/delete_file/filename (POST)**: Deletes a specific uploaded file, redirecting to /dashboard.
- **/edit_note/int:note_id (GET/POST):** Renders edit_note.html for editing a note or processes the update, redirecting to /dashboard.
- **/chat:** Renders chat.html for real-time chat with other users using Socket.IO.
- **/send_message (POST):** Handles message sending (not fully shown but implied), storing in the database and updating via Socket.IO.
- **/view_messages:** Renders view_messages.html to list received messages for the logged-in user.
- **/logout:** Clears the session and redirects to /.

This version uses HTML templates, Tailwind CSS, Bootstrap, and Socket.IO for real-time features, providing a complete web application experience.

2. **API Routes:** The API version of HIVE provides a RESTful interface for programmatic access, returning JSON responses for integration into custom applications or clients (e.g., mobile apps, third-party services). Below are the key routes and their functionalities:

- **/register (POST):** Registers a new user by accepting JSON data ("username": "string", "password": "string"), validates input, checks for duplicates, and returns a JSON response ("message": "Registration successful! Please login.") with status 200 on success, or an error message with 400/403.
- **/login (POST):** Logs in a user by accepting JSON data ("username": "string", "password": "string"), authenticates against users.db, sets a session, and returns ("message": "Login successful.", "user": "username") with status 200 on success, or an error with 400.
- **/add_note (POST):** Adds a new note for the logged-in user by accepting JSON data ("content": "string"), validates input, and returns ("message": "Note added successfully.", "note_id": int) with status 201 on success, or an error with 400/401/404.
- **/view_note/int:note_id (GET):** Retrieves a specific note for the logged-in user, returning ("id": int, "content": "string") with status 200 on success, or an error with 401/404/500.
- **/edit_note/int:note_id (PUT):** Updates a specific note for the logged-in user by accepting JSON data ("content": "string"), validates input, and returns ("message": "Note updated successfully.") with status 200 on success, or an error with 400/401/403/404.
- **/delete_note/int:note_id (DELETE):** Deletes a specific note for the logged-in user, returning ("message": "Note deleted successfully.") with status 200 on success, or an error with 403/404.
- **/upload (POST):** Uploads a file for the logged-in user by accepting a multipart/form-data request with a file field, saving it to uploads, and returning ("message": "File uploaded successfully.", "filename": "string") with status 201, or an error with 400/401.
- **/uploads/filename (GET):** Serves an uploaded file from the uploads directory, returning the file content on success, or ("message": "File not found.") with status 404 if the file doesn't exist.
- **/logout (POST):** Logs out the current user by clearing the session, returning ("message": "Logout successful.") with status 200, or ("message": "No user is logged in.") with status 400.

This version uses JSON for data exchange, eliminating HTML templates and focusing on RESTful API interactions for programmatic access.

GitHub Links and Local Installation

1. **HTML Routes (Web Interface)**

- **GitHub Link:** https://github.com/srujan-zenshastra/Week1-Main-Project
- **Description:** This repository contains the HTML-based version of HIVE, featuring a full web interface with templates (index.html, login.html, dashboard.html, etc.), Tailwind CSS, Bootstrap, and Socket.IO for real-time chat.

2. **API Routes (Programmatic Interface)**

- **GitHub Link:** https://github.com/srujan-zenshastra/week1_project3_ThunderClient_API
- **Description:** This repository contains the API-based version of HIVE, featuring RESTful JSON endpoints for programmatic access, without HTML templates, suitable for integration into custom applications or clients.

How to Install and Run Locally(Main Project)

1. **Clone the Repository:** git clone https://github.com/srujan-zenshastra/Week1-Main-Project.git cd Week1-Main-Project
2. **Install Dependencies:** Use pip to install the required packages: pip install flask flask-sqlalchemy flask-wtf flask-socketio werkzeug
3. **Set Up the Database:** Ensure users.db is in the instance folder (or run the app once to create it): python app.py
4. **Run the Application**: Execute app.py to start the Flask server: python app.py The app will run on http://0.0.0.0:8000 in debug mode.
5. **Access the App:** Open a web browser and navigate to http://0.0.0.0:8000 (or localhost:8000).

How to Install and Run Locally(API Routing)

1. **Clone the Repository:** git clone https://github.com/srujan-zenshastra/week1_project3_ThunderClient_API.git cd week1_project3_ThunderClient_API

2. **Install Dependencies:** Use pip to install the required packages: pip install flask flask-sqlalchemy flask-wtf werkzeug Note: This version does not require flask-socketio as it lacks real-time chat (but could be added).
3. **Set Up the Database**: Ensure users.db is in the project root (or run the app once to create it): python app.py
4. **Run the Application:** Execute app.py to start the Flask server: python app.py The app will run on http://0.0.0.0:8002 in debug mode.
5. **Access the API:** Use a tool like Postman, cURL, or Thunder Client to send HTTP requests to endpoints (e.g., POST http://0.0.0.0:8002/register with JSON data).

User Guide

Getting Started

1.) HTML Version

1. **Access the Application:** Open a web browser and navigate to http://0.0.0.0:8000 (or localhost:8000) after running the application.
2. **Register:** Visit index.html or register.html to create an account with a username and password.
3. **Log In:** Use login.html to enter credentials and access the dashboard.
4. **Explore the Dashboard:** Use the dashboard to manage notes, upload files, chat with others, and view messages.

2.) API Version

1. **Set Up a Client:** Use a tool like Postman, cURL, or a custom script (e.g., Python with requests) to interact with the API.
2. **Register:** Send a POST request to http://0.0.0.0:8002/register with JSON data ("username": "string", "password": "string") to create an account.
3. **Log In:** Send a POST request to http://0.0.0.0:8002/login with JSON data to authenticate and receive a session.
4. **Use Features**: Send requests to endpoints like /add_note, /view_note, /edit_note, /delete_note, /upload, and /logout to manage notes and files programmatically.

Using Features

- **Notes (Both Versions):** Create, edit, and delete notes (HTML via forms/links, API via JSON endpoints).
- **Files (Both Versions):** Upload, view, and delete files (HTML via forms/links, API via JSON/file requests).
- **Chat (HTML Only):** Navigate to "Chat" for real-time messaging (not available in API but could be extended).
- **Messages (HTML Only):** View received messages in the dashboard or view_messages.html (API could add /view_messages for JSON responses).