# Acknowledgement

I express my profound gratitude to **Ms. Aliza Raza Rizvi**, Department of Computer Science & Engineering, for her valuable guidance and support. in the preparation of this report on my project, "**KeyScore**". Her insightful advice, continuous encouragement, and constructive feedback have been crucial to the successful completion of this project.

I would also like to extend my heartfelt thanks to the faculty members and support staff of the Computer Science & Engineering Department for creating a supportive learning environment. and providing all the necessary resources that helped facilitate the smooth progress of this project.

Finally, I am deeply thankful to my friends and classmates for their constant support and cooperation, which inspired me to reach my goals and complete this project successfully.

**DATE -** **AYUSH AWASTHI**

**(CR-59)**

# DECLARATION

I, **Ayush Awasthi**, a student of Computer Science & Engineering at **BBDNIIT LUCKNOW**, hereby declare that the project titled **"KeyScore"** is my original work. This project has been developed as a part of my academic curriculum and has not been submitted previously to any other institution or organization for any purpose.

The project is designed to serve as a file management system, allowing users to securely upload, store, and manage their files with ease. It features a user-friendly interface, secure login system, and real-time feedback functionality. The development was carried out using **Python (Flask Framework)**, **HTML**, **CSS**, and **JavaScript**, with **SQLite** used for database management.

I have diligently ensure that the content, code, and design elements of the project adhere to ethical and academic standards. Any references or external resources used during the development process have been properly acknowledged in the report.

I take full responsibility for the originality and authenticity of this work.

Date-

Ayush Awasthi

(CR- 59)

# TABLE OF CONTENT

# INTRODUCTION — KeyScore

The **KeyScore** project is an innovative file management system designed to simplify the way users store, manage, and retrieve their documents. In today's digital age, effective file management is crucial, whether for personal or professional use. Keyscore aims to provide users with a seamless, efficient, and secure way to upload, store, and organize their files online.

The system is built with a user-friendly interface and allows for easy file uploading, viewing, and management. With features such as secure login and real-time file updates, it ensures users can access their files anytime and anywhere securely. The project is designed to handle various file types, including images, documents, and PDFs, ensuring compatibility and versatility for different user needs.

In addition to basic file storage, **Keyscore** offers advanced features like file sorting, search functionalities, and user-specific access, making it a comprehensive solution for users who need an organized and accessible file management system. Users can upload multiple files at once and categorize them with descriptions, making the platform both intuitive and practical for a wide range of applications.

To enhance the user experience, the platform also integrates real-time feedback functionality, which allows users to leave comments or notes on their files. This feature makes collaboration and sharing information easier, especially in environments that require frequent updates.

This project is developed using modern technologies such as **Python (Flask Framework)** for the backend, **HTML**, **CSS**, and **JavaScript** for the frontend, and **SQLite** as the database, ensuring that it is both lightweight and scalable. The system's flexibility allows it to be easily adapted for various use cases, from individual users managing their personal files to organizations requiring secure and structured document storage solutions.

The goal of **Keyscore** is to provide an organized, secure, and efficient platform that streamlines the way users interact with their digital content. With its focus on simplicity and functionality, Keyscore is designed to meet the needs of both casual and power users alike, ensuring an enhanced file management experience.

By prioritizing security, accessibility, and user convenience, **Keyscore** aims to become a reliable tool for personal and professional file management, contributing to a more efficient and organized digital workflow.

**KeyScore: Empowering You to Achieve More**

---

# OBJECTIVE OF PROJECT

The **KeyScore** project aims to create a comprehensive, user-friendly, and secure platform that simplifies file management for individuals and organizations. With the increasing reliance on digital content, there is a pressing need for an efficient system to store, organize, and access files securely. This project addresses these needs by offering a variety of features designed to streamline file management, improve collaboration, and ensure data security.

**1. Simplified File Management**

**KeyScore** simplifies file organization by providing an intuitive platform for users to upload, categorize, and retrieve files easily. The system enables users to organize files into folders, label them meaningfully, and quickly search for specific documents. This simplicity is designed to eliminate the complexity of traditional file management systems, making it accessible to users with varying levels of technical expertise.

**2. Security and Privacy**

Security is a key feature of **KeyScore**. The platform employs strong authentication methods and encryption techniques to ensure that files are stored securely and accessed only by authorized users. With features like secure logins and encrypted file storage, **KeyScore** ensures that users' sensitive data remains protected from unauthorized access.

**3. Real-Time File Updates**

**KeyScore** supports real-time file updates, ensuring that changes made to files or folders are instantly reflected across the platform. This feature is particularly useful in collaborative environments, allowing multiple users to work on shared files and ensuring that everyone has access to the latest versions of documents.

**4. Collaboration and Sharing**

Collaboration is at the heart of **KeyScore**. Users can share files with specific individuals or groups, controlling access permissions such as read-only or editable access. The platform also allows users to comment or annotate files, facilitating collaboration and ensuring that teams can work efficiently on shared projects.

**5. User-Centric Interface**

The user interface of **KeyScore** is designed with simplicity and functionality in mind. The platform offers a clean and responsive design that works seamlessly across various devices, from desktop

computers to mobile phones. This ensures that users can manage their files efficiently, regardless of the device they are using.

**6. Powerful Search and Sorting Features**

**KeyScore** offers a robust search function, enabling users to find files quickly based on various parameters such as file name, type, or upload date. Additionally, files can be sorted by different criteria, helping users organize their content efficiently and save time when searching for specific documents.

**7. Scalability and Performance**

As the platform grows, **KeyScore** is designed to handle increased user traffic and larger volumes of files without compromising performance. The backend system is optimized for scalability, ensuring fast file uploads, downloads, and searches even as the number of users and files increases.

**8. Cross-Platform Accessibility**

To ensure maximum accessibility, **KeyScore** is built to work seamlessly across different platforms. Whether users are accessing the system on a desktop computer, tablet, or smartphone, they will enjoy the same high-quality user experience, making it a versatile solution for file management.

**9. Cross-Platform Accessibility and Retrieval from Anywhere**

A key objective of **KeyScore** is to provide users the ability to **log in and retrieve their files from anywhere**. Regardless of the device or location, users will have continuous access to their files, thanks to the platform's cloud-based solution. Whether at home, in the office, or while traveling, users can securely access their data from any internet-enabled device, making **KeyScore** a truly portable solution for file management.

**10. Ethical File Management**

Finally, **KeyScore** promotes ethical file management by encouraging users to adhere to copyright laws and academic integrity. The platform offers features that help users properly attribute sources and manage their files responsibly, contributing to a culture of ethical digital content management.

---

# FEATURES OF PROJECT

**KeyScore** is a web-based application designed to manage users' files securely. Built using Python with the Flask framework, **KeyScore** provides users with a secure platform for uploading, organizing, and retrieving their files anytime, anywhere. With functionalities such as user authentication, file management, and email notifications, **KeyScore** ensures a user-friendly experience and optimal performance for file storage and retrieval.

---

1. **User Authentication & Registration:**
    - **Sign Up & Login:** Allows users to create accounts and log in securely to access their files.
    - **Session Management:** Maintains user sessions to keep track of logged-in users and ensure that only authenticated users can upload, download, or delete files.
    - **Secure Passwords:** Ensures password security through proper validation and storage practices.

---

2. **File Upload & Storage:**
    - **Multi-file Upload:** Users can upload multiple files at once. The files can be images, PDFs, and more.
    - **File Validation:** Only valid file types (JPEG, PNG, GIF, PDF) are allowed for upload. Invalid files are rejected with a proper error message.
    - **Secure File Saving:** Files are securely stored on the server with sanitized filenames to prevent security vulnerabilities.

---

3. **File Management:**
    - **File Organization:** Users can store files in an organized manner and view them on the dashboard. Files are displayed with metadata such as upload time.
    - **File Sorting:** Files can be sorted by name or upload time in ascending or descending order.
    - **Folder Creation:** Users can create folders to organize their files more efficiently.

---

4. **File Retrieval:**
    - **Download Files:** Users can easily download their uploaded files from the server whenever needed.
    - **View Files:** The uploaded files are displayed on the dashboard, where users can view their names and other details.

---

5. **File Deletion:**
    - **Delete Files:** Users have the option to delete files that are no longer needed from their account, keeping their file storage organized.

---

6. **Email Notifications:**
    - **Feedback Form:** Users can submit feedback about the platform, which is then sent via email to the administrator.
    - **Automated Emailing:** Emails are automatically generated using Flask-Mail to send feedback or alert the admin of new feedback.

---

7. **Feedback Management:**
    - **User Feedback Form:** A simple feedback form allows users to rate their experience and leave comments, which can help improve the platform.
    - **Email Alerts:** After submitting feedback, an email is sent to a designated address, alerting the administrator.

---

8. **Dashboard:**
    - **User Dashboard:** Upon logging in, users are redirected to a dashboard displaying the list of their uploaded files.
    - **File Overview:** The dashboard provides an overview of files, including file names, upload time, and file paths.

---

9. **Responsive User Interface:**
    - **Web Interface:** Designed using HTML, CSS, and JavaScript for easy navigation and a seamless user experience.
    - **Responsive Design:** The interface is fully responsive, ensuring it works well on different devices like desktops, tablets, and mobile phones.

---

10. **Security Features:**
    - **Session Security:** User sessions are managed with Flask's session handling, ensuring data privacy and secure access.
    - **File Path Security:** All uploaded files are stored in a specific directory with a secure file path to prevent access to unauthorized files.

---

11. **Folder Structure:**
    - **Folder Creation:** Users can create and manage folders to store and categorize their files efficiently.
    - **Dynamic Folder Management:** The platform supports dynamic folder creation and organization without the need for manual directory setup.

---

12. **Help & Support:**
    - **Help Page:** Users can access a help page for assistance regarding the platform's functionality, providing guidance on how to upload files, use the dashboard, and manage their accounts.

---

13. **Customizable Features:**
    - **Email Configuration:** Admin can configure the email settings in the app for sending feedback notifications and alerts using their own SMTP server.
    - **File Sorting Options:** Users can choose to sort their files by name, date, or upload time, providing flexibility in how files are displayed.

---

14. **Data Management:**

    - **Database Integration:** The application uses SQLite for storing user credentials and file metadata. This ensures efficient data storage and retrieval.
    - **File Metadata:** Metadata like file names, paths, upload time, and descriptions are stored in the database, allowing easy access to file details.

---

15. **Secure File Access:**

    - **File Download:** Users can securely download files they uploaded, ensuring only the logged-in user has access to their files.
    - **Access Control:** Only authenticated users are allowed to upload, download, or delete files, ensuring security and privacy.

---

16. **Error Handling:**

    - **Flash Messages:** The platform uses Flask's `flash` method to display success or error messages, keeping users informed about the status of their actions.
    - **Error Logging:** If any errors occur, they are logged and displayed to the user in a friendly manner, ensuring a smooth user experience.

---

# TECHNOLOGY OF PROJECT

**KeyScore** is developed using modern web technologies to provide users with a seamless and interactive platform for managing and storing their files. The following outlines the hardware and software requirements for the project:

---

## 1. Hardware Requirements

**For End-Users (Users accessing the KeyScore platform):**

1. **Device**: Desktop, laptop, tablet, or smartphone with internet access.
2. **Processor**:
   - Minimum: 1 GHz processor
   - Recommended: Dual-core or higher for better performance.
3. **RAM**:
   - Minimum: 2 GB
   - Recommended: 4 GB or more for a smoother experience.
4. **Storage**:
   - At least 100 MB free space for storing user data and browser cache.
5. **Display**:
   - Minimum screen resolution of **1024x768 pixels** for an optimal viewing experience.
6. **Internet Connection**:
   - Minimum speed: **2 Mbps** for smooth access to files and real-time data.

**For Developers (During development of KeyScore)**

1. **Processor**:
   - Recommended: Intel Core i5 or higher (for smooth development experience).
2. **RAM**:
   - Minimum: 8 GB (16 GB recommended for heavy multitasking).
3. **Storage**:
   - At least **500 MB** free space for project files and dependencies.
4. **Display**:
   - Full HD (1920x1080) resolution for better clarity while coding and designing.
5. **Internet Connection**:
   - High-speed broadband (for testing, hosting, and project deployment).

## 2. Software Requirements

**For End-Users:**

1. **Web Browser**:
    - **Google Chrome** (Version 90 or above)
    - **Mozilla Firefox** (Version 85 or above)
    - **Microsoft Edge** (Version 85 or above)
    - **Safari** (Version 14 or above)
2. **Operating System**:
    - **Windows**: 7, 8, 10, 11
    - **macOS** (Version 10.12 or above)
    - **Android** (Version 7.0 or above)
    - **iOS** (Version 11.0 or above)

**For Developers:**

1. **Operating System**:
    - **macOS** (Preferred for development, version 10.14 or above)
    - **Windows 10/11**
    - **Linux (Ubuntu 20.04 or above)**
2. **Code Editors/IDEs**:
    - **Visual Studio Code** (Recommended for its support with Python, HTML, CSS, and JavaScript)
    - **Sublime Text** or **Atom** (Alternative text editors)
3. **Development Tools & Libraries**:
    - **Flask**: Used for backend development (Python-based web framework).
    - **HTML5**: For creating the structure of the web pages.
    - **CSS3**: For styling the KeyScore platform and making it responsive.
    - **JavaScript**: For frontend interactivity and dynamic content.
    - **SQLite**: For local database storage and user authentication.
    - **Werkzeug**: Used for handling file uploads securely.
    - **Pillow (PIL)**: For image handling (in case of profile pictures or other media).

- **PyPDF2**: For handling PDF file operations (e.g., reading, uploading PDFs).
- **Flask-Mail**: For sending feedback and notifications via email.

4. **Version Control**:
   - **Git**: For managing project versions and code collaboration (if multiple developers are involved).
5. **Web Hosting/Deployment**:
   - **Heroku**: For deploying the KeyScore platform online.
   - **GitHub**: For code storage and collaboration.
6. **Testing Tools**:
   - **Chrome DevTools** (For debugging web applications)
   - **Lighthouse** (For performance and accessibility testing)
   - **Postman** (For API testing)

---

# HTML

HTML (HyperText Markup Language) is the standard language used to structure content on the web. It forms the backbone of web development and provides a framework for creating web pages. It consists of a series of elements, commonly known as "tags," that tell the browser how to display text, images, and other content.

---

## Key Features of HTML

1. **Markup Language**:
    - HTML (HyperText Markup Language) is a markup language used to create and structure content on the web. It uses a system of tags to define elements on a webpage.
2. **Platform-Independent**:
    - HTML is platform-independent, meaning that web pages created with HTML can be viewed on any device (computers, tablets, smartphones) with a web browser.
3. **Support for Multimedia**:
    - HTML5 introduced native support for audio, video, and graphics, allowing developers to embed multimedia content easily without relying on third-party plugins.
4. **Semantic Elements**:
    - HTML5 includes semantic tags like `<header>`, `<footer>`, `<article>`, `<section>`, and `<nav>` to make content more meaningful and enhance search engine optimization (SEO).
5. **Forms and Input Handling**:
    - HTML provides robust support for creating interactive forms with input elements like text boxes, radio buttons, checkboxes, and submit buttons to collect user input.
6. **Hyperlinks**:
    - HTML enables the creation of hyperlinks (`<a>` tag) to link to other web pages, websites, or resources, making navigation easy.
7. **Tables and Lists**:
    - HTML allows you to display tabular data using the `<table>` element and organize content in ordered (`<ol>`) or unordered (`<ul>`) lists.
8. **Extensibility**:
    - HTML can be extended with custom attributes, making it highly adaptable to different needs.
9. **Interactivity**:

- With JavaScript integration, HTML can enable interactive elements like buttons, forms, and dynamic content.

**Basic Structure of an HTML Document**

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to Alpha World</title>
</head>
<body>
<h1>Hello, World!</h1>
<p>Making the transition for the youth of the world smooth to live a successful life</p>
<a href="https://example.com">Learn More</a>
</body>
</html>
```

**Explanation of the Structure**:

- **`<!DOCTYPE html>`**: Declares the document type and tells the browser that the document is written in HTML5.
- **`<html>`**: The root element that wraps all the content of the webpage.
- **`<head>`**: Contains metadata like the character set, viewport settings, and the title of the webpage. This section is not visible to users but is important for search engines and proper rendering.
- **`<body>`**: Contains the visible content of the webpage, including headings, paragraphs, images, links, and more.

**Importance of HTML in Web Development**

- **Foundation of Web Pages**: Structures content with elements like headings, paragraphs, images, and links.
- **Cross-Browser Compatibility**: Ensures websites work across different browsers and devices.
- **Semantic Structure**: Helps search engines and users understand the content through meaningful tags.
- **Accessibility**: Makes websites accessible to all users, including those with disabilities.
- **Integration with CSS & JavaScript**: Works alongside CSS for styling and JavaScript for interactivity.
- **Search Engine Optimization (SEO)**: Improves search engine ranking through proper tag use and content structure.
- **Content Management**: Organizes and updates content efficiently.
- **Multimedia Integration**: Supports embedding images, videos, and audio.
- **Mobile Responsiveness**: Ensures websites are optimized for mobile devices.
- **Enhances UX**: Organizes content for better user experience and usability.
- **Dynamic Content**: Allows creation of interactive forms and user inputs.
- **Easy to Learn**: Simple syntax with a short learning curve, ideal for beginners.

---

# CSS

CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation and styling of a document written in HTML or XML. It allows web developers to control the look and feel of web pages by defining how HTML elements should appear. CSS is responsible for the layout, colors, fonts, spacing, and overall aesthetic of a webpage.

---

## Key Features of CSS

1. **Separation of Content and Style**:
    - CSS allows you to separate the content (HTML) from the presentation (style), making the structure of the document cleaner and easier to maintain.
2. **Styling Control**:
    - CSS gives precise control over the layout, fonts, colors, spacing, and positioning of HTML elements on the page.
3. **Reusability**:
    - Styles defined in a CSS file can be reused across multiple HTML pages. This makes it easy to apply consistent design throughout a website.
4. **Responsive Design**:
    - CSS enables responsive design by allowing developers to use media queries, making it easier to design websites that work on different devices (desktops, tablets, smartphones).
5. **Cascading and Inheritance**:
    - The "cascading" nature of CSS means that styles are applied based on specificity. CSS rules can be inherited from parent elements to child elements, reducing the need for redundant code.
6. **Visual Effects**:
    - CSS supports various visual effects such as animations, transitions, and transformations to create interactive and dynamic webpages.
7. **Customizability**:
    - With CSS, web pages can be easily customized by changing colors, font styles, background images, and much more, without altering the underlying HTML structure.
8. **Cross-browser Compatibility**:
    - CSS is designed to work across multiple browsers (Chrome, Firefox, Safari, Edge, etc.), although certain features might need workarounds to ensure consistent behavior.

**Types of CSS:**

1. **Inline CSS**

- **Definition**: Inline CSS is applied directly within an HTML element using the `style` attribute.
- **Syntax:**

```
<h1 style="color: blue; text-align: center;">Hello,
World!</h1>
```

**2. Internal CSS**

- **Definition**: Internal CSS is written inside the `<style>` tag within the `<head>` section of the HTML document.
- **Syntax**:

```
 <head>
<style>
    h1 {
        color: blue;
        text-align: center;
    }
</style>
</head>
<body>
  <h1>Hello, World!</h1>
</body>
```

**External CSS**

- **Definition**: External CSS is defined in a separate `.css` file, which is linked to an HTML document. This is the most efficient and scalable method of using CSS.
- **Syntax**:
  **CSS File** (`styles.css`):`css`

```
h1 {
    color: blue;
    text-align: center;}
```

**HTML File**:`html`

```
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <h1>Hello, World!</h1>
</body>
```

# JAVASCRIPT

JavaScript is a high-level, interpreted programming language primarily used for creating interactive and dynamic content on web pages. It is one of the core technologies of web development, alongside HTML and CSS, and is essential for building modern web applications. JavaScript allows developers to manipulate HTML and CSS, control multimedia, animate images, handle user interactions, and communicate with remote servers, all without refreshing the web page.

---

## Key Features of JavaScript:

1. **Client-Side Scripting**:
    - JavaScript runs directly in the browser, reducing server load and improving speed.
2. **DOM Manipulation**:
    - JavaScript allows dynamic modification of HTML and CSS on the page in response to user actions.
3. **Asynchronous Programming**:
    - JavaScript uses asynchronous functions like AJAX and Promises to handle tasks like fetching data without reloading the page.
4. **Dynamic Typing**:
    - Variables in JavaScript do not require a predefined type, allowing greater flexibility.
    - ed to be lightweight and efficient for quick execution on the client-side.
5. **Object-Oriented**:
    - JavaScript supports object-oriented programming (OOP) principles, allowing the creation of objects with properties and methods.
6. **Versatility**:
    - JavaScript can be used for both front-end and back-end development (with Node.js), enabling full-stack development.

## Applications of JavaScript:

1. **Web Development**:

- It is the core technology for web development, working with HTML and CSS to create responsive, interactive websites.

2. **Mobile Applications**:

- JavaScript can be used to develop mobile apps using frameworks like React Native, Ionic, and Cordova. These frameworks allow code sharing between web and mobile platforms.

# PYTHON (FLASK)

Flask is a lightweight and flexible web framework written in Python, commonly used for building web applications and APIs. It is designed to be simple and easy to use, providing just the essentials for web development while allowing developers to add functionality through extensions as needed.

---

## Key Features of Flask:

- **Lightweight & Minimalistic**: Flask comes with minimal built-in components, allowing developers to add only the features they need, making it fast and efficient.
- **Flexibility**: Flask's modularity enables developers to choose the tools and libraries that best fit their project needs, making it highly adaptable for various use cases.
- **Built-in Development Server**: Flask comes with an integrated web server for testing and debugging, facilitating rapid development and debugging of web applications.
- **Jinja2 Template Engine**: Flask uses Jinja2, a powerful templating engine, to generate dynamic content in HTML templates, providing a better user experience.
- **RESTful API Support**: Flask is particularly useful for building RESTful APIs. It provides simple and efficient ways to handle HTTP methods like GET, POST, and PUT.
- **Extensibility**: Flask can be easily extended with third-party libraries, enabling developers to integrate advanced features like database connections, user authentication, and form validation.

## Structure of a Flask Application:

1. **Routing**: Flask routes map URLs to specific functions in your application, allowing for easy handling of HTTP requests.
2. **Request Handling**: Flask handles incoming requests and generates appropriate responses, including HTML, JSON, or files.
3. **Templates**: Flask uses the Jinja2 templating engine to generate dynamic web pages, allowing Python code to be embedded directly within HTML.

## Advantages of Flask:

- **Rapid Prototyping**: Flask allows for quick development of applications, making it ideal for prototyping and small-scale projects.
- **Great for APIs**: Flask is often used for creating RESTful APIs due to its simplicity and flexibility.
- **Minimal Configuration**: With Flask, developers can quickly get started without the need for extensive configuration or boilerplate code.

## Applications:

1. **Web Applications**: Flask is commonly used for building web applications that require custom functionality and lightweight architecture.
2. **APIs**: Flask's simplicity and scalability make it a popular choice for creating APIs for web, mobile, or IoT applications.

---

# DESIGN MODEL

## 1. User Interface Design (UI)

**Landing Page**

- **Description**: The entry point of the platform, offering links to **Login**, **Signup**, and **Help** pages. It's designed to be simple and welcoming.
- **UI Elements**: Links for **Login**, **Signup**, **Help**, and a welcoming message to guide users into the platform.

**Login Page**

- **Description**: Allows users to authenticate with their email and password to access their dashboard.
- **UI Elements**: Input fields for **email** and **password**, **Login button**, and error messages for invalid credentials. User-friendly messages guide the user for corrections.

**Signup Page**

- **Description**: New users can register by providing **username**, **email**, **password**, and **confirm password**. The page is equipped with validation to ensure proper data submission.
- **UI Elements**: Input fields, **Submit button**, and validation messages for errors like **weak password** or **email format issues**.

**Dashboard Page**

- **Description**: Displays the user's uploaded files, including options to **Upload**, **Download**, and **Delete** files.
- **UI Elements**: List of recent files, **Upload button**, sorting options (ascending, descending), and file action buttons like **Download** and **Delete** to manage files.

**File Upload Page**

- **Description**: Users can upload multiple files and provide a description for each file.
- **UI Elements**: File selector input, **Description** field for additional context, and **Upload button** to send the files to the server.

**Feedback Page**

- **Description**: A page for users to provide feedback by submitting a **rating** and **comments** about the platform’s services.
- **UI Elements**: Rating system, **Comment box** for users to write their thoughts, and **Submit button** for sending feedback.

## 2. Backend Logic Design

**User Authentication**

- **Signup**: Validates user data, ensures email uniqueness, hashes the password securely, and stores the data in the **users** table. This process includes error handling for already existing users.

- **Login**: Authenticates users using the stored password hash, sets up a session, and allows access to the dashboard after a successful login.
- **Session Management**: Uses Flask's built-in **session** to store and track logged-in user details (e.g., user_id, username).

**File Management**

- **Upload**: Users can upload files (images, PDFs, etc.). The backend saves the file in a specified folder and stores file metadata (file name, path) in the **files** table for easy retrieval.
- **Download**: Files can be downloaded from the server. The backend ensures the correct file path is used for safe downloads.
- **Delete**: Users can delete files from the platform, which removes both the file from the server and its entry in the database to maintain integrity.

**Database Design**

- **users Table**: Stores **id**, **username**, **email**, and **password**. Ensures uniqueness for **email**.
- **files Table**: Stores file information such as **id**, **user_id** (for foreign key), **file_name**, **file_path**, **upload_time**, and **description**.
- **feedback Table** (optional): Could store **id**, **user_id**, **rating**, and **comments**, allowing admins to review user feedback.

**Email System**

- Sends feedback via **Flask-Mail** to notify the admin of user feedback. This feature enhances user engagement by providing a direct channel to communicate with the admin.

**Error Handling**

- The system displays **validation messages** for missing fields, incorrect file types, and login issues. If the upload fails, the user receives feedback, ensuring a better user experience.

## 3. Data Flow

1. **User Flow**:
    - **Start** → User lands on the **Landing Page** → User clicks **Login** or **Signup** → Registers/logs in → Redirected to **Dashboard** where files are displayed and can be managed.
2. **File Management Flow**:
    - **Dashboard Page** → User uploads files → Files are saved on the server → File metadata is stored in the database → Files appear on the dashboard for easy access.
    - Users can delete or download the files using action buttons available for each file.
3. **Feedback Flow**:
    - **Feedback Page** → User submits their **rating** and **comments** → Email is sent to the admin using the **Flask-Mail** system → User is redirected to the **Thank You Page** with a message confirming the submission.

## 4. System Architecture

- **Frontend (UI)**: The user-facing part of the web app, which includes the **login**, **signup**, **file upload**, **dashboard**, and **feedback** pages. These pages are built using **HTML**, **CSS**, and **JavaScript** for dynamic behavior.
- **Backend (Flask App)**: Handles HTTP requests, serves the UI, performs database operations, and manages file uploads. Flask's **Jinja2** templating engine is used to render dynamic content.
- **Database (SQLite)**: A lightweight relational database used to store user data, files, and feedback. It ensures data consistency and makes retrieval efficient.
- **File Storage**: The **uploads folder** on the server holds user files, which are accessible for download or deletion.

## 5. Interaction Flow

1. **User Registration**:
    - User enters their details → Backend checks for existing emails → If the email is unique, the user is saved in the database → Redirected to Login page.
2. **User Login**:
    - User provides credentials → Backend validates and creates a session → Redirected to Dashboard if successful.
3. **File Upload**:
    - User selects and uploads files → Backend stores them securely on the server → File metadata (name, path) is stored in the database for later access.
4. **Feedback Submission**:
    - User provides feedback → Feedback is sent to admin via email using **Flask-Mail** → User is redirected to **Thank You Page**.

## 6. Security and Error Handling

- **Password Hashing**: To ensure secure password storage, passwords are hashed before being stored in the database using **bcrypt** or **werkzeug.security**.
- **File Validation**: The backend ensures only allowed file types (e.g., images, PDFs) are uploaded.
- **Session Management**: Sensitive actions like uploading or deleting files are restricted to logged-in users only, ensuring security.

---

# REQUIREMENT ANALYSIS

**Overview**:
KeyScore is a Flask-based web application for users to manage their files. It supports user registration, login, file uploads, folder management, and feedback submission. The system stores files and user data in an SQLite database and sends feedback notifications via email.

**Features**:

1. **User Management**:
    - **Registration**: Users create accounts with username, email, and password.
    - **Login**: Users log in with their credentials.
    - **Session Management**: Users remain logged in until they logout, which clears the session.
2. **File Uploads & Management**:
    - **Upload**: Users can upload files (images, PDFs).
    - **Delete**: Users can delete their files.
    - **Download**: Users can download their uploaded files.
    - **Sorting**: Files can be sorted by name or upload time.
3. **Folder Management**:
    - **Create Folders**: Users can create folders to organize files.
4. **Feedback**:
    - Users can submit feedback, which is sent via email using Flask-Mail.

**Technology Stack**:

- **Backend**: Flask (Python)
- **Database**: SQLite
- **Libraries**: PIL (image handling), PyPDF2 (PDF handling), Flask-Mail (email)
- **File Storage**: Local server storage

**Security**:

- Passwords should be hashed before storage (currently plain text, which is insecure).
- File uploads are validated to ensure allowed file types.

**UI Pages**:

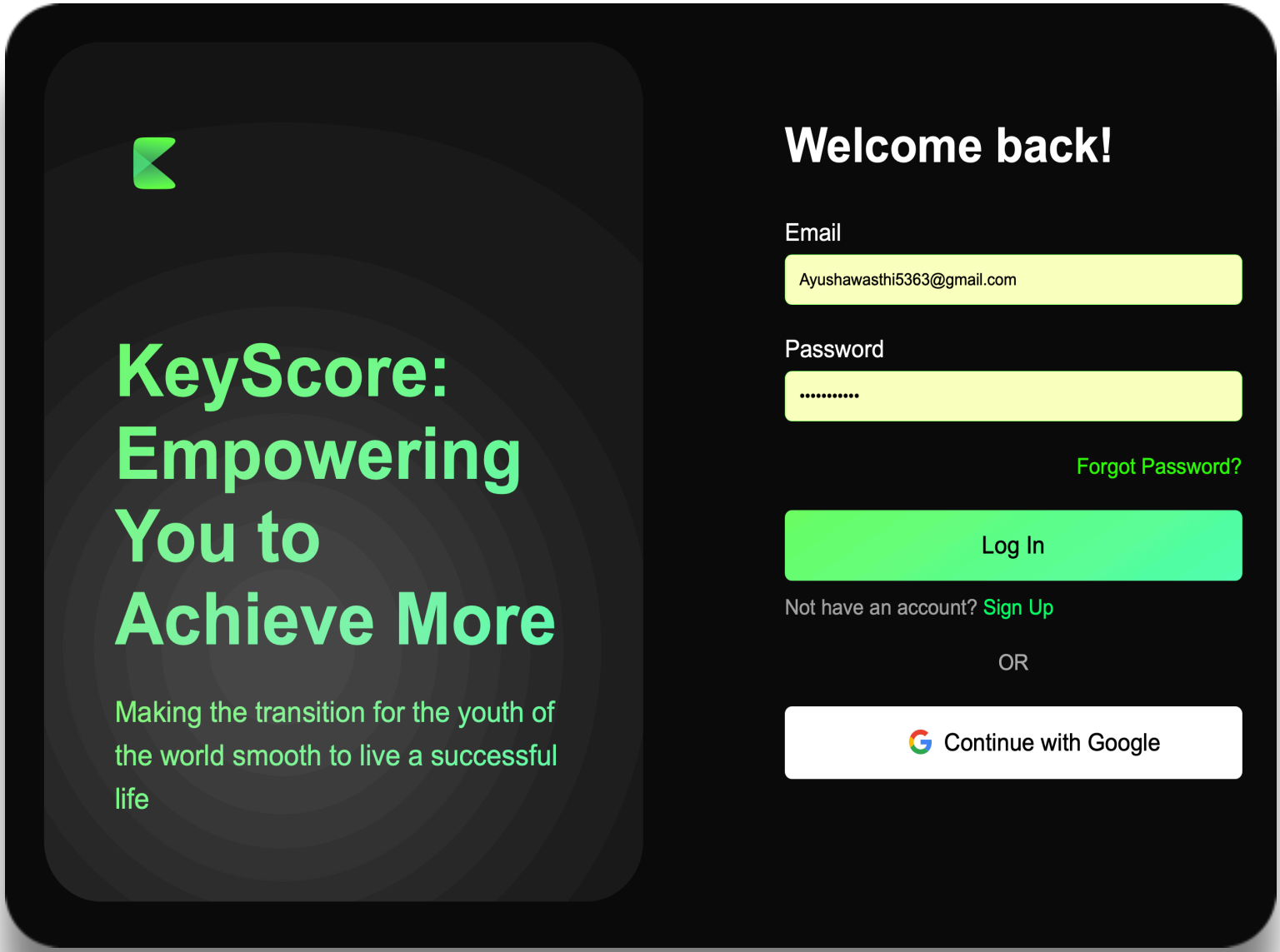
- **Login**: For user authentication.
- **Signup**: For user registration.
- **Dashboard**: Displays uploaded files.
- **Feedback**: Allows users to submit feedback.
- **Upload**: For file uploads.

**Enhancements**:

- Implement password hashing for security.
- Add file previews for images and PDFs.
- Introduce a file search feature.

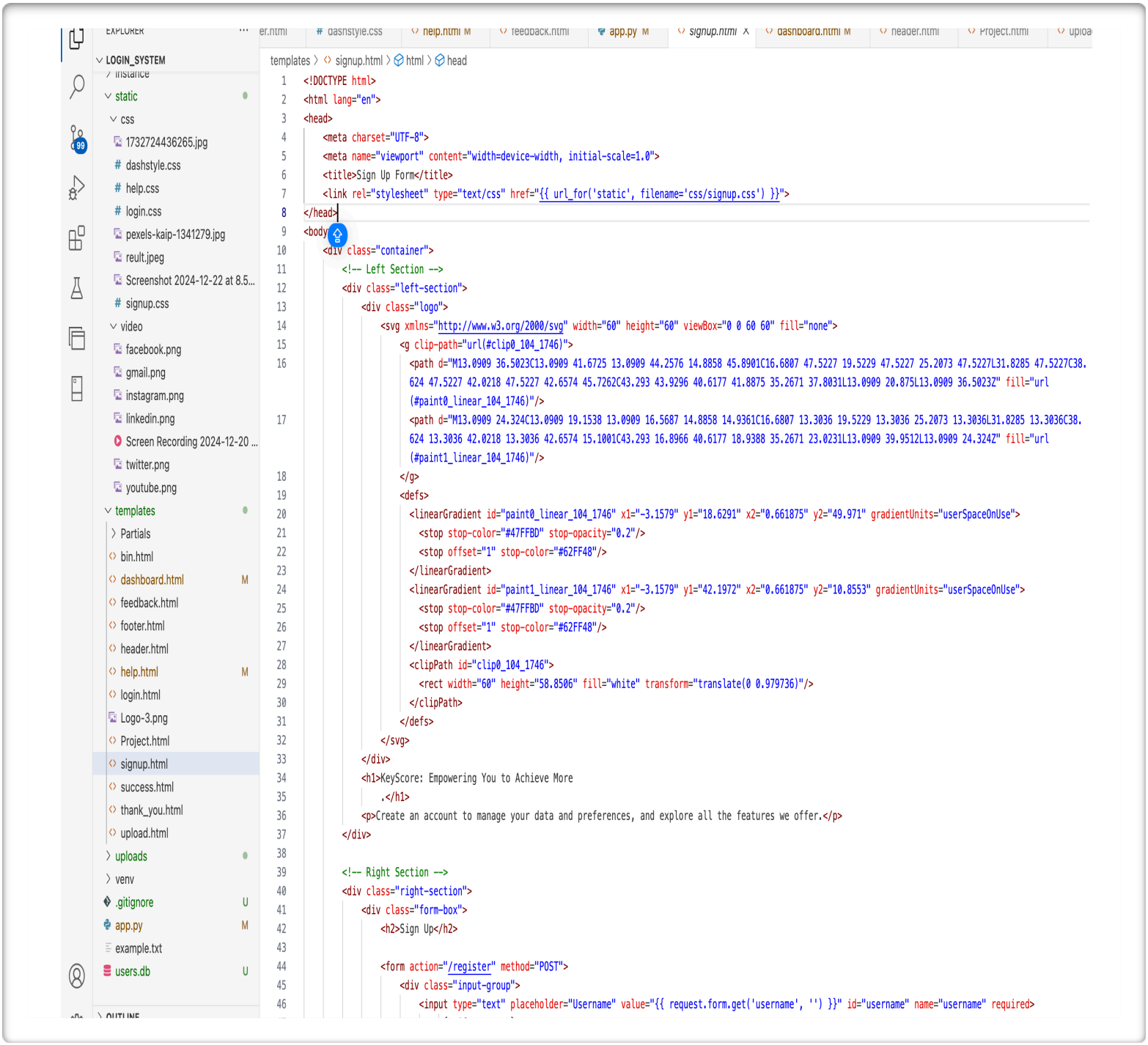
# DESIGN PHASE OF PROJECT

## FIRST LOOK

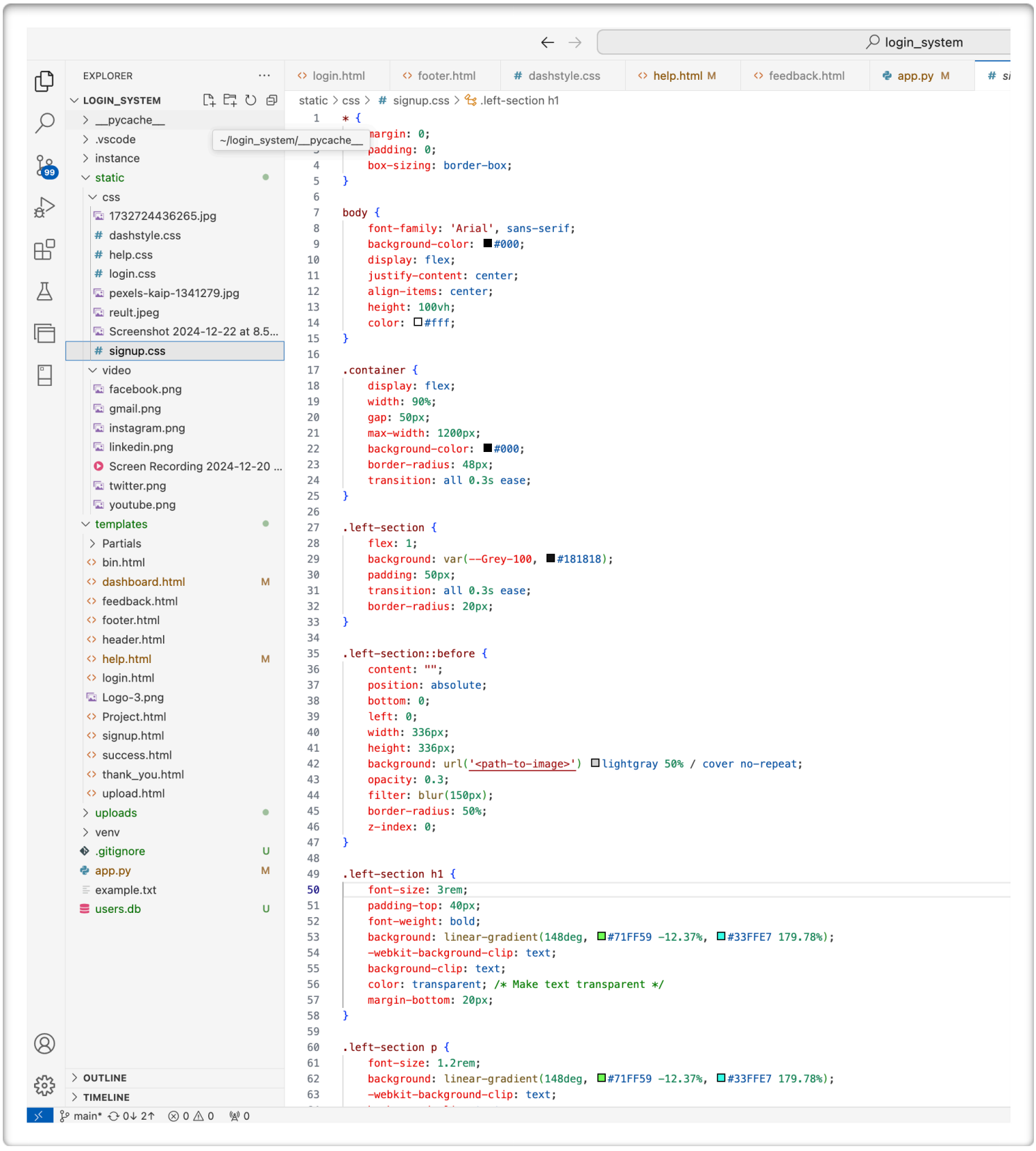


# ALGORITHM

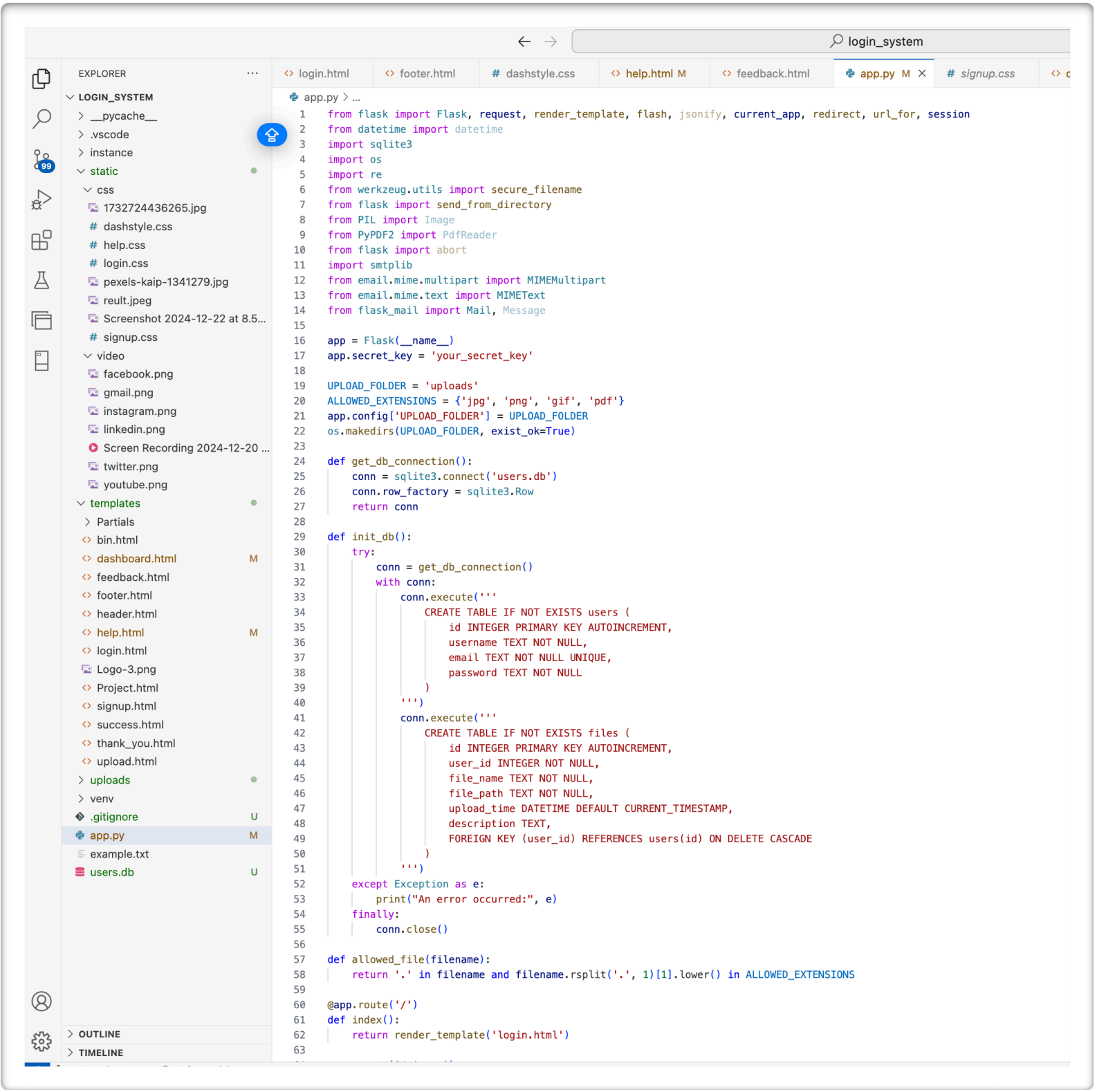
SIGNUP.HTML



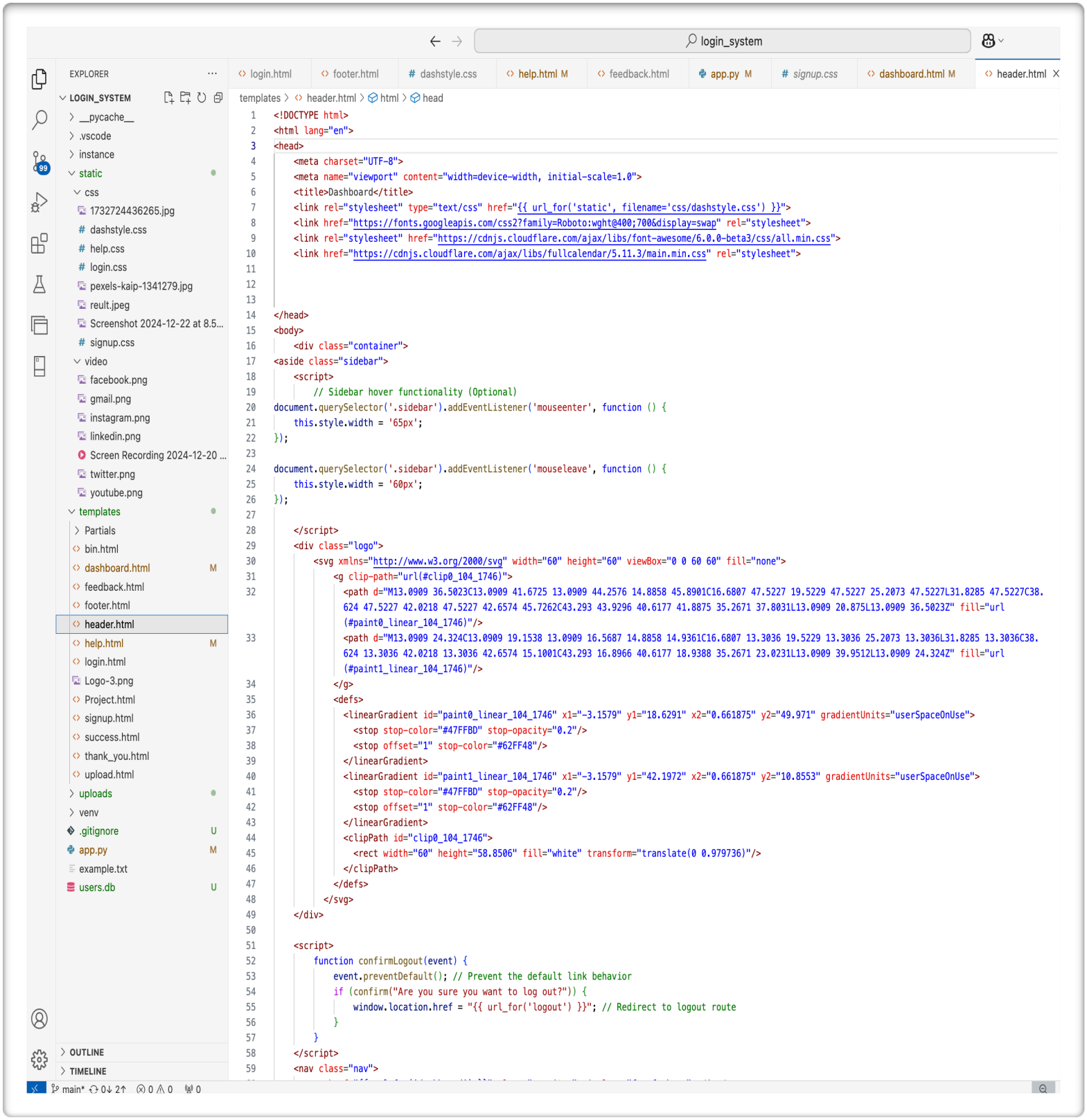
# SIGNUP.CSS


login_system
EXPLORER
LOGIN_SYSTEM
__pycache__
~/login_system/__pycache__
.vscode
instance
static
css
1732724436265.jpg
dashstyle.css
help.css
login.css
pexels-kaip-1341279.jpg
reult.jpeg
Screenshot 2024-12-22 at 8.5...
signup.css
video
facebook.png
gmail.png
instagram.png
linkedin.png
Screen Recording 2024-12-20 ...
twitter.png
youtube.png
templates
Partials
bin.html
dashboard.html M
feedback.html
footer.html
header.html
help.html M
login.html
Logo-3.png
Project.html
signup.html
success.html
thank_you.html
upload.html
uploads
venv
.gitignore U
app.py M
example.txt
users.db U
OUTLINE
TIMELINE
login.html
footer.html
dashstyle.css
help.html M
feedback.html
app.py M
static > css > # signup.css > .left-section h1
* {
margin: 0;
padding: 0;
box-sizing: border-box;
}
body {
font-family: 'Arial', sans-serif;
background-color: #000;
display: flex;
justify-content: center;
align-items: center;
height: 100vh;
color: #fff;
}
.container {
display: flex;
width: 90%;
gap: 50px;
max-width: 1200px;
background-color: #000;
border-radius: 48px;
transition: all 0.3s ease;
}
.left-section {
flex: 1;
background: var(--Grey-100, #181818);
padding: 50px;
transition: all 0.3s ease;
border-radius: 20px;
}
.left-section::before {
content: "";
position: absolute;
bottom: 0;
left: 0;
width: 336px;
height: 336px;
background: url('<path-to-image>') lightgray 50% / cover no-repeat;
opacity: 0.3;
filter: blur(150px);
border-radius: 50%;
z-index: 0;
}
.left-section h1 {
font-size: 3rem;
padding-top: 40px;
font-weight: bold;
background: linear-gradient(148deg, #71FF59 -12.37%, #33FFE7 179.78%);
-webkit-background-clip: text;
background-clip: text;
color: transparent; /* Make text transparent */
margin-bottom: 20px;
}
.left-section p {
font-size: 1.2rem;
background: linear-gradient(148deg, #71FF59 -12.37%, #33FFE7 179.78%);
-webkit-background-clip: text;
main* 0↓ 2↑ 0 0 0

# APP.PY


login_system
EXPLORER
LOGIN_SYSTEM
__pycache__
.vscode
instance
static
css
1732724436265.jpg
dashstyle.css
help.css
login.css
pexels-kaip-1341279.jpg
reult.jpeg
Screenshot 2024-12-22 at 8.5...
signup.css
video
facebook.png
gmail.png
instagram.png
linkedin.png
Screen Recording 2024-12-20 ...
twitter.png
youtube.png
templates
Partials
bin.html
dashboard.html M
feedback.html
footer.html
header.html
help.html M
login.html
Logo-3.png
Project.html
signup.html
success.html
thank_you.html
upload.html
uploads
venv
.gitignore U
app.py M
example.txt
users.db U
OUTLINE
TIMELINE
login.html
footer.html
dashstyle.css
help.html M
feedback.html
app.py M
signup.css
app.py > ...
1 from flask import Flask, request, render_template, flash, jsonify, current_app, redirect, url_for, session
2 from datetime import datetime
3 import sqlite3
4 import os
5 import re
6 from werkzeug.utils import secure_filename
7 from flask import send_from_directory
8 from PIL import Image
9 from PyPDF2 import PdfReader
10 from flask import abort
11 import smtplib
12 from email.mime.multipart import MIMEMultipart
13 from email.mime.text import MIMEText
14 from flask_mail import Mail, Message
15
16 app = Flask(__name__)
17 app.secret_key = 'your_secret_key'
18
19 UPLOAD_FOLDER = 'uploads'
20 ALLOWED_EXTENSIONS = {'jpg', 'png', 'gif', 'pdf'}
21 app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
22 os.makedirs(UPLOAD_FOLDER, exist_ok=True)
23
24 def get_db_connection():
25 conn = sqlite3.connect('users.db')
26 conn.row_factory = sqlite3.Row
27 return conn
28
29 def init_db():
30 try:
31 conn = get_db_connection()
32 with conn:
33 conn.execute('''
34 CREATE TABLE IF NOT EXISTS users (
35 id INTEGER PRIMARY KEY AUTOINCREMENT,
36 username TEXT NOT NULL,
37 email TEXT NOT NULL UNIQUE,
38 password TEXT NOT NULL
39 )
40 ''')
41 conn.execute('''
42 CREATE TABLE IF NOT EXISTS files (
43 id INTEGER PRIMARY KEY AUTOINCREMENT,
44 user_id INTEGER NOT NULL,
45 file_name TEXT NOT NULL,
46 file_path TEXT NOT NULL,
47 upload_time DATETIME DEFAULT CURRENT_TIMESTAMP,
48 description TEXT,
49 FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
50 )
51 ''')
52 except Exception as e:
53 print("An error occurred:", e)
54 finally:
55 conn.close()
56
57 def allowed_file(filename):
58 return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
59
60 @app.route('/')
61 def index():
62 return render_template('login.html')
63

# HEADER.HTML


login_system
EXPLORER
LOGIN_SYSTEM
__pycache__
.vscode
instance
static
css
1732724436265.jpg
dashstyle.css
help.css
login.css
pexels-kaip-1341279.jpg
reult.jpeg
Screenshot 2024-12-22 at 8.5...
signup.css
video
facebook.png
gmail.png
instagram.png
linkedin.png
Screen Recording 2024-12-20 ...
twitter.png
youtube.png
templates
Partials
bin.html
dashboard.html M
feedback.html
footer.html
header.html
help.html M
login.html
Logo-3.png
Project.html
signup.html
success.html
thank_you.html
upload.html
uploads
venv
.gitignore U
app.py M
example.txt
users.db U
OUTLINE
TIMELINE
login.html
footer.html
dashstyle.css
help.html M
feedback.html
app.py M
signup.css
dashboard.html M
header.html
templates > header.html > html > head
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Dashboard</title>
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/dashstyle.css') }}">
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
<link href="https://cdnjs.cloudflare.com/ajax/libs/fullcalendar/5.11.3/main.min.css" rel="stylesheet">
</head>
<body>

<aside class="sidebar">
<script>
// Sidebar hover functionality (Optional)
document.querySelector('.sidebar').addEventListener('mouseenter', function () {
this.style.width = '65px';
});
document.querySelector('.sidebar').addEventListener('mouseleave', function () {
this.style.width = '60px';
});
</script>

<svg xmlns="http://www.w3.org/2000/svg" width="60" height="60" viewBox="0 0 60 60" fill="none">
<g clip-path="url(#clip0_104_1746)">
<path d="M13.0909 36.5023C13.0909 41.6725 13.0909 44.2576 14.8858 45.8901C16.6807 47.5227 19.5229 47.5227 25.2073 47.5227L31.8285 47.5227C38.
624 47.5227 42.0218 47.5227 42.6574 45.7262C43.293 43.9296 40.6177 41.8875 35.2671 37.8031L13.0909 20.875L13.0909 36.5023Z" fill="url
(#paint0_linear_104_1746)"/>
<path d="M13.0909 24.324C13.0909 19.1538 13.0909 16.5687 14.8858 14.9361C16.6807 13.3036 19.5229 13.3036 25.2073 13.3036L31.8285 13.3036C38.
624 13.3036 42.0218 13.3036 42.6574 15.1001C43.293 16.8966 40.6177 18.9388 35.2671 23.0231L13.0909 39.9512L13.0909 24.324Z" fill="url
(#paint1_linear_104_1746)"/>
</g>
<defs>
<linearGradient id="paint0_linear_104_1746" x1="-3.1579" y1="18.6291" x2="0.661875" y2="49.971" gradientUnits="userSpaceOnUse">
<stop stop-color="#47FFBD" stop-opacity="0.2"/>
<stop offset="1" stop-color="#62FF48"/>
</linearGradient>
<linearGradient id="paint1_linear_104_1746" x1="-3.1579" y1="42.1972" x2="0.661875" y2="10.8553" gradientUnits="userSpaceOnUse">
<stop stop-color="#47FFBD" stop-opacity="0.2"/>
<stop offset="1" stop-color="#62FF48"/>
</linearGradient>
<clipPath id="clip0_104_1746">
<rect width="60" height="58.8506" fill="white" transform="translate(0 0.979736)"/>
</clipPath>
</defs>
</svg>

<script>
function confirmLogout(event) {
event.preventDefault(); // Prevent the default link behavior
if (confirm("Are you sure you want to log out?")) {
window.location.href = "{{ url_for('logout') }}"; // Redirect to logout route
}
}
</script>
<nav class="nav">
main*
0 2
0 0
0

# DASHSTYLE.CSS

login.html | footer.html | # dashstyle.css 1, M | help.ht

static > css > # dashstyle.css > ...

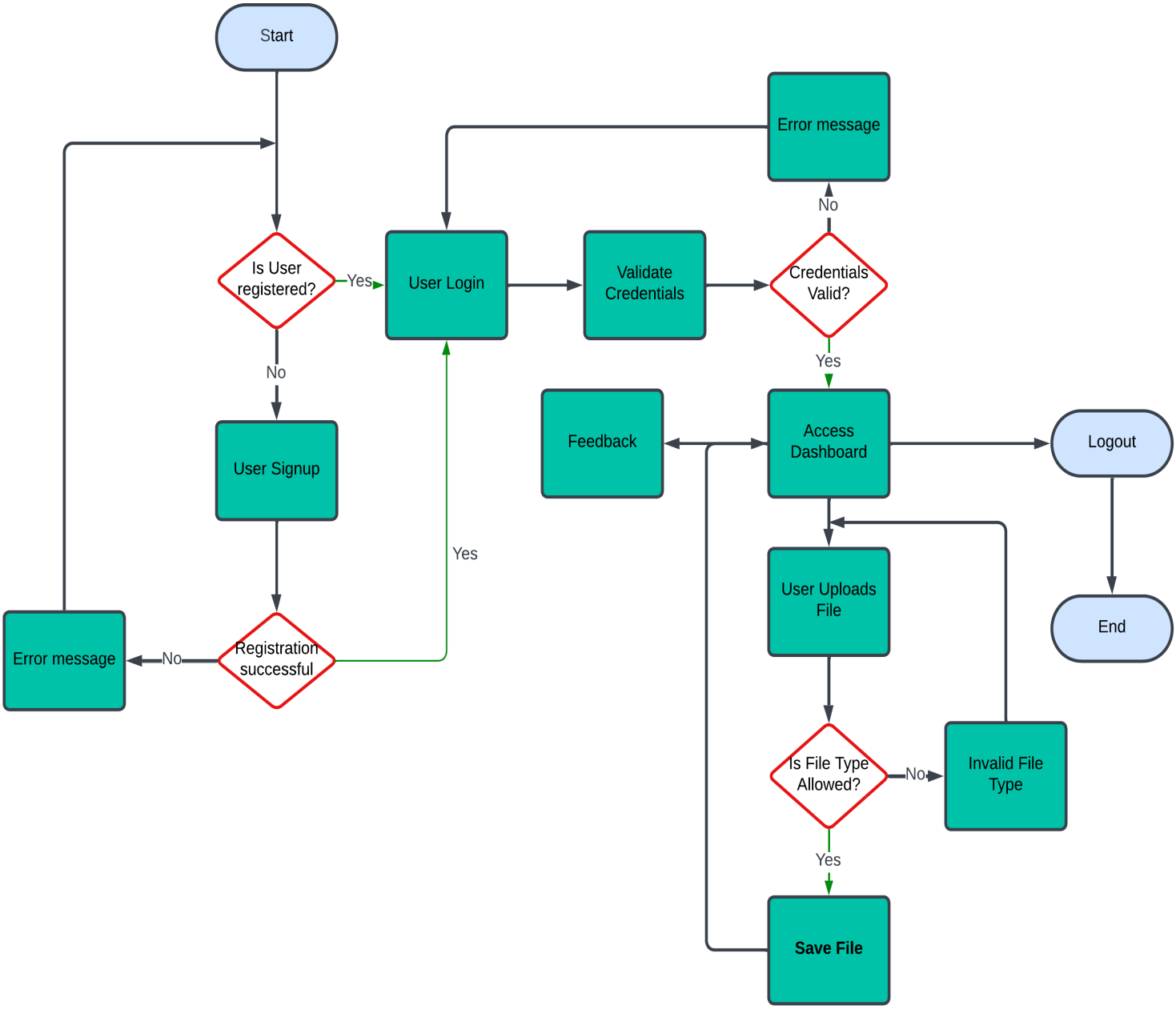
```
3   * {
4       margin: 0;
5       padding: 0;
6       box-sizing: border-box;
7       font-family: 'Roboto', sans-serif;
8   }
9
10  body {
11      background: var(--Black, #0D0D0D);
12      color: white;
13  }
14
15  .container {
16      display: flex;
17      height: 100vh;
18  }
19
20  .footer {
21      background-color: #232323;
22      color: #fff;
23      padding: 30px 20px;
24      text-align: center;
25      position: relative;
26      width: 100%;
27      bottom: 0;
28      box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
29      border-top-left-radius: 15px;
30      border-top-right-radius: 15px;
31  }
32
33  .footer-content p {
34      margin: 10px 0;
35  }
36
37  .social-links {
38      margin-top: 20px;
39      display: flex;
40      justify-content: center;
41      gap: 20px;
42      flex-wrap: wrap;
43  }
44
45  .social-links a {
46      display: inline-block;
47      transition: transform 0.3s ease;
48  }
49
50  .social-links img {
51      filter: none;
52      max-width: 40px;
53      height: auto;
54      transition: transform 0.3s ease, opacity 0.3s ease;
55  }
56
57  .social-links a:hover img {
58      transform: scale(1.2);
59      opacity: 0.8;
60  }
61
62  .contact-options {
63      margin-top: 30px;
64      text-align: center;
65      font-size: 1.1rem;
```

login.html | footer.html | # dashstyle.css 1, M

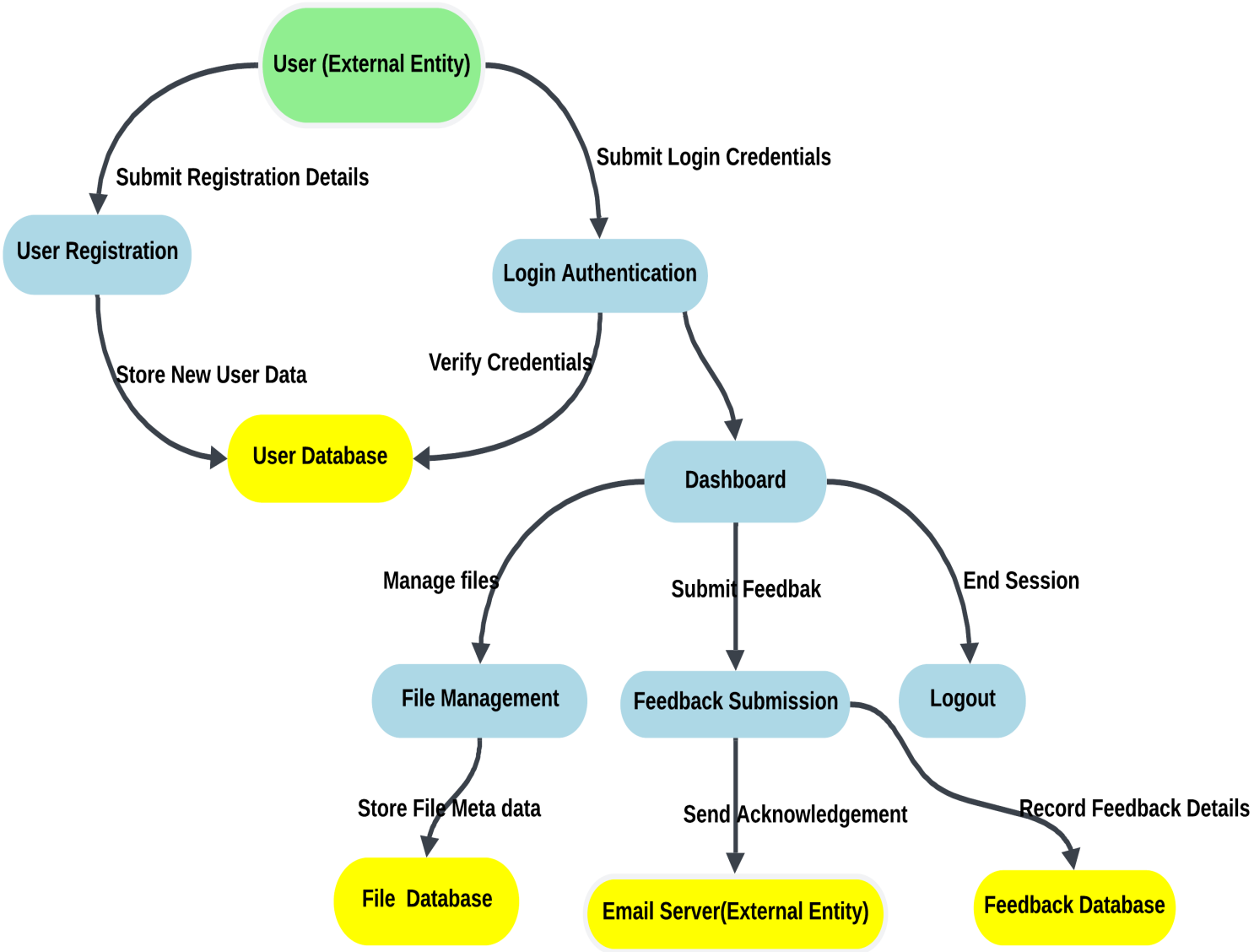
static > css > # dashstyle.css > ...

```
68  .contact-options p {
69      margin: 8px 0;
70  }
71
72  .contact-icon {
73      max-width: 25px;
74      margin-right: 10px;
75      vertical-align: middle;
76      transition: transform 0.3s ease;
77  }
78
79  .contact-options a:hover .contact-icon {
80      transform: scale(1.2);
81  }
82
83  .contact-options a:hover {
84      color: #a3d65e;
85  }
86
87  .footer-content a {
88      color: #fff;
89      text-decoration: none;
90  }
91
92  .footer-content a:hover {
93      color: #a3d65e;
94  }
95
96  @media (max-width: 768px) {
97      .social-links {
98          gap: 15px;
99      }
100     .contact-options p {
101         font-size: 1rem;
102     }
103     .contact-icon {
104         max-width: 20px;
105     }
106 }
107
108 @media (max-width: 480px) {
109     .social-links {
110         gap: 10px;
111     }
112     .contact-options p {
113         font-size: 0.9rem;
114     }
115     .contact-icon {
116         max-width: 18px;
117     }
118 }
119
120
121
122
123 .parent-dashbaord {
124     background-color: grey;
125     padding: 20px;
126     border-radius: 10px;
127     display: flex;
128 }
129
130 .dashboard {
```

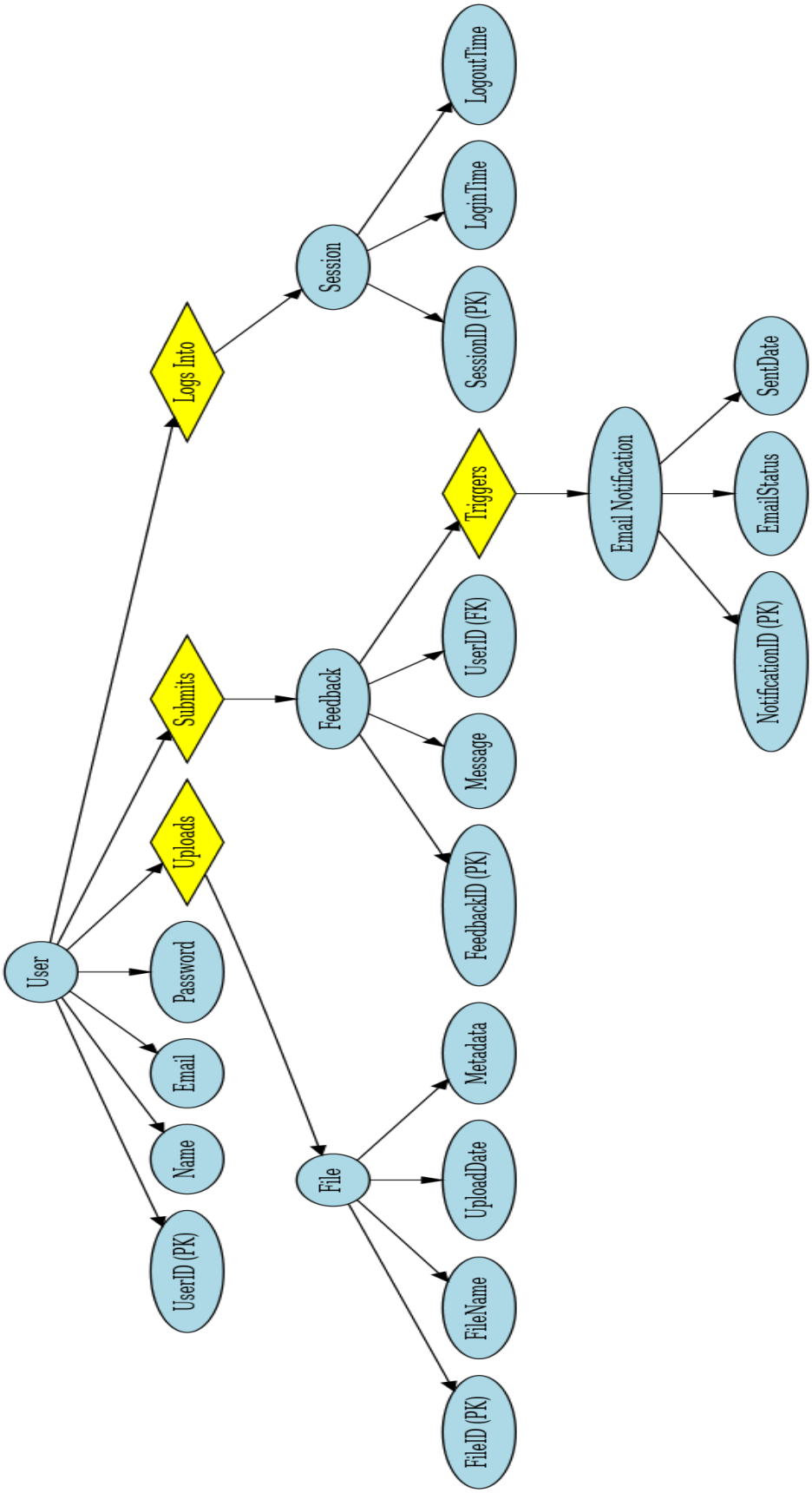
# Flowchart


Start
Is User registered?
Yes
No
User Login
User Signup
Error message
No
Registration successful
Yes
Error message
Validate Credentials
Credentials Valid?
No
Yes
Feedback
Access Dashboard
Logout
End
User Uploads File
Is File Type Allowed?
No
Invalid File Type
Yes
Save File

# DATA FLOW DIAGRAM



# ER DIAGRAM


User
Logs Into
Submits
Uploads
Session
Feedback
File
Triggers
Email Notification
UserID (PK)
Name
Email
Password
SessionID (PK)
LoginTime
LogoutTime
FeedbackID (PK)
Message
UserID (FK)
FileID (PK)
FileName
UploadDate
Metadata
NotificationID (PK)
EmailStatus
SentDate

# REFERENCE

## General Flask Development

1. **Flask Official Documentation**
    - Flask Quickstart – Covers core Flask functionalities.
    - Flask Patterns – Advanced patterns, including file uploads and error handling.
2. **SQLite with Flask**
    - Flask-SQLite Integration – Shows how to connect SQLite with Flask.
3. **Session Management**
    - Flask Sessions – Covers session handling for login/logout.

## Specific Features in Your Project

1. **File Upload and Management**
    - Secure File Uploads in Flask – Ensures secure file handling.
    - Handling File Uploads with Werkzeug.
2. **Creating Folders Dynamically**
    - Using os.makedirs – Python's standard library documentation for creating directories.
3. **Flask-Mail for Email Notifications**
    - Flask-Mail Documentation – Guide on configuring and sending emails in Flask.
4. **User Authentication**
    - Building a Login System in Flask – Covers building a basic login/logout system.

## Security Best Practices

1. **Password Hashing**
    - Use a library like bcrypt or Flask-Bcrypt to hash passwords securely instead of storing plain text passwords.
2. **Input Validation**
    - WTForms for Validation – Recommended for form handling and validation.
3. **Environment Variables for Secrets**
    - Store sensitive information (e.g., `MAIL_USERNAME`, `MAIL_PASSWORD`, `SECRET_KEY`) using python-dotenv or Flask's `config.from_envvar()`.

## Frontend Integration

1. **Using Jinja2 for Templates**

- Jinja2 Documentation – Template rendering with Flask.

2. **Bootstrap for UI**
   - Bootstrap-Flask – A library for integrating Bootstrap with Flask apps.

## Project Hosting

1. **Deploying Flask Applications**
   - Deploying on Heroku – Steps to deploy Flask projects.
   - Deploying on AWS EC2.
2. **Database Hosting for SQLite**
   - Consider migrating to PostgreSQL or MySQL for production using SQLAlchemy.

## Useful Flask Tutorials

1. Flask Mega-Tutorial by Miguel Grinberg
   - Comprehensive guide covering almost every feature in Flask.
2. Real Python Flask Tutorials
   - A wide range of Flask tutorials from beginner to advanced levels.

---

# FUTURE SCOPE

### 1. Integration of Cloud Hosting

- Transition to hosting KeyScore on popular cloud platforms like AWS, Google Cloud, or Azure for enhanced scalability and availability.
- Enable cross-device synchronization by storing data in the cloud.

### 2. Backup and Recovery

- Add automated data backup and recovery mechanisms to prevent data loss and ensure reliability.

### 3. Enhanced Security Features

- Implement advanced security measures such as two-factor authentication (2FA), role-based access control, and encryption of stored files.
- Add features like audit logs to track user activity.

### 4. AI-Powered Features

- Use AI for intelligent search, file organization, and duplicate file detection.
- Add AI-based analytics to provide insights about uploaded files, usage patterns, or result trends.

### 5. Collaboration Tools

- Enable real-time collaboration on files with features like sharing, commenting, and versioning.
- Add user-specific permissions for shared files.

### 6. Mobile App Development

- Develop Android and iOS apps for seamless access to files on the go.
- Utilize features like offline access and notifications for new uploads or feedback.

### 7. Advanced Feedback System

- Introduce feedback analytics to generate reports on user satisfaction and improvement suggestions.
- Add the ability for users to rate and comment on specific files.

### 8. API Development

- Develop APIs for third-party integrations, enabling KeyScore to work with other tools like Slack, Microsoft Teams, or Google Workspace.

### 9. Environmental Initiatives

- Incorporate features to promote sustainability, like file deduplication or monitoring to reduce digital clutter.

# CONCLUSION

This Flask application, KeyScore, enables user authentication, file uploads, and feedback submission, with secure session management and email integration. It provides a user-friendly interface for managing files, including organizing them into folders, downloading, and deleting them. The application uses SQLite for efficient data storage and integrates error handling to ensure smooth operation. Additionally, it facilitates feedback collection via email, contributing to better user interaction. With its core features in place, the app offers a solid foundation for expanding functionality and enhancing security, such as adding password hashing and more robust file management options. The ability to manage multiple file formats, including images and PDFs, further increases the application's versatility. Future improvements could include adding real-time notifications for file uploads or feedback submissions. The application is designed to be scalable, allowing for easy integration of additional features like file sharing and advanced user management. With a secure login system, it ensures a safe and personalized experience for each user.

---