**Code Documentation**

# **Project Overview:**

Using AWS S3, this project creates a cloud-based storage system that lets users upload, download, view, and remove data. It is constructed with the Flask web framework for Python, which uses HTML, CSS, and Bootstrap to provide a straightforward but powerful user interface. Render, which provides continuous deployment and automatic scalability, has been used to deliver the project. The solution uses AWS S3 to manage data redundancy and secure access, with a focus on safe, scalable file storage. The project setup, features, architecture, and planned enhancements are all covered in this paper.

**This project contains following files of codes:**

# **1. App.py**

This project uses Flask and AWS S3 to construct a cloud-based storage system. Users can register, log in, and upload and download files to and from S3 buckets using this system. It has file management, secure user authentication, and interface with AWS S3 for cloud storage that is scalable.

1. **Starting the Flask application:**  
   The Flask() constructor is used to initialize the Flask application. A secret key is set up in the program to control cookies and user sessions. It also uses boto3, the Python AWS SDK, to interface with AWS S3.

**Configuration of AWS S3:**

* The S3 bucket name that will be used to store files is defined by S3\_BUCKET.
* The S3 bucket's AWS region is specified by S3\_REGION.
* The S3 connection is initialized for file operations via boto3.client().

1. **Management of sessions and user authentication:**  
   Flask-Login is used by the application to handle sessions. It consists of :

* **LoginManager** : Sets up redirects for login and handling of user sessions.
* **UserMixin** : A utility class that gives Flask-Login's mandatory user functions default implementations.
* **Database**: User emails and hashed passwords are stored in a table called users, which is powered by SQLite.

1. **Configuring a Database**

* **The init\_db()** function establishes a users table to hold user credentials (password, email address, and ID) and initializes the SQLite database.
* **User class**: A simple user class that represents the user that is logged in at that moment.

1. **Routes**

* **/ (Main Route):** Sends visitors who are not verified to the registration page. Users that have verified their identity are taken to the main page (/index).  
  Only users who are logged in can view the home page at /index.
* **/signup:** Enables new users to register by providing their password and email address. Before being stored in the database, passwords are hashed using werkzeug.security.generate\_password\_hash().
* **/login:** Verifies a user's identity by comparing their password and email address to credentials that are stored in the database.
* **/logout:** Ends the current user's session and logs them out.

1. **S3 Integration**

* **/buckets:** Use s3.list\_buckets() to get a list of all accessible S3 buckets. Only users who have been authenticated can access this route.
* **/upload:** Manages the S3 bucket's file uploads. S3.upload\_fileobj() is used to upload files, streaming the file straight to S3.
* **/download:** Utilizes the S3 bucket to download files. Send\_from\_directory() is used to transmit the requested file to the user once it has been locally saved. s3.download\_file() is used to download the file from S3.

1. **Mistake Management**  
   Error management is included in the application for:  
   1. Many attempts by users to sign up.  
   2. Invalid login credentials.  
   3. S3 processes that flash problems to the user, such as file uploads and downloads.
2. **A User Loader for Flask-Login**  
   In order to guarantee session persistence between requests, the function load\_user() is used to fetch user information from the database based on user ID.
3. **File Management**

* **File Upload:** Through a form on the index page, users can upload files to the S3 bucket.
* **File Download:** By providing the file name, users can download files directly from the S3 bucket.

1. **Running the Application**  
   Testing and debugging Flask is made simpler by operating it in debug mode. Upon execution, a local web server is launched.

# **2. Index.html**

A straightforward, user-friendly interface for uploading and downloading files to and from the Cloud Storage System is offered by this HTML website. The webpage interacts with AWS S3 for file management and combines with Flask back-end routes (/upload and /download). An description of each part and its main elements may be found below.

1. **HTML Organization**  
   With a declaration, the document adheres to the standard HTML5 structure, guaranteeing compatibility with contemporary browsers.  
   The language of the page is English (lang="en"), and responsive design and character encoding are supported using meta tags.
2. **CSS Style**  
   To give the page a polished look, an embedded

* **Background:** A random cloud/technology-themed image (url('https://source.unsplash.com/random/1920x1080/?cloud,technology')) is dynamically retrieved from Unsplash to create the background image. The image is centered and set to fill the viewport (background-size: cover).
* **Font and Layout:** Arial is the font picked since it is readable.  
  Flexbox (display: flex; justify-content: center; align-items: center; height: 100vh;) is used to layout the website such that it is centered both vertically and horizontally, providing consistent alignment across all screen sizes.  
  To create contrast with the dark, semi-transparent background, the text color is set to white.
* **Container Design:** A div that acts as a content wrapper is given the.container class. To improve readability against the backdrop image, it has a semi-transparent background (rgba(0, 0, 0, 0.7)).  
  For a contemporary appearance, use rounded corners (border-radius: 10px) and padding for spacing.  
  Box shadow enhances visual attractiveness and depth.

1. **Heading**

(Cloud Storage Application): This heading uses a bottom margin for space and a font size of 24 pixels to introduce the application

1. **Upload Form**

* **Form Action:**

When a file is uploaded, the Flask back-end handles the HTTP POST request made by the first form to the /upload route.

* **Fields of Input:**Uploading a file from the user's system is possible with the File Input (input[type="file"]). The necessary feature guarantees that submitting the form won't be possible without choosing a file.
* **Submit Button (input[type="submit"]):** The "Upload" button initiates the submission of the form.
* **Style:** To ensure user comfort, inputs are padded, border-radius-styled, and filled with the available width (width: calc(100% - 20px)).  
  The submit button has a hover effect that changes the color for interactivity (background-color: #bddc22) and a background color of #1596d2.

1. **Download Form**

Click the Form to Download It. In order to download files, the second form sends an HTTP POST request to the /download route, which is managed by Flask.

* **Fields of Input:**(input[type="text"]) Text Input: By doing this, the user can provide the file name they want to download from the cloud storage. It also has a property that is necessary for validation.
* **Submit Button**The "Submit" button This button is called "Download" and it submits the form, just like the upload button does.

1. **Flow of User Interaction**

* **Upload:**

By filling out the form, users can choose a file from their computer and upload it to the cloud. Using this form, the file is sent to the server for additional processing and AWS S3 storage.

* **Download:**

By filling out the download form, users can download a file by entering its filename. The file will be retrieved and served for download via the back-end.

1. **Responsiveness to Mobile**

The page is responsive and mobile-friendly because of the meta tag, which makes sure that the layout changes to fit various screen sizes and devices.

# **3. Login.html**

The Cloud Storage System's login interface is represented by this HTML page. It is constructed with custom CSS for extra visual appeal, Flask for user authentication and form submission, and Bootstrap 4 for styling and responsive design. The elements of the page are broken down into detail below:

1. **HTML Organization**  
   The document is organized according to the HTML5 standard:  
   **Language:** English (lang="en") is the language selected for this page.  
   **Meta Tags:** charset="UTF-8": guarantees correct encoding of characters.  
   **viewport:** Makes sure the page is responsively designed and adjusts to various screen widths.  
   X-UA-Compatible: Guarantees that the rendering modes of Internet Explorer are compatible.
2. **Title:** The title of the page is defined by the <title> tag and will appear as Login in the Browser Tab.
3. **Integration with Bootstrap**  
   The page uses a CDN to quickly customize the buttons and login form with minimal additional CSS by referencing Bootstrap 4.5.2. The following are advantages of using bootstrap:

* **Form controls** include buttons, form layouts, and styled text inputs.
* **Responsive design:** modifies the layout for desktop, tablet, and mobile screens automatically.

1. **Personalized Style**  
   A <style> block that is embedded adds the following modifications:

* **Background picture:**

Unsplash (url='https://source.unsplash.com/random/1920x1080/?city,night')) provided the dynamic background image for this page. The picture is centered for aesthetic appeal and fills the screen (background-size: cover).

* **Container for Login:**  
  For readability, a semi-transparent white background (rgba(255, 255, 255, 0.9)) encloses the form in a.login-container div.  
  Borders and Cushioning: For a contemporary appearance, use rounded corners (border-radius: 8px) and spacing of 30px. The form gains depth from the box-shadow.  
  Flexbox is used to center the container on the page.

1. **The Login Form**  
   The following elements go into creating the login form:

* **Formal Inaction:** The Flask route url\_for('login'), which manages the login logic, is assigned as the form's action. Sensitive user information, including as email addresses and passwords, is securely sent to the server thanks to the POST form technique.
* **E-mail Input:** makes sure that the email address entered by the user is legitimate.  
  formatted consistently by utilizing the form-control class from Bootstrap.  
  This field cannot be left empty thanks to the necessary attribute.  
  Password Input: For security, conceals the user's password as they type.  
  possesses the necessary characteristic and is styled using the form-control class as well.
* **The "Submit" button**  
  For a blue theme, the button is decorated with Bootstrap's btn and btn-primary classes.  
  The background color of the button can be altered using custom CSS for its hover effect.

1. **Registration URL**

If a user does not have an account, the page has a link below the form that takes them to the signup page (url\_for('signup')).

1. **Messages in Flash**  
   Get\_flashed\_messages() in the Flask framework is used to show feedback messages, including failed login attempts or any alerts.  
   The block checks for messages using Jinja templating ({% %}) and presents them inside a Bootstrap alert (alert-warning) with margin and text centering.
2. **JavaScript Dependencies for Bootstrap**To make sure that any interactive Bootstrap elements—like modals and dropdowns—function properly, the page incorporates jQuery, Popper.js, and Bootstrap's JavaScript over CDN connections. Though added for completeness, these are optional in this instance.

# **4. Signup.html**

This document describes the Cloud Storage System web application's Signup Interface. For layout and styling, Flask is used in development, and Bootstrap 4.5.2 is used for backend connectivity. A summary of the main elements and their functions may be found below.

1. **HTML Organization**

The document follows the structure specified by HTML5:

* **Language:** English is the page language indicated by the lang="en" attribute.
* **Metadata:**For compatibility, specify the character encoding with charset="UTF-8".
* **viewport:** Guarantees responsiveness on a range of screen sizes, including desktop and mobile.

Internet Explorer compatibility with previous versions is indicated by the code X-UA-Compatible.

1. **Title**

"Signup" is the title of the webpage that appears in the browser tab and is defined by the <title> element.

**3. Integration with Bootstrap**

With the least amount of work, modern, responsive styling is provided using Bootstrap 4.5.2 linked via CDN. It guarantees that the button and form elements are styled correctly:

* offers a responsive, uniform layout for all devices.
* makes use of the grid concept in Bootstrap to manage layout flexibility.

**4. Personalized Style**

Extra modification of the design is possible with a <style> block:

* **Background Image:** The page's full-screen background (background-size: cover) is an arbitrary Unsplash image with a water and nature theme that is centered for aesthetic appeal.
* **Signup Form Container:** For style purposes, enclosed in a.signup-container div.
* To improve readability of the form content over the background, a semi-transparent white background (rgba(255, 255, 255, 0.9)) is applied to the container.
* **Borders and Cushioning:** The user experience is improved by the 30px padding and rounded corners (border-radius: 8px).

The container appears slightly elevated thanks to a box shadow.

**5. Signup Form:**

This form uses to collect user information

* **Form Action:** The Flask route url\_for('signup'), which manages user registration, receives a submission from the form. The form securely sends data to the backend via the POST method.
* **Email Input:** <input type="email"> makes sure that the email entered by the user is legitimate, styled for consistency and usability using the form-control class from Bootstrap.

The field can never be left empty thanks to the mandatory attribute.

* **Password Input:** To ensure that the password remains hidden while typing, use <input type="password">.

further decorated with the needed property for validation using the form-control class from Bootstrap.

* **The "Submit" button**

styled for a button with a blue theme using Bootstrap's btn and btn-primary classes.

The button's color can be altered using CSS to modify the hover effect.

**6. Login URL**

Those who already have an account are invited to log in via a message beneath the form.

* This message contains a link to the url\_for('login') login page.
* The text is balanced in the middle.

**7. Messages in Flash**

* Flash messages (such as errors or notifications) are shown from the backend using Flask's get\_flashed\_messages() function:
  + If there are any flash messages, they are displayed with text centered inside a Bootstrap alert component called alert-warning.

**8. JavaScript Dependencies for Bootstrap**

The JavaScript libraries for Bootstrap, Popper.js, and jQuery are added to handle any dynamic behavior or interaction that the Bootstrap components may need. Whole Bootstrap functionality is ensured by these dependencies.