Case Study Assignment [Level 3]:
Python-Based Web Application for Document Management

Objective:

The objective of this assignment is to develop a web-based document management system using Flask, HTML, and CSS. This assignment will help demonstrate understanding of web development concepts, including building a web application with Flask, creating user interfaces with HTML and CSS, and implementing CRUD operations with file upload and download features.

Background:

You have been hired as a web developer for a company that needs a document management system. The system should allow users to upload, view, update, and delete documents. Each document should have a title, description, upload date, and the actual file.

Assignment Tasks:

1. **System Design:**
    - Design a database schema for storing documents using SQLite3 (or another database of your choice). The schema should include a table `documents` with columns `id`, `title`, `description`, `upload_date`, and `file_path`.
    - Design a Flask application structure with appropriate routes for each CRUD operation.

2. **Flask Application:**
    - Set up a Flask project with the necessary dependencies.

- Implement the following routes:
  - `/`: Home page displaying all documents.
  - `/upload`: Page to upload a new document.
  - `/document/<int:id>`: Page to view a specific document.
  - `/update/<int:id>`: Page to update an existing document.
  - `/delete/<int:id>`: Endpoint to delete a document.
  - `/download/<int:id>`: Endpoint to download a document.

3. **HTML and CSS:**
   - Create HTML templates for each route using Jinja2 templating engine.
     - Home page template (`index.html`) displaying a list of all documents with options to view, update, delete, and download.
     - Upload page template (`upload.html`) with a form to upload a new document.
     - View page template (`view.html`) to display document details.
     - Update page template (`update.html`) with a form to update document details.
   - Style the templates using CSS to create a user-friendly interface.

4. **File Handling:**
   - Implement file upload functionality using Flask's `request.files` to handle file uploads and save them to a specified directory.
   - Implement file download functionality to allow users to download documents.

5. **CRUD Operations:**
   - Implement the necessary logic in the Flask routes to perform CRUD operations:
     - Create: Save uploaded documents with their metadata to the database.

        - Read: Retrieve and display documents from the database.
        - Update: Update document details and optionally replace the file.
        - Delete: Remove documents from the database and delete the associated file from the server.

6. **Validation and Error Handling:**
    - Add validation to ensure that documents have valid titles, descriptions, and files.
    - Implement error handling to manage scenarios such as file upload failures, missing documents, and invalid update requests.

Deliverables:

- Python code implementing the Flask web application.
- HTML and CSS files for the user interface.
- A short documentation (PDF or Word document) explaining the design and usage of the system.

Evaluation Criteria:

- Correctness and completeness of the Flask routes and application logic.
- Functionality of the document management system, including file upload, view, update, delete, and download.
- Effectiveness of the database integration.
- Usability and aesthetics of the HTML and CSS user interface.
- Quality and clarity of the code, including documentation and comments.
- Handling of validation and error scenarios.

Submission Instructions:

- Submit your Python code, HTML, CSS files, and documentation in a single compressed folder.
- Ensure that your code is well-documented and includes comments explaining key steps and decisions.
- Name your submission folder as `DocumentManager_<YourName>`.

Tips:

- Make use of Flask extensions such as `Flask-WTF` for form handling and `Flask-Uploads` for file uploads.
- Focus on creating a user-friendly interface to make the document management system easy to use.
- Pay attention to code readability and organization.
- Ensure that uploaded files are stored securely and handled appropriately.