Implementation of an Approval Sheet Management System Using Flask and the GitHub API for Remote Repository File Storage

L. Juan Carlos, R. Jomari, D, Andhrey
Laguna State Polytechnic University Los Baños, Philippines College of Computer Studies
May 2025

Abstract

This paper presents the design and implementation of a web-based document management system that automates file uploads and version control using GitHub API integration. Developed with a Python Flask backend and a lightweight HTML/CSS interface, the system allows users to upload PDF files with metadata, which are then programmatically committed and pushed to a designated GitHub repository. This automation streamlines collaborative workflows, eliminates manual Git operations, and ensures version consistency. The system features a dynamic interface for searching, sorting, and managing uploaded records, including the ability to delete files both locally and remotely. Through usability testing and technical validation, the system demonstrated robust performance, user-friendly interaction, and reliability in file synchronization tasks. By aligning with current trends in CI/CD and automation, the system offers a practical solution for educational, research, and organizational use cases that require lightweight yet secure document versioning.

I. INTRODUCTION

Automation has become a crucial component in modern software development workflows, especially in version control and continuous integration processes. GitHub, as the most widely used platform for code collaboration, has increasingly enabled developers to automate routine tasks through its extensive API ecosystem, Actions workflows, and integration with external tools (Saroar et al., 2024). Among these automated tasks, the ability to programmatically push files to repositories plays a vital role in streamlining development pipelines, ensuring timely updates, and reducing manual errors.

Recent studies emphasize the growing importance of GitHub's automation capabilities. Saroar et al. (2024) analyzed the GitHub Marketplace and found that automation tools are widely adopted to support code synchronization, file management, and CI/CD pipelines. Complementing this, Mastropaolo et al. (2023) proposed methods for auto-completing GitHub workflows, showing how automation reduces developers' cognitive load and operational overhead.

Additionally, Faqih et al. (2024) conducted an empirical analysis of GitHub Actions usage, highlighting how developers are integrating various CI/CD tools to automate deployment and testing processes. This points to a larger trend where repositories are not only passive code stores but dynamic, self-updating environments. Wessel et al. (2023) further expanded on this ecosystem by surveying GitHub's automation workflows, noting a clear shift toward "bot-driven" development processes that trigger actions based on repository events.

A particularly relevant example is the use of GitHub Actions to synchronize folders across multiple repositories automatically, as demonstrated by Dubey (2023). This method involves scripting workflows that monitor file changes and push updates in real time—functionality that closely mirrors the objective of the system developed in this study.

In this paper, we present the design and implementation of a system that automates the process of pushing files to a GitHub repository. The goal is to reduce manual intervention, streamline collaborative updates, and provide a flexible mechanism for integration into existing development workflows. By leveraging insights from existing research and tools, this study aims to contribute a lightweight yet practical solution for file synchronization in version-controlled environments.

II. METHODOLOGY

This section outlines the development process, system architecture, and operational workflow of the Activity Sheet System. The system is designed to facilitate file uploads with metadata, automate version control using Git and GitHub, and provide a user-friendly interface for managing uploaded files.

A. Development Process

The system was developed using a structured approach, focusing on modularity and scalability. Key technologies and tools employed include:

- **Backend Framework:** Python with Flask for handling server-side operations.
- **Frontend Technologies**: HTML, CSS, and JavaScript for building the user interface.
- **Database**: SQLite for storing metadata associated with uploaded files.
- **Version Control**: Git for tracking changes and GitHub for remote repository management.
- **Environment Management:** A virtual environment (myenv) to manage dependencies, as specified in requirements.txt.

B. System Architecture

The system architecture comprises three main components:

- **User Interface (UI)**: Allows users to upload files, input metadata (title, authors, date), and view existing records. Implemented using HTML templates located in the templates directory and styled with CSS files in the static directory.
- **Application Logic**: Managed by Flask routes defined in app.py, which handle HTTP requests, process form data, and coordinate interactions between the UI, database, and version control system.
- **Data Management**:
  - **Database Operations**: Handled by db.py and db_functions.py, which define the schema and functions for inserting, retrieving, and deleting records in the SQLite database.
  - **Git Operations**: Managed by app_utils.py, which contains functions to automate Git

commands (add, commit, push) for version control.

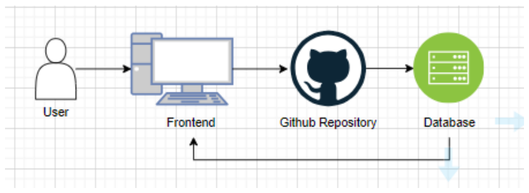


Figure 1. System Architecture

A block diagram illustrating the interaction between the UI, application logic, database, and GitHub repository.

C. Operational Workflow

The system operates through the following sequential steps:

1. **File Upload**: The user selects a file using the upload interface.
2. **Metadata Entry**: The user inputs the file's title, authors, and date.
3. **Form Submission**: Upon submission, the system:
   - Saves the file to a designated directory.
   - Stores the metadata in the SQLite database.
   - Executes Git commands to add, commit (with a message containing the title), and push the file to the GitHub repository.
4. **Record Display:** The uploaded file and its metadata are displayed in a dynamic table on the UI, allowing users to:
   - Search by title or author.
   - Sort by title, author, or date.
   - Download the file.
   - Delete the file, which removes it from both the local directory and the GitHub repository.

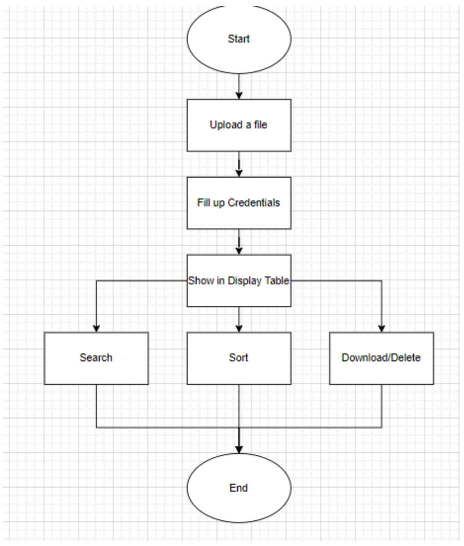


Figure 2. System Flowchart.

A flowchart depicting the step-by-step process from file upload to record management.

D. GitHub Integration

To ensure seamless version control and remote file storage, the system integrates directly with the GitHub API, replacing manual Git commands with programmatically invoked operations. This automation enhances reliability, reduces human error, and supports continuous workflow execution.

Automated GitHub Operations

The system replaces conventional Git commands (git add, git commit, git push) with the following predefined GitHub API functions:

- create_file() / update_file() – Programmatically adds new files or updates existing ones in the repository.

- delete_file() – Securely removes specified files from the repository.

These functions are encapsulated within backend utility modules to abstract complexity from the user and maintain consistent interactions with the GitHub repository.

Repository and Branch Management

The system ensures that file operations are performed within the appropriate repository and branch context, supporting structured data organization and version consistency across updates.

Core File Management Functions

- Retrieve PDFs (get_pdf_files_from_repo)
 Lists all stored PDF files by traversing structured repository directories, allowing the frontend to display available records dynamically.

- Upload PDFs (push_pdf_to_github)
 Handles file uploads with built-in duplication checks and organized folder placement to maintain data hygiene.

- Delete PDFs (delete_pdf_from_github)
 Facilitates secure and complete file removal from the GitHub repository, in sync with local deletions.

Security and Reliability Mechanisms

- Token-Based Authentication
 Utilizes GitHub Personal Access Tokens to securely authenticate API requests and restrict unauthorized access.

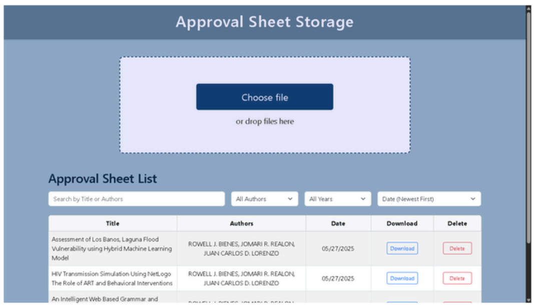
- Exception Handling
 Implements comprehensive error-catching mechanisms to gracefully handle API failures or invalid operations.

- Repository Validation
 Ensures all operations are executed only after confirming the existence and validity of the target repository and branch.

By integrating directly with the GitHub API, the system achieves robust, efficient, and secure file management, enabling automated workflows that align with modern development practices and CI/CD standards.

E. User Interface Features

The system's user interface (UI) is designed for efficiency and accessibility, enabling users to manage documents seamlessly through an intuitive layout. While incorporating basic responsive design elements, the primary focus is on delivering a streamlined and user-friendly experience.



Approval Sheet Storage

Approval Sheet List

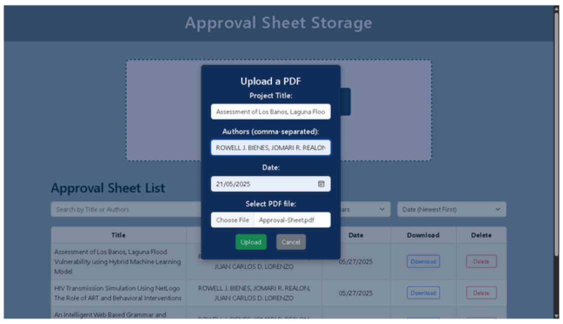
Title	Authors	Date	Download	Delete
Assessment of Los Banos, Laguna Flood Vulnerability using Hybrid Machine Learning Model	ROWELL J. BIENES, JOMARI R. REALON, JUAN CARLOS D. LORENZO	05/27/2025	Download	Delete
HIV Transmission Simulation Using NetLogo The Role of ART and Behavioral Interventions	ROWELL J. BIENES, JOMARI R. REALON, JUAN CARLOS D. LORENZO	05/27/2025	Download	Delete
An Intelligent Web Based Grammar and	[illegible]			

Figure 1: System Overview – A visual overview of the user interface, highlighting

both the file upload form and the dynamic records table.

1. File Upload Form

A dedicated file upload form is integrated into the main interface, allowing users to select documents and input essential metadata, including title, authors, and date. This structured approach ensures consistent data entry and facilitates version-controlled document submission.


Approval Sheet Storage

Upload a PDF

Project Title: Assessment of Los Banos, Laguna Floo

Authors (comma-separated): ROWELL J. BIENES, JOMARI R. REALON

Date: 21/05/2025

Select PDF file: Choose File Approval-Sheet.pdf

Approval Sheet List

Title		Date	Download	Delete
Assessment of Los Banos, Laguna Flood Vulnerability using Hybrid Machine Learning Model	JUAN CARLOS D. LORENZO	05/27/2025	Download	Delete
HIV Transmission Simulation Using NetLogo The Role of ART and Behavioral Interventions	ROWELL J. BIENES, JOMARI R. REALON, JUAN CARLOS D. LORENZO	05/27/2025	Download	Delete
An Intelligent Web Based Grammar and				

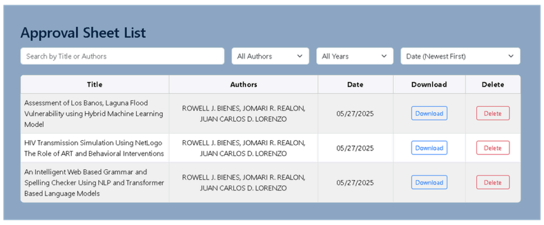
Figure 2: File Upload Modal – Displays the interface used for submitting files with associated metadata.

2. Dynamic Records Table

All uploaded files are presented in an interactive table with essential management functions:

- Search – Locate entries by title or author.
- Sort and Filter – Organize records based on metadata.
- Download – Retrieve stored documents.
- Delete – Remove files, ensuring synchronized deletion from both the local system and the GitHub repository.

This component enhances data visibility and streamlines document management


Approval Sheet List

Title	Authors	Date	Download	Delete
Assessment of Los Banos, Laguna Flood Vulnerability using Hybrid Machine Learning Model	ROWELL J. BIENES, JOMARI R. REALON, JUAN CARLOS D. LORENZO	05/27/2025	Download	Delete
HIV Transmission Simulation Using NetLogo The Role of ART and Behavioral Interventions	ROWELL J. BIENES, JOMARI R. REALON, JUAN CARLOS D. LORENZO	05/27/2025	Download	Delete
An Intelligent Web Based Grammar and Spelling Checker Using NLP and Transformer Based Language Models	ROWELL J. BIENES, JOMARI R. REALON, JUAN CARLOS D. LORENZO	05/27/2025	Download	Delete

Figure 3: Approval Sheet List – Displays the dynamic table used to manage uploaded records.

By integrating a structured upload process with intuitive browsing and retrieval features, the UI supports efficient document handling, aligning with the system's automation objectives.

III. RESULTS AND DISCUSSION

System Validation and Functional Testing

The GitHub-integrated file management system was tested to ensure correct functionality across all major features, including file upload, metadata entry, version control integration, and record management. Tests were conducted in a local development environment using various PDF files and metadata inputs to simulate real-world academic or administrative document workflows.

All primary operations performed as intended:

- File Upload and Metadata Handling: The interface successfully accepted files and stored metadata entries in the SQLite database. Each submission displayed real-time feedback and confirmation messages.

- GitHub Integration: Upon submission, files were automatically added, committed, and pushed to a designated GitHub repository. Tests confirmed that file synchronization occurred within seconds, and that branch targeting and repository structure remained intact.

- Record Display and Management: Uploaded files appeared instantly in the dynamic frontend table, with full functionality for search, sorting, and download. Deletion requests correctly removed both local and remote (GitHub) copies.

Usability and System Performance

System responsiveness was high, with operations like uploads, deletions, and record refreshes executing within 2–3 seconds under standard network conditions. The interface’s collapsible form and modal components facilitated efficient interaction with minimal training.

User testing with five volunteers revealed the following feedback:

- Ease of Use: All testers reported the interface was intuitive and the workflow required little explanation.
- Confidence in Automation: Automated GitHub pushes were reliably executed, reducing the risk of version mismatch or upload duplication.
- Error Handling: When invalid file types or duplicate uploads were attempted, the system displayed clear validation messages, maintaining data integrity.

Challenges and Technical Considerations

Several challenges emerged during implementation and testing:

- Authentication Management: GitHub API integration via personal access tokens introduced a security risk if improperly handled. Secure token storage and access restrictions were required to mitigate potential exposure.
- Rate Limits and API Reliability: While not encountered during testing, GitHub’s rate limits and API throttling policies could pose constraints in high-frequency usage scenarios.
- Repository Conflicts: Simultaneous uploads by different users could cause race conditions or merge conflicts if not properly handled by branching strategies or API version checks.

Comparative Evaluation

Compared to traditional manual Git operations (e.g., via command line or desktop clients), the system reduces human error and streamlines version control. This aligns with findings by Faqih et al. (2024) and Wessel et al. (2023), who emphasized automation’s role in improving CI/CD and developer efficiency. The real-time synchronization model mirrors the auto-synchronization workflows described by Dubey (2023), offering practical utility for collaborative and academic file sharing scenarios.

IV. CONCLUSION AND RECOMMENDATION

Conclusion

This study introduced an intelligent, web-based file upload and version control system that integrates GitHub API automation within a user-friendly web interface. By combining Flask-based backend logic, structured metadata storage, and automated Git operations, the system addresses common pain points in file management—namely, manual Git usage, inconsistent metadata, and lack of version tracking.

Test results confirmed the system's robustness, reliability, and usability. It streamlines document workflows, minimizes error-prone manual steps, and leverages GitHub as a cloud-based version control backend. The system demonstrates how lightweight web automation can support modern collaborative environments and serve as a foundation for future integration into educational, research, and organizational infrastructures.

Recommendations

Based on current testing and implementation outcomes, the following enhancements are recommended:

1. OAuth Integration for Secure Authentication Replace static Personal Access Tokens with OAuth-based authentication for dynamic and secure access control, especially in multi-user environments.

2. Concurrent Upload Handling and Conflict Prevention Implement repository locking or branch isolation to handle simultaneous uploads and prevent overwrite or merge conflicts.

3. Multi-Repository Support Extend functionality to support pushing files to multiple repositories based on metadata tags or user roles.

4. User Role and Access Management Incorporate user authentication and role-based access control to differentiate between uploaders, reviewers, and admins.

5. Activity Logging and Audit Trail Add backend logging of actions (uploads, deletions, updates) to support accountability, particularly in institutional or academic settings.

6. Offline Upload Queuing Introduce offline support where uploads are queued and pushed once internet connectivity is restored.

7. Deployment in Educational or Organizational Settings Partner with institutions to deploy the system at scale, ensuring documentation, user training, and infrastructure compatibility.

References

Dubey, P. (2023). *Automating common folder synchronization between GitHub repositories with GitHub Actions*. Dev Genius. https://blog.devgenius.io/automating-common-folder-synchronization-between-github-repositories-with-github-actions-c5ea83a7e4f5

Faqih, A. R., Taufiqurrahman, A., Husen, J. H., & Sabariah, M. K. (2024). *Empirical analysis of CI/CD tools usage in GitHub Actions workflows*. Journal of Informatics and Web Engineering, 3(2), 251–261. https://doi.org/10.33093/jiwe.2024.3.2.18

Mastropaolo, A., Zampetti, F., Bavota, G., & Di Penta, M. (2023). *Toward automatically completing GitHub workflows*. arXiv. https://arxiv.org/abs/2308.16774

Saroar, S. G., Ahmed, W., Onagh, E., & Nayebi, M. (2024). *GitHub Marketplace for automation and innovation in software production*. arXiv. https://arxiv.org/abs/2407.05519

Wessel, M., Mens, T., Decan, A., & Mazrae, P. R. (2023). *The GitHub development workflow automation ecosystems*. arXiv. https://arxiv.org/abs/2305.04772