# BANASTHALI VIDYAPITH

**INTERNSHIP REPORT 2023- 24**

****

*A report submitted in partial fulfilment of the requirements for the Award of Degree of*

#### BACHELOR OF TECHNOLOGY

**in**

***Computer Science Engineering***

#### by

**Unnati Saini**

**2216433**

**Under the Supervision of**

**Smt.Bhavna N. Merh**

**SOG , ACSD**

**Raja Ramanna Centre for Advanced**

**Technology (RRCAT) , Indore**

**(Duration: 15th May 2025 to 15th July 2025)**

**2022 – 2023**

**DECLARATION**

I hereby declare that work, which is being presented in the Internship Report as the partial fulfilment for the award of degree of **Bachelor of Technology** in **Computer Science Engineering** in the **Department of Computer Science Engineering** at **Banasthali Vidyapith**, **Jaipur**, is an authentic record of my work carried out under the Mentorship of Industry Mentor Smt. Bhavna N. Merh **, SOG, ACSD** and The matter embodied in this internship report has not been submitted for the award of any degree.

Roll Number: 2216433 Student Signature Date:

**2022 – 2023**

**INTERNSHIP APPROVAL SHEET**

This is to certify that **Unnati Saini** Roll Number 2216433 has successfully completed her industrial internship starting from 15th May to 15th July 2025 and has submitted the final report. Her work has been found satisfactory and it is recommended to accept it as a partial fulfilment for the award of a degree of **Bachelor of Technology, Computer Science Engineering** of the **Department of Computer Science Engineering** at **Banasthali Vidyapith**, **Jaipur.**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

# **ACKNOWLEDGEMENT**

I would wish to express my heartfelt gratitude to my Sh. R.K. Agrawal, Head, Software & Embedded Systems Section, ACSD of RRCAT who provided me with the opportunity in his reputed organization. I would like to acknowledge an opportunity provided by Industry Mentor of RRCAT to work at their highly reputed organization.

Secondly, I would like to appreciate Smt. Bhavna N. Merh, Scientific Officer G, ACSD and Dr. Surendra Yadav, Scientific Officer G with whom I worked in RRCAT and guided me in my work. Making it fun to work in, they managed to expose people to their patience and accessibility. It is with a very high feeling of pleasure, and an immense sense of gratitude that I take this opportunity of acknowledging the assistance of these individuals.

Special thanks or gratitude must go to Mr. Abhisek Nayak, who helped significantly and guided me sincerely without which my work would have not been completed successfully. His helpful and skilled support would have made it very difficult to achieve my objective if it were not done with his assistance.

*Name: Unnati Saini*

*Branch: Computer Science Engineering*

**EXECUTIVE SUMMARY**

### I have also satisfactorily finished my internship at the Raja Ramanna Centre of Advanced Technology (RRCAT) in the stream of Full Stack Web Development with Version Control Systems. The task that I was solving in the course of this internship was one of the most topical and applicable solutions in the current approach to working collaboratively, namely, the creation of a native version control system of documents called MyVCS (My Version Control System).

### The name of the project that I participated in was **Design and Development of a Document-Centric Version Control System to be Shared by a Collaboration**.

### The main aim of this project was to develop a web-based program with the help of which multiple users can engage in managing, tracking, and controlling versions of the common document formats, e.g., .pdf, .docx, and .txt versions. This was specifically intended to address the problematic situations which happens when using source-code versioning tools such as Git that are too complex and unsuitable for document-based workflows in academic and organizational settings.

### The project consisted of the total design of the system architecture that was built on Fast API as backend and PostgreSQL as the database where the metadata of users, repositories, commits, and access control could be stored. Version snapshot and documents files were stored safely in the local file system. I did so as well by introducing JWT-based authentication and role-based access control, creating an HTML, CSS (glass morphism design), and JavaScript-based completely responsive frontend dashboard to provide an intuitive interface to the user.

### During the internship, I faced and solved various problems with file versioning, merge conflicts, access rights of repositories and user interface responsiveness. All this was addressed with the help of constant debugging, diligent database schema development, and safe integration of REST API.

### The internship sharply improved my technical knowledge about full-stack development web skills, API design, and safe document management. More to the point, it made me understand how to create scalable and friendly user-systems that address real-life issues in collaborative computerized settings.

### 

### **Abstract**

The current project is named MyVCS - A Document Version Control System that has been created as an outcome of the summer internship and targeted at taking care of the problem of utilizing the collaborative document management in academic and organizational environment. In contrast to the classic source-code-based version control systems, the platform is self-hosted, a web-based application uniquely tailored to the particular set of document formats, 149, docx, and pdf. It is also capable of enabling the user to compose repositories, add new and modify files, follow temporal progress and revert to previous versions using a simplified user interface using dash boards.

It is constructed with the aid of Fast API and SQLAlchemy ORM on the back end and PostgreSQL, which represents the database containing metadata regarding users, repositories, commits and access management. The operating format continues to retain the document files, and its attached versioned snapshots, on the local file system which helps improve the performance and security. Granular data model presents the opportunity to describe in detail the tracking versions, merge history and support of the auditing trail, and particular entities, such as Commit, Snapshot, FileVersion, MergeHistory, and Log, aid in ensuring that there is a solid coverage of tracking document development and collaboration.

In supplement of the system, role-based access control is employed which helps in ensuring that the repository owners allow viewer/collaborators/ or administrators access. The authentication used is the one which uses the mechanism dependent on JWT and is sufficient to the users in order to control their credentials and it is equipped with an inbuilt log-in component. Its main characteristics include the availability of versioned commits, commit merging, rollbacks and revision logging of activity. A frontend dashboard is user-friendly, therefore nodes can be navigated, view documents, control commits and access-control made available even to users that are not technical.

Overall, MyVCS is an efficient, scalable environment, which eases the versioning of documents while enhancing data integrity and implementing productive collaboration. It solves the need to have a user-friendly and secure product that could be helpful in meeting the document-focus requirement of modern organizations.

I**NDEX**

1. **Introduction**

1.1 Problem Statement

1.2 Objectives

1.3 Scope and Importance

1. **System Design**

2.1 Overview of Architecture

2.2 Technology Stack

2.3 Database Model

2.4 Entity-Relationship (ER) Diagram

1. **Implementation**

3.1 Backend Implementation

3.1.1 Project Structure

3.1.2 API Endpoints

3.1.3 Key Backend Modules

3.1.4 Versioning Logic

3.1.5 Core Versioning Logic Components

3.1.6 Audit Trail and Logging

3.1.7 Access Control

3.1.8 Security & Authentication

3.2 Frontend Implementation

3.2.1 Overview

3.2.2 Features

1. **Use Case & Features**

4.1 Superadmin Features

4.1.1 Dashboard Overview

4.1.2 Users Section

4.1.3 Repositories Section

4.1.4 Activity Log

4.1.5 Reset Password

4.1.6 Register User

4.1.7 Logout

4.2 User Features

4.2.1 Home Section

4.2.2 Your Repositories Section

4.2.3 Public Repositories Section

4.2.4 Activity / History Section

4.2.5 Snapshot Section

4.2.6 Logout

4.3 Repository Dashboard Features

4.3.1 Common Features (All Roles)

4.3.2 With Write Access (Collaborator/Admin)

4.3.3 Admin-Only Features

4.4 Role Summary

1. **Testing and Results**

5.1 Testing Methodology

5.2 Test Cases

5.3 Sample Output Screens

1. **Conclusion**
2. **Appendix**
3. **References**

### **Introduction**

### The synchronization of document changes has become a frequent dilemma in academic, institutional, and group settings. Shared files being edited include draft reports, meeting notes, policies, or research records all along the life of the team. Lack of a proper version control system will typically cause ambiguity over the latest version, accidentally overwriting data, losing changes, and even complicate rollback operations as traditional systems like Git provide full version control features but are more oriented toward software source code and often need command line access and experience to use. As a result, it may seem that such tools are not meant to be used by non-developers who work with binary or text-based files (e.g., .txt, .pdf, .doc and .docx).

### **1.1 Problem Statement**

### Due to the absence of special management tools, the organization has a row of problems:

### The absence of a centralized repository of handling document revision: The revision of documents is ill-scattered across devices and software and extremely difficult to kept track of the most recent one.

### Inability to keep changes and contributors in check: The changes and contributors are difficult to be accounted.

### No easy rollback, version history: The traditional storage methods do not offer an extensive version history of the documents.

### Current solutions are not document-based: More useful are solutions such as Git, which are not practical to use to manage documents with other people who are not necessarily software developers.

### The current approaches are either very broad or too complex, especially to non-developers.

### **1.2 Objectives**

### Create a web-based raw version control of documents that is self-hosted: Give privacy, control of data.

### Encourage storage-based organization and systematic teamwork: rational organization by type of document.

### Used powerful version control and graceful recalculation: Keep a full history of every document.

### Offer role based, fine-grained access control: To viewer, collaborator, or admin.

### Make authentication and data security secure: authenticate and protect data with JWT tokens.

### Provide easy to use, easy to understand dashboard: Employees, casual users.

### Promote full audit logs and activities: Log every important user activity.

### Aid effective teamwork and dispute resolution: Introduce merge tools and change comparison.

### **1.3 Scope and Importance**

### MyVCS is specifically designed to be used with files in formats like .pdf, .docx, .doc and .txt, and it is thus quite useful in academic groups, offices, and research teams. MyVCS bridges the gap between advanced developer tools and the platform of common collaborators by locating version control in a newly defined document management state of affairs, thus providing an answer to the question of versioning documents. This report presents the design, implementation, testing, and results of the system followed by the discussion of its effect and potential future improvements.

### **System Design**

### This section describes the general architecture, technologies invested, the database model, and the functionality of the key components of MyVCS that make it possible to realize the version control of the documents.

### **2.1 Overview of Architecture**

### MyVCS is implemented in a modular web-based system in which there is a clear distinction between frontend, backend, database, and file store. This architecture would provide scalability, security, maintainability and ease of user experience to technical and non-technical users.

### **High-Level Architecture Diagram**

A diagram of a computer server

AI-generated content may be incorrect.

**Layer Descriptions**

Frontend (Web Browser):

* HTML, CSS (glass morphism) and JavaScript were used to develop it.
* Presents an informative dashboard that allows to manage repositories, navigate within the files, preview documents, commit the changes, revert, or merge them, and control access.
* It communicates with the backend through safe REST API requests.

Backend (Fast API):

It is written in Python, developed with Fast API: high-performance and convenient API-development.

* Manages entire business logic: authorization, authentication, files, repositories and versions, merge, and logs.
* Secures the operations in the product through role-based access and JWT access control.
* Communicate with PostgreSQL and local file system (metadata and doc storage respectively).

Database (PostgreSQL):

* Keeps all metadata: users, repositories, commits, snapshots, access control, logs, file versions, merge history, etc.
* Administered through SQLAlchemy ORM, to provide strong data modeling, and relationships.
* Has data integrity safeguards such as foreign keys, unique constraints and deletes cascade.

Storage of files (Local File System):

* Unloads the actual document file and its versioned snapshots.
* The files are sorted on a per-repository and per-version basis and paths and hashes are logged in the database to roll back and de-duplicate.
* It can only be accessed through authenticated backend API calls to be secure.

The most important Architectural elements are:

Separation of Concerns: Every layer (frontend, backend, database, file storage) is separated, and a system is simple to maintain and extend.

* Security: JWT authentication is used to protect all the API endpoints. Role-based endorsing implies that only approve users can access sensitive activities.
* Scalability: The modular design is scalable in future and can do things like distributed file storage or integrating with cloud services.
* Auditability: Each of the important actions (commit, revert, merge, change access) is traceable and responsible.
* User Experience: The dashboard is intended to be usable to both technical and non-technical users as it provides intuitive navigational control mechanisms and feedback is made clear.

### **2.2 Technology Stack**

Frontend

* HTML, CSS (glass morphism), JavaScript: HTML, CSS (glass morphism), JavaScript: These modern scripts give the dashboard response, as well as providing a modern interface in the management of the repository to navigate files, preview documents, cope with commit/revert/merge operations and access control. Initiates secure REST API requests with backend.

Backend

* Fast API (Python): A fast, high-performance, web framework designed to develop RESTful APIs. Processes the complete application logic distributions, such as the authentication processes, the repository and file actions, versioning, merging, as well as logging.
* Pydantic Data validation and settings management of Fast API.
* Uvicorn: ASGI server to use with Fast API.

Database

* PostgreSQL: Strong relational database that stores all the metadata (users, repositories, commits, snapshots, access control, logs, file versions, merging history etc.).
* SQLAlchemy ORM- Object relational mapper to Python, which allows modeling of databases and management of relationships therein easily.

Authentication

* JWT (JSON Web Tokens): Secure stateless Authentication of all API endpoints. Restricts access of or any changes to resources to authorized users.

Storage of files

* Local File System: Contains the real document files and versioned versions of the files. The files are sorted by repository, branch and version and the paths and hashes are stored to the database to support rollback and deduplication. Its content can be accessed only through authorized backend API requests.

Testing

* curl, Postman: useful tools to test the API endpoints, endpoints, and features manually and automatically.

**2.3 Data Base Model**

The schema of the database is able to provide sturdy version tracking, restriction identification, and scrutiny recording. In the section below, we have summarized the primary entities and their association with each other, using your SQLAlchemy models:

**Tables**

## **Users**: Stores user accounts.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| username | String | Unique, Indexed |
| password\_hash | String | Hashed password |

### Relationships:

### One-to-Many with repositories

### One-to-Many with access\_control

### One-to-Many with log

### 

## **Repositories**: Stores repository metadata.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| name | String | Indexed |
| owner\_id | Integer | ForeignKey to users.id |
| visibility | String | Default 'private' |
| description | Text | Optional |
| created\_at | DateTime | Auto timestamp |

### Relationships:

### Many-to-One with users

### One-to-Many with commits

### One-to-Many with access\_control

### One-to-Many with log

### 

## **Commits**: Tracks file commits and changes.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| repo\_id | Integer | ForeignKey to repositories.id |
| author\_id | Integer | ForeignKey to users.id |
| parent\_commit\_id | Integer | Self-reference |
| message | Text | Commit message |
| original\_filename | String | Original file name |
| versioned\_filename | String | Generated name |
| snapshot\_path | String | Path to file copy |
| status | String | 'proposed' or 'merged' |
| timestamp | DateTime | Auto timestamp |
| diff\_text | Text | Unified diff |

### Relationships:

### Many-to-One with repositories

### Many-to-One with users

### Self-referencing parent-child

### One-to-One with snapshot

## **access\_control**: Controls user access to repositories.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| user\_id | Integer | ForeignKey to users.id |
| repository\_id | Integer | ForeignKey to repositories.id |
| role | String | 'viewer', 'collaborator', 'admin' |

### Relationships:

### Many-to-One with users

### Many-to-One with repositories

### 

## **log**: Audit logs of actions.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| user\_id | Integer | ForeignKey to users.id |
| repo\_id | Integer | ForeignKey to repositories.id |
| commit\_id | Integer | ForeignKey to commits.id |
| action | String | Action name |
| description | Text | What was done |
| timestamp | DateTime | Auto timestamp |

### Relationships:

### Many-to-One with users

### Many-to-One with repositories

### 

## **snapshot**: Stores actual file snapshots.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| commit\_id | Integer | ForeignKey to commits.id |
| file\_path | String | Snapshot file path |
| content\_hash | String | File content hash |
| size | BigInteger | File size |
| created\_at | DateTime | Timestamp |
| is\_deleted | Boolean | Soft delete |
| operation | String | 'add', 'modify', 'delete' |

### Relationship:

### Many-to-One with commits

### 

## **file\_versions**: Tracks latest version of each file.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| repo\_id | Integer | ForeignKey to repositories.id |
| original\_filename | String | Original name |
| latest\_version | Integer | Version number |

### Relationship:

### Unique per (repo\_id, original\_filename)

### 

## 

## **file\_version\_history**: History of file versions.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| repo\_id | Integer | ForeignKey to repositories.id |
| original\_filename | String | Original file name |
| version\_no | Integer | Version number |
| commit\_id | Integer | ForeignKey to commits.id |
| versioned\_filename | String | e.g. main\_v2.py |
| snapshot\_path | String | Snapshot path |
| merged\_by | Integer | User who merged |
| timestamp | DateTime | Auto timestamp |

### Relationship:

### Unique per (repo\_id, original\_filename, version\_no)

### 

## **merge\_history**: Tracks merged commits and outcomes.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Constraints |
| id | Integer | Primary Key |
| repo\_id | Integer | ForeignKey to repositories.id |
| base\_commit\_id | Integer | ForeignKey to commits.id |
| merged\_commit\_id | Integer | ForeignKey to commits.id |
| result\_commit\_id | Integer | ForeignKey to commits.id |
| result\_filename | String | Merged file name |
| merged\_by | Integer | User who merged |
| timestamp | DateTime | Timestamp |

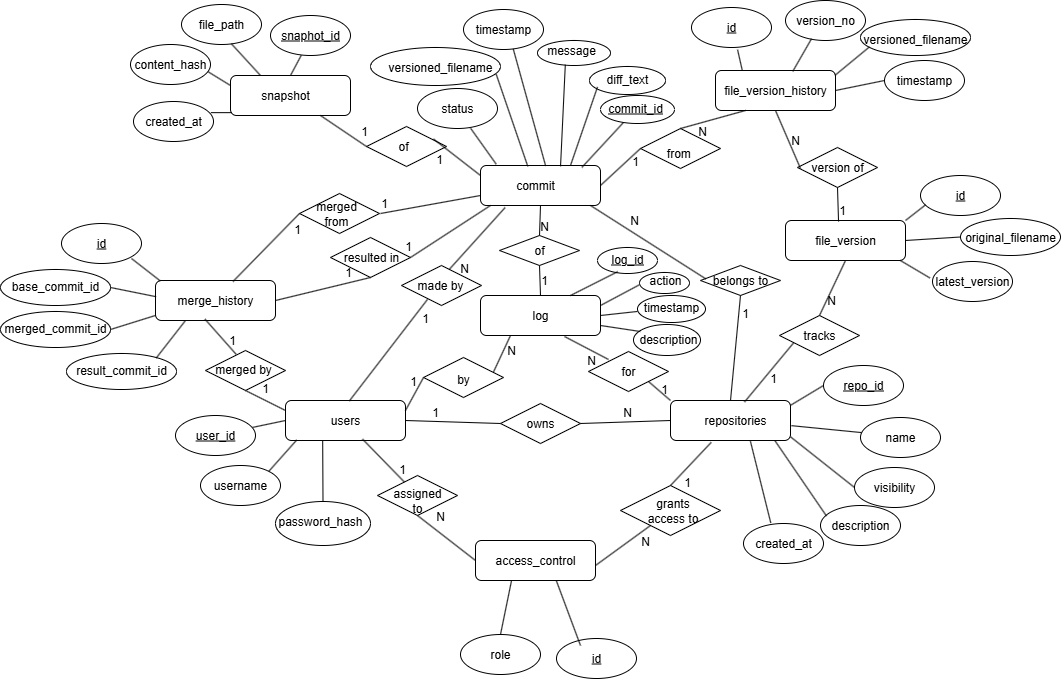
### Relationship:

### Links 4 commits to result

### 

### 

### **Entity-Relationship (ER) Diagram**



#### 

#### **Entity Descriptions**

## User: Hashed passwords were used in registering users on the stores. The users will be able to own repositories, access other ones, and create logs.

## Repository: An array of documents. Every repository is owned, seen publicly/privately and has a description.

## Commit: Every commit notes a modification of a file, such as the author, parent commit and message, and also a reference to the snapshot (real contents of the file). Facilitates parent-child relations of version trees.

## Snapshot: Memorizes the content, hash and metadata of each file in each commit such that it can be reverted, and dupes can be removed.

## AccessControl: Gives each user a role (viewer, collaborator, admin) at a per-repository level to use fine-grained access control.

## Log: Stores user activities (e.g. commit, revert, merge) in order to have audit trails and activity history.

## FileVersion: Monitoring the number of the latest version on every file on a repository so that there is different versioning.

## FileVersionHistory: This keeps a record of all historic versions of a file as referenced against the creation commit and snapshot.

## MergeHistory: Records information about file merges, including base and merged commits, result commit and the user who carried out the merge.

## Summary

### The system design of MyVCS ensures a robust, secure, and user-friendly platform for document version control. By combining a modern web frontend, a scalable Fast API backend, a normalized PostgreSQL schema, and secure file storage, MyVCS delivers all the essential features needed for collaborative document management in academic, office, or research environments.

### **3. Implementation**

## This section describes how the MyVCS system was built, focusing on the backend logic, database integration, versioning mechanisms, access control, and the frontend dashboard. Code snippets and explanations are provided to illustrate key components.

### **3.1 Backend Implementation**

#### **3.1.1 Project Structure**

### The backend is built with Fast API (Python), and SQLAlchemy ORM was chosen to model databases, and PostgreSQL is the database. The document files as well as the snapshot versions of the documents are maintained at the local file system of the server.

### Key Directories:

### server/models.py — SQLAlchemy ORM models (models.py)

### server/routes/ — Fast API route handlers for authentication, repositories, files, commits, merges, etc.

### server/db.py — Database session and engine configuration

#### **3.1.2 API Endpoints**

All the primary operations have endpoints that are made available through the backend in the form of RESTful endpoints. JWT has applied authentication to every endpoint and role-based access control.

## 1. Authentication APIs

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /auth/register | Register a new user |
| POST | /auth/login | User login, returns JWT token |

### 

## 2. Repository APIs

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /repositories/create | Create a new repository |
| PUT | /repositories/{repo\_id} | Update repository details |
| DELETE | /repositories/{repo\_id} | Delete repository |
| PUT | /repositories/{repo\_id}/visibility | Change visibility (private/public) |
| GET | /repositories/public-repositories | Get list of public repositories |
| GET | /accessible-repositories | Get list of accessible repositories |

### 

## 3. File Upload & Management APIs

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /repositories/{repo\_id}/upload | Upload a file to repository |
| DELETE | /repositories/{repo\_id}/file | Delete a file |
| POST | /repositories/{repo\_id}/restore\_file | Restore deleted file |
| GET | /repositories/{repo\_id}/trash\_files | List trashed files |
| POST | /repositories/{repo\_id}/create\_file | Create a new empty file |
| POST | /repositories/{repo\_id}/create\_folder | Create a folder in repository |
| GET | /repositories/{repo\_id}/files | Get file tree |
| GET | /repositories/{repo\_id}/file | Get file content |
| PUT | /repositories/{repo\_id}/edit\_file | Edit file content |
| GET | /repositories/{repo\_id}/edit\_file\_text | Convert file to editable text |

### 

## 4. Commit & Version APIs

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /repositories/{repo\_id}/commit | Create a commit |
| GET | /repositories/{repo\_id}/file\_commits | Get commit history of a file |
| POST | /repositories/{repo\_id}/revert/{commit\_id} | Revert to a specific commit |
| POST | /repositories/{repo\_id}/revert\_version/{version\_filename} | Revert a version file |
| GET | /repositories/{repo\_id}/commit\_preview/{commit\_id} | Preview commit changes |
| GET | /repositories/{repo\_id}/commit\_file | Get committed file content |
| GET | /repositories/{repo\_id}/versions | List version history of files |
| GET | /repositories/{repo\_id}/version\_files | Get all versioned files |
| POST | /repositories/{repo\_id}/merge | Merge multiple versions or commits |
| POST | /repositories/{repo\_id}/merge/{commit\_id} | Merge a specific commit |
| GET | /repositories/{repo\_id}/file\_merge\_info | Get merge info |
| POST | /repositories/{repo\_id}/revert\_version\_commit/{version\_filename} | Revert a versioned commit file |
| GET | /repositories/{repo\_id}/convert\_version | Convert version to latest format |
| GET | /repositories/{repo\_id}/download\_version | Download version file |

### 

## 5. Snapshot APIs

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /repositories/{repo\_id}/snapshot\_repo | Take full snapshot of repository |
| GET | /my\_snapshots | List personal snapshots |
| POST | /my\_snapshots/restore | Restore snapshot from ZIP |
| DELETE | /my\_snapshots/{snapshot\_name} | Delete a snapshot archive |

### 

## 6. Access Control APIs

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /repositories/{repo\_id}/access-control | Update access (add/remove users) |
| GET | /repositories/{repo\_id}/role | Get current user’s role in repo |

### 

## 7. Logs & Stats

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| GET | /repositories/{repo\_id}/log | Get repository logs |
| GET | /users/{user\_id}/log | Get user activity logs |
| GET | /repositories/{repo\_id}/stats | Get repo statistics |
| GET | /api/superadmin/logs | All logs (admin view) |

### 

## 8. Superadmin APIs

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| GET | /superadmin-login | Login form |
| POST | /superadmin-login | Superadmin login |
| GET | /superadmin | Superadmin dashboard |
| GET | /superadmin/section/{section} | Load dashboard section |
| GET | /api/superadmin/users | Search users |
| GET | /api/superadmin/users-reset | List all users |
| DELETE | /api/superadmin/users/{user\_id} | Delete a user |
| GET | /api/superadmin/repositories | List all repositories |
| DELETE | /api/superadmin/repositories/{repo\_id} | Delete a repository |
| POST | /api/superadmin/repositories/{repo\_id}/toggle | Toggle repository visibility |
| GET | /api/superadmin/logs/user/{user\_id} | Logs by user |
| GET | /api/superadmin/logs/repo/{repo\_id} | Logs by repository |
| POST | /api/superadmin/reset-password/{user\_id} | Reset password |
| GET | /api/superadmin/impersonate/{user\_id} | Login as another user |
| GET | /impersonated | Confirm impersonation |
| GET | /superadmin-logout | Superadmin logout |

### 

## 9. UI Routes

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| GET | / | Main login form |
| GET | /login | Login page |
| POST | /login | Login POST |
| GET | /logout | Logout and clear session |
| GET | /dashboard | User dashboard (authenticated) |
| GET | /repositories/{repo\_id}/dashboard | Repo dashboard UI |

### 

### **3.1.3 Key Backend Modules**

* Authentication & Authorization: JWT-based secure authentication; all of the sensitive operations are guarded with a role-based access scheme.
* Repository and File Management: CRUD interaction with repositories and files (upload, edit, and delete, and commit).
* Versioning Logic: The change of the file generates a new commit and snapshot; files are tracked per version numbers.
* Merge & Revert: Admins have the power to merge file versions or move the file back to its previous versions, and all the actions will be logged and versioned.
* Audit Logging: All major user activity is logged within the log table so that they are traceable.

#### **3.1.4 Versioning Logic**

#### MyVCS has its own version control logic based on Git referring to non-code documents such as .docx, .pdf, and .txt. MyVCS, unlike Git, removes command-line functionality and branching logic into an intuitive web user interface and well-defined backend API that is easy to use by an end user or a non-developer. This system enables users to see the transformation of any file by time, offers managing of files of more than one version of documents, overview or rollback previous states of files, and even join different versions of files with the administration control.

#### **Versioning Design Philosophy**

#### The key objective of the version control logic design has been:

#### Not to code, to support document-based collaboration.

#### To offer the rollback and merge facilities in every file.

#### To maintain an overview of who did what and when, including 100 percent audit trails.

#### To reduce information losses, as well as overwriting files.

#### To enable non-technical users to use powerful functionality of version control through

|  |  |
| --- | --- |
| **Component** | **Description** |
| Commit | Records every change made to a file. Contains commit message, file info, author, timestamp, and links to snapshots. |
| Snapshot | Stores actual file content and metadata (path, size, hash). Each commit has one snapshot, which allows rollback. |
| FileVersion | Tracks the current/latest version of each file in a repository. Ensures that version numbers increase consistently. |
| FileVersionHistory | Maintains full history of version numbers and filenames for each file, linking to corresponding commits and snapshots. |
| MergeHistory | Logs administrative merges between file versions. Tracks source, base, and result of merge. |
| Log | Logs all versioning operations (commit, revert, merge) for auditability. |

#### **3.1.5 Core Versioning Logic Components**

#### A commit-snapshot version model drives the system, the most important of which are tables and roles listed below:

#### **Workflow of Step-by-Step Versioning**

#### **A. Uploading or creating files**

#### In case of a new file uploaded by a user:

#### It is saved within the file system under the repository directory.

#### A message “Initial Upload Commit” is created.

#### Full metadata Snapshot is generated.

#### The newest version = 1 is added in the FileVersion.

#### Version 1 is given FileVersionHistory entry.

#### **API Used**: POST /repositories/{repo\_id}/upload

#### **B. Checking In**

#### The user can commit the change when a file is edited (UI or API):

#### A new version number using FileVersion which is computed.

#### A new file such as doc\_v2.docx is saved in versions/ directory.

#### A Snapshot is made and associated with this commit.

#### Commit stores parent\_commit\_id to chain up build.

#### The file version is changed to reflect the new file version.

#### A new row is added into FileVersionHistory.

#### **API Used**: POST / repositories / {repo id} / commit

#### Some of the fields I used might include original\_filename, versioned\_filename, snapshot\_path, status, diff\_text.

#### **C. Un-Checking a file.**

#### A user can undo a particular file to a prior held version:

#### The system gets the relevant Snapshot of FileVersionHistory with the help of commit\_id or version\_filename.

#### The snapshot content is restored to the last file version.

#### This automatically creates a new commit that uses message = Reveted to vX.

#### A Log saves a rollback log.

#### **API Used:**

#### • POST /repositories/{repo\_id}/revert/{commit\_id}

#### • POST /repositories/{repo\_id}/revert\_version/{version\_filename}

#### **D. Files Merging**

#### Admins are able to combine variants of the same file (say v2 and v3) into a new one v4:

#### The UI retrieves the differences with the help of diff\_text.

#### The two versions get downloaded by the admin, merged manually, and then uploaded, by checking the merged file.

#### A new commit and snapshot is made.

#### MergeHistory records both commits on source and output.

#### FileVersion and FileVersionHistory are adjustable.

#### **API Used**: POST /repositories/{repo\_id}/merge

#### **Merge Info**: base\_commitid, merged\_commitid, result\_commitid

### **E. Versioned Filenames**

### Every version of a file is stored using a well-organized name such as:

### makefile

### CopyEdit

### original: budget.docx

### versioned: budget\_v3.docx

### path: /repo\_id/versions/budget\_v3.docx

### The giving of this name makes all versions not overwritten and thus be accessed individually.

### This is a logic that occurs when you create a commit and get a versioned\_filename.

### **3.1.6 Audit Trail and Logging**

### Every version-related action is logged into the Log table:

|  |  |
| --- | --- |
| **Field** | **Value** |
| user\_id | Who did it |
| repo\_id | On which repo |
| commit\_id | Which commit was involved |
| action | commit, revert, merge, etc. |
| description | Commit message or explanation |
| timestamp | Auto-filled |

### **API Used:** GET /repositories/{repo\_id}/log

### **Benefits of Custom Version Control Logic in MyVCS**

|  |  |
| --- | --- |
| **Feature** | **Benefit** |
| Document-Oriented | Built for .docx, .pdf, .txt instead of code |
| Simplified Workflow | Abstracts complex versioning behind one-click UI |
| Secure File Access | Files stored outside web root, accessed via token-protected APIs |
| Full Audit Trail | All actions logged for traceability and accountability |
| Rollback Support | Revert files to any historical version |
| Manual Merge Control | Admins can merge conflicting versions |
| Scalable | Clean separation of metadata (PostgreSQL) and file storage (disk) |

### **Conclusion of Version Control Logic**

### MyVCS version control logic is a stable, easy to use and safe way of document change tracking. MyVCS integrates commit-based tracking, versionable snapshots, and structured metadata to offer Git-like capabilities to non-developer teams at universities and in organizations in an easy to use and completely controlled manner, without depending on new tools and interfaces.

**3.1.7 Access Control**

MyVCS supports a repository-scoped role-based access control (RBAC) with the help of the access\_control table. This system also makes sure that different users can only carry out the actions as per the assigned roles. It will allow working safely and maintaining the integrity of data.

**Roles in MyVCS**

Any user of a repository is assigned one of the following roles:

* **Admin**: He has full access to the repository. They are able to configure repository settings, add and remove files, commits, merges, snapshots and even change access control by addition or removal of other users.
* **Collaborator**: Is able to upload files, edit and commit them. The collaborators have however, no freedom to modify the access privileges or to delete the repository.
* **Viewer**: Just able to view and look at version history. They are unable to make any modifications, post files, and do commits.

The owner of the repository automatically get the role of the administrator when the repository is created and can also manage the roles of other users.

**Implementation Details**

The table access\_control keeps track of the correspondence which is between:

* user\_id - the accessable user
* repository\_id - the concerned repository
* role - role assigned to the user (viewer, collaborator or admin)

Before performing any sensitive action (committing a file, undoing to a previous revision, deleting some content, assigning access), the backend asks the current user their role on this repository. This is enforced with the help of Fast API through a list of dependencies and intimate logic in acquiring role information in the access piece of control table.

**Key Points:**

* Legacy: You are only allowed to change the repository settings or to manage your collaborators as an admin.
* Collaborators are able to create commits yet they are incapable of altering access.
* Viewers simply have read-only operations.
* The role enforcement occurs at API level which blocks unauthorised access in case of the bypass of frontend restrictions.

This modular role system is also granular that way that it offers flexible but secure collaboration among the people with varying responsibilities.

**3.1.8 Security & Authentication**

The MyVCS system is set on solid security principles so as to allow data privacy and security as well as controlled access.

**Authentication**

Stateless and secure user sessions are made with JWT (JSON Web Tokens).

* The token is provided after successful login and it should be prefixed to each API request through the Authorization header.
* Tokens consist of expiration and identity claims about the user, which are validated per request.

**Authorization**

* Role-based access checks are applied to each sensitive route that a user may access (such as commits, merges, deletes, etc.).
* Malicious users who have tried to perform unauthorized actions are also served with adequate HTTP error messages.

**Security of file storage**

* Files and snapshots are stashed in non web root directory.
* Authenticated backend API calls are limited to access (no file URL access is available).
* Existing file paths are cleaned and verified to avoid path-traversal attacks.

**3.2 Frontend Implementation**

**3.2.1 Overview**

The frontend of MyVCS is patterned as a powerful, responsive web interface and offers an accessible and friendly user interface to handle the repositories of documents and version management processes. Engineered with HTML, CSS (glass morphism design principles), and JavaScript, the frontend permits users to use the backend APIs without any disruption to the routine, making the processes comfortable to both technical and non-technical users.

Frontend Main goals are:

* In order to provide the dashboard, through which users can view, create and manage repositories.
* To ensure that it has a file tree navigation system through which documents of each repository can be browsed and ordered.
* In order to have inline document preview of supported files formats like ￼, ￼, and ￼.
* Make version control tasks such as committing, revert to earlier versions, merging and viewing commits easier.
* To provide the possibility of role-based access control such that the user is only able to view and it acts on the things within its assigned roles (admin, collaborator, viewer).
* To show the list of activity logs and system notifications in order to be transparent and auditable.

The frontend and the backend interact by sending RESTful API requests only, and JWT tokens ensure secure authentication and authorization. Any operation that involves any kind of sensitivity (e.g.: file upload, modification, and access control) is secured and a valid user session is necessary.

It is easy to understand and has an accessible user interface that uses a clean layout, curative navigation menus, and action buttons. The methods of responsive design will make the dashboard user-friendly on diverse devices, including desktops to tablets.

**3.2.2 Features**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Repository List | Displays all accessible repositories |
| File Tree | Navigable folder/file structure |
| Document Viewer | Inline preview for .txt and .pdf files |
| File Commit | Upload or commit with message |
| Version Revert | Rollback to earlier file version |
| Merge Versions | Admin merge utility |
| Access Control | Collaborator role management |
| Logs View | Displays recent repository actions |

### The main features of the frontend are summarized in the following table.

### **4. Use Case & Features**

### This paragraph includes the description of the most important MyVCS system use cases and characteristics, which are planned for different user roles. The platform is allowed to have the use of Superadmin roles, Admin roles, Collaborator roles, and Viewer roles with varying varieties of access and control.

### **4.1 Superadmin Features**

### Superadmin has a globally privileged user dashboard or Superadmin Dashboard.

### **4.1.1 Dashboard Overview**

### View number of users, number of repositories and their public/private distribution.

### **4.1.2 Users Section**

### See all the registered accounts including their IDs.

### Permanently remove a user within the system.

### Impersonate a user and directly access their dashboard without requiring a password by clicking on their name.

### **4.1.3 Repositories Section**

### Access to all the repositories and: Name, Visibility, Owner and Repository ID.

### Make any repo visible or private.

### Remove permanently any repository.

### **4.1.4 Activity Log**

### Choose a user or a repository and see a full log activity pertaining to the respective user or repository.

### **4.1.5 Reset Password**

### Re-set the password of any user-account.

### **4.1.6 Register User**

### Register a new user by entering personal credentials-username and password.

### **4.1.7 Logout**

### Log off on the Superadmin dashboard safely.

### **4.2 User Features**

### The availability of a User Dashboard with several sections to the authenticated users:

### **4.2.1 Home Section**

### Make another repository.

### During the creation of the repository, it is possible to upload a file.

### **4.2.2 Your Repositories Section**

### Shows all the repositories the user has or has been given the rights to.

### Look at the: Name, Description, Owner, whether visible, the date they were created and Role assigned.

### In case the user is the Owner (Admin):

### Share with other users (viewer/collaborator).

### Cancelling the repository.

### Public/private.

### Make a snapshot of the whole repository.

### To open the repository dashboard, click on the name of the repository.

### **4.2.3 Public Repositories Section**

### View all public repositories.

### Open any public repo in repo dashboard for viewing its content.

### **4.2.4 Activity / History Section**

### View activity logs of all owned repositories.

### Filter logs by action type (e.g., commit, merge) or repository name.

### **4.2.5 Snapshot Section**

### View all created snapshots.

### Restore any snapshot as a new repository.

### **4.2.6 Logout**

### Log out of the user dashboard securely.

### **4.3 Repository Dashboard Features**

### All repositories are equipped with interactive dashboard, where features depend on the level of access to them.

### **4.3.1 Common Features (All Roles)**

### View all files and folders in the repository tree.

### View the content of each file inline.

### **4.3.2 With Write Access (Collaborator/Admin)**

### View commit history of files.

### Edit files and commit changes with commit messages.

### **4.3.3 Admin-Only Features**

### Merge commits to create a new version of the file.

### Merge versions of files to create a new version.

### Delete files from the repository.

### Revert files to previous versions using commit/version.

### Upload new files to the repository.

### Create new empty files or folders.

### **4.4 Role Summary**

|  |  |
| --- | --- |
| **Role** | **Access Summary** |
| Superadmin | Global access to all users, repositories, logs, impersonation, and password reset. |
| Admin | Full control over owned repositories including access, file management, merge, revert. |
| Collaborator | Read/write access to assigned repositories; can commit but not manage access. |
| Viewer | Read-only access to repositories and version history. |

### **5. Testing and Results**

### MyVCS system was tested to the maximum to verify the functionality, consistency and compliance to the anticipated version control behaviors. The backend APIs, as well as frontend interfaces, have been tested with the real-life scenario with multiple users and repositories.

### **5.1 Testing Methodology**

### Testing was carried out through the combination of:

### **Unit Tests** – To verify each and every activity and in all databases.

### **Manual API Testing** – By curl, Postman, and Swagger UI.

### **UI Testing** – Web dashboard- interactions verification manually.

### **Integration Testing** – Mimicking user processes like design of repositories, file versioning, addition and roll-back of the files.

### **5.2 Test Cases**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Action** | **Expected Output** |
| User Registration | Register new user via /auth/register | Success message, user in DB |
| User Login | Login via /auth/login | JWT token generated |
| Create Repository | Create repo with name/desc | Repository in DB |
| Upload File | Upload main.py | File in file tree |
| Edit and Commit | Edit file, commit | New version saved |
| View Versions | Fetch /versions | Version history displayed |
| Revert Commit | Revert using commit ID | Old state restored |
| Merge Versions | Merge two version files | Merged file created |
| Delete File | Soft-delete file | Moved to trash |
| Restore File | Use /restore\_file | File restored |
| Access Control | Assign role to user | Collaborator can edit |
| Snapshot | Take & restore snapshot | Repo restored |
| Impersonate | Superadmin impersonates | Acting as target user |
| Logs | Fetch /logs/user | Chronological actions |
| Role Check | Get role for user/repo | Role string returned |
| UI Testing | Browser-based UI tests | Files, versions, actions working |

### **5.3 Sample Output Screens**

### **Login Dashboard** –Successful authentication loads a dashboard that is unique and customized.

### **Repo View Panel** – Tree file explorer with file viewers panel.

### **Access Control Panel** – Enables one to handle rights through a neat user interface.

### **Snapshot Manager** – Shows snapshots and restores them.

### *(Note: Screenshots can be added in the appendix of the report.)*

### **Conclusion**

### The MyVCS (My Version Control System) project is a working project providing an all-purpose version control system based on the GitHub code base adapted to collaborative file management, version control, and sharing of repositories. MyVCS offers a user-friendly interface with a responsive and modern front-end combined with a custom backend developed based on Fast API and PostgreSQL, which provides a simplified process of creating, uploading, committing, reverting, and merging the versions of files across repositories.

### The system supports the key principles of the version control system, including:

### Snapshot-based version tracking

### Commit history with a diff management

### Role-based access control (RBAC)

### Audit logging of all the critical operations

### Repository visibility toggle (public/private)Merge and revert functionality

### The testing of the dashboard and REST API

### Those experiences have enhanced my awareness on how to achieve full-stack development, as well as, secure collaborative system design

### .MyVCS is a versatile, scalable and extendable approach, having regards to how to address the gap between complex developer tools and the requirements of ordinary users in accessing collaborative document workflows, which also enhances data integrity.

### Future solutions may involve expanding support to other file types, providing robust diff and merge strategies, and adding mobile-compatible user interface. In general, such an experience has been priceless, and this project added an effective tool to manage documents version control in a team environment.

## 

### 3. References

List all documentation, libraries, and resources you used.

#### Sample List:

* [Fast API Documentation](https://fastapi.tiangolo.com/)
* [SQLAlchemy Documentation](https://docs.sqlalchemy.org/)
* [PostgreSQL Documentation](https://www.postgresql.org/docs/)
* [JWT.io](https://jwt.io/)
* [Your MyVCS GitHub Repo](https://github.com/unnatissaini/myvcs) *(if public)*
* Internship Report Format

### 4. Appendix

Include screenshots, code snippets, diagrams, or any extra material that supports your report.

#### Suggestions:

* Dashboard UI screenshots (repository list, file tree, version history)
* Sample API requests/responses
* Code snippets for key modules (e.g., commit creation, merge logic)
* ER diagram (text or graphical)