**Project Title** : CompactVCS

**Name** : D.Anish Reddy

**Roll No** : 23B81A05D4

**Batch** : 2

**Mentor Name** : Karunakar Eede

**Abstract :**

CompactVCS addresses the challenge of efficiently managing and tracking changes to software projects, a task crucial to both collaboration and code quality in modern development. Without version control, teams and individuals risk losing work, struggling with code conflicts, and duplicating effort—a scenario that can seriously hinder productivity and project stability.

Many existing solutions can be complex or require significant technical overhead, making them less accessible for educational, lightweight, or custom deployment needs.

The proposed solution, CompactVCS, is a mini version control system built with pure object-oriented programming concepts and backed by Supabase for robust cloud-based storage of files and version metadata. By automating the tracking of changes, managing historical snapshots, and supporting secure remote storage and retrieval, CompactVCS streamlines team workflows and ensures project integrity even in distributed environments.

This project is important because it makes version control concepts accessible and practical for learning and prototyping, while also empowering small teams or individuals to safeguard their code with a simple yet effective system.

**Introduction:**

**Background of the Domain**

* Version Control Systems (VCS) play a crucial role in modern software development by tracking changes, enabling collaboration, and preserving project history. While powerful tools like Git and SVN are widely used, they can be complex and overwhelming for beginners or for small-scale projects where advanced features may not be necessary. A compact and lightweight VCS provides a more approachable way to understand the underlying principles of version control while still maintaining essential functionality.Version Control Systems (VCS) play a crucial role in modern software development by tracking changes, enabling collaboration, and preserving project history. While powerful tools like Git and SVN are widely used, they can be complex and overwhelming for beginners or for small-scale projects where advanced features may not be necessary. A compact and lightweight VCS provides a more approachable way to understand the underlying principles of version control while still maintaining essential functionality.

## **Motivation for the Project**

The motivation behind CompactVCS stems from the need to create a simpler alternative to existing version control tools, particularly for students, hobbyists, and small teams. By implementing the core ideas of version control—such as commits, branching, and rollbacks—using object-oriented programming (OOP) concepts, this project not only provides a learning experience but also a practical tool. Integrating Supabase as a backend adds persistence and remote storage, making it possible to manage versions beyond just local environments.

## **Key Objectives**

* Provide a basic VCS system to track file changes and project versions.
* Support essential operations: initialize repository, commit changes, view history, and rollback.
* Apply OOP design principles to build a modular and extensible system.
* Use Supabase for storing version metadata and enabling remote persistence.
* Deliver a lightweight, user-friendly, and educational solution.

**Problem Statement :**

Traditional version control systems such as Git and SVN are highly feature-rich and optimized for large-scale software development. However, their complexity and steep learning curve often make them difficult for beginners, students, or small teams to adopt effectively. For educational purposes and lightweight projects, only a subset of version control features—such as tracking changes, committing updates, and rolling back to previous versions—is truly necessary. The lack of a simple, minimal, and user-friendly VCS solution creates a gap for learners who want to grasp the fundamentals without being overwhelmed by advanced concepts like distributed workflows, rebasing, or merging strategies.

This project focuses on building a lightweight, compact version control system (CompactVCS) that demonstrates the core principles of version control in an accessible way. It aims to provide a practical and educational tool that helps users understand the inner workings of VCS while remaining simple, user-friendly, and extensible for small-scale projects. By leveraging object-oriented programming (OOP) concepts and Supabase for storage, CompactVCS bridges the gap between theoretical learning and real-world application.

**Scope of the Project :**

This projects includes :

* Repository initialization and configuration.
* File tracking (add/update/remove files).
* Commit management (create, rollback, view history).
* Branch management (create, switch, merge branches).
* CLI interface for user interaction.
* Basic logging and history viewing.

Exclude:

* Graphical user interface(GUI) or moblie/web app
* Advanced Analytics
* Distributed version control (like remote push/pull).
* Multi-user concurrency handling.

**System Requirements:**

### **Functional Requirements (Features):**

* Initialize and configure repositories.
* Add/Edit/Delete files from version control.
* Commit changes and maintain commit history.
* Manage branches (create, switch, merge).
* Rollback to previous versions.
* View logs and history of changes.
* Provide a simple CLI for user operations.

### **Non-Functional Requirements:**

* Security: Ensure integrity and confidentiality of stored data.
* Scalability: Should handle large numbers of files and commits efficiently.
* Usability: Provide an intuitive CLI with clear commands and help options.
* Performance: Commit and retrieval operations should be fast.
* Reliability: The system should not lose data during crashes or rollbacks.

**System Analysis:**

## **Existing System :**

* Developers often use manual backups, simple file copies, or basic tools like zip folders / cloud storage to maintain versions.
* Tracking changes and rollbacks using such methods is inefficient and prone to mistakes.
* Collaboration and branch management are either manual or non-existent.

## **Proposed System**

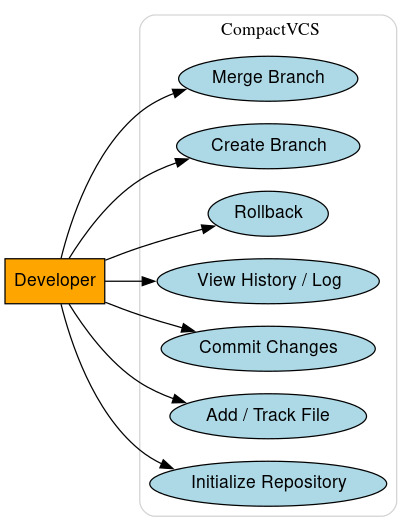
* An automated lightweight version control system (VCS) built with Python and database connectivity.
* Provides structured modules for repositories, commits, branches, and rollbacks.
* Offers a command-line interface (CLI) to easily execute version control commands.

## **Advantages of Proposed System**

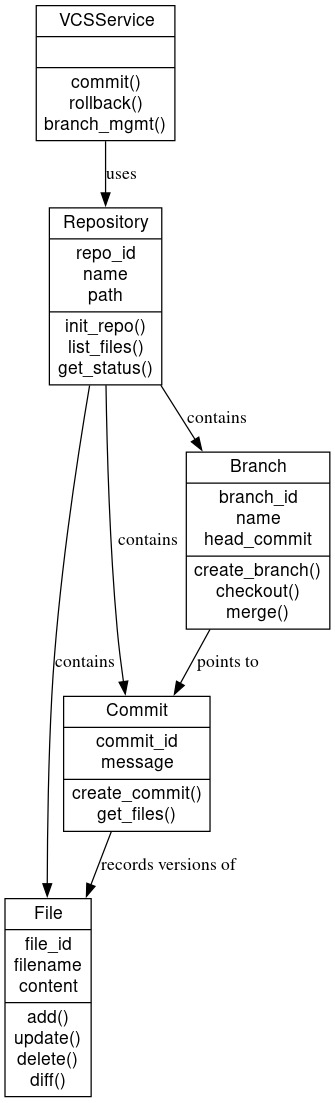
* Reduces human error in tracking versions.
* Saves time compared to manual backups or file renaming.
* Provides commit history and rollback options.
* Improves collaboration with branch management.
* Ensures data integrity with database storage.

**System Design :**

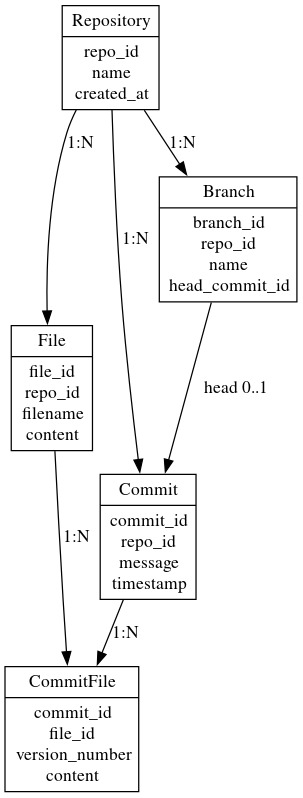
**UseCase Diagram :**

****

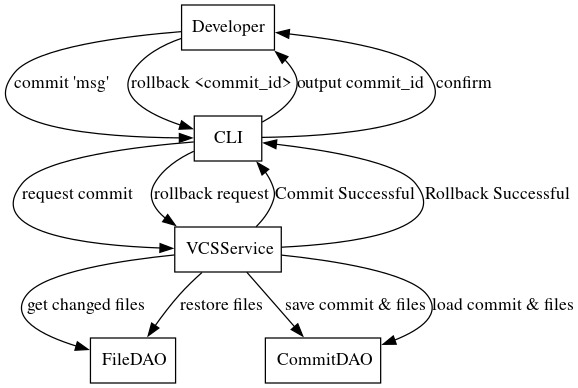
**Class Diagram :**

****

**ER Diagram :**

****

**Sequence Diagram :**

****

**DataBase Design :**

Tables and Attributes (descriptive, no SQL code):

* **Repository**: repo\_id,name,created\_at– stores information about each repository.
* **File:** file\_id,repo\_id,filename,count – stores files tracked in repositories.
* **Commit**: commit\_id,repo\_id,message,timestamp – stores commit metadata for version tracking.
* **CommitFile:** commit\_id,file\_id,version\_number,content– stores specific versions of files in a commit.
* **Branch:** branch\_id,repo\_id,name,head\_commit\_id– stores branch details and points to the latest commit.

**System Architecture :**

**CLI → Service Layer → DAO → Supabase (Database)**

**CLI (Command Line Interface)** – Developer interacts with CompactVCS using commands like commit, rollback, branch, etc. Handles input and output.

**Service Layer** – Implements core VCS logic: committing changes, tracking file versions, branching, and rollback. Coordinates with DAOs for database operations.

**DAO (Data Access Object)** – Abstracts database interactions for repositories, files, commits, and branches. Responsible for saving, updating, and retrieving data from Supabase.

**Supabase (Database)** – Stores persistent data for repositories, files, commits, commit files, and branches.

### **Expected Outcomes**

* Track file changes and maintain version history for repositories.
* Commit, rollback, and merge operations work reliably across files and branches.
* Branch management for parallel development.
* Simple CLI interface for developers to perform all VCS operations.
* Accurate and consistent repository state maintained in the database.

**Future Enhancements**

* GUI/Web interface for a more user-friendly experience.
* Multi-repository support with project-level organization.
* Real-time visualization of branches and commit history.
* Integration with external tools like IDEs or cloud storage.
* Advanced merge conflict resolution and notifications.

### **Conclusion**

**CompactVCS** is a lightweight version control system designed to efficiently track files, commits, and branches through a simple CLI interface. By storing metadata and file versions in a structured database, it ensures accurate version history, easy rollback, and seamless branch management.

This system solves the problem of maintaining organized code repositories, providing developers with a reliable, extendable, and easy-to-use tool. Its modular architecture allows for future enhancements such as GUI, multi-repository management, and integration with developer tools, making it practical for both personal projects and small team collaborations.