COMP3000 Computing Project

2024/2025

Janus Version Control System

Links

Source code:
https://github.com/benjaminsanderswyatt/Janus-Version-Control

Monday link (This will act as an alternative to Trello):
https://riley636485.monday.com/boards/1651099335/views/20696102

Project Vision

The Janus Version Control System is a solution designed to assist developers, project managers and organisations who need a secure, customizable and scalable tool to manage different versions of their projects, on one or more devices.

Janus combines a Command Line Interface (CLI) for local version control, with a Dockerised web application for remote access to their projects, allowing users to manage, commit and push their changes from one device to be stored remotely and access them from any other device.

Unlike cloud systems like github, Janus will be run completely on Docker and can thus be deployed internally within organisations. This internal deployment offers enhanced security, as sensitive data about the project remains within the company's control, which can ensure that the organisations security regulations are maintained. The use of Docker also means that the Janus system can be scaled both horizontally and vertically, letting the system grow to suit the size and complexity of the users' projects.

In addition, the Janus CLI is designed to be customisable allowing organisations to enhance the system to suit their needs. By utilising the ability to add plugins to the CLI, Janus can be used and adapted to meet many more unique issues or requirements the users might have, overall improving their efficiency in development their projects.

The security, customisation and scalability provide a great solution for organisations and teams who want more control over their version control system and development of their projects.

User Stories

Command Line Interface (CLI)

As a *(role)*	**I want *(goal)***	**So that *(benefit)***	**Acceptance Criteria**
Developer	Initialize a local repository	I can start tracking the changes in my project from command line	The repository is created locally with a hidden .janus folder.
Developer	Add files to the repository	I can stage specific files for version control	Staged files are 'marked' for the next commit. Success message is given to the user.
Developer	Commit changes to the local repository	I can create a instance of my project at a specific point in time	The changes are committed with a message. A success message is given to the user.
Developer	View the commit history from the CLI	I can review past versions and changes made	A list of previous commits, timestamps & messages is displayed.
Developer	Revert to a previous commit	I can restore my project to an earlier state	The local project files are reverted back to a previous state. Success message is given to the user.
Developer	Handle merge conflicts in the CLI	I can resolve conflicts when pushing to the remote repository	Conflicting files are shown. The wanted version can be chosen.
Developer	Push my changes to a remote repository	I can share my changes with others	The changes are pushed to the remote repository. A success message is given to the user.
Developer	Push my changes to a remote repository	I can access my changes from another device	The changes are pushed to the remote repository. The changes can be fetched from the remote repository.

Web Application

As a *(role)*	**I want to *(goal)***	**So that *(benefit)***	**Acceptance Criteria**
User	Log into the web app securely	I can access my projects from anywhere	The user is authenticated with JWT tokens and logged in.
User	Mange my account details	I can change my details (password, username…)	Users can change any information on their account.
User	Be able to delete my account	I can remove my data and stop using the system	The user's account is deleted along with all related data.
User	To view all my repositories	I can see all the projects I am working on	The web app displays all repositories the user has access to.

User	To view the commit history of a repository	I can see the modifications which have been made	A commit history with timestamps & messages are can be seen for the repository.
User	Download projects from a previous commit	I can retrieve a previous version of the project	The user can select a commit and download the files from that version.
Owner	Change who has access to the repository	I can control who can view or modify the repository	The owner can add and remove users with access.
Owner	Change the visibility of a repository	I can make the repository public or private	The owner can change the visibility so that other accounts cannot see the repository.
Admin	Be able to restore the system from a backup	I can recover data in case its lost or corrupted	Backups can be selected and restored to return the system to a previous state.
Organisation	Have all transferred data be secure	Sensitive information is protected from unauthorised access	Add data transfers are encrypted.

Core Features

Command Line Interface (CLI)

- **Repository Initialisation:** Users can initialize a local repository. Which will create a hidden .janus folder for version control.
- **File Staging:** The CLI allows users to stage files to be ready for the next commit (janus add).
- **Committing Changes:** The user commits their changes to their local repository with a message. This creates a version of the project.
- **Commit History:** The user can view all their previous project versions.
- **Revert to Previous Commit:** Projects can be reverted to previous versions.
- **Merge Conflict Resolution:** In case conflicts occur between changes the related files will be shown and the conflict can be resolved.
- **Pushing changes to remote repository:** Users can push their local repository to the remote repository.
- **Remote access:** The users can fetch their project from the remote repository from any device.

Web Application

- **User Authentication & Authorisation:** Secure login and access control to repository.
- **Repository Dashboard:** The users can view all repositories they have access to.
- **View Commit History:** There is a display to see the commit history for each repository.
- **Download Specific Versions:** Each version of the project can be downloaded from a specific commit.

- **Repository Management:** Repository owners can manage the access of other users and change the repositories visibility.
- **Multi Device Design:** The web app works and displays correctly on many different devices.

Dockerised Infrastructure

- **Dockerised Frontend, Backend & Database:** All parts of the product are split into microservices and run on Docker.
- **Horizontal and Vertical Scalability:** Docker containers allow for the system to scale both horizontally and vertically.
- **Consistent Environments:** Running on Docker ensures that the Janus product can run in different environments.

Backend server

- **API for CLI and Web App:** The backend utilises APIs to allow the CLI, web app and database to communicate with each other.

Database (MySQL)

- **Secure Data Storage:** All data is encrypted to protect sensitive data.
- **Data Backup & Recovery:** Data is backed up regularly to prevent data loss and can be restored easily.
- **Version Control Storage:** The backend manages the storage of data into the MySQL database.

Security

- **Data Encryption:** All data transfers are encrypted to prevent unauthorized access.
- **Access Control:** The repository owners can manage who can access the repository.
- **Security Policies Compliance:** The Janus system can be deployed internally within organisations. This ensures data is kept within their control, following security procedures and reducing the risk of data breaches.

Customisability

- **Plugin System for CLI:** Plugins can easily be added to the CLI to allow organisations to add their own functionality to suit their specific needs.

Technologies & Component Architecture

Command Line Interface (CLI)

Framework: *.NET Core*
Language: *C#*

The CLI will be built using **.NET Core** and written in **C#**.

I chose this as .NET Core is an excellent framework to use for cross platform development, working well on Windows, Linux and MacOS. C# was chosen as it is high performance language with simple I/O file operations, which is necessary for the CLI version control.

Backend

Framework: *ASP.NET Core*
Language: *C#*
Authentication: *JSON Web Tokens (JWT)*
Database Interaction: *Entity Framework Core (EFC)*

The backend will utilise a **ASP.NET Core** framework to handle the communication between CLI, web app and the database using web APIs. **JSON Web Tokens (JWT)** will be used to authenticate and authorise users/actions. While the database management will use **Entity Framework Core (EFC)** to interact with the database. All of these will be written in **C#**.

ASP.NET Core is a scalable, secure and high performing framework making it well suited for the backend APIs. EFC is an object-relational mapper (ORM) that allows the backend to easily interact with the database using C# objects, without having to use SQL queries. This simplifies the interaction between the backend and the database.

Database

Database: *MySQL*
Object-Relational Mapper (ORM): *Entity Framework Core (EFC)*

MySQL will be used for the database and mentioned previously, **Entity Framework Core** will be used so that the backend can easily interact with the database.

MySQL is a scalable and well performing relational database which interacts well with the .NET backend. Additionally, MySQL supports transactions for commits, pushes and rollbacks. Transactions are critical as they ensure that data consistency is maintained.

Web Application

Web Page Layout: *HTML*
Web Page Styling: *CSS*
Dynamic Page Elements: *React (JavaScript)*

The frontend web application will be developed using **HTML**, **CSS** and **React** with **JavaScript**.

React was chosen as it integrates well with the JWT and the backend APIs, while also having a wide selection of libraries and tools available for use. The component-based architecture allows for a modular design that makes development easier and improves scalability.

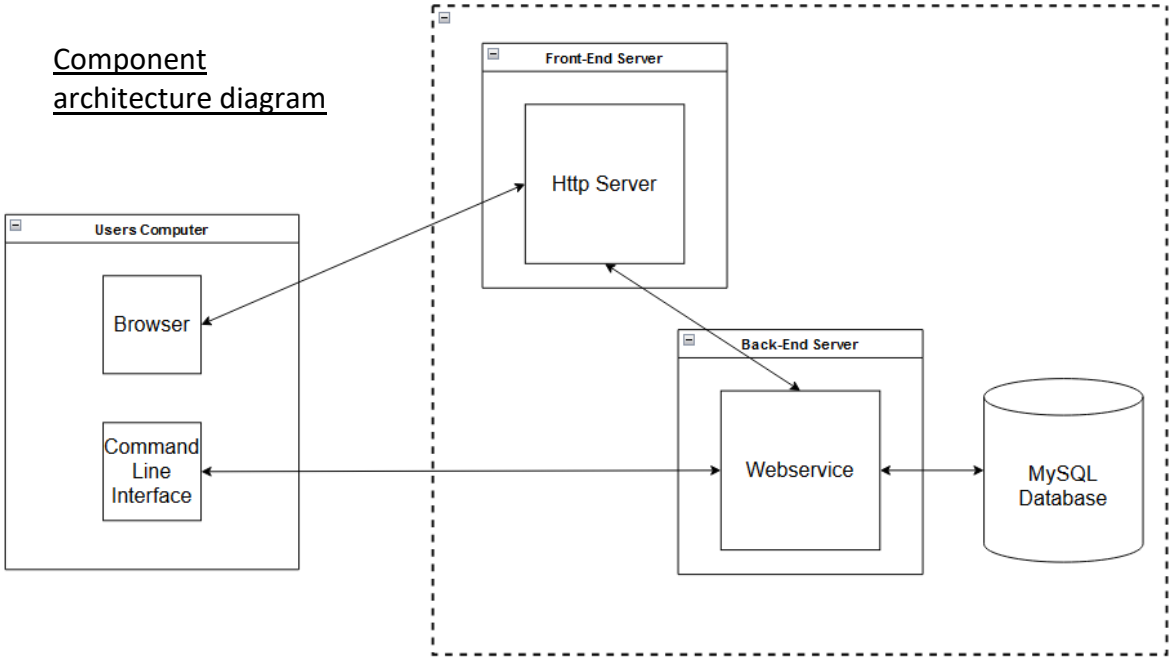
Deployment

System Deployment: *Docker Containers*

The CLI will use the **.NET installer** and to be used has to be installed locally on the user's machine. The web app, backend and database however will be on **Docker** containers.

Docker ensures that the system can run in many environments and enable easy scaling.

Component architecture diagram



Above is a diagram showing how each part of the system interacts with each other. The dashed line representing Docker with each part inside being run in a container.

Risk Management

Risk	Description	Probability (1-10)	Impact (1-10)	Mitigation
Feature Creep	Deviating from the planned features, delaying project development.	4	6	Adhere to my planned sprints. Leave additional features for future sprints, if time allows it. Regularly ensure that development is in line with the core features of the project.
Ease of use	The product is unintuitive or difficult for users to understand and use.	5	7	Utilise user feedback throughout the project to understand what areas might needs improvement. Clearly document how the product should be used, ensuring its readability. Ensure interfaces are intuitive and follow expected behaviours.
Scalability	The product fails to handle increased loads effectively.	4	7	Design the project to account for scalability early on. Perform stress testing with larger loads throughout development.

Performance	Slow processing or a lot of user traffic can lower performance making the product unusable.	5	9	Perform stress testing throughout development to identify bottlenecks in the system.
Security Risks	An unauthorised user gains access to private data.	7	10	Implement strong encryption for data transfers. Use JWT tokens for authentication and authorization. Follow best practices for data storage and transfer.
Data Loss	Loss of version histories or project data, violating data integrity.	5	9	Implement automated database backups. Set up redundancy features. Document a clean data recovery plan.
Licensing Issues	The project violates software licenses.	2	7	Check all third-party libraries and dependencies to ensure compliance with licenses. Document all licenses used.
Legal Issues	Dealing with sensitive data requires compliance with regulations such as GDPR.	3	6	Encrypt all stored and transferred data. Ensure users have the ability to delete their data. Create terms of service and privacy policy.
Legal Liability	Users can store copyrighted or illegal data through our product.	5	9	Create terms of service clearly stating the responsibilities of the user and the Janus system.
Platform Compatibility	The product might not work properly on different platforms.	4	6	User cross-platform frameworks to develop the project. Conduct testing on all supported platforms. Plan and address compatibility issues early on in development.
Concurrency Issues	Users attempt to access or modify data simultaneously, causing conflicts.	7	8	Implement optimistic locking mechanisms to detect and manage conflicts. Use transaction systems to ensure data consistency.
Data Consistency	Internet connection problems, incomplete data or conflicts can cause issues.	5	6	Use transactions to ensure that if any part of the process fails it is rolled back to a consistent state.
Integration Issues	Problems occur while integrating parts of the system. (CLI, web app, database…)	6	9	Plan early on how each part will communicate with each other. Test integration of parts regularly.
Technical Issues	Issues such as bugs or crashes delay development.	7	7	Implement unit, integration and system testing throughout development. During sprint planning, account of unexpected delays which could occur.

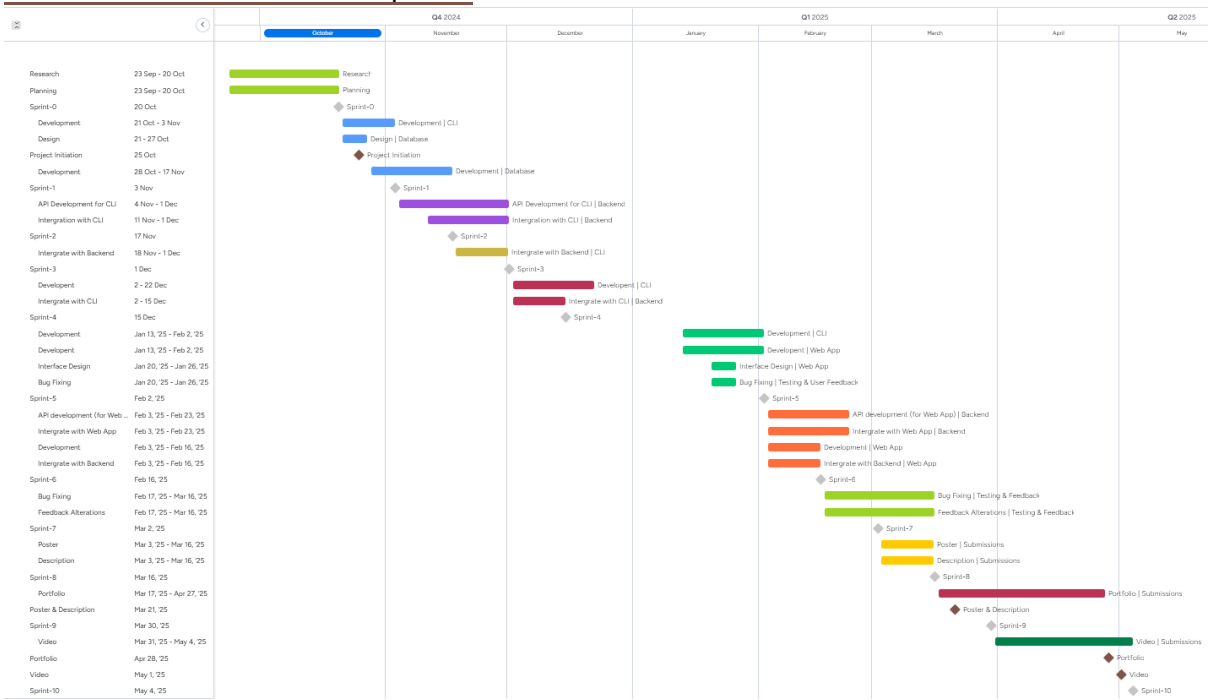
Gantt Chart

The Gantt Chart can be seen in full through the Monday link in the Links section.

Key:

Grey Diamond: End of sprint

Brown Diamond: Submission points



Keywords

- Version Control
- Command Line Interface
- Web App
- Database
- Project Management
- Commit History
- Backup
- Integrity
- Cloud
- Cross Platform
- Version History
- Docker
- Scalable