University of Plymouth

School of Engineering,

Computing, and Mathematics

COMP3000

Computing Project

2024/2025

Distributed Version Control System

Benjamin Sanders-Wyatt

10808929

BSc (Hons) Computer Science

Acknowledgements

Supervisors

Peers

Testers

Supporters

Abstract

(250-300 ~3%)

Stand alone summary

Background, aim, key objectives, methodology, major results, conclusions

Contents

[1 Introduction 4](#_Toc195313202)

[1.1 Project Context and Motivation 4](#_Toc195313203)

[1.2 Problem Statement 4](#_Toc195313204)

[1.3 Objectives 4](#_Toc195313205)

[1.4 Structure of the Report 4](#_Toc195313206)

[2 Background 4](#_Toc195313207)

[2.1 Enterprise needs 4](#_Toc195313208)

[2.2 Evolution of Version Control Systems 4](#_Toc195313209)

[2.3 Gaps 4](#_Toc195313210)

[3 Methodology 4](#_Toc195313211)

[3.1 Project Management 4](#_Toc195313212)

[3.1.1 Agile 4](#_Toc195313213)

[3.1.2 Tools 4](#_Toc195313214)

[3.2 Tools & Technologies 5](#_Toc195313215)

[4 Methodology & Implementation 5](#_Toc195313240)

[4.1 Software Development Lifecycle 5](#_Toc195313242)

[4.2 Tools & Technologies 5](#_Toc195313243)

[4.3 System Architecture & Design 5](#_Toc195313244)

[5 Legal, Social, Ethical, & Professional (LSEP) Issues 5](#_Toc195313245)

[5.1 Legal Considerations 5](#_Toc195313246)

[5.1.1 Data Protection & Privacy Compliance 5](#_Toc195313247)

[5.1.2 Secure Authentication & Account Management 5](#_Toc195313248)

[5.1.3 Licensing & Intellectual Property 6](#_Toc195313249)

[5.1.4 Audit & Accountability 6](#_Toc195313250)

[5.2 Social Considerations 6](#_Toc195313251)

[5.2.1 User Trust & Data Sovereignty 6](#_Toc195313252)

[5.2.2 Accessibility & Transparency 6](#_Toc195313253)

[5.3 Ethical Considerations 7](#_Toc195313254)

[5.3.1 Responsible Data Handling 7](#_Toc195313255)

[5.3.2 Automated systems 7](#_Toc195313256)

[5.3.3 Transparency in Operations 7](#_Toc195313257)

[5.4 Professional Considerations 7](#_Toc195313258)

[5.4.1 Adherence to Industry Standards 7](#_Toc195313259)

[5.4.2 Quality Assurance & Continuous Improvement 7](#_Toc195313260)

[5.4.3 Documentation 8](#_Toc195313261)

[5.4.4 Risk Management & Incident Response 8](#_Toc195313262)

[6 Project Management 8](#_Toc195313263)

[6.1 Planning & Scheduling 8](#_Toc195313264)

[6.2 Risk Management 8](#_Toc195313265)

[6.3 Tools & Techniques 8](#_Toc195313266)

[7 End-Project Report 8](#_Toc195313267)

[7.1 Evaluation of Objectives 8](#_Toc195313268)

[8 References 10](#_Toc195313269)

[9 Appendices 12](#_Toc195313270)

[9.1 User Guide 12](#_Toc195313271)

[9.1.1 Instillation Instructions 12](#_Toc195313272)

[9.1.2 Minimum System Requirements 12](#_Toc195313273)

[9.1.3 Usage Guide 12](#_Toc195313274)

[9.2 Project Management 12](#_Toc195313275)

WORD COUNT: **10000**

https://github.com/benjaminsanderswyatt/COMP3000-JanusVersionControl

# Introduction

## Background

In today’s rapidly evolving technical landscape, version control systems (VCS) play an essential role in collaborative software development. Over the years, a fundamental shift from Centralised Version Control Systems (CVCS) to Distributed Version Control Systems has revolutionised the management of codebases [REF (Overview of the evolution from CVCS to DVCS (e.g. a textbook chapter or review article on version control history))].

Traditional CVCS models rely on a single repository as a source of truth for code management, which introduces challenges such as performance bottlenecks, limited scalability, and a high risk of single points of failure [REF (Study/report highlighting the limitations of centralised systems)]. In contrast, DVCS solutions allow each developer to have a complete local copy of the repository, resulting in greater flexibility, redundancy and resilience.

Recent industry trends have further driven the need for secure, internal repositories. As the importance of data sovereignty and internal compliance rises, enterprises are become increasingly wary of public cloud-based platforms such as Github and GitLab. These platforms although feature rich have inherent risks; industry reports suggest that nearly 70% of data breaches in recent years have been linked to misconfigured cloud services and more than 60% of companies have expressed concerns regarding data sovereignty and regulatory compliance [REF (Industry report detailing data breaches and compliance concerns, e.g. Smith et al., 2023; Industry Security Report, 2024)]

While popular DVCS tools are robust in many aspects, they are often designed for broad general purposes and often lack the granular access control and seamless integration required by modern enterprises. This market gap justifies the need for Janus, a DVCS purpose built for internal enterprise use.

## Problem Statement

Despite the widespread adoption of DVCS platforms, several limitations make them suboptimal for large scale enterprise deployment. Relying on external cloud services exposes organisations to significant security vulnerabilities; their centralised nature makes them prime targets for cyberattacks (see Security Analysis of Cloud-Hosted Solutions, [REF]). Additionally, many existing solutions offer only basic access control functionalities, which falls short of the stringent internal policies and granular permission management organisations demand [REF (Comparative study on access control in existing DVCS platforms)].

Integrating public DVCS platforms into corporate infrastructures can be both complex and resource intensive. Modern enterprises increasingly require systems that support robust local operations through an intuitive Command Line Interface (CLI) together with secure, containerised remote management. There is also a growing need for systems that are easily extensible, so that custom functionality can be seamlessly integrated as requirements evolve. These challenges display the necessity for a dedicated solution such as Janus, tailored specifically for the enterprise environment.

## Project Vision & Objectives

### Project Vision

Janus is a secure and flexible Distributed Version Control System that empowers enterprises to manage their codebases internally. The system is named after the Roman god Janus, which is depicted with two faces, one looking into the future and the other into the past, symbolising the responsibilities a version control system holds.

Janus aims to eliminate dependence on external cloud services and provide organisations with comprehensive control over their intellectual property and development processes [REF (Literature discussing the benefits of internal repository management for enterprise security)].

### Primary Objectives

The core objectives of Janus are as follows:

* **Intuitive, Cross-Platform CLI:** Develop a user-friendly CLI that supports local repository management across all major operating systems (Windows, macOS, and Linux) [REF (User studies or technical reviews on cross-platform CLI usability)].
* **Dockerised Web Application:** Create a secure, Dockerised web interface to facilitate remote access and repository management ensuring data remains within controlled environments [REF (Research on containerisation benefits in enterprise software solutions)].
* **Plugin Architecture:** Establish a flexible plugin framework that allows for future expansion and custom integrations to meet evolving enterprise needs [REF (Case studies on modular software architectures)].
* **LSEP Compliance:** Ensure that the system complies with Legal, Social, Ethical, and Professional (LSEP) standards, addressing data protection, compliance, and internal governance requirements [REF (Overview of LSEP standards in enterprise IT environments)].

### Scope & Limitations

The deliverable of Janus focuses on providing the essential functionalities needed for an enterprise grade DVCS. This includes a fully functional local CLI that supports repository initialisation, file staging, commit logging, branch management, merging and pushing changes to a secure internal remote repository, and a foundations plugin framework will be included to ensure CLI extensibility. Additionally, a Dockerised web application which can be deployed on-premise to offer secure remote access. Robust security measures such as audit logging and secure authentication and integral to this [REF (Technical documentation on essential security measures in DVCS systems)]. Future enhancements may incorporate advanced features, such as third-party tool integration, Kubernetes implementation for improved efficiency, a community framework for sharing plugins, and advanced encryption protocols for data at rest [REF (Roadmaps in DVCS advancements and future trends)].

## Structure of the Report

The remainder of this report is structured to provide a comprehensive understanding of the Janus project. [TODO when the rest of the report is done]

# Context & Literature Review

This section provides an in-depth analysis of both traditional and modern approaches to version control systems. It covers the historical progression of VCSs, a comparison between centralised and distributed systems, and an evaluation of popular DVCS tools, identifying the gaps that the Janus system aims to address.

## Version Control Systems History

Version control systems have evolved considerably over the past several decades. Early systems such as CVS relied on a central repository model which, while straightforward in concept, suffered from significant limitations such as single points of failure and lack of offline access [REF (Historical review of version control systems, e.g., academic texts on CVS and early systems)]. The introduction of DVCS systems, such as Git and Mercurial revolutionised software development by enabling developers to maintain a full copy of the repository, thereby enhancing performance, supporting sophisticated branching, and enabling more robust collaboration [REF (see Historical Review of Version Control Systems, [REF]). (Technical analysis of DVCS improvements over CVCS)].

## Comparison Between Centralised & Distributed Systems

Centralised systems have a singles souse of truth; making them vulnerable to outages and performance bottlenecks in large scale collaborative environments [REF (Comparative study on CVCS vulnerabilities)]. In contrast, distributed systems allow each user a complete copy of the local repository, enhancing resilience, supporting offline work, and enabling more flexible development utilising branching. However, DVCS models also introduce challenges in maintaining data integrity, access control and consistency across different environments [REF (Comparative analysis of DVCS challenges and benefits)].

## Popular DVCS Tools

Among the popular DVCS tools; Git is widely recognised for its robust branching and merging capabilities. However, Gits reliance on public cloud hosting services introduces difficulties complying with strict security and compliance within enterprises [REF (Review of Git limitations in enterprise settings)]. Similarly, Mercurial is noted for its simplicity and performance, yet its limited market penetration and public support hinder its suitability for large scale enterprise adoption [REF (Comparative review of Mercurial and Git for enterprises)].

## Gaps in Existing Solutions

Despite their strengths, current DVCS solutions present several critical shortcomings for enterprise environments:

* Dependence on Public Cloud Services: Public hosted solutions are prone to data breaches and potential loss of proprietary information [REF (Industry analysis on DVCS security risks)].
* Insufficient Granularity in Access Control: Many existing systems fail to offer the level of detailed permission management necessary for strict internal security policies [REF (Case study on internal access control failures)].
* Integration Complexity: The operational complexity involved in integrating these tools into corporate infrastructures is often significant, leading to resource-intensive deployments [REF (Technical report on integration challenges in enterprise DVCS deployments)].
* Scalability and Adaptability Challenges: Existing systems struggle to scale effectively within large organisations and to seamlessly integrate with diverse internal systems [REF (Research on scalability challenges in DVCS platforms)].

These gaps illustrate the necessity for Janus, a tailored solution offering secure, internal deployment and enhanced customisability.

## Industry Requirements & Relevance

Modern enterprises face strict regulatory demands regarding data sovereignty, internal compliance, and overall governance. Organisations must maintain full control over their sensitive information, ensuring that data does not leave their internal environment [REF (Overview of enterprise data sovereignty requirements, e.g., GDPR analysis)].

Janus addresses these requirements through an internal, Dockerised deployment, with comprehensive audit logging, and rigorous authentication. Its flexible plugin architecture and scalable design also serve to future proof the systems for evolving enterprise needs [REF (Studies on modular and scalable system architectures in enterprise IT)].

# Project Scope & Deliverable

This section outlines the deliverables of the of the Janus project, detailing the components that comprise the final version, while clearly stating what is out of scope for the project’s development.

## Main Components

The Janus project is structured around several core components:

Dockerised Frontend, Backends & Database:

The project will deploy a containerised environment using Docker including:

* Frontend: A responsive, user-friendly web interface that provides secure web access to repository data and management tools [REF (Literature on benefits of Dockerised web applications)].
* Backend: RESTful APIs designed to support CLI and web interactions, managing authentication, audit logging and data processing [REF (Technical documentation on RESTful API design in secure systems)].
* Database: A robust data storage solution for secure and efficient data retrieval and storage [REF (Case study on database security best practices)].

Local CLI:

The Command Line Interface (CLI) is the core of Janus’s local operations and will support essential functionalities such as:

* Repository initialisations (Creating a hidden directory containing the local repository for version control)
* File staging and commit history loggings
* Branch management, merging and conflict detection
* Pushing changes to a secure, internal remote repository [REF (User manual or case study on effective CLI usage in DVCS environments)].

Plugin Framework:

The CLI will be developed to allow users to develop and integrate custom plugins, enabling custom functionality to tailor the system to their specific needs [REF (Research on plugin architectures in modular software systems)].

Documentation:

Comprehensive guides covering CLI installation, usage and extension will be provided to ensure that both technical and non-technical users can effectively utilise and maintain the system [REF (Best practices guide for user documentation in software projects)].

### Out of Scope

For the delivered release, Janus will focus on core DVCS functionalities and secure internal deployment. Advanced features, such as integration with third-party tools (e.g. CI/CD pipelines), highly customisable UI components, additional collaboration features, and advanced CLI commands (e.g. rebasing) are considered for future enhancements [REF (Roadmap discussion on future DVCS features)].

# Introduction

## Background

In today’s rapidly evolving technological landscape, version control systems (VCS) play and essential role in collaborative software development. Over the years, there has been a fundamental shift from Centralised Version Control Systems (CVCS) to Distributed Version Control Systems; this transition has revolutionised how codebases are managed. While traditional CVCS models rely on a single repository to manage and store code, this centralisation often introduces challenges such as performance bottlenecks, limited scalability, and a high risk of single points of failure. In contrast, DVCS solutions allow each developer to maintain a complete local copy of the repository, resulting in greater flexibility, redundancy and resilience.

Recent industry trends have further driven the need for secure internal repositories; as issues of data sovereignty and internal compliance become more important, enterprises are increasingly wary of public cloud-based platforms such as Github and GitLab. These platforms while feature rich, come with potential risks; according to industry reports, nearly 70% of data breaches in recent years involved misconfigured cloud services and more than 60% of companies have expressed concerns regarding data sovereignty and regulatory compliance [REF Smith et al., 2023; Industry Security Report, 2024].

Whie popular DVCS tools are robust in many respects, they are often designed for general use in mind, lacking the granular access control and seamless integration required by modern enterprises. This market gap justifies the need for Janus, a DVCS purpose built for internal enterprise use.

## Problem Statement

Despite the widespread adoption of DVCS platforms, several limitations make them less than ideal for large scale enterprise deployment. Relying on external cloud services presents significant security vulnerabilities; such services become attractive targets for cyberattacks due to their centralised nature. IN addition, existing solutions typically offer only basic, one-size-fits-all access control, which falls short of the stringent internal policies and granular permission management that many organisations require.

Integrating public DVCS platforms into corporate infrastructures can be both complex and resource intensive. Furthermore, enterprises now demand systems that support robust local operations through an intuitive Command Line Interface (CLI) coupled with secure, containerised remote management. There is also a growing need for platforms that are easily extensible, allowing custom features to be integrated as business requirements evolve. These challenges display the pressing need for a dedicated solution such as Janus, which is tailored specifically to address these concerns.

## Project Vision & Objectives

### Vision Statement

Janus is envisioned as a secure and flexible Distributed Version Control System that empowers enterprises to manage their codebases internally. By ensuring rigorous compliance, robust security and usability, Janus aims to eliminate dependence on external cloud services and provide organisations with comprehensive control over their intellectual property and development processes.

### Primary Objectives

* Intuitive, Cross-Platform CLI: Develop a user-friendly CLI that supports local repository management across all major operating systems, Windows MacOf and Linux.
* Dockerised Web Application: Create a secure, Dockerised web interface to facilitate remote access and repository management.
* Plugin Architecture: Establish a flexible plugin framework, allowing future expansion and custom integrations to meet evolving enterprise needs.
* LSEP Compliance: Ensure that the system complies with Legal, Social, Ethical and Professional (LSEP) standards, addressing data protection, compliance and internal governance requirements.

The core objectives of the Janus project are:

The development of an intuitive cross-platform CLI which facilitates local repository management across Windows, macOS and Linus environments.

A secure Dockerised web application, to enable remote repository management, ensuring that corporate data remains internal and is accessed only through controlled channels.

Janus will feature a flexible plugin architecture that supports custom integrations by user developed plugins to get custom features.

The system will be designed to comply with Legal Social Ethical and Professional standards, addressing key aspects of data protection, regulatory compliance and internal governance.

### Scope & Limitations

The deliverable of Janus focuses on providing the essential functionalities needed for an enterprise grade DVCS. This includes a fully functional CLI that supports repository initialisation, file staging, commits, branching and pushing changes to a secure internal remote repository. A Dockerised web application will also be deployed to offer secure remote access, and a foundations plugin framework will be included to ensure extensibility. Robust security measures such as audit logging and secure authentication and integral to this realises. Future enhancements may incorporate advanced features, such as third-party tool integration, the implementation of Kubernetes for improved efficiency, a community framework for sharing plugins, and advanced encryption protocols for data at rest.

## Structure of the Report

The remainder of this report is structured to provide a comprehensive understanding of the Janus project. Section 2 presents a detailed context and literature review that examines traditional and modern approaches to version control, including the evolution from CVCS to DVCS and a critical comparison of existing systems. In Section 3, the project scope and deliverables are discussed in detail, outlining the core components to be delivered in the initial release as well as identifying areas that are out of scope and potential future enhancements. [TODO when the rest of the report is done]

# Context & Literature Review

This section provides an examination of both traditional and modern version control systems, supporting the need for the Janus system. It delves into historical developments, compares centralised and distributed systems, and reviews popular DVCS tools, before identifying the specific gaps that Janus aims to fill.

## Version Control Systems History

Version control systems have evolved significantly over the past few decades. Early systems such as CVS relied on a central repository model, which, while simple, had limitations such as single points of failure and lack of offline access. With the development of DVCS, exemplified by systems like Git and Mercurial, developers gained the ability to maintain local repository compiles. This transition addressed many of the limitations of CVCS by providing improved performance, branching capabilities and enhanced collaboration.

(Isn’t this section covered by the background?)

The history of version control is marked by events of technological advancements. Early systems such as CVS operated on a central repository model that, while straightforward suffers from limitations including single points of failure and restricted offline accessibility. The rise of distributed version control systems, such as Git and Mercurial, represents a shift in how codebases are being managed. DVCS allowed developers to maintain complete copies of repositories on their local machines, significantly improving performance, enabling branching and merging strategies for feature development, and providing increased redundancy.

## Comparison Between Centralised & Distributed Systems

Centralised systems have a single source of truth; making them vulnerable to outages and susceptible to performance issues when managing large teams or complex repositories. In contrast, distributed systems provide each user with a complete local copy of the repository. This decentralisation offers increased resilience, offline work capability and more flexible branching methodologies. Despite these advantages. DVCS models introduce challenges with data integrity and consistency across different environments.

## Popular DVCS Tools

Among the most popular DVCS solutions; Git stands out for its robust branching and merging capabilities. However, Gits dependence on public cloud hosting services poses challenges for enterprises with strict security and compliance requirements. Similar, Mercurial is recognised for its simplicity and performance, but its relatively limited market penetration and public support hinder its suitability for large scale enterprise adoption. While both tools offer a solid foundation for general purpose version control, their design paradigms do not fully cater to the needs of organisations that demand secure data sovereignty.

## Gaps in Existing Solutions

Despite theirstrengths, current DVCS solutions exhibit several critical shortcomings for enterprise enviroments:

Public cloud hosted solutions are vulnerable to data braches and loss of proprietary information

Furthermore these systems often lack the necessary granularity in access control; in an enterprise context, these is a need to enforce highly specific permission structures to safeguard sensitive data.

The operational complexity of intergrating these tools into existing corporate infrastrucurres further complicates their usage

Additionaly limitations include challenges in scaling effectively within large oragnisations and intergrating with internal systems.

These gaps display the necessity for a system like Janus, which is designed for internal deployment, provides enhanced security and is easily customisable.

## Industry Requirements & Relevance

Modern enterprises face stringent demands in terms of data sovereignty, internal compliance and regulatory compliance. Organisations must maintain strict control over their data, ensuring that sensitive information remains protected withing internal systems. Regulatory frameworks such as GDPR impose strict standards on data handling, necessitating advanced security measures and robust auditing capabilities. Janus is designed to meet these challenges through an internal, Dockerised deployment model, secure audit logging, and rigorous authentication protocols. Moreover, its flexible plugin architecture and scalable design ensure that the system can adapt to evolving enterprise needs, thereby future-proofing the investment.

# Project Scope & Deliverable

This section outlines the tangible outputs of the Janus project, detailing the components that constitute the initial release and noting the boundaries for future enhancement.

## Main Components

Dockerised Frontend, Backends & Database:

The project will deploy a complete containerised environment using Docker. This environment will include:

* Frontend: A responsive, user-friendly web interface providing secure repository data and management access.
* Backend: RESTful APIs built to support CLI and web interactions, managing authentication, audit logging and data processing.
* Database: A robust data storage solution for secure and efficient data storage and retrieval.

Local CLI:

The Command Line Interface (CLI) is the core of Janus’s local operations. Key functionalities include:

* Repository initialisations (Creating a hidden directory for version control)
* File staging and commit history loggings
* Branch management, merging and conflict resolution
* Pushing changes to a secure, internal remote repository

Plugin Framework:

An extensible framework that allows the addition of custom functionalities, Users can develop and integrate custom plugins to tailor the DVCS system to meet specific organisational needs.

Documentation:

Comprehensive guides for the CLI installation, usage and extension will be provided. This documentation will ensure that both technical and non-technical users can effectively utilise and maintain the system.

### Out of Scope

For the delivered release, Janus will focus on core DVCS functionalities and secure internal deployment. Advanced features include integration with third-party tools (e.g. CI/CD pipelines), highly customisable UI components, additional collaboration features, and advanced CLI commands (e.g. rebasing). Are considered for future enhancements.

# Introduction

## Background

In todays rapidly evolving technological landscape, version control systems (VCS) play an essential role in collaborative software development. Over the years, the evolution from Centralised Version control Systems (CVCS) to Distributed Version Control Systems has significantly enhances the way codebases are managed and maintained. Traditional CVCS models provided a single point of management; however, this centralisation often prosed issues related to performance, scalability and single points of failure. In contrast DVCS solutions enable multiple copies of repositories on local machines, offering greater flexibility and resilience.

Recent trends have increased the demand for secure, internal code repository’s especially within the corporate environments. The need for data sovereignty, strict internal compliance and robust security measures has never been more critical. With public cloud-based platforms, such as GitHub, GitLab and others, increasingly susceptible to external security breaches, enterprises are now reconsidering their dependence on these platforms. For instance, industry reports indicate that nearly 70% of data breaches in recent years involved improper configuration of cloud services, and over 60% of company’s haves expressed concerns regarding data sovereignty and regulatory compliance [REF Smith et al., 2023; industry Security Repot, 2024].

Furthermore, while widely used DVCS solutions like Git and GitLab offer a comprehensive range of features, they often fall short in catering to specific enterprise requirements, particularly with regard to granular access control and seamless integration into corporate infrastructures. These platforms are generally built for general use rather than addressing the stringent security, regulatory and integration needs of modern organisations. In light of these industry trends and statistics, these is a clear market demand for an internally deployable, secure and flexible DVCS. This necessity forms the basis for developing Janus to address these specific enterprise challenges.

## Problem Statement

Despite the widespread adoption of DVCS platforms, several limitations render them less than ideal for enterprise use:

* Security Vulnerabilities: Relying on external cloud services exposes organisations to potential data breaches. The centralised nature of many public platforms makes them attractive targets for cyberattacks.
* Access Control Issues: Many current solutions offer only basic one-size-fits-all access control, lacking the granular permissions that enterprises require to enforce strict internal policies
* Integration Complexity: Adapting public DVCS systems into established corporate infrastructures can be complex and resource intensive. The mismatch between public could functionalities and corporate requirements often leads to operational inefficiencies.
* Dual Operational Needs: There is an emerging need for systems that support both robust local operations, managed via an intuitive Command Line Interface (CLI), and secure, containerised remote management. Additionally, enterprises require a platform that is easily extensible via plugins to integrate custom features.

These challenges highlight the gap that exists in the current market and underscores the necessity for a system like Janus, which is tailored specifically for internal enterprise use.

## Project Vision & Objectives

### Vision Statement

Janus is envisioned as a secure and flexible Distributed Version Control System that empowers enterprises to manage their codebases internally. By ensuring rigorous compliance, robust security and ease of use, Janus eliminated dependence on external cloud services and provides the control that modern organisations require.

### Primary Objectives

* Intuitive, Cross-Platform CLI: Develop a user-friendly CLI that supports local repository management across all major operating systems, Windows MacOf and Linux.
* Dockerised Web Application: Create a secure, Dockerised web interface to facilitate remote access and repository management.
* Plugin Architecture: Establish a flexible plugin framework, allowing future expansion and custom integrations to meet evolving enterprise needs.
* LSEP Compliance: Ensure that the system complies with Legal, Social, Ethical and Professional (LSEP) standards, addressing data protection, compliance and internal governance requirements.

### Scope & Limitations

The delivered version of Janus will focus on:

* Delivering a fully functional local CLI for repository operations (initialisation, file staging, commits, branch management, merging and pushing).
* Implementing a basic Dockerised web application that enables secure remote repository management.
* Providing a foundational plugin framework to support extensibility
* Ensuring robust security measures, including audit logging and secure authentication.

Future enhancements may include advanced collaboration features, integration with additional enterprise tools, such as CI/CD pipelines, and deeper customisation options to accommodate more complex organisational workflows. Better efficiency (kuberneties), a community for sharing janus plugins, advanced encryption??

## Structure of the Report?

This report is organised to provide a comprehensive overview of the Janus project:

* Section 1 – Introduction: Presents the background, problem statement, and the project vision with its core objectives and limitations.
* Section 2 – Context and Literature Review: Offers a detailed exploration of traditional and modern approaches to version control, compares existing systems, and discusses identified gaps alongside industry requirements.
* Section 3 – Project Scope & Deliverables: Outlines the main components of Janus, details the deliverables, and clarifies what is in and out of scope for the initial release.

# Context & Literature Review

This section delved into both traditional and modern approaches to version control systems, providing critical analysis and technical context that supports the need for Janus.

## Version Control Systems History

Version control systems have evolved significantly over the past few decades. Early systems such as CVS relied on a central repository model, which while simple had limitations such as single points of failure and lack of offline access. With the development of DVCS, exemplified by systems like Git and Mercurial, developers gained the ability to maintain local repository compiles. This transition addressed many of the limitations of CVCS by providing improved performance, branching capabilities and enhanced collaboration.

(Isn’t this section covered by the background?)

## Comparison Between Centralised & Distributed Systems

### Centralised systems

Centralised VCS offer simplified management and enforce a single source of truth; however, they are vulnerable to service outages and can be slow when handling large teams or complex projects.

### Distributed Systems

DVCS enable decentralised workflows, where each developer has a complete copy of the repository. This model allows for greater redundancy, offline work and flexible branching, but can introduce complexity managing repository integrity and enforcing access controls withing large organisations.

(This comparison should go into more detail and be improved)

## Popular DVCS Tools

Among the popular DVCS solutions:

Git: Highly popular and widely adopted, Git provides robust branching and merging capabilities, however its steep learning curve and reliance on public cloud platforms for hosting can be challenging for enterprises with strict security and compliance requirements.

Mercurial: Known for its simplicity and performance, Mercurial offers a similar distributed model. Yet its market penetration is lower and it may not offer the same extensive community and support resources as Git.

Both systems are designed primarily for general use, often overlooking the specific needs of enterprises that demand internal security and control.

(This should cover more systems in detail)

## Gaps in Existing Solutions

Despite their many strengths, current DVCS solutions pose several challenges for enterprises:

* Dependence on Cloud Services: Public DVCS platforms introduce risks related to data breaches and loss of control over sensitive intellectual property.
* Complexity for non-experts: While powerful, existing tools often require advanced technical knowledge, making them less accessible to users without specialised training. (Janus isn’t much less complex than git)
* Limited Internal Security Controls: Many DVCS do not offer the granular access control or compliance features required by modern enterprises.
* Scalability and integration issues: The integration of public DVCS with existing corporate systems can be complex particularly when scaling the solution to meet large organisational demands.

The rationale for Janus is to address these gaps by providing an internally deployable system with a secure, Dockerised environment and a flexible plugin system that caters specifically to enterprise needs.

(Come up with some more gaps)

## Industry Requirements & Relevance

Enterprises today face stringent regulatory and security requirements:

* Data Sovereignty and Internal Compliance: Organisations must ensure that their data remains within controlled environments. Regulatory frameworks such as GDPR and internal policies demand robust data management and security practices.
* Enhanced Security and Audit Capabilities: Janus will incorporate comprehensive audit logging, secure authentication (via JWT tokens), and role-based access control to meet these demands.
* Future Proofing and Scalability: As corporate needs evolve; the system must be scalable and customisable. Januss Dockerised deployment and plugin architecture are designed to enable horizontal and vertical scaling and to facilitate future enhancements.

These industry trends underline the necessity for a tailored DVCS solutions like Janus that aligns with the regulatory and operational demands of modern enterprises.

# Project Scope & Deliverable

This section outlives the tangible outputs of the Janus project, detailing the components that constitute the initial release as well as noting the boundaries for future enhancement.

## Main Components

Dockerised Frontend, Backends & Database:

The project will deploy a complete containerised environment using Docker. This environment will include:

* Frontend: A responsive, user-friendly web interface that provides secure access to repository data and management.
* Backend: RESTful APIs built to support CLI and web interactions, managing authentication, audit logging and data processing.
* Database: A robust data storage solution, designed for secure and efficient data storage and retrieval.

Local CLI:

The Command Line Interface (CLI) is the core of Januss local operations. Key functionalities include:

* Repository initialisations (Creating a hidden directory for version control)
* File staging and commit history loggings
* Branch management, merging and conflict resolution
* Pushing changes to a secure, internal remote repository

Plugin Framework:

An extensible framework that allows the addition of custom functionalities. Users can develop and integrate custom plugins to tailor the DVCS system to meet specific organisational needs.

Documentation:

Comprehensive guides for installation, usage and extension of the CLI will be provided. This documentation will ensure that both technical and non-technical users can effectively utilise and maintain the system.

### Out of Scope

For the delivered release, Janus will focus on core DVCS functionalities and secure internal deployment. Advanced features, such as integration with third party tools (e.g. CI/CD pipelines), highly customisable UI components, additional collaboration features, advanced CLI command (e.g. rebasing). Are considered for future enhancements.

# Introduction

## Project Context and Motivation

## Problem Statement

## Objectives

Functional and non functional requirements

## Structure of the Report

This report will

# Background

## Enterprise needs

## Evolution of Version Control Systems

Centralised systems

Distrubuted

## Gaps

# Methodology

## Project Management

### Agile

### Tools

## Tools & Technologies

# Methodology & Implementation

## Software Development Lifecycle

## Tools & Technologies

## System Architecture & Design

# Legal, Social, Ethical, & Professional (LSEP) Issues

Significant attention was given to ensuring that legal, social, ethical, and professional issues were addressed both in and before the development of Janus. This section evaluates the measures to manage these concerns and discusses their implications for system design and accountability.

## Legal Considerations

### Data Protection & Privacy Compliance

Janus is designed to operate entirely on-premise, ensuring that sensitive code and personal data remains within the organisation’s control. By leveraging Docker to deploy the system within a controlled subnet, the system minimises risks associated with external data exposure; this approach supports compliance with data protection regulations such as GDPR (European Union, 2016) and the Data Protection Act (UK Government, 2018).

Additionally, enforcing HTTPS for all data transfers provides an essential layer of encryption that safeguards data in transit and mitigates the risk of interception (OWASP, 2025).

### Secure Authentication & Account Management

Robust security is implemented using JSON Web Tokens (JWT) for API authentication, enabling secure communications between system components; this method is widely recognised for its efficiency in distributed environments (Jones, et al., 2015). The system utilises Personal Access Tokens (PAT) that are revocable and have configurable expiry times, reducing the reliance on static passwords and enhancing session security (National Instuture of Standards and Technology, 2017).

Furthermore, passwords are salted and hashed using industry-standard cryptographic practices (600,000 iterations of PBKDF2 with SHA256), ensuring resilience against brute force and dictionary attacks (OWASP, 2025). The use of 128-bit salt is balanced between collision risk and performance (NIST, 2017). These measures ensure that stored credentials are robust and that data integrity is maintained through transactional data interactions (Oracle, 2025).

### Licensing & Intellectual Property

Janus integrates various third-party libraries and frameworks, all of which have been reviewed for licensing compatibility; this minimises legal risks related to open-source or proprietary components (OSI, 2024).

In addition, Janus has plugin functionality that allows users to develop custom commands; this functionality is designed for user customisation of the system. As a result, the intellectual property of the custom-developed plugins will be held by the user or organisation (Svitla, 2024).

### Audit & Accountability

An essential component of Janus is its comprehensive audit logging, which records all database interactions to create a solid audit trail. This supports internal audits and serves as legal evidence in cases of data breaches or non-compliance; transparency in system operations is maintained because both the old and new states of data are logged, ensuring accountability and regulatory compliance (Souppaya & Kent, 2006).

## Social Considerations

### User Trust & Data Sovereignty

A key social advantage of Janus is the enhanced control organisations have over their codebases; by eliminating the need for external cloud services, user trust is improved as all information is managed internally. This approach not only reinforces data sovereignty but ensures that users know exactly who handles their information (Scherenberg, et al., 2024).

### Accessibility & Transparency

Janus has been designed with the user experience in mind. The React-based web interface adheres to most modern accessibility standards, such as the WCAG guidelines (WCAG, 2024), ensuring that users from any background can effectively navigate and utilise the system.

Clear documentation is provided, including detailed usage instructions for the CLI. While features like light/dark themes support usability by accommodating user preferences and reducing eye strain (Kristallovich & Eisfeld, 2020).

## Ethical Considerations

### Responsible Data Handling

Ethically, Janus prioritises the responsible management of user data. Users must accept the Terms of Use and Privacy Policy before creating an account, ensuring informed consent regarding data handling.

Sensitive data remains confined within the organisation, minimising the risk of unauthorised exposure; this approach not only protects individual privacy rights but also upholds ethical standards in data management (Chang, et al., 2016).

### Automated systems

Janus deliberately avoids automated resolutions, such as automated merge conflict handling, ensuring that users fully hold control over critical actions: this places accountability with the users and reduces the risk of compromising data integrity.

### Transparency in Operations

The detailed audit logging mechanism, combined with explicit Terms of Use and Privacy Policy, ensures that users are well informed about data collection and processing practices; the transparency is essential for ethical accountability and enabling users to make informed decisions about their data (ICO, 2025).

## Professional Considerations

### Adherence to Industry Standards

From a professional standpoint, Janus adheres to established industry best practices in software development. The implementation of design principles such as DRY (Don’t Repeat Yourself) (Thomas & Hunt, 2000) and the use of modular, reusable code components contribute to maintainability and scalability (Parnas, 1972). The selection of industry-standard frameworks, such as .NET Core and React, ensures that the system is robust and that professional standards are upheld (Anjum & Alam, 2019).

### Quality Assurance & Continuous Improvement

Professional responsibility is demonstrated through rigorous testing and continuous integration/continuous deployment (CI/CD) pipelines. Regular unit, integration, system and usability tests ensure that Janus maintains high standards of quality and reliability {REF Software testing and quality assurance guidelines (explains testing methodologies)}. Moreover, the agile development methodology and sprint planning ensure that professional standards are maintained throughout the development lifecycle (Dybå & Dingsøyr, 2008).

### Documentation

Comprehensive documentation is essential for maintaining professionalism. Janus provides detailed documentation for the CLI, along with user guides and technical documents {REF appendix}, ensuring that users and developers can understand and effectively utilise the system {REF Documentation standard in software development (provides guidelines for effective documentation)}. This clear documentation demonstrates the project’s commitment to professional clarity and accountability.

### Risk Management & Incident Response

Finally, Janus incorporates a robust risk mitigation strategy that addresses potential issues ranging from feature creep to security vulnerabilities. The use of revocable PATs, audit logging and continuous testing reflects the approach to managing professional and ethical risks; this approach safeguards the system and aligns with the professional duty to anticipate and mitigate potential threats {REF Risk management frameworks in IT (outlines best practices in risk mitigation)}.

# Project Management

## Planning & Scheduling

## Risk Management

## Tools & Techniques

# End-Project Report

## Evaluation of Objectives

# References

Anjum, N. & Alam, S., 2019. A Comparative Analysis on Widely used Web Frameworks to Choose the Requirement based Development Technology. *IARJSET,* pp. 16-24.

Chang, H., Zankl, W. & Lee, W. W., 2016. [Online]   
Available at: https://www.isaca.org/resources/isaca-journal/issues/2016/volume-6/an-ethical-approach-to-data-privacy-protection  
[Accessed 7 April 2025].

Dybå, T. & Dingsøyr, T., 2008. *Empirical studies of agile software development: A systematic review.* [Online]   
Available at: https://doi.org/10.1016/j.infsof.2008.01.006  
[Accessed 8 April 2025].

European Union, 2016. *GDPR.* [Online]   
Available at: https://www.legislation.gov.uk/eur/2016/679/contents  
[Accessed 7 April 2025].

ICO, 2025. *Transparency.* [Online]   
Available at: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/accountability-and-governance/accountability-framework/transparency  
[Accessed 8 April 2025].

Jones, M. B., Bradley, J. & Sakimura, N., 2015. *RFC 7519.* [Online]   
Available at: https://datatracker.ietf.org/doc/html/rfc7519  
[Accessed 7 April 2025].

Kristallovich, F. & Eisfeld, H., 2020. [Online]   
Available at: https://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-50563  
[Accessed 7 April 2025].

National Instuture of Standards and Technology, 2017. *NIST Special Publication 800-63B.* [Online]   
Available at: https://pages.nist.gov/800-63-3/sp800-63b.html  
[Accessed 7 April 2025].

NIST, 2017. *Digital Identity Guidelines.* [Online]   
Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf  
[Accessed 7 April 2025].

Oracle, 2025. *Transactions.* [Online]   
Available at: https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/transactions.html  
[Accessed 7 April 2025].

OSI, 2024. *The License Review process.* [Online]   
Available at: https://opensource.org/licenses/review-process  
[Accessed 7 April 2025].

OWASP, 2025. *OWASP Cheet Sheet Series.* [Online]   
Available at: https://cheatsheetseries.owasp.org/cheatsheets/Transport\_Layer\_Security\_Cheat\_Sheet.html  
[Accessed 7 April 2025].

OWASP, 2025. *Password Storage Cheat Sheet.* [Online]   
Available at: https://cheatsheetseries.owasp.org/cheatsheets/Password\_Storage\_Cheat\_Sheet.html  
[Accessed 7 April 2025].

Parnas, D. L., 1972. *On the criteria to be used in decomposing systems into modules,* New York: Association for Computing Machinery.

Scherenberg, F. v., Hellmeier, M. & Otto, B., 2024. *Data Sovereignty in Information Systems.* [Online]   
Available at: https://doi.org/10.1007/s12525-024-00693-4  
[Accessed 8 April 2025].

Souppaya, M. & Kent, K., 2006. [Online]   
Available at: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf  
[Accessed 7 April 2025].

Svitla, 2024. *Intellectual Property in Software Development.* [Online]   
Available at: https://svitla.com/blog/intellectual-property-software/  
[Accessed 8 April 2025].

Thomas, D. & Hunt, A., 2000. In: *The pragmatic programmer: from journeyman to master.* Boston: Addison-Wesley Longman Publishing Co., Inc..

UK Government, 2018. *Data Protection Act.* [Online]   
Available at: https://www.legislation.gov.uk/ukpga/2018/12/contents  
[Accessed 7 April 2025].

WCAG, 2024. *Web Content Accessibility Guidelines.* [Online]   
Available at: https://www.w3.org/TR/WCAG21/  
[Accessed 7 April 2025].

# Appendices

System Architecture Diagrams

Gantt Chart

Sprint documentation

Detailed Risk Management table

User guide (Installation, minimum system requirements, usage instructions)

## User Guide

### Instillation Instructions

### Minimum System Requirements

### Usage Guide

## Project Management

Monday Board (sprints, Kanban, user stories)

Requirements, plan and sprint reviews