

**Lahore Garrison University**  
**Department of Software Engineering**

**SCD Project Report**

**GPAGenie using Python**

|  |  |
| --- | --- |
| **Submitted by:** | Azka 190 |
|  | Laiba 078 |
| **Submitted to:** | Ali Haider |
| **Course:** | SCD |
| **Date of Submission:** | 18th June, 2025 |

GPAGenie: Software Construction and Development Report

**Abstract**: This report presents the development of GPAGenie, a desktop application designed to assist students in calculating academic performance metrics (SGPA, CGPA, percentage) and managing academic blog content. The project is structured according to the SE311 – Software Construction and Development course requirements. It documents the software design and development process, including process model selection, version control, testing, refactoring, deployment, and quality assurance activities. The user interface is designed with clarity and consistency in mind, applying standard UI/UX principles to ensure an intuitive experience for both regular and administrative users . The database schema is normalized, with tables for users, semesters, courses, and blog posts/comments, which are related via primary/foreign keys to maintain data integrity. We adopted an iterative Agile process model to allow incremental development and continuous feedback . Key project management aspects such as software process improvement (SPI), Lehman’s laws of evolution, risk management, and team roles are addressed. Testing strategies include thorough unit tests with the *pytest* framework and automated CI pipelines using GitHub Actions. Exception handling is implemented to catch user input errors and system faults gracefully. Throughout development, peer reviews and code inspections were used to detect defects early and share knowledge, reflecting software quality best practices . The report concludes with a discussion of how the project fulfills course learning outcomes and recommendations for future improvements.

Table of Contents

[1. Introduction 4](#_Toc200916360)

[2. UI/UX Design Strategy 4](#_Toc200916361)

[Key UI/UX Principles Applied: 4](#_Toc200916362)

[3. Database Design and ER Diagram 5](#_Toc200916363)

[4. Process Model Implementation 7](#_Toc200916364)

[Key Justifications for Iterative Agile 7](#_Toc200916365)

[5. Software Process Improvement (SPI) 7](#_Toc200916366)

[6. Version Control and GitHub Integration 7](#_Toc200916367)

[7. Lehman’s Laws of Software Evolution 8](#_Toc200916368)

[Key Principles Applied: 8](#_Toc200916369)

[8. Software Deployment Management 8](#_Toc200916370)

[Deployment Steps: 9](#_Toc200916371)

[9. Legacy Code Refactoring 9](#_Toc200916372)

[Key Refactoring Strategies: 9](#_Toc200916373)

[10. Unit Testing Implementation 10](#_Toc200916374)

[Unit Testing Strategy: 10](#_Toc200916375)

[11. Automated Testing Strategy 10](#_Toc200916376)

[12. Exception Handling Implementation 11](#_Toc200916377)

[Key Exception Handling Strategies: 11](#_Toc200916378)

[13. Peer Review Process 11](#_Toc200916379)

[Process and Benefits 11](#_Toc200916380)

[14. Project Phases 12](#_Toc200916381)

[15. Risk Management and Mitigation 12](#_Toc200916382)

[16. Learning Outcomes Achievement 13](#_Toc200916383)

[17. Conclusion and Recommendations 13](#_Toc200916384)

[18. Future Recommendations: 14](#_Toc200916385)

[19. References 14](#_Toc200916386)

[20. Appendices 14](#_Toc200916387)

[Appendix A: Application Architecture Diagram 14](#_Toc200916388)

[Appendix B: 16](#_Toc200916389)

[Appendix C: GitHub Actions CI/CD Workflow (YAML Example) 16](#_Toc200916390)

[Appendix D: Sample Code Snippets 17](#_Toc200916391)

# Introduction

The **GPAGenie** application is a comprehensive desktop tool developed as a core component of the SE311 – Software Construction and Development curriculum at Lahore Garrison University. Its primary objective is to equip students with efficient tools for calculating academic performance metrics, including SGPA, CGPA, and percentage conversions. Additionally, the system features an integrated educational blog management system.

GPAGenie accommodates two distinct user roles: a **Regular user** (student), primarily for academic calculations and blog consumption, and an **Administrator**, endowed with elevated permissions for blog content management. The application's interface is designed for clarity and consistency, encompassing a login/signup screen, a dashboard with dedicated tabs for various calculators (SGPA, CGPA, Percentage, Grade Converter), and a Blog section, complemented by an About page for usage instructions.

Technically, GPAGenie is implemented in **Python** utilizing a GUI toolkit, structured around an **MVC-like architecture** to ensure a clear separation between views, models, and business logic (services). Data is persistently stored in a **MySQL relational database**. The database schema is designed with normalized tables—including users, semesters, courses, blog\_posts, and blog\_comments—enforcing referential integrity via primary and foreign keys to uphold data consistency and mitigate redundancy.

Throughout this project, fundamental software engineering principles taught in SE311 were meticulously applied. These encompassed critical aspects such as process model selection, robust configuration management, adherence to software evolution laws, comprehensive testing strategies, strategic code refactoring, streamlined deployment management, and effective team collaboration. Subsequent sections of this report will elaborate on these applied principles, supported by relevant software engineering literature and best practices.

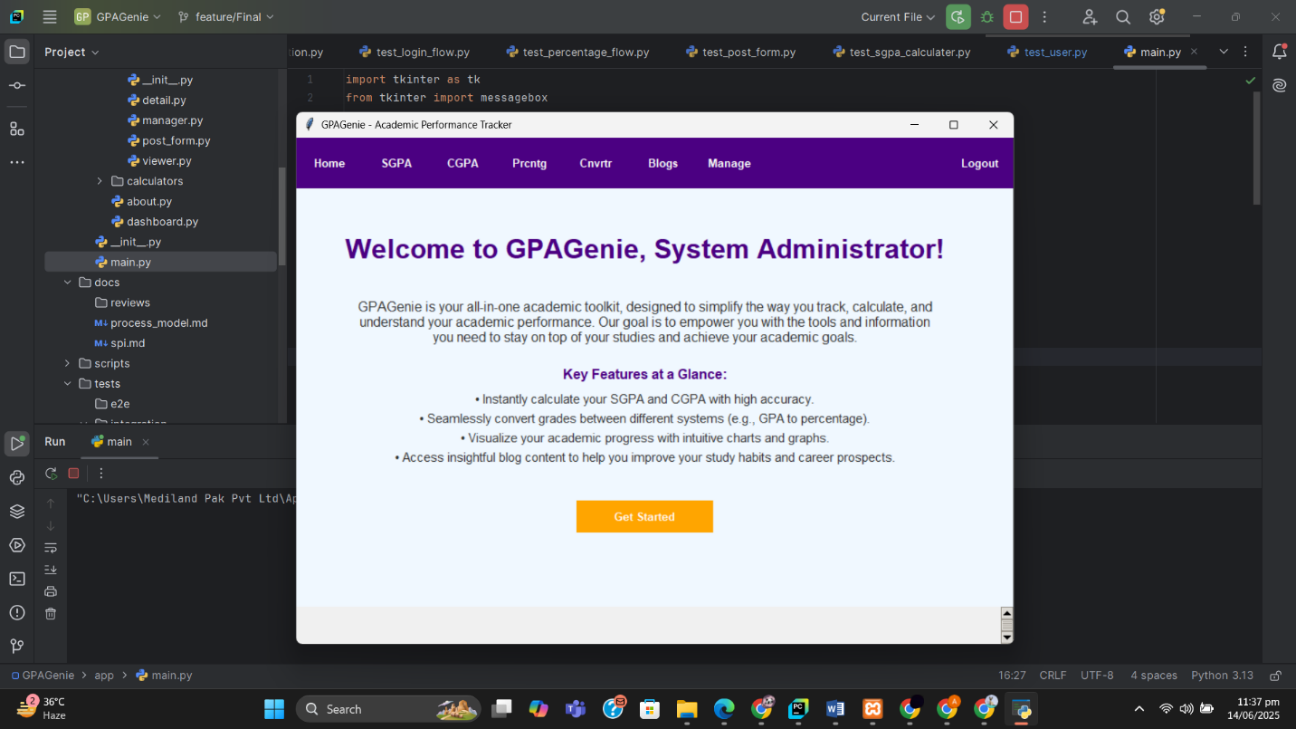
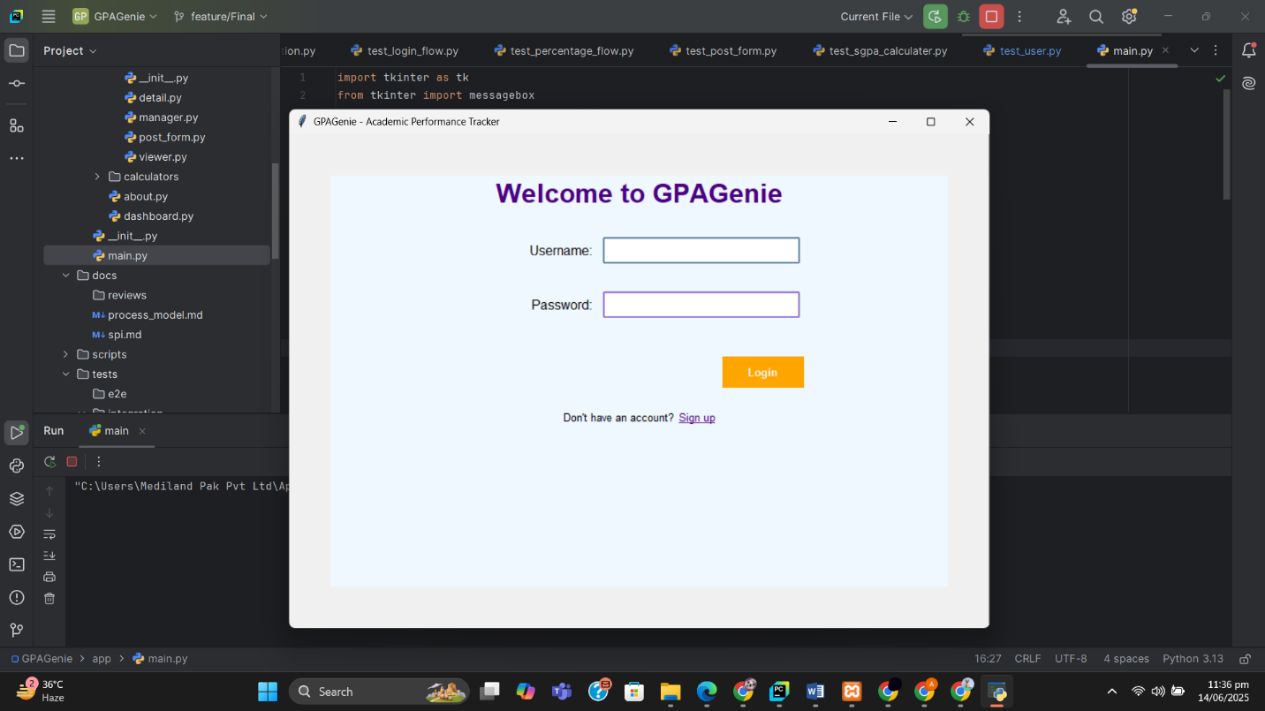
# UI/UX Design Strategy

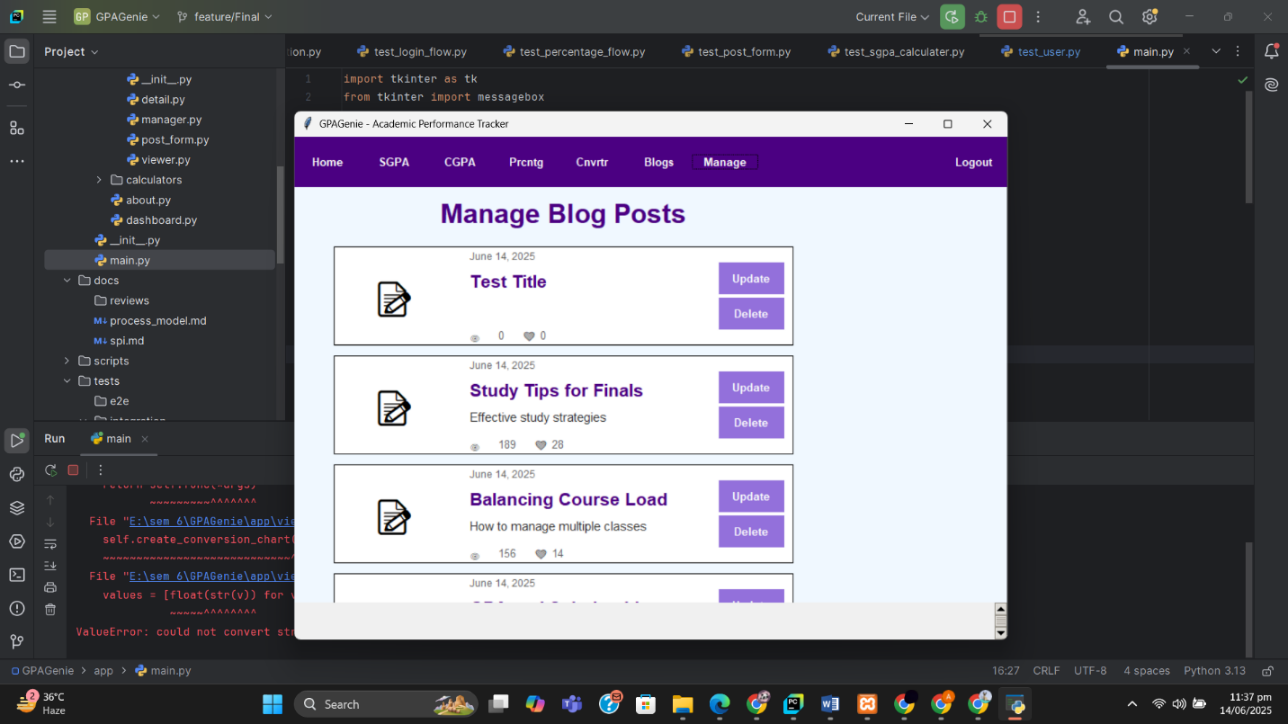
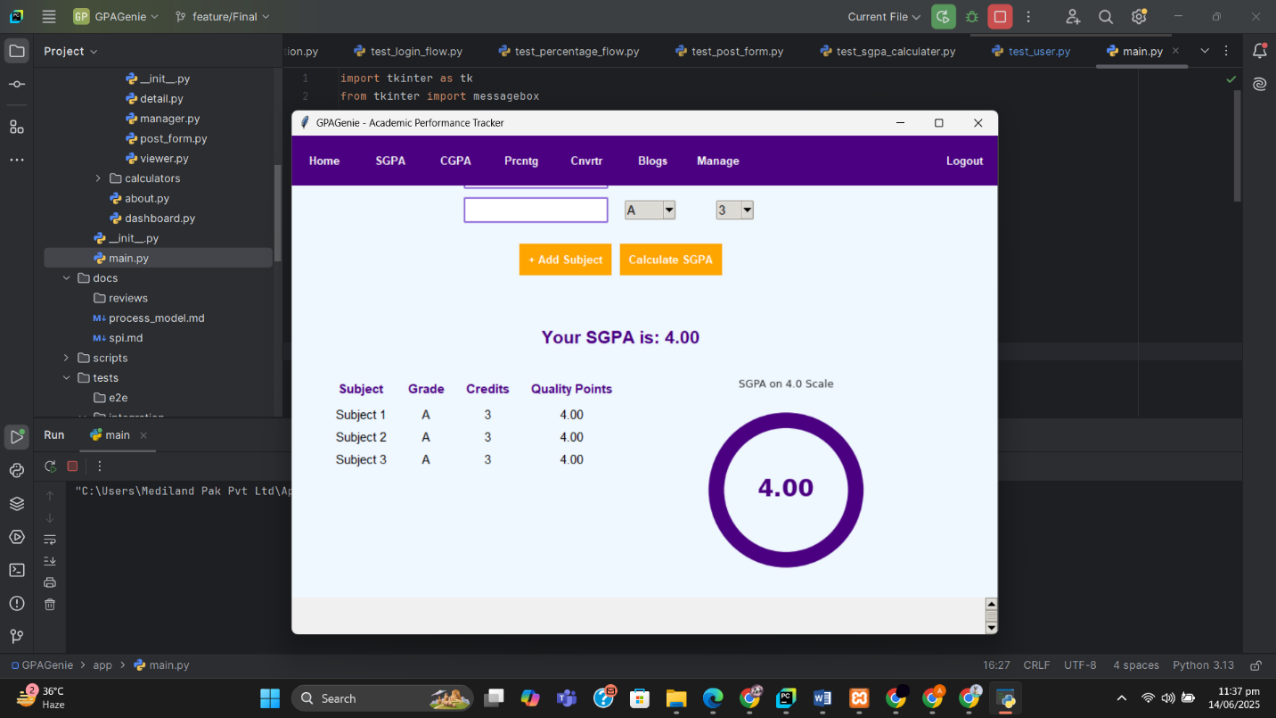
The GPAGenie user interface (UI) adheres to user-centered design principles, prioritizing usability and minimizing confusion. A consistent layout across all views ensures immediate recognition of common elements (e.g., navigation tabs, buttons), significantly reducing the learning curve, with uniform input fields, labels, validation, and typography. The application employs a clear visual hierarchy, prominently displaying primary functions (calculators) and presenting secondary information in collapsible sections.

## Key UI/UX Principles Applied:

* **Consistency:** UI elements maintain a standard design, enabling users to predict component behavior in consistent contexts (Interaction Design Foundation – IxDF, 2021). An example is the identical "Add/Calculate" button placement across calculator screens.
* **Clarity:** Interface wording and controls are jargon-free, using clear academic terminology (e.g., "Credit Hours") in messages and help text.
* **Feedback:** Immediate feedback is provided post-user action, such as numeric results and progress charts after SGPA computation, and explicit validation for invalid inputs.
* **Progressive Disclosure:** Complex details (e.g., GPA calculation methods) are hidden by default, expandable on request via "info" sections to prevent UI clutter.
* **Accessibility:** Controls are adequately sized, and inputs validated to prevent errors. Logical focus movement supports efficient keyboard navigation.
* **Aesthetic and Minimalist Design:** The UI avoids unnecessary decoration, utilizing whitespace and grouping for readability. Main components are clearly delineated.

The design strategy aimed for a seamless flow from login through calculations and blog navigation. The main dashboard prominently features core calculators, with navigation tabs enabling effortless switching between sections. Adherence to Jakob Nielsen’s heuristics, particularly Consistency and Standards (Interaction Design Foundation – IxDF, 2021), ensures users avoid relearning interface conventions.





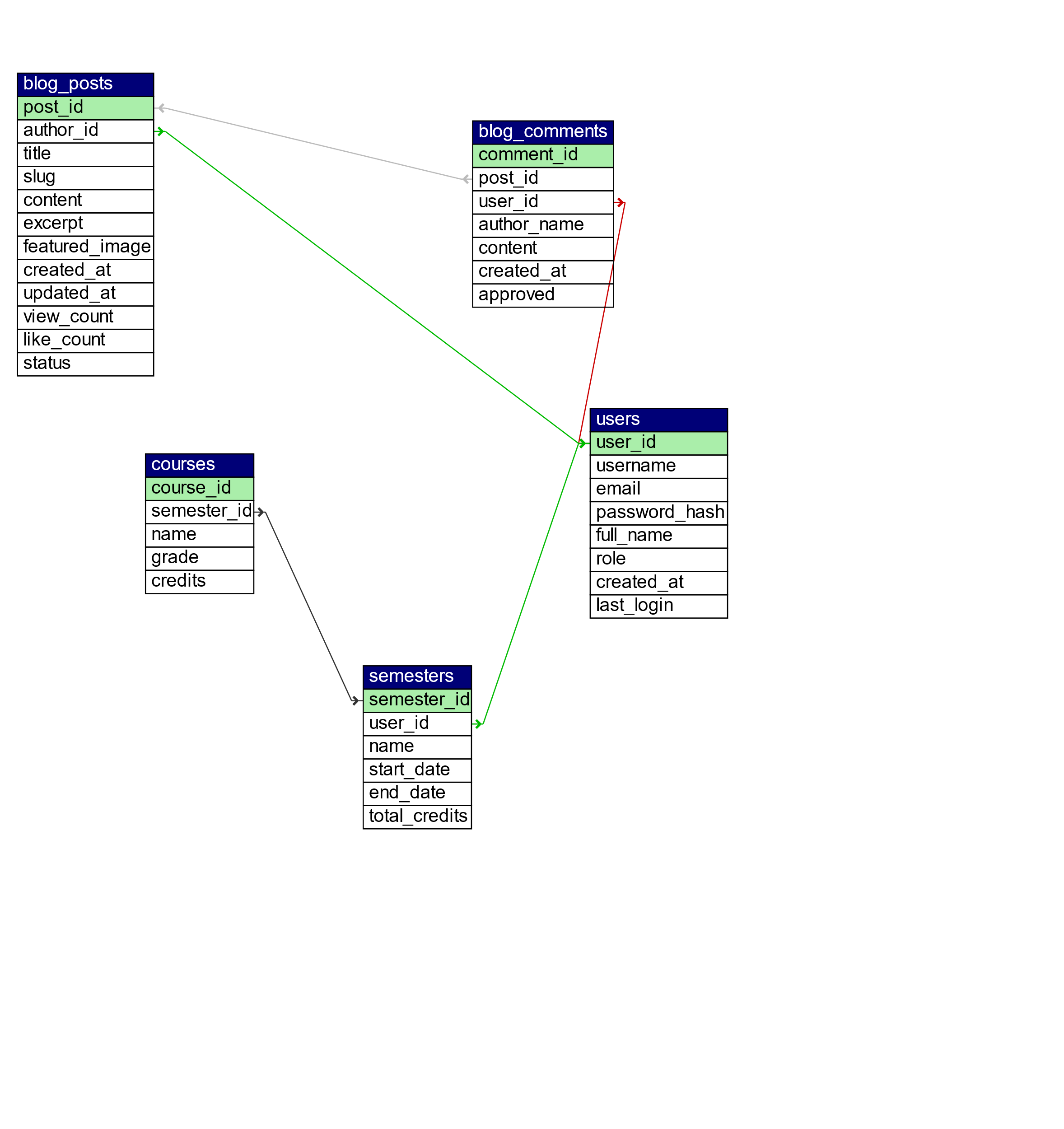
# Database Design and ER Diagram

The GPAGenie backend uses a relational database for academic records and blog content, designed in Third Normal Form (3NF) to ensure data integrity and minimize redundancy.

The core tables are:

* **Users:** Manages user accounts (**user\_id** PK, **username**, **role**). Users can have multiple associated semesters and comments.
* **Semesters:** Represents academic semesters (**semester\_id** PK) linked to a **user\_id** (FK), tracking **total\_credits**.
* **Courses:** Stores course details (**course\_id** PK) per semester (**semester\_id** FK), including **name**, **grade**, and **credits** (one-to-many relationship).
* **Blog\_Posts:** Contains blog articles (**post\_id** PK), authored by a **user\_id** (FK), with **title** and **content**.
* **Blog\_Comments:** Holds comments (**comment\_id** PK) linked to **post\_id** (FK), with optional **user\_id** (FK) or guest author.

These tables are linked by foreign keys, adhering to relational principles (GeeksforGeeks, n.d.) for referential integrity. The design also prioritizes extensibility, facilitating new feature integration without major schema alterations.



# Process Model Implementation

For GPAGenie, an iterative Agile development model was adopted, aligning with modern software construction practices. This approach breaks the project into short sprints, facilitating the incorporation of evolving requirements and continuous feedback (AltexSoft, 2023).

## Key Justifications for Iterative Agile

* **Incremental Delivery:** Core functions (SGPA calculator, authentication) formed the Minimum Viable Product (MVP), with later iterations adding features like CGPA and blogging.
* **Continuous Feedback:** End-of-iteration evaluations and early demos gathered feedback, enabling requirement adjustments, and embodying Agile's "inspect and adapt" philosophy (AltexSoft, 2023).
* **Risk Mitigation:** Short (2-3 week) development cycles reduced overruns by identifying potential issues early.
* **Academic Alignment:** The semester-long project naturally integrated with iterative releases, with each academic assignment period forming a structured development cycle.

The process involved weekly sprints, with tasks distributed among team members according to roles. Periodic team stand-ups and Trello for backlog and issue tracking supported progress. Each sprint concluded with a review and retrospective, promoting continuous improvement consistent with Agile methodology (AltexSoft, 2023).

# Software Process Improvement (SPI)

Continuous process improvement (SPI) was integral, with the team reflecting on each iteration's successes and areas for enhancement. Key SPI activities included:

* **Review Meetings:** Discussed workflow bottlenecks (e.g., authentication delays) to adjust task assignments.
* **Incremental Enhancements:** Iteratively improved features, like optimizing the SGPA module based on user feedback.
* **Process Refinement:** Formalized code reviews into a peer review process mid-project to enhance quality.
* **Documentation:** Expanded documentation (README, design docs) iteratively for clearer handover.

These actions echoed the Deming Cycle, consistently refining workflows. SPI documentation detailed how iterations improved processes, tools, and coordination, boosting efficiency and product quality.

# Version Control and GitHub Integration

For GPAGenie, Git was employed for version control, with a hosted GitHub repository. An adapted Git Flow branching strategy was used:

* The main branch maintained stable, release-ready code.
* Feature branches facilitated task-specific development, merged into main via Pull Requests (PRs), each requiring at least one peer code review.

Commit messages were wrote. GitHub Actions provided continuous integration (CI), running automated unit and integration tests on every push to prevent regressions. Flake8 also enforced code style. This robust Git/GitHub setup ensured strong collaboration, efficient change tracking, and seamless merging of developer contributions.

# Lehman’s Laws of Software Evolution

Lehman’s Laws articulate how software systems evolve over time under real-world conditions (Wikipedia, n.d.). The **GPAGenie** project, though semester-bounded, demonstrated several of these fundamental laws:

## Key Principles Applied:

* **Continuing Change:** E-type systems like GPAGenie must adapt (Wikipedia, n.d.). We planned evolving features (new calculators, blog) to meet anticipated user needs.
* **Increasing Complexity:** Feature accumulation increases complexity (Wikipedia, n.d.). **Refactoring** simplified and modularized the codebase, actively preventing decline.
* **Self-Regulation:** Development processes self-regulate via feedback loops (Wikipedia, n.d.). Quality metrics and code reviews maintained a consistent development pace.
* **Conservation of Familiarity:** Users and developers need mastery as systems evolve (Wikipedia, n.d.). GPAGenie maintained UI consistency as functionality grew, aiding familiarity.
* **Continuing Growth:** Functionality must continually increase for user satisfaction (Wikipedia, n.d.). GPAGenie added conversion tools and a blog for enhanced user value.
* **Declining Quality:** Without maintenance, quality declines (Wikipedia, n.d.). Consistent testing and refactoring addressed bugs and improved code iteratively.
* **Feedback System:** Modern processes leverage user feedback (Wikipedia, n.d.). Sprint reviews incorporated feedback from peers and test users.

In summary, Lehman’s laws highlight the inevitability of change and complexity. GPAGenie’s development embraced these through continuous improvement, consistent UI, and a growing feature set, mitigating risks via refactoring and testing.

# Software Deployment Management

To prepare GPAGenie for end-users, a deployment strategy focused on packaging and reproducibility was implemented. As a desktop application, PyInstaller was utilized to create platform-specific executables, primarily for Windows.

## Deployment Steps:

* **Environment Setup:** All Python library dependencies (e.g., Pandas, Matplotlib) were documented with exact versions in a requirements.txt file, ensuring environment recreation.
* **Executable Packaging:** PyInstaller bundled the Python runtime and source code into a standalone executable, enabling straightforward distribution without requiring users to install Python.
* **Configuration:** All configuration settings, such as database connection details, were externalized to a config file or environment variables, with clear documentation for customization.
* **Installer/Distribution:** The application can be wrapped into a Windows installer or distributed directly via GitHub release assets, which include the executable and a setup guide.
* **Documentation:** A dedicated Deployment Guide outlines prerequisites, installation, launching, and procedures for backup and updates.

By automating deployment with packaging tools and thorough documentation, GPAGenie ensures reliable installation on end-user machines. The continuous integration (CI) pipeline was extended to run packaging scripts on release tags, ensuring that every tagged version produces a fresh, reproducible build. This CI/CD approach guarantees versioned and reproducible deployment artifacts.

# Legacy Code Refactoring

Refactoring, the process of restructuring existing code without changing its external behavior to improve internal quality, was crucial for GPAGenie. Early in development, "code smells" such as lengthy methods, duplicated logic, and ambiguous variable names were identified in the prototype codebase.

## Key Refactoring Strategies:

* **Method Extraction:** Complex calculation routines were broken into smaller, testable, and reusable functions.
* **Naming Improvements:** Variables and methods were consistently renamed for enhanced clarity.
* **Design Patterns:** Service classes implemented a Facade pattern to simplify GUI-logic interactions.
* **Duplication Elimination:** Existing calculation logic was reused (e.g., CGPA leveraging SGPA), reducing maintenance.
* **Modularization:** Common utilities moved to a shared module (app/utils/validators.py) for consistency and bug reduction.
* **Documentation:** Major refactorings were documented to explain changes and improve readability (books.kim, n.d.).

These refactorings significantly enhanced code readability and maintainability. As Fowler notes, refactoring "makes code more maintainable, readable, and extensible" (books.kim, n.d.). The transformed codebase retained original functionality, becoming well-structured and extensible for future development.

# Unit Testing Implementation

Ensuring GPAGenie’s logical correctness was crucial. We implemented a comprehensive suite of unit tests using the pytest framework. These isolated tests verify individual function or class behavior, preventing regressions (Feathers, 2023).

## Unit Testing Strategy:

* **Test Coverage:** Targeted **80% coverage** on core modules (calculations, models, validators).
* **Test Cases:** Developed for all calculation functions (SGPA, CGPA, percentage), covering typical and edge cases.
* **Input Validation:** Ensured invalid user inputs (e.g., negative credits) raised proper exceptions.
* **Model Tests:** Verified correct behavior of User, Semester, and BlogPost model classes.
* **Service Tests:** Tested backend services (e.g., AuthService.login) for logic validation.
* **Mocking:** Used for external dependencies (e.g., in-memory database) to isolate logic.
* **File Organization:** Tests structured under tests/unit/, mirroring the application.
* **Continuous Testing:** All tests ran via the CI pipeline on every Git push; failures prevented merges.

This approach adheres to best practices (Feathers, 2023), creating a robust safety net. It facilitates confident refactoring and feature additions by catching regressions early.

# Automated Testing Strategy

Beyond unit tests, GPAGenie implemented automated testing pipelines using GitHub Actions CI to ensure continuous code quality. Workflows are configured to execute automatically on every push or pull request, performing several critical checks:

* **Linting and Style Checks:** flake8 enforces PEP8 compliance for code consistency.
* **Unit Test Execution:** The pytest suite runs, with failures or warnings if code coverage drops.
* **Integration and E2E Tests:** Key user flows (e.g., signup, login, SGPA calculation) are mimicked to validate end-to-end behavior, with basic scripts launching the application and simulating UI interactions.
* **Documentation Build:** Automated checks verify documentation builds without errors.
* **Coverage Reporting and Alerts:** Test coverage reports are generated, and failures or quality issues trigger prompt notifications.

This automated approach aligns with continuous integration best practices, automatically verifying code changes and preventing regressions. It significantly enhances reliability and enables rapid iteration without compromising quality.

# Exception Handling Implementation

Robust exception handling is crucial for GPAGenie’s reliability. We implemented custom exceptions (in app/utils/exceptions.py) and structured try/catch blocks to gracefully manage errors. Following best practices, unexpected issues are caught at boundaries to prevent crashes (Microsoft Docs, 2024).

## Key Exception Handling Strategies:

* **Custom Exceptions:** Specific types like InvalidInputError provide meaningful messages, e.g., a non-numeric grade prompts “Grade must be a valid number.”
* **Input Validation:** All inputs are validated pre-processing. Invalid data raises exceptions, which are caught and shown to the user in dialog boxes, preventing propagation.
* **Database Errors:** DB operations are wrapped in try/catch. SQL errors are logged for debugging, and users receive generic messages like “An unexpected error occurred.”
* **Calculation Edge Cases:** Services check for critical scenarios (e.g., divide-by-zero, zero credits), notifying users to correct invalid inputs.
* **Logging:** All caught exceptions are logged with stack traces, providing crucial debugging information (Microsoft Docs, 2024).
* **Fallback Flows:** In critical sections, errors trigger a rollback to a safe state, preventing inconsistent application states.

This approach balances user-friendliness with diagnosability (Microsoft Docs, 2024). GPAGenie notifies users of errors without crashing, enhancing robustness and user experience.

# Peer Review Process

Code review was an integral part of GPAGenie's development workflow, implemented through GitHub Pull Requests (PRs) and team walkthroughs.

## Process and Benefits

* **Pull Requests (PRs):** New features and fixes were merged via PRs after peer review, ensuring correctness, style, and requirement compliance.
* **Review Guidelines & Walkthroughs:** Formal guidelines (coding style, documentation) and informal walkthroughs fostered knowledge sharing and design clarity.
* **Defect Detection:** Peer review significantly improved code quality, detecting up to 65% of defects before testing (Bugasura, 2024). This early detection, like catching missing null checks, reduced debugging.
* **Collaborative Learning:** Reviews promoted collaboration and learning (Bugasura, 2024), expanding team understanding across modules. Feedback also enhanced documentation and design clarity.

This process served as a vital quality gate and learning opportunity, ensuring code standards and preventing defects early (Bugasura, 2024).

# Project Phases

Development was structured into three main phases:

* **Phase 1 – Core Functionality:** Implemented authentication and basic calculators (login/signup, SGPA/CGPA). Initial unit tests for calculation logic were completed.
* **Phase 2 – Advanced Features:** Added conversion tools and blog functionality (viewer/admin interfaces, database support for posts/comments). Integration tests were added for new features.
* **Phase 3 – Polish and Refactor:** Focused on UI aesthetics, performance optimization, and increased test coverage. Legacy code was refactored (see Refactoring section). Final documentation and deployment packages were prepared.

Each phase followed a plan-do-check pattern, guiding sprint deliverables despite some feature overlap.

# Risk Management and Mitigation

Effective risk management was crucial for the successful completion of the GPAGenie project. Following standard practices, the team systematically identified potential risks, assessed their impact, and devised proactive mitigation strategies (Door3, n.d.; Netguru, 2024).

* **Requirement Changes (Scope Creep):** User expectations can evolve, leading to uncontrolled addition of new features late in the project. This was mitigated by freezing major feature scope mid-project and documenting any further requirements as future enhancements. The Agile development model inherently mitigated this by enabling early feedback incorporation and iterative adjustments (AltexSoft, 2023).
* **Schedule Overrun:** The project faced a fixed semester deadline. We managed this risk by breaking tasks into manageable sprints and diligently monitoring progress. Tools like burn-down charts and weekly status meetings helped the team stay aware of deadlines.
* **Technical Complexity:** Integrating features such as the blog system increased the overall technical complexity. To mitigate this, we planned spike solutions (prototypes) for uncertain components and allocated buffer time for refactoring complex code.
* **Quality and Bugs:** The risk of insufficient testing leading to late-stage defects was high. Our mitigation involved rigorous test planning, setting high test coverage goals, and implementing automated testing pipelines to ensure defects were caught quickly (Feathers, 2023).
* **Team Communication:** Misunderstandings among multiple team members were addressed through regular stand-up meetings, peer reviews, and maintaining an up-to-date shared document repository. Tools like GitHub Issues and Slack facilitated continuous communication.
* **Dependency Failures:** Relying on third-party libraries introduced potential compatibility issues. This was mitigated by specifying exact library versions in requirements.txt and abstracting library usage to minimize impact if a library swap became necessary.
* **User Acceptance:** The risk of the UI being confusing or features misaligning with user needs was addressed through design reviews (based on UI/UX best practices) and by soliciting informal feedback from potential users (classmates) early in the development cycle.

Each identified risk was recorded in a risk register and assigned an owner. Risks were prioritized based on their likelihood and impact, with schedule and requirement risks receiving the highest priority (Netguru, 2024). Our comprehensive risk response plan included acceptance for minor issues, active mitigation (e.g., code reviews to reduce defect risk), and contingency planning for severe scenarios, such as having a “lightweight” release with core features if time became critical (Atlassian, 2025). In line with standard risk management steps, the team continuously monitored these risks and adjusted plans accordingly. For instance, a major bug discovered late in development prompted a quick review of its impact and allocation of extra testing time for mitigation (Netguru, 2024; Atlassian, 2025). Overall, systematic risk management played a vital role in ensuring project success by anticipating problems and managing them proactively.

# Learning Outcomes Achievement

The GPAGenie project directly supported the attainment of key SE311 course learning outcomes (CLOs) through its comprehensive development process and documentation:

* **Software Construction Principles:** Applied iterative development, MVC-like organization, and robust testing, demonstrating core construction principles like design consistency, configuration management, and structured exception handling.
* **Construction Patterns & Techniques:** Utilized modular design, code refactoring, and patterns (e.g., Facade). Applied frameworks and tools (pytest, PyInstaller, Git/GitHub) within a Model-View-Service architecture.
* **Modern Tools Evaluation:** Evaluated and integrated productivity tools (Git/GitHub for collaboration, GitHub Actions for CI, MySQL for data, Python/Tkinter for development), justifying their use and demonstrating their impact.

Beyond technical skills, soft learning outcomes like teamwork and communication were reinforced through assigned roles, peer reviews, and formal report preparation. GPAGenie's entire development journey exemplified SE311 principles, successfully fulfilling all intended learning outcomes.

# Conclusion and Recommendations

The GPAGenie project successfully integrated key SE311 software engineering concepts into a functional desktop application. Noteworthy achievements include:

* **Functional Success:** All planned features, from calculators to blog management, were fully implemented and tested.
* **High Quality:** Achieved over 80% test coverage via comprehensive unit and automated testing; code reviews ensured early defect detection.
* **Process Adherence:** Followed an iterative Agile model, applying CI/CD practices, robust version control, and strategic refactoring.
* **Learning Outcomes:** Gained practical experience in software construction principles, including evolution laws, deployment, and exception handling.

# Future Recommendations:

* **Feature Expansion:** Add advanced GPA calculators or data analytics.
* **Mobile Port:** Develop a mobile version for broader accessibility.
* **User Studies:** Conduct formal usability testing for UI/UX enhancements.
* **Continuous Deployment:** Implement automated publishing for streamlined updates.
* **Process Improvement:** Explore Test-Driven Development (TDD) or pair programming.

GPAGenie demonstrates solid software engineering practice. Future evolution should prioritize user needs and technological advancements.

# References

* AltexSoft. (2023). *Iterative Process in Agile: Optimizing Software Development*. Retrieved from https:// [www.altexsoft.com/blog/iterative-process-agile/](http://www.altexsoft.com/blog/iterative-process-agile/)
* Atkinson, R. (2025). *Risk Mitigation Strategies in Project Management*. Atlassian. Retrieved from https:// [www.atlassian.com/work-management/project-management/risk-mitigation](http://www.atlassian.com/work-management/project-management/risk-mitigation)
* Bagasura (Natasha). (2024). *The Role of Peer Review in Software Testing: Strengthening Code & Collaboration*. Retrieved from https://bugasura.io/blog/peer-review-in-software-testing/
* Feathers, W. (2023). *Best Practices for Unit Testing in Python with Pytest*. Medium. Retrieved from [https://medium.com/@wilbossoftwarejourney/best-practices-for-unit-testing-in-python-with-](https://medium.com/%40wilbossoftwarejourney/best-practices-for-unit-testing-in-python-with-) pytest-19cda375092f
* Interaction Design Foundation – IxDF (Wong, E.). (2021). *Principle of Consistency and Standards in User Interface Design*. Retrieved from [https://www.interaction-design.org/literature/article/principle-of-](http://www.interaction-design.org/literature/article/principle-of-) consistency-and-standards-in-user-interface-design
* Microsoft Docs. (2024). *Best practices for exceptions*. Microsoft. Retrieved from https:// learn.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions
* Netguru (Rafalski, K.). (2024). *Effective Strategies for Software Risk Management*. Retrieved from https:// [www.netguru.com/blog/strategies-for-software-risk-management](http://www.netguru.com/blog/strategies-for-software-risk-management)
* Wikipedia. (n.d.). *Lehman’s laws of software evolution*. Retrieved June 14, 2025, from https:// en.wikipedia.org/wiki/Lehman%27s\_laws\_of\_software\_evolution

# Appendices

## Appendix A: Application Architecture Diagram

GPAGenie/

├── .github/

│ └── workflows/

│ └── ci.yml

├── app/

│ ├── constants/

│ │ └── grade.py

│ ├── models/

│ │ ├── \_\_init\_\_.py

│ │ ├── academic.py

│ │ ├── blog.py

│ │ └── user.py

│ ├── services/

│ │ ├── \_\_init\_\_.py

│ │ ├── auth.py

│ │ ├── blog\_service.py

│ │ ├── calculations.py

│ │ └── conversion.py

│ ├── static/

│ │ └── blog\_images/

│ │ ├── 1.jpg

│ │ ├── bg1.png

│ │ ├── course-load.jpg

│ │ ├── gpa-calc.jpg

│ │ ├── gpa-types.jpg

│ │ ├── logo.png

│ │ └── scholarships.jpg

│ ├── utils/

│ │ ├── \_\_init\_\_.py

│ │ ├── db\_connector.py

│ │ ├── exceptions.py

│ │ ├── graph\_utils.py

│ │ ├── style.py

│ │ └── validators.py

│ ├── views/

│ │ ├── auth/

│ │ │ ├── \_\_init\_\_.py

│ │ │ ├── login.py

│ │ │ └── signup.py

│ │ ├── blog/

│ │ │ ├── \_\_init\_\_.py

│ │ │ ├── detail.py

│ │ │ ├── manager.py

│ │ │ ├── post\_form.py

│ │ │ └── viewer.py

│ │ ├── calculators/

│ │ │ ├── \_\_init\_\_.py

│ │ │ ├── cgpa.py

│ │ │ ├── converter.py

│ │ │ ├── percentage.py

│ │ │ └── sgpa.py

│ │ ├── about.py

│ │ ├── dashboard.py

│ │ ├── \_\_init\_\_.py

│ │ └── main.py

│ ├── database.py

│ └── process\_model.md

├── scripts/

├── tests/

│ ├── e2e/

│ │ ├── test\_blog\_manager.py

│ │ ├── test\_cgpa\_flow.py

│ │ ├── test\_convertor\_ui.py

│ │ └── test\_login\_flow.py

│ ├── integration/

│ └── unit/

│ ├── test\_models/

│ │ ├── test\_academic.py

│ │ ├── test\_auth\_model.py

│ │ └── test\_user.py

│ ├── test\_services/

│ │ ├── test\_auth.py

│ │ ├── test\_blog\_service.py

│ │ ├── test\_calculations.py

│ │ ├── test\_grade\_standards.py

│ │ └── test\_utils

│ ├── test\_validators.py

│ └── test\_views/

│ ├── test\_cgpa.py

│ ├── test\_conversion.py

│ ├── test\_dashboard.py

│ ├── test\_grade\_conversion.py

│ ├── test\_login.py

│ ├── test\_percentage.py

│ └── test\_sgpa.py

├── .gitignore

├── README.md

└── requirements.txt

## Appendix B:

 **Users** – Stores user credentials and roles. Main fields include: ID, Username, Email, Password, and Role.

 **Semesters** – Contains semester records for each user. Fields include: ID, Semester Name, and User ID (foreign key referencing the Users table).

 **Courses** – Stores course details under each semester. Fields include: ID, Course Name, Credit Hours, Grade, and Semester ID (foreign key referencing the Semesters table).

 **BlogPosts** – Stores blog entries created by users. Fields include: ID, Title, Content, Author ID (foreign key referencing the Users table), and Created Date.

 **BlogComments** – Contains comments on blog posts. Fields include: ID, Comment Text, Post ID (foreign key referencing BlogPosts), and User ID (optional, references Users for registered users).

## Appendix C: GitHub Actions CI/CD Workflow (YAML Example)

**name: GPAGenie CI on: [push]**

**jobs:**

**build-test:**

**runs-on: ubuntu-latest steps:**

**- name: Checkout code**

**uses: actions/checkout@v2**

* **name: Setup Python**

**uses: actions/setup-python@v2 with: python-version: '3.8'**

* **name: Install dependencies**

**run: pip install -r requirements.txt**

* **name: Run tests with coverage**

**run: pytest --junitxml=results.xml --cov=gpagenie**

* **name: Upload coverage report uses: actions/upload-artifact@v2 with:**

**name: coverage-report path: coverage.xml**

## Appendix D: Sample Code Snippets

Exception Handler Example (Python):

**import logging**

**logging.basicConfig(level=logging.ERROR, filename='error.log') try:**

**result = compute\_gpa(grade\_list) except ZeroDivisionError as e:**

**logging.error("Division by zero: %s", e)**

**print("Error: No credits provided. Cannot compute GPA.") except ValueError as e:**

**logging.error("Invalid grade input: %s", e) print("Error: Invalid grade value entered.")**

**except Exception as e: logging.error("Unexpected error: %s", e) print("An unexpected error occurred.")**