GROUP P SOFTWARE ENGINEERING

ST PAUL’S UNIVERSITY

LIMURU CAMPUS

REGULAR PROGRAM

DEPARTMENT OF

SCHOOL OF COMPUTER SCIENCE

BCS 3106: SOFTWARE ENGINEERING

DOCUMENTATION

REGISTRATION NUMBER:

**BSCLMR165923- CHRISTINE GUYO**

**BSCLRM162523- EDDYJOY MATEMO**

**BSCLMR164923- HASSAN M. TACHE**

**BSCLMR402922 JEREMIAH EKITELA**

**BSCLMR165220-**

LECTURER(S): CECILIA NANFUKA

MARCH 8th, 2024

* **Software Architecture & Design**

**1. Overall Architecture:**

**1. Client-Server Communication:**

* In the client-server architecture, the client (web browser) sends requests to the backend server, which processes these requests and returns responses to the client.
* The communication between the client and server follows the HTTP protocol, with the client typically initiating requests (e.g., submitting a vote) and the server responding accordingly (e.g., confirming the vote was received and processed).

**2. Scalability:**

* The architecture is designed to be scalable, meaning it can handle an increasing number of users and requests without sacrificing performance or reliability.
* Scalability is achieved through various techniques such as load balancing, horizontal scaling (adding more servers), vertical scaling (increasing server resources), and caching.
* By adopting scalable architecture patterns, the application can accommodate a growing number of students participating in the elections without experiencing performance degradation or downtime.

**3. Backend Server:**

* The backend server is the heart of the application, responsible for executing the core functionalities such as authentication, registration, voting, and data management.
* It consists of software components implemented using a server-side programming language such as Python, along with a web framework like Flask or Django.
* The server-side logic is organized into modules or components, each responsible for a specific task or functionality (e.g., authentication module, voting module).
* The backend server communicates with the frontend client via HTTP requests and responses, handling user interactions and data processing seamlessly.

**4. Separation of Concerns:**

* The architecture emphasizes the separation of concerns principle, which divides the application's functionality into distinct modules or layers, each responsible for a specific aspect of the system.
* Separation of concerns promotes modularity, maintainability, and reusability by isolating different concerns (e.g., presentation logic, business logic, data access logic) from each other.
* For example, the authentication module focuses solely on user authentication and access control, while the voting module handles the process of casting votes and ensuring data integrity.

**5. Relational Database Management System (RDBMS):**

* The database serves as the persistent storage layer for storing and managing data related to student information, votes, and other election-related data.
* A relational database management system (RDBMS) such as MySQL or PostgreSQL is commonly used due to its robustness, scalability, and support for complex data relationships.
* The database schema is designed to reflect the application's data model, with tables representing entities such as students, votes, candidates, and election sessions.
* Proper indexing, normalization, and optimization techniques are applied to ensure efficient data retrieval, storage, and querying, maintaining data integrity and consistency.

**6. Middleware and APIs:**

* Middleware components and APIs may be utilized to facilitate communication between the frontend and backend, providing a standardized interface for exchanging data and invoking functionality.
* APIs (Application Programming Interfaces) define the contract between different software components, allowing them to communicate and interact seamlessly.
* Middleware components, such as authentication middleware or logging middleware, intercept and process requests before they reach the application logic, enabling cross-cutting concerns to be addressed in a modular and reusable manner.

**7. Components:**

* + Client Side:
* The user interface (UI) is developed using HTML, CSS, and JavaScript to provide an engaging and interactive experience for students. HTML provides the structure, CSS styles the elements, and JavaScript adds dynamic behavior to the UI.
  + Server Side:
* The Flask web framework is chosen for the backend due to its simplicity, flexibility, and ease of use. Flask facilitates the development of web applications by providing tools and utilities for routing, request handling, and template rendering.
* Authentication and registration modules are implemented to manage user access to the application securely. Authentication verifies user identity during login, while registration allows new users to create accounts and participate in the election process.
* The voting module ensures the integrity and confidentiality of the voting process. It allows registered students to cast their votes securely while preventing multiple votes from the same user.
* Database management functionalities interact with the database to store and retrieve student data, manage votes, and maintain data integrity. SQLAlchemy, an Object-Relational Mapping (ORM) library, is utilized to facilitate database operations and ensure compatibility with various database systems.
* Security measures such as encryption, authentication, and authorization are implemented to protect sensitive data and prevent unauthorized access. Passwords are hashed before being stored in the database to safeguard user credentials, while access control mechanisms ensure that only authorized users can access certain functionalities.

**8. Data Flow:**

* The data flow within the application follows a structured path, starting from user interactions on the frontend and culminating in data processing and storage on the backend.
* During the registration and login processes, student data is transmitted from the frontend to the backend, where it is validated and securely stored in the database.
* When students cast their votes, the vote data is securely transmitted from the frontend to the backend, processed, and stored in the database. Timestamps are added to the vote records to track when the votes were cast, ensuring transparency and accountability in the election process.
* Authorized personnel can access and analyze the stored data to tally votes and determine the election results accurately. They can retrieve data from the database using SQL queries or ORM methods provided by SQLAlchemy, enabling them to perform complex data analysis and generate reports.

**9. Database Schema:**

* The database schema defines the structure and organization of the database, including tables, columns, relationships, and constraints.
* For the student table, fields such as student ID, username, password (hashed), email, and registration date are included to store student information securely. The student ID serves as the primary key to uniquely identify each student record, while the username and email fields are indexed to facilitate efficient data retrieval and search operations.
* The vote table stores data related to votes cast by students, including the student ID, candidate ID, timestamp, and any additional metadata. The student ID and candidate ID fields serve as foreign keys to establish relationships with the student and candidate tables, ensuring referential integrity and enforcing data consistency.
* Proper normalization techniques are applied to eliminate data redundancy and ensure data integrity. Indexes and constraints are added to enforce data integrity rules, such as unique constraints on the username and email fields to prevent duplicate records.

**10. User Interface Design:**

* The user interface design is critical for ensuring a positive user experience and facilitating user interactions with the application.
* The design principles of simplicity, clarity, and consistency are adhered to throughout the user interface to minimize cognitive load and improve usability. Clear navigation paths and intuitive layouts guide users through the registration, login, and voting processes seamlessly.
* Visual elements such as color schemes, typography, and iconography are carefully chosen to create a visually appealing and cohesive design. Attention is paid to contrast, hierarchy, and spacing to enhance readability and highlight important information.
* Feedback mechanisms such as error messages, notifications, and confirmation dialogs provide timely feedback to users, helping them understand their actions and navigate the application effectively.
* Responsive design techniques are employed to ensure that the user interface adapts gracefully to different screen sizes and devices, providing a consistent experience across desktops, laptops, tablets, and smartphones.

**Test Plan*:***

**1. Unit Testing:**

* Unit testing is performed to validate the correctness and robustness of individual components or modules in isolation.
* Test cases are developed for each function or method within the application, covering various scenarios such as valid inputs, invalid inputs, edge cases, and error conditions.
* Mock objects or test doubles may be used to simulate external dependencies or isolate the component being tested, ensuring that unit tests remain focused and independent of external factors.
* Test coverage metrics are used to assess the effectiveness of unit testing, ensuring that critical paths and edge cases are adequately covered by test cases.

**2. Integration Testing:**

* Integration testing verifies the interaction and interoperability of different components or modules within the application.
* Test cases are designed to validate the data flow and communication between frontend and backend components, ensuring that data is transmitted and processed correctly.
* Integration tests may involve testing APIs, database interactions, and external integrations to ensure that all components work together seamlessly.
* Mocking or stubbing may be used to simulate external systems or dependencies that are not readily available in the test environment, enabling comprehensive integration testing without relying on external resources.

**3. User Interface Testing:**

* User interface testing evaluates the usability, accessibility, and visual design of the application's user interface.
* Test cases are developed to assess the intuitiveness of user interactions, navigation flows, and visual elements within the user interface.
* Usability heuristics and accessibility guidelines are used as criteria for evaluating the effectiveness of the user interface design, ensuring that it meets the needs of diverse user groups.
* Cross-browser testing and device testing are performed to verify that the user interface renders correctly and functions consistently across different web browsers and devices.
* Automated testing tools may be used to simulate user interactions and validate the behavior of the user interface under various conditions, enabling efficient and repeatable testing of UI components.

**4. Security Testing:**

* Security testing is conducted to identify vulnerabilities, weaknesses, and threats to the application's security posture.
* Various techniques such as penetration testing, vulnerability scanning, and code analysis are used to assess the security of the application's infrastructure, codebase, and configurations.
* Authentication mechanisms, authorization controls, data encryption, and secure communication protocols are scrutinized to ensure that sensitive information is protected from unauthorized access or disclosure.
* Compliance with industry standards and best practices such as OWASP Top 10, PCI DSS, and GDPR is validated to ensure that the application meets regulatory requirements and industry guidelines.
* Security testing is performed throughout the software development lifecycle, from design and implementation to deployment and maintenance, to identify and remediate security vulnerabilities proactively.

**5. Performance Testing:**

* Performance testing evaluates the responsiveness, scalability, and reliability of the application under various load conditions.
* Test scenarios are designed to simulate realistic user traffic and workload patterns, including peak usage periods, to assess the application's performance characteristics.
* Performance metrics such as response time, throughput, resource utilization, and scalability are measured and analyzed to identify performance bottlenecks and scalability limits.
* Load testing, stress testing, and endurance testing are performed to evaluate the application's behavior under different levels of load and stress, ensuring that it remains stable and responsive under challenging conditions.
* Performance testing tools and frameworks such as Apache JMeter, Gatling, and Locust may be used to automate test execution, collect performance metrics, and generate reports, enabling efficient and repeatable performance testing.

**6. End-to-End Testing:**

* End-to-end testing validates the functionality, reliability, and usability of the entire application workflow from start to finish.
* Test scenarios are designed to simulate real-world user interactions and usage scenarios, covering registration, login, voting, and result viewing processes.
* End-to-end tests may involve multiple system components, including frontend, backend, and database layers, to ensure that all components work together seamlessly to deliver the desired functionality.
* Realistic test data and user personas are used to simulate different user behaviors and usage patterns, ensuring comprehensive coverage of the application's features and functionalities.
* End-to-end testing is performed in environments that closely resemble the production environment, enabling accurate validation of the application's behavior and performance in a production-like setting.

**7. Regression Testing:**

* Regression testing ensures that existing functionalities are not affected by changes or updates made to the application.
* Test suites consisting of previously executed test cases are re-run after each software change to verify that existing features and functionalities still work as expected.
* Automated regression testing tools and frameworks are used to streamline the regression testing process, enabling efficient and thorough validation of software changes.
* Regression tests may be prioritized based on risk factors and criticality, focusing on high-impact areas and frequently modified components to maximize test coverage and effectiveness.
* Continuous integration and continuous deployment (CI/CD) pipelines are leveraged to automate regression testing as part of the software delivery pipeline, ensuring that software changes are thoroughly tested and validated before being deployed to production.

**8. User Acceptance Testing (UAT):**

* User acceptance testing involves real users testing the application to ensure that it meets their needs, requirements, and expectations.
* Test scenarios are derived from user stories, use cases, and user requirements, covering the most critical and commonly used features and functionalities of the application.
* UAT participants may include end-users, stakeholders, and domain experts who provide feedback and validation on the application's usability, functionality, and overall user experience.
* UAT sessions may be conducted in controlled environments or real-world settings, allowing users to interact with the application in familiar contexts and scenarios.
* Feedback gathered during UAT sessions is analyzed, prioritized, and incorporated into the software development process to address usability issues, functional gaps, and user preferences effectively.

**9. Deployment Testing:**

* Deployment testing ensures that the application is successfully deployed and operational in the production environment.
* Test cases are designed to validate the deployment process, including configuration management, environment setup, and application deployment steps.
* Smoke testing is performed to verify basic functionality and accessibility of the deployed application, ensuring that critical features are operational and accessible to users.
* Deployment testing may involve deploying the application to staging or pre-production environments first, allowing for final validation and testing before deploying to the production environment.
* Rollback procedures and contingency plans are established to address deployment failures or issues, ensuring minimal disruption to users and maintaining service availability.

**10. Documentation:**

* Comprehensive documentation is essential for ensuring the repeatability, maintainability, and understandability of the testing process and outcomes.
* Test plans, test cases, and test scripts document the testing approach, strategies, and procedures followed during testing, providing guidance and instructions for executing tests and interpreting results.
* Test reports summarize the results of testing activities, including findings, observations, and recommendations for improvement, providing stakeholders with insights into the quality and readiness of the software.
* User documentation provides instructions, tutorials, and troubleshooting guides for users to effectively use the application, helping them navigate the interface, understand features, and address common issues independently.
* Release notes document changes, enhancements, and bug fixes introduced in each software release, providing transparency and accountability in the software delivery process and keeping stakeholders informed about the latest developments and improvements.