**Customer Management Information System**

SID:20066316

Module Title: Programming for Information Systems

Module Code: B9IS123

Instructor: Paul Laid

Word Count: 2231

# 1. Introduction and System Requirements

The Customer Management Information System (CMIS) has been constructed and developed to serve as a proof-of-concept application for small businesses that need efficient customer data management capabilities. This system is essential in that it meets the basic requirements of an organization to store, organize, and manipulate customer data in a web-based interface. The adoption of a customer management system is appropriately based on the fact that it is universally applicable in any business environment that encompasses retail centers, service providers (e.g., banks), and other organizations where customer relationship management is the core of running the business.

The system's architecture is contemporary, as it leverages a strict distinction between the front end and back end. The frontend code is written in HTML, CSS, and JavaScript technologies to render an interactive user interface. In contrast, the backend is built using the Flask framework in Python to generate RESTful API endpoints. This is an architectural solution that offers scalability and maintainability, and it complies with modern web development requirements. The system reflects the central work of CRUD (Create, Read, Update, Delete) operations needed by any data management applications but is specifically adapted to work with customer data.

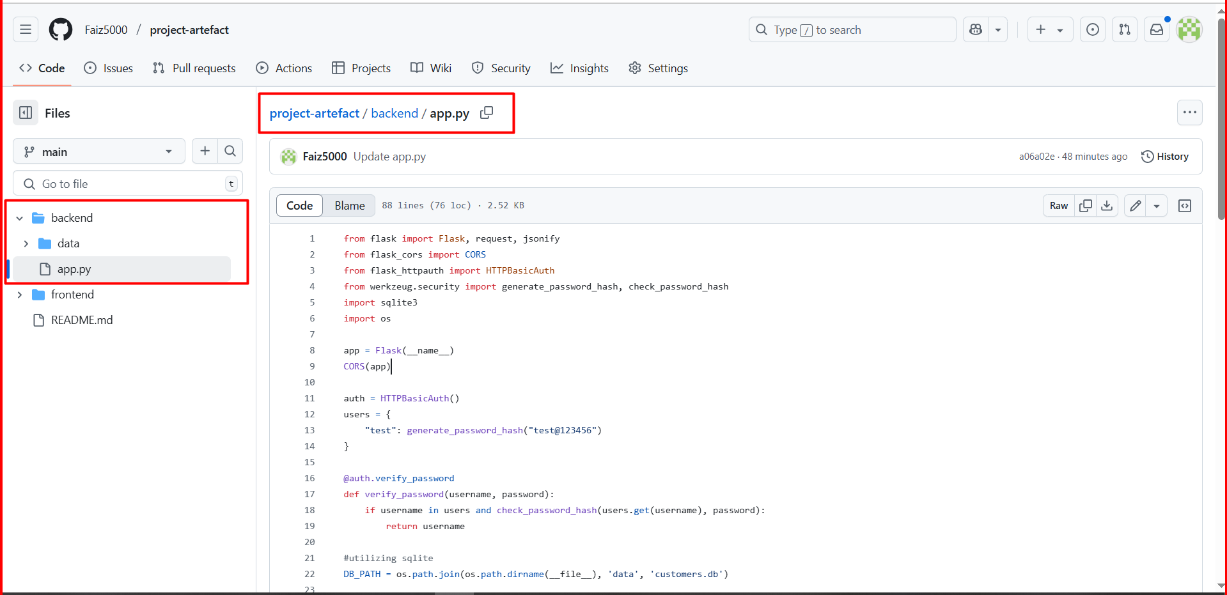
The primary objective of this implementation is to demonstrate the operating information system, which can serve as the basis for the subsequent development of more advanced customer-handling tools. The system requirements encompass both functional and non-functional aspects, covering areas such as data integrity, user-friendly interface design, and reliable API communication between the front and backend components (Grow Solutions, 2024). The implementation demonstrates how web development technologies can be applied in the real world while also focusing on core programming concepts rather than aesthetic designs.

The system also considers the special requirements of the small business that involve instant access to the information of the customers, the ability to enter new customers into the system easily, and the possibility to change the information about the existing customers in case something is changed and to delete useless archived information when they are required. These are the basic operations that a customer management system should offer to be considered a viable concept that businesses can utilize. Its implementation will enable these operations to be carried out in a reliable and efficient manner, with well-organized code and proper error handling procedures.

# 2. System Architecture and Implementation

The Customer Management Information System is a three-type architecture consisting of a data layer, an applications layer, and a presentation layer. The presentation layer is implemented using the HTML5 markup language, where JavaScript fragments are utilized to dynamically process the application's results and respond to user interactions. The HTML architecture provides meaning to form inputs, tables displaying data, and action buttons, which are also accessible and easy to maintain through user interface elements.

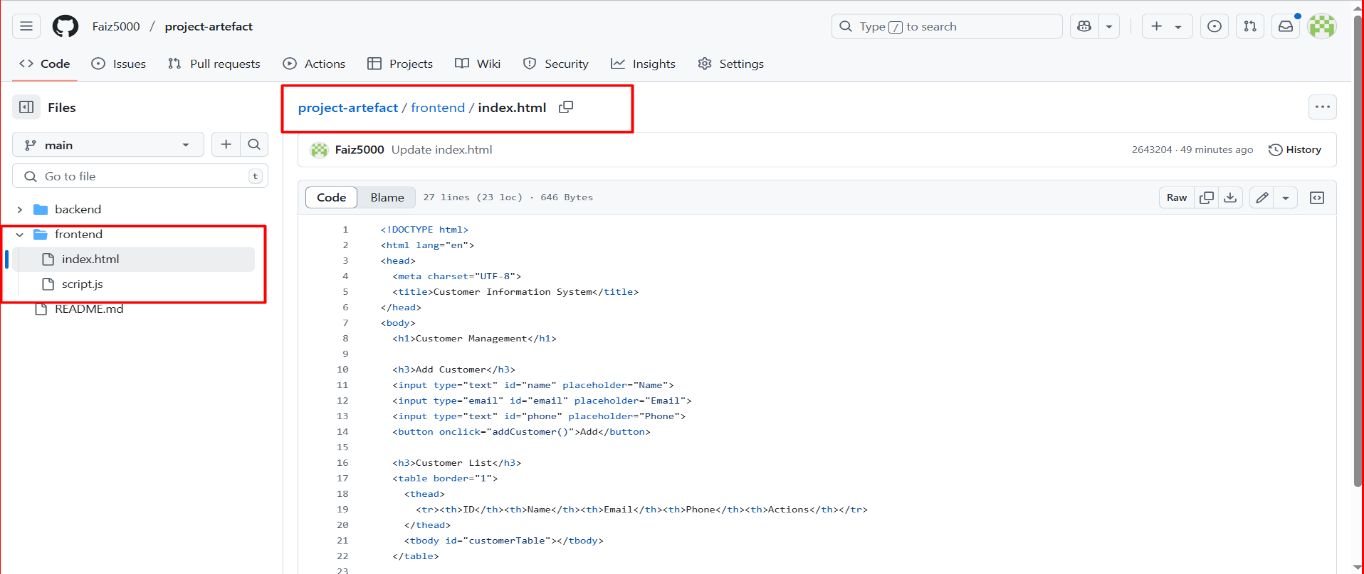
The application layer is designed using the Python Flask framework to generate a well-defined backend API that processes HTTP requests and responses (Flask Development Team, 2023). Flask was chosen due to its simplicity, flexibility, and comprehensive documentation; it was a suitable choice for creating a proof-of-concept. The backend uses the principles of RESTful API and defines specific endpoints for every CRUD action. The core application file, app.py, includes (route) definitions that lead to the creation, retrieval, update, and deletion of customers. Each of the routes is created to address particular HTTP requests (GET, POST, PUT, DELETE) and respond with corresponding JSON objects.



It is based on the data layer implementation with in-memory storage in the form of Python dictionaries and lists, which is sufficient to allow a demonstration yet does not require setting up and configuring a database. This strategy enables one to focus on essential programming principles and API design without the burden of database management. It is set with sample data, so it becomes functional immediately after deployment, featuring a sample customer record with all fields populated with information.

Security concerns and authentication issues are reported to be resolved with the assistance of HTTP Basic Authentication by implementing Flask-HTTPAuth. It consists of a basic user authentication system, where the credentials are hard-coded for use as a showcase. Although this practice is successful in the proof-of-concept stage, production deployments require more advanced forms of authentication, such as multifactor authentication, password hashing, session handling, and the inclusion of specialized authentication providers.

The frontend JavaScript code demonstrates asynchronous communication with a backend API using the modern fetch() API requests. The getCustomers() method retrieves the product data (in this scenario, the customer's data) from the backend and dynamically fills the current customer information into the table. The addCustomer() function retrieves the data provided by users in the form, checks it based on basic prerequisites, and makes POST requests to store new customer entries. These functions display appropriate error handling and user feedback, which are the prerequisites of resilient web applications.

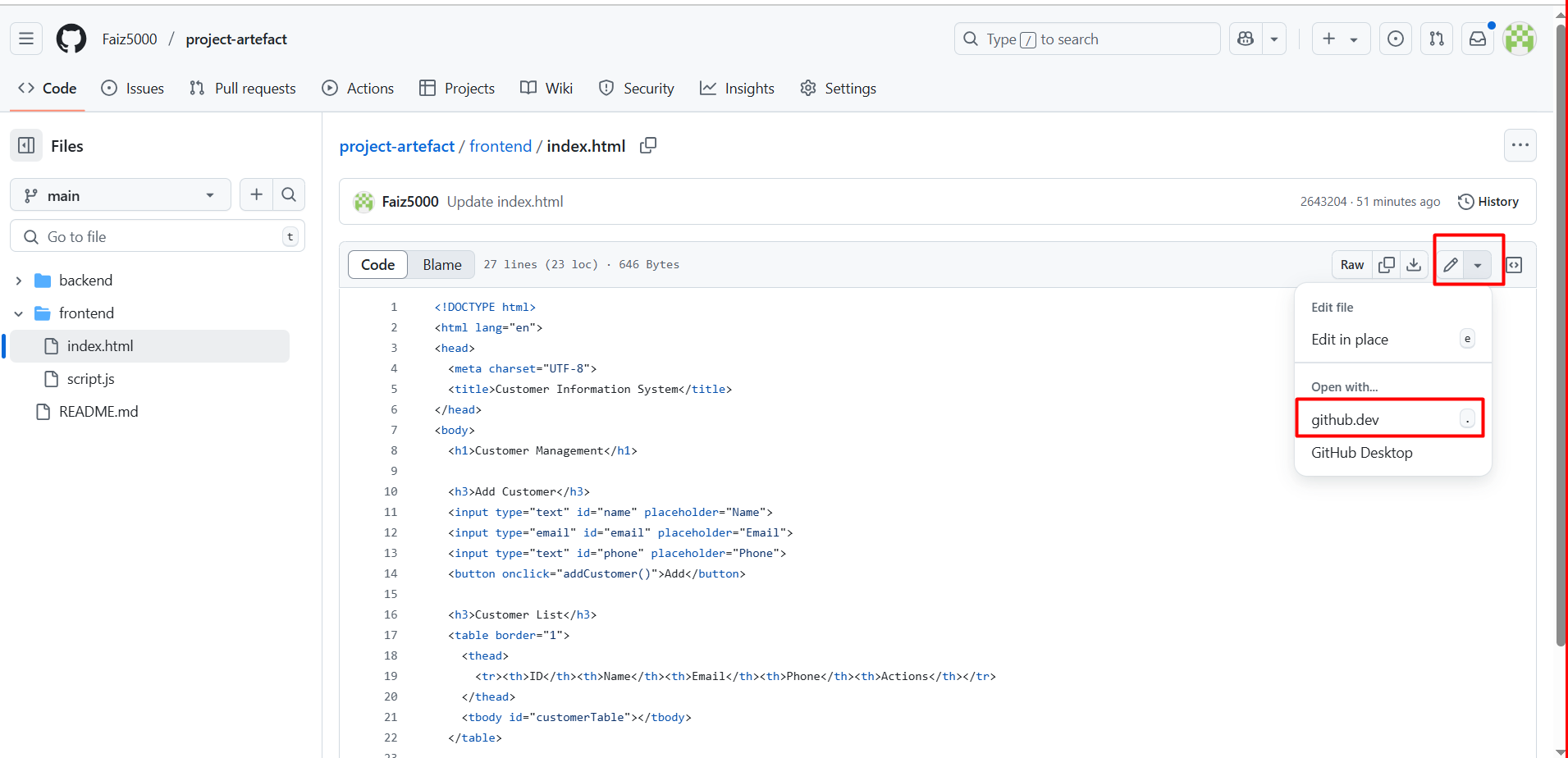


If there is an error in the entire system, including client and server-side validation, the frontend performs data checks on the forms the user enters and prevents submitting requests to the server, thereby ensuring the user receives feedback. The backend gives the right status codes and error messages to inform the user of success or failure. It is a comprehensive method for error processing, ensuring the system is stable and user-friendly and providing clear feedback on the system's performance.

# 3. CRUD Operations and Testing

The basic operations of the Customer Management Information System can be defined as the Create, Read, Update, and Delete (CRUD) actions, all of which were carried out with substantial attention to user privacy and experience. The Create operation allows adding new customer records with the help of a web form interface that retrieves the required information, including the customer's name, email address, and phone number. It also utilizes client-side validation in its implementation to ensure that all required fields are not left blank before sending the data. Therefore, there will be no missing records in the system.

Creating a customer is programmed to run the addCustomer() JavaScript command, which specifies the input data entered in the form. This command then creates and sends a JSON message, as well as a POST request, to Flask, the backend developer in this project. The backend route handler will verify the received information, create a unique key based on the newly entered customer record, and insert it into the in-memory storage code. When such creation is successful, the system will automatically refresh the customer list display to reflect the newly inserted customer record, providing instant visual feedback on the operation's success.



The read operation includes reading a record as well as a list of records in a dataset. The system utilizes a method called getCustomers(), which retrieves all customer records using the backend API and presents them in a well-structured format within an HTML table. The record of each customer displays all the fields related to information, allowing users to view the entire data stored. The table structure allows one to easily identify their data columns and consistently structure the order, making it easier to read.

The Update operation enables the editing of available customer records, specifically for inline editing properties. The system will change the cells of the table that were not editable to a list of input fields when a user presses the "Update" button corresponding to the one to be edited. After that, it is possible to change the customer's data directly. The application retains its original values as placeholders, allowing users to view up-to-date information while still making changes. After the edit is done, the system updates the backend by sending a PUT request that contains the new information and updates the display accordingly.

The delete operation leads to the removal of a customer's records, and the suitable user verification procedures are also secure. The system also deploys a delete button on each customer record, which, upon clicking, performs a DELETE request to the backend API, and the customer identifier to be deleted is passed as part of the request. The backend acknowledges the delete command, deletes the designated piece of information from storage, and returns success in its place. The frontend proceeds to update the screen to remove the deleted record from the user interface.

Testing CRUD operations occurs in unit testing of each separate functionality, as well as in integration testing of the entire workflow for the user. Unit tests ensure that every single CRUD action is tested alone and works with valid and invalid input, as well as edge cases correctly. Integration testing is a comprehensive test that combines the entire user flow, from page loading to the display of various CRUD operations and functionality. This helps ensure that the frontend and backend systems communicate effectively and maintain consistent data as the user continues to interact with the interface.

The testing would involve checking the handling of errors and ensuring that the system responds accordingly to network down failures, incorrect input data, and unexpected responses from the server. In performance testing, one can assess how responsive their system is under normal conditions. In usability testing, one can ensure that the user interface has clear modes of action and is easy to operate across all CRUD processes.

# 4. Development Process and Technical Implementation

The phase involved an iterative development process, starting with the analysis of requirements and system design, proceeding through the implementation stages, and concluding with testing and documentation. The first stage involved organizing the project structure, separating the frontend and backend properly, creating the required directories, and setting up the development environment for Python Flask development and frontend web technologies.

GitHub implemented version control management in the development process, where each commit was regularly checked to record progress during project development and preserve a complete history of code changes. The repository layout adheres to best practices for web application development, separating the backend Python source code, HTML/CSS/JavaScript customer files, and documentation. Every one of the relevant feature additions or modifications was committed independently, which provides us with an extensive audit trail of the development process and makes rolling back easy in case something went wrong during development.

The implementations of the front end exhibit contemporary JavaScript programming practices, including promise-based asynchronous programming, dynamic updates to the DOM to refresh the page after views have been created, and event-driven programming to respond to user interactions. The HTML design adheres to the semantic markup rules, making it not only accessible but also maintainable. The validation of the form involves both client-side JavaScript validation, which allows the user to receive an immediate response, and server-side validation, ensuring the information is considered secure and reliable. The user interface design prioritizes functionality over form, with a focus on presenting information clearly and ensuring ease of use.

Implemented in the backend, Python Flask demonstrates the best features of development, including the correct organization of routes, processing of HTTP methods, and response formatting in JSON format. The authentication system has been demonstrated to incorporate security considerations suitable for the development of a proof of concept, recognizing the need to achieve a higher level of security in a production environment. In-memory storage enables database abstraction with reasonable capabilities, allowing for a demonstration of the idea without relying on external tools.

The deployment follows the pattern of GitHub Codespaces for consistent development environments and GitHub Pages for hosting the frontend, whereas the Flask backend is deployed to provide API functionality. Configuring port forwarding enables smooth communication between the frontend and backend components, which is necessary for development and testing purposes. The system demonstrates how various technologies and platforms were integrated to create a unified web application solution.

The qualities of code include consistency of formatting, extensive commenting, and the sensibility of placing functions and modules. Error handling must cover both expected errors and unforeseen failures by returning helpful feedback to users and recording relevant records in debugging activities. The practical execution demonstrates an understanding of web security concepts, including input validation, output encoding, and secure communication implementation.

Some performance optimization mechanisms include proper DOM manipulation, fewer HTTP requests, and correct start caching of static resources. Future extensions to the system architecture include database connections, more elaborate authentication systems, and other customer management functionalities. The modular design strategy enables maintenance and the distribution of functionality without the need for a complete reorganization of code.

# Conclusion

The Customer Management Information System is a successful example of a fully functional web-based application that incorporates the latest development strategies and technologies. The system takes into consideration all requirements and specifications outlined in the CRUD operations and provides a solid foundation for future improvements and scalability. The project demonstrates the execution of web developing concepts, API designing, and user interface development methods.

The project is a mix of many things, which include, but are not limited to, frontend development using HTML, CSS, and JavaScript, back-end development using Python, Flask, API, and design, and mastering Git and GitHub tools and deployment techniques of web applications. The iterative development model, accompanied by intensive documentation, allows for a closer examination of some of the sophisticated approaches in professional software development solutions and project management processes.

The next version will integrate databases on persistent storage, provide more robust user authentication and authorization, offer customer data analytics and reporting capabilities, and enhance mobile-responsive design. The existing implementation can serve as an excellent basis for these additions and proves to be competent in both web application development and information system design.

# REFERENCES

Flask Development Team. (2023). Flask Documentation: A Python Microframework. Retrieved from  [https://flask.palletsprojects.com/en/stable/](%20https://flask.palletsprojects.com/en/stable/%20)

Grow Solutions. (2024, July 12). Functional and non-functional requirements: The ultimate checklist with examples. *Medium*. https://medium.com/@growsolutions/functional-and-non-functional-requirements-the-ultimate-checklist-with-examples-cde16aba33d7

Mozilla Developer Network. (2023). Web APIs: Fetch API. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API>

GitHub, Inc. (2023). GitHub Codespaces Documentation. Retrieved from <https://docs.github.com/en/codespaces>