XLN4

Athul Krishna

C3006718

**Table of Contents**

[Situation 3](#_Toc157147822)

[Tasks 4](#_Toc157147823)

[Actions 5](#_Toc157147824)

[Result 5](#_Toc157147825)

[Appendices 6](#_Toc157147826)

Situation

The goal of this project is to create a case management system that assists companies in effectively managing questions, concerns, and requests while avoiding duplication of effort. Essential information like a unique ID, timestamp, department, case reason, status (open or closed), and optional notes are all included in every case. To make sure that no two open cases have the same department or reason, the system must look for duplicate cases before permitting the creation of a new one. While closed cases shouldn't limit the creation of new ones, the system should alert the user and prevent duplication if a matching open case already exists.

In addition to functionality, the system needs to be scalable, easy to use, and able to store case data for a long time. Whether data is stored in CSV, JSON, or SQLite, it should be available even after system restarts. When cases are created or marked as duplicates, the user interface ought to give unambiguous feedback. To guarantee dependability, thorough testing is also necessary, especially for the duplicate prevention logic.

MoSCoW :

Must Have:

* Case Creation with unique case details
* Duplicate Case Prevention

Should Have:

* Admin Dashboard
* Filter through department, reasoning and/or status

Could Have:

* Filtering and Searching for cases

Won’t Have:

* AI prediction for solutions

The proper implementation of the duplicate-checking mechanism is the project's top priority. The system as a whole becomes ineffective if this feature is defective, which results in redundant cases and inefficiencies. Ensuring data persistence is equally crucial because the system would become unreliable if case records were not kept. Another important consideration is user experience, since usability may be restricted by an unclear or ineffective interface. In order to guarantee that the system stays effective as the number of cases increases, code maintainability and scalability must also be taken into account.

A number of possible obstacles might affect the project's outcome. The accuracy of duplicate detection is a significant problem, minor differences in case descriptions may result in accidental duplicates or erroneous restrictions. Easily fixable by putting in validation checks and, if necessary, a matching strategy. Data corruption is another issue that can be lessened by selecting a structured database, such as SQLite, and putting backup procedures in place. If the system is unable to manage an increasing volume of cases, performance degradation is another possibility, necessitating effective database indexing and query optimisation. Automated unit tests are crucial for confirming core functionalities because insufficient testing may also fail to detect important bugs.

An organised approach to development is required to reduce these risks. Early planning and execution of the fundamental logic, followed by thorough testing under various conditions, will aid in system improvement.. Git version control and appropriate documentation will help maintainability even more. The project can successfully deliver a strong and efficient case management solution by taking these factors into account.

Tasks

The team carefully divided up the work in order to create the case management system in a short amount of time. While Saniya and I worked with C# and SQLite to handle the backend, Adhi, Jyothish, and Navneet concentrated on frontend development using HTML, CSS, and PHP. GitHub enabled version control, and Trello was utilised to arrange tasks, guaranteeing seamless collaboration.

Saniya and I worked on implementing case creation logic and organising the database on the backend. Preventing duplicate cases was one of our top priorities, which meant maximising query performance for fast lookups without sacrificing responsiveness. Because of its lightweight efficiency, SQLite was selected, and unit tests were created in tandem with implementation to confirm duplicate detection's functionality.

Adhi, Jyothish, and Navneet made up the frontend team, which made sure the interface was easy to use. They developed and constructed case entry forms, put real-time validation in place to stop duplicate submissions, and made a display system to keep track of open cases. To make sure the application worked as intended, it was crucial that the frontend and backend communicated smoothly.

During the sprint, a set schedule was kept in Setting up the database schema and creating the user interface layout took up the first two days. By the middle of the week, the backend was in charge of case creation and validation, while the frontend had a basic structure in place. The last day was devoted to system testing and improvement, making sure duplicate cases were appropriately identified and the application continued to work in various situations.

Major UI changes were postponed to the following phase because styling was not a priority for this sprint. Rather, attention was kept on the system's essential features, making sure it was functional and met customer requirements. The team was able to successfully finish the sprint while pinpointing areas that needed improvement in the future.

Actions

In order to guarantee a clear system design, the team spent two weeks prior to the sprint creating the necessary documentation, such as user stories and ERD diagrams. To lay the groundwork for development, the next two weeks were devoted to configuring the backend and system architecture.

The sprint progress was organised effectively. The team used PHP to create the main user interface (UI) on the first day, emphasising an easy-to-use layout that let users create and manage cases. In order to ensure seamless database transactions and case storage, the second day was devoted to integrating SQLite with C#. By the third day, the frontend and backend were connected, allowing the database and user interface to communicate in real time. In order to avoid duplicate case entries, this step also entailed fixing early data validation problems. On the last day, the system was tested and improved, especially the duplicate detection to guarantee accuracy.

There were a number of obstacles that needed to be overcome quickly during the sprint. Database errors were one of the main problems, but they were fixed by thoroughly debugging and improving query execution. Inconsistencies in duplicate detection were first fixed by improving query logic to prevent the entry of identical cases more than once. Some UI inconsistencies were also found, but because of time constraints, significant styling fixes were put off until the following sprint, giving priority to core functionality.

The team's strong collaboration and well organised task distribution allowed them to maintain consistent progress despite the strict timeline. The system was operational by the end of the sprint, fulfilling the main goals and pointing out areas that needed more work in the following stage of development.

Result

The Case Management System's foundation was successfully established during the first sprint. We created a working backend that prevented duplicate entries and allowed case creation. We made sure there was communication between the two parts, and the frontend offered a simple interface for case management. Although the system satisfied the fundamental need of handling cases, the client review's comments pointed out a number of areas that needed work.

The necessity of improving user roles and access control was one of the review's main conclusions. Admins currently handle customer calls by creating all cases by hand, but efficiency would be increased with a more organised role system. Our main goal for the upcoming sprint will be to implement the recommendation to implement a role management system. This entails giving administrators, user managers, and agents distinct roles. While managers will supervise their team's cases and monitor progress, agents will be in charge of logging cases. Increasing role-based access will improve system organisation and visibility.

Account management is another important area that needs to be improved. The client proposed enabling administrators to create new accounts directly within the system, as accounts are currently created manually through the database. In order to minimise reliance on database-level operations and expedite user onboarding, this feature will be given priority.

Usability and design were also emphasised. Although the frontend works, better layouts and styles are needed to improve the user experience. To guarantee improved navigation and understandable role-based dashboards, we will improve the user interface.

Moving forward, the next sprint will focus on implementing role-based access control, expanding case tracking capabilities, improving account management, and refining the system’s overall design. The lessons from this sprint have emphasized the importance of scalability and structured workflows. By applying these improvements, the system will become more efficient, user-friendly, and aligned with client expectations.

Appendices

A computer screen shot of a computer screen

AI-generated content may be incorrect.A screenshot of a white sheet of paper

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a black screen

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A white paper with black text

AI-generated content may be incorrect.