Student Application - Report

1 Introduction

The Student Application is a software project aimed at facilitating the management of student data, including adding, searching, and updating student information. This report provides an in-depth overview of the development process, highlighting the approach taken, challenges encountered, their resolutions, and adherence to project requirements, including the bonus rules.

2 Approach Taken

2.1 Initial phase

The development of the Student Management Application followed an iterative and incremental approach, ensuring adherence to project requirements. The project was built using Java as the primary programming language and leveraged the JavaFX framework for creating the graphical user interface (GUI).

The initial phase of development involved setting up the project structure and creating the main application window. Maven, a powerful build automation and dependency management tool, was utilized to manage project dependencies, simplify the build process, and enable easy integration with external libraries. Maven's standardized project structure facilitated code organization and ensured consistency.

Git, a distributed version control system, played a vital role in managing the source code. Git allowed for seamless branching, merging, and tracking of changes made. It provided a centralized repository for storing the codebase, ensuring version control and enabling easy rollback to previous versions if necessary.

2.2 Development

2.2.1 Challenges Encountered and Resolutions

Object-Oriented Programming (OOP): The application follows OOP concepts by defining a `Student` class to represent student entities. Each student object includes attributes such as name, student ID, email, and date of birth. This class provides encapsulation and data abstraction, ensuring proper organization and management of student data.

Use of Generics and Collections: To store and manage student records efficiently, Generics are implemented. This allows for flexibility in handling different types of data. Collections, such as ArrayList, are used to manage the list of student records, providing convenient methods for adding, retrieving, and modifying records.

Command-Line Interface (CLI): The application incorporates a command-line interface (CLI) that enables users to interact with the application through text-based commands. The CLI provides a user-friendly and intuitive way to perform operations, such as adding, searching, and updating student records.

Persistence with File I/O: To ensure data persistence, the application supports reading and writing student records to a file. This enables the user to store and retrieve student data even after closing and reopening the application. The usage of the Apache library as a Maven dependency enhances file I/O operations, improving efficiency and reliability.

GUI Design and Layout: One challenge during development was designing an intuitive and visually appealing user interface. Initially, arranging the buttons, table view, and search bar in an aesthetically pleasing manner required several iterations. By utilizing layout containers such as VBox and HBox, the components were organized in a structured manner, resulting in a more user-friendly interface.

Data Filtering: Implementing the filtering functionality based on different search criteria (name, ID, date of birth, email) presented a challenge. By leveraging lambda expressions and Java 8's stream

API, the application efficiently filtered student records based on user input. The filtering logic was implemented in separate functions for each search criterion, enhancing code readability and maintainability.

Error Handling: Ensuring robust error handling mechanisms was crucial for the application's stability. Exceptions (like IOException) were handled using try-catch blocks, allowing graceful error messages to be displayed when invalid inputs or unexpected scenarios occurred. Additionally, logging frameworks such as Java's built-in Logger were employed to capture and track runtime issues, simplifying debugging and troubleshooting.

3 Adherence to Project Requirements and Bonus Rules

Throughout the development process, utmost attention was given to adhering to project requirements. The application successfully met the specified functionality of adding, searching, and updating student information. Furthermore, the bonus rules, such as implementing data filtering or using the Apache library for file I/O operations, were seamlessly integrated into the application.

Throughout the development process, extensive testing was conducted to ensure the application's correctness and reliability. Various scenarios, including different inputs and edge cases, were considered during the testing phase to identify and address any potential issues.

4 Issue Encounter

During the development of the student application, several challenges were encountered. One notable issue arose when configuring the Maven dependencies. Indeed, at first, the application failed to recognize and import the required external libraries, hindering its functionality. To address this issue, I meticulously examined the Maven configuration, ensuring compatibility and resolving any conflicts. By carefully specifying the correct dependencies and resolving version compatibility, I successfully resolved the Maven dependency issue, allowing the application to properly utilize the required libraries.

Another challenge involved working with the class type for student grades. As per the project requirements, it was aimed to implement the use of Generics to store and manage student records efficiently. However, defining a generic class for student grades posed a unique issue. I needed to design a flexible data type capable of accommodating various grade types, such as integers, doubles, or other custom types. Eventually, I implemented a solution using a generic class called `StudentGrades`, which dynamically stored different grade types while maintaining type safety and integrity. This approach enabled the application to handle diverse grade types seamlessly, ensuring the accurate representation of student performance.

Additionally, while testing the application, I encountered a situation where the user entered incorrect data types during input. This triggered exceptions and led to unexpected program behavior. By implementing appropriate try-catch blocks and exception classes, I prevent incorrect user input, providing informative error messages and guiding the user to enter valid data.

Furthermore, the integration of the LOGGER library presented some challenges. LOGGER was employed to log important events in the application, such as adding or removing student records. Configuring the LOGGER library correctly and customizing the logging format to suit the project's needs was a big issue. I had to ensure that the log messages were appropriately formatted and captured relevant information without impacting application performance, and I had to make the LOGGER dependencies work with Maven.

5 Conclusion

In conclusion, the Student Application successfully achieved its objectives by effectively managing student data. Challenges related to GUI design, data filtering, error handling, and user experience were addressed through a combination of solutions and iterative refinements. The utilization of development practices, such as Maven for dependency management and Git for version control, significantly contributed to the project's success. The final product adhered to all project requirements, including the bonus rules, resulting in a good application.