Student App - Design Specification

1 Introduction

The Design Specification outlines the architectural structure and main components of the project, providing a comprehensive understanding of its organization and codebase. This document serves as a guide for developers, stakeholders, and everyone involved in the project. By defining the architecture and highlighting the key code components, this specification sets a solid foundation for the successful development and maintenance of the student app.

2 Architecture

The project follows a modular architecture, designed to promote scalability, maintainability, and separation of concerns. It is in fact the default maven architecture. It consists of the following architectural components:

- **StudentApp:** This is the main application module, encapsulating the core functionality of the student grading system. It contains the main Java source code and resources required for running the application.
- **Dependency Management:** The `pom.xml` file manages the project's dependencies using Maven. It ensures that all necessary libraries and frameworks, such as Apache and JavaFX, are included and resolved correctly.
- **File Management:** The `file` directory within the StudentApp module contains various example files and resources. These files are used for interactions between different parts of the application. Notable files include `BuiltinJean.txt`, `LotsofStudent.txt`, `Report.pdf`, and `UserGuide.pdf`.
- **Source Code:** The `java` directory within the StudentApp module holds the main Java source code files. It includes classes such as `AddStudent`, `App`, `ListGrades`, `Main`, `MaximizedWindow`, `SecondaryWindow`, `StudentGrades`, and `Student`. These classes collectively define the user interface and all aspects of the application.
- **Resource Files:** The `resources` directory within the StudentApp module contains additional resources for the application. Notably, it includes

the `META-INF` directory, which houses the `MANIFEST.MF` file. This file is crucial for executing the application after compilation, as it specifies the main class (`App`) for launching the application.

- **Target Directory:** The `target` directory within the StudentApp module stores the compiled files and build artifacts generated by Maven. It includes the `classes` directory, which contains compiled `.class` files for each Java class. Additionally, it includes the `generated-sources` and `maven-status` directories, which store generated source files and compilation-related metadata respectively.
- **StudentWorklist:** This component represents another module within the project. While not described in detail here, it can be assumed that this module relates to managing and displaying student worklists within the application.

3 Main Code

The main code of the project resides within the Java source files mentioned earlier. These files implement key functionalities of the student grading system:

- **AddStudent:** This class handles the addition of new students to the system. It validates and processes student information, ensuring data integrity.
- **App:** The central class of the application, responsible for managing the user interface, handling user interactions, and coordinating the flow of the application. It utilizes JavaFX for creating windows, displaying data, and managing application states.
- **ListGrades:** This class manages the list of grades for each student. It provides methods to add, retrieve, and manipulate grades within the student grading system.
- **Main:** This class serves as the entry point for launching the application. It initializes the necessary components and starts the application's user interface.
- **MaximizedWindow and SecondaryWindow:** These classes represent different windows or UI components within the application. They handle specific functionalities or user interactions, contributing to the overall user experience.
- **StudentGrades:** This class represents the grades of a student. It leverages the concept of generics to allow for grades of different types. It encapsulates the grade value and provides methods for accessing and modifying it.

- **Student:** This class encapsulates student information, including a unique ID, name, email, and date of birth. It provides methods for managing student details and their associated grades.

The codebase adheres to established coding best practices, including modularization, encapsulation, and proper separation of concerns. It utilizes libraries such as Apache and JavaFX to enhance functionality and provide a rich user experience.

4 Conclusion

In conclusion, the Design Specification provides a detailed overview of the project's architecture and main code components. The modular architecture promotes scalability, maintainability, and separation of concerns, allowing for efficient development and future expansion. The codebase encompasses classes such as AddStudent, App, ListGrades, Main, MaximizedWindow, SecondaryWindow, StudentGrades, and Student, which collectively define the business logic, user interface, and data management aspects of the application.

By adhering to established coding best practices and leveraging libraries such as Apache and JavaFX, the project ensures code reusability, enhances functionality, and provides a seamless user experience. The clear organization of the project's directory structure, with the StudentApp module at its core, facilitates effective code management and promotes efficient collaboration among team members.

Overall, this Design Specification serves as a valuable reference for understanding the project's architecture and code structure, empowering developers to build upon the existing foundation, stakeholders to make informed decisions, and the team to deliver a robust and user-friendly student grading system.