**7COM1025 COURSE WORK REPORT**

**1.0 INTRODUCTION**

This project report aims to provide details of the design and implementation of the Hatfield Junior Swimming School application developed to manage the school's booking system, learner onboarding, coach, and learner reporting.

**1.1 ASSUMPTIONS**

In the development of the Hatfield Junior Swimming School (HJSS) management software, certain assumptions were made to clarify system requirements and guide the design and implementation process. These assumptions are outlined below:

1. **Pre-registered Coaches and Some Learners**: The system assumes that a portion of the coaches and learners are already pre-registered within the system. This implies the existence of a pre-existing database containing coach and learner information, alleviating the need for the software to handle the registration of all coaches and learners from scratch. Fifteen (15) learners and 5 coaches were pre-registered on the application bootstrap.

2. **Timetable Design**: The final deliverable of the software includes a timetable spanning at least 4 weeks (44 lessons), covering all grades and coaches. It is assumed that the timetable design can either repeat weekly or vary from week to week, depending on the design preference of the developer. In the implementation of this application, the timetable was designed to repeat week to week.

4. **No External Database**: The system does not require integration with an external database for storage purposes. Instead, all necessary data can be managed internally within the program. Reports generated by the system can be printed to the console (System.out) or output to a suitable graphical user interface (GUI) without reliance on external databases. No Database was used in the implementation of this application as the array list was temporarily used to hold data in memory during code execution.

5. **Security Protocol Not Needed**: As specified in the requirements, the system does not necessitate the implementation of any security protocols. Therefore, user authentication and authorization features are not required for accessing the system's functionalities.

6. **Cancellation and Booking Rules**: The software enforces rules allowing learners to change or cancel their bookings, provided there are available spaces in the newly selected lesson. It is assumed that the system performs checks for lesson availability before allowing changes or cancellations to be processed.

7. **Age Range for Learners**: Learners must fall within the age range of 4 to 11 years to participate in swimming lessons. The system is assumed to enforce this age restriction when adding new learners to ensure compliance with the specified age criteria.

8. **Grade-Level Restrictions**: Grade 1 learners are permitted to book Grade 1 or Grade 2 lessons but are restricted from booking Grade 3 to Grade 5 lessons. Similarly, learners are only allowed to attend lessons corresponding to their current grade level or one level higher. The system is expected to validate and enforce these grade-level restrictions during the booking and attendance processes to maintain lesson integrity and adherence to learner capabilities.

**2.0 DESIGN, STRUCTURE AND ARCHITECTURE**

The Hatfield Junior Swimming School application was developed using the Java programming language, leveraging its robust features to establish a solid foundation for the software.

Built upon the fundamental principles of Object-Oriented Programming (OOP), such as encapsulation of class properties, inheritance, and abstraction of objects, along with method polymorphism, the application embodies a structured and organized approach to software design.

Accessible through a Command Line Interface (CLI), the application seamlessly integrates with users' interaction channels. To ensure efficiency and maintainability, the software adheres to the Model-View-Controller (MVC) design pattern. While reminiscent of renowned web frameworks like PHP's Laravel and C#'s .NET, the MVC architecture has been customized to meet the unique requirements of the swimming school management system, guaranteeing optimal performance and user experience.

In this architecture, model classes serve as the direct representations of real-life entities, including learners, lessons, and bookings. Controller classes oversee the business logic associated with their corresponding model counterparts, orchestrating data manipulation and processing. On the other hand, view classes act as intermediaries between users and the system, facilitating input/output operations and presenting information in a comprehensible manner.

**2.1 APPLICATION BOOTSTRAP**

At the heart of the Hatfield Junior Swimming School (HJSS) application lies the **Main** class within the root of the HJSS package. This class features a main method, serving as the entry point for the application. Within this method, an instance of the **APP** class is instantiated, and its **start** method is invoked.

The **APP** class encapsulates the entirety of the Hatfield Junior Swimming School, representing it as an object within the software. This design facilitates scalability and flexibility, allowing for the creation and execution of multiple instances of the school within the **Main** class. This approach was implemented with an emphasis on code reusability, ensuring that the application architecture remains adaptable to various scenarios, such as franchise expansion.

**2.1.1 APPLICATION INSTANTIATION**

Upon instantiation of the **App** class, the constructor method is invoked. Within this method, the application's controllers, representing distinct functional areas, are instantiated, and initialized with empty lists to manage their respective data entities.

Student, Coach, Lesson, Booking, and Review Controllers: These controllers are responsible for managing the application's core entities: students, coaches, lessons, bookings, and reviews, respectively. Each controller is initialized with an empty list to store instances of its corresponding data entity.

Additionally, the controllers responsible for managing lessons and coaches pre-create lessons and coaches during the application's bootstrap phase. This ensures that essential data is available for scheduling and assignment during runtime.

Once the controllers are instantiated and initialized, they are injected into the **AppController**. The **AppController** acts as the main orchestrator, coordinating interactions between different components of the application.

**2.1.2 CONTROLLER INTEGRATION**

The AppController is instantiated with references to all the initialized controllers, allowing it to access and coordinate their functionalities seamlessly then the start() method of the **App** class is invoked to initiate the application's execution. This method delegates control to the **AppController**, which, in turn, displays the main menu interface to the user.

The **start** method enters a loop, continuously displaying the main menu interface and handling user input until the user manually exits the application. This iterative process ensures a user-friendly and interactive experience for learners interacting with the HJSS application.

In summary, the **App** class encapsulates the initialization of essential components, including controllers and their associated data structures, and orchestrates the execution flow of the HJSS application.

**2.2 MODEL CLASSES**

The core of the Hatfield Junior Swimming School (HJSS) application lies in its model classes, which represent the fundamental entities and data structures manipulated by the system. These model classes encapsulate key properties and behaviors of various real-world entities within the swimming school environment.

**2.2.1 STUDENT CLASS**

The **Student** class represents individual learners enrolled in the swimming school, storing essential information such as their name, gender, age, emergency contact phone number, and current grade level. Additionally, the **Student** class ensures that a learner's age falls within the acceptable range of 4 to 11 years, adhering to the specified constraints of the system.

**2.2.2 COACH CLASS**

Similarly, the **Coach** class models swimming instructors employed by the HJSS, capturing their names as an attribute along with their ID. Coaches play a pivotal role in conducting swimming lessons across all grade levels, and their availability is crucial for effective lesson management.

**2.2.3 LESSON CLASS**

The **Lesson** class represents individual swimming sessions conducted by the school, denoting details such as the grade level, time slot, day, and assigned coach. Lessons are structured to accommodate up to four learners per session and are available at specific time slots on designated days of the week.

**2.2.4 BOOKING CLASS**

Bookings are instances of learners reserving slots for specific swimming lessons. The **Booking** class tracks details such as the learner associated with the booking, the lesson they have reserved, and the status of the booking (e.g., booked, canceled, attended). Learners can make multiple bookings but are restricted from making duplicate bookings or exceeding lesson capacity.

**2.2.5 REVIEW CLASS**

Upon attending a swimming lesson, learners can provide feedback through the **Review** class. Reviews include a numerical rating ranging from 1 to 5, reflecting the learner's satisfaction level with the lesson. These reviews contribute to assessing the quality of lessons and instructors, facilitating continuous improvement within the swimming school.

**2.2.6 IMPLEMENTATION APPROACH**

Each model class is designed to encapsulate its respective entity's attributes and behaviors, adhering to object-oriented programming principles such as encapsulation, inheritance, and abstraction. Additionally, the classes enforce data integrity and adhere to system constraints, ensuring consistency and reliability in data manipulation and processing.

In summary, the model classes form the foundation of the HJSS application, providing a structured representation of essential entities within the swimming school ecosystem. Through the design and implementation, these classes enable efficient management of learners, coaches, lessons, bookings, and feedback, facilitating a seamless and enriching experience for all stakeholders involved.

**3.0 CONTROLLER CLASSES**

The Hatfield Junior Swimming School application relies on a series of controller classes to manage the core business logic, interact with the user interface, and manipulate data within the system. These controllers serve as the backbone of the application, orchestrating various operations and ensuring a cohesive user experience.

**3.1 APP CONTROLLER**

The **AppController** serves as the main controller of the application, managing the flow of operations and user interactions. It coordinates actions such as logging in users, booking swimming lessons, changing or canceling bookings, attending lessons, and generating reports. Let's delve deeper into its functionality:

**3.1.1 RESPONSIBILITIES**

* **Dependency Injection**: The **AppController** constructor accepts instances of other controller classes, including **StudentController**, **LessonController**, **CoachController**, **BookingController**, and **ReviewController**, enabling seamless integration and collaboration between different components of the application.
* **Menu Navigation**: The **showMainMenu** method displays the main menu of the application, allowing users to choose from various options such as logging in, booking lessons, viewing reports, or exiting the application. Based on user input, it invokes corresponding methods to handle specific actions.
* **User Authentication**: The **handleLogInUser** method facilitates user authentication, prompting users to log in and retrieve their profile information. If a user opts to register as a new student, it invokes the registration process through the **handleRegisterStudent** method.
* **Lesson Booking**: Methods such as **handleBookSwimmingLesson**, **handleBookByDay**, **handleBookByCoach**, and **handleBookByGrade** facilitate the booking of swimming lessons. Users can select their preferred booking method, view available lessons, and make bookings accordingly.
* **Booking Management**: The **handleChangeOrCancelBooking** method allows users to change or cancel existing bookings. It retrieves the user's bookings, displays them, and provides options to modify or cancel bookings based on user input.
* **Lesson Attendance and Review**: Users can attend lessons and provide reviews through the **handleAttendALesson** method. It prompts users to select a booking, attend the lesson, and submit a review, enhancing the feedback mechanism within the application.
* **Report Generation**: The **handleShowLearnerReport** and **handleShowCoachReport** methods generate detailed reports for learners and coaches, respectively. These reports provide insights into learners' booking history, attendance, and reviews, as well as coaches' performance based on average ratings.

Overall, the **AppController** orchestrates the diverse functionalities of the HJSS application, ensuring smooth navigation, efficient data management, and comprehensive reporting capabilities. Through its robust design and functionality, it contributes to the seamless operation and management of the swimming school system.

**3.2 COACH CONTROLLER**

The Coach Controller in the Hatfield Junior Swimming School application is responsible for managing operations related to coaches.

**3.2.1 RESPONSIBILITIES**

- **Coach Management**: The `CoachController` maintains a list of coach objects and provides methods to manage them, including creation and retrieval.

- **Coach Creation**: It offers a method **createCoaches** to populate the list of coaches with predefined coach objects. This method initializes the coach list with a set of predefined coaches, each identified by a unique ID and name.

- **Coach Retrieval**: The controller allows users to retrieve information about coaches by providing methods such as **getCoaches** to fetch the entire list of coaches and **getCoach** to retrieve a specific coach based on their ID.

By encapsulating coach-related functionalities within the **CoachController**, the application achieves modular design and separation of concerns. This controller ensures efficient management of coach data and seamless interaction with other components of the system.

**3.3 STUDENT CONTROLLER**

The Student Controller in the Hatfield Junior Swimming School application oversees operations related to student management.

**3.3.1 RESPONSIBILITIES**

- **Student Management**: The **StudentController** is responsible for managing student-related operations, including registration, login, and retrieval of student information.

- **Student Registration**: It facilitates the registration process for new students by collecting essential details such as name, age, gender, grade, and emergency contact number. Upon successful registration, a new student object is created and added to the list of students.

- **Student Login**: The controller allows existing students to log in by displaying a login menu and prompting them to select their profile. If the user chooses to exit or register a new student, the method returns null.

- **Predefined Students**: It provides a method **createStudents** to initialize the list of students with a predefined set of student objects. This method ensures that the application starts with a populated list of students for demonstration purposes.

By encapsulating student-related functionalities within the **StudentController**, the application achieves modular design and separation of concerns. This controller ensures efficient management of student data and seamless interaction with other components of the system.

**3.4 STUDENT CONTROLLER**

The Lesson Controller in the Hatfield Junior Swimming School application is responsible for managing lessons.

**3.4.1 RESPONSIBILITIES**

* **Lesson Management**: The **LessonController** handles operations related to lessons, including retrieval, creation, and generation of timetables.
* **Lesson Retrieval**:
  + **By ID**: It retrieves a lesson based on its ID.
  + **By Day**: It retrieves lessons scheduled for a specific day.
  + **By Coach**: It retrieves lessons coached by a particular coach.
  + **By Grade**: It retrieves lessons for a specific grade.
* **Lesson Creation**: The controller pre-creates lessons for a specified duration, generating a total of 44 lessons over a 4-week period. It assigns grades, days, times, and coaches to each lesson randomly, ensuring a diverse schedule.
* **Timetable Generation**: It generates a formatted timetable string based on the provided list of lessons. Lessons are grouped by weeks, with each week containing a maximum of 11 lessons.

By encapsulating lesson-related functionalities within the **LessonController**, the application maintains a clear separation of concerns and facilitates efficient management of lesson data. This controller ensures that lessons are organized, accessible, and represented effectively within the application.

**3.5 BOOKING CONTROLLER**

The Booking Controller in the Hatfield Junior Swimming School application is responsible for managing booking-related operations.

**3.5.1 RESPONSIBILITIES**

* **Booking Management**: The **BookingController** oversees various operations related to bookings, including creation, retrieval, modification, and cancellation.
* **Booking Validation**: It ensures that booking requests meet certain criteria before processing, such as matching the student's grade level with the lesson's grade requirement and checking lesson availability.
* **Attendance Tracking**: The controller tracks attendance for bookings and provides methods to mark bookings as attended.
* **Exception Handling**: It handles exceptions that may occur during booking operations, such as not matching grade, maximum lesson capacity reached, duplicate bookings, forbidden actions, and attended bookings.
* **Retrieving Filtered Bookings**: The controller allows retrieving lists of bookings based on different criteria, such as canceled bookings and attended bookings for a specific student.

By centralizing booking-related operations within the **BookingController**, the application maintains consistency and ensures that bookings are managed efficiently while adhering to business rules and constraints. Additionally, the controller provides mechanisms for error handling and user interaction, enhancing the overall user experience.

**3.6 REVIEW CONTROLLER**

The Review Controller in the Hatfield Junior Swimming School application manages review-related operations, allowing users to provide feedback and ratings for lessons they have attended.

**3.6.1 RESPONSIBILITIES**

* **Review Creation**: The **ReviewController** facilitates the creation of new reviews for lessons attended by prompting users to provide ratings and feedback.
* **Rating and Feedback Collection**: It requests users to input ratings and feedback for the lessons attended, ensuring an interactive and user-friendly experience.
* **Review Retrieval**: The controller retrieves all reviews associated with a specified coach, enabling the coach to view feedback provided by students.

**4.0 VIEW CLASSES**

All view classes are inherited from the abstract class View. This abstract class provides a foundation for implementing various views within the application. By inheriting from View, each specific view class gains access to the common functionality defined in the abstract class, such as menu display and input retrieval. Subclasses then implement their specific behavior by overriding the abstract methods, allowing for customization of menu displays and input handling tailored to the needs of each view.

By utilizing inheritance and abstraction, the View class provides a flexible and reusable framework for building views across the swimming school application. It promotes code reusability, simplifies maintenance, and ensures consistency in user interaction throughout the application.

**4.1 RESPONSIBILITIES**

The View class serves as a blueprint for creating different views within the swimming school application. Views are responsible for displaying menus and retrieving user input.

* **Constructor**: It has a constructor that initializes a Scanner object for reading input from the user and sets the length of the menu options.
* **Abstract Methods**:
  + **displayMenu**: Abstract method responsible for displaying the menu options. Each subclass must implement this method to provide its specific menu display.
  + **getMenuChoice**: Abstract method for retrieving the user's menu choice. Subclasses implement this method to customize how the user's input is obtained and validated.
* **Concrete Methods:**
  + **isValidMenuChoice**: Checks if the provided menu choice is valid based on the length of the menu options.
  + **padToTwoDigits**: Helper method that pads a given number to two digits. It's used for formatting purposes in displaying menu options or other numerical values.
* **Error Handling**: The **getMenuChoice** method handles input mismatch exceptions and invalid menu choices gracefully by providing error messages and prompting the user to enter valid input.

**5.0 VIEWS SNAPSHOTS**

**A screenshot of a computer

Description automatically generated**

Figure 1: MainView Class

A black rectangular object with a black border

Description automatically generated

Figure 2: LoginStudentView class.

A black rectangular object with a black border

Description automatically generated

Figure 3: Invalid Menu choice error message.

A black rectangle with white text

Description automatically generated

Figure 4: BookLessonView class.

A screen shot of a computer

Description automatically generated

Figure 5: TimeTableView class.

A screenshot of a computer

Description automatically generated

Figure 6: Invalid selection id

A screenshot of a computer program

Description automatically generated

Figure 7: Grade Mismatch

A black rectangle with white text

Description automatically generated

Figure 8: Successful booking.

A screenshot of a computer

Description automatically generated

Figure 9: ChangeCancelView class.

A screenshot of a computer

Description automatically generated

Figure 10: Attend Lesson

A black screen with green text

Description automatically generated

Figure 11: Review

A screenshot of a computer

Description automatically generated

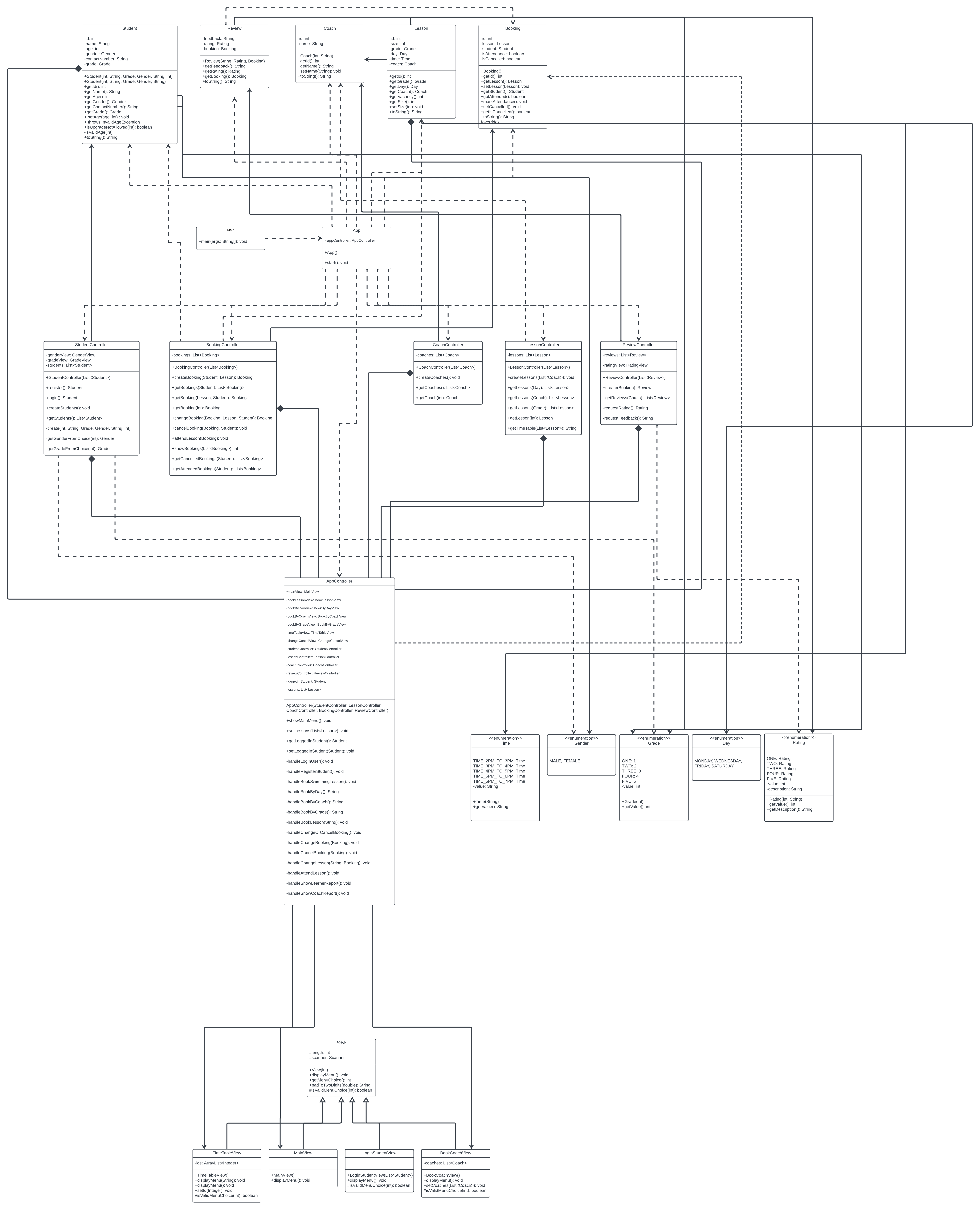
Figure 12: Monthly Learner Report

A black rectangle with white dots

Description automatically generated

Figure 13: Monthly Coach Report

**6.0 UML CLASS DIAGRAM**

****

**7.0 RUNNING AS JAR APPLICATION**

To launch the HJSS application, execute the following command:

**Java -jar hjss.jar**

**8.0 GIT COMMITS**

Below are the commit messages and corresponding actions taken throughout the development of HJSS. These messages provide insights into the development process, decision-making, and the evolution of the system's business logic.

The Git repository is available at [repository\_link].