Table of Contents

1. **Software Architecture** 1

   1.1 Overview 1

   1.2 Software Architecture Diagram 2

   1.3 Design Principles fulfilled by Software Architecture Design 3

   1.4 Consideration of Alternative Architectures 4

   1.5 Architecture Evaluation Process 4

   1.6 Final Rationale behind Selection of Candidate Architecture 6

   1.7 Persistent Data Management 7

   1.8 Boundary Conditions 8

1. Software Architecture

1.1 Overview

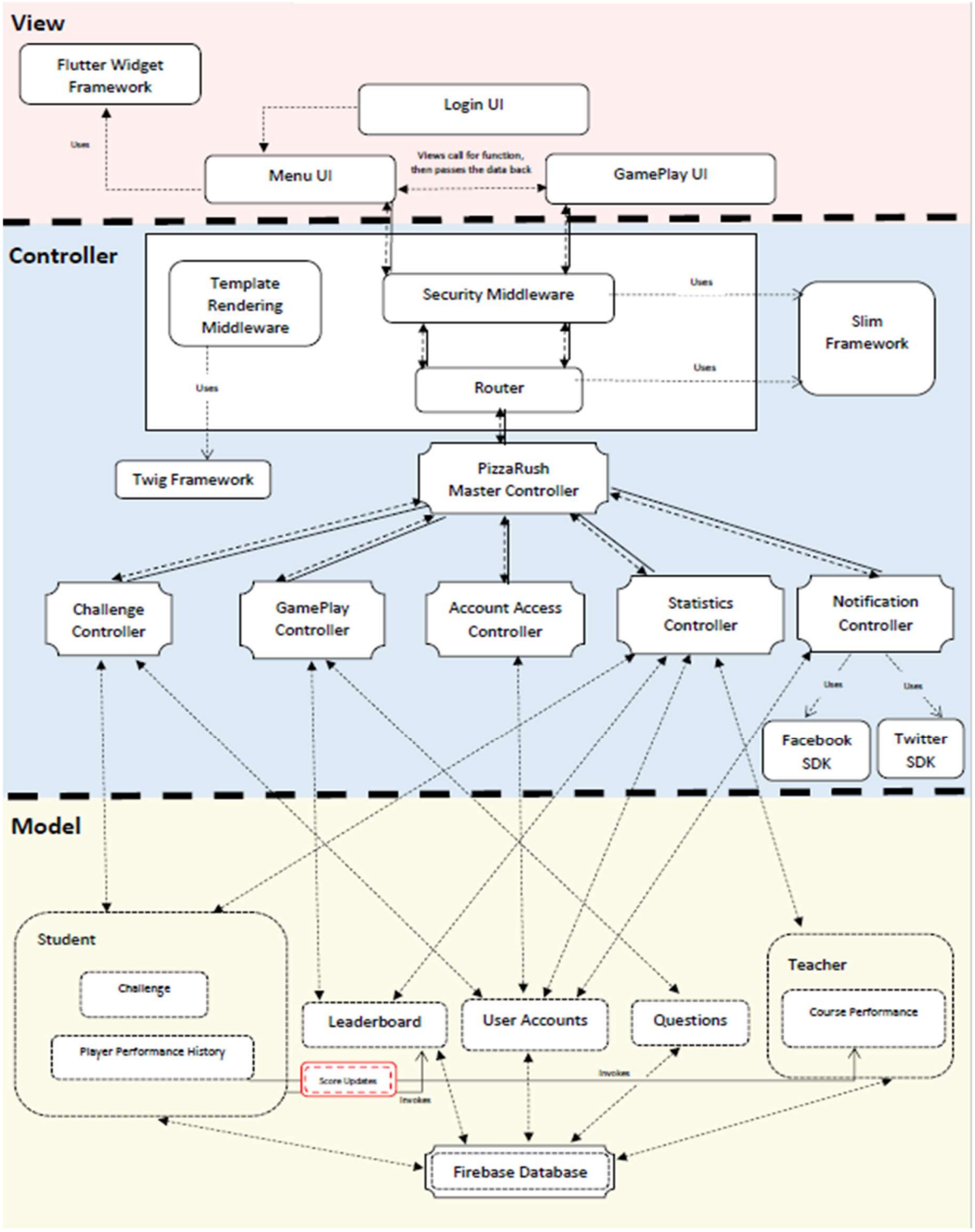
The software architecture design of the 'PizzaRush' mobile application follows a hierarchically and locationally heterogeneous model. The overall architecture locationally follows 'Model-View-Controller', 'Abstract Data Type' and ' Implicit Invocation' . However, hierarchically its subsystem design adheres to a ' Main Program and Subroutine' architectural style.

The core requirements of the application were prioritized while finalizing on the software architecture design to be adopted for our application. Modularity of the program was the highest priority, which is why *Model-View-Controller* (or MVC) heavily features in the system architecture. Moreover, change of data representation and reusability were also very important, which led to the choice of *Implicit Invocation* and *Abstract Data Type*. Changes in system usage and functions were factors of lower assigned importance.
Each of these architectural styles were effectively combined to best design the system at hand, in a heterogeneous manner.

The MVC architecture seamlessly facilitates high cohesion and low coupling in the design, by separating the primary modules - into Model, View and Controller.

Additionally the Abstract Data Type approach enables separation of different segments of the Model class, to account for selective access to subroutine calls, which here are the Controller classes. This design approach enables the division between access to student and teacher data, and the common data (Leaderboard, User Accounts, Questions).
The causality between different modules of the architecture is shown by employing Implicit Invocation. As Player Performance History is updated, the corresponding scores are also reflected on the Leaderboard and Teacher end Course Performance.
Hence, this heterogeneous mix of architectural styles effectively conveys the application's static architecture and portrays its dynamic behaviour.

1.2 Software Architecture Diagram



1.3 Design Principles fulfilled by Software Architecture Design

The chosen candidate architecture aims to effectively fulfil the functional and non-functional design requirements and principles envisioned for this application.

Select examples that showcase the achievement of the above goal are as follows:

1. **Performance**
   Abstract data type and Main Program & Subroutine architectures lend a boost to performance, hence fulfilling a core requirement of the application.
2. **Scalability**
   Adding a new type of views is very easy in MVC pattern as Model part does not depend on the views part. So, any changes in the Model will never affect the entire architecture. This ensures the number of stages in the game can be expanded as well as extended to other educational topics. Hence this ensures easier updates to be achieved.
3. **Security**
   The candidate architecture caters for discrete data models, thus introducing selective access to data by subroutines. This design choice yields higher security for the data of the application.
4. **Maintainability**
   Easier to maintain and debug as the application is structured to be divided into discrete segments.
5. **Modularity**
   Effective modularisation achieved by employing the Model-View-Controller architecture.
6. **Reusability**
   Lesser code duplication since data and business logic is seperate from the display ensuring higher performance.

SOLID Design Principles fulfilled by the Candidate Architecture:

1. **Open Close Principle**
   The different functionalities available in the application are closed for modification but are open for extension. For instance, using external modules the game can be further extended to provide more difficulty levels for questions, or to add different topics, or even other subjects apart from Mathematics. However, these functionalities are closed for modification from external modules altogether.

2. **Interface Segregation Principle**
   There is a clear distinction and separation of interfaces for the two primary users of the

application - namely, the Students and Teachers, instead of having just a single general-purpose interface for the two groups. The interfaces designed to cater to the various relevant functionalities for Students will not be accessible to the Teachers and vice-versa.

1.4 Consideration of Alternative Architectures

Following at the alternative architecture designs considered during the candidate architecture selection process:

- Layered pattern architecture
    - This pattern can be used to structure programs that can be decomposed into groups of subtasks, each of which is at a particular level of abstraction. Each layer provides services to the next higher layer.
- Pipe and Filter architecture
    - The Pipe and Filter is an architectural design pattern that allows for stream/asynchronous processing. In this pattern, there are many components, which are referred to as filters, and connectors between the filters that are called pipes.
- Batch sequential architecture
    - This data processing model entails a data transformation subsystem can initiate its process only after its previous subsystem is completely through the flow of data, and carries a batch of data as a whole from one subsystem to another.

1.5 Architecture Evaluation Process

The goal of the architecture evaluation process is to assess the advantages and drawbacks of chosen architecture styles which identify the suitable architecture style that fulfill the quality and functional requirements of the application.

ARCHITECTURE	ADVANTAGES	DRAWBACKS

Layered Pattern	● Work can be done parallelly with minimal dependencies ● Different parts of the application can be maintained, deployed and updated independently ● Able to control the accessibility of components to different stakeholders	● Does not allow coupling. ● No dependency inversion ● Adds complexity to the application ● Increases the runtime for accessing the application
Implicit Invocation	● Loose coupling and minimal dependency allow modification such as add and delete ● Component can be replace by new component without changing the interface	● Unable to share data with other components ● Have no control over the events ● Unable to determine which method to be called
Pipe and Filter	● Able to work on different components concurrently and parallelly ● Able to reuse the filter and combine with new filters	● Add complexity to the application ● Negative impacts on application if filter crashes ● Not recommend for more interaction between filters except for input/output

Model-View-Controller	● Allow multiple developers to work together ● Easy to modify the application ● Able to conduct test of components independently ● Able to apply different security level to different components	● Required more layers lead to negative impact on performance ● Increase complexity in the design
Main Program and Subroutine (Call & Return)	● Provides flexibility to interrupt and query the program ● Able to do modification during runtime ● Subroutine can be replaced easily without affecting the interface	● Negative impact on performance due to interrupts ● Possibility of have two subroutine doing the same task ● It is not scalable for big system
Batch Sequential	● Each subsystem is independent program to perform input/output data ● Easy to change behavior of a component ● Able to solve complex tasks by dividing it into several tasks	● High latency and low throughput ● Unable to work on different components concurrently and parallely ● No interaction between subsystems except for data passing

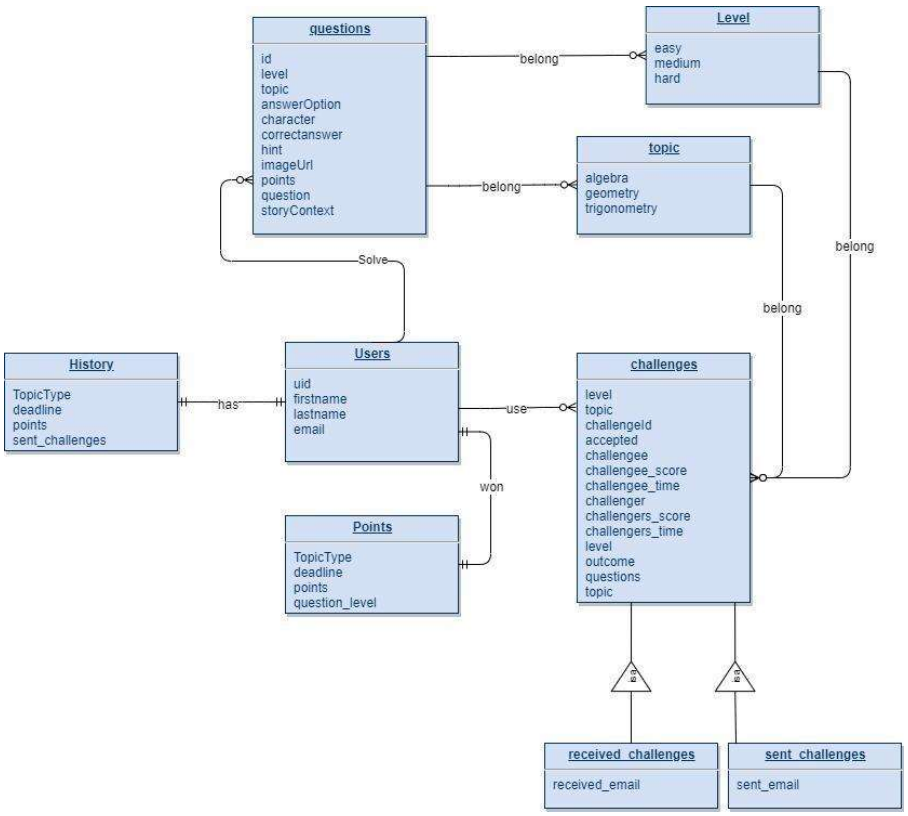
1.6 Final Rationale behind Selection of Candidate Architecture

Due to conflicting requirements and consequent performance trade offs, the core requirements of the application were prioritized to facilitate the decision making process.

Modularity of program was the highest priority, which is why *Model-View-Controller* heavily features in the system architecture. Moreover, change of data representation and reusability were also very important, which led to the choice of *Implicit Invocation* and *Abstract Data Type*. Changes in system usage and functions were factors of lower assigned importance.
Each of these architectural styles were effectively combined to best design the system at hand, in a heterogeneous manner.

1.7 Persistent Data Management

In this project, the persistent data will be stored in Firebase. Firebase is a realtime database that stores and syncs data with their cloud database. The data will remain available even if the app is offline. The data to store in Firebase for this project consists of login credentials, student and course progress data. The Firebase also provides authentication for account validation when a user performs a sign in action. The data structure, in the form of collections and documents, for this application may be observed in the diagram below.



Based on the Firebase database used in this project, 9 collections of data are created to support the persistent data management for the application. They are questions, history, users, points, received challenges, sent challenges, challenges, level and topic. Each collection contains its down corresponding documents, the details of which may be observed from the above data structure diagram. The history of the past activities in the game and points gained are stored for users to view their performance in the game. The received(child) and sent challenges(child) are stored with challenges(parent) under user's account for references. The challenges are categorized based on the topic and level of difficulty in the game. The questions solved by the user from different topics and difficulty levels are stored with a unique id for references.

1.8 Boundary Conditions

Startup

It is an initialization that changes from non-initialized state to steady state.

The user will launch the application on their mobile phone. After the launching, the app should display the Login interface of the *Pizza Rush*, the user will not access any data by the time. The Login interface will prompt the user to login to the app through registered account or Facebook account. These will happen at the login subsystem.

Shutdown

The resources are cleaned up and the system is notified upon termination.

A single subsystem is allowed to terminate the app. Other subsystems will be notified by the termination action. The data will be saved in real time and available even when the app is offline.

Error

Possible errors could occur in the system internally and externally. Some errors can be bugs and power supply shortage. The system should forecast the fatal errors. The system will provide error messages instantaneously when there are errors in the app. The system will provide solution options in the messages such as Cancel, Back and Ignored. The system state will change accordingly based on the user's option to the errors.