ASSIGNMENT OF TASKS

Andy Tran	Came up with and conceptualized the project idea. Researched and decided upon the programming language and mobile development framework (Dart and Flutter). Delegated tasks to other team members. Set up Firebase for its realtime database to store useful account information. Implemented a functional sign-in system for the app using Firebase. Delivered all presentations and wrote all reports.
Henry Tu	Implemented cue card creation and a robust filesystem (comprised of Decks, Folders, and a Root directory) in which to organize cue cards which can be managed through various hand gestures. iCuecards are highly customizable and allow for different colours, can be rearranged within a deck, edited, and deleted. The viewing experience of cue cards is done with intuitive hand gestures. Going forward, will endeavor to implement a search function and the ability to insert photos into iCueCards.
Shuqi Yang	In charge of the home screen from the gaming portion of our application in which users can select either Kahoot! or Heads Up! Implemented Kahoot! Currently has a start button to begin playing, randomizes questions and multiple choice options, and has a results page with a detailed summary. Also implemented a navigation bar at the bottom of the screen. For the future, will include background music to complement the Kahoot! game.
Yumeng Zhou	Implemented a Heads Up game. It will enable players to choose the category which is based on the folder of the cue card and ask players for the question number. During the game, it will generate all the wrong questions in a list and pass it to the review page.The Result page will navigate to Mistakes Review page. In the Mistake Review Page. Each wrong guess will be made into a flip card. It also enables the player to swipe left/right the cards so that players have the option to review forward/backward. To be added later: For the testing purpose, the device titling will be implemented later. There will be a timer for each round. Improve the UI

PROJECT ARCHITECTURE

Functionality Overview

youtu.be/3s-X0LlOKDM

(1) **User Authentication**. Students who use the application are able to register a personal, password-protected account using an email address. This has been accomplished by leveraging Firebase Authentication backend services to manage user registration, user login and authentication.

(2) **Cue Card Customization**.
   I. **Cue Card Colour**. Students are able to customize the colour of their cue cards to put emphasis on particular key concepts discussed in their classes by selecting a colour from the pallet icon situated at the bottom of the Cue Card Creation/Editing page.
   II. **Insert photos**. [Has yet to be implemented] Students will be able to append photos of diagrams or charts to the backside of their cue cards when text alone is not enough to convey information.

(3) **Cue Card Organization**. Cue cards can be stored into hierarchical directories reminiscent of how files are stored on a desktop PC or laptop. The hierarchy is as follows: [Cue Card < Deck < Folder < Root Directory]. A collection of cue cards that pertain to a particular section or chapter of a course forms a Deck of cue cards. Multiple Decks can be organized together into a Folder representing the course the student is taking. Upon logging in, the student will be welcomed to a root directory in which all of their course folders can be viewed.

(4) **User-friendly Cue Card Browsing**. Students can review the cue cards they create in one of two views: a traditional view and a list view.
   I. **Traditional View**. Also known as 'practice mode', the cue cards are displayed as a stack of 3D cards with the frontside of the cue card facing the user. Tapping the cue card reveals the backside of the cue card. Swiping towards the left indicates that the user has mastered the contents of this cue card and it is temporarily discarded. Swiping towards the right will push the cue card to the back of the stack so that it can be reviewed again during the browsing session.
   II. **List View**. The list view is the default view of ICUE CARDS and provides a more comprehensive view of cue cards in a Student's deck. The cue cards are displayed vertically such that Students can easily scroll through them. Both the front and backsides of the cue card are displayed (only the first ~20 characters of the backside is shown so that more cue cards can be displayed on the mobile screen at once). It is possible to rearrange cue cards in this list view by dragging cue cards to the desired location.

(5) **Two Game Modes**. Our version of Heads Up! and Kahoot! provide an alternative way to study that is fun and engaging compared to typical study prep.

System Design representation as a UML Class diagram

The system design of ICUE CARDS strictly follows an Object-Oriented Design. The UML class diagram representing the system design of our mobile application.

The system is composed of 9 entity classes, a Control class and a View class.
The Student class represents a user of ICUE CARDS. Each student is unique and is in fact a FireBaseUser object that contains pertinent attributes such as time created, an associated unique ID, and the date and time the user last logged in. a student also has a name, a preferred language which can either be English or French, and a theme which will manifest as a button the user can toggle to switch between light and dark themes. The Student class is currently still in its early stages and more attributes will be added as the project furthest along. Most importantly, however, a Student has a Root object which provides access to all the cue cards he or she has created.

The Root object represents the top of a hierarchy of a file system implemented so that students using the app can organize the cue cards they create by class subject, and by subject sections. The hierarchy described briefly in the Architectural Design section of this report is as follows: The Root object has many folders which represent the courses the student is taking. Each folder can have multiple Deck objects which represent the chapters within a course the student needs to study upon. Finally, a Deck object consists of iCueCard objects which are modelled after flashcards. The Root, Folder, and Deck objects can be given a name to appropriately identify the term, the course, and the chapter corresponding to it. They each have a time attribute representing the time at which the object was created which later enables the user to sort their decks, or folders based on time created. This sorting order will be expanded upon to also sort in alphabetical order.

Of particular note is the Deck class which has a number of methods associated to it. A deck is able to manage its inventory of iCueCard objects by editing them, adding more, or deleting them. It also has a function capable of configuring the order by which the individual cue cards of the deck are organized.

The iCueCard class is the centrepiece of ICUECARDs. All iCueCard objects have an id that uniquely identifies them. An iCueCard has a front and backside which currently represent the text on the front and backside of a cue card. Typically the frontside of a cue card will contain the keyword or concept the student needs to understand and the backside will contain a description of that keyword in a paragraph or so. The backSide attribute of the iCueCard object is subject to some modifications particularly when inserting photos onto the backside of a cue card becomes possible. Lastly, an iCueCard object has a colour object that by default is grey but can be changed so that the cue card can be categorized.

The non functional requirements of ICUE CARDS remain at the forefront for the team. They are as follows:

- The app was programmed with user friendliness in mind so that students can focus more on their studies and less on struggling with a confusing layout or UI.
- The performance of ICUE CARDS should be very fast; creating or editing existing cue cards or directories should make the appropriate changes instantly, loading between pages should be snappy, and user authentication done in seconds.
- The project is designed with scalability in mind to allow for new game modes or new cue card features.

Architectural Design Decision

The architecture of this project is a combination of a Client - Server architectural style and an Object-Oriented architectural style.

ICUE CARDS uses Firebase as a backend server to store the cue card decks the Student creates within the app. The implementation of Firebase as a Backend-As-A-Service follows a Client-Server architectural style. In this layered architectural style, the client and server sides of the application are easily visualized and partitioned.

Figure 1 below depicts an abstracted overview of the client - server architecture of ICUE CARDS. When a student opens the mobile application to begin a new or resume a previous session, they are prompted with the login page. A stream has been set up between our application and Firebase. When the user is logged in, Firebase sends the application a FirebaseUser object which all have a unique ID used to identify individual students. When the user clicks the logout button on the homepage, Firebase sends a NULL pointer to signal this change in status and the application responds by logging the user out of his/her session. When the user wishes to log back in, they must pass in their credentials correctly into the provided text fields of the login page. The application packages these fields into a request to authenticate to the Firebase Authentication backend, to which Firebase returns a NULL pointer if a user with those credentials cannot be found in its user database or the FirebaseUser object itself if there is a match.

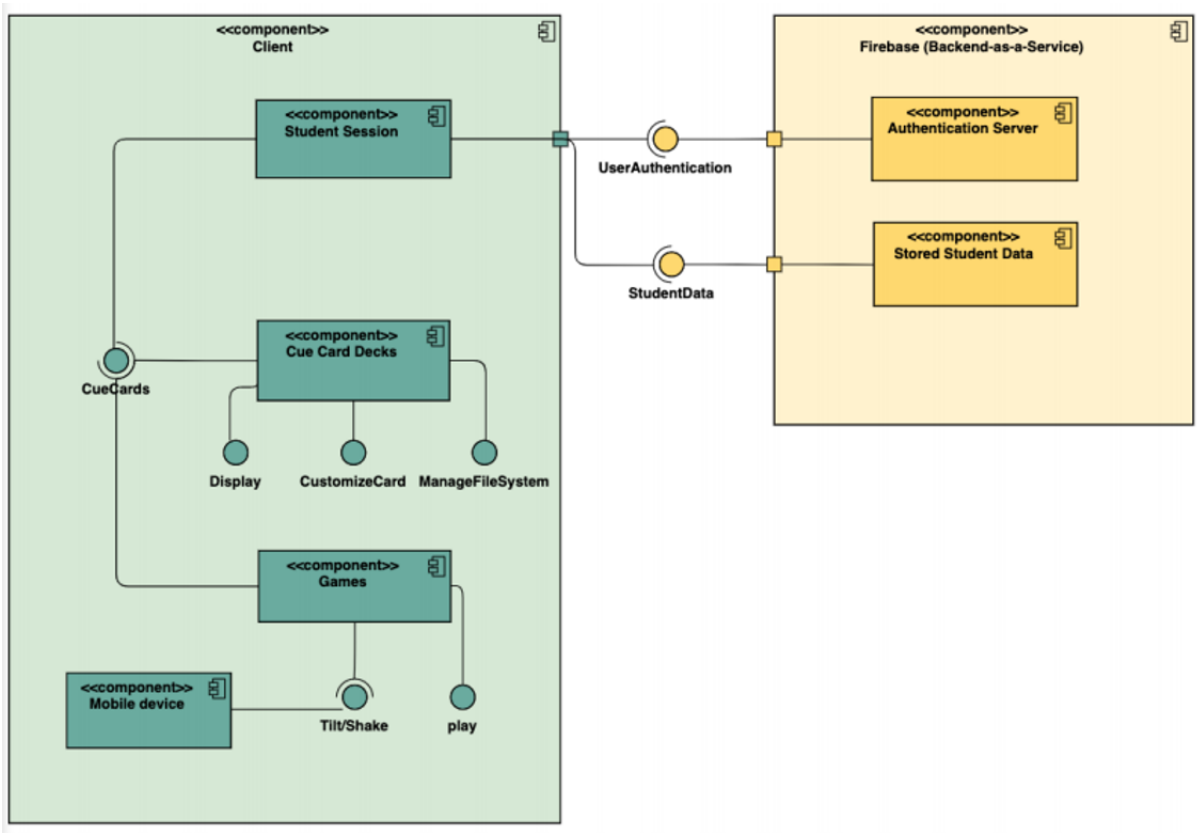


Figure 1. ICUE CARDS Client-Server component diagram

Client-Server architecture aside, this project strictly adheres to the principles of an Object-Oriented architecture.

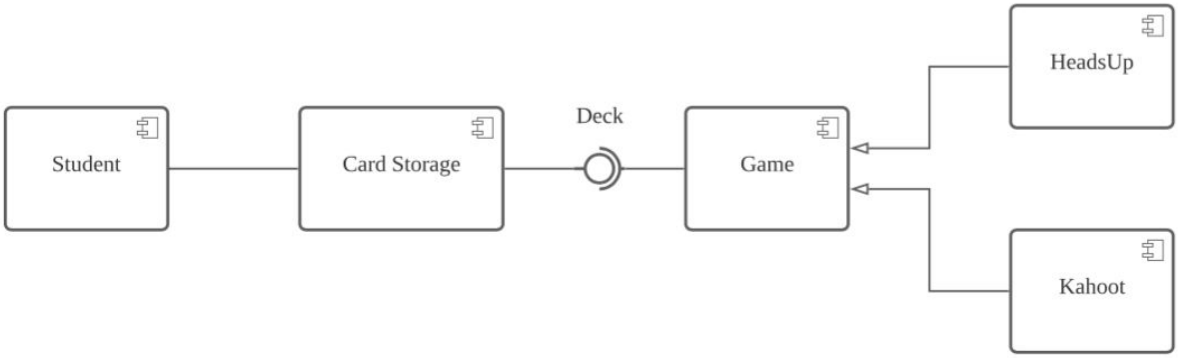
Development of our mobile application is being done in the Flutter mobile framework so that ICUE CARDS can be available on both iOS and Android. Programming in this framework necessitates the use of Dart, an object-oriented programming language that is conducive to an Object-Oriented architectural style.

This style was chosen not just for its compatibility with our chosen programming language of choice but also because ICUE CARDS is, at its core, a flashcard app meant to simulate real flashcards that have a frontside, a backside, that can be colour-coded, drawn on when simple text is insufficient, shuffled into a random order with other cue cards in a deck, and practiced upon over and over again. It became very natural to partition the architecture of our system into components based on these real-life entities that are already very familiar to us. The benefit of being able to visualize our architecture as a web of connections between these components modeled after real-life entities allows our team to effectively manage its complexity.

Besides being a very natural way to design the architecture of our system, this architectural style has the benefit of encapsulation, one of the fundamentals of

Object Oriented Programming. Our team was able to work independently with minimal fear of merge conflicts because we had established this design early in development. If one team member decided to add some dependencies to an external library or framework then this does not necessarily affect another team member who is building upon a completely separate service of the app.

This design allows for easier debugging and code maintenance. For example, if there is an issue with a program's user interface, then the bug can be traced to the View class of our system. If a bug surfaces relating to the calculation of user scores at the end of a game of Kahoot! or Heads Up!, then debugging needs to occur in the Results class. And since our design is well encapsulated, a bug is typically centralized in a single class rather than spread across the entire code base.



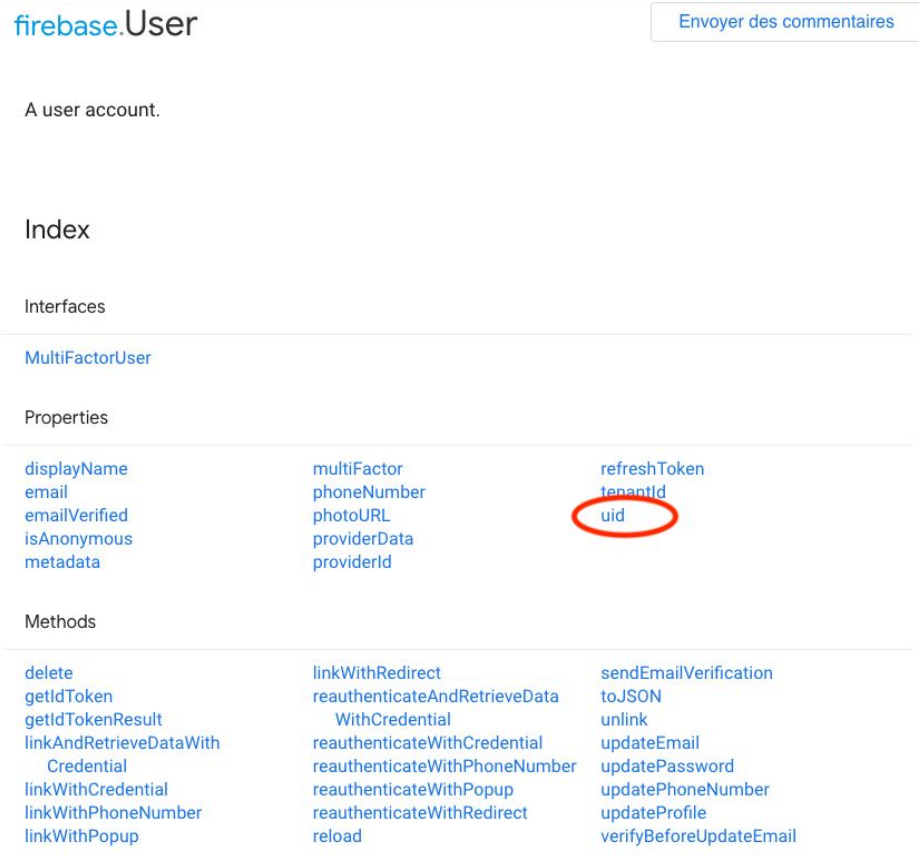
ICUE CARDS client-side Object-Oriented Architecture view

This figure depicts the five main classes in the architecture of our project which include Student, Card Storage, Game, HeadsUp and Kahoot. The Student class represents a typical user of our application which is primarily geared towards students. Students interact with a card storage with which they can manage the cue cards they create into a filesystem that makes sense for them. Students will be able to create new cue cards, delete them, customize their colour, and include pictures among many other features to enjoy. The Game class is one that requires a Deck object from the Student's Card Storage to play with. The two game modes currently implemented in ICUE CARDS are represented here as the HeadsUp and Kahoot class.

SYSTEM DESIGN

The design of our system is a hybrid of multiple design patterns owing to the components in our system having very different and distinct behaviours.

One design pattern our application leverages is the Facade design pattern which is a structural design pattern. It is one such that a simple interface or class called a facade is used to greatly manage the complexity of a library or framework that is otherwise very complex. In other words, this design pattern entails providing a simple and clean interface to a complex system in order to integrate only the pertinent or relevant functionalities into our application. Another benefit to this design pattern is that it reduces the extent of coupling between our client code and external libraries by only implementing the specific snippets of the library that our client code requires to execute properly.
An example where this design pattern is utilized in our application is in the Student class. A FirebaseUser object contains dozens of properties and methods that are irrelevant to the needs of our cue card application. This includes attributes such as phoneNumber, multiFactor, and refreshToken which our application does not need to track. Rather than work with FirebaseUser objects, we opted for an abstracted version of this which we termed a Student class which only needs FirebaseUser.uid as its parameter.



Properties and Methods of a FirebaseUser Object

Another design pattern our application leverages is the Factory Method, which is a creational design pattern.

This particular design principle is prevalent in the gaming portion of our mobile application. Heads Up! and Kahoot! are different games in concept but share some similar attributes and methods which we capture in a superclass we've called the Game class. The addition of an abstract Game class allows the team to reuse source code and avoid duplication. In particular, both Kahoot! and Heads Up! must select a deck a cue card from the user's card storage to play from, they require a function to launch their respective gaming modes, and when the game has been completed, they both display a results page summarizing the cue cards the student still needs to review.

But the way two game modes shuffle the questions are quite different which will implement separately in each class. For the HeadsUp, different from the Kahoot game, it needs to display in a horizontal view and it only needs to show the keyword of the Cue Cards on the screen. It also needs to use device titling to go on the game. For the Kahoot, it needs to generate the choice (which is the description of the cue card) which is not necessary in the HeadsUp game. When doing the implementation, it only needs to focus on characteristics of each mode.
Another element that the factory design pattern provided is the interface. For here, it is the Result class. In the Result class it records the player's score and the time they use. It also enables players to review their mistakes.
These four components with the factory design pattern makes the Game mode a complete and efficient system.