W9 PRACTICE

QUIZ APP

Important

The reflection part will be done in **teams of 2** (*designing*) and 4 (*sharing*)
The coding part needs to be submitted **individually**

Learning objectives

Handle **navigation** between **multiple screens** – *Using a state (not router for now…)*
Pass data between screens
Separate **UI logic** from **business logic**: using a model folder
Reflect on the best approaches (***data, states, widgets***) to maintain a clean architecture

How to submit?

- ✓ **Push** your final code on **your GitHub repository**
- ✓ Then **attach the GitHub path** to the MS Team assignment and **turn it in**

Functional Requirements

For this practice (W9)

The player can **start the quiz** and **answer each question** one by one
Only single choice questions
Once finished, the app shows the **score and the questions results**

For Bonus

The history of the previous scores can be reviewed
The **quiz questions** and **player submission** are persisted in JSON file

For next practice (W10)

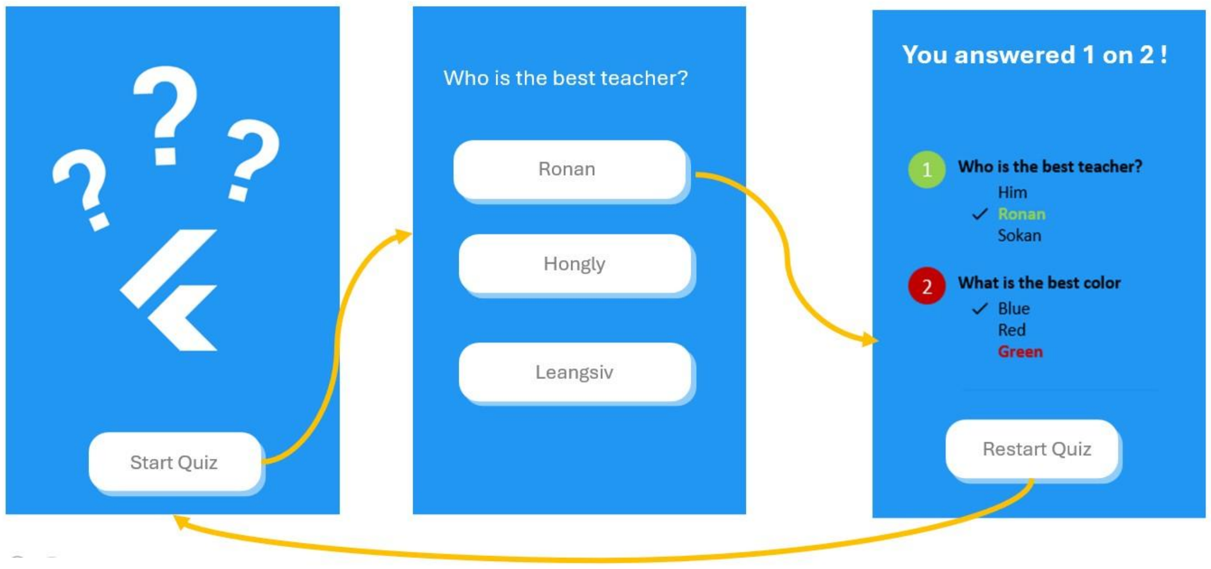
The player **enters his/her name** before starting
It's possible to **edit the quiz questions**

Non-Functional Requirements

The application must **implement the provided user flow and mockups**

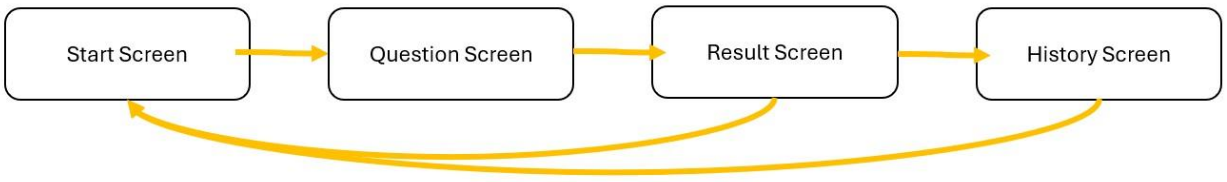
User Flow

For this practice, the following **user flow**/mockup are required:



BONUS

To include the **history of the previous scores**, the **user flow** can evolve as follows:



Layer structure

The application is structured around 3 layers: DATA > DOMAIN > UI

data	Repositories to load **domain objects** from **data sources**
model	Contain the **domain classes**
ui/screen	Screen widgets and sub-screen widgets
ui/widgets	Re usable widgets (button, inputs…)

Here is an **example** of project structure *(just an example, not the correct one)*

```
lib/  ├
data/
 │     └ repositories/
 │            └ quiz_json_repository.dart
 │             └ quiz_mock_repository.dart
 ├ models/
 │     └ quiz.dart
 │
 └ ui/
       ├ screens/
       │     └ welcome_screen.dart
       │     └ question_screen.dart
       └ widgets/
             └ app_button.dart
              └ app_button.dart

       main.dart
```

Layer interaction

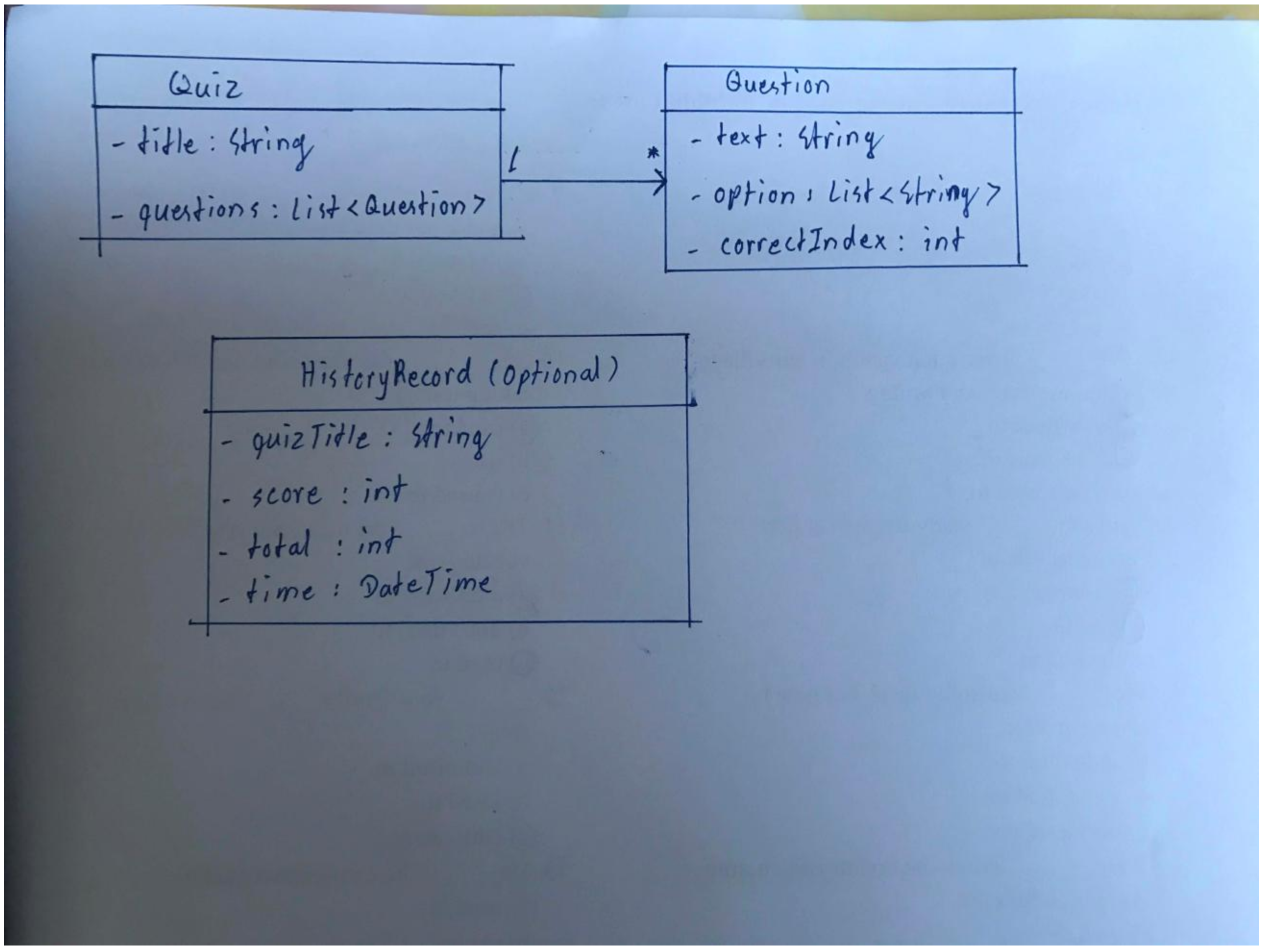
1. The **main** loads the **quiz data** (*from mock data or from a Json file)*
2. The **main** create the **quiz screen**, passing the quiz data as parameter

PART 1 – REFLECTIONS

MODEL

To handle the functional requirements for this practice, and be ready for the next practice, how are you going to structure your model?

Q1 – Drop below the **UML diagram of your model**



Q2 – Where do you **keep player submission,** so that you can display the last screen?

1. **In-memory during quiz**: In the QuestionScreen state, we maintain a List<int?> answers array where each index corresponds to a question, and the value is the selected option index (or null if unanswered).
2. **Persistent storage (History)**: After completing the quiz, the score and quiz details are saved to:
   - **Web**: Browser's localStorage via HistoryRepositoryWeb
   - **Mobile/Desktop**: Local file history.json via HistoryRepository

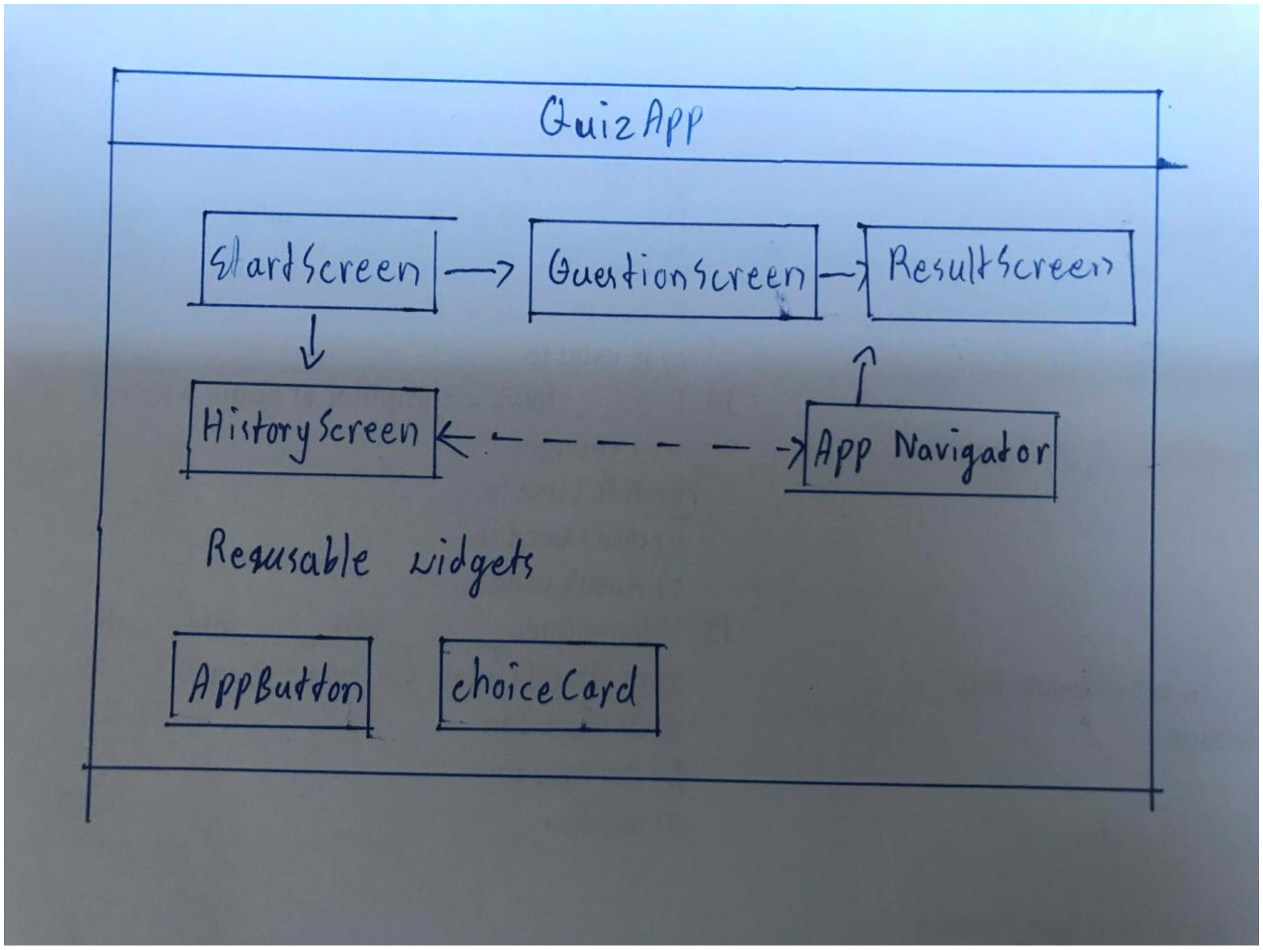
UI – **Screens**

We have 3 screens (start, question and result)

Q3 – Identify for **each widget** their properties

WIDGET	TYPE (SL / SF)	PARAMETERS	STATES
StartScreen	SL	quiz: Quiz	None
QuestionScreen	SF	quiz: Quiz	current: int answers: List<int>
ResultScreen	SF	quiz: Quiz answers:List<int> score: int	None (only calls initState to save score)
HistoryScreen	SF	None	history: List<Map<String, dynamic>>

Q4 – Draw the **COMPONENT DIAGRAM** of the application



Q5 – Where and How do you **manage the navigation** to the **next questions** and to the **last result screen**?

1. **Between questions within QuestionScreen**:
    - **State management**: current index tracks which question is displayed
    - **Next button**: Increments current until last question
    - **Previous button**: Decrements current (disabled on first question)
    - **No screen change** - just state update within the same screen
2. **To ResultScreen**:
    - When current == questions.length - 1 and user clicks "Finish"
    - Navigator.pushReplacement replaces QuestionScreen with ResultScreen
    - Passes quiz, answers, and computed score as parameters
3. **Back to StartScreen**:
    - Navigator.popUntil(context, (route) => route.isFirst) in ResultScreen
    - This pops all screens until reaching the first screen (StartScreen)

UI – **Reusable widget**

List down the widget you are **planning to re-use** on different screens (button, card..)

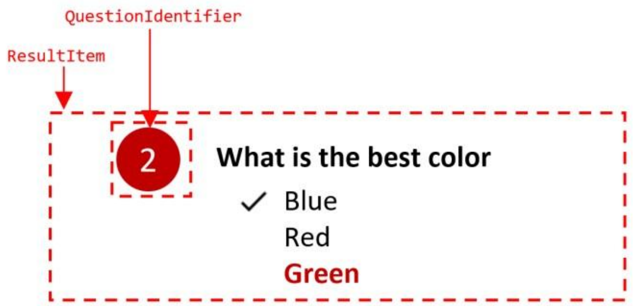
WIDGET	TYPE (SL / SF)	PARAMETERS	STATES
AppButton	SL	label: String onPressed: VoidCallback enabled: bool	None
ChoiceCard	SL	Text: String Selected: bool onTap: VoidCallback showCorrect: bool isCorrect: bool	None

PART 2 – IMPLEMENTATION

The **coding part** needs to be submitted **individually**

HINTS

Tip: you can divide each screen into many **stateless screen-widgets**, for example:



This widget takes as parameter a question and a player choice and handle the color computation, the choices highlighting etc..