Math Quiz App Documentation

Present: Avishay Elrom 207946591

1. Project Proposal-

Purpose- The Math Quiz app is designed to offer a fun and engaging way for users to test and improve their math skills. This app aims to make learning math more interactive and accessible, providing users with immediate feedback to help them understand their strengths and areas that need improvement.

Target Audience-

- **Students:** Particularly those in primary and secondary education looking to practice and enhance their math skills.
- **Teachers/Educators:** As a tool for assessing students' understanding of mathematical concepts and for creating engaging homework or class activities.
- **Parents:** For parents who want to help their children practice math at home in a structured and enjoyable way.
- **General Math Enthusiasts:** Anyone interested in testing their math knowledge in a fun and challenging format.

Key Features-

- **Interactive Quiz:** Users can answer math questions and receive immediate feedback on their answers.
- **Profile Management:** Users can create and save a profile with their name, which personalizes their experience.
- **Dynamic Question Generation:** Questions are generated dynamically, providing a unique experience every time.
- **Feedback System:** The app provides feedback on whether the answer is correct or not, aiding learning.
- **User-Friendly Navigation**: The app is easy to navigate, with intuitive buttons for starting quizzes, saving profiles, and navigating between screens.

2. Technical Documentation-

Architecture-

The Math Quiz app is built using the Model-View-ViewModel (MVVM) architecture. This architecture helps in separating concerns, making the app easier to manage, test, and scale.

- **Model:** Represents the data layer of the app, including data classes and business logic.
- **View:** Contains all the UI elements and is responsible for displaying data to the user and capturing user inputs.
- **ViewModel:** Serves as a bridge between the View and the Model, holding UI data in a lifecycle-conscious way and managing the UI-related logic.

Main Components-

1.MainActivity:

- The entry point of the application.
- Sets up the theme and initializes the navigation graph.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyAppTheme {
                NavGraph(navController = rememberNavController()) // Pass it to NavGraph
            }
        }
    }
}
```

2.NavGraph:

- Manages the navigation between different composable/screens.
- Defines navigation routes and handles the passing of arguments between screens.

```
@Composable
fun NavGraph(navController: NavHostController){
    NavHost(navController = navController, startDestination = "homeScreen") {
        composable( route: "homeScreen") { HomeScreen(navController) }//composable destination and the screen that match
        composable(
            route = "quizScreen/{userName}",
            arguments = listOf(navArgument( name: "userName") { type = NavType.StringType })
        ) { backStackEntry ->
            val userName = backStackEntry.arguments?.getString( key: "userName") ?: ""
            QuizScreen(navController, userName)
        }
        composable( route: "profileScreen") { ProfileScreen(navController) }
    }
}
```

3. QuizScreen:

- Displays a list of dynamically generated math questions.
- Allows the user to input their answers and submit them for feedback.
- Shows feedback based on the correctness of the user's answers.

```
@Composable
fun QuizScreen(navController: NavController, userName: String, viewModel: QuizViewModel = viewModel()) {
    // Handle the userName as needed.
    val questionList = viewModel.questions.collectAsState().value
    val userAnswers = remember { mutableStateListOf<String>() } // List to store all user answers
    var showResults by remember { mutableStateOf( value: false) } // State to control when to show results

    // Initialize userAnswers with empty strings if not already initialized
    if (userAnswers.isEmpty()) {
        userAnswers.addAll(List(questionList.size) { "" })
    }
```

```
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        LazyColumn(
            modifier = Modifier.weight(1f), // Take up the remaining space
            contentPadding = PaddingValues(bottom = 16.dp) // Avoid overlap with the bottom button
        ) {
            items(questionList.size) { index ->
                val question = questionList[index]

                Column(modifier = Modifier.padding(bottom = 16.dp)) {
                    Text(text = question.text, style = MaterialTheme.typography.headlineSmall)
                    Spacer(modifier = Modifier.height(8.dp))
                    TextField(
                        value = userAnswers[index], // Display the current answer
                        onValueChange = {
                            userAnswers[index] = it // Update the answer in the list
                        },
                        label = { Text( text: "Your Answer") }
                    )
                    Spacer(modifier = Modifier.height(16.dp))
                    if (showResults) {
                        val feedbackMessage = if (viewModel.isAnswerCorrect(index, userAnswers[index])) {
                            "Correct!"
                        } else {
                            "Not Correct!"
                        }
                        Text(text = feedbackMessage, style = MaterialTheme.typography.bodyLarge)
                        Spacer(modifier = Modifier.height(8.dp))
                    }
                }
            }
        }
```

The rest of QuizScreen:

```
        // Submit Answer button at the bottom
        Button(
            onClick = { showResults = true },
            modifier = Modifier
                .width(300.dp)
                .padding(top = 16.dp)
        ) {
            Text(text = "Submit Answer")
        }

        Spacer(modifier = Modifier.height(16.dp))

        // Back to Home button at the bottom
        Button(
            onClick = { navController.navigate( route: "homeScreen") },
            modifier = Modifier
                .width(300.dp)
        ) {
            Text(text = "Back to Home")
        }
    }
}
```

4.ProfileScreen-

- Allows the user to input their name and save it as a part of their profile.
- Provides feedback on profile save operation and navigates back to the Home Screen upon saving.

```
@Composable
fun ProfileScreen(navController: NavController, viewModel: ProfileViewModel = viewModel()) {
    val userName = viewModel.userName.collectAsState().value
    var feedbackMessage by remember { mutableStateOf( value: "") } // State to hold the feedback message
    var shouldNavigate by remember { mutableStateOf( value: false) } // State to control navigation after delay

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        TextField(
            value = userName,
            onValueChange = { viewModel.onNameChanged(it) },
            label = { Text( text: "Your Name") }
        )
        Spacer(modifier = Modifier.height(16.dp))

        Button(onClick = {
            viewModel.saveProfile()
            feedbackMessage = "$userName profile saved."
            shouldNavigate = true // Set flag to true to trigger navigation after delay
        }, Modifier.width(200.dp)) {
            Text(text = "Save and start quiz")
        }

        Spacer(modifier = Modifier.height(16.dp))

        if (feedbackMessage.isNotEmpty()) {
            Text(text = feedbackMessage, style = MaterialTheme.typography.bodyLarge)
        }

        // Back to Home screen button
        Button(
            onClick = { navController.navigate( route: "homeScreen") },
            modifier = Modifier
                .width(200.dp)
                .padding(top = 16.dp)
        ) {
            Text(text = "Back to Home")
        }
    }

    // delay and navigate after the delay
    if (shouldNavigate) {
        LaunchedEffect(key1 = true) {
            delay( timeMillis: 500L) // 0.5-second delay
            navController.navigate( route: "quizScreen/$userName") // Navigate to QuizScreen with userName as NavArg
        }
    }
```

5.QuizModel-

- Maintains the state for the quiz questions and user answers.
- Provides methods to validate user answers and fetch questions.

```
data class Question(val text: String, val correctAnswer: String)

class QuizViewModel : ViewModel() {
    private val _questions = MutableStateFlow(listOf(
        Question( text: "2 + 2:",  correctAnswer: "4"),
        Question( text: "2X2:",  correctAnswer: "4"),
        Question( text: "2X4",  correctAnswer: "8"),
        Question( text: "5+7",  correctAnswer: "12"),
        Question( text: "9X3",  correctAnswer: "27"),
        Question( text: "15/3",  correctAnswer: "5"),
        Question( text: "what is the square root of 49?",  correctAnswer: "7"),
        Question( text: "what is 25% of 200?",  correctAnswer: "50"),
        Question( text: "3!",  correctAnswer: "6"),
        Question( text: "what is the sum of the angles in a triangle?(in degrees)",  correctAnswer: "180")


    ))
    val questions: StateFlow<List<Question>> = _questions

    fun isAnswerCorrect(index: Int, userAnswer: String): Boolean {
        val question = questions.value.getOrNull(index)
        return question?.correctAnswer.equals(userAnswer, ignoreCase = true)
    }
}
```

Code Explanations

- **Data Classes**:
  - Question: represents a single math question with text and the correct answer.
- **ViewModels**:
  - QuizViewModel: handles quiz logic, including fetching questions and validating answers.
  - ProfileViewModel: manages user profile data, handling changes to the user's name.
- **Composable Screens**:
  - HomeScreen: The landing screen where users can choose to start the quiz or update their profile.
  - QuizScreen: Displays questions and captures user input.
  - ProfileScreen: Allows users to input their name and save their profile.

3. User Manual

How to Use the Application

1. **Home Screen:**
    - Start Quiz: Click the "Start Quiz" button to navigate to the QuizScreen. You'll be prompted to answer a series of math questions.
    - Profile/Settings: Click the "Profile/Settings" button to navigate to the ProfileScreen, where you can enter and save your profile information.
2. **Profile Screen:**
    - Enter Name: Enter your name in the text field provided.
    - Save Profile: Click the "Save" button to save your profile. A confirmation message will appear, and you will be automatically navigated back to the Home Screen after a short delay.
    - Back to Home: You can also directly click the "Back to Home" button to return to the Home

3. **Quiz Screen:**
    - Answer Questions: For each question displayed, type your answer in the text field.
    - Submit Answers: Click the "Submit Answer" button to check your answers. Feedback will be displayed next to each question, indicating whether the answer was "Correct!" or "Not Correct!"
    - Back to Home: After checking your answers, click the "Back to Home" button to return to the Home Screen.

Tips for Effective Use

- Profile Setup: Make sure to set up your profile before starting the quiz for a personalized experience.
- Immediate Feedback: Use the feedback provided after submitting answers to learn and improve.
- Practice Regularly: The more you practice, the better your math skills will become.