**Technical Spesification Document (TSD)**

# SOCIAL STYLE QUIZ APPLICATION

**Prepared by:**

Nabila Khairunnisa

Version:

1.0

July 6, 2025

# Table of Contents

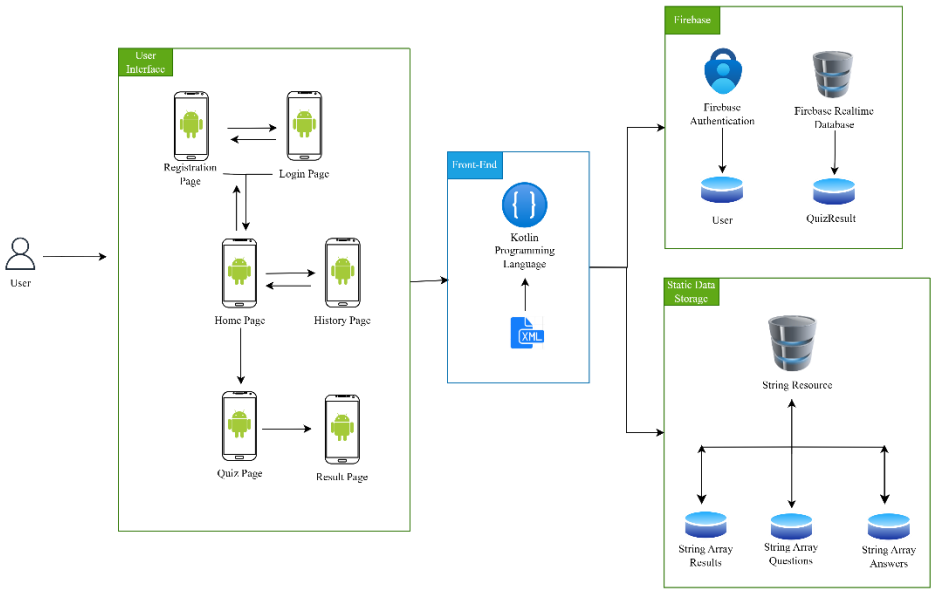
## 1. Introduction

This document outlines the technical implementation plan for the Social Style Quiz app. It includes the system architechture, component structure, data models, logic flow, technologies used, and potential integration points. This document detailing how each function will be implemented.

## 2. Architecture & Technology Stack

| Area | Technology Used |
|---|---|
| Mobile App | Android |
| Architecture Pattern | MVVM (Model-View-ViewModel) |
| State Mangement | ViewModel + LiveData |
| UI Layer | XML, Kotlin, ViewBinding |
| Data Layer | Strings.xml (local) & Firebase (remote) |
| Interface Engine | Forward Chaining rule processor |
| Testing | Junit & Mockito |
| Build Tool | Gradle |
| Version Control | Git & Github |

## 3. Information Architecture




This information architecture shows several components, from the UI and Front-End to the Back-End. The application uses Firebase to store quiz result data and String Array Resources to display the Q&A and social style information.

## 4. Key Modules

```
-- data
|   -- QuizResult.kt
|   -- User.kt
-- ui
|   -- start
|      | -- StartViewModel.kt
|      | -- StartActivity.kt
|      | -- StartUtils.kt
|      | -- SessionViewModel.kt
|      | -- SessionViewModelFactory.kt
|   -- quiz
|      | -- QuizActivity.kt
|      | -- QuizUtils.kt
|      | -- QuizViewModel.kt
|   -- result
|      | -- ResultActivity.kt
|      | -- ResultUtils.kt
|      | -- ResultViewModel.kt
|   -- history
|      | -- HistoryActivity.kt
|      | -- ResultAdapter.kt
|      | -- HistoryUtils.kt
|   -- signin
|      | -- SignInActivity.kt
|      | -- SignInViewModel.kt
|   -- signup
|      | -- SignUpActivity.kt
|      | -- SignUpViewModel.kt
```

| Module Name | Class Name | Description / Function |
|---|---|---|
| signin | `SignInActivity.kt` | Manages UI, user input, and navigation. |
| | `SignInViewModel.kt` | Handles sign-in logic, validates input, and communicates with Firebase Authentication. |
| signup | `SignUpActivity.kt` | Manages the UI and captures user input for the registration process. |
| | `SignUpViewModel.kt` | Contains sign-up logic, validates input, creates user accounts via Firebase, and stores data in the Realtime Database. |
| start | `StartActivity.kt` | Displays introductory information, handles UI interactions, navigation, and logout. |
| | `StartUtils.kt` | Provides utility functions for UI manipulation. |
| | `StartViewModel.kt` | Manages state and logic for displaying intro content. |
| | `SessionViewModel.kt` | Handles user session management and authentication logic. |

| Module Name | Class Name | Description / Function |
|---|---|---|
| | `ViewModelFactory.kt` | A factory class for creating `SessionViewModel` instances with dependency injection. |
| quiz | `QuizActivity.kt` | Manages the quiz UI, displays questions, handles user input, and navigates to the result screen. |
| | `QuizUtils.kt` | Contains helper functions for UI rendering and user feedback. |
| | `QuizViewModel.kt` | Manages quiz data, tracks question index, and tallies user answers. |
| result | `ResultActivity.kt` | Displays quiz results, processes data from the ViewModel, and handles actions like sharing. |
| | `ResultUtils.kt` | Provides utility functions for formatting and displaying results. |
| | `ResultViewModel.kt` | Handles result logic, calculates scores, saves results to Firebase, and prepares shareable images. |
| history | `HistoryActivity.kt` | Fetches and displays a list of past quiz results from Firebase using a real-time listener. |
| | `ResultAdapter.kt` | Populates the quiz result list and manages item expansion. |
| | `HistoryUtils.kt` | Provides descriptions for each social style. |

## 5. String Resources Structure

| Section | Resource Name | Description / Content |
|---|---|---|
| **Questions** | `string-array questions` | Contains 20 unique questions used in the quiz. |
| **Answers** | `string-array amiableAnswers` | Answer choices tailored to the Amiable social style. |
| | `string-array analyticalAnswers` | Answer choices tailored to the Analytical social style. |
| | `string-array expressiveAnswers` | Answer choices tailored to the Expressive social style. |
| | `string-array driverAnswers` | Answer choices tailored to the Driver social style. |
| **Results** | `desc_ami`, `desc_ana`, `desc_exp`, `desc_dri` | Descriptions of each social style's characteristics. |
| | `strengths_ami`, `strengths_ana`, `strengths_exp`, `strengths_dri` | Lists of strengths associated with each social style. |
| | `weakness_ami`, `weakness_ana`, `weakness_exp`, `weakness_dri` | Lists of weaknesses associated with each social style. |
| | `solution_ami`, `solution_ana`, `solution_exp`, `solution_dri` | Advice or solutions tailored to each social style. |

## 6. QnA Display Logic

This pseudocode shows how the QnA is displayed in the application:

```
FUNCTION showQna(index)
    DISPLAY questions[index] ON questionTextView
    DISPLAY amiableAnswers[index] ON amiableAnswerTextView
    DISPLAY analyticalAnswers[index] ON analyticalAnswerTextView
    DISPLAY expressiveAnswers[index] ON expressiveAnswerTextView
    DISPLAY driverAnswers[index] ON driverAnswerTextView
END FUNCTION
```

7. **Save Answer Logic**

This function increases the count of a specific social style based on the user's selected answer.

```
FUNCTION saveAnswer(selectedId)
    IF selectedId EQUALS amiableId THEN
        INCREMENT amiableCount
    ELSE IF selectedId EQUALS analyticalId THEN
        INCREMENT analyticalCount
    ELSE IF selectedId EQUALS expressiveId THEN
        INCREMENT expressiveCount
    ELSE IF selectedId EQUALS driverId THEN
        INCREMENT driverCount
    END IF
END FUNCTION
```

8. **Determine Social Style Logic**

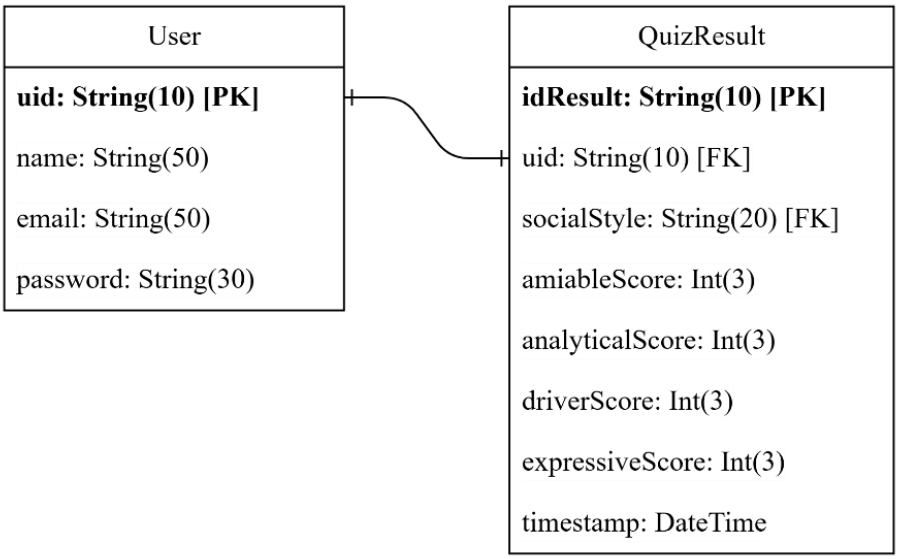
This function returns a String value indicating the dominant social style. The >= operator is used to compare the scores of each social style.

```
FUNCTION determineSocialStyle
    IF ami ≥ ana AND ami ≥ exp AND ami ≥ dri THEN
        RETURN "Amiable"
    ELSE IF ana ≥ ami AND ana ≥ exp AND ana ≥ dri THEN
        RETURN "Analytical"
    ELSE IF exp ≥ ami AND exp ≥ ana AND exp ≥ dri THEN
        RETURN "Expressive"
    ELSE
        RETURN "Driver"
END FUNCTION
```

9. **Database Design**

a. Table Schema and Relation

The relationship between the User table and the QuizResult table is one-to-one, where each user has only one quiz result history.



b. JSON Schema

The following represents the JSON structure stored in Firebase Realtime Database.

```
"QuizResult": {
  "res01": {
    "amiableScore": 30,
    "analyticalScore": 35,
    "driverScore": 20,
    "expressiveScore": 15,
    "name": "Nabila Khairunnisa",
    "socialStyleName": "Analytical",
    "timeStamp": "2 August 2025 14:39",
    "uid": "uid01"
  }
}
"User": {
  "uid01": {
    "email": "nabila@gmail.com",
    "name": "Nabila Khairunnisa"
  }
}
```

**QuizResult Object**: Stores individual quiz results keyed by unique result IDs.
**User Object**: Stores user profile data keyed by UID.

## 10. Testing

a. Black Box Testing

| No | Precondition | Test Scenario | Expected Result | Test Result |
|---|---|---|---|---|
| 1 | First question is displayed and an answer is selected | Tap the **"Next"** button | Moves to the next question | Valid |
| 2 | Second or later question is displayed | Tap the **"Previous"** button | Moves to the previous question | Valid |
| 3 | Question is displayed with no answer selected | Tap the **"Next"** button | Button is disabled and error message appears | Valid |
| 4 | Last question is displayed and all questions are answered | Tap the **"Finish"** button | Navigates to the quiz result screen | Valid |
| 5 | Last question is displayed and not all questions are answered | Tap the **"Finish"** button | Button is disabled and error message appears | Valid |

b. Unit Testing

1)
```
@Test
fun `if amiable score 10, then amiable percentage should 50`() {
    val amiablePercentage = result.amiablePercentage(10).toInt()
    assertEquals(50, amiablePercentage)
}
```

2)
```
@Test
fun `if analytical score 3, then analytical percentage should 15`() {
    val analyticalPercentage = result.analyticalPercentage(3).toInt()
    assertEquals(15, analyticalPercentage)
}
```

3)
```
@Test
fun `if expressive score 2, then expressive percentage should 10`() {
    val expressivePercentage = result.expressivePercentage(2).toInt()
    assertEquals(10, expressivePercentage)
}
```

4)
```
@Test
fun `if driver score 5, then driver percentage should 25`() {
    val driverPercentage = result.driverPercentage(5).toInt()
    assertEquals(25, driverPercentage)
}
```

Unit test result: