Overview

This design provides a somewhat comprehensive blueprint for building the **Amazin' Quizzer App**. Each component is clearly defined with a focus on modularity, making the application easier to develop, maintain, and expand. However, this is not the complete function because not all the features are added, and as this is added the design will adapt to meet evolving requirements.

Overall Flow

1. **Main Execution Flow (`main()`)**:
   - Coordinates the entire quiz process.
   - Handles the sequential execution from user input to displaying results.

Detailed Component Breakdown

1. **User Input Handling**

   - **`getUserCategoryChoice()`**:
     - **Purpose**: Obtain the user's choice for quiz category and difficulty level.
     - **Returns:** An object containing the selected category and difficulty.

   - **Enhancements:**
     - Validate user input to ensure it's within the available options.
     - Provide default or random options if the user skips selection.

   - TypeScript:

   ```
   getUserCategoryChoice() -> dict:
     # Prompt user for category and difficulty
     # Validate inputs
     # Return as {'category': 'Math', 'difficulty': 'easy'}
   ```

2. **Category Management**

- **`generateCategory(category)`**:
    - **Purpose**: Automatically generate a new category based on user input.
    - **Returns**: A unique ID for the new category.
    - **Enhancements**:
        - Handle duplicates and allow for category management (edit/delete).

TypeScript

```
generateCategory(category: str) -> str:
    # Generate unique ID for the category
    # Store category in the database
    # Return category ID
```

- **addToListOfCategories(category)`**:
    - **Purpose:** Add the generated category to the list of available categories.
    - **Returns**: Boolean indicating success.

TypeScript

```
addToListOfCategories(category: str) -> bool:
    # Generate unique ID for the category
    # Add category to the list
    # Return True if successful
```

3. Quiz Generation

- **`generateQuizFromCategory(category, difficultyLevel, minimumQuestion=20)`**:
    - **Purpose**: Create a list of questions for the quiz based on the selected category and difficulty.
    - **Returns:** A string or object representing the generated quiz.
    - **Enhancement:**
        - Allow for dynamic question count based on user preferences or category availability.
        - Integrate adaptive difficulty based on user performance.

Typescript

```
generateQuizFromCategory(category: str, difficultyLevel: str, minimumQuestion: int = 20) -> list:
    # Generate unique quiz ID
    # Retrieve and filter questions from the database
    # Store questions in a quiz format
    # Return the generated quiz
```

```

- **storeQuestionsToDatabase(quiz, categoryName, level)**:
    - **Purpose:** Store the generated quiz in the mock database.
    - **Returns**: A string or confirmation message indicating success.

**TypeScript**

```
storeQuestionsToDatabase(quiz: list, categoryName: str, level: str) -> str:
    # Save the quiz questions to the database
    # Return confirmation message
```

4. **Quiz Setup and Display**

- **setUpQuiz(quiz):**
    - **Purpose**: Prepare the quiz questions and answers for presentation.
    - **Returns**: String or status indicating setup completion.

**Typescript**

```
setUpQuiz(quiz: list) -> str:
    # Initialize quiz settings
    # Prepare each question and answer
    # Return setup status
```

- **displayQuiz(quiz)**:
    - **Purpose:** Manage the display of quiz questions, including a timer.
    - **Returns**: String or status after displaying the quiz.

TypeScript

```
displayQuiz(quiz: list) -> str:
    # Loop through questions
    # Display each question with a timer
    # Return status after quiz display
```
```

5. **User Interaction During Quiz**

- **renderToUser(quiz):**
- **Purpose**: Handle the flow of presenting questions to the user and gathering their answers.
- **Returns**: String or status after rendering the quiz.

TypeScript

```
renderToUser(quiz: list) -> str:
    # Loop through questions
    # Render each question
    # Check for end of quiz
    # Display results
    # Return status after rendering
```

renderQuestionToUser(question):

- **Purpose**: Display a single question to the user and manage the timer.
- **Returns:** None or status indicating if the timer has finished.

TypeScript

```
renderQuestionToUser(question: dict) -> bool:
    # Start timer for the question
    # Display question and choices
    # Keep track of the questions e.g next question, previous question, etc
    # Return timer status
```

- **startTimer(question, timer=120)**:
- **Purpose:** Start a countdown timer for each question.
- **Returns**: Boolean indicating if the timer is still running.

TypeScript

```
startTimer(question: dict, timer: int = 120) -> bool:
    # Initialize and start timer
    # Return True if time remains, False if time is up
```

6. **Question Navigation**

- **getNextQuestion(quiz):**
  - **Purpose**: Retrieve the next question in the quiz sequence.
  - **Returns**: The next question object.

TypeScript

```
getNextQuestion(quiz: list) -> dict:
    # Get the next question in the sequence
    # Return the question object
```

- **isEndOfQuiz(quiz):**
  - **Purpose**: Determine if the quiz has reached its end.
  - **Returns**: Boolean indicating the end of the quiz.

TypeScript

```
isEndOfQuiz(quiz: list) -> bool:
    # Check if all questions have been answered
    # Return True if end, False otherwise
```

7. **Answer Verification and Scoring**

- **`isAnswerCorrect(id, quiz)`**:
  - **Purpose**: Check if a given answer is correct.
  - **Returns**: Boolean indicating if the answer is correct.

TypeScript

```
isAnswerCorrect(id: str, quiz: list) -> bool:
    # Compare provided answer to correct answer
    # Return True if correct, False otherwise
```

- **getScore(quiz)**
  - **Purpose:** Calculate the user's score based on correct answers.
  - **Returns:** The calculated score.

Typescript

```
getScore(quiz: list) -> int:
    # Calculate the total score
    # Return the score
```

- **isPassed(score)**:
  - **Purpose:** Determine if the user passed the quiz based on their score.
  - **Returns:** Boolean indicating pass or fail.

Typescript

```
isPassed(score: int) -> bool:
    # Define passing criteria
    # Return True if passed, False otherwise
```

8. **Result Display**

- **displayResultOfQuiz(quiz):**
  - **Purpose**: Show the final results and provide options for next steps.
  - **Returns**: None.

TypeScript

```
displayResultOfQuiz(quiz: list) -> None:
    # Calculate and display score
    # Provide options to retry or start a new quiz
```

- **displayToUser(result):**
  - **Purpose**: Present detailed results and feedback to the user.
  - **Returns**: None.

TypeScript

```
displayToUser(result: dict) -> None:
    # Show detailed result analysis
    # Provide feedback and next steps
```

9. Mock Database Storage

databaseStorage():

- **Structure:** Represents how quizzes and questions are stored.

```json
{
  "quizzes": {
    "<someCategoryName>": {
      "categoryID": "some id",
      "difficulty": "easy",
      "questions": [
        {
          "questionID": {
            "question": "Some question here",
            "choices": ["A: choice 1", "B: choice 2", "C: choice 3", "D: choice 4", "E: choice 5"],
            "correctAnswer": "<correct answer here>"
          }
        },
        // More questions...
      ]
    }
  }
}
```