🚀 Développement Étape par Étape - Quiz App Flutter

📋 ÉTAPE 1 : Configuration initiale du projet

1.1 - Création du projet Flutter

flutter create quiz_app
cd quiz_app

1.2 - Configuration pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.6              # ← AJOUTÉ pour API
  html_unescape: ^2.0.0      # ← AJOUTÉ pour décoder HTML
  flutter_localizations:     # ← AJOUTÉ pour multi-langue
    sdk: flutter
  intl: ^0.19.0              # ← AJOUTÉ pour multi-langue
  shared_preferences: ^2.2.2  # ← AJOUTÉ pour stockage local
  audioplayers: ^5.2.1       # ← AJOUTÉ pour sons
  vibration: ^1.8.4          # ← AJOUTÉ pour vibrations

flutter:
  assets:                    # ← AJOUTÉ pour sons
    - assets/sounds/
```

Pourquoi ces packages ?

- `http` → Faire des requêtes vers l'API
- `html_unescape` → Les questions API contiennent du HTML encodé
- `audioplayers` → Jouer les sons correct/incorrect
- `vibration` → Vibrer quand erreur
- `shared_preferences` → Sauvegarder scores et paramètres

📁 ÉTAPE 2 : Structure des dossiers

Comment on a organisé :

```
lib/
├── main.dart              # ← Point d'entrée
├── Models/
│   └── settings.dart      # ← Modèle pour paramètres
├── screens/
│   ├── home_screen.dart   # ← Écran accueil
│   ├── quiz_setup_screen.dart # ← Configuration quiz
│   ├── quiz_screen.dart   # ← Le quiz principal
│   ├── result_screen.dart # ← Résultats
│   └── settings_screen.dart # ← Paramètres
└── services/
    ├── settings_service.dart # ← Gestion paramètres
    ├── theme_service.dart   # ← Gestion thème
    └── local_storage_service.dart # ← Gestion scores
```

Pourquoi cette organisation ?

- **Models** → Classes pour stocker nos données
- **Screens** → Chaque écran dans son fichier
- **Services** → Toute la logique métier séparée

🎯 ÉTAPE 3 : Modèle Settings - `Models/settings.dart`

3.1 - Problème à résoudre :

"Comment stocker les préférences utilisateur ?"

3.2 - Solution implémentée :

```
class Settings {
  final String languageCode;  // 'en', 'fr', 'ar'
  final bool enableSound;     // true/false

  Settings({required this.languageCode, required this.enableSound});

  // Pour modifier un paramètre sans tout recréer
  Settings copyWith({String? languageCode, bool? enableSound}) {
    return Settings(
      languageCode: languageCode ?? this.languageCode,
      enableSound: enableSound ?? this.enableSound,
    );
  }
```

```
  // Pour sauvegarder en JSON
  Map<String, dynamic> toJson() => {
    'languageCode': languageCode,
    'enableSound': enableSound,
  };

  // Pour charger depuis JSON
  factory Settings.fromJson(Map<String, dynamic> json) => Settings(
    languageCode: json['languageCode'] ?? 'en',
    enableSound: json['enableSound'] ?? true,
  );
}
```

Étapes de réflexion :

1. **Besoin** → Stocker langue + son activé/désactivé
2. **Méthodes** → `copyWith()` pour modification, `toJson()`/`fromJson()` pour persistance
3. **Défaut** → Anglais + son activé par défaut

🏠 ÉTAPE 4 : Écran d'accueil - `screens/home_screen.dart`

4.1 - Fonctionnalités à implémenter :

- Menu avec 3 boutons principaux
- Accès aux paramètres
- Titre multilingue

4.2 - Comment on l'a fait :

```
class HomeScreen extends StatelessWidget {
  final VoidCallback toggleTheme;      // ← Fonction pour changer thème
  final bool isDark;                   // ← État du thème actuel
  final Settings settings;             // ← Paramètres utilisateur
  final ValueChanged<Settings> updateSettings; // ← Fonction maj paramètres

  @override
  Widget build(BuildContext context) {
    final loc = AppLocalizations.of(context)!; // ← Traductions

    return Scaffold(
      appBar: AppBar(
        title: Text(loc.appTitle),        // ← Titre traduit
```

```
        actions: [
          IconButton(                    // ← Bouton paramètres
            icon: Icon(Icons.settings),
            onPressed: () => Navigator.pushNamed(context, '/settings'),
          ),
        ],
      ),
      body: Center(
        child: Column(
          children: [
            ElevatedButton(              // ← Bouton commencer quiz
              onPressed: () => Navigator.pushNamed(context, '/setup'),
              child: Text(loc.startQuiz),
            ),
            ElevatedButton(              // ← Bouton classement
              onPressed: () => Navigator.pushNamed(context, '/leaderboard'),
              child: Text(loc.leaderboard),
            ),
          ],
        ),
      ),
    );
  }
}
```

Étapes de développement :

1. **Wireframe** → Dessiner l'interface sur papier
2. **StatelessWidget** → Pas besoin d'état, juste affichage
3. **Navigation** → `Navigator.pushNamed()` vers les autres écrans
4. **Traductions** → `AppLocalizations.of(context)`

⚙️ ÉTAPE 5 : Configuration Quiz - `screens/quiz_setup_screen.dart`

5.1 - Défis techniques :

- Récupérer catégories depuis API
- 3 dropdowns liés
- Validation avant démarrage

5.2 - Comment résolu étape par étape :

class _QuizSetupScreenState extends State<QuizSetupScreen> {

```
// ÉTAPE 1 - Variables d'état
List<Map<String, dynamic>> categories = [];
String? selectedCategoryId;
String? selectedDifficulty;
int? selectedAmount;
bool isLoading = true;

// ÉTAPE 2 - Chargement des catégories au démarrage
@override
void initState() {
  super.initState();
  fetchCategories();
}

// ÉTAPE 3 - Appel API pour récupérer catégories
Future<void> fetchCategories() async {
  final url = Uri.parse('https://opentdb.com/api_category.php');
  final response = await http.get(url);
  final data = json.decode(response.body);

  setState(() {
    categories = List<Map<String, dynamic>>.from(data['trivia_categories']);
    isLoading = false;
  });
}

// ÉTAPE 4 - Interface avec 3 dropdowns
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Column(
      children: [
        // Dropdown catégories
        DropdownButtonFormField<String>(
          items: categories.map((cat) =>
            DropdownMenuItem(
              value: cat['id'].toString(),
              child: Text(cat['name']),
            )
          ).toList(),
          onChanged: (val) => setState(() => selectedCategoryId = val),
        ),

        // Dropdown difficulté
        DropdownButtonFormField<String>(
          items: ['easy', 'medium', 'hard'].map((d) =>
            DropdownMenuItem(value: d, child: Text(d))
          ).toList(),
```

```
          onChanged: (val) => setState(() => selectedDifficulty = val),
        ),

        // Dropdown nombre questions
        DropdownButtonFormField<int>(
          items: [5, 10, 15, 20].map((n) =>
            DropdownMenuItem(value: n, child: Text('$n'))
          ).toList(),
          onChanged: (val) => setState(() => selectedAmount = val),
        ),

        // ÉTAPE 5 - Bouton validation avec contrôle
        ElevatedButton(
          onPressed: (selectedCategoryId != null &&
                      selectedDifficulty != null &&
                      selectedAmount != null)
              ? () => Navigator.pushNamed(context, '/quiz', arguments: {
                  'category': selectedCategoryId,
                  'difficulty': selectedDifficulty,
                  'amount': selectedAmount,
                })
              : null,  // ← Bouton désactivé si pas tout sélectionné
          child: Text('Start Quiz'),
        ),
      ],
    ),
  );
 }
}
```

Progression étape par étape :

1. **Variables** → Définir quelles données on a besoin
2. **API** → Récupérer les catégories au démarrage
3. **UI** → 3 dropdowns pour sélection
4. **Validation** → Bouton activé seulement si tout sélectionné
5. **Navigation** → Passer les paramètres au quiz

🎮 ÉTAPE 6 : Le Quiz Principal - `screens/quiz_screen.dart`

6.1 - Fonctionnalités complexes à implémenter :

- Timer par question

- Sons correct/incorrect
- Vibration sur erreur
- Mélange des réponses
- Gestion du score

6.2 - Développement par fonctionnalité :

A) Timer - Comment fait :

```
// VARIABLES
Timer timer;
int timeLeft = 10;

// DÉMARRER TIMER
void _startTimer() {
  timeLeft = 10;
  timer = Timer.periodic(Duration(seconds: 1), (timer) {
    setState(() {
      timeLeft--;
      if (timeLeft <= 0) {
        timer.cancel();
        _autoNextQuestion();  // ← Passer automatiquement
      }
    });
  });
}

// AFFICHAGE TIMER
Text('Time left: $timeLeft s',
     style: TextStyle(color: Colors.red))
```

Étapes de réflexion :

1. **Besoin** → Limiter temps par question
2. **Solution** → `Timer.periodic()` chaque seconde
3. **Logique** → Si temps = 0 → question suivante auto
4. **UI** → Afficher temps restant en rouge

B) Sons - Comment fait :

```
// DÉCLARATION
final player = AudioPlayer();

// JOUER SON
Future<void> playSound(String type) async {
  try {
    await player.stop();                          // ← Arrêter son précédent
    await player.play(AssetSource('sounds/$type.mp3')); // ← Jouer nouveau
  } catch (e) {
```

```
    debugPrint('Erreur son: $e');
  }
}

// UTILISATION
if (answer == correctAnswer) {
  score++;
  if (widget.settings.enableSound) {          // ← Vérifier si son activé
    await playSound('right');                 // ← Son victoire
  }
} else {
  if (widget.settings.enableSound) {
    await playSound('wrong');                 // ← Son échec
  }
}
```

Étapes de développement :

1. **Assets** → Ajouter `right.mp3` et `wrong.mp3` dans `assets/sounds/`
2. **Package** → `audioplayers` pour lecture
3. **Logique** → Son différent selon bonne/mauvaise réponse
4. **Paramètres** → Respecter choix utilisateur (son on/off)

C) Vibration - Comment fait :

```
Future<void> vibrate() async {
  try {
    if (await Vibration.hasVibrator() ?? false) {  // ← Vérifier si vibreur existe
      Vibration.vibrate(duration: 300);           // ← Vibrer 300ms
    }
  } catch (e) {
    debugPrint('Erreur vibration: $e');
  }
}

// UTILISATION - seulement sur mauvaise réponse
if (answer != correctAnswer) {
  await vibrate();
}
```

D) Mélange réponses - Comment fait :

```
List<String> _getShuffledAnswers(Map question) {
  final List<String> answers = List<String>.from(question['incorrect_answers']); // ← 3
mauvaises réponses
  answers.add(question['correct_answer']);  // ← Ajouter bonne réponse
  answers.shuffle(Random());                // ← Mélanger tout
  return answers;                           // ← Retourner 4 réponses mélangées
```

}

Étapes de réflexion :

1. **Problème** → API donne réponse correcte séparée
2. **Solution** → Combiner tout et mélanger
3. **Résultat** → 4 boutons dans ordre aléatoire

E) Interface colorée - Comment fait :

```
// DANS LE BUILD DES BOUTONS
...answers.map((ans) {
  final isCorrect = ans == correct;
  final isSelected = ans == selectedAnswer;

  Color? bgColor;
  if (answered) {                    // ← Après avoir répondu
    if (isCorrect) bgColor = Colors.green;        // ← Bonne réponse = vert
    else if (isSelected) bgColor = Colors.red;    // ← Ma réponse fausse = rouge
    else bgColor = Colors.grey[300];              // ← Autres = gris
  }

  return ElevatedButton(
    style: ElevatedButton.styleFrom(backgroundColor: bgColor ?? Colors.blue),
    onPressed: answered ? null : () => checkAnswer(ans), // ← Désactiver après réponse
    child: Text(unescape.convert(ans)),  // ← Décoder HTML
  );
})
```

💾 ÉTAPE 7 : Stockage Local - `services/local_storage_service.dart`

7.1 - Besoin :

Sauvegarder les scores pour le classement

7.2 - Comment fait :

```
class LocalStorageService {
  static const String _key = 'quiz_scores';

  // SAUVEGARDER UN SCORE
  static Future<void> saveScore({
    required String category,
    required String difficulty,
```

```
    required int score,
    required int total,
  }) async {
    final prefs = await SharedPreferences.getInstance();

    // Créer objet score
    final scoreEntry = {
      'category': category,
      'difficulty': difficulty,
      'score': score,
      'total': total,
      'timestamp': DateTime.now().toIso8601String(),
    };

    // Récupérer scores existants
    final List<String> scores = prefs.getStringList(_key) ?? [];

    // Ajouter nouveau score
    scores.add(json.encode(scoreEntry));

    // Sauvegarder
    await prefs.setStringList(_key, scores);
  }

  // CHARGER TOUS LES SCORES
  static Future<List<Map<String, dynamic>>> loadScores() async {
    final prefs = await SharedPreferences.getInstance();
    final List<String> scores = prefs.getStringList(_key) ?? [];
    return scores.map((e) => json.decode(e) as Map<String, dynamic>).toList();
  }
}
```

Étapes de développement :

1. **Choix** → SharedPreferences pour simple et efficace
2. **Format** → JSON pour flexibilité
3. **Clé** → Une clé fixe 'quiz_scores'
4. **Structure** → Liste de maps avec toutes les infos

🌍 ÉTAPE 8 : Multilingue - Fichiers l10n

8.1 - Configuration pubspec.yaml :

```
flutter:
  generate: true              # ← IMPORTANT
```

```
flutter_gen:
  localization:
    arb-dir: lib/l10n
    template-arb-file: app_en.arb
```

8.2 - Fichier de base `lib/l10n/app_en.arb` :

```
{
  "appTitle": "Quiz App",
  "startQuiz": "Start Quiz",
  "settings": "Settings",
  "question": "Question",
  "timeLeft": "Time left",
  "yourScore": "Your score",
  "retry": "Retry"
}
```

8.3 - Traduction française `lib/l10n/app_fr.arb` :

```
{
  "appTitle": "Application Quiz",
  "startQuiz": "Commencer Quiz",
  "settings": "Paramètres",
  "question": "Question",
  "timeLeft": "Temps restant",
  "yourScore": "Votre score",
  "retry": "Recommencer"
}
```

8.4 - Utilisation dans le code :

```
final loc = AppLocalizations.of(context)!;
Text(loc.appTitle)        // ← Automatiquement traduit
Text(loc.startQuiz)       // ← Selon langue choisie
```

Progression :

1. **Setup** → Configuration dans pubspec.yaml
2. **Fichiers** → Un .arb par langue
3. **Génération** → Flutter génère classes automatiquement
4. **Usage** → `AppLocalizations.of(context)!.clé`

🎨 ÉTAPE 9 : Thèmes - `services/theme_service.dart`

9.1 - Service simple :

```
class ThemeService {
  static const _themeKey = 'isDarkMode';

  // SAUVEGARDER CHOIX THÈME
  static Future<void> saveTheme(bool isDarkMode) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setBool(_themeKey, isDarkMode);
  }

  // CHARGER CHOIX THÈME
  static Future<bool> loadTheme() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getBool(_themeKey) ?? false;  // ← Défaut : thème clair
  }
}
```

9.2 - Integration dans main.dart :

```
class _QuizAppState extends State<QuizApp> {
  late bool isDarkMode;

  void toggleTheme() {
    setState(() => isDarkMode = !isDarkMode);
    ThemeService.saveTheme(isDarkMode);        // ← Sauvegarder choix
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(brightness: Brightness.light),      // ← Thème clair
      darkTheme: ThemeData(brightness: Brightness.dark),   // ← Thème sombre
      themeMode: isDarkMode ? ThemeMode.dark : ThemeMode.light, // ← Choix actuel
    );
  }
}
```

🔗 ÉTAPE 10 : Navigation - main.dart

10.1 - Routes nommées :

```
MaterialApp(
  routes: {
    '/setup': (context) => QuizSetupScreen(),
    '/quiz': (context) => QuizScreen(settings: settings),
    '/result': (context) => ResultScreen(score: 0, total: 0),
    '/leaderboard': (context) => LeaderboardScreen(),
  },
)
```

10.2 - Passage de paramètres :

```
// ENVOI
Navigator.pushNamed(context, '/quiz', arguments: {
  'category': selectedCategory,
  'difficulty': selectedDifficulty,
});

// RECEPTION
final args = ModalRoute.of(context)!.settings.arguments as Map<String, dynamic>;
final category = args['category'];
```

🏁 ORDRE DE DÉVELOPPEMENT CHRONOLOGIQUE

Semaine 1 - Base

1. Créer projet Flutter
2. Configurer pubspec.yaml
3. Créer structure dossiers
4. HomeScreen basique

Semaine 2 - Navigation

5. QuizSetupScreen avec API catégories
6. Navigation entre écrans
7. QuizScreen basique (sans timer/son)

Semaine 3 - Fonctionnalités

8. Timer dans QuizScreen
9. Sons et vibrations
10. Système de score

Semaine 4 - Stockage & Polish

11. LocalStorageService pour scores

12. ResultScreen et LeaderboardScreen
13. Multilingue
14. Thèmes sombre/clair
15. SettingsScreen

Semaine 5 - Finitions

16. Gestion d'erreurs
17. UX (couleurs boutons, animations)
18. Tests et debug

💡 CONSEIL POUR LA SOUTENANCE

Quand on te demande "Comment vous avez fait X ?"

Réponds en 3 parties :

1. **Le problème** → "Je devais permettre à l'utilisateur de..."
2. **La solution technique** → "J'ai utilisé le package Y parce que..."
3. **L'implémentation** → "Concrètement, j'ai créé une fonction qui..."

Exemple pour les sons :

1. **Problème** → "Je voulais un feedback audio pour les bonnes/mauvaises réponses"
2. **Solution** → "J'ai utilisé audioplayers car il support les assets locaux"
3. **Implémentation** → "J'ai une fonction playSound() qui joue right.mp3 ou wrong.mp3 selon la réponse"

Tu maîtrises ton projet ! 🚀