Présentation du Projet: Application Quiz

Hamza Boubaker

April 25, 2025

Présentation du Projet

L'application de quiz permet aux utilisateurs de répondre à des questions et de tester leurs connaissances en langage Java. Elle est développée avec **Flutter** pour le frontend, utilisant l'architecture **MVVM** (Model-View-ViewMode), et une API **Spring Boot** pour le backend, qui se connecte à **MongoDB** pour la gestion des données.

Technologies Utilisées

- **Frontend:** Flutter (Architecture MVVM)
- **Backend:** Spring Boot (API REST)
- **Base de données:** MongoDB
- **Communication:** HTTP pour la communication entre Flutter et Spring Boot

Modèle des Collections MongoDB

- **Questions:** Collection contenant les questions, les réponses possibles et la bonne réponse.
- **Results:** Collection pour enregistrer les utilisateurs et leurs scores.

Exemple de structure pour la collection :

Structure MongoDB Questions

```
{
     "_id": ObjectId("..."),
     "question": "Quel mot-clé est utilisé pour hériter d'une classe en Java ?",
    "options": [
                "implement",
                "inherit",
                "extends",
                "super"
                ],
    "answer": 2
}
```

MongoDB Results

```
{
    "_id": ObjectId(".."),
    "joueur": "hamza",
    "score": 10,
    "date": "2025-04-25T10:54:10.035630"
}
```

Captures d'Écran de l'Application

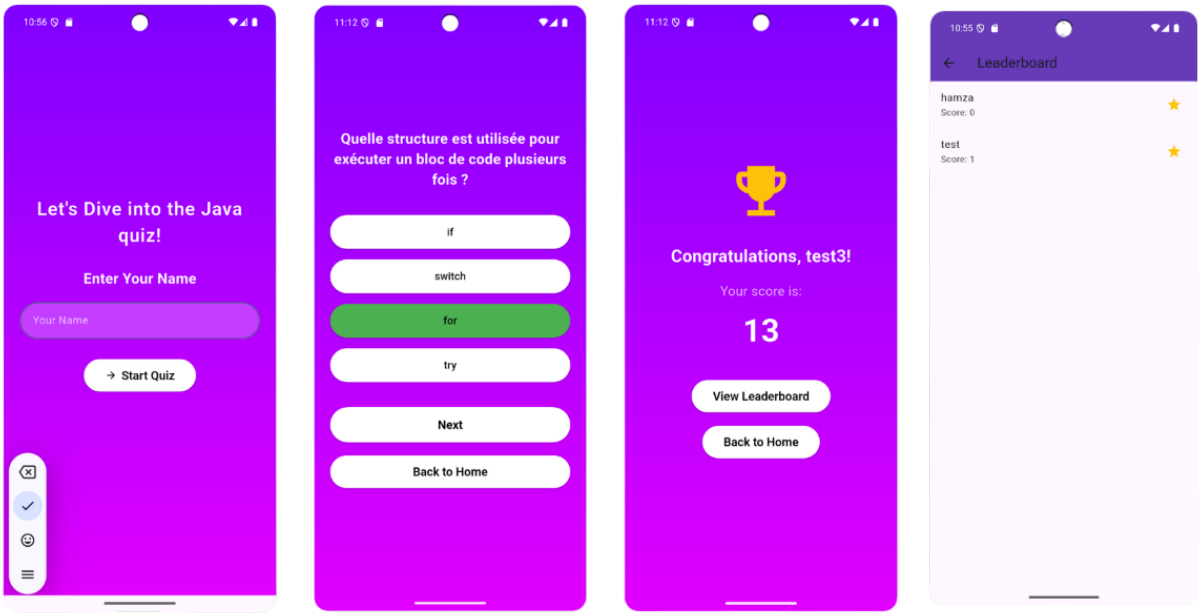

Figure 1: les différentes pages

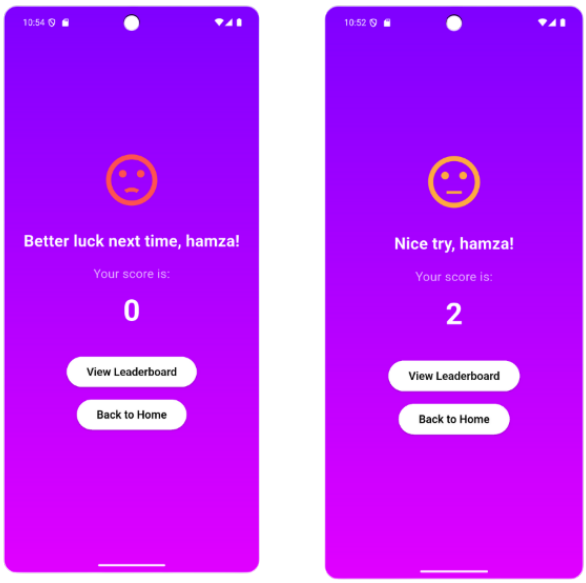

Figure 2: Score pages

Explication de l'Architecture

Frontend

Le frontend est développé avec **Flutter**, utilisant l'architecture **MVVM**. Le **Modèle** gère les données, la **Vue** est responsable de l'interface utilisateur, et le **ViewModel** gère la logique métier. Le frontend consomme les données via l'**API** REST exposée par le **backend**.

Backend

Le backend est une API REST construite avec **Spring Boot**. Elle gère les requêtes des utilisateurs et interagit avec **MongoDB** pour récupérer ou enregistrer des données.

Communication

La communication entre le frontend et le backend se fait via des requêtes HTTP. Le frontend envoie des demandes pour obtenir des questions de quiz et soumettre des scores.

Partie du Code Source Commentée

Exemple de code backend (Spring Boot):

Code Backend - Spring Boot

```
@GetMapping("/questions")
public List<Map<String, Object>> getQuestions() {
    List<?> rawList = mongoTemplate.findAll(Map.class, "
       questions");
    List<Map<String, Object>> questions = new ArrayList<>();
    for (Object obj : rawList) {
        if (obj instanceof Map<?, ?> map) {
            @SuppressWarnings("unchecked")
            Map<String, Object> question = (Map<String, Object
               >) map;
            question.put("_id", ((ObjectId) question.get("_id")
               ).toString());

            // Convert answer to index
            Object answerObj = question.get("answer");
            Object optionsObj = question.get("options");

            if (answerObj instanceof String && optionsObj
               instanceof List) {
                List<String> options = (List<String>)
                   optionsObj;
                int index = options.indexOf((String) answerObj)
                   ;
                question.put("answer", index);
            }

            questions.add(question);
        }
    }
    return questions;
}
```

Exemple de code frontend (Flutter):

Code Frontend - Flutter

```
class QuizScreen extends StatefulWidget {
  final String playerName;

  const QuizScreen({Key? key, required this.playerName}) : super(
     key: key);

  @override
  _QuizScreenState createState() => _QuizScreenState();
}

class _QuizScreenState extends State<QuizScreen> {
  @override
  void initState() {
    super.initState();
    context.read<QuizViewModel>().fetchQuestions();
  }

  @override
  Widget build(BuildContext context) {
    final quizViewModel = context.watch<QuizViewModel>();

    return Scaffold(
      body: Container(
        width: double.infinity,
        height: double.infinity,
        decoration: const BoxDecoration(
          gradient: LinearGradient(
            colors: [Color(0xFF7F00FF), Color(0xFFE100FF)],
            begin: Alignment.topCenter,
            end: Alignment.bottomCenter,
          ),
        ),
        child: quizViewModel.questions.isEmpty
            ? const Center(child: CircularProgressIndicator(color:
               Colors.white))
            : Consumer<QuizViewModel>(
          builder: (context, viewModel, child) {
            final question = viewModel.questions[viewModel.
               currentQuestionIndex];

            return Center(
              child: Padding(
                padding: const EdgeInsets.symmetric(horizontal:
                   24.0),
                child: Column(
                    .....
                      ],
```

Conclusion

Le projet d'application de quiz vise à fournir une plateforme interactive et engageante pour les utilisateurs souhaitant tester leurs connaissances en Java. Grâce à l'utilisation de **Flutter** pour le frontend et de **Spring Boot** pour le backend, l'application offre une expérience fluide et performante. L'intégration avec **MongoDB** permet une gestion efficace des données, et l'architecture **MVVM** assure une séparation claire des responsabilités, favorisant une meilleure maintenabilité et évolutivité du projet.

Les utilisateurs peuvent ainsi répondre à des questions, suivre leurs scores, et tester leurs compétences en programmation de manière dynamique. Ce projet a également permis d'appliquer et de renforcer des compétences en développement mobile, backend, ainsi qu'en gestion de bases de données.