Projet Intégré

QuizAcademy: Application mobile avec architecture microservices

Dr. El Hadji Bassirou TOURE
Département de Mathématiques et Informatique
Faculté des Sciences et Techniques
Université Cheikh Anta Diop

2025

Résumé

Ce projet a pour but d'intégrer les compétences acquises en développement mobile avec Flutter et en architectures logicielles modernes à base de microservices. Les étudiants construiront une application de réseau social académique de type questions/réponses (similaire à Quora) pour leur établissement. Le projet implique le développement d'une application mobile Flutter et de deux microservices backend (Java/Spring Boot et Node.js/Express) déployables via Docker Compose, offrant ainsi une expérience complète de développement full-stack moderne.

Ressources pour le projet

Note importante : Pour faciliter la réalisation de ce projet, l'ensemble des composants avec leur structure de base est disponible sur GitHub à l'adresse suivante :

— Code de départ complet : `https://github.com/elbachir67/quizacademy-starter.git`

Ce dépôt contient tous les services nécessaires (backend et frontend) avec leur structure de base. Des détails complets sur l'installation et l'utilisation de ces ressources sont fournis dans la section **"Ressources complémentaires et implémentation pratique"** à la fin de ce document.

Note sur les extraits de code

Important : Les extraits de code présentés dans ce document ont été délibérément simplifiés pour se concentrer sur les aspects essentiels du projet. Les codes complets et fonctionnels sont disponibles dans le dépôt GitHub référencé ci-dessus.
Cependant, certaines parties du code contiennent intentionnellement des sections marquées avec des identifiants **TODO** que vous devrez compléter pour mettre en pratique les concepts étudiés en cours. Ces identifiants (comme **TODO-USER1**, **TODO-FL1**, etc.) sont uniques et correspondent aux exercices décrits dans ce document.

Table des matières

1 Introduction et objectifs

Concept fondamental

Ce projet vise à appliquer concrètement les principes théoriques des microservices et du développement mobile vus en cours. Vous développerez une application de partage de connaissances académiques nommée QuizAcademy, permettant aux étudiants et enseignants de poser des questions et d'y répondre, dans un format similaire à Quora mais adapté au contexte universitaire.

1.1 Objectifs d'apprentissage

À la fin de ce projet, vous serez en mesure de :

- Concevoir et développer une application mobile avec Flutter qui communique avec des API REST
- Implémenter des microservices avec différentes technologies (Java/Spring Boot et Node.js/Express)
- Mettre en place une communication efficace entre les services
- Gérer l'authentification et les autorisations dans un système distribué
- Utiliser Docker et Docker Compose pour containeriser et orchestrer les services
- Implémenter un design responsive et une expérience utilisateur cohérente sur mobile
- Travailler avec une base de code existante et la faire évoluer

1.2 Vue d'ensemble de l'application

L'application QuizAcademy est une plateforme de partage de connaissances avec les fonctionnalités suivantes :

- Inscription et authentification des utilisateurs
- Gestion des profils utilisateurs
- Création et consultation de questions catégorisées par matière/discipline
- Possibilité de répondre aux questions
- Système de vote pour les réponses
- Recherche de questions et réponses

L'application sera composée de trois composants principaux :

1. **Frontend Mobile** (Flutter) : Interface utilisateur mobile
2. **Service Utilisateurs** (Java/Spring Boot) : Gestion des utilisateurs, authentification et profils
3. **Service Content** (Node.js/Express) : Gestion des questions, réponses et votes

Chaque service aura sa propre base de données et sera déployé indépendamment, illustrant ainsi le principe de polyglot programming et l'autonomie des services.

Intuition

Cette architecture illustre comment différents composants d'une application peuvent être développés et déployés indépendamment, tout en travaillant ensemble pour offrir une expérience cohérente à l'utilisateur final. Le frontend mobile ne communique qu'avec les APIs exposées, sans se soucier de la façon dont les données sont stockées ou traitées en coulisses.

FIGURE 1 – Architecture de l'application QuizAcademy

2 Environnement de travail et prérequis

2.1 Outils et technologies nécessaires

Pour réaliser ce projet, vous aurez besoin des éléments suivants :

— **Flutter SDK** pour le développement de l'application mobile
— **JDK 17+** pour le service Utilisateurs
— **Node.js 18+** pour le service Content
— **Docker et Docker Compose** pour l'orchestration
— **Git** pour la gestion du code source
— Un IDE adapté à chaque technologie (**Android Studio/VS Code** pour Flutter, **IntelliJ/Eclipse** pour Java, **VS Code** pour Node.js)
— **Postman** ou équivalent pour tester les API REST

2.2 Structure du projet

Le projet sera organisé selon la structure suivante :

- `quizacademy/`
  - `backend/`
    - `user-service/`
      - `src/`
      - `pom.xml`
      - `Dockerfile`
    - `content-service/`
      - `src/`
      - `package.json`
      - `Dockerfile`
    - `docker-compose.yml`
  - `mobile/`
    - `lib/`
    - `pubspec.yaml`
    - `Dockerfile`
  - `README.md`

Point clé à retenir

Cette structure multi-technologie illustre comment différentes équipes peuvent travailler sur des composants distincts en utilisant les technologies les plus adaptées à leurs besoins. Cela renforce la compréhension des principes de séparation des responsabilités et d'indépendance des composants.

3 Partie 1 : Service Utilisateurs (Java/Spring Boot)

3.1 Présentation du service

Le service Utilisateurs sera responsable de :

— La gestion des comptes utilisateurs (inscription, mise à jour, suppression)

— L'authentification et la génération de tokens JWT

— La gestion des profils utilisateurs (informations personnelles, préférences)

Ce service sera développé en Java avec Spring Boot, offrant robustesse et sécurité.

3.2 Modèle de données

User.java - ModÃÍle d'utilisateur (extrait)

```
package com.quizacademy.userservice.model;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String password;

    @Column(name = "profile_picture")
    private String profilePicture;

    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(name = "user_roles", joinColumns = @JoinColumn(name = "user_id"))
    @Column(name = "role")
    private Set<String> roles = new HashSet<>();

    @Column(nullable = false)
    private LocalDateTime createdAt;

    // TODO-USER1: Ajouter les getters et setters pour tous les attributs
    // Assurez-vous d'initialiser createdAt dans les constructeurs

    // Constructeurs, getters et setters
}
```

3.3 Service d'authentification

AuthService.java - Service d'authentification (extrait)

```
package com.quizacademy.userservice.service;

import com.quizacademy.userservice.model.User;
import com.quizacademy.userservice.repository.UserRepository;
import com.quizacademy.userservice.security.JwtTokenProvider;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

@Service
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtTokenProvider jwtTokenProvider;
    private final AuthenticationManager authenticationManager;

    @Autowired
    public AuthService(UserRepository userRepository, PasswordEncoder passwordEncoder,
                       JwtTokenProvider jwtTokenProvider, AuthenticationManager authenticationManager) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
        this.jwtTokenProvider = jwtTokenProvider;
        this.authenticationManager = authenticationManager;
    }

    // TODO-USER2: Implementer la methode d'inscription
    // Cette methode doit :
    // - Verifier si l'utilisateur existe deja (email ou username)
    // - Encoder le mot de passe avec passwordEncoder
    // - Assigner le role "ROLE_USER" par defaut
    // - Initialiser createdAt a la date actuelle
    // - Sauvegarder l'utilisateur et retourner l'objet cree
    public User register(User user) {
        // A implementer
    }

    // TODO-USER3: Implementer la methode d'authentification
    // Cette methode doit :
    // - Authentifier l'utilisateur avec authenticationManager
    // - Generer un token JWT avec jwtTokenProvider
    // - Retourner une Map contenant le token et les infos utilisateur
    public Map<String, Object> login(String username, String password) {
        // A implementer
    }
}
```

3.4 Contrôleur REST

AuthController.java - API REST pour l'authentification (extrait)

```
package com.quizacademy.userservice.controller;

import com.quizacademy.userservice.dto.LoginRequest;
import com.quizacademy.userservice.model.User;
import com.quizacademy.userservice.service.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;

@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "*")
public class AuthController {

    private final AuthService authService;

    @Autowired
    public AuthController(AuthService authService) {
        this.authService = authService;
    }

    // TODO-USER4: Implementer l'endpoint d'inscription
    // Cet endpoint doit :
    // - Appeler authService.register avec l'utilisateur recu
    // - Retourner un code 201 CREATED avec l'utilisateur cree (sans le mot de passe)
    // - Gerer les erreurs possibles (ex: email deja utilise)
    @PostMapping("/register")
    public ResponseEntity<?> registerUser(@RequestBody User user) {
        // A implementer
    }

    // TODO-USER5: Implementer l'endpoint de connexion
    // Cet endpoint doit :
    // - Appeler authService.login avec username et password
    // - Retourner le token et les infos utilisateur avec un code 200 OK
    // - Gerer les erreurs d'authentification
    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest) {
        // A implementer
    }
}
```

Tâche à réaliser

Exercice 1.1 : Implémentation du modèle User (TODO-USER1)
Complétez la classe `User.java` dans le service Utilisateurs pour :

- Ajouter les getters et setters pour tous les attributs
- Créer un constructeur par défaut qui initialise `createdAt` à la date et heure actuelles
- Créer un constructeur avec paramètres (username, email, password) qui initialise également `createdAt`

Solution pour User.java (TODO-USER1)

```
// Getters et setters
public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getProfilePicture() {
    return profilePicture;
}

public void setProfilePicture(String profilePicture) {
    this.profilePicture = profilePicture;
}

public Set<String> getRoles() {
    return roles;
}

public void setRoles(Set<String> roles) {
    this.roles = roles;
}

public LocalDateTime getCreatedAt() {
    return createdAt;
}

public void setCreatedAt(LocalDateTime createdAt) {
    this.createdAt = createdAt;
}

// Constructeurs
public User() {
    this.createdAt = LocalDateTime.now();
}

public User(String username, String email, String password) {
    this.username = username;
    this.email = email;
    this.password = password;
    this.createdAt = LocalDateTime.now();
    this.roles = new HashSet<>(Collections.singletonList("ROLE_USER"));
}
```

Tâche à réaliser

Exercice 1.2 : Implémentation du service d'inscription (TODO-USER2)
Implémentez la méthode `register` dans `AuthService.java` pour :

- Vérifier si l'utilisateur existe déjà (email ou username)
- Encoder le mot de passe avec passwordEncoder
- Assigner le rôle "ROLE_USER" par défaut
- Initialiser createdAt à la date actuelle
- Sauvegarder l'utilisateur et retourner l'objet créé

Solution pour AuthService.java (TODO-USER2)

```
public User register(User user) {
    // Verify if user already exists
    if (userRepository.existsByUsername(user.getUsername())) {
        throw new RuntimeException("Username is already taken");
    }

    if (userRepository.existsByEmail(user.getEmail())) {
        throw new RuntimeException("Email is already in use");
    }

    // Encode password
    user.setPassword(passwordEncoder.encode(user.getPassword()));

    // Set default role
    user.setRoles(new HashSet<>(Collections.singletonList("ROLE_USER")));

    // Set creation date
    user.setCreatedAt(LocalDateTime.now());

    // Save user
    return userRepository.save(user);
}
```

Tâche à réaliser

Exercice 1.3 : Implémentation du service d'authentification (TODO-USER3)
Implémentez la méthode `login` dans `AuthService.java` pour :

- Authentifier l'utilisateur avec authenticationManager
- Générer un token JWT avec jwtTokenProvider
- Retourner une Map contenant le token et les infos utilisateur

Solution pour AuthService.java (TODO-USER3)

```
public Map<String, Object> login(String username, String password) {
    try {
        // Authenticate user
        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

        // Find user
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new RuntimeException("User not found");
        }

        // Generate token
        String token = jwtTokenProvider.createToken(username, user.getRoles());

        // Prepare response
        Map<String, Object> response = new HashMap<>();
        response.put("token", token);
        response.put("user", user);

        return response;
    } catch (AuthenticationException e) {
        throw new RuntimeException("Invalid username/password");
    }
}
```

Tâche à réaliser

Exercice 1.4 : Implémentation du contrôleur d'inscription (TODO-USER4)
Implémentez la méthode `registerUser` dans `AuthController.java` pour :

- Appeler authService.register avec l'utilisateur reçu
- Retourner un code 201 CREATED avec l'utilisateur créé (sans le mot de passe)
- Gérer les erreurs possibles (ex : email déjà utilisé)

Solution pour AuthController.java (TODO-USER4)

```
@PostMapping("/register")
public ResponseEntity<?> registerUser(@RequestBody User user) {
    try {
        User createdUser = authService.register(user);

        // Don't return password in response
        createdUser.setPassword(null);

        return new ResponseEntity<>(createdUser, HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(Map.of("error", e.getMessage()), HttpStatus.BAD_REQUEST);
    }
}
```

Tâche à réaliser

Exercice 1.5 : Implémentation du contrôleur de connexion (TODO-USER5)
Implémentez la méthode `login` dans `AuthController.java` pour :

— Appeler authService.login avec username et password

— Retourner le token et les infos utilisateur avec un code 200 OK

— Gérer les erreurs d'authentification

Solution pour AuthController.java (TODO-USER5)

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest) {
    try {
        Map<String, Object> result = authService.login(loginRequest.getUsername(), loginRequest.getPassword());

        // Remove password from user object
        User user = (User) result.get("user");
        user.setPassword(null);

        return ResponseEntity.ok(result);
    } catch (Exception e) {
        return new ResponseEntity<>(Map.of("error", e.getMessage()), HttpStatus.UNAUTHORIZED);
    }
}
```

3.5 Checkpoint - Service Utilisateurs

Vérification du service Utilisateurs avec Postman

Une fois que vous avez complété les TODOs USER1 à USER5, vérifiez le fonctionnement du service Utilisateurs avec Postman :

1. **Démarrer le service** :
   - Avec Docker : `docker-compose up -d user-service`
2. **Créer un utilisateur** :
   - Créez une nouvelle requête POST dans Postman
   - URL : `http://localhost:8080/api/auth/register`
   - Headers : `Content-Type: application/json`
   - Body (raw, JSON) :

```
{
  "username": "user1",
  "email": "user1@example.com",
  "password": "password123"
}
```

   - Cliquez sur Send
3. **Vérifier la création** : Le service devrait répondre avec les détails de l'utilisateur créé et un statut 201
4. **Tester la connexion** :
   - Créez une nouvelle requête POST dans Postman
   - URL : `http://localhost:8080/api/auth/login`
   - Headers : `Content-Type: application/json`
   - Body (raw, JSON) :

```
{
  "username": "user1",
  "password": "password123"
}
```

   - Cliquez sur Send
5. **Vérifier l'authentification** : Le service devrait répondre avec un token JWT et les détails de l'utilisateur
6. **Tester avec des identifiants invalides** : Modifiez la requête de connexion avec un mot de passe incorrect et vérifiez que vous recevez une erreur 401

Points à vérifier :

- La création d'utilisateur fonctionne correctement avec validation
- Les mots de passe sont correctement encodés
- L'authentification fonctionne et génère des tokens JWT valides
- Les erreurs sont gérées proprement avec des messages clairs

Si tous ces tests réussissent, votre service Utilisateurs est prêt et vous pouvez passer à la partie suivante.

4 Partie 2 : Service Content (Node.js/Express)

4.1 Présentation du service

Le service Content sera responsable de :

- La gestion des questions (création, mise à jour, suppression)
- La gestion des réponses aux questions
- Le système de vote pour les réponses
- La catégorisation des questions par matière

Ce service sera développé en Node.js avec Express, offrant flexibilité et performance.

4.2 Modèle de données

models/question.model.js - ModÃÍle de question

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// TODO-CONTENT1: Definir le schema de Question
// Le schema doit contenir les champs suivants :
// - title : titre de la question (obligatoire)
// - content : contenu detaille de la question (obligatoire)
// - authorId : ID de l'utilisateur qui a pose la question (obligatoire)
// - authorName : nom d'utilisateur de l'auteur (obligatoire)
// - categoryId : ID de la categorie (obligatoire)
// - tags : tableau de tags (optionnel)
// - viewCount : nombre de vues (defaut: 0)
// - createdAt : date de creation (auto)
// - updatedAt : date de mise a jour (auto)

const QuestionSchema = new Schema(
  {
    // A implementer
  },
  {
    timestamps: true
  }
);

// Indexes pour ameliorer les performances des requetes
QuestionSchema.index({ title: 'text', content: 'text' });
QuestionSchema.index({ authorId: 1 });
QuestionSchema.index({ categoryId: 1 });
QuestionSchema.index({ tags: 1 });

module.exports = mongoose.model('Question', QuestionSchema);
```

models/answer.model.js - ModÃÍle de rÃľponse

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// TODO-CONTENT2: Definir le schema de Answer
// Le schema doit contenir les champs suivants :
// - questionId : reference a la question (obligatoire)
// - content : contenu de la reponse (obligatoire)
// - authorId : ID de l'utilisateur qui a repondu (obligatoire)
// - authorName : nom d'utilisateur de l'auteur (obligatoire)
// - votes : tableau d'objets contenant userId et vote (+1 ou -1)
// - score : score total des votes (defaut: 0)
// - createdAt : date de creation (auto)
// - updatedAt : date de mise a jour (auto)

const AnswerSchema = new Schema(
  {
    // A implementer
  },
  {
    timestamps: true
  }
);

// Indexes
AnswerSchema.index({ questionId: 1 });
AnswerSchema.index({ authorId: 1 });
AnswerSchema.index({ score: -1 });

module.exports = mongoose.model('Answer', AnswerSchema);
```

4.3 Contrôleur des questions

controllers/question.controller.js - ContrÃ´leur pour la gestion des questions (extrait)

```
const Question = require('../models/question.model');
const Answer = require('../models/answer.model');
const userService = require('../services/user.service');

// TODO-CONTENT3: Implementer la fonction de creation de question
// Cette fonction doit :
// 1. Extraire les donnees de la requete (title, content, categoryId, tags)
// 2. Verifier que l'utilisateur existe via userService.verifyToken
// 3. Creer la question en base de donnees
// 4. Retourner la question creee avec un statut 201
exports.createQuestion = async (req, res) => {
  try {
    // A implementer
  } catch (error) {
    console.error('Error creating question:', error);
    res.status(500).json({ error: 'Failed to create question' });
  }
};

// TODO-CONTENT4: Implementer la fonction de recuperation des questions par categorie
// Cette fonction doit :
// 1. Extraire l'ID de categorie des parametres de route
// 2. Recuperer les questions filtrees par categorie
// 3. Trier par date de creation (plus recentes d'abord)
// 4. Paginer les resultats (utiliser req.query.page et req.query.limit)
exports.getQuestionsByCategory = async (req, res) => {
  try {
    // A implementer
  } catch (error) {
    console.error('Error fetching questions by category:', error);
    res.status(500).json({ error: 'Failed to fetch questions' });
  }
};

// Les autres fonctions du controleur sont deja implementees...
```

4.4 Contrôleur des réponses

controllers/answer.controller.js - ContrÃ´leur pour la gestion des rÃ©ponses (extrait)

```
const Answer = require('../models/answer.model');
const Question = require('../models/question.model');
const userService = require('../services/user.service');

// TODO-CONTENT5: Implementer la fonction de creation de reponse
// Cette fonction doit :
// 1. Extraire les donnees de la requete (content) et l'ID de question des parametres
// 2. Verifier que l'utilisateur existe via userService.verifyToken
// 3. Verifier que la question existe
// 4. Creer la reponse en base de donnees
// 5. Retourner la reponse creee avec un statut 201
exports.createAnswer = async (req, res) => {
  try {
    // A implementer
  } catch (error) {
    console.error('Error creating answer:', error);
    res.status(500).json({ error: 'Failed to create answer' });
  }
};

// TODO-CONTENT6: Implementer la fonction de vote pour une reponse
// Cette fonction doit :
// 1. Extraire l'ID de reponse des parametres et le vote du corps (1 ou -1)
// 2. Verifier que l'utilisateur existe via userService.verifyToken
// 3. Verifier si l'utilisateur a deja vote pour cette reponse
// 4. Mettre a jour le tableau de votes et recalculer le score
// 5. Retourner la reponse mise a jour
exports.voteAnswer = async (req, res) => {
  try {
    // A implementer
  } catch (error) {
    console.error('Error voting for answer:', error);
    res.status(500).json({ error: 'Failed to vote for answer' });
  }
};

// Les autres fonctions du controleur sont deja implementees...
```

Tâche à réaliser

Exercice 2.1 : Implémentation du modèle Question (TODO-CONTENT1)
Complétez le schema `QuestionSchema` dans `models/question.model.js` pour définir tous les champs requis avec leurs types et contraintes.

Solution pour models/question.model.js (TODO-CONTENT1)

```
const QuestionSchema = new Schema(
  {
    title: {
      type: String,
      required: true,
      trim: true,
      minlength: 5,
      maxlength: 150
    },
    content: {
      type: String,
      required: true,
      trim: true,
      minlength: 10
    },
    authorId: {
      type: String,
      required: true
    },
    authorName: {
      type: String,
      required: true
    },
    categoryId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Category',
      required: true
    },
    tags: {
      type: [String],
      default: []
    },
    viewCount: {
      type: Number,
      default: 0
    }
  },
  {
    timestamps: true
  }
);
```

Tâche à réaliser

Exercice 2.2 : Implémentation du modèle Answer (TODO-CONTENT2)
Complétez le schema `AnswerSchema` dans `models/answer.model.js` pour définir tous les champs requis avec leurs types et contraintes.

Solution pour models/answer.model.js (TODO-CONTENT2)

```
const AnswerSchema = new Schema(
  {
    questionId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Question',
      required: true
    },
    content: {
      type: String,
      required: true,
      trim: true,
      minlength: 5
    },
    authorId: {
      type: String,
      required: true
    },
    authorName: {
      type: String,
      required: true
    },
    votes: {
      type: [{
        userId: String,
        vote: { type: Number, enum: [1, -1] }
      }],
      default: []
    },
    score: {
      type: Number,
      default: 0
    }
  },
  {
    timestamps: true
  }
);
```

Tâche à réaliser

Exercice 2.3 : Implémentation de la création de question (TODO-CONTENT3)
Implémentez la fonction `createQuestion` dans `controllers/question.controller.js` pour créer une nouvelle question en vérifiant l'authentification et en validant les données.

Solution pour controllers/question.controller.js (TODO-CONTENT3)

```
exports.createQuestion = async (req, res) => {
  try {
    // Extract data from request
    const { title, content, categoryId, tags } = req.body;

    // Verify authentication
    const decodedToken = userService.verifyToken(req);
    if (!decodedToken) {
      return res.status(401).json({ error: 'Unauthorized' });
    }

    // Create question
    const question = new Question({
      title,
      content,
      categoryId,
      tags: tags || [],
      authorId: decodedToken.userId,
      authorName: decodedToken.username
    });

    // Save to database
    const savedQuestion = await question.save();

    // Return with 201 Created status
    res.status(201).json(savedQuestion);
  } catch (error) {
    console.error('Error creating question:', error);
    res.status(500).json({ error: 'Failed to create question' });
  }
};
```

Tâche à réaliser

Exercice 2.4 : Implémentation de la récupération des questions par catégorie (TODO-CONTENT4)
Implémentez la fonction `getQuestionsByCategory` dans `controllers/question.controller.js` pour récupérer les questions d'une catégorie donnée avec pagination.

Solution pour controllers/question.controller.js (TODO-CONTENT4)

```
exports.getQuestionsByCategory = async (req, res) => {
  try {
    // Extract category ID from route params
    const { categoryId } = req.params;

    // Extract pagination parameters
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 10;
    const skip = (page - 1) * limit;

    // Query questions
    const questions = await Question.find({ categoryId })
      .sort({ createdAt: -1 })
      .skip(skip)
      .limit(limit);

    // Count total questions for pagination info
    const totalQuestions = await Question.countDocuments({ categoryId });

    // Return questions with pagination metadata
    res.status(200).json({
      questions,
      pagination: {
        currentPage: page,
        totalPages: Math.ceil(totalQuestions / limit),
        totalItems: totalQuestions,
        hasMore: skip + questions.length < totalQuestions
      }
    });
  } catch (error) {
    console.error('Error fetching questions by category:', error);
    res.status(500).json({ error: 'Failed to fetch questions' });
  }
};
```

Tâche à réaliser

Exercice 2.5 : Implémentation de la création de réponse (TODO-CONTENT5)
Implémentez la fonction `createAnswer` dans `controllers/answer.controller.js` pour créer une nouvelle réponse à une question en vérifiant l'authentification.

Solution pour controllers/answer.controller.js (TODO-CONTENT5)

```
exports.createAnswer = async (req, res) => {
  try {
    // Extract data
    const { content } = req.body;
    const { questionId } = req.params;

    // Verify authentication
    const decodedToken = userService.verifyToken(req);
    if (!decodedToken) {
      return res.status(401).json({ error: 'Unauthorized' });
    }

    // Check if question exists
    const questionExists = await Question.exists({ _id: questionId });
    if (!questionExists) {
      return res.status(404).json({ error: 'Question not found' });
    }

    // Create answer
    const answer = new Answer({
      questionId,
      content,
      authorId: decodedToken.userId,
      authorName: decodedToken.username,
      votes: [],
      score: 0
    });

    // Save to database
    const savedAnswer = await answer.save();

    // Return created answer
    res.status(201).json(savedAnswer);
  } catch (error) {
    console.error('Error creating answer:', error);
    res.status(500).json({ error: 'Failed to create answer' });
  }
};
```

Tâche à réaliser

Exercice 2.6 : Implémentation du système de vote (TODO-CONTENT6)
Implémentez la fonction `voteAnswer` dans `controllers/answer.controller.js` pour permettre aux utilisateurs de voter pour les réponses et mettre à jour le score.

Solution pour controllers/answer.controller.js (TODO-CONTENT6)

```
exports.voteAnswer = async (req, res) => {
  try {
    // Extract data
    const { answerId } = req.params;
    const { vote } = req.body;

    // Validate vote value
    if (vote !== 1 && vote !== -1) {
      return res.status(400).json({ error: 'Vote must be 1 or -1' });
    }

    // Verify authentication
    const decodedToken = userService.verifyToken(req);
    if (!decodedToken) {
      return res.status(401).json({ error: 'Unauthorized' });
    }

    // Find answer
    const answer = await Answer.findById(answerId);
    if (!answer) {
      return res.status(404).json({ error: 'Answer not found' });
    }

    // Check if user has already voted
    const existingVoteIndex = answer.votes.findIndex(v => v.userId === decodedToken.userId);

    if (existingVoteIndex !== -1) {
      // Remove existing vote from score
      answer.score -= answer.votes[existingVoteIndex].vote;

      // Update or remove existing vote
      if (answer.votes[existingVoteIndex].vote === vote) {
        // If same vote value, remove the vote (toggle off)
        answer.votes.splice(existingVoteIndex, 1);
      } else {
        // If different vote value, update the vote
        answer.votes[existingVoteIndex].vote = vote;
        answer.score += vote;
      }
    } else {
      // Add new vote
      answer.votes.push({ userId: decodedToken.userId, vote });
      answer.score += vote;
    }

    // Save updated answer
    const updatedAnswer = await answer.save();

    // Return updated answer
    res.status(200).json(updatedAnswer);
  } catch (error) {
    console.error('Error voting for answer:', error);
    res.status(500).json({ error: 'Failed to vote for answer' });
  }
};
```

4.5 Checkpoint - Service Content

Vérification du service Content avec Postman

Après avoir complété les TODOs CONTENT1 à CONTENT6, vérifiez le fonctionnement du service Content :

1. **Assurez-vous que le service Utilisateurs est en cours d'exécution**
2. **Démarrer le service Content** :
   - Avec Docker : `docker-compose up -d content-service`
3. **Créer une catégorie** (endpoint déjà implémenté) :
   - Créez une nouvelle requête POST dans Postman
   - URL : `http://localhost:3000/api/categories`
   - Headers :
     - `Content-Type: application/json`
     - `Authorization: Bearer [VOTRE_TOKEN_JWT]`
   - Body (raw, JSON) :

```
{
  "name": "Mathématiques",
  "description": "Questions sur les mathématiques"
}
```

   - Cliquez sur Send
4. **Créer une question** :
   - Créez une nouvelle requête POST dans Postman
   - URL : `http://localhost:3000/api/questions`
   - Headers :
     - `Content-Type: application/json`
     - `Authorization: Bearer [VOTRE_TOKEN_JWT]`
   - Body (raw, JSON) :

```
{
  "title": "Comment résoudre une équation du second degré ?",
  "content": "Je n'arrive pas à appliquer la formule pour résoudre ax^2 + bx + c = 0. Pouvez-vous explic
  "categoryId": "[ID_DE_LA_CATEGORIE]",
  "tags": ["équations", "algèbre"]
}
```

   - Cliquez sur Send
5. **Récupérer les questions par catégorie** :
   - Créez une nouvelle requête GET dans Postman
   - URL : `http://localhost:3000/api/categories/[ID_DE_LA_CATEGORIE]/questions?page=1&limit=10`
   - Cliquez sur Send
6. **Créer une réponse** :
   - Créez une nouvelle requête POST dans Postman
   - URL : `http://localhost:3000/api/questions/[ID_DE_LA_QUESTION]/answers`
   - Headers :
     - `Content-Type: application/json`
     - `Authorization: Bearer [VOTRE_TOKEN_JWT]`
   - Body (raw, JSON) :

```
{
  "content": "Pour résoudre une équation du second degré ax^2 + bx + c = 0, utilisez le discriminant del
}
```

   - Cliquez sur Send
7. **Voter pour une réponse** :
   - Créez une nouvelle requête POST dans Postman
   - URL : `http://localhost:3000/api/answers/[ID_DE_LA_REPONSE]/vote`
   - Headers :
     - `Content-Type: application/json`
     - `Authorization: Bearer [VOTRE_TOKEN_JWT]`
   - Body (raw, JSON) :

```
{
  "vote": 1
}
```

   - Cliquez sur Send

Points à vérifier :

- La création de questions et réponses fonctionne correctement
- La pagination des questions par catégorie fonctionne
- Le système de vote permet d'ajouter, modifier et supprimer un vote
- Les erreurs sont bien gérées (authentification, validation)

Si tous ces tests réussissent, votre service Content est fonctionnel et vous pouvez passer à la partie Frontend.

5 Partie 3 : Application mobile Flutter

5.1 Présentation de l'application

L'application mobile Flutter permettra aux utilisateurs d'accéder à QuizAcademy depuis leurs appareils mobiles. Elle implémentera les fonctionnalités suivantes :

— Inscription et connexion des utilisateurs

— Consultation des questions par catégorie

— Visualisation des détails d'une question et ses réponses

— Possibilité de poser des questions et d'y répondre

— Système de vote pour les réponses

— Recherche de questions

— Gestion du profil utilisateur

5.2 Configuration du projet

pubspec.yaml - Configuration des dÃ‍pendances (extrait)

```
name: quizacademy
description: A Q&A platform for academic knowledge sharing.

publish_to: 'none'
version: 1.0.0+1

environment:
  sdk: ">=2.17.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.5
  provider: ^6.0.3
  shared_preferences: ^2.0.15
  intl: ^0.17.0
  flutter_markdown: ^0.6.13
  cupertino_icons: ^1.0.5

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.1

flutter:
  uses-material-design: true
  assets:
    - assets/images/
```

5.3 Services API

lib/services/auth_service.dart - Service d'authentification

```
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:shared_preferences/shared_preferences.dart';
import '../config/api_config.dart';
import '../models/user.dart';

class AuthService {
  final String baseUrl = '${ApiConfig.baseUrl}/auth';

  // TODO-FL1: Implementer la methode d'inscription
  // Cette methode doit :
  // - Faire une requete POST a /auth/register avec les donnees utilisateur
  // - Gerer les reponses de succes et d'erreur
  // - Retourner l'utilisateur cree en cas de succes
  Future<User> register(String username, String email, String password) async {
    // A implementer
  }

  // TODO-FL2: Implementer la methode de connexion
  // Cette methode doit :
  // - Faire une requete POST a /auth/login avec username et password
  // - Sauvegarder le token JWT recu dans les SharedPreferences
  // - Retourner l'utilisateur connecte
  Future<User> login(String username, String password) async {
    // A implementer
  }

  // Les autres methodes sont deja implementees...
}
```

lib/services/question_service.dart - Service de gestion des questions

```
import 'dart:convert';
import 'package:http/http.dart' as http;
import '../config/api_config.dart';
import '../models/question.dart';
import '../models/category.dart';
import 'auth_service.dart';

class QuestionService {
  final String baseUrl = '${ApiConfig.baseUrl}';
  final AuthService authService = AuthService();

  // TODO-FL3: Implementer la methode pour recuperer les questions par categorie
  // Cette methode doit :
  // - Faire une requete GET a /categories/{categoryId}/questions
  // - Gerer la pagination (parametres page et limit)
  // - Parser la reponse en liste de Question
  Future<Map<String, dynamic>> getQuestionsByCategory(String categoryId, {int page = 1, int limit = 10}) async {
    // A implementer
  }

  // TODO-FL4: Implementer la methode pour creer une question
  // Cette methode doit :
  // - Recuperer le token JWT avec authService.getToken()
  // - Faire une requete POST a /questions avec les donnees et le token
  // - Retourner la question creee
  Future<Question> createQuestion(String title, String content, String categoryId, List<String> tags) async {
    // A implementer
  }

  // Les autres methodes sont deja implementees...
}
```

5.4 Écrans principaux

lib/screens/auth/register_screen.dart - ÃLcran d'inscription (version abrÃŀgÃŀe)

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../providers/auth_provider.dart';
import 'login_screen.dart';

class RegisterScreen extends StatefulWidget {
  @override
  _RegisterScreenState createState() => _RegisterScreenState();
}

class _RegisterScreenState extends State<RegisterScreen> {
  final _formKey = GlobalKey<FormState>();
  final _usernameController = TextEditingController();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _confirmPasswordController = TextEditingController();
  bool _isLoading = false;
  String? _errorMessage;

  @override
  void dispose() {
    _usernameController.dispose();
    _emailController.dispose();
    _passwordController.dispose();
    _confirmPasswordController.dispose();
    super.dispose();
  }

  // TODO-FL5: Implementer la methode d'inscription
  // Cette methode doit :
  // - Valider le formulaire (_formKey.currentState!.validate())
  // - Verifier que les mots de passe correspondent
  // - Appeler authProvider.register avec les donnees du formulaire
  // - Gerer l'etat de chargement et les erreurs
  // - Naviguer vers l'ecran principal apres inscription reussie
  void _register() async {
    // A implementer
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Inscription'),
      ),
      body: SingleChildScrollView(
        padding: EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              if (_errorMessage != null)
                Padding(
                  padding: EdgeInsets.only(bottom: 16.0),
                  child: Text(_errorMessage!,
                    style: TextStyle(color: Colors.red),
                  ),
                ),
              // Champs de formulaire...
              // [Code réduit pour concision]
              TextFormField(
                controller: _usernameController,
                decoration: InputDecoration(
                  labelText: 'Nom d\'utilisateur',
                  border: OutlineInputBorder(),
                ),
                // Validations...
              ),
              SizedBox(height: 16),
              // Autres champs du formulaire...
              ElevatedButton(
                onPressed: _isLoading ? null : _register,
                child: _isLoading
                    ? CircularProgressIndicator(color: Colors.white)
                    : Text('S\'inscrire'),
                style: ElevatedButton.styleFrom(
                  padding: EdgeInsets.symmetric(vertical: 16.0),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

lib/screens/questions/question_list_screen.dart - ÃĽcran de liste des questions (version abrÃľgÃľe)

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../models/category.dart';
import '../../models/question.dart';
import '../../providers/category_provider.dart';
import '../../providers/question_provider.dart';
import '../../widgets/question_card.dart';
import 'question_detail_screen.dart';
import 'create_question_screen.dart';

class QuestionListScreen extends StatefulWidget {
  @override
  _QuestionListScreenState createState() => _QuestionListScreenState();
}

class _QuestionListScreenState extends State<QuestionListScreen> {
  Category? _selectedCategory;
  bool _isLoading = false;
  int _currentPage = 1;
  bool _hasMore = true;

  @override
  void initState() {
    super.initState();
    _loadCategories();
  }

  Future<void> _loadCategories() async {
    final categoryProvider = Provider.of<CategoryProvider>(context, listen: false);

    // Code réduit pour concision...
    // Charge les catégories et initialise la première catégorie
  }

  // TODO-FL6: Implementer la methode pour charger les questions
  // Cette methode doit :
  // - Verifier qu'une categorie est selectionnee
  // - Appeler questionProvider.fetchQuestionsByCategory
  // - Gerer l'etat de chargement et les erreurs
  // - Mettre a jour _hasMore selon la reponse
  Future<void> _loadQuestions({bool refresh = false}) async {
    // A implementer
  }

  @override
  Widget build(BuildContext context) {
    final categoryProvider = Provider.of<CategoryProvider>(context);
    final questionProvider = Provider.of<QuestionProvider>(context);

    return Scaffold(
      appBar: AppBar(
        title: Text('Questions'),
        actions: [
          IconButton(
            icon: Icon(Icons.search),
            onPressed: () => Navigator.pushNamed(context, '/search'),
          ),
        ],
      ),
      body: _isLoading
          ? Center(child: CircularProgressIndicator())
          : Column(
              children: [
                _buildCategorySelector(categoryProvider),
                Expanded(
                  // Liste des questions avec pagination...
                  // Code réduit pour concision
                ),
              ],
            ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () {
          // Code pour naviguer vers l'écran de création de question
        },
      ),
    );
  }

  // TODO-FL7: Implementer le widget de selection de categorie
  // Ce widget doit :
  // - Afficher un DropdownButton avec les categories disponibles
  // - Permettre de selectionner une categorie
  // - Appeler _loadQuestions quand la categorie change
  Widget _buildCategorySelector(CategoryProvider categoryProvider) {
    // A implementer
  }

  Widget _buildLoadMoreButton() {
    // Widget pour le bouton "Charger plus"
    // Code réduit pour concision
  }
}
```

Tâche à réaliser

Exercice 3.1 : Implémentation de la méthode d'inscription (TODO-FL1)
Complétez la méthode `register` dans `auth_service.dart` pour :

- Faire une requête POST à /auth/register avec les données utilisateur
- Gérer les réponses de succès et d'erreur
- Retourner l'utilisateur créé en cas de succès

Solution pour lib/services/auth_service.dart (TODO-FL1)

```
Future<User> register(String username, String email, String password) async {
  final response = await http.post(
    Uri.parse('$baseUrl/register'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({
      'username': username,
      'email': email,
      'password': password
    }),
  );

  if (response.statusCode == 201) {
    final responseData = json.decode(response.body);
    return User.fromJson(responseData);
  } else {
    final errorData = json.decode(response.body);
    throw Exception(errorData['error'] ?? 'Failed to register user');
  }
}
```

Tâche à réaliser

Exercice 3.2 : Implémentation de la méthode de connexion (TODO-FL2)
Complétez la méthode `login` dans `auth_service.dart` pour :

- Faire une requête POST à /auth/login avec username et password
- Sauvegarder le token JWT reçu dans les SharedPreferences
- Retourner l'utilisateur connecté

Solution pour lib/services/auth_service.dart (TODO-FL2)

```
Future<User> login(String username, String password) async {
  final response = await http.post(
    Uri.parse('$baseUrl/login'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({
      'username': username,
      'password': password
    }),
  );

  if (response.statusCode == 200) {
    final responseData = json.decode(response.body);

    // Save token to SharedPreferences
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('auth_token', responseData['token']);

    // Return user object
    return User.fromJson(responseData['user']);
  } else {
    final errorData = json.decode(response.body);
    throw Exception(errorData['error'] ?? 'Failed to login');
  }
}
```

Tâche à réaliser

Exercice 3.3 : Implémentation de la méthode pour récupérer les questions (TODO-FL3)

Complétez la méthode `getQuestionsByCategory` dans `question_service.dart` pour :

- Faire une requête GET à /categories/{categoryId}/questions
- Gérer la pagination (paramètres page et limit)
- Parser la réponse en liste de Question

Solution pour lib/services/question_service.dart (TODO-FL3)

```
Future<Map<String, dynamic>> getQuestionsByCategory(String categoryId, {int page = 1, int limit = 10}) async {
  final response = await http.get(
    Uri.parse('$baseUrl/categories/$categoryId/questions?page=$page&limit=$limit'),
  );

  if (response.statusCode == 200) {
    final responseData = json.decode(response.body);

    // Parse questions
    final List<Question> questions = (responseData['questions'] as List)
        .map((questionJson) => Question.fromJson(questionJson))
        .toList();

    // Return questions with pagination metadata
    return {
      'questions': questions,
      'pagination': responseData['pagination'],
    };
  } else {
    final errorData = json.decode(response.body);
    throw Exception(errorData['error'] ?? 'Failed to fetch questions');
  }
}
```

Tâche à réaliser

Exercice 3.4 : Implémentation de la méthode de création de question (TODO-FL4)

Complétez la méthode `createQuestion` dans `question_service.dart` pour :

- Récupérer le token JWT avec authService.getToken()
- Faire une requête POST à /questions avec les données et le token
- Retourner la question créée

Solution pour lib/services/question_service.dart (TODO-FL4)

```
Future<Question> createQuestion(String title, String content, String categoryId, List<String> tags) async {
  // Get auth token
  final token = await authService.getToken();
  if (token == null) {
    throw Exception('User not authenticated');
  }

  final response = await http.post(
    Uri.parse('$baseUrl/questions'),
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer $token',
    },
    body: json.encode({
      'title': title,
      'content': content,
      'categoryId': categoryId,
      'tags': tags,
    }),
  );

  if (response.statusCode == 201) {
    final responseData = json.decode(response.body);
    return Question.fromJson(responseData);
  } else {
    final errorData = json.decode(response.body);
    throw Exception(errorData['error'] ?? 'Failed to create question');
  }
}
```

Tâche à réaliser

Exercice 3.5 : Implémentation de la méthode d'inscription (TODO-FL5)
Complétez la méthode `_register` dans `register_screen.dart` pour :

- Valider le formulaire (_formKey.currentState!.validate())
- Vérifier que les mots de passe correspondent
- Appeler authProvider.register avec les données du formulaire
- Gérer l'état de chargement et les erreurs
- Naviguer vers l'écran principal après inscription réussie

Solution pour lib/screens/auth/register_screen.dart (TODO-FL5)

```
void _register() async {
  // Hide previous error messages
  setState(() {
    _errorMessage = null;
  });

  // Validate form
  if (!_formKey.currentState!.validate()) {
    return;
  }

  // Check passwords match
  if (_passwordController.text != _confirmPasswordController.text) {
    setState(() {
      _errorMessage = 'Les mots de passe ne correspondent pas';
    });
    return;
  }

  // Set loading state
  setState(() {
    _isLoading = true;
  });

  try {
    // Get the auth provider
    final authProvider = Provider.of<AuthProvider>(context, listen: false);

    // Attempt to register
    await authProvider.register(
      _usernameController.text,
      _emailController.text,
      _passwordController.text,
    );

    // Navigate to home page on success
    Navigator.pushReplacementNamed(context, '/home');
  } catch (e) {
    // Display error message
    setState(() {
      _errorMessage = e.toString();
    });
  } finally {
    // Reset loading state
    setState(() {
      _isLoading = false;
    });
  }
}
```

Tâche à réaliser

Exercice 3.6 : Implémentation du chargement des questions (TODO-FL6)
Complétez la méthode `_loadQuestions` dans `question_list_screen.dart` pour :

- Vérifier qu'une catégorie est sélectionnée
- Appeler questionProvider.fetchQuestionsByCategory
- Gérer l'état de chargement et les erreurs
- Mettre à jour _hasMore selon la réponse

Solution pour lib/screens/questions/question_list_screen.dart (TODO-FL6)

```
Future<void> _loadQuestions({bool refresh = false}) async {
  // Check if category is selected
  if (_selectedCategory == null) {
    return;
  }

  // Set loading state
  setState(() => _isLoading = true);

  try {
    // Reset page if refreshing
    if (refresh) {
      _currentPage = 1;
    }

    // Get the question provider
    final questionProvider = Provider.of<QuestionProvider>(context, listen: false);

    // Fetch questions for selected category
    final result = await questionProvider.fetchQuestionsByCategory(
      _selectedCategory!.id,
      page: _currentPage,
      refresh: refresh,
    );

    // Update pagination
    setState(() {
      _isLoading = false;
      _hasMore = result['hasMore'] ?? false;

      // Increment page for next fetch if there are more items
      if (_hasMore) {
        _currentPage++;
      }
    });
  } catch (e) {
    setState(() => _isLoading = false);

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Erreur: Impossible de charger les questions')),
    );
  }
}
```

Tâche à réaliser

Exercice 3.7 : Implémentation du sélecteur de catégorie (TODO-FL7)
Complétez la méthode `_buildCategorySelector` dans `question_list_screen.dart` pour :

- Afficher un DropdownButton avec les catégories disponibles
- Permettre de sélectionner une catégorie
- Appeler _loadQuestions quand la catégorie change

Solution pour lib/screens/questions/question_list_screen.dart (TODO-FL7)

```
Widget _buildCategorySelector(CategoryProvider categoryProvider) {
  if (categoryProvider.categories.isEmpty) {
    return SizedBox.shrink();
  }

  return Container(
    padding: EdgeInsets.all(16.0),
    color: Theme.of(context).colorScheme.surface,
    child: Row(
      children: [
        Text('Catégorie:', style: TextStyle(fontWeight: FontWeight.bold)),
        SizedBox(width: 16.0),
        Expanded(
          child: DropdownButton<Category>(
            isExpanded: true,
            value: _selectedCategory,
            items: categoryProvider.categories.map((Category category) {
              return DropdownMenuItem<Category>(
                value: category,
                child: Text(category.name),
              );
            }).toList(),
            onChanged: (Category? newValue) {
              if (newValue != null && newValue != _selectedCategory) {
                setState(() {
                  _selectedCategory = newValue;
                  _currentPage = 1; // Reset pagination
                });
                _loadQuestions(refresh: true);
              }
            },
          ),
        ),
      ],
    ),
  );
}
```

6 Déploiement avec Docker

6.1 Configuration Docker pour les services backend

backend/user-service/Dockerfile - Dockerfile pour le service Java

```
FROM maven:3.8-openjdk-17 as build
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn package -DskipTests

FROM openjdk:17-jdk-slim
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

backend/content-service/Dockerfile - Dockerfile pour le service Node.js

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

6.2 Configuration Docker Compose

docker-compose.yml - Orchestration des services

```
version: '3'

services:
  user-service:
    build: ./backend/user-service
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATASOURCE_URL=jdbc:h2:mem:userdb
      - SPRING_DATASOURCE_USERNAME=sa
      - SPRING_DATASOURCE_PASSWORD=password
      - JWT_SECRET=your_jwt_secret_key_here
    networks:
      - quizacademy-network
    restart: always

  content-service:
    build: ./backend/content-service
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - MONGODB_URI=mongodb://mongodb:27017/quizacademy
      - USER_SERVICE_URL=http://user-service:8080
      - JWT_SECRET=your_jwt_secret_key_here
    depends_on:
      - user-service
      - mongodb
    networks:
      - quizacademy-network
    restart: always

  mongodb:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db
    networks:
      - quizacademy-network

networks:
  quizacademy-network:
    driver: bridge

volumes:
  mongodb_data:
```

Tâche à réaliser

Exercice 4.1 : Building et déploiement des images Docker
Suivez ces étapes pour déployer l'application via Docker :

- Naviguer à la racine du projet
- Construire et démarrer les services avec `docker-compose up -build`
- Vérifier que tous les services démarrent correctement
- Tester l'application via les endpoints backend et l'application mobile

Assurez-vous de publier vos images sur Docker Hub :

```
# Se connecter à Docker Hub
docker login

# Taguer les images
docker tag quizacademy_user-service votrenom/quizacademy-user-service:v1
docker tag quizacademy_content-service votrenom/quizacademy-content-service:v1

# Publier les images
docker push votrenom/quizacademy-user-service:v1
docker push votrenom/quizacademy-content-service:v1
```

7 Rapport technique

En plus de l'implémentation du code, vous devez rendre un rapport technique complet (10-15 pages) qui comprend :

1. **Introduction** - Présentation du projet et de ses objectifs
2. **Architecture** - Description détaillée de l'architecture mise en place
3. **Choix techniques** - Justification des technologies utilisées et des décisions de conception
4. **Implémentation** - Explication des fonctionnalités principales implémentées
5. **Difficultés rencontrées** - Problèmes techniques et solutions adoptées
6. **Résultats** - Captures d'écran de l'application et démonstrations
7. **Améliorations futures** - Pistes d'évolution et d'amélioration
8. **Conclusion** - Synthèse et enseignements tirés
9. **Références** - Sources et documentation utilisées

Le rapport doit être rédigé en français avec une mise en page professionnelle et des illustrations appropriées.

8 Ressources complémentaires et implémentation pratique

8.1 Code source du projet

Pour faciliter la réalisation de ce projet, les codes sources de base de tous les composants sont disponibles sur GitHub :

Point d'attention

Important : Avant de commencer le développement, assurez-vous de cloner le dépôt GitHub ci-dessous et de vous familiariser avec sa structure. La compréhension du code existant est essentielle pour mener à bien les tâches d'implémentation.
Notez que le code du dépôt contient des sections marquées avec des identifiants **TODO** que vous devrez compléter, correspondant aux exercices de ce document. Ces identifiants vous guideront pour localiser précisément les parties à modifier.

Code de dÃ¯part complet

`https://github.com/elbachir67/quizacademy-starter.git`

Ce dépôt contient la structure de base de tous les composants. Pour l'installer et le configurer :

```
# Cloner le dépôt
git clone https://github.com/elbachir67/quizacademy-starter.git

# Accéder au répertoire
cd quizacademy-starter

# Explorer la structure du projet
ls -la

# Initialiser et démarrer les services backend
cd backend
docker-compose up -d
```

```
# Installer les dépendances du frontend Flutter
cd ../mobile
flutter pub get
```

Point clé à retenir

Pour travailler efficacement sur ce projet :

- Commencez par comprendre la structure globale du projet
- Travaillez sur un service à la fois, en suivant l'ordre logique : Backend (Utilisateurs → Content) → Frontend
- Testez chaque fonctionnalité de manière isolée avant de l'intégrer
- Utilisez les outils de débogage de Flutter pour identifier les problèmes
- Consultez la documentation officielle de chaque technologie en cas de besoin

9 Conclusion

Bénéfice

Ce projet vous a permis d'appliquer concrètement les connaissances acquises en développement mobile avec Flutter et en architecture logicielle à base de microservices. Vous avez conçu et implémenté une application complète avec :

- Une architecture backend distribuée avec plusieurs microservices
- Une interface mobile responsive et intuitive
- Un déploiement via conteneurs Docker
- Une compréhension des enjeux de communication entre services

Ces compétences sont très recherchées sur le marché du travail et constituent une base solide pour votre future carrière de développeur.

Point clé à retenir

Points clés à retenir :

- Le frontend mobile ne doit pas connaître les détails d'implémentation du backend
- La communication entre microservices doit être bien conçue pour minimiser le couplage
- L'architecture à base de microservices offre flexibilité et scalabilité mais ajoute de la complexité
- La containerisation facilite le déploiement et garantit la cohérence entre environnements
- Une bonne architecture est modulaire et permet de faire évoluer chaque composant indépendamment

Point d'attention

Dans un contexte professionnel, les projets de ce type nécessiteraient également :

- Des tests automatisés (unitaires, intégration, E2E)
- Des procédures CI/CD pour l'intégration et le déploiement continus
- Des stratégies de monitoring et d'observabilité
- Une documentation API plus complète (OpenAPI/Swagger)
- Des stratégies de sécurité plus avancées

Gardez ces aspects à l'esprit pour vos projets futurs.