Pour onglet revision : je que les fichiers soit stocke dans ma machine en local pas de lien comment faire

Pour onglet quiz : il me dit impossible denregistre mais resultat donc mes problemes la sont ca et je vois pas un button affilier a quiz pour la relecture aussi delai munitage pour chaue question aussi les explication lie a une question

Pour onglet progression : on doit voir les classement par bts(1OU2) en general par rapport a un quiz ou par rapport a un module et le tout de tout les utilisateurs. Aussi les badge des utilisateurs par rapport a un quiz les resultats dun etudiant donnee etc bref tout ce qui est stat

Pour onglet profil : lajout de photo ne passe pas

Pour onglet admin pour les admin : le crud de fichier, videos ne passe pas seul student et quiz passe

Propose-moi un plan pour dire cest quoi le probleme et les solutions en se basant sur :

Voici mes models

// lib/models/course\_model.dart

import 'fiche\_model.dart';

import 'video\_model.dart';

import 'quiz\_model.dart';

class Course {

  final String id;

  final String title;

  final String description;

  final List<String> tags;

  final String? imageUrl;

  final List<Fiche> fiches;

  final List<VideoItem> videos;

  final List<QuizModel> quizzes; // 🔹 Quizzes liés au module

  Course({

    required *this*.id,

    required *this*.title,

    required *this*.description,

*this*.tags = const [],

*this*.imageUrl,

*this*.fiches = const [],

*this*.videos = const [],

*this*.quizzes = const [],

  });

  // --- Helpers (computed counts) ---

  int get countFiches => fiches.length;

  int get countVideos => videos.length;

  int get countQuizzes => quizzes.length;

  // --- Factory ---

  factory Course.*fromMap*(Map<String, dynamic> map, {required String id}) {

    final fichesList = (map['fiches'] as List<dynamic>? ?? [])

        .*map*((f) => Fiche.*fromMap*(

              Map<String, dynamic>.*from*(f),

              id: f['id'] ?? '',

            ))

        .*toList*();

    final videosList = (map['videos'] as List<dynamic>? ?? [])

        .*map*((v) => VideoItem.*fromMap*(

              Map<String, dynamic>.*from*(v),

              id: v['id'] ?? '',

            ))

        .*toList*();

    final quizList = (map['quizzes'] as List<dynamic>? ?? [])

        .*map*((q) => QuizModel.*fromMap*(

              Map<String, dynamic>.*from*(q),

              id: q['id'] ?? '',

              moduleId: id, // 🔹 FIX : passage obligatoire du moduleId

            ))

        .*toList*();

**return** Course(

      id: id,

      title: (map['title'] ?? '').*toString*(),

      description: (map['description'] ?? '').*toString*(),

      tags: List<String>.*from*(map['tags'] ?? const []),

      imageUrl: map['imageUrl'] as String?,

      fiches: fichesList,

      videos: videosList,

      quizzes: quizList,

    );

  }

  // --- Serialize ---

  Map<String, dynamic> *toMap*() {

**return** {

      'id': id,

      'title': title,

      'description': description,

      'tags': tags,

      'imageUrl': imageUrl,

      'fiches': fiches.*map*((f) => f.*toMap*()).*toList*(),

      'videos': videos.*map*((v) => v.*toMap*()).*toList*(),

      'quizzes': quizzes.*map*((q) => q.*toMap*()).*toList*(),

    };

  }

}

// lib/models/fiche\_model.dart

class Fiche {

  final String id;

  final String title;

  final String url; // lien Firebase Storage (PDF)

  final String? description;

  final int?

      duration; // minutes (optionnel, tu peux remplir à la main si besoin)

  final int? pages; // nombre de pages

  final String? level;

  final List<String> tags;

  final int order; // ✅ pour trier correctement les fiches (1,2,3...)

  Fiche({

    required *this*.id,

    required *this*.title,

    required *this*.url,

*this*.description,

*this*.duration,

*this*.pages,

*this*.level,

*this*.tags = const [],

*this*.order = 0,

  });

  // --- Helpers ---

  static int? *\_toNullableInt*(dynamic v) {

**if** (v == null) **return** null;

**if** (v is int) **return** v;

**if** (v is double) **return** v.*toInt*();

**if** (v is num) **return** v.*toInt*();

**if** (v is String) **return** int.*tryParse*(v);

**return** null;

  }

  static List<String> *\_toStringList*(dynamic v) {

**if** (v == null) **return** [];

**if** (v is List) {

**return** v

          .*map*((e) => e?.*toString*() ?? '')

          .*where*((s) => s.isNotEmpty)

          .*toList*();

    }

**if** (v is String) {

**return** v

          .*split*(',')

          .*map*((s) => s.*trim*())

          .*where*((s) => s.isNotEmpty)

          .*toList*();

    }

**return** [];

  }

  // --- Factory ---

  factory Fiche.*fromMap*(Map<String, dynamic> map, {required String id}) {

**return** Fiche(

      id: id,

      title: (map['title'] ?? '').*toString*(),

      url: (map['url'] ?? '')

          .*toString*(), // 🔥 toujours fourni par Firebase Storage

      description: (map['description'])?.*toString*(),

      duration: *\_toNullableInt*(map['duration']),

      pages: *\_toNullableInt*(map['pages']),

      level: (map['level'])?.*toString*(),

      tags: *\_toStringList*(map['tags']),

      order: *\_toNullableInt*(map['order']) ?? 0, // 🔥 pour l’ordre correct

    );

  }

  // --- Serialize ---

  Map<String, dynamic> *toMap*() {

**return** {

      'id': id,

      'title': title,

      'url': url,

      'description': description,

      'duration': duration,

      'pages': pages,

      'level': level,

      'tags': tags,

      'order': order,

    };

  }

}

// lib/models/question\_model.dart

class QuestionModel {

  final String question;

  final List<String> options;

  final int correctIndex;

  final String? explanation; // 🔹 Explication feedback

  QuestionModel({

    required *this*.question,

    required *this*.options,

    required *this*.correctIndex,

*this*.explanation,

  });

  factory QuestionModel.*fromMap*(Map<String, dynamic> map) {

**return** QuestionModel(

      question: (map['question'] ?? '').*toString*(),

      options: List<String>.*from*(map['options'] ?? []),

      correctIndex: map['correctIndex'] is int

          ? map['correctIndex']

          : int.*tryParse*(map['correctIndex']?.*toString*() ?? '0') ?? 0,

      explanation: map['explanation']?.*toString*(),

    );

  }

  Map<String, dynamic> *toMap*() {

**return** {

      'question': question,

      'options': options,

      'correctIndex': correctIndex,

      'explanation': explanation,

    };

  }

}

// lib/models/quiz\_model.dart

import 'question\_model.dart';

class QuizModel {

  final String id;

  final String title;

  final String moduleId; // 🔹 Pour savoir à quel module appartient le quiz

  final List<QuestionModel> questions;

  final Map<String, int> badgeThresholds;

  QuizModel({

    required *this*.id,

    required *this*.title,

    required *this*.moduleId,

    required *this*.questions,

*this*.badgeThresholds = const {

      'Bronze': 50,

      'Argent': 70,

      'Or': 90,

    },

  });

  factory QuizModel.*fromMap*(

    Map<String, dynamic> map, {

    required String id,

    required String moduleId, // 🔹 Ajout pour forcer l’appartenance

  }) {

**return** QuizModel(

      id: id,

      title: (map['title'] ?? 'Quiz').*toString*(),

      moduleId:

          map['moduleId']?.*toString*() ?? moduleId, // fallback au paramètre

      questions: (map['questions'] as List<dynamic>? ?? [])

          .*map*((q) => QuestionModel.*fromMap*(Map<String, dynamic>.*from*(q)))

          .*toList*(),

    );

  }

  Map<String, dynamic> *toMap*() {

**return** {

      'id': id,

      'title': title,

      'moduleId': moduleId,

      'questions': questions.*map*((q) => q.*toMap*()).*toList*(),

    };

  }

  /// 🔹 copyWith pour modifier uniquement certains champs

  QuizModel *copyWith*({

    String? id,

    String? title,

    String? moduleId,

    List<QuestionModel>? questions,

  }) {

**return** QuizModel(

      id: id ?? *this*.id,

      title: title ?? *this*.title,

      moduleId: moduleId ?? *this*.moduleId,

      questions: questions ?? *this*.questions,

    );

  }

}

// lib/models/quiz\_result\_model.dart

import 'package:cloud\_firestore/cloud\_firestore.dart'; // 🔹 Import nécessaire

class QuizResult {

  final String id;

  final String moduleId;

  final String quizId;

  final int score;

  final int total;

  final int bonnes;

  final int fautes;

  final int duree;

  final int percent;

  final String badge;

  final DateTime date;

  final List<int> answers;

  QuizResult({

    required *this*.id,

    required *this*.moduleId,

    required *this*.quizId,

    required *this*.score,

    required *this*.total,

    required *this*.bonnes,

    required *this*.fautes,

    required *this*.duree,

    required *this*.percent,

    required *this*.badge,

    required *this*.date,

*this*.answers = const [],

  });

  /// 🔹 Création depuis Firestore

  factory QuizResult.*fromFirestore*(DocumentSnapshot<Map<String, dynamic>> doc) {

    final data = doc.*data*() ?? {};

**return** QuizResult(

      id: doc.id,

      moduleId: data['moduleId'] ?? '',

      quizId: data['quizId'] ?? '',

      score: data['score'] ?? 0,

      total: data['total'] ?? 0,

      bonnes: data['bonnes'] ?? 0,

      fautes: data['fautes'] ?? 0,

      duree: data['duree'] ?? 0,

      percent: data['percent'] ?? 0,

      badge: data['badge'] ?? '',

      date: DateTime.*tryParse*(data['date'] ?? '') ?? DateTime.*now*(),

      answers: List<int>.*from*(data['answers'] ?? []),

    );

  }

  /// 🔹 Conversion vers Map (utile pour sauvegarde)

  Map<String, dynamic> *toMap*() {

**return** {

      "moduleId": moduleId,

      "quizId": quizId,

      "score": score,

      "total": total,

      "bonnes": bonnes,

      "fautes": fautes,

      "duree": duree,

      "percent": percent,

      "badge": badge,

      "date": date.*toIso8601String*(),

      "answers": answers,

    };

  }

}

// 📁 lib/models/user\_model.dart

class AppUser {

  final String uid;

  final String name;

  final String email;

  final String phone;

  final String school;

  final String speciality;

  final String role;

  final String password;

  final String? photoUrl;

  final String? niveau;

  AppUser({

    required *this*.uid,

    required *this*.name,

    required *this*.email,

*this*.phone = '',

*this*.school = '',

*this*.speciality = '',

*this*.role = 'etudiant',

*this*.password = '',

*this*.photoUrl,

*this*.niveau,

  });

  Map<String, dynamic> *toMap*() {

**return** {

      'uid': uid,

      'name': name,

      'email': email,

      'phone': phone,

      'school': school,

      'speciality': speciality,

      'role': role,

      'password': password,

      'photoUrl': photoUrl,

      'niveau': niveau,

    };

  }

  factory AppUser.*fromMap*(Map<String, dynamic> map) {

**return** AppUser(

      uid: map['uid'] ?? '',

      name: map['name'] ?? '',

      email: map['email'] ?? '',

      phone: map['phone'] ?? '',

      school: map['school'] ?? '',

      speciality: map['speciality'] ?? '',

      role: map['role'] ?? 'etudiant',

      password: map['password'] ?? '',

      photoUrl: map['photoUrl'],

      niveau: map['niveau'],

    );

  }

  // ======= copyWith =======

  AppUser *copyWith*({

    String? uid,

    String? name,

    String? email,

    String? phone,

    String? school,

    String? speciality,

    String? role,

    String? password,

    String? photoUrl,

    String? niveau,

  }) {

**return** AppUser(

      uid: uid ?? *this*.uid,

      name: name ?? *this*.name,

      email: email ?? *this*.email,

      phone: phone ?? *this*.phone,

      school: school ?? *this*.school,

      speciality: speciality ?? *this*.speciality,

      role: role ?? *this*.role,

      password: password ?? *this*.password,

      photoUrl: photoUrl ?? *this*.photoUrl,

      niveau: niveau ?? *this*.niveau,

    );

  }

}

// lib/models/video\_model.dart

class VideoItem {

  final String id;

  final String title;

  final String url; // 🔥 Firebase Storage (MP4) ou YouTube (si externe)

  final String? description;

  final int? duration; // secondes (ou minutes si tu standardises)

  final String? level;

  final List<String> tags;

  final int order; // ✅ pour trier correctement les vidéos

  VideoItem({

    required *this*.id,

    required *this*.title,

    required *this*.url,

*this*.description,

*this*.duration,

*this*.level,

*this*.tags = const [],

*this*.order = 0,

  });

  // --- Helpers ---

  static int? *\_toNullableInt*(dynamic v) {

**if** (v == null) **return** null;

**if** (v is int) **return** v;

**if** (v is double) **return** v.*toInt*();

**if** (v is num) **return** v.*toInt*();

**if** (v is String) **return** int.*tryParse*(v);

**return** null;

  }

  static List<String> *\_toStringList*(dynamic v) {

**if** (v == null) **return** [];

**if** (v is List) {

**return** v

          .*map*((e) => e?.*toString*() ?? '')

          .*where*((s) => s.isNotEmpty)

          .*toList*();

    }

**if** (v is String) {

**return** v

          .*split*(',')

          .*map*((s) => s.*trim*())

          .*where*((s) => s.isNotEmpty)

          .*toList*();

    }

**return** [];

  }

  // --- Factory ---

  factory VideoItem.*fromMap*(Map<String, dynamic> map, {required String id}) {

**return** VideoItem(

      id: id,

      title: (map['title'] ?? '').*toString*(),

      url: (map['url'] ?? '').*toString*(), // 🔥 Firebase Storage obligatoire

      description: (map['description'])?.*toString*(),

      duration: *\_toNullableInt*(map['duration']),

      level: (map['level'])?.*toString*(),

      tags: *\_toStringList*(map['tags']),

      order: *\_toNullableInt*(map['order']) ?? 0, // 🔥 ordre correct

    );

  }

  // --- Serialize ---

  Map<String, dynamic> *toMap*() {

**return** {

      'id': id,

      'title': title,

      'url': url,

      'description': description,

      'duration': duration,

      'level': level,

      'tags': tags,

      'order': order,

    };

  }

}

Voici mes imports (tout en rapport avec firebase)

// fiches\_m1.json

[

  {

    "id": "f1",

    "title": "Fiche 1 - Bilan",

    "url": "https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf",

    "pages": 3,

    "level": "BTS1",

    "tags": ["math"]

  },

  {

    "id": "f2",

    "title": "Fiche 2 - Algèbre",

    "url": "https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf",

    "pages": 4,

    "level": "BTS2",

    "tags": ["math"]

  }

]

// fiches\_m2.json

[

  {

    "id": "f1",

    "title": "Fiche 1 - Introduction Comptabilité",

    "url": "https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf",

    "pages": 3,

    "level": "BTS1",

    "tags": ["compta"]

  },

  {

    "id": "f2",

    "title": "Fiche 2 - Bilan Comptable",

    "url": "https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf",

    "pages": 4,

    "level": "BTS2",

    "tags": ["compta"]

  },

  {

    "id": "f3",

    "title": "Fiche 3 - Compte de résultat",

    "url": "https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf",

    "pages": 5,

    "level": "BTS1",

    "tags": ["compta"]

  }

]

// import.js   
const admin = require("firebase-admin");

const *fs* = require("fs");

const *path* = *require*("path");

const KEY = "./serviceAccountKey.json";

**if** (!fs.existsSync(KEY)) {

  console.*error*("Erreur: serviceAccountKey.json introuvable dans le dossier.");

  process.*exit*(1);

}

admin.*initializeApp*({

  credential: admin.*credential*.*cert*(*require*(*KEY*)),

});

const *db* = admin.*firestore*();

// Supprimer une collection (batch 500)

async function *deleteCollection*(*collectionPath*) {

  const *colRef* = db.*collection*(collectionPath);

**while** (true) {

    const *snapshot* = **await** colRef.*limit*(500).*get*();

**if** (snapshot.empty) **return**;

    const *batch* = db.*batch*();

    snapshot.*docs*.*forEach*((*doc*) => batch.*delete*(doc.ref));

**await** batch.*commit*();

  }

}

async function *importModules*(*reset* = false) {

  const *modulesFile* = path.*join*(\_\_dirname, "modules.json");

**if** (!fs.*existsSync*(modulesFile)) **throw** new Error("modules.json introuvable");

  const *modules* = *JSON*.*parse*(fs.*readFileSync*(modulesFile));

**for** (const *m* of modules) {

    const *id* = m.id;

    console.*log*(`\n=== Module ${id} ===`);

    const *moduleRef* = db.*collection*("modules").*doc*(id);

**if** (reset) {

**try** {

**await** *deleteCollection*(`modules/${id}/fichesSynthese`);

**await** *deleteCollection*(`modules/${id}/videos`);

**await** *deleteCollection*(`modules/${id}/quizzes`);

        console.*log*(`  -> reset sous-collections (fiches, videos, quizzes) ok`);

      } **catch** (e) {

        console.*log*(`  -> reset partiel:`, e.message || e);

      }

    }

    // Doc module (merge pour ne rien casser)

    const moduleDoc = {

      title: m.title || "",

      description: m.description || "",

      countFiches: m.countFiches || 0,

      countVideos: m.countVideos || 0,

      countQuizzes: m.countQuizzes || 0, // sera recalculé après import

      tags: m.tags || [],

      imageUrl: m.imageUrl || "",

      updatedAt: admin.firestore.FieldValue.serverTimestamp(),

    };

    await moduleRef.set(moduleDoc, { merge: true });

    // Fiches

    const fichesFile = path.join(\_\_dirname, `fiches\_${id}.json`);

    if (fs.existsSync(fichesFile)) {

      const fiches = JSON.parse(fs.readFileSync(fichesFile));

      for (const f of fiches) {

        await moduleRef

          .collection("fichesSynthese")

          .doc(f.id)

          .set(f, { merge: true });

      }

      console.log(`  -> fiches importées: ${fiches.length}`);

    } else {

      console.log(`  -> fiches\_${id}.json absent (ok)`);

    }

    // Videos

    const videosFile = path.join(\_\_dirname, `videos\_${id}.json`);

    if (fs.existsSync(videosFile)) {

      const videos = JSON.parse(fs.readFileSync(videosFile));

      for (const v of videos) {

        await moduleRef.collection("videos").doc(v.id).set(v, { merge: true });

      }

      console.log(`  -> vidéos importées: ${videos.length}`);

    } else {

      console.log(`  -> videos\_${id}.json absent (ok)`);

    }

    // Quizzes (NOUVEAU)

    const quizzesFile = path.join(\_\_dirname, `quizzes\_${id}.json`);

    let quizzesCount = 0;

    if (fs.existsSync(quizzesFile)) {

      const quizzes = JSON.parse(fs.readFileSync(quizzesFile));

      for (const q of quizzes) {

        const qDoc = {

          id: q.id,

          moduleId: q.moduleId || id,

          title: q.title || "Quiz",

          description: q.description || "",

          durationSeconds: q.durationSeconds ?? null,

          allowRetake: q.allowRetake ?? true,

          order: q.order ?? 0,

          badgeThresholds: q.badgeThresholds || {

            gold: 90,

            silver: 75,

            bronze: 50,

          },

          questions: (q.questions || []).map((qq, idx) => ({

            question: (qq.question || "").toString(),

            options: Array.isArray(qq.options) ? qq.options.map(String) : [],

            correctIndex: Number.isInteger(qq.correctIndex)

              ? qq.correctIndex

              : parseInt(qq.correctIndex ?? 0, 10) || 0,

            explanation: (qq.explanation || "").toString(),

          })),

          questionCount: Array.isArray(q.questions) ? q.questions.length : 0,

          createdAt: admin.firestore.FieldValue.serverTimestamp(),

          updatedAt: admin.firestore.FieldValue.serverTimestamp(),

        };

        await moduleRef

          .collection("quizzes")

          .doc(q.id)

          .set(qDoc, { merge: true });

        quizzesCount++;

      }

      console.log(`  -> quizzes importés: ${quizzesCount}`);

    } else {

      console.log(`  -> quizzes\_${id}.json absent (ok)`);

    }

    // Mise à jour countQuizzes calculé

    await moduleRef.set(

      {

        countQuizzes: quizzesCount,

        updatedAt: admin.firestore.FieldValue.serverTimestamp(),

      },

      { merge: true }

    );

  }

  console.log("\nImport terminé ✅");

}

// Exécution

const doReset = process.argv.includes("--reset");

importModules(doReset).catch((err) => {

  console.error("Erreur import:", err);

  process.exit(1);

});

Modules.json  
[

  {

    "id": "m1",

    "title": "Mathématiques BTS",

    "description": "Module de révision en mathématiques pour BTS 1 & 2",

    "tags": ["math", "bts"],

    "imageUrl": "",

    "countFiches": 10,

    "countVideos": 10,

    "countQuizzes": 3

  },

  {

    "id": "m2",

    "title": "Comptabilité BTS",

    "description": "Module de révision en comptabilité pour BTS 1 & 2",

    "tags": ["compta", "bts"],

    "imageUrl": "",

    "countFiches": 10,

    "countVideos": 10,

    "countQuizzes": 3

  }

]

Quizzes\_m1.json  
[

  {

    "id": "q1",

    "moduleId": "m1",

    "title": "Quiz Math 1 - Fondamentaux",

    "description": "Opérations de base, dérivées/ intégrales simples, équation linéaire.",

    "durationSeconds": 600,

    "allowRetake": true,

    "order": 1,

    "badgeThresholds": { "gold": 90, "silver": 75, "bronze": 50 },

    "questions": [

      {

        "question": "1 + 1 = ?",

        "options": ["1", "2", "3", "4"],

        "correctIndex": 1,

        "explanation": "Addition de deux unités : 1 + 1 = 2."

      },

      {

        "question": "2 + 2 = ?",

        "options": ["2", "3", "4", "5"],

        "correctIndex": 2,

        "explanation": "2 + 2 = 4. Pense à grouper par paires."

      },

      {

        "question": "Quelle est la dérivée de x² ?",

        "options": ["x", "2x", "x²", "2"],

        "correctIndex": 1,

        "explanation": "Règle de puissance : d(x^n)/dx = n·x^(n-1). Pour x², c’est 2x."

      },

      {

        "question": "∫ 1 dx = ?",

        "options": ["x + C", "1 + C", "0 + C", "2x + C"],

        "correctIndex": 0,

        "explanation": "La primitive de 1 est x (constante 1 → pente 0). On ajoute la constante C."

      },

      {

        "question": "Résoudre 2x = 6",

        "options": ["1", "2", "3", "4"],

        "correctIndex": 2,

        "explanation": "Diviser par 2 des deux côtés : x = 3."

      }

    ]

  },

  {

    "id": "q2",

    "moduleId": "m1",

    "title": "Quiz Math 2 - Dérivées",

    "description": "Dérivées de base (test rapide).",

    "durationSeconds": 480,

    "allowRetake": true,

    "order": 2,

    "badgeThresholds": { "gold": 90, "silver": 75, "bronze": 50 },

    "questions": [

      {

        "question": "La dérivée de sin(x) est…",

        "options": ["cos(x)", "-cos(x)", "sin(x)", "-sin(x)"],

        "correctIndex": 0,

        "explanation": "d/dx[sin(x)] = cos(x)."

      },

      {

        "question": "La dérivée de 3x est…",

        "options": ["x", "3", "3x²", "0"],

        "correctIndex": 1,

        "explanation": "d/dx[3x] = 3 (constante \* x → constante)."

      }

    ]

  },

  {

    "id": "q3",

    "moduleId": "m1",

    "title": "Quiz Math 3 - Algèbre",

    "description": "Équations et matrices (test rapide).",

    "durationSeconds": 480,

    "allowRetake": true,

    "order": 3,

    "badgeThresholds": { "gold": 90, "silver": 75, "bronze": 50 },

    "questions": [

      {

        "question": "Solve 2x = 6",

        "options": ["1", "2", "3", "4"],

        "correctIndex": 2,

        "explanation": "x = 3, en divisant par 2."

      },

      {

        "question": "Le déterminant d’une matrice 2×2 [[a,b],[c,d]] est…",

        "options": ["a+b+c+d", "ad - bc", "ab + cd", "a-d"],

        "correctIndex": 1,

        "explanation": "det = ad − bc."

      }

    ]

  }

]

Quizzes\_m2.json  
[

  {

    "id": "q1",

    "moduleId": "m2",

    "title": "Quiz Compta 1 - Fondamentaux",

    "description": "Bilan, TVA, amortissements, stocks, partie double.",

    "durationSeconds": 600,

    "allowRetake": true,

    "order": 1,

    "badgeThresholds": { "gold": 90, "silver": 75, "bronze": 50 },

    "questions": [

      {

        "question": "Le bilan regroupe…",

        "options": ["Charges", "Actifs et passifs", "Revenus", "Dépenses"],

        "correctIndex": 1,

        "explanation": "Le bilan présente la situation patrimoniale : actifs et passifs."

      },

      {

        "question": "La TVA à décaisser figure dans…",

        "options": [

          "Le bilan",

          "Le compte de résultat",

          "L’annexe",

          "Le journal"

        ],

        "correctIndex": 0,

        "explanation": "La TVA collectée/déductible et le solde à décaisser apparaissent au bilan."

      },

      {

        "question": "Un amortissement est…",

        "options": [

          "Une dépense non monétaire",

          "Une dette",

          "Une recette",

          "Une charge variable"

        ],

        "correctIndex": 0,

        "explanation": "C’est une charge comptable qui répartit le coût d’un actif sans sortie de trésorerie immédiate."

      },

      {

        "question": "Les stocks sont valorisés selon…",

        "options": [

          "Le coût historique",

          "La valeur de marché",

          "Le moins élevé des deux",

          "A et B"

        ],

        "correctIndex": 2,

        "explanation": "Principe du plus bas du coût ou de la valeur nette réalisable."

      },

      {

        "question": "Principe de la partie double :",

        "options": [

          "Chaque écriture a un débit et un crédit",

          "On enregistre une seule fois",

          "On n’enregistre que les recettes",

          "C’est facultatif"

        ],

        "correctIndex": 0,

        "explanation": "Toute opération affecte au moins deux comptes : un débit et un crédit."

      }

    ]

  },

  {

    "id": "q2",

    "moduleId": "m2",

    "title": "Quiz Compta 2 - Amortissements",

    "description": "Rappels rapides.",

    "durationSeconds": 480,

    "allowRetake": true,

    "order": 2,

    "badgeThresholds": { "gold": 90, "silver": 75, "bronze": 50 },

    "questions": [

      {

        "question": "L’amortissement linéaire répartit le coût…",

        "options": [

          "En parts inégales",

          "De manière constante",

          "Uniquement la première année",

          "Jamais"

        ],

        "correctIndex": 1,

        "explanation": "Linéraire = même dotation chaque période."

      }

    ]

  },

  {

    "id": "q3",

    "moduleId": "m2",

    "title": "Quiz Compta 3 - Stocks",

    "description": "FIFO/LIFO (test rapide).",

    "durationSeconds": 480,

    "allowRetake": true,

    "order": 3,

    "badgeThresholds": { "gold": 90, "silver": 75, "bronze": 50 },

    "questions": [

      {

        "question": "FIFO signifie…",

        "options": [

          "First In, First Out",

          "Fast In, Fast Out",

          "First In, Final Out",

          "Fixed In, Fixed Out"

        ],

        "correctIndex": 0,

        "explanation": "Les premières entrées sont les premières sorties."

      },

      {

        "question": "Le LIFO est autorisé en IFRS ?",

        "options": ["Oui", "Non"],

        "correctIndex": 1,

        "explanation": "Le LIFO n’est pas autorisé en IFRS."

      }

    ]

  }

]

Videos\_m1.json  
[

  {

    "id": "v1",

    "title": "Intro Mathématiques",

    "url": "https://sample-videos.com/video123/mp4/720/big\_buck\_bunny\_720p\_1mb.mp4",

    "duration": 120,

    "level": "BTS1",

    "tags": ["math"]

  },

  {

    "id": "v2",

    "title": "Algèbre - Bases",

    "url": "https://sample-videos.com/video123/mp4/720/big\_buck\_bunny\_720p\_1mb.mp4",

    "duration": 180,

    "level": "BTS2",

    "tags": ["math"]

  }

]

Videos\_m2.json  
[

  {

    "id": "v1",

    "title": "Intro Comptabilité",

    "url": "https://sample-videos.com/video123/mp4/720/big\_buck\_bunny\_720p\_1mb.mp4",

    "duration": 140,

    "level": "BTS1",

    "tags": ["compta"]

  }

]

Voici mes codes  
// lib/app/app\_routes.dart

import 'package:flutter/material.dart';

import 'package:reussirbts/app/splash\_screen.dart';

import 'package:reussirbts/screens/auth/login\_screen.dart';

import 'package:reussirbts/screens/auth/register\_screen.dart';

import 'package:reussirbts/screens/dashboard.dart';

import 'package:reussirbts/screens/tabs/home\_tab.dart';

import 'package:reussirbts/screens/tabs/revision\_tab.dart';

import 'package:reussirbts/screens/tabs/quiz\_tab.dart';

import 'package:reussirbts/screens/tabs/progress\_tab.dart';

import 'package:reussirbts/screens/tabs/profile\_tab.dart';

import 'package:reussirbts/screens/admin/admin\_panel.dart';

class AppRoutes {

  // Routes publiques

  static const splash = '/';

  static const login = '/login';

  static const register = '/register';

  // Routes protégées (Dashboard et Admin)

  static const dashboard = '/dashboard';

  static const adminPanel = '/admin';

  // Onglets dashboard

  static const homeTab = '/dashboard/home';

  static const revisionTab = '/dashboard/revision';

  static const quizTab = '/dashboard/quiz';

  static const progressTab = '/dashboard/progress';

  static const profileTab = '/dashboard/profile';

  static Map<String, WidgetBuilder> routes = {

    splash: (context) => const SplashScreen(),

    login: (context) => const LoginScreen(),

    register: (context) => const RegisterScreen(),

    dashboard: (context) => const Dashboard(),

    adminPanel: (context) => const AdminPanel(),

    homeTab: (context) => const HomeTab(),

    revisionTab: (context) => const RevisionTab(),

    quizTab: (context) => const QuizTab(),

    progressTab: (context) => const ProgressTab(),

    profileTab: (context) => const ProfileTab(),

  };

}

// lib/app/app\_theme.dart

import 'package:flutter/material.dart';

class AppTheme {

  // 🎨 Palette de couleurs

  static const Color primaryColor = Color(0xFF2563EB); // Bleu doux

  static const Color secondaryColor = Color(0xFFF3F4F6); // Gris clair

  static const Color accentColor = Color(0xFF10B981); // Vert doux

  static const Color errorColor = Color(0xFFDC2626); // Rouge

  // 🌞 Thème clair

  static ThemeData get lightTheme {

**return** ThemeData(

      brightness: Brightness.light,

      fontFamily: 'Roboto',

      primaryColor: primaryColor,

      scaffoldBackgroundColor: Colors.white, // UNIQUE

      colorScheme: const ColorScheme.*light*(

        primary: primaryColor,

        secondary: accentColor,

        error: errorColor,

      ),

      visualDensity: VisualDensity.adaptivePlatformDensity,

      // Textes

      textTheme: const TextTheme(

        headlineLarge: TextStyle(

            fontSize: 32, fontWeight: FontWeight.bold, color: Colors.black87),

        headlineMedium: TextStyle(

            fontSize: 24, fontWeight: FontWeight.w600, color: Colors.black87),

        bodyLarge: TextStyle(fontSize: 16, color: Colors.black87),

        bodyMedium: TextStyle(fontSize: 14, color: Colors.grey),

        bodySmall: TextStyle(fontSize: 12, color: Colors.grey),

      ),

      // Boutons

      elevatedButtonTheme: ElevatedButtonThemeData(

        style: ElevatedButton.*styleFrom*(

          backgroundColor: primaryColor,

          foregroundColor: Colors.white,

          minimumSize: const Size.*fromHeight*(48),

          shape:

              RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(12)),

          textStyle: const TextStyle(fontWeight: FontWeight.w600),

        ),

      ),

      textButtonTheme: TextButtonThemeData(

        style: TextButton.*styleFrom*(

          foregroundColor: primaryColor,

          textStyle: const TextStyle(fontWeight: FontWeight.w600),

        ),

      ),

      outlinedButtonTheme: OutlinedButtonThemeData(

        style: OutlinedButton.*styleFrom*(

          foregroundColor: primaryColor,

          side: const BorderSide(color: primaryColor),

          shape:

              RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(12)),

        ),

      ),

      // Champs de saisie

      inputDecorationTheme: InputDecorationTheme(

        border: OutlineInputBorder(

          borderRadius: BorderRadius.*circular*(12),

          borderSide: const BorderSide(color: Colors.grey),

        ),

        focusedBorder: OutlineInputBorder(

          borderRadius: BorderRadius.*circular*(12),

          borderSide: const BorderSide(color: primaryColor, width: 2),

        ),

        contentPadding:

            const EdgeInsets.*symmetric*(horizontal: 16, vertical: 14),

      ),

      // Cartes

      cardTheme: CardTheme(

        elevation: 2,

        shape: RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(12)),

        margin: const EdgeInsets.*symmetric*(vertical: 8, horizontal: 0),

      ),

      // Icônes & AppBar

      iconTheme: const IconThemeData(color: primaryColor, size: 24),

      appBarTheme: const AppBarTheme(

        backgroundColor: primaryColor,

        foregroundColor: Colors.white,

        elevation: 0,

      ),

    );

  }

  // 🌙 Thème sombre

  static ThemeData get darkTheme {

**return** ThemeData.*dark*().*copyWith*(

      primaryColor: primaryColor,

      scaffoldBackgroundColor: const Color(0xFF1E1E1E),

      colorScheme: const ColorScheme.*dark*(

        primary: primaryColor,

        secondary: accentColor,

        error: errorColor,

      ),

      textTheme: const TextTheme(

        headlineLarge: TextStyle(

            fontSize: 32, fontWeight: FontWeight.bold, color: Colors.white),

        headlineMedium: TextStyle(

            fontSize: 24, fontWeight: FontWeight.w600, color: Colors.white70),

        bodyLarge: TextStyle(fontSize: 16, color: Colors.white70),

        bodyMedium: TextStyle(fontSize: 14, color: Colors.grey),

      ),

      elevatedButtonTheme: ElevatedButtonThemeData(

        style: ElevatedButton.*styleFrom*(

          backgroundColor: primaryColor,

          foregroundColor: Colors.white,

          shape:

              RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(12)),

        ),

      ),

      cardTheme: CardTheme(

        color: const Color(0xFF2C2C2C),

        shape: RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(12)),

      ),

      appBarTheme: const AppBarTheme(

        backgroundColor: Color(0xFF111111),

        foregroundColor: Colors.white,

        elevation: 0,

      ),

    );

  }

}

// 📁 lib/app/auth\_gate.dart

import 'dart:convert';

import 'dart:io';

import 'package:flutter/foundation.dart' show kIsWeb;

import 'package:path\_provider/path\_provider.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import '../models/user\_model.dart';

import '../services/local\_user\_service.dart';

class AuthService {

  final FirebaseAuth \_auth = FirebaseAuth.instance;

  final FirebaseFirestore \_db = FirebaseFirestore.instance;

  final LocalUserService \_localQueue = LocalUserService();

  // ---------------- SIGN UP ----------------

  Future<AppUser> *register*({

    required String name,

    required String email,

    required String phone,

    required String school,

    required String speciality,

    required String password,

    String niveau = 'BTS1',

  }) **async** {

**try** {

      final cred = **await** \_auth.*createUserWithEmailAndPassword*(

        email: email,

        password: password,

      );

      final uid = cred.user!.uid;

      final user = AppUser(

        uid: uid,

        name: name,

        email: email,

        phone: phone,

        school: school,

        speciality: speciality,

        role: 'etudiant',

        password: password,

        niveau: niveau,

      );

**await** \_db.*collection*('users').*doc*(uid).*set*({

        'uid': uid,

        'name': name,

        'email': email,

        'phone': phone,

        'school': school,

        'speciality': speciality,

        'role': 'etudiant',

        'niveau': niveau,

        'createdAt': DateTime.*now*().*toIso8601String*(),

      });

**await** *\_saveUserLocal*(user);

**return** user;

    } **catch** (e) {

      final fakeUid = 'pending\_${DateTime.*now*().*millisecondsSinceEpoch*}';

      final user = AppUser(

        uid: fakeUid,

        name: name,

        email: email,

        phone: phone,

        school: school,

        speciality: speciality,

        role: 'etudiant',

        password: password,

        niveau: niveau,

      );

**await** \_localQueue.*savePendingUserMap*({

        'name': name,

        'email': email,

        'phone': phone,

        'school': school,

        'speciality': speciality,

        'password': password,

        'role': 'etudiant',

        'niveau': niveau,

        'createdAt': DateTime.*now*().*toIso8601String*(),

      });

**await** *\_saveUserLocal*(user);

**return** user;

    }

  }

  // ---------------- LOGIN ----------------

  Future<AppUser> *login*({

    required String email,

    required String password,

  }) **async** {

**try** {

      final cred = **await** \_auth.*signInWithEmailAndPassword*(

        email: email,

        password: password,

      );

      final snap = **await** \_db.*collection*('users').*doc*(cred.user!.uid).*get*();

**if** (snap.exists) {

        final data = snap.*data*()!;

        final user = AppUser.*fromMap*(data);

**await** *\_saveUserLocal*(user);

**return** user;

      }

      final localUser = **await** *getLocalUserByUid*(cred.user!.uid);

**if** (localUser != null) **return** localUser;

**return** AppUser(

        uid: cred.user!.uid,

        name: cred.user!.displayName ?? '',

        email: cred.user!.email ?? email,

        phone: '',

        school: '',

        speciality: '',

        role: 'etudiant',

        password: password,

        niveau: 'BTS1',

      );

    } **catch** (e) {

      final user = **await** *getLocalUserByEmail*(email);

**if** (user != null && user.password == password) {

**return** user;

      }

**throw** Exception("Impossible de se connecter");

    }

  }

  // ---------------- RESET PASSWORD ----------------

  Future<void> *resetPassword*(String email) **async** {

**try** {

**await** \_auth.*sendPasswordResetEmail*(email: email);

    } **catch** (e) {

**throw** Exception("Impossible d’envoyer l’email de réinitialisation : $*e*");

    }

  }

  // ---------------- LOGOUT ----------------

  Future<void> *logout*() **async** {

**try** {

**await** \_auth.*signOut*();

    } **catch** (\_) {}

**await** *\_clearLocalSession*();

  }

  // ---------------- REPLAY PENDING USERS ----------------

  Future<void> *replayPendingUsers*() **async** {

**try** {

      final pending = **await** \_localQueue.*loadPendingUsers*();

**if** (pending.isEmpty) **return**;

      final List<Map<String, dynamic>> stillPending = [];

**for** (final u **in** pending) {

**try** {

          final cred = **await** \_auth.*createUserWithEmailAndPassword*(

            email: u['email'] as String,

            password: u['password'] as String,

          );

          final uid = cred.user!.uid;

**await** \_db.*collection*('users').*doc*(uid).*set*({

            'uid': uid,

            'name': u['name'],

            'email': u['email'],

            'phone': u['phone'],

            'school': u['school'],

            'speciality': u['speciality'],

            'role': 'etudiant',

            'niveau': u['niveau'],

            'createdAt': u['createdAt'],

            'syncedAt': DateTime.*now*().*toIso8601String*(),

          });

        } **catch** (\_) {

          stillPending.*add*(u);

        }

      }

**if** (stillPending.isEmpty) {

**await** \_localQueue.*clearPendingUsers*();

      } **else** {

**await** \_localQueue.*replacePendingUsers*(stillPending);

      }

    } **catch** (e) {

**throw** Exception("❌ replayPendingUsers error: $*e*");

    }

  }

  // ---------------- LOCAL JSON ----------------

  Future<void> *\_saveUserLocal*(AppUser user) **async** {

**if** (kIsWeb) **return**;

**try** {

      final dir = **await** *getApplicationDocumentsDirectory*();

      final file = File('${*dir*.*path*}/users.json');

      List<AppUser> users = **await** *\_getAllUsersLocal*();

      users.*removeWhere*((u) => u.uid == user.uid);

      users.*add*(user);

**await** file.*writeAsString*(

*jsonEncode*(users.*map*((u) => u.*toMap*()).*toList*()),

      );

    } **catch** (e) {

**throw** Exception("❌ \_saveUserLocal error: $*e*");

    }

  }

  Future<List<AppUser>> *\_getAllUsersLocal*() **async** {

**if** (kIsWeb) **return** [];

**try** {

      final dir = **await** *getApplicationDocumentsDirectory*();

      final file = File('${*dir*.*path*}/users.json');

**if** (!**await** file.*exists*()) **return** [];

      final jsonStr = **await** file.*readAsString*();

      final List<dynamic> jsonList = *jsonDecode*(jsonStr);

**return** jsonList.*map*((e) => AppUser.*fromMap*(e)).*toList*();

    } **catch** (\_) {

**return** [];

    }

  }

  Future<AppUser?> *getLocalUserByUid*(String uid) **async** {

    final users = **await** *\_getAllUsersLocal*();

**try** {

**return** users.*firstWhere*((u) => u.uid == uid);

    } **catch** (\_) {

**return** null;

    }

  }

  Future<AppUser?> *getLocalUserByEmail*(String email) **async** {

    final users = **await** *\_getAllUsersLocal*();

**try** {

**return** users.*firstWhere*((u) => u.email == email);

    } **catch** (\_) {

**return** null;

    }

  }

  Future<void> *\_clearLocalSession*() **async** {

**if** (kIsWeb) **return**;

**try** {

      final dir = **await** *getApplicationDocumentsDirectory*();

      final file = File('${*dir*.*path*}/session.json');

**if** (**await** file.*exists*()) {

**await** file.*delete*();

      }

    } **catch** (\_) {}

  }

}

// lib/screens/splash\_screen.dart

import 'dart:async';

import 'dart:ui';

import 'package:flutter/material.dart';

import '../utils/app\_constants.dart';

import '../utils/connectivity\_service.dart';

class SplashScreen extends StatefulWidget {

  const SplashScreen({*super*.key});

  @override

  State<SplashScreen> *createState*() => \_SplashScreenState();

}

class \_SplashScreenState extends State<SplashScreen>

    with SingleTickerProviderStateMixin {

  late final AnimationController \_bgController;

  late final Animation<double> \_bgAnim;

  StreamSubscription<bool>? \_connSub;

  bool \_isOnline = true;

  @override

  void *initState*() {

*super*.*initState*();

    // Fond animé léger (boucles lentes pour l'effet pro)

    \_bgController = AnimationController(

      vsync: *this*,

      duration: const Duration(seconds: 20),

    )..*repeat*(reverse: true);

    \_bgAnim = CurvedAnimation(parent: \_bgController, curve: Curves.easeInOut);

    // Écoute réseau en direct

    \_connSub = ConnectivityService.connectivityStream.*listen*((online) {

**if** (mounted) *setState*(() => \_isOnline = online);

    });

    // Check initial du réseau

    ConnectivityService.*isOnline*().*then*((online) {

**if** (mounted) *setState*(() => \_isOnline = online);

    });

  }

  @override

  void *dispose*() {

    \_connSub?.*cancel*();

    \_bgController.*dispose*();

*super*.*dispose*();

  }

  void *\_openBottomSheet*({

    required String title,

    required List<Widget> children,

  }) {

*showModalBottomSheet*(

      context: context,

      isScrollControlled: true,

      backgroundColor: Theme.*of*(context).colorScheme.surface,

      shape: const RoundedRectangleBorder(

        borderRadius: BorderRadius.*vertical*(top: Radius.*circular*(24)),

      ),

      builder: (\_) => DraggableScrollableSheet(

        expand: false,

        initialChildSize: 0.7,

        minChildSize: 0.5,

        maxChildSize: 0.95,

        builder: (context, controller) => Padding(

          padding: const EdgeInsets.*fromLTRB*(16, 16, 16, 24),

          child: Column(

            children: [

              Container(

                width: 48,

                height: 5,

                decoration: BoxDecoration(

                  color: Colors.grey.shade400,

                  borderRadius: BorderRadius.*circular*(12),

                ),

              ),

              const SizedBox(height: 12),

              Text(

                title,

                style: Theme.*of*(context).textTheme.titleLarge?.*copyWith*(

                      fontWeight: FontWeight.w800,

                    ),

              ),

              const SizedBox(height: 16),

              Expanded(

                child: ListView(

                  controller: controller,

                  children: children,

                ),

              ),

            ],

          ),

        ),

      ),

    );

  }

  Widget *\_networkBanner*() {

**if** (\_isOnline) **return** const SizedBox.*shrink*();

**return** Container(

      width: double.infinity,

      padding: const EdgeInsets.*symmetric*(horizontal: 12, vertical: 8),

      color: Colors.redAccent,

      child: const SafeArea(

        bottom: false,

        child: Text(

          'Vous êtes hors ligne — certaines fonctionnalités peuvent être limitées.',

          style: TextStyle(color: Colors.white, fontWeight: FontWeight.bold),

          textAlign: TextAlign.center,

        ),

      ),

    );

  }

  // Section Hero avec CTA

  Widget *\_heroSection*() {

    final color = Theme.*of*(context).colorScheme;

**return** Padding(

      padding: const EdgeInsets.*fromLTRB*(16, 24, 16, 24),

      child: LayoutBuilder(

        builder: (context, constraints) {

          final isWide = constraints.maxWidth > 900;

**return** Row(

            crossAxisAlignment: CrossAxisAlignment.center,

            children: [

              Expanded(

                flex: isWide ? 6 : 10,

                child: Column(

                  crossAxisAlignment: isWide

                      ? CrossAxisAlignment.start

                      : CrossAxisAlignment.center,

                  children: [

                    Text(

                      'Réussir BTS',

                      textAlign: isWide ? TextAlign.left : TextAlign.center,

                      style: Theme.*of*(context).textTheme.displaySmall?.*copyWith*(

                            fontWeight: FontWeight.w900,

                            height: 1.05,

                            letterSpacing: -0.5,

                          ),

                    ),

                    const SizedBox(height: 12),

                    Text(

                      'Ton hub d’apprentissage moderne : révisions intelligentes, micro-learnings, quiz adaptatifs, suivi de progression.',

                      textAlign: isWide ? TextAlign.left : TextAlign.center,

                      style: Theme.*of*(context).textTheme.titleMedium?.*copyWith*(

                            color: color.onSurface.*withOpacity*(0.75),

                            height: 1.35,

                          ),

                    ),

                    const SizedBox(height: 20),

                    Wrap(

                      spacing: 12,

                      runSpacing: 12,

                      alignment:

                          isWide ? WrapAlignment.start : WrapAlignment.center,

                      children: [

                        ElevatedButton.*icon*(

                          onPressed: () =>

                              Navigator.*pushNamed*(context, '/login'),

                          icon: const Icon(Icons.login),

                          style: ElevatedButton.*styleFrom*(

                            padding: const EdgeInsets.*symmetric*(

                                horizontal: 18, vertical: 14),

                            shape: RoundedRectangleBorder(

                              borderRadius: BorderRadius.*circular*(14),

                            ),

                          ),

                          label: const Text('Se connecter'),

                        ),

                        OutlinedButton.*icon*(

                          onPressed: () =>

                              Navigator.*pushNamed*(context, '/register'),

                          icon: const Icon(Icons.person\_add\_alt),

                          style: OutlinedButton.*styleFrom*(

                            padding: const EdgeInsets.*symmetric*(

                                horizontal: 18, vertical: 14),

                            shape: RoundedRectangleBorder(

                              borderRadius: BorderRadius.*circular*(14),

                            ),

                          ),

                          label: const Text("Créer un compte"),

                        ),

                      ],

                    ),

                    const SizedBox(height: 18),

                    Row(

                      mainAxisSize: MainAxisSize.min,

                      mainAxisAlignment: isWide

                          ? MainAxisAlignment.start

                          : MainAxisAlignment.center,

                      children: [

*\_badge*('Révisions rapides', Icons.timer),

                        const SizedBox(width: 8),

*\_badge*('Suivi personnalisé', Icons.insights),

                        const SizedBox(width: 8),

*\_badge*('100% gratuit', Icons.favorite),

                      ],

                    ),

                  ],

                ),

              ),

**if** (isWide) const SizedBox(width: 24),

**if** (isWide)

                Expanded(

                  flex: 5,

                  child: *\_heroIllustration*(),

                ),

            ],

          );

        },

      ),

    );

  }

  Widget *\_heroIllustration*() {

    final radius = BorderRadius.*circular*(24);

**return** AspectRatio(

      aspectRatio: 16 / 10,

      child: ClipRRect(

        borderRadius: radius,

        child: Stack(

          fit: StackFit.expand,

          children: [

            // Image d’illustration (optionnelle)

            Image.*asset*(

              AppConstants.splashImage,

              fit: BoxFit.cover,

              errorBuilder: (\_, \_\_, \_\_\_) => Container(

                decoration: const BoxDecoration(

                  gradient: LinearGradient(

                    colors: [Color(0xFF1E3C72), Color(0xFF2A5298)],

                    begin: Alignment.topLeft,

                    end: Alignment.bottomRight,

                  ),

                ),

              ),

            ),

            Container(

              decoration: BoxDecoration(

                gradient: LinearGradient(

                  colors: [

                    Colors.black.*withOpacity*(0.2),

                    Colors.black.*withOpacity*(0.05),

                  ],

                  begin: Alignment.bottomCenter,

                  end: Alignment.topCenter,

                ),

              ),

            ),

            Positioned(

              left: 16,

              right: 16,

              bottom: 16,

              child: *\_glassCard*(

                child: Row(

                  children: const [

                    Icon(Icons.flash\_on, size: 20),

                    SizedBox(width: 8),

                    Expanded(

                      child: Text(

                        'Commence par une session de micro-learning de 10 min.',

                        style: TextStyle(fontWeight: FontWeight.w600),

                      ),

                    ),

                  ],

                ),

              ),

            ),

          ],

        ),

      ),

    );

  }

  Widget *\_badge*(String text, IconData icon) {

**return** Container(

      padding: const EdgeInsets.*symmetric*(horizontal: 10, vertical: 6),

      decoration: BoxDecoration(

        color: Theme.*of*(context).colorScheme.primary.*withOpacity*(0.08),

        borderRadius: BorderRadius.*circular*(999),

        border: Border.*all*(

          color: Theme.*of*(context).colorScheme.primary.*withOpacity*(0.25),

        ),

      ),

      child: Row(

        children: [

          Icon(icon, size: 16),

          const SizedBox(width: 6),

          Text(

            text,

            style: const TextStyle(fontWeight: FontWeight.w600),

          ),

        ],

      ),

    );

  }

  // Section fonctionnalités

  Widget *\_featuresSection*() {

    final tiles = [

*\_featureTile*(

        icon: Icons.menu\_book\_rounded,

        title: 'Modules structurés',

        desc:

            'Accès rapide aux cours, fiches et vidéos. Organisation claire par matières et niveaux.',

      ),

*\_featureTile*(

        icon: Icons.bolt,

        title: 'Micro-learning',

        desc:

            'Capsules rapides (8–10 min) pour réviser efficacement tes notions clés.',

      ),

*\_featureTile*(

        icon: Icons.quiz\_outlined,

        title: 'Quiz intelligents',

        desc:

            'Questions ciblées, correction instantanée et historique des résultats.',

      ),

*\_featureTile*(

        icon: Icons.bar\_chart\_rounded,

        title: 'Suivi de progression',

        desc:

            'Visualise tes scores, tendances et objectifs. Reste motivé jour après jour.',

      ),

*\_featureTile*(

        icon: Icons.shield\_outlined,

        title: 'Données sécurisées',

        desc:

            'Synchronisation quand tu es en ligne + fallback local hors-ligne.',

      ),

*\_featureTile*(

        icon: Icons.emoji\_events,

        title: 'Préparation optimale',

        desc:

            'Tous les outils pour réussir ton BTS : cours, quiz, suivi et conseils méthodologiques.',

      ),

    ];

**return** Padding(

      padding: const EdgeInsets.*fromLTRB*(16, 8, 16, 8),

      child: Column(

        crossAxisAlignment: CrossAxisAlignment.start,

        children: [

*\_sectionTitle*('Pourquoi Réussir BTS ?'),

          const SizedBox(height: 12),

          LayoutBuilder(

            builder: (\_, c) {

              final w = c.maxWidth;

              int crossAxis = 1;

**if** (w > 1200)

                crossAxis = 3;

**else** **if** (w > 800) crossAxis = 2;

**return** GridView.*count*(

                crossAxisCount: crossAxis,

                shrinkWrap: true,

                childAspectRatio: 1.35,

                physics: const NeverScrollableScrollPhysics(),

                mainAxisSpacing: 12,

                crossAxisSpacing: 12,

                children: tiles,

              );

            },

          ),

        ],

      ),

    );

  }

  Widget *\_featureTile*(

      {required IconData icon, required String title, required String desc}) {

    final cardColor = Theme.*of*(context).colorScheme.surface;

**return** *\_elevCard*(

      child: Padding(

        padding: const EdgeInsets.*all*(16),

        child: Column(

          crossAxisAlignment: CrossAxisAlignment.start,

          children: [

*\_iconCircle*(icon),

            const SizedBox(height: 12),

            Text(title,

                style:

                    const TextStyle(fontSize: 16, fontWeight: FontWeight.w800)),

            const SizedBox(height: 8),

            Expanded(

              child: Text(

                desc,

                style: TextStyle(

                  color:

                      Theme.*of*(context).colorScheme.onSurface.*withOpacity*(.75),

                ),

              ),

            ),

            const SizedBox(height: 8),

          ],

        ),

      ),

      color: cardColor,

    );

  }

  Widget *\_microCard*(String title, String duration) {

**return** SizedBox(

      width: 220,

      child: *\_elevCard*(

        child: InkWell(

          borderRadius: BorderRadius.*circular*(16),

          onTap: () {

            // CTA simple pour amener au login/inscription

            Navigator.*pushNamed*(context, '/login');

          },

          child: Padding(

            padding: const EdgeInsets.*all*(14),

            child: Column(

              crossAxisAlignment: CrossAxisAlignment.start,

              children: [

                const Icon(Icons.flash\_on, size: 28),

                const Spacer(),

                Text(

                  title,

                  style: const TextStyle(fontWeight: FontWeight.w800),

                ),

                const SizedBox(height: 6),

                Text(duration,

                    style: TextStyle(

                      color: Theme.*of*(context)

                          .colorScheme

                          .onSurface

                          .*withOpacity*(.7),

                    )),

              ],

            ),

          ),

        ),

      ),

    );

  }

  // Section Statistiques (look "pro")

  Widget *\_statsSection*() {

    final stats = [

*\_stat*('+35', 'Questions de quiz'),

*\_stat*('98%', 'Satisfaction'),

*\_stat*('10 min', 'Sessions rapides'),

*\_stat*('3', 'Plateformes'),

    ];

**return** Padding(

      padding: const EdgeInsets.*symmetric*(vertical: 16),

      child: Container(

        decoration: BoxDecoration(

          gradient: LinearGradient(

            colors: [

              Theme.*of*(context).colorScheme.primary.*withOpacity*(.08),

              Theme.*of*(context).colorScheme.secondary.*withOpacity*(.08),

            ],

          ),

        ),

        padding: const EdgeInsets.*symmetric*(horizontal: 16, vertical: 18),

        child: LayoutBuilder(

          builder: (\_, c) {

            final w = c.maxWidth;

            final isWide = w > 900;

**return** Wrap(

              alignment: WrapAlignment.center,

              spacing: 24,

              runSpacing: 16,

              children: stats

                  .*map*((s) => SizedBox(

                        width: isWide ? (w / 5) : 160,

                        child: *\_glassCard*(child: s),

                      ))

                  .*toList*(),

            );

          },

        ),

      ),

    );

  }

  Widget *\_stat*(String value, String label) {

**return** Padding(

      padding: const EdgeInsets.*all*(14),

      child: Column(

        children: [

          Text(

            value,

            style: const TextStyle(

              fontSize: 26,

              fontWeight: FontWeight.w900,

              letterSpacing: -0.5,

            ),

          ),

          const SizedBox(height: 6),

          Text(label,

              textAlign: TextAlign.center,

              style: TextStyle(

                color: Theme.*of*(context).colorScheme.onSurface.*withOpacity*(.75),

              )),

        ],

      ),

    );

  }

  // Section Description BTS

  Widget *\_btsDescriptionSection*() {

**return** Padding(

      padding: const EdgeInsets.*fromLTRB*(16, 8, 16, 8),

      child: *\_elevCard*(

        child: Padding(

          padding: const EdgeInsets.*all*(18),

          child: Column(

            crossAxisAlignment: CrossAxisAlignment.start,

            children: [

*\_sectionTitle*('Le BTS, c’est quoi ?'),

              const SizedBox(height: 8),

              Text(

                "Le Brevet de Technicien Supérieur (BTS) est un diplôme d’État en deux ans, très orienté pratique et professionnalisation. "

                "Il prépare efficacement à l’insertion professionnelle ou à la poursuite d’études (Bachelor, Licence, etc.). "

                "Avec Réussir BTS, tu retrouves les notions clés, des entraînements ciblés et un suivi de progression pour maximiser tes chances.",

                style: TextStyle(

                  height: 1.35,

                  color:

                      Theme.*of*(context).colorScheme.onSurface.*withOpacity*(.85),

                ),

              ),

              const SizedBox(height: 12),

              Wrap(

                spacing: 10,

                runSpacing: 10,

                children: [

*\_chip*('Comptabilité'),

*\_chip*('Maths appli'),

*\_chip*('Eco-Gestion'),

*\_chip*('Culture Éco'),

*\_chip*('Projet tutoré'),

                ],

              ),

            ],

          ),

        ),

      ),

    );

  }

  // Section FAQ

  Widget *\_faqSection*() {

    final faqs = [

      {

        'q': 'Comment l’app peut m’aider à réussir mon BTS ?',

        'a':

            'En te proposant des cours résumés, des quiz intelligents et un suivi de progression clair pour rester motivé.',

      },

      {

        'q': 'Combien de temps faut-il réviser chaque jour ?',

        'a':

            'Nous recommandons 10 à 15 minutes de micro-learning quotidien pour progresser efficacement.',

      },

    ];

**return** Padding(

      padding: const EdgeInsets.*fromLTRB*(16, 8, 16, 8),

      child: Column(

        crossAxisAlignment: CrossAxisAlignment.start,

        children: [

*\_sectionTitle*('FAQ'),

          const SizedBox(height: 8),

*\_elevCard*(

            child: ListView.*separated*(

              shrinkWrap: true,

              physics: const NeverScrollableScrollPhysics(),

              itemCount: faqs.length,

              separatorBuilder: (\_, \_\_) => const Divider(height: 1),

              itemBuilder: (\_, i) {

                final f = faqs[i];

**return** ExpansionTile(

                  title: Text(

                    f['q']!,

                    style: const TextStyle(fontWeight: FontWeight.w700),

                  ),

                  childrenPadding:

                      const EdgeInsets.*symmetric*(horizontal: 16, vertical: 8),

                  children: [

                    Align(

                      alignment: Alignment.centerLeft,

                      child: Text(

                        f['a']!,

                        style: TextStyle(

                          color: Theme.*of*(context)

                              .colorScheme

                              .onSurface

                              .*withOpacity*(.8),

                        ),

                      ),

                    ),

                    const SizedBox(height: 8),

                  ],

                );

              },

            ),

          ),

        ],

      ),

    );

  }

  // CTA final

  Widget *\_ctaSection*() {

**return** Padding(

      padding: const EdgeInsets.*fromLTRB*(16, 8, 16, 16),

      child: *\_elevCard*(

        child: Container(

          decoration: BoxDecoration(

            gradient: LinearGradient(

              colors: [

                Theme.*of*(context).colorScheme.primary.*withOpacity*(.12),

                Theme.*of*(context).colorScheme.secondary.*withOpacity*(.12),

              ],

            ),

            borderRadius: BorderRadius.*circular*(16),

          ),

          padding: const EdgeInsets.*all*(18),

          child: Column(

            children: [

              Text(

                'Prêt·e à booster tes révisions ?',

                style: Theme.*of*(context).textTheme.titleLarge?.*copyWith*(

                      fontWeight: FontWeight.w900,

                    ),

              ),

              const SizedBox(height: 8),

              Text(

                'Crée un compte en 30 secondes ou connecte-toi pour reprendre là où tu t’étais arrêté·e.',

                textAlign: TextAlign.center,

                style: TextStyle(

                  color:

                      Theme.*of*(context).colorScheme.onSurface.*withOpacity*(.8),

                ),

              ),

              const SizedBox(height: 14),

              Wrap(

                spacing: 10,

                runSpacing: 10,

                alignment: WrapAlignment.center,

                children: [

                  ElevatedButton(

                    onPressed: () => Navigator.*pushNamed*(context, '/register'),

                    style: ElevatedButton.*styleFrom*(

                      padding: const EdgeInsets.*symmetric*(

                        horizontal: 20,

                        vertical: 14,

                      ),

                      shape: RoundedRectangleBorder(

                        borderRadius: BorderRadius.*circular*(14),

                      ),

                    ),

                    child: const Text("S'inscrire"),

                  ),

                  OutlinedButton(

                    onPressed: () => Navigator.*pushNamed*(context, '/login'),

                    style: OutlinedButton.*styleFrom*(

                      padding: const EdgeInsets.*symmetric*(

                        horizontal: 20,

                        vertical: 14,

                      ),

                      shape: RoundedRectangleBorder(

                        borderRadius: BorderRadius.*circular*(14),

                      ),

                    ),

                    child: const Text('Se connecter'),

                  ),

                ],

              ),

            ],

          ),

        ),

      ),

    );

  }

  // Footer avec liens (À propos, Politique, Contact)

  Widget *\_footerSection*() {

    final onSurfaceMuted =

        Theme.*of*(context).colorScheme.onSurface.*withOpacity*(.7);

**return** Container(

      width: double.infinity,

      decoration: BoxDecoration(

        gradient: LinearGradient(

          colors: [

            Theme.*of*(context).colorScheme.surfaceVariant.*withOpacity*(.5),

            Theme.*of*(context).colorScheme.surface,

          ],

          begin: Alignment.topCenter,

          end: Alignment.bottomCenter,

        ),

        border: Border(

          top: BorderSide(

            color: Theme.*of*(context).dividerColor.*withOpacity*(.4),

          ),

        ),

      ),

      padding: const EdgeInsets.*fromLTRB*(16, 20, 16, 28),

      child: SafeArea(

        top: false,

        child: Column(

          crossAxisAlignment: CrossAxisAlignment.stretch,

          children: [

            Wrap(

              alignment: WrapAlignment.spaceBetween,

              crossAxisAlignment: WrapCrossAlignment.center,

              runSpacing: 12,

              children: [

                Row(

                  mainAxisSize: MainAxisSize.min,

                  children: [

                    const Icon(Icons.school\_outlined),

                    const SizedBox(width: 8),

                    Text(

                      'Réussir BTS',

                      style: Theme.*of*(context).textTheme.titleMedium?.*copyWith*(

                            fontWeight: FontWeight.w900,

                          ),

                    ),

                  ],

                ),

                Wrap(

                  spacing: 8,

                  runSpacing: 8,

                  children: [

*\_footerLink*(

                      label: 'À propos',

                      onTap: () => *\_openBottomSheet*(

                        title: 'À propos',

                        children: [

                          const SizedBox(height: 8),

                          const Text(

                            "Réussir BTS est une application pensée pour des révisions modernes et efficaces. "

                            "Notre objectif : te fournir le bon contenu au bon moment, avec une expérience fluide sur toutes les plateformes.",

                          ),

                          const SizedBox(height: 12),

                          Text(

                            "Version : 1.0.0",

                            style: TextStyle(color: onSurfaceMuted),

                          ),

                        ],

                      ),

                    ),

*\_footerLink*(

                      label: 'Politique de confidentialité',

                      onTap: () => *\_openBottomSheet*(

                        title: 'Politique de confidentialité',

                        children: const [

                          SizedBox(height: 8),

                          Text(

                            "Nous respectons ta vie privée. Les données collectées servent à l’authentification, la sauvegarde des progrès et l’amélioration du service. "

                            "En mode hors-ligne, certaines informations sont stockées localement sur ton appareil.",

                          ),

                          SizedBox(height: 8),

                          Text(

                            "En te connectant, tu acceptes notre politique de confidentialité.",

                          ),

                        ],

                      ),

                    ),

*\_footerLink*(

                      label: 'Conditions d’utilisation',

                      onTap: () => *\_openBottomSheet*(

                        title: "Conditions d’utilisation",

                        children: const [

                          SizedBox(height: 8),

                          Text(

                            "L’application est fournie en l’état. Les contenus pédagogiques sont destinés à l’entraînement. "

                            "Il est interdit de copier ou redistribuer sans autorisation.",

                          ),

                        ],

                      ),

                    ),

*\_footerLink*(

                      label: 'Contact',

                      onTap: () => *\_openBottomSheet*(

                        title: 'Contact',

                        children: const [

                          SizedBox(height: 8),

                          Text("Email : devopsdesigngest@gmail.com"),

                          SizedBox(height: 6),

                          Text("Réponse sous 48h ouvrées."),

                        ],

                      ),

                    ),

                  ],

                ),

              ],

            ),

            const SizedBox(height: 18),

            Text(

              '© ${DateTime.*now*().*year*} Réussir BTS — Tous droits réservés.',

              textAlign: TextAlign.center,

              style: TextStyle(color: onSurfaceMuted),

            ),

          ],

        ),

      ),

    );

  }

  Widget *\_footerLink*({required String label, required VoidCallback onTap}) {

**return** InkWell(

      borderRadius: BorderRadius.*circular*(8),

      onTap: onTap,

      child: Padding(

        padding: const EdgeInsets.*symmetric*(horizontal: 10, vertical: 6),

        child: Text(

          label,

          style: const TextStyle(fontWeight: FontWeight.w700),

        ),

      ),

    );

  }

  // ---------- Helpers UI ----------

  Widget *\_sectionTitle*(String text) {

**return** Text(

      text,

      style: Theme.*of*(context).textTheme.titleLarge?.*copyWith*(

            fontWeight: FontWeight.w900,

          ),

    );

  }

  Widget *\_iconCircle*(IconData icon) {

**return** Container(

      width: 38,

      height: 38,

      decoration: BoxDecoration(

        color: Theme.*of*(context).colorScheme.primary.*withOpacity*(.12),

        shape: BoxShape.circle,

      ),

      child: Icon(icon),

    );

  }

  Widget *\_elevCard*({required Widget child, Color? color}) {

**return** Container(

      decoration: BoxDecoration(

        color: color ?? Theme.*of*(context).colorScheme.surface,

        borderRadius: BorderRadius.*circular*(16),

        boxShadow: [

          BoxShadow(

            color: Colors.black.*withOpacity*(.06),

            blurRadius: 14,

            spreadRadius: 2,

            offset: const Offset(0, 6),

          ),

        ],

        border: Border.*all*(

          color: Theme.*of*(context).dividerColor.*withOpacity*(.25),

        ),

      ),

      child: ClipRRect(

        borderRadius: BorderRadius.*circular*(16),

        child: child,

      ),

    );

  }

  Widget *\_glassCard*({required Widget child}) {

**return** ClipRRect(

      borderRadius: BorderRadius.*circular*(16),

      child: BackdropFilter(

        filter: ImageFilter.*blur*(sigmaX: 10, sigmaY: 10),

        child: Container(

          decoration: BoxDecoration(

            color: Theme.*of*(context).colorScheme.surface.*withOpacity*(.6),

            borderRadius: BorderRadius.*circular*(16),

            border: Border.*all*(

              color: Theme.*of*(context).colorScheme.outline.*withOpacity*(.3),

            ),

          ),

          padding: const EdgeInsets.*all*(12),

          child: child,

        ),

      ),

    );

  }

  Widget *\_chip*(String label) {

**return** Chip(

      label: Text(label),

      visualDensity: VisualDensity.compact,

      padding: const EdgeInsets.*symmetric*(horizontal: 6),

    );

  }

  // Fond animé avec blobs/gradients doux (sans package externe)

  Widget *\_animatedBackground*() {

**return** AnimatedBuilder(

      animation: \_bgAnim,

      builder: (\_, \_\_) {

        final t = \_bgAnim.value;

**return** Stack(

          children: [

            // Gradient principal

            Container(

              decoration: const BoxDecoration(

                gradient: LinearGradient(

                  colors: [Color(0xFFF5F7FF), Color(0xFFEFF3FF)],

                  begin: Alignment.topLeft,

                  end: Alignment.bottomRight,

                ),

              ),

            ),

            // Blobs décoratifs

            Positioned(

              top: 80 + 20 \* (t - 0.5),

              left: -120,

              child: *\_blob*(const Color(0xFFB3C7FF), 260),

            ),

            Positioned(

              bottom: -100,

              right: -80 + 25 \* (0.5 - t),

              child: *\_blob*(const Color(0xFFAAD9D9), 220),

            ),

            Positioned(

              top: 300,

              right: -140,

              child: *\_blob*(const Color(0xFFE5B8FF), 280),

            ),

          ],

        );

      },

    );

  }

  Widget *\_blob*(Color color, double size) {

**return** Transform.*rotate*(

      angle: 0.5,

      child: Container(

        width: size,

        height: size,

        decoration: BoxDecoration(

          color: color.*withOpacity*(.28),

          shape: BoxShape.circle,

        ),

      ),

    );

  }

  // ---------- BUILD ----------

  @override

  Widget *build*(BuildContext context) {

**return** Scaffold(

      body: Stack(

        children: [

*\_animatedBackground*(),

          // Contenu scrollable

          CustomScrollView(

            slivers: [

              SliverAppBar(

                pinned: true,

                floating: false,

                snap: false,

                elevation: 0,

                backgroundColor:

                    Theme.*of*(context).colorScheme.surface.*withOpacity*(.9),

                flexibleSpace: const FlexibleSpaceBar(

                  titlePadding:

                      EdgeInsetsDirectional.*only*(start: 16, bottom: 12),

                  title: Text('Réussir BTS'),

                ),

              ),

              SliverToBoxAdapter(child: *\_networkBanner*()),

              SliverToBoxAdapter(child: *\_heroSection*()),

              SliverToBoxAdapter(child: *\_featuresSection*()),

              // SliverToBoxAdapter(child: \_microlearningSection()),

              SliverToBoxAdapter(child: *\_statsSection*()),

              SliverToBoxAdapter(child: *\_btsDescriptionSection*()),

              SliverToBoxAdapter(child: *\_faqSection*()),

              SliverToBoxAdapter(child: *\_ctaSection*()),

              SliverToBoxAdapter(child: *\_footerSection*()),

            ],

          ),

        ],

      ),

    );

  }

}  
// 📁 lib/provider/theme\_provider.dart

import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

class ThemeProvider with ChangeNotifier {

  bool \_isDark = false;

  bool get isDark => \_isDark;

  ThemeProvider() {

*\_loadTheme*();

  }

  void *toggleTheme*() {

    \_isDark = !\_isDark;

*\_saveTheme*();

*notifyListeners*();

  }

  Future<void> *\_loadTheme*() **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

    \_isDark = prefs.*getBool*('isDark') ?? false;

*notifyListeners*();

  }

  Future<void> *\_saveTheme*() **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

**await** prefs.*setBool*('isDark', \_isDark);

  }

}

// 📁 lib/provider/user\_provider.dart

import 'package:flutter/material.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import '../models/user\_model.dart';

class UserProvider with ChangeNotifier {

  final FirebaseAuth \_auth = FirebaseAuth.instance;

  final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

  AppUser? \_user;

  AppUser? get user => \_user;

  /// ===== Inscription d’un utilisateur =====

  Future<void> *register*({

    required String name,

    required String email,

    required String phone,

    required String school,

    required String speciality,

    required String password,

    required bool offline,

    String role = 'etudiant',

    String niveau = 'BTS1',

  }) **async** {

**try** {

      // 🔹 Création du compte Firebase Auth

      UserCredential cred = **await** \_auth.*createUserWithEmailAndPassword*(

        email: email,

        password: password,

      );

      final uid = cred.user!.uid;

      // 🔹 Création de notre modèle AppUser

      AppUser newUser = AppUser(

        uid: uid,

        name: name,

        email: email,

        phone: phone,

        school: school,

        speciality: speciality,

        role: role,

        password: password, // ⚠️ stocker en clair seulement pour dev

        niveau: niveau,

      );

      // 🔹 Sauvegarde Firestore

**await** \_firestore.*collection*("users").*doc*(uid).*set*(newUser.*toMap*());

      \_user = newUser;

*notifyListeners*();

    } **on** FirebaseAuthException **catch** (e) {

**throw** e.message ?? "Erreur d'inscription";

    }

  }

  /// ===== Connexion =====

  Future<void> *login*(String email, String password) **async** {

**try** {

      UserCredential cred = **await** \_auth.*signInWithEmailAndPassword*(

        email: email,

        password: password,

      );

      final uid = cred.user!.uid;

      final doc = **await** \_firestore.*collection*("users").*doc*(uid).*get*();

**if** (doc.exists) {

        \_user = AppUser.*fromMap*(doc.*data*()!);

*notifyListeners*();

      }

    } **on** FirebaseAuthException **catch** (e) {

**throw** e.message ?? "Erreur de connexion";

    }

  }

  /// ===== Déconnexion =====

  Future<void> *logout*() **async** {

**await** \_auth.*signOut*();

    \_user = null;

*notifyListeners*();

  }

  /// ===== Charger l’utilisateur courant =====

  Future<void> *loadCurrentUser*() **async** {

    final firebaseUser = \_auth.currentUser;

**if** (firebaseUser != null) {

      final doc =

**await** \_firestore.*collection*("users").*doc*(firebaseUser.uid).*get*();

**if** (doc.exists) {

        \_user = AppUser.*fromMap*(doc.*data*()!);

*notifyListeners*();

      }

    }

  }

  void *setUser*(AppUser user) {

    \_user = user;

*notifyListeners*();

  }

}

// lib/screens/admin/admin\_panel.dart

import 'package:flutter/material.dart';

import 'student\_crud\_screen.dart';

import 'quiz\_crud\_screen.dart';

import 'course\_crud\_screen.dart';

class AdminPanel extends StatefulWidget {

  const AdminPanel({*super*.key});

  @override

  State<AdminPanel> *createState*() => \_AdminPanelState();

}

class \_AdminPanelState extends State<AdminPanel>

    with SingleTickerProviderStateMixin {

  late TabController \_tabController;

  @override

  void *initState*() {

*super*.*initState*();

    \_tabController = TabController(length: 3, vsync: *this*);

  }

  @override

  void *dispose*() {

    \_tabController.*dispose*();

*super*.*dispose*();

  }

  @override

  Widget *build*(BuildContext context) {

**return** Scaffold(

      appBar: AppBar(

        title: const Text(

          "Panneau Admin",

          style: TextStyle(

            fontWeight: FontWeight.bold,

            fontSize: 20,

          ),

        ),

        centerTitle: true,

        backgroundColor: Colors.indigo,

        bottom: TabBar(

          controller: \_tabController,

          labelColor: Colors.white, // Texte onglet actif

          unselectedLabelColor: Colors.white70, // Texte onglets inactifs

          indicatorColor: Colors.amber, // Couleur soulignement

          indicatorWeight: 3,

          tabs: const [

            Tab(icon: Icon(Icons.person), text: "Étudiants"),

            Tab(icon: Icon(Icons.quiz), text: "Quiz"),

            Tab(icon: Icon(Icons.library\_books), text: "Cours"),

          ],

        ),

      ),

      body: TabBarView(

        controller: \_tabController,

        children: const [

          StudentCrud(),

          QuizCrud(),

          CourseCrud(), // ✅ gère fiches + vidéos

        ],

      ),

    );

  }

}

// lib/screens/admin/course\_crud\_screen.dart

import 'package:flutter/material.dart';

import '../../services/firestore\_service.dart';

import '../../models/fiche\_model.dart';

import '../../models/video\_model.dart';

class CourseCrud extends StatefulWidget {

  const CourseCrud({*super*.key});

  @override

  State<CourseCrud> *createState*() => \_CourseCrudState();

}

class \_CourseCrudState extends State<CourseCrud> {

  final FirestoreService \_firestore = FirestoreService();

  bool \_loading = true;

  List<String> \_modules = []; // ex: ['m1', 'm2']

  String? \_selectedModule;

  List<Fiche> \_fiches = [];

  List<VideoItem> \_videos = [];

  @override

  void *initState*() {

*super*.*initState*();

*\_loadModules*();

  }

  Future<void> *\_loadModules*() **async** {

*setState*(() => \_loading = true);

**try** {

      // On récupère tous les modules depuis Firestore

      final modules = **await** \_firestore.*getModules*();

      \_modules = modules.*map*((m) => m.id).*toList*();

**if** (\_modules.isNotEmpty) \_selectedModule = \_modules.first;

**await** *\_loadContent*();

    } **catch** (e) {

*print*("Erreur chargement modules : $*e*");

    } **finally** {

*setState*(() => \_loading = false);

    }

  }

  Future<void> *\_loadContent*() **async** {

**if** (\_selectedModule == null) **return**;

*setState*(() => \_loading = true);

    final fiches = **await** \_firestore.*getFichesForModule*(\_selectedModule!);

    final videos = **await** \_firestore.*getVideosForModule*(\_selectedModule!);

*setState*(() {

      \_fiches = fiches;

      \_videos = videos;

      \_loading = false;

    });

  }

  Future<void> *\_openDialog*({String? docId, String type = "fiche"}) **async** {

    final isFiche = type == "fiche";

    final titleController = TextEditingController();

    final urlController = TextEditingController();

    final pagesController = TextEditingController();

    String level = "BTS1"; // Par défaut

**if** (docId != null) {

**if** (isFiche) {

        final fiche = \_fiches.*firstWhere*((f) => f.id == docId);

        titleController.text = fiche.title;

        urlController.text = fiche.url;

        pagesController.text = fiche.pages?.*toString*() ?? "";

        level = fiche.level ?? "BTS1";

      } **else** {

        final video = \_videos.*firstWhere*((v) => v.id == docId);

        titleController.text = video.title;

        urlController.text = video.url;

        level = video.level ?? "BTS1";

      }

    }

**await** *showDialog*(

      context: context,

      builder: (\_) {

**return** AlertDialog(

          title: Text(docId == null

              ? "Ajouter ${*isFiche* ? 'fiche' : 'vidéo'}"

              : "Modifier ${*isFiche* ? 'fiche' : 'vidéo'}"),

          content: SingleChildScrollView(

            child: Column(

              children: [

                TextField(

                  controller: titleController,

                  decoration: const InputDecoration(labelText: "Titre"),

                ),

                TextField(

                  controller: urlController,

                  decoration: InputDecoration(

                      labelText: isFiche ? "URL PDF" : "URL vidéo"),

                ),

**if** (isFiche)

                  TextField(

                    controller: pagesController,

                    decoration: const InputDecoration(labelText: "Pages"),

                    keyboardType: TextInputType.number,

                  ),

                DropdownButton<String>(

                  value: level,

                  items: const [

                    DropdownMenuItem(value: "BTS1", child: Text("BTS1")),

                    DropdownMenuItem(value: "BTS2", child: Text("BTS2")),

                  ],

                  onChanged: (val) {

**if** (val != null) level = val;

                  },

                ),

              ],

            ),

          ),

          actions: [

            TextButton(

              onPressed: () => Navigator.*pop*(context),

              child: const Text("Annuler"),

            ),

            ElevatedButton(

              onPressed: () **async** {

**if** (\_selectedModule == null) **return**;

**if** (isFiche) {

                  final fiche = Fiche(

                    id: docId ?? "",

                    title: titleController.text.*trim*(),

                    url: urlController.text.*trim*(),

                    pages: int.*tryParse*(pagesController.text.*trim*()) ?? 0,

                    level: level,

                    tags: [],

                  );

**if** (docId == null) {

**await** \_firestore.*addFiche*(\_selectedModule!, fiche);

                  } **else** {

**await** \_firestore.*updateFiche*(\_selectedModule!, fiche);

                  }

                } **else** {

                  final video = VideoItem(

                    id: docId ?? "",

                    title: titleController.text.*trim*(),

                    url: urlController.text.*trim*(),

                    duration: 0,

                    level: level,

                    tags: [],

                  );

**if** (docId == null) {

**await** \_firestore.*addVideo*(\_selectedModule!, video);

                  } **else** {

**await** \_firestore.*updateVideo*(\_selectedModule!, video);

                  }

                }

                Navigator.*pop*(context);

**await** *\_loadContent*();

              },

              child: Text(docId == null ? "Ajouter" : "Modifier"),

            ),

          ],

        );

      },

    );

  }

  Future<void> *\_deleteItem*(String id, String type) **async** {

**if** (\_selectedModule == null) **return**;

**if** (type == "fiche") {

**await** \_firestore.*deleteFiche*(\_selectedModule!, id);

    } **else** {

**await** \_firestore.*deleteVideo*(\_selectedModule!, id);

    }

**await** *\_loadContent*();

  }

  @override

  Widget *build*(BuildContext context) {

**return** Scaffold(

      appBar: AppBar(

        title: const Text("Gestion Fiches & Vidéos"),

        actions: [

**if** (\_modules.isNotEmpty)

            DropdownButton<String>(

              value: \_selectedModule,

              items: \_modules

                  .*map*((m) => DropdownMenuItem(value: m, child: Text(m)))

                  .*toList*(),

              onChanged: (val) {

**if** (val != null) {

*setState*(() => \_selectedModule = val);

*\_loadContent*();

                }

              },

            ),

        ],

      ),

      body: \_loading

          ? const Center(child: CircularProgressIndicator())

          : ListView(

              children: [

                const Padding(

                  padding: EdgeInsets.*all*(8.0),

                  child: Text("📘 Fiches", style: TextStyle(fontSize: 18)),

                ),

                ...\_fiches.*map*(

                  (f) => Card(

                    child: ListTile(

                      leading: const Icon(Icons.article),

                      title: Text(f.title),

                      subtitle: Text("Fiche • ${*f*.*level*}"),

                      trailing: Row(

                        mainAxisSize: MainAxisSize.min,

                        children: [

                          IconButton(

                            icon: const Icon(Icons.edit, color: Colors.blue),

                            onPressed: () =>

*\_openDialog*(docId: f.id, type: "fiche"),

                          ),

                          IconButton(

                            icon: const Icon(Icons.delete, color: Colors.red),

                            onPressed: () => *\_deleteItem*(f.id, "fiche"),

                          ),

                        ],

                      ),

                    ),

                  ),

                ),

                const Padding(

                  padding: EdgeInsets.*all*(8.0),

                  child: Text("🎬 Vidéos", style: TextStyle(fontSize: 18)),

                ),

                ...\_videos.*map*(

                  (v) => Card(

                    child: ListTile(

                      leading: const Icon(Icons.video\_library),

                      title: Text(v.title),

                      subtitle: Text("Vidéo • ${*v*.*level*}"),

                      trailing: Row(

                        mainAxisSize: MainAxisSize.min,

                        children: [

                          IconButton(

                            icon: const Icon(Icons.edit, color: Colors.blue),

                            onPressed: () =>

*\_openDialog*(docId: v.id, type: "video"),

                          ),

                          IconButton(

                            icon: const Icon(Icons.delete, color: Colors.red),

                            onPressed: () => *\_deleteItem*(v.id, "video"),

                          ),

                        ],

                      ),

                    ),

                  ),

                ),

              ],

            ),

      floatingActionButton: Column(

        mainAxisSize: MainAxisSize.min,

        children: [

          FloatingActionButton.*extended*(

            heroTag: "fiche",

            icon: const Icon(Icons.article),

            label: const Text("Ajouter fiche"),

            onPressed: () => *\_openDialog*(type: "fiche"),

          ),

          const SizedBox(height: 10),

          FloatingActionButton.*extended*(

            heroTag: "video",

            icon: const Icon(Icons.video\_library),

            label: const Text("Ajouter vidéo"),

            onPressed: () => *\_openDialog*(type: "video"),

          ),

        ],

      ),

    );

  }

}

// lib/screens/admin/quiz\_crud\_screen.dart

import 'package:flutter/material.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

class QuizCrud extends StatefulWidget {

  const QuizCrud({*super*.key});

  @override

  State<QuizCrud> *createState*() => \_QuizCrudState();

}

class \_QuizCrudState extends State<QuizCrud> {

  String \_selectedModuleId = "m1"; // module par défaut

  /// 🔹 Ajout ou édition quiz

  Future<void> *\_openQuizDialog*(BuildContext context,

      {String? moduleId, String? docId, Map<String, dynamic>? data}) **async** {

    // 🔹 Contrôleurs principaux

    final titleController =

        TextEditingController(text: data != null ? data['title'] : '');

    final descController =

        TextEditingController(text: data != null ? data['description'] : '');

    final durationController = TextEditingController(

        text:

            (data != null ? (data['durationSeconds'] ?? 600) : 600).*toString*());

    final orderController = TextEditingController(

        text: (data != null ? (data['order'] ?? 0) : 0).*toString*());

    bool allowRetake = data != null ? (data['allowRetake'] ?? true) : true;

    Map<String, dynamic> badgeThresholds =

        data?['badgeThresholds'] ?? {"gold": 90, "silver": 75, "bronze": 50};

    // 🔹 Questions

    List<Map<String, dynamic>> questions =

        data != null ? List<Map<String, dynamic>>.*from*(data['questions']) : [];

    void *addQuestion*() {

      questions.*add*({

        "question": "",

        "options": ["", "", "", ""],

        "correctIndex": 0,

        "explanation": ""

      });

    }

**await** *showDialog*(

      context: context,

      builder: (\_) {

**return** StatefulBuilder(

          builder: (context, setState) {

**return** AlertDialog(

              title: Text(docId == null ? "Ajouter quiz" : "Modifier quiz"),

              content: SingleChildScrollView(

                child: Column(

                  children: [

                    TextField(

                      controller: titleController,

                      decoration:

                          const InputDecoration(labelText: "Titre du quiz"),

                    ),

                    TextField(

                      controller: descController,

                      decoration:

                          const InputDecoration(labelText: "Description"),

                    ),

                    const SizedBox(height: 10),

                    Row(

                      children: [

                        Expanded(

                          child: TextField(

                            decoration: const InputDecoration(

                                labelText: "Durée (secondes)"),

                            keyboardType: TextInputType.number,

                            controller: durationController,

                          ),

                        ),

                        const SizedBox(width: 10),

                        Expanded(

                          child: TextField(

                            decoration:

                                const InputDecoration(labelText: "Ordre"),

                            keyboardType: TextInputType.number,

                            controller: orderController,

                          ),

                        ),

                      ],

                    ),

                    SwitchListTile(

                      value: allowRetake,

                      title: const Text("Autoriser plusieurs tentatives"),

                      onChanged: (val) {

*setState*(() => allowRetake = val);

                      },

                    ),

                    const Divider(),

                    const Text("Seuils des badges"),

                    Row(

                      children: [

                        Expanded(

                          child: TextField(

                            decoration:

                                const InputDecoration(labelText: "Or (%)"),

                            keyboardType: TextInputType.number,

                            controller: TextEditingController(

                                text: badgeThresholds['gold'].*toString*()),

                            onChanged: (val) => badgeThresholds['gold'] =

                                int.*tryParse*(val) ?? 90,

                          ),

                        ),

                        Expanded(

                          child: TextField(

                            decoration:

                                const InputDecoration(labelText: "Argent (%)"),

                            keyboardType: TextInputType.number,

                            controller: TextEditingController(

                                text: badgeThresholds['silver'].*toString*()),

                            onChanged: (val) => badgeThresholds['silver'] =

                                int.*tryParse*(val) ?? 75,

                          ),

                        ),

                        Expanded(

                          child: TextField(

                            decoration:

                                const InputDecoration(labelText: "Bronze (%)"),

                            keyboardType: TextInputType.number,

                            controller: TextEditingController(

                                text: badgeThresholds['bronze'].*toString*()),

                            onChanged: (val) => badgeThresholds['bronze'] =

                                int.*tryParse*(val) ?? 50,

                          ),

                        ),

                      ],

                    ),

                    const Divider(),

                    Column(

                      children: [

                        ...List.*generate*(questions.length, (qIdx) {

                          final q = questions[qIdx];

                          final qController =

                              TextEditingController(text: q['question']);

                          final expController =

                              TextEditingController(text: q['explanation']);

**return** Card(

                            margin: const EdgeInsets.*symmetric*(vertical: 5),

                            child: Padding(

                              padding: const EdgeInsets.*all*(8.0),

                              child: Column(

                                children: [

                                  TextField(

                                    decoration: InputDecoration(

                                        labelText: "Question ${*qIdx* + 1}"),

                                    controller: qController,

                                    onChanged: (val) =>

                                        q['question'] = val.*trim*(),

                                  ),

                                  const SizedBox(height: 5),

                                  ...List.*generate*(4, (optIdx) {

                                    final optController = TextEditingController(

                                        text: q['options'][optIdx]);

**return** TextField(

                                      decoration: InputDecoration(

                                          labelText: "Option ${*optIdx* + 1}"),

                                      controller: optController,

                                      onChanged: (val) =>

                                          q['options'][optIdx] = val.*trim*(),

                                    );

                                  }),

                                  DropdownButton<int>(

                                    value: q['correctIndex'],

                                    items: List.*generate*(

                                      4,

                                      (i) => DropdownMenuItem(

                                        value: i,

                                        child: Text(

                                            "Bonne réponse : Option ${*i* + 1}"),

                                      ),

                                    ),

                                    onChanged: (val) {

**if** (val != null) {

*setState*(() => q['correctIndex'] = val);

                                      }

                                    },

                                  ),

                                  TextField(

                                    decoration: const InputDecoration(

                                        labelText: "Explication"),

                                    controller: expController,

                                    onChanged: (val) =>

                                        q['explanation'] = val.*trim*(),

                                  ),

                                  Align(

                                    alignment: Alignment.centerRight,

                                    child: IconButton(

                                      icon: const Icon(Icons.delete,

                                          color: Colors.red),

                                      onPressed: () {

*setState*(() {

                                          questions.*removeAt*(qIdx);

                                        });

                                      },

                                    ),

                                  ),

                                ],

                              ),

                            ),

                          );

                        }),

                        const SizedBox(height: 10),

                        ElevatedButton.*icon*(

                          icon: const Icon(Icons.add),

                          label: const Text("Ajouter une question"),

                          onPressed: () {

*setState*(() => *addQuestion*());

                          },

                        )

                      ],

                    ),

                  ],

                ),

              ),

              actions: [

                TextButton(

                  onPressed: () => Navigator.*pop*(context),

                  child: const Text("Annuler"),

                ),

                ElevatedButton(

                  onPressed: () **async** {

                    final dataToSave = {

                      "id": docId ??

                          DateTime.*now*().millisecondsSinceEpoch.*toString*(),

                      "title": titleController.text.*trim*(),

                      "description": descController.text.*trim*(),

                      "durationSeconds":

                          int.*tryParse*(durationController.text) ?? 600,

                      "allowRetake": allowRetake,

                      "order": int.*tryParse*(orderController.text) ?? 0,

                      "badgeThresholds": badgeThresholds,

                      "questions": questions,

                      "questionCount": questions.length,

                      "createdAt": FieldValue.*serverTimestamp*(),

                      "updatedAt": FieldValue.*serverTimestamp*(),

                      "moduleId": moduleId ?? \_selectedModuleId,

                    };

                    final collection = FirebaseFirestore.instance

                        .*collection*("modules")

                        .*doc*(moduleId ?? \_selectedModuleId)

                        .*collection*("quizzes");

**if** (docId == null) {

                      // 🔹 Ajout

**await** collection

                          .*doc*(dataToSave['id'].*toString*())

                          .*set*(dataToSave, SetOptions(merge: true));

                    } **else** {

                      // 🔹 Édition

**await** collection.*doc*(docId).*update*(dataToSave);

                    }

                    Navigator.*pop*(context);

                  },

                  child: Text(docId == null ? "Ajouter" : "Modifier"),

                ),

              ],

            );

          },

        );

      },

    );

  }

  /// 🔹 Suppression quiz

  Future<void> *\_deleteQuiz*(String moduleId, String docId) **async** {

**await** FirebaseFirestore.instance

        .*collection*("modules")

        .*doc*(moduleId)

        .*collection*("quizzes")

        .*doc*(docId)

        .*delete*();

  }

  @override

  Widget *build*(BuildContext context) {

**return** Scaffold(

      appBar: AppBar(title: const Text("Gestion Quiz")),

      body: Column(

        children: [

          // 🔹 Sélecteur de module

          Padding(

            padding: const EdgeInsets.*all*(8.0),

            child: DropdownButton<String>(

              value: \_selectedModuleId,

              items: const [

                DropdownMenuItem(value: "m1", child: Text("Math")),

                DropdownMenuItem(value: "m2", child: Text("Comptabilite")),

                DropdownMenuItem(value: "m3", child: Text("Informatique")),

              ],

              onChanged: (val) {

**if** (val != null) {

*setState*(() => \_selectedModuleId = val);

                }

              },

            ),

          ),

          Expanded(

            child: StreamBuilder<QuerySnapshot>(

              stream: FirebaseFirestore.instance

                  .*collection*("modules")

                  .*doc*(\_selectedModuleId)

                  .*collection*("quizzes")

                  .*orderBy*("createdAt", descending: true)

                  .*snapshots*(),

              builder: (context, snapshot) {

**if** (!snapshot.hasData) {

**return** const Center(child: CircularProgressIndicator());

                }

                final docs = snapshot.data!.docs;

**if** (docs.isEmpty) {

**return** const Center(child: Text("Aucun quiz disponible"));

                }

**return** ListView.*builder*(

                  itemCount: docs.length,

                  itemBuilder: (context, index) {

                    final doc = docs[index];

                    final data = doc.*data*() as Map<String, dynamic>;

**return** Card(

                      child: ListTile(

                        leading:

                            const Icon(Icons.quiz, color: Colors.deepPurple),

                        title: Text(data['title'] ?? "Sans titre"),

                        subtitle: Text(

                            "${*data*['description'] ?? ''} • ${*data*['questionCount'] ?? 0} questions"),

                        trailing: Row(

                          mainAxisSize: MainAxisSize.min,

                          children: [

                            IconButton(

                              icon: const Icon(Icons.edit, color: Colors.blue),

                              onPressed: () => *\_openQuizDialog*(

                                context,

                                moduleId: \_selectedModuleId,

                                docId: doc.id,

                                data: data,

                              ),

                            ),

                            IconButton(

                              icon: const Icon(Icons.delete, color: Colors.red),

                              onPressed: () =>

*\_deleteQuiz*(\_selectedModuleId, doc.id),

                            ),

                          ],

                        ),

                      ),

                    );

                  },

                );

              },

            ),

          ),

        ],

      ),

      floatingActionButton: FloatingActionButton.*extended*(

        icon: const Icon(Icons.add),

        label: const Text("Ajouter quiz"),

        onPressed: () => *\_openQuizDialog*(context, moduleId: \_selectedModuleId),

      ),

    );

  }

}

// lib/screens/admin/student\_crud\_screen.dart

import 'package:flutter/material.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

class StudentCrud extends StatelessWidget {

  const StudentCrud({*super*.key});

  /// 🔹 Ajout ou édition étudiant

  Future<void> *\_openStudentDialog*(BuildContext context,

      {String? docId, Map<String, dynamic>? data}) **async** {

    final nameController =

        TextEditingController(text: data != null ? data['name'] : '');

    final emailController =

        TextEditingController(text: data != null ? data['email'] : '');

    final phoneController =

        TextEditingController(text: data != null ? data['phone'] : '');

    final schoolController =

        TextEditingController(text: data != null ? data['school'] : '');

    final specialityController =

        TextEditingController(text: data != null ? data['speciality'] : '');

    final passwordController =

        TextEditingController(text: data != null ? data['password'] : '');

    String role = data != null ? data['role'] ?? 'etudiant' : 'etudiant';

    String niveau = data != null ? data['niveau'] ?? 'BTS1' : 'BTS1';

**await** *showDialog*(

      context: context,

      builder: (\_) {

**return** StatefulBuilder(

          builder: (context, setState) {

**return** AlertDialog(

              shape: RoundedRectangleBorder(

                borderRadius: BorderRadius.*circular*(16),

              ),

              title: Text(

                docId == null

                    ? "➕ Ajouter un étudiant"

                    : "✏️ Modifier étudiant",

                style: const TextStyle(fontWeight: FontWeight.bold),

              ),

              content: SingleChildScrollView(

                child: Column(

                  crossAxisAlignment: CrossAxisAlignment.start,

                  children: [

*\_buildTextField*("Nom complet", nameController),

*\_buildTextField*("Email", emailController,

                        keyboardType: TextInputType.emailAddress),

*\_buildTextField*("Téléphone", phoneController,

                        keyboardType: TextInputType.phone),

*\_buildTextField*("École", schoolController),

*\_buildTextField*("Spécialité", specialityController),

**if** (docId == null) // mot de passe seulement à la création

*\_buildTextField*("Mot de passe", passwordController,

                          isPassword: true),

                    const SizedBox(height: 12),

                    // Dropdowns

                    Row(

                      children: [

                        Expanded(

                          child: DropdownButtonFormField<String>(

                            value: role,

                            decoration: const InputDecoration(

                              labelText: "Rôle",

                              border: OutlineInputBorder(),

                            ),

                            items: const [

                              DropdownMenuItem(

                                  value: "etudiant", child: Text("Étudiant")),

                              DropdownMenuItem(

                                  value: "admin", child: Text("Admin")),

                            ],

                            onChanged: (val) {

**if** (val != null) *setState*(() => role = val);

                            },

                          ),

                        ),

                        const SizedBox(width: 12),

                        Expanded(

                          child: DropdownButtonFormField<String>(

                            value: niveau,

                            decoration: const InputDecoration(

                              labelText: "Niveau",

                              border: OutlineInputBorder(),

                            ),

                            items: const [

                              DropdownMenuItem(

                                  value: "BTS1", child: Text("BTS 1")),

                              DropdownMenuItem(

                                  value: "BTS2", child: Text("BTS 2")),

                            ],

                            onChanged: (val) {

**if** (val != null) *setState*(() => niveau = val);

                            },

                          ),

                        ),

                      ],

                    ),

                  ],

                ),

              ),

              actions: [

                TextButton(

                  onPressed: () => Navigator.*pop*(context),

                  child: const Text("Annuler"),

                ),

                ElevatedButton.*icon*(

                  icon: const Icon(Icons.save),

                  label: Text(docId == null ? "Ajouter" : "Modifier"),

                  style: ElevatedButton.*styleFrom*(

                    backgroundColor: Colors.indigo,

                    foregroundColor: Colors.white,

                  ),

                  onPressed: () **async** {

                    final collection =

                        FirebaseFirestore.instance.*collection*("users");

**if** (docId == null) {

                      // 🔹 Création

**try** {

                        final cred = **await** FirebaseAuth.instance

                            .*createUserWithEmailAndPassword*(

                          email: emailController.text.*trim*(),

                          password: passwordController.text.*trim*(),

                        );

                        final uid = cred.user!.uid;

                        final dataToSave = {

                          "uid": uid,

                          "name": nameController.text.*trim*(),

                          "email": emailController.text.*trim*(),

                          "phone": phoneController.text.*trim*(),

                          "school": schoolController.text.*trim*(),

                          "speciality": specialityController.text.*trim*(),

                          "password": passwordController.text.*trim*(), // ⚠️ dev

                          "role": role,

                          "niveau": niveau,

                          "createdAt": FieldValue.*serverTimestamp*(),

                        };

**await** collection.*doc*(uid).*set*(dataToSave);

                      } **on** FirebaseAuthException **catch** (e) {

                        ScaffoldMessenger.*of*(context).*showSnackBar*(

                          SnackBar(content: Text("Erreur: ${*e*.*message*}")),

                        );

                      }

                    } **else** {

                      // 🔹 Mise à jour

                      final dataToUpdate = {

                        "name": nameController.text.*trim*(),

                        "email": emailController.text.*trim*(),

                        "phone": phoneController.text.*trim*(),

                        "school": schoolController.text.*trim*(),

                        "speciality": specialityController.text.*trim*(),

                        "role": role,

                        "niveau": niveau,

                      };

**await** collection.*doc*(docId).*update*(dataToUpdate);

                    }

                    Navigator.*pop*(context);

                  },

                ),

              ],

            );

          },

        );

      },

    );

  }

  /// 🔹 Widget textfield stylisé

  Widget *\_buildTextField*(String label, TextEditingController controller,

      {bool isPassword = false,

      TextInputType keyboardType = TextInputType.text}) {

**return** Padding(

      padding: const EdgeInsets.*symmetric*(vertical: 6),

      child: TextField(

        controller: controller,

        obscureText: isPassword,

        keyboardType: keyboardType,

        decoration: InputDecoration(

          labelText: label,

          border: OutlineInputBorder(borderRadius: BorderRadius.*circular*(10)),

        ),

      ),

    );

  }

  /// 🔹 Suppression étudiant

  Future<void> *\_deleteStudent*(String docId) **async** {

**await** FirebaseFirestore.instance.*collection*("users").*doc*(docId).*delete*();

  }

  @override

  Widget *build*(BuildContext context) {

**return** Scaffold(

      appBar: AppBar(

        title: const Text("👨‍🎓 Gestion Étudiants"),

        backgroundColor: Colors.indigo,

      ),

      body: StreamBuilder<QuerySnapshot>(

        stream: FirebaseFirestore.instance

            .*collection*("users")

            .*orderBy*("createdAt", descending: true)

            .*snapshots*(),

        builder: (context, snapshot) {

**if** (!snapshot.hasData) {

**return** const Center(child: CircularProgressIndicator());

          }

          final docs = snapshot.data!.docs;

**if** (docs.isEmpty) {

**return** const Center(

              child: Text("Aucun étudiant disponible",

                  style: TextStyle(fontSize: 16, fontWeight: FontWeight.w500)),

            );

          }

**return** ListView.*builder*(

            padding: const EdgeInsets.*all*(12),

            itemCount: docs.length,

            itemBuilder: (context, index) {

              final doc = docs[index];

              final data = doc.*data*() as Map<String, dynamic>;

**return** Card(

                elevation: 3,

                margin: const EdgeInsets.*symmetric*(vertical: 6),

                shape: RoundedRectangleBorder(

                  borderRadius: BorderRadius.*circular*(12),

                ),

                child: ListTile(

                  contentPadding:

                      const EdgeInsets.*symmetric*(horizontal: 16, vertical: 8),

                  leading: const CircleAvatar(

                    backgroundColor: Colors.indigo,

                    child: Icon(Icons.person, color: Colors.white),

                  ),

                  title: Text(

                    data['name'] ?? "Sans nom",

                    style: const TextStyle(

                        fontWeight: FontWeight.bold, fontSize: 16),

                  ),

                  subtitle: Text(

                    "${*data*['email']} • ${*data*['niveau']} • ${*data*['role']}",

                  ),

                  trailing: Row(

                    mainAxisSize: MainAxisSize.min,

                    children: [

                      IconButton(

                        icon: const Icon(Icons.edit, color: Colors.blue),

                        onPressed: () => *\_openStudentDialog*(

                          context,

                          docId: doc.id,

                          data: data,

                        ),

                      ),

                      IconButton(

                        icon: const Icon(Icons.delete, color: Colors.red),

                        onPressed: () => *\_deleteStudent*(doc.id),

                      ),

                    ],

                  ),

                ),

              );

            },

          );

        },

      ),

      floatingActionButton: FloatingActionButton.*extended*(

        icon: const Icon(Icons.add),

        label: const Text("Ajouter étudiant"),

        backgroundColor: Colors.indigo,

        onPressed: () => *\_openStudentDialog*(context),

      ),

    );

  }

}

// 📁 lib/screens/auth/login\_screen.dart

import 'package:flutter/material.dart';

import '../../services/auth\_service.dart';

import '../../app/app\_routes.dart';

class LoginScreen extends StatefulWidget {

  const LoginScreen({Key? key}) : *super*(key: key);

  @override

  State<LoginScreen> *createState*() => \_LoginScreenState();

}

class \_LoginScreenState extends State<LoginScreen>

    with TickerProviderStateMixin {

  final \_emailController = TextEditingController();

  final \_passwordController = TextEditingController();

  final AuthService \_authService = AuthService();

  bool \_isLoading = false;

  bool \_obscurePassword = true;

  bool \_rememberMe = false;

  late final AnimationController \_formController;

  late final Animation<double> \_fadeAnimation;

  late final Animation<Offset> \_slideAnimation;

  @override

  void *initState*() {

*super*.*initState*();

    \_formController = AnimationController(

      vsync: *this*,

      duration: const Duration(milliseconds: 800),

    );

    \_fadeAnimation = CurvedAnimation(

      parent: \_formController,

      curve: Curves.easeIn,

    );

    \_slideAnimation = Tween<Offset>(

      begin: const Offset(0, 0.2),

      end: Offset.zero,

    ).*animate*(CurvedAnimation(

      parent: \_formController,

      curve: Curves.easeOut,

    ));

    // 🔹 Auto-remplissage si "Se souvenir de moi" actif

    \_authService.*getSavedCredentials*().*then*((saved) {

**if** (saved != null && mounted) {

*setState*(() {

          \_emailController.text = saved['email']!;

          \_passwordController.text = saved['password']!;

          \_rememberMe = true;

        });

      }

    });

    \_formController.*forward*();

  }

  @override

  void *dispose*() {

    \_emailController.*dispose*();

    \_passwordController.*dispose*();

    \_formController.*dispose*();

*super*.*dispose*();

  }

  Future<void> *\_login*() **async** {

    final email = \_emailController.text.*trim*();

    final password = \_passwordController.text;

**if** (email.isEmpty || password.isEmpty) {

      ScaffoldMessenger.*of*(context).*showSnackBar*(

        const SnackBar(content: Text("Veuillez remplir tous les champs.")),

      );

**return**;

    }

*setState*(() => \_isLoading = true);

**try** {

      // 🔹 Authentification et récupération du rôle

      final user = **await** \_authService.*login*(

        email: email,

        password: password,

        rememberMe: \_rememberMe,

      );

**if** (\_rememberMe) {

**await** \_authService.*saveCredentials*(email, password);

      }

**if** (mounted) {

        // 🔹 Redirection selon rôle

**if** (user.role == 'admin') {

          Navigator.*pushReplacementNamed*(context, AppRoutes.adminPanel);

        } **else** {

          Navigator.*pushReplacementNamed*(context, AppRoutes.dashboard);

        }

      }

    } **catch** (e) {

**if** (mounted) {

        ScaffoldMessenger.*of*(context).*showSnackBar*(

          SnackBar(content: Text("Erreur de connexion : $*e*")),

        );

      }

    } **finally** {

**if** (mounted) *setState*(() => \_isLoading = false);

    }

  }

  void *\_showResetPasswordDialog*() {

    final emailController = TextEditingController();

*showDialog*(

      context: context,

      builder: (context) {

**return** AlertDialog(

          title: const Text("Réinitialiser le mot de passe"),

          content: TextField(

            controller: emailController,

            decoration: const InputDecoration(

              labelText: "Entrez votre email",

            ),

          ),

          actions: [

            TextButton(

              onPressed: () => Navigator.*pop*(context),

              child: const Text("Annuler"),

            ),

            TextButton(

              onPressed: () **async** {

                final email = emailController.text.*trim*();

**if** (email.isEmpty) {

                  ScaffoldMessenger.*of*(context).*showSnackBar*(

                    const SnackBar(content: Text("Veuillez entrer un email.")),

                  );

**return**;

                }

**try** {

**await** \_authService.*resetPassword*(email);

**if** (mounted) {

                    Navigator.*pop*(context); // ferme la pop-up

                    ScaffoldMessenger.*of*(context).*showSnackBar*(

                      const SnackBar(

                        content: Text("Email de réinitialisation envoyé ✅"),

                      ),

                    );

                  }

                } **catch** (e) {

**if** (mounted) {

                    ScaffoldMessenger.*of*(context).*showSnackBar*(

                      SnackBar(content: Text("Erreur : $*e*")),

                    );

                  }

                }

              },

              child: const Text("Envoyer"),

            ),

          ],

        );

      },

    );

  }

  @override

  Widget *build*(BuildContext context) {

    final maxWidth = 400.0;

**return** Scaffold(

      body: Container(

        width: double.infinity,

        decoration: const BoxDecoration(

          gradient: LinearGradient(

            colors: [Color(0xFF2563EB), Color(0xFF10B981)],

            begin: Alignment.topLeft,

            end: Alignment.bottomRight,

          ),

        ),

        child: SafeArea(

          child: Center(

            child: SingleChildScrollView(

              padding: const EdgeInsets.*all*(24),

              child: FadeTransition(

                opacity: \_fadeAnimation,

                child: SlideTransition(

                  position: \_slideAnimation,

                  child: ConstrainedBox(

                    constraints: BoxConstraints(maxWidth: maxWidth),

                    child: Container(

                      padding: const EdgeInsets.*all*(24),

                      decoration: BoxDecoration(

                        color: Colors.white,

                        borderRadius: BorderRadius.*circular*(20),

                        boxShadow: const [

                          BoxShadow(

                            color: Colors.black26,

                            blurRadius: 12,

                            offset: Offset(0, 6),

                          ),

                        ],

                      ),

                      child: Column(

                        mainAxisSize: MainAxisSize.min,

                        children: [

                          const Text(

                            "Connexion",

                            style: TextStyle(

                              fontSize: 28,

                              fontWeight: FontWeight.bold,

                              color: Colors.black87,

                            ),

                          ),

                          const SizedBox(height: 24),

                          TextField(

                            controller: \_emailController,

                            keyboardType: TextInputType.emailAddress,

                            decoration: InputDecoration(

                              labelText: "Email",

                              border: OutlineInputBorder(

                                borderRadius: BorderRadius.*circular*(12),

                              ),

                              prefixIcon: const Icon(Icons.email),

                            ),

                          ),

                          const SizedBox(height: 16),

                          TextField(

                            controller: \_passwordController,

                            obscureText: \_obscurePassword,

                            decoration: InputDecoration(

                              labelText: "Mot de passe",

                              border: OutlineInputBorder(

                                borderRadius: BorderRadius.*circular*(12),

                              ),

                              prefixIcon: const Icon(Icons.lock),

                              suffixIcon: IconButton(

                                icon: Icon(

                                  \_obscurePassword

                                      ? Icons.visibility

                                      : Icons.visibility\_off,

                                ),

                                onPressed: () => *setState*(

                                    () => \_obscurePassword = !\_obscurePassword),

                              ),

                            ),

                          ),

                          const SizedBox(height: 12),

                          Row(

                            children: [

                              Checkbox(

                                value: \_rememberMe,

                                onChanged: (v) =>

*setState*(() => \_rememberMe = v ?? false),

                              ),

                              const Text("Se souvenir de moi"),

                              const Spacer(),

                              TextButton(

                                onPressed: () => *\_showResetPasswordDialog*(),

                                child: const Text("Mot de passe oublié ?"),

                              ),

                            ],

                          ),

                          const SizedBox(height: 20),

                          SizedBox(

                            width: double.infinity,

                            height: 50,

                            child: ElevatedButton(

                              onPressed: \_isLoading ? null : \_login,

                              style: ElevatedButton.*styleFrom*(

                                shape: RoundedRectangleBorder(

                                  borderRadius: BorderRadius.*circular*(12),

                                ),

                                padding:

                                    const EdgeInsets.*symmetric*(vertical: 12),

                                backgroundColor: Colors.green,

                              ),

                              child: \_isLoading

                                  ? const CircularProgressIndicator(

                                      color: Colors.white,

                                      strokeWidth: 2,

                                    )

                                  : const Text(

                                      "Se connecter",

                                      style: TextStyle(

                                          fontSize: 16,

                                          fontWeight: FontWeight.bold),

                                    ),

                            ),

                          ),

                          const SizedBox(height: 16),

                          TextButton(

                            onPressed: () => Navigator.*pushReplacementNamed*(

                                context, AppRoutes.register),

                            child: const Text(

                              "Vous n'avez pas encore de compte ? S'inscrire",

                              style: TextStyle(

                                  fontSize: 14, color: Colors.black87),

                            ),

                          ),

                          TextButton(

                            onPressed: () => Navigator.*pushReplacementNamed*(

                                context, AppRoutes.splash),

                            child: const Text(

                              "Page Accueil",

                              style: TextStyle(

                                  fontSize: 14, color: Colors.black87),

                            ),

                          ),

                        ],

                      ),

                    ),

                  ),

                ),

              ),

            ),

          ),

        ),

      ),

    );

  }

}

// 📁 lib/screens/register\_screen.dart

import 'package:flutter/material.dart';

import 'package:provider/provider.dart';

import '../../widgets/custom\_input.dart';

import '../../widgets/custom\_button.dart';

import '../../provider/user\_provider.dart';

import '../../utils/validators.dart';

class RegisterScreen extends StatefulWidget {

  const RegisterScreen({*super*.key});

  @override

  State<RegisterScreen> *createState*() => \_RegisterScreenState();

}

class \_RegisterScreenState extends State<RegisterScreen>

    with TickerProviderStateMixin {

  final \_nameCtrl = TextEditingController();

  final \_emailCtrl = TextEditingController();

  final \_phoneCtrl = TextEditingController();

  final \_schoolCtrl = TextEditingController();

  final \_specialityCtrl = TextEditingController();

  final \_passwordCtrl = TextEditingController();

  final \_confirmCtrl = TextEditingController();

  // 🔹 Rôle forcé à "etudiant"

  String \_role = 'etudiant';

  String \_niveau = 'BTS1';

  bool \_loading = false;

  bool \_offline = false;

  String? \_nameError;

  String? \_emailError;

  String? \_phoneError;

  String? \_schoolError;

  String? \_specialityError;

  String? \_passwordError;

  String? \_confirmError;

  late final AnimationController \_formController;

  late final AnimationController \_btsController;

  late final AnimationController \_footerController;

  @override

  void *initState*() {

*super*.*initState*();

    // Validation en temps réel

    \_nameCtrl.*addListener*(\_validateField);

    \_emailCtrl.*addListener*(\_validateField);

    \_phoneCtrl.*addListener*(\_validateField);

    \_schoolCtrl.*addListener*(\_validateField);

    \_specialityCtrl.*addListener*(\_validateField);

    \_passwordCtrl.*addListener*(\_validateField);

    \_confirmCtrl.*addListener*(\_validateField);

    \_formController = AnimationController(

        vsync: *this*, duration: const Duration(milliseconds: 800));

    \_btsController = AnimationController(

        vsync: *this*, duration: const Duration(milliseconds: 800));

    \_footerController = AnimationController(

        vsync: *this*, duration: const Duration(milliseconds: 800));

    \_formController.*forward*().*then*((\_) {

      \_btsController.*forward*().*then*((\_) {

        \_footerController.*forward*();

      });

    });

  }

  @override

  void *dispose*() {

    \_nameCtrl.*dispose*();

    \_emailCtrl.*dispose*();

    \_phoneCtrl.*dispose*();

    \_schoolCtrl.*dispose*();

    \_specialityCtrl.*dispose*();

    \_passwordCtrl.*dispose*();

    \_confirmCtrl.*dispose*();

    \_formController.*dispose*();

    \_btsController.*dispose*();

    \_footerController.*dispose*();

*super*.*dispose*();

  }

  void *\_validateField*() {

*setState*(() {

      \_nameError = Validators.*fullName*(\_nameCtrl.text);

      \_emailError = Validators.*email*(\_emailCtrl.text);

      \_phoneError = Validators.*phone*(\_phoneCtrl.text);

      \_schoolError = Validators.*minLengthNoRepeat*(\_schoolCtrl.text, 4);

      \_specialityError = Validators.*minLengthNoRepeat*(\_specialityCtrl.text, 4);

      \_passwordError = Validators.*password*(\_passwordCtrl.text);

      \_confirmError = \_confirmCtrl.text != \_passwordCtrl.text

          ? 'Les mots de passe ne correspondent pas'

          : null;

    });

  }

  void *\_register*() **async** {

*\_validateField*();

**if** (\_nameError != null ||

        \_emailError != null ||

        \_phoneError != null ||

        \_schoolError != null ||

        \_specialityError != null ||

        \_passwordError != null ||

        \_confirmError != null) **return**;

*setState*(() => \_loading = true);

    final userProvider = Provider.*of*<UserProvider>(context, listen: false);

**try** {

**await** userProvider.*register*(

        name: \_nameCtrl.text.*trim*(),

        email: \_emailCtrl.text.*trim*(),

        phone: \_phoneCtrl.text.*trim*(),

        school: \_schoolCtrl.text.*trim*(),

        speciality: \_specialityCtrl.text.*trim*(),

        password: \_passwordCtrl.text,

        offline: \_offline,

        role: \_role, // 🔹 toujours "etudiant"

        niveau: \_niveau, // 🔹 BTS1 ou BTS2

      );

**if** (!mounted) **return**;

      ScaffoldMessenger.*of*(context).*showSnackBar*(

        const SnackBar(

            content: Text(

                "✅ Inscription réussie ! Veuillez vérifier votre email devopsdesigngest@gmail.com")),

      );

      Navigator.*pop*(context);

    } **catch** (e) {

**if** (!mounted) **return**;

      ScaffoldMessenger.*of*(context).*showSnackBar*(

        SnackBar(content: Text("❌ Erreur : $*e*")),

      );

    } **finally** {

**if** (mounted) *setState*(() => \_loading = false);

    }

  }

  Widget *buildAnimatedCard*(

      {required Widget child, required AnimationController controller}) {

**return** AnimatedBuilder(

      animation: controller,

      builder: (context, \_) {

**return** Opacity(

          opacity: controller.value,

          child: Transform.*translate*(

            offset: Offset(0, 50 \* (1 - controller.value)),

            child: child,

          ),

        );

      },

    );

  }

  @override

  Widget *build*(BuildContext context) {

    final width = MediaQuery.*of*(context).size.width;

    final maxWidth = 500.0;

    Widget *buildCard*({required Widget child}) {

**return** Card(

        elevation: 10,

        shape: RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(16)),

        margin: const EdgeInsets.*symmetric*(vertical: 16, horizontal: 0),

        child: Padding(

          padding: const EdgeInsets.*all*(24),

          child: ConstrainedBox(

            constraints: BoxConstraints(maxWidth: maxWidth),

            child: child,

          ),

        ),

      );

    }

**return** Scaffold(

      body: Container(

        width: double.infinity,

        decoration: const BoxDecoration(

          gradient: LinearGradient(

            colors: [Color(0xFF2563EB), Color(0xFF10B981)],

            begin: Alignment.topLeft,

            end: Alignment.bottomRight,

          ),

        ),

        child: SafeArea(

          child: Center(

            child: SingleChildScrollView(

              padding: const EdgeInsets.*symmetric*(horizontal: 20, vertical: 30),

              child: Column(

                crossAxisAlignment: CrossAxisAlignment.center,

                children: [

                  Icon(Icons.school,

                      size: width < 600 ? 100 : 150, color: Colors.white),

                  const SizedBox(height: 16),

                  Text("Réussir BTS",

                      style: TextStyle(

                          color: Colors.white,

                          fontSize: width < 600 ? 32 : 48,

                          fontWeight: FontWeight.bold),

                      textAlign: TextAlign.center),

                  const SizedBox(height: 8),

                  Text("Inscrivez-vous pour accéder aux cours et quiz",

                      style: TextStyle(

                          color: Colors.white70,

                          fontSize: width < 600 ? 16 : 20),

                      textAlign: TextAlign.center),

                  const SizedBox(height: 32),

                  // Formulaire animé

*buildAnimatedCard*(

                    controller: \_formController,

                    child: *buildCard*(

                      child: Column(

                        children: [

                          CustomInput(

                              label: "Nom complet",

                              controller: \_nameCtrl,

                              errorText: \_nameError),

                          CustomInput(

                              label: "Email",

                              controller: \_emailCtrl,

                              errorText: \_emailError),

                          CustomInput(

                              label: "Téléphone",

                              controller: \_phoneCtrl,

                              errorText: \_phoneError),

                          CustomInput(

                              label: "École",

                              controller: \_schoolCtrl,

                              errorText: \_schoolError),

                          CustomInput(

                              label: "Spécialité",

                              controller: \_specialityCtrl,

                              errorText: \_specialityError),

                          const SizedBox(height: 12),

                          // 🔹 Seul le champ niveau reste affiché

                          DropdownButtonFormField<String>(

                            value: \_niveau,

                            decoration:

                                const InputDecoration(labelText: 'Niveau'),

                            items: const [

                              DropdownMenuItem(

                                  value: 'BTS1', child: Text('BTS 1')),

                              DropdownMenuItem(

                                  value: 'BTS2', child: Text('BTS 2')),

                            ],

                            onChanged: (v) => *setState*(() => \_niveau = v!),

                          ),

                          const SizedBox(height: 12),

                          CustomInput(

                              label: "Mot de passe",

                              controller: \_passwordCtrl,

                              isPassword: true,

                              errorText: \_passwordError),

                          CustomInput(

                              label: "Confirmer mot de passe",

                              controller: \_confirmCtrl,

                              isPassword: true,

                              errorText: \_confirmError),

                          const SizedBox(height: 24),

                          // 🔹 Bouton ou loader

                          \_loading

                              ? const CircularProgressIndicator()

                              : CustomButton(

                                  text: "S'inscrire", onPressed: \_register),

                          const SizedBox(height: 24),

                          // Ligne de séparation

                          Row(

                            children: const [

                              Expanded(child: Divider(thickness: 1)),

                              Padding(

                                padding: EdgeInsets.*symmetric*(horizontal: 8),

                                child: Text("ou continuer avec"),

                              ),

                              Expanded(child: Divider(thickness: 1)),

                            ],

                          ),

                          const SizedBox(height: 16),

                          // Bouton continuer avec email

                          CustomButton(

                            text: "📧 Continuer avec Email",

                            onPressed: \_register, // utilise déjà ton flow email

                          ),

                          const SizedBox(height: 12),

                          // Bouton Google (désactivé pour l’instant)

                          CustomButton(

                            text: "🔵 Continuer avec Google",

                            onPressed: () {

                              ScaffoldMessenger.*of*(context).*showSnackBar*(

                                const SnackBar(

                                    content:

                                        Text("Google Auth pas encore activé")),

                              );

                            },

                          ),

                          const SizedBox(height: 12),

                          // Bouton Apple (désactivé pour l’instant)

                          CustomButton(

                            text: " Continuer avec Apple",

                            onPressed: () {

                              ScaffoldMessenger.*of*(context).*showSnackBar*(

                                const SnackBar(

                                    content:

                                        Text("Apple Auth pas encore activé")),

                              );

                            },

                          ),

                          const SizedBox(height: 16),

                          TextButton(

                              onPressed: () => Navigator.*pushReplacementNamed*(

                                  context, '/login'),

                              child: const Text(

                                  "Vous avez déjà un compte ? Se connecter",

                                  style: TextStyle(

                                      decoration: TextDecoration.underline))),

                          TextButton(

                            onPressed: () =>

                                Navigator.*pushReplacementNamed*(context, '/'),

                            child: const Text(

                              "Page Accueil",

                              style: TextStyle(

                                  fontSize: 14, color: Colors.black87),

                            ),

                          ),

                        ],

                      ),

                    ),

                  ),

                  // Section BTS

*buildAnimatedCard*(

                    controller: \_btsController,

                    child: *buildCard*(

                      child: Column(

                        crossAxisAlignment: CrossAxisAlignment.start,

                        children: const [

                          Text("À propos du BTS",

                              style: TextStyle(

                                  fontWeight: FontWeight.bold, fontSize: 18)),

                          SizedBox(height: 8),

                          Text(

                            "Le BTS est un diplôme national qui permet de se former rapidement "

                            "dans un domaine professionnel précis et d'acquérir des compétences recherchées.",

                            style: TextStyle(fontSize: 14),

                          ),

                        ],

                      ),

                    ),

                  ),

                  // Mentions légales

*buildAnimatedCard*(

                    controller: \_footerController,

                    child: *buildCard*(

                      child: Column(

                        crossAxisAlignment: CrossAxisAlignment.start,

                        children: const [

                          Text("Mentions légales",

                              style: TextStyle(

                                  fontWeight: FontWeight.bold, fontSize: 18)),

                          SizedBox(height: 8),

                          Text(

                            "Politique de confidentialité : nous respectons vos données et votre vie privée.\n\n"

                            "Conditions d'utilisation : en utilisant cette application, vous acceptez nos règles.\n\n"

                            "Contact : pour toute question, envoyez-nous un email à devopsdesigngest@gmail.com",

                            style: TextStyle(fontSize: 14),

                          ),

                        ],

                      ),

                    ),

                  ),

                ],

              ),

            ),

          ),

        ),

      ),

    );

  }

}

// lib/screens/revision/module\_detail\_screen.dart

import 'package:flutter/foundation.dart' show kIsWeb;

import 'package:flutter/material.dart';

import 'package:connectivity\_plus/connectivity\_plus.dart';

import 'package:dio/dio.dart';

import 'package:open\_file/open\_file.dart';

import 'package:path\_provider/path\_provider.dart';

import 'package:url\_launcher/url\_launcher.dart';

import '../../services/firestore\_service.dart';

import '../../models/video\_model.dart';

import '../../models/fiche\_model.dart';

class ModuleDetailScreen extends StatefulWidget {

  final Map<String, dynamic> moduleData;

  const ModuleDetailScreen({*super*.key, required *this*.moduleData});

  @override

  State<ModuleDetailScreen> *createState*() => \_ModuleDetailScreenState();

}

class \_ModuleDetailScreenState extends State<ModuleDetailScreen> {

  final FirestoreService \_firestore = FirestoreService();

  List<VideoItem> \_videos = [];

  List<Fiche> \_fiches = [];

  bool \_loading = true;

  bool \_online = true;

  @override

  void *initState*() {

*super*.*initState*();

*\_listenConnectivity*();

*\_loadData*();

  }

  void *\_listenConnectivity*() {

    Connectivity().onConnectivityChanged.*listen*((status) {

*setState*(() => \_online = status != ConnectivityResult.none);

    });

  }

  Future<void> *\_loadData*() **async** {

*setState*(() => \_loading = true);

**try** {

      final moduleId = widget.moduleData['id'] ?? '';

      final vids = **await** \_firestore.*getVideosForModule*(moduleId);

      final fics = **await** \_firestore.*getFichesForModule*(moduleId);

*setState*(() {

        \_videos = vids;

        \_fiches = fics;

        \_loading = false;

      });

    } **catch** (e) {

*print*("Erreur Firestore: $*e*");

*setState*(() {

        \_videos = [];

        \_fiches = [];

        \_loading = false;

      });

    }

  }

  Future<void> *\_openUrl*(String url) **async** {

**if** (kIsWeb) {

**if** (**await** *canLaunchUrl*(Uri.*parse*(url))) {

**await** *launchUrl*(Uri.*parse*(url), webOnlyWindowName: "\_blank");

      }

**return**;

    }

**await** *launchUrl*(Uri.*parse*(url), mode: LaunchMode.externalApplication);

  }

  Future<void> *\_downloadResource*(String url, String filename) **async** {

**if** (kIsWeb) **return** *\_openUrl*(url);

**try** {

      final dir = **await** *getApplicationDocumentsDirectory*();

      final savePath = "${*dir*.*path*}/$*filename*";

**await** Dio().*download*(url, savePath);

**if** (!mounted) **return**;

      ScaffoldMessenger.*of*(context).*showSnackBar*(

        SnackBar(content: Text("Téléchargé : $*filename*")),

      );

**await** OpenFile.*open*(savePath);

    } **catch** (e) {

**if** (!mounted) **return**;

      ScaffoldMessenger.*of*(context).*showSnackBar*(

        SnackBar(content: Text("Erreur téléchargement : $*e*")),

      );

    }

  }

  Widget *\_buildVideoTile*(VideoItem v, String moduleTitle) {

    final mins = ((v.duration ?? 0) / 60).*ceil*();

    final levelText = v.level != null ? " • ${*v*.*level*}" : "";

**return** ListTile(

      leading: const Icon(Icons.play\_circle\_fill, size: 36, color: Colors.blue),

      title: Text(v.title),

      subtitle: Text("$*mins* min$*levelText*"),

      trailing: Row(

        mainAxisSize: MainAxisSize.min,

        children: [

          IconButton(

            icon: const Icon(Icons.download),

            tooltip: 'Télécharger',

            onPressed: () =>

*\_downloadResource*(v.url, '${*moduleTitle*}\_${*v*.*id*}.mp4'),

          ),

          IconButton(

            icon: const Icon(Icons.open\_in\_new),

            tooltip: 'Ouvrir',

            onPressed: () => *\_openUrl*(v.url),

          ),

        ],

      ),

      onTap: () => *\_openUrl*(v.url),

    );

  }

  Widget *\_buildFicheTile*(Fiche f, String moduleTitle) {

    final levelText = f.level != null ? " • ${*f*.*level*}" : "";

**return** ListTile(

      leading: const Icon(Icons.description, size: 32, color: Colors.green),

      title: Text(f.title),

      subtitle: Text('${*f*.*pages* ?? "?"} pages$*levelText*'),

      trailing: Row(

        mainAxisSize: MainAxisSize.min,

        children: [

          IconButton(

            icon: const Icon(Icons.download),

            tooltip: 'Télécharger',

            onPressed: () =>

*\_downloadResource*(f.url, '${*moduleTitle*}\_${*f*.*id*}.pdf'),

          ),

          IconButton(

            icon: const Icon(Icons.open\_in\_new),

            tooltip: 'Ouvrir',

            onPressed: () => *\_openUrl*(f.url),

          ),

        ],

      ),

      onTap: () => *\_openUrl*(f.url),

    );

  }

  @override

  Widget *build*(BuildContext context) {

    final moduleTitle = widget.moduleData['title'] ?? 'Module';

    final moduleSubtitle = widget.moduleData['description'] ?? '';

**return** Scaffold(

      appBar: AppBar(

        title: Column(

          crossAxisAlignment: CrossAxisAlignment.start,

          children: [

            Text(moduleTitle),

**if** (moduleSubtitle.isNotEmpty)

              Text(moduleSubtitle,

                  style: const TextStyle(fontSize: 12, color: Colors.white70)),

          ],

        ),

        actions: [

          IconButton(

              icon: const Icon(Icons.refresh),

              tooltip: 'Recharger',

              onPressed: \_loadData),

        ],

      ),

      body: \_loading

          ? const Center(child: CircularProgressIndicator())

          : Column(

              children: [

**if** (!\_online)

                  Container(

                    width: double.infinity,

                    color: Colors.orange,

                    padding: const EdgeInsets.*all*(8),

                    child: const Text(

                      '⚠️ Hors ligne — lecture limitée',

                      textAlign: TextAlign.center,

                      style: TextStyle(color: Colors.white),

                    ),

                  ),

                Expanded(

                  child: DefaultTabController(

                    length: 2,

                    child: Column(

                      children: [

                        Container(

                          margin: const EdgeInsets.*all*(8),

                          decoration: BoxDecoration(

                              color: Colors.grey.shade100,

                              borderRadius: BorderRadius.*circular*(8)),

                          child: const TabBar(

                            labelColor: Colors.blue,

                            unselectedLabelColor: Colors.black54,

                            tabs: [

                              Tab(icon: Icon(Icons.videocam), text: 'Vidéos'),

                              Tab(

                                  icon: Icon(Icons.insert\_drive\_file),

                                  text: 'Fiches'),

                            ],

                          ),

                        ),

                        Expanded(

                          child: TabBarView(

                            children: [

                              \_videos.isEmpty

                                  ? const Center(child: Text("Aucune vidéo"))

                                  : ListView.*builder*(

                                      itemCount: \_videos.length,

                                      itemBuilder: (context, i) =>

*\_buildVideoTile*(

                                              \_videos[i], moduleTitle),

                                    ),

                              \_fiches.isEmpty

                                  ? const Center(child: Text("Aucune fiche"))

                                  : ListView.*builder*(

                                      itemCount: \_fiches.length,

                                      itemBuilder: (context, i) =>

*\_buildFicheTile*(

                                              \_fiches[i], moduleTitle),

                                    ),

                            ],

                          ),

                        ),

                      ],

                    ),

                  ),

                ),

              ],

            ),

    );

  }

}

// 📁 lib/screens/tabs/home\_tab.dart

import 'package:flutter/material.dart';

class HomeTab extends StatefulWidget {

  const HomeTab({*super*.key});

  @override

  State<HomeTab> *createState*() => \_HomeTabState();

}

class \_HomeTabState extends State<HomeTab> {

  final TextEditingController \_searchCtrl = TextEditingController();

  List<Map<String, String>> \_modules = [];

  List<Map<String, String>> \_filtered = [];

  bool \_loading = true;

  @override

  void *initState*() {

*super*.*initState*();

*\_loadModules*();

    \_searchCtrl.*addListener*(() => *\_search*(\_searchCtrl.text));

  }

  @override

  void *dispose*() {

    \_searchCtrl.*dispose*();

*super*.*dispose*();

  }

  Future<void> *\_loadModules*() **async** {

**await** Future.*delayed*(const Duration(milliseconds: 500)); // Simule un délai

    final modules = [

      {

        "title": "Économie Générale (Niveau 1)",

        "description":

            "Introduction aux notions fondamentales d’économie : offre, demande, marché, inflation, croissance et développement. Cours adapté aux réalités économiques du Sénégal et de l’Afrique de l’Ouest."

      },

      {

        "title": "Comptabilité Générale (Niveau 1)",

        "description":

            "Bases de la comptabilité : bilan, compte de résultat, journal, grand livre. Étude des entreprises sénégalaises comme cas pratiques."

      },

      {

        "title": "Droit des Affaires (Niveau 2)",

        "description":

            "Introduction au droit OHADA, contrats commerciaux, sociétés, procédures collectives. Mise en contexte avec des exemples de jurisprudence sénégalaise."

      },

      {

        "title": "Mathématiques Financières (Niveau 2)",

        "description":

            "Calculs d’intérêts simples et composés, actualisation, annuités, amortissements. Application aux crédits bancaires courants au Sénégal."

      },

      {

        "title": "Communication Professionnelle (Niveau 1)",

        "description":

            "Méthodes de rédaction administrative, techniques de communication orale et écrite adaptées au contexte professionnel sénégalais."

      },

      {

        "title": "Gestion des Ressources Humaines (Niveau 2)",

        "description":

            "Gestion du personnel, recrutement, formation et droit du travail sénégalais. Cas pratiques sur les PME locales."

      },

    ];

*setState*(() {

      \_modules = modules;

      \_filtered = modules;

      \_loading = false;

    });

  }

  void *\_search*(String q) {

    final query = q.*trim*().*toLowerCase*();

*setState*(() {

**if** (query.isEmpty) {

        \_filtered = \_modules;

      } **else** {

        \_filtered = \_modules

            .*where*((m) => m["title"]!.*toLowerCase*().*contains*(query))

            .*toList*();

      }

    });

  }

  void *\_goQuizHub*() => Navigator.*pushNamed*(context, '/dashboard/quiz');

  void *\_goRevisionHub*() => Navigator.*pushNamed*(context, '/dashboard/revision');

  void *\_goProgressHub*() => Navigator.*pushNamed*(context, '/dashboard/progress');

  void *\_openModule*(Map<String, String> m) {

    Navigator.*push*(

      context,

      MaterialPageRoute(

        builder: (\_) => ModuleDetail(module: m),

      ),

    );

  }

  @override

  Widget *build*(BuildContext context) {

    final theme = Theme.*of*(context);

    final isDark = theme.brightness == Brightness.dark;

**return** Scaffold(

      backgroundColor: theme.colorScheme.background,

      body: RefreshIndicator(

        onRefresh: \_loadModules,

        child: CustomScrollView(

          physics: const BouncingScrollPhysics(

              parent: AlwaysScrollableScrollPhysics()),

          slivers: [

            // ===== HEADER =====

            SliverToBoxAdapter(

              child: Container(

                height: 200,

                decoration: BoxDecoration(

                  gradient: LinearGradient(

                    colors: isDark

                        ? [

                            theme.colorScheme.primary,

                            theme.colorScheme.secondary

                          ]

                        : const [Color(0xFF2563EB), Color(0xFF10B981)],

                    begin: Alignment.topLeft,

                    end: Alignment.bottomRight,

                  ),

                ),

                padding: const EdgeInsets.*fromLTRB*(16, 40, 16, 16),

                child: Column(

                  crossAxisAlignment: CrossAxisAlignment.start,

                  children: [

                    Row(children: [

                      Icon(Icons.school,

                          color: theme.colorScheme.onPrimary, size: 36),

                      const SizedBox(width: 10),

                      Text("Réussir BTS",

                          style: theme.textTheme.headlineSmall?.*copyWith*(

                              color: theme.colorScheme.onPrimary,

                              fontWeight: FontWeight.w800)),

                    ]),

                    const SizedBox(height: 10),

                    Text("Modules BTS Sénégal – Niveaux 1 et 2.",

                        style: theme.textTheme.bodyMedium?.*copyWith*(

                            color:

                                theme.colorScheme.onPrimary.*withOpacity*(.9))),

                    const SizedBox(height: 14),

                    TextField(

                      controller: \_searchCtrl,

                      style: TextStyle(color: theme.colorScheme.onSurface),

                      decoration: InputDecoration(

                        hintText: "Rechercher un module…",

                        hintStyle: TextStyle(

                            color: theme.colorScheme.onSurface.*withOpacity*(.7)),

                        filled: true,

                        fillColor: isDark

                            ? theme.colorScheme.surfaceVariant

                            : Colors.white,

                        prefixIcon: Icon(Icons.search,

                            color: theme.colorScheme.onSurface.*withOpacity*(.7)),

                        border: OutlineInputBorder(

                          borderRadius: BorderRadius.*circular*(12),

                          borderSide: BorderSide.none,

                        ),

                        contentPadding: EdgeInsets.zero,

                      ),

                    ),

                  ],

                ),

              ),

            ),

            // ===== LIENS RAPIDES =====

            SliverToBoxAdapter(

              child: Padding(

                padding: const EdgeInsets.*all*(16),

                child: Row(children: [

                  \_QuickLink(

                      title: "Quiz",

                      icon: Icons.quiz\_outlined,

                      onTap: \_goQuizHub,

                      color: Colors.indigo),

                  const SizedBox(width: 12),

                  \_QuickLink(

                      title: "Révision",

                      icon: Icons.menu\_book\_outlined,

                      onTap: \_goRevisionHub,

                      color: Colors.green),

                  const SizedBox(width: 12),

                  \_QuickLink(

                      title: "Progression",

                      icon: Icons.bar\_chart\_rounded,

                      onTap: \_goProgressHub,

                      color: Colors.orange),

                ]),

              ),

            ),

            // ===== LISTE DES MODULES =====

            SliverToBoxAdapter(

              child: Padding(

                padding:

                    const EdgeInsets.*symmetric*(horizontal: 16, vertical: 12),

                child: Text("Modules",

                    style: theme.textTheme.titleMedium?.*copyWith*(

                      fontWeight: FontWeight.bold,

                      color: theme.colorScheme.onBackground,

                    )),

              ),

            ),

**if** (\_loading)

              const SliverToBoxAdapter(

                  child: Center(child: CircularProgressIndicator()))

**else** **if** (\_filtered.isEmpty)

              SliverToBoxAdapter(

                  child: Center(

                      child: Padding(

                padding: const EdgeInsets.*all*(24),

                child: Text("Aucun module trouvé",

                    style: theme.textTheme.bodyLarge

                        ?.*copyWith*(color: theme.colorScheme.onBackground)),

              )))

**else**

              SliverList(

                delegate: SliverChildBuilderDelegate(

                  (context, i) {

                    final m = \_filtered[i];

**return** ListTile(

                      leading: Icon(Icons.menu\_book,

                          color: theme.colorScheme.primary),

                      title: Text(m["title"]!,

                          style:

                              TextStyle(color: theme.colorScheme.onBackground)),

                      subtitle: Text(m["description"]!,

                          style: TextStyle(

                              color: theme.colorScheme.onBackground

                                  .*withOpacity*(.7))),

                      trailing: Icon(Icons.chevron\_right,

                          color: theme.colorScheme.onBackground),

                      onTap: () => *\_openModule*(m),

                    );

                  },

                  childCount: \_filtered.length,

                ),

              ),

          ],

        ),

      ),

    );

  }

}

class \_QuickLink extends StatelessWidget {

  final String title;

  final IconData icon;

  final Color color;

  final VoidCallback onTap;

  const \_QuickLink(

      {required *this*.title,

      required *this*.icon,

      required *this*.onTap,

      required *this*.color});

  @override

  Widget *build*(BuildContext context) {

    final theme = Theme.*of*(context);

    final isDark = theme.brightness == Brightness.dark;

**return** Expanded(

      child: InkWell(

        borderRadius: BorderRadius.*circular*(16),

        onTap: onTap,

        child: Ink(

          height: 80,

          decoration: BoxDecoration(

            color: isDark ? color.*withOpacity*(.2) : color.*withOpacity*(.1),

            borderRadius: BorderRadius.*circular*(16),

          ),

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              Icon(icon, color: color, size: 28),

              const SizedBox(height: 6),

              Text(title,

                  style: TextStyle(

                      color: color, fontWeight: FontWeight.w600, fontSize: 14)),

            ],

          ),

        ),

      ),

    );

  }

}

// 📄 Détail du module (texte brut)

class ModuleDetail extends StatelessWidget {

  final Map<String, String> module;

  const ModuleDetail({*super*.key, required *this*.module});

  @override

  Widget *build*(BuildContext context) {

    final theme = Theme.*of*(context);

**return** Scaffold(

      appBar: AppBar(title: Text(module["title"]!)),

      body: Padding(

        padding: const EdgeInsets.*all*(20),

        child: ListView(

          children: [

            Text(module["title"]!,

                style: theme.textTheme.headlineSmall?.*copyWith*(

                    fontWeight: FontWeight.bold,

                    color: theme.colorScheme.onBackground)),

            const SizedBox(height: 12),

            Text(module["description"]!,

                style: theme.textTheme.bodyLarge?.*copyWith*(

                    color: theme.colorScheme.onBackground, height: 1.5)),

          ],

        ),

      ),

    );

  }

}

// 📁 lib/screens/tabs/profile\_tab.dart

import 'dart:typed\_data';

import 'package:flutter/material.dart';

import 'package:provider/provider.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:firebase\_storage/firebase\_storage.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:file\_picker/file\_picker.dart';

import '../../provider/user\_provider.dart';

import '../../provider/theme\_provider.dart';

import '../../models/user\_model.dart';

import '../../services/firestore\_service.dart';

import '../../services/auth\_service.dart';

class ProfileTab extends StatefulWidget {

  const ProfileTab({*super*.key});

  @override

  State<ProfileTab> *createState*() => \_ProfileTabState();

}

class \_ProfileTabState extends State<ProfileTab> {

  bool \_isLoading = true;

  AppUser? \_user;

  String? \_error;

  String? \_photoUrl;

  @override

  void *initState*() {

*super*.*initState*();

*\_loadUser*();

  }

  Future<String?> *\_tryReadPhotoUrl*(String uid) **async** {

**try** {

      final doc =

**await** FirebaseFirestore.instance.*collection*('users').*doc*(uid).*get*();

**return** (doc.*data*() ?? const {})['photoUrl'] as String?;

    } **catch** (\_) {

**return** null;

    }

  }

  Future<void> *\_loadUser*() **async** {

**try** {

      final userProvider = Provider.*of*<UserProvider>(context, listen: false);

**if** (userProvider.user != null) {

        \_user = userProvider.user;

        \_photoUrl = (\_user?.photoUrl?.isNotEmpty ?? false)

            ? \_user!.photoUrl

            : **await** *\_tryReadPhotoUrl*(\_user!.uid);

*setState*(() => \_isLoading = false);

**return**;

      }

      final uid = FirebaseAuth.instance.currentUser?.uid;

**if** (uid == null) {

*setState*(() {

          \_error = "Utilisateur non connecté.";

          \_isLoading = false;

        });

**return**;

      }

      final fetchedUser = **await** FirestoreService().*getUser*(uid);

**if** (fetchedUser != null) {

        userProvider.*setUser*(fetchedUser);

        \_user = fetchedUser;

        \_photoUrl = (fetchedUser.photoUrl?.isNotEmpty ?? false)

            ? fetchedUser.photoUrl

            : **await** *\_tryReadPhotoUrl*(uid);

*setState*(() => \_isLoading = false);

      } **else** {

*setState*(() {

          \_error = "Utilisateur introuvable.";

          \_isLoading = false;

        });

      }

    } **catch** (e) {

*setState*(() {

        \_error = "Erreur de chargement : $*e*";

        \_isLoading = false;

      });

    }

  }

  Future<void> *\_pickAndUploadPhoto*() **async** {

    final uid = FirebaseAuth.instance.currentUser?.uid;

**if** (uid == null) **return**;

**try** {

      final result = **await** FilePicker.platform.*pickFiles*(

        type: FileType.image,

        allowMultiple: false,

        withData: true,

      );

**if** (result == null || result.files.isEmpty) **return**;

      final file = result.files.first;

      final Uint8List? bytes = file.bytes;

**if** (bytes == null) **return**;

      // Taille max 5 Mo

**if** (bytes.lengthInBytes > 5 \* 1024 \* 1024) {

        ScaffoldMessenger.*of*(context).*showSnackBar*(

            const SnackBar(content: Text("Image trop lourde (max 5 Mo).")));

**return**;

      }

// ===== Supprimer avatar =====

      Future<void> *\_removeAvatar*() **async** {

        final uid = FirebaseAuth.instance.currentUser?.uid;

**if** (uid == null) **return**;

        final confirm = **await** *showDialog*<bool>(

          context: context,

          builder: (\_) => AlertDialog(

            title: const Text("Supprimer la photo ?"),

            content: const Text("Cette action retirera ta photo de profil."),

            actions: [

              TextButton(

                  onPressed: () => Navigator.*pop*(context, false),

                  child: const Text("Annuler")),

              ElevatedButton(

                style: ElevatedButton.*styleFrom*(backgroundColor: Colors.red),

                onPressed: () => Navigator.*pop*(context, true),

                child: const Text("Supprimer"),

              ),

            ],

          ),

        );

**if** (confirm != true) **return**;

**try** {

          // Supprimer tous les formats possibles

**for** (final ext **in** ['jpg', 'jpeg', 'png']) {

**await** FirebaseStorage.instance

                .*ref*('avatars/$*uid*.$*ext*')

                .*delete*()

                .*catchError*((\_) {});

          }

**await** FirebaseFirestore.instance

              .*collection*('users')

              .*doc*(uid)

              .*update*({'photoUrl': FieldValue.*delete*()});

          final updated = \_user!.*copyWith*(photoUrl: '');

          context.*read*<UserProvider>().*setUser*(updated);

*setState*(() {

            \_user = updated;

            \_photoUrl = null;

          });

          ScaffoldMessenger.*of*(context)

              .*showSnackBar*(const SnackBar(content: Text("Photo supprimée.")));

        } **catch** (e) {

          ScaffoldMessenger.*of*(context).*showSnackBar*(

              SnackBar(content: Text("Erreur suppression photo : $*e*")));

        }

      }

      // Extension

      final ext = (file.extension ?? '').*toLowerCase*();

      final allowed = ['png', 'jpg', 'jpeg'];

**if** (!allowed.*contains*(ext)) {

        ScaffoldMessenger.*of*(context).*showSnackBar*(const SnackBar(

            content: Text("Format non supporté (png/jpg/jpeg)")));

**return**;

      }

      final ref = FirebaseStorage.instance.*ref*().*child*('avatars/$*uid*.$*ext*');

**await** ref.*putData*(bytes, SettableMetadata(contentType: 'image/$*ext*'));

      final url = **await** ref.*getDownloadURL*();

**await** FirebaseFirestore.instance

          .*collection*('users')

          .*doc*(uid)

          .*update*({'photoUrl': url, 'photoUpdatedAt': DateTime.*now*()});

      final updated = \_user!.*copyWith*(photoUrl: url);

      context.*read*<UserProvider>().*setUser*(updated);

*setState*(() {

        \_user = updated;

        \_photoUrl = url;

      });

      ScaffoldMessenger.*of*(context)

          .*showSnackBar*(const SnackBar(content: Text("Photo mise à jour ✅")));

    } **catch** (e) {

      ScaffoldMessenger.*of*(context)

          .*showSnackBar*(SnackBar(content: Text("Erreur upload : $*e*")));

    }

  }

  Future<void> *\_removeAvatar*() **async** {

    final uid = FirebaseAuth.instance.currentUser?.uid;

**if** (uid == null) **return**;

    final confirm = **await** *showDialog*<bool>(

      context: context,

      builder: (\_) => AlertDialog(

        title: const Text("Supprimer la photo ?"),

        content: const Text("Cette action retirera ta photo de profil."),

        actions: [

          TextButton(

              onPressed: () => Navigator.*pop*(context, false),

              child: const Text("Annuler")),

          ElevatedButton(

            onPressed: () => Navigator.*pop*(context, true),

            style: ElevatedButton.*styleFrom*(backgroundColor: Colors.red),

            child: const Text("Supprimer"),

          ),

        ],

      ),

    );

**if** (confirm != true) **return**;

**try** {

**await** FirebaseStorage.instance

          .*ref*('avatars/$*uid*.jpg')

          .*delete*()

          .*catchError*((\_) {});

**await** FirebaseFirestore.instance

          .*collection*('users')

          .*doc*(uid)

          .*update*({'photoUrl': FieldValue.*delete*()});

      final updated = \_user!.*copyWith*(photoUrl: '');

      context.*read*<UserProvider>().*setUser*(updated);

*setState*(() {

        \_user = updated;

        \_photoUrl = null;

      });

      ScaffoldMessenger.*of*(context)

          .*showSnackBar*(const SnackBar(content: Text("Photo supprimée.")));

    } **catch** (e) {

      ScaffoldMessenger.*of*(context).*showSnackBar*(

          SnackBar(content: Text("Erreur suppression photo : $*e*")));

    }

  }

  Future<void> *\_deleteAccount*() **async** {

    final confirmed = **await** *showDialog*<bool>(

      context: context,

      builder: (\_) => AlertDialog(

        title: const Text('Confirmer la suppression'),

        content: const Text('Supprimer ton compte définitivement ?'),

        actions: [

          TextButton(

              onPressed: () => Navigator.*pop*(context, false),

              child: const Text('Annuler')),

          ElevatedButton(

            style: ElevatedButton.*styleFrom*(backgroundColor: Colors.red),

            onPressed: () => Navigator.*pop*(context, true),

            child: const Text('Supprimer'),

          ),

        ],

      ),

    );

**if** (confirmed == true) {

**try** {

**await** AuthService().*deleteAccount*();

**if** (mounted) Navigator.*pushReplacementNamed*(context, '/');

      } **catch** (e) {

        ScaffoldMessenger.*of*(context)

            .*showSnackBar*(SnackBar(content: Text("Erreur suppression : $*e*")));

      }

    }

  }

  void *\_openEditDialog*() {

**if** (\_user == null) **return**;

*showDialog*(

      context: context,

      builder: (\_) => EditProfileDialog(user: \_user!),

    ).*then*((\_) **async** {

      final u = context.*read*<UserProvider>().user;

**if** (mounted && u != null) {

*setState*(() {

          \_user = u;

          \_photoUrl =

              (u.photoUrl?.isNotEmpty ?? false) ? u.photoUrl : \_photoUrl;

        });

      }

    });

  }

  void *\_openPasswordDialog*() {

*showDialog*(context: context, builder: (\_) => const ChangePasswordDialog());

  }

  @override

  Widget *build*(BuildContext context) {

    final themeProvider = Provider.*of*<ThemeProvider>(context);

    final isDark = themeProvider.isDark;

**if** (\_isLoading) **return** const Center(child: CircularProgressIndicator());

**if** (\_error != null)

**return** Center(

          child: Text(\_error!, style: const TextStyle(color: Colors.red)));

**if** (\_user == null)

**return** const Center(child: Text("Aucune donnée utilisateur."));

    final colors = isDark

        ? [const Color(0xFF0F172A), const Color(0xFF111827)]

        : [const Color(0xFF2563EB), const Color(0xFF10B981)];

    final bgColor = isDark ? const Color(0xFF0B1220) : const Color(0xFFF6F8FB);

**return** Scaffold(

      backgroundColor: bgColor,

      body: SafeArea(

        child: Column(

          children: [

            // ===== HEADER =====

            Container(

              width: double.infinity,

              padding: const EdgeInsets.*fromLTRB*(20, 22, 20, 26),

              decoration: BoxDecoration(

                gradient: LinearGradient(

                    colors: colors,

                    begin: Alignment.topLeft,

                    end: Alignment.bottomRight),

                borderRadius: const BorderRadius.*only*(

                    bottomLeft: Radius.*circular*(28),

                    bottomRight: Radius.*circular*(28)),

                boxShadow: [

                  BoxShadow(

                      color: Colors.black.*withOpacity*(.12),

                      blurRadius: 16,

                      offset: const Offset(0, 8))

                ],

              ),

              child: Column(

                children: [

                  Stack(

                    alignment: Alignment.bottomRight,

                    children: [

                      CircleAvatar(

                        radius: 56,

                        backgroundColor: Colors.white,

                        backgroundImage: (\_photoUrl?.isNotEmpty ?? false)

                            ? NetworkImage(\_photoUrl!)

                            : null,

                        child: (\_photoUrl?.isEmpty ?? true)

                            ? Icon(Icons.person,

                                size: 56,

                                color: isDark

                                    ? Colors.grey[700]

                                    : Colors.grey[400])

                            : null,

                      ),

                      Row(

                        mainAxisSize: MainAxisSize.min,

                        children: [

                          \_RoundIconButton(

                              icon: Icons.edit,

                              tooltip: "Changer la photo",

                              onTap: \_pickAndUploadPhoto),

                          const SizedBox(width: 8),

**if** (\_photoUrl?.isNotEmpty ?? false)

                            \_RoundIconButton(

                                icon: Icons.delete,

                                tooltip: "Supprimer la photo",

                                onTap: \_removeAvatar),

                        ],

                      ),

                    ],

                  ),

                  const SizedBox(height: 12),

                  Text(\_user!.name,

                      style: TextStyle(

                          color: Colors.white,

                          fontSize: 22,

                          fontWeight: FontWeight.w800)),

                  const SizedBox(height: 4),

                  Text(\_user!.email, style: TextStyle(color: Colors.white70)),

                  const SizedBox(height: 10),

                  InkWell(

                    onTap: () => themeProvider.*toggleTheme*(),

                    borderRadius: BorderRadius.*circular*(100),

                    child: Container(

                      padding: const EdgeInsets.*symmetric*(

                          horizontal: 14, vertical: 8),

                      decoration: BoxDecoration(

                          color: Colors.white.*withOpacity*(.15),

                          borderRadius: BorderRadius.*circular*(100)),

                      child: Row(

                        mainAxisSize: MainAxisSize.min,

                        children: [

                          Icon(isDark ? Icons.dark\_mode : Icons.light\_mode,

                              color: Colors.white),

                          const SizedBox(width: 8),

                          Text(isDark ? "Mode sombre" : "Mode clair",

                              style: const TextStyle(color: Colors.white)),

                        ],

                      ),

                    ),

                  ),

                ],

              ),

            ),

            // ===== BODY =====

            Expanded(

              child: ListView(

                padding: const EdgeInsets.*fromLTRB*(16, 18, 16, 24),

                children: [

                  \_InfoCard(

                      title: "Téléphone",

                      value: \_user?.phone ?? '',

                      icon: Icons.phone\_iphone,

                      dark: isDark,

                      onTap: \_openEditDialog),

                  \_InfoCard(

                      title: "Établissement",

                      value: \_user?.school ?? '',

                      icon: Icons.school\_outlined,

                      dark: isDark,

                      onTap: \_openEditDialog),

                  \_InfoCard(

                      title: "Spécialité",

                      value: \_user?.speciality ?? '',

                      icon: Icons.badge\_outlined,

                      dark: isDark,

                      onTap: \_openEditDialog),

                  \_InfoCard(

                      title: "Niveau",

                      value: \_user?.niveau ?? '',

                      icon: Icons.grade,

                      dark: isDark,

                      onTap: \_openEditDialog),

                  const SizedBox(height: 16),

                  ElevatedButton.*icon*(

                    onPressed: \_openEditDialog,

                    icon: const Icon(Icons.edit),

                    label: const Text("Modifier mon profil"),

                    style: ElevatedButton.*styleFrom*(

                      minimumSize: const Size.*fromHeight*(52),

                      shape: RoundedRectangleBorder(

                          borderRadius: BorderRadius.*circular*(14)),

                    ),

                  ),

                  const SizedBox(height: 12),

                  ElevatedButton.*icon*(

                    onPressed: \_openPasswordDialog,

                    icon: const Icon(Icons.lock\_reset),

                    label: const Text("Changer le mot de passe"),

                    style: ElevatedButton.*styleFrom*(

                      minimumSize: const Size.*fromHeight*(52),

                      shape: RoundedRectangleBorder(

                          borderRadius: BorderRadius.*circular*(14)),

                    ),

                  ),

                  const SizedBox(height: 12),

                  ElevatedButton.*icon*(

                    onPressed: \_deleteAccount,

                    icon: const Icon(Icons.delete),

                    label: const Text("Supprimer mon compte"),

                    style: ElevatedButton.*styleFrom*(

                      backgroundColor: Colors.red,

                      minimumSize: const Size.*fromHeight*(52),

                      shape: RoundedRectangleBorder(

                          borderRadius: BorderRadius.*circular*(14)),

                    ),

                  ),

                  const SizedBox(height: 12),

                  OutlinedButton.*icon*(

                    onPressed: () **async** {

**await** AuthService().*logout*();

**if** (context.mounted)

                        Navigator.*pushReplacementNamed*(context, '/login');

                    },

                    icon: const Icon(Icons.logout),

                    label: const Text("Déconnexion"),

                    style: OutlinedButton.*styleFrom*(

                      minimumSize: const Size.*fromHeight*(52),

                      foregroundColor: Colors.red,

                      side: const BorderSide(color: Colors.red),

                      shape: RoundedRectangleBorder(

                          borderRadius: BorderRadius.*circular*(14)),

                    ),

                  ),

                ],

              ),

            ),

          ],

        ),

      ),

    );

  }

}

// ===== Widgets =====

class \_RoundIconButton extends StatelessWidget {

  final IconData icon;

  final VoidCallback onTap;

  final String tooltip;

  const \_RoundIconButton(

      {required *this*.icon, required *this*.onTap, required *this*.tooltip});

  @override

  Widget *build*(BuildContext context) {

**return** Tooltip(

      message: tooltip,

      child: InkWell(

        onTap: onTap,

        borderRadius: BorderRadius.*circular*(100),

        child: Ink(

          width: 40,

          height: 40,

          decoration:

              const BoxDecoration(color: Colors.white, shape: BoxShape.circle),

          child: Icon(icon, size: 20),

        ),

      ),

    );

  }

}

class \_InfoCard extends StatelessWidget {

  final String title;

  final String value;

  final IconData icon;

  final bool dark;

  final VoidCallback onTap;

  const \_InfoCard(

      {required *this*.title,

      required *this*.value,

      required *this*.icon,

      required *this*.dark,

      required *this*.onTap});

  @override

  Widget *build*(BuildContext context) {

    final bg = dark ? const Color(0xFF0B1220) : Colors.white;

    final fg = dark ? Colors.white : Colors.black87;

**return** Card(

      color: bg,

      elevation: 4,

      margin: const EdgeInsets.*only*(bottom: 12),

      shape: RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(16)),

      child: ListTile(

        onTap: onTap,

        leading: CircleAvatar(

          backgroundColor: dark ? Colors.white10 : Colors.blue.*withOpacity*(.1),

          child: Icon(icon, color: fg),

        ),

        title: Text(title,

            style: TextStyle(fontWeight: FontWeight.bold, color: fg)),

        subtitle: Text((value.isEmpty ? "Non renseigné" : value),

            style: TextStyle(color: fg.*withOpacity*(.8))),

        trailing: Icon(Icons.chevron\_right, color: fg),

      ),

    );

  }

}

// ===== ChangePasswordDialog =====

class ChangePasswordDialog extends StatefulWidget {

  const ChangePasswordDialog({*super*.key});

  @override

  State<ChangePasswordDialog> *createState*() => \_ChangePasswordDialogState();

}

class \_ChangePasswordDialogState extends State<ChangePasswordDialog> {

  final oldCtrl = TextEditingController();

  final newCtrl = TextEditingController();

  final confirmCtrl = TextEditingController();

  bool loading = false;

  Future<void> *changePassword*() **async** {

    final oldPwd = oldCtrl.text.*trim*();

    final newPwd = newCtrl.text.*trim*();

    final confirmPwd = confirmCtrl.text.*trim*();

**if** (newPwd != confirmPwd) {

      ScaffoldMessenger.*of*(context).*showSnackBar*(const SnackBar(

          content: Text("Les mots de passe ne correspondent pas")));

**return**;

    }

**if** (newPwd.isEmpty || oldPwd.isEmpty) {

      ScaffoldMessenger.*of*(context).*showSnackBar*(

          const SnackBar(content: Text("Veuillez remplir tous les champs")));

**return**;

    }

**try** {

*setState*(() => loading = true);

      final user = FirebaseAuth.instance.currentUser;

**if** (user == null) **throw** "Utilisateur non connecté";

      final email = user.email;

**if** (email == null) **throw** "Email introuvable";

      final cred = EmailAuthProvider.*credential*(email: email, password: oldPwd);

**await** user.*reauthenticateWithCredential*(cred);

**await** user.*updatePassword*(newPwd);

**if** (mounted) Navigator.*pop*(context);

      ScaffoldMessenger.*of*(context).*showSnackBar*(

          const SnackBar(content: Text("Mot de passe mis à jour ✅")));

    } **catch** (e) {

      ScaffoldMessenger.*of*(context)

          .*showSnackBar*(SnackBar(content: Text("Erreur : $*e*")));

    } **finally** {

**if** (mounted) *setState*(() => loading = false);

    }

  }

  @override

  void *dispose*() {

    oldCtrl.*dispose*();

    newCtrl.*dispose*();

    confirmCtrl.*dispose*();

*super*.*dispose*();

  }

  @override

  Widget *build*(BuildContext context) {

**return** AlertDialog(

      title: const Text("Changer mon mot de passe"),

      content: Column(

        mainAxisSize: MainAxisSize.min,

        children: [

          TextField(

            controller: oldCtrl,

            decoration: const InputDecoration(labelText: "Ancien mot de passe"),

            obscureText: true,

          ),

          const SizedBox(height: 8),

          TextField(

            controller: newCtrl,

            decoration:

                const InputDecoration(labelText: "Nouveau mot de passe"),

            obscureText: true,

          ),

          const SizedBox(height: 8),

          TextField(

            controller: confirmCtrl,

            decoration: const InputDecoration(

                labelText: "Confirmer le nouveau mot de passe"),

            obscureText: true,

          ),

        ],

      ),

      actions: [

        TextButton(

            onPressed: () => Navigator.*pop*(context),

            child: const Text("Annuler")),

        ElevatedButton(

          onPressed: loading ? null : changePassword,

          child: Text(loading ? "..." : "Changer"),

        ),

      ],

    );

  }

}

// ===== EditProfileDialog =====

class EditProfileDialog extends StatefulWidget {

  final AppUser user;

  const EditProfileDialog({*super*.key, required *this*.user});

  @override

  State<EditProfileDialog> *createState*() => \_EditProfileDialogState();

}

class \_EditProfileDialogState extends State<EditProfileDialog> {

  late final TextEditingController nameCtrl;

  late final TextEditingController phoneCtrl;

  late final TextEditingController schoolCtrl;

  late final TextEditingController specialityCtrl;

  late final TextEditingController niveauCtrl;

  bool loading = false;

  @override

  void *initState*() {

*super*.*initState*();

    nameCtrl = TextEditingController(text: widget.user.name);

    phoneCtrl = TextEditingController(text: widget.user.phone);

    schoolCtrl = TextEditingController(text: widget.user.school);

    specialityCtrl = TextEditingController(text: widget.user.speciality);

    niveauCtrl = TextEditingController(text: widget.user.niveau);

  }

  @override

  void *dispose*() {

    nameCtrl.*dispose*();

    phoneCtrl.*dispose*();

    schoolCtrl.*dispose*();

    specialityCtrl.*dispose*();

    niveauCtrl.*dispose*();

*super*.*dispose*();

  }

  Future<void> *saveChanges*() **async** {

    final newName = nameCtrl.text.*trim*();

**if** (newName.isEmpty) {

      ScaffoldMessenger.*of*(context).*showSnackBar*(

          const SnackBar(content: Text("Le nom ne peut pas être vide")));

**return**;

    }

*setState*(() => loading = true);

    final updatedUser = widget.user.*copyWith*(

      name: newName,

      phone: phoneCtrl.text.*trim*(),

      school: schoolCtrl.text.*trim*(),

      speciality: specialityCtrl.text.*trim*(),

      niveau: niveauCtrl.text.*trim*(),

    );

**try** {

**await** FirestoreService().*updateUser*(updatedUser);

**if** (!mounted) **return**;

      context.*read*<UserProvider>().*setUser*(updatedUser);

      Navigator.*pop*(context);

      ScaffoldMessenger.*of*(context)

          .*showSnackBar*(const SnackBar(content: Text("Profil mis à jour ✅")));

    } **catch** (e) {

      ScaffoldMessenger.*of*(context)

          .*showSnackBar*(SnackBar(content: Text("Erreur sauvegarde : $*e*")));

    } **finally** {

**if** (mounted) *setState*(() => loading = false);

    }

  }

  Widget *\_field*(String label, TextEditingController ctrl,

      {TextInputType keyboard = TextInputType.text}) {

**return** Padding(

      padding: const EdgeInsets.*only*(bottom: 12),

      child: TextField(

        controller: ctrl,

        keyboardType: keyboard,

        decoration: InputDecoration(

          labelText: label,

          border: OutlineInputBorder(borderRadius: BorderRadius.*circular*(12)),

        ),

      ),

    );

  }

  @override

  Widget *build*(BuildContext context) {

**return** AlertDialog(

      title: const Text("Modifier mon profil"),

      content: SingleChildScrollView(

        child: Column(

          mainAxisSize: MainAxisSize.min,

          children: [

*\_field*("Nom", nameCtrl),

*\_field*("Téléphone", phoneCtrl, keyboard: TextInputType.phone),

*\_field*("Établissement", schoolCtrl),

*\_field*("Spécialité", specialityCtrl),

*\_field*("Niveau", niveauCtrl),

          ],

        ),

      ),

      actions: [

        TextButton(

            onPressed: () => Navigator.*pop*(context),

            child: const Text("Annuler")),

        ElevatedButton(

          onPressed: loading ? null : saveChanges,

          child: Text(loading ? "Enregistrement..." : "Enregistrer"),

        ),

      ],

    );

  }

}

// // lib/screens/tabs/progress\_tab.dart

// import 'dart:io';

// import 'dart:async';

// import 'package:flutter/foundation.dart' show kIsWeb;

// import 'package:flutter/material.dart';

// import 'package:flutter/services.dart';

// import 'package:path\_provider/path\_provider.dart';

// import 'package:connectivity\_plus/connectivity\_plus.dart';

// import 'package:firebase\_auth/firebase\_auth.dart';

// import '../../services/firestore\_service.dart';

// import '../../models/course\_model.dart';

// import '../../models/quiz\_result\_model.dart';

// import '../../models/fiche\_model.dart';

// import '../../models/video\_model.dart';

// class ProgressTab extends StatefulWidget {

//   const ProgressTab({super.key});

//   @override

//   State<ProgressTab> createState() => \_ProgressTabState();

// }

// class \_ProgressTabState extends State<ProgressTab> {

//   final FirestoreService \_fs = FirestoreService();

//   final \_auth = FirebaseAuth.instance;

//   bool \_loading = true;

//   bool \_online = true;

//   List<Course> \_modules = [];

//   final Map<String, List<QuizResult>> \_quizResultsByModule = {};

//   final Map<String, List<Fiche>> \_fichesByModule = {};

//   final Map<String, List<VideoItem>> \_videosByModule = {};

//   StreamSubscription<ConnectivityResult>? \_connectivitySub;

//   @override

//   void initState() {

//     super.initState();

//     \_connectivitySub = Connectivity().onConnectivityChanged.listen((status) {

//       setState(() => \_online = status != ConnectivityResult.none);

//     });

//     \_loadAll();

//   }

//   @override

//   void dispose() {

//     \_connectivitySub?.cancel();

//     super.dispose();

//   }

//   Future<void> \_loadAll({bool force = false}) async {

//     setState(() => \_loading = true);

//     final user = \_auth.currentUser;

//     if (user == null) {

//       setState(() {

//         \_modules = [];

//         \_quizResultsByModule.clear();

//         \_fichesByModule.clear();

//         \_videosByModule.clear();

//         \_loading = false;

//       });

//       return;

//     }

//     try {

//       final modules = await \_fs.getModules(forceRefresh: force);

//       \_modules = modules;

//       \_quizResultsByModule.clear();

//       \_fichesByModule.clear();

//       \_videosByModule.clear();

//       for (final m in modules) {

//         // Quiz results

//         try {

//           final results = await \_fs.getQuizResults(user.uid, m.id);

//           results.sort((a, b) => b.date.compareTo(a.date));

//           \_quizResultsByModule[m.id] = results;

//         } catch (\_) {

//           \_quizResultsByModule[m.id] = [];

//         }

//         // Fiches

//         try {

//           final fiches = await \_fs.getFichesForModule(m.id);

//           \_fichesByModule[m.id] = fiches;

//         } catch (\_) {

//           \_fichesByModule[m.id] = [];

//         }

//         // Videos

//         try {

//           final videos = await \_fs.getVideosForModule(m.id);

//           \_videosByModule[m.id] = videos;

//         } catch (\_) {

//           \_videosByModule[m.id] = [];

//         }

//       }

//     } finally {

//       setState(() => \_loading = false);

//     }

//   }

//   Map<String, dynamic> \_computeQuizStats(String moduleId) {

//     final list = \_quizResultsByModule[moduleId] ?? [];

//     if (list.isEmpty)

//       return {'best': 0, 'avg': 0.0, 'count': 0, 'scores': <int>[]};

//     final scores = list

//         .map((r) => r.total > 0 ? ((r.score / r.total) \* 100).round() : 0)

//         .toList();

//     final best = scores.isNotEmpty ? scores.reduce((a, b) => a > b ? a : b) : 0;

//     final avg = scores.isNotEmpty

//         ? scores.reduce((a, b) => a + b) / scores.length

//         : 0.0;

//     return {'best': best, 'avg': avg, 'count': scores.length, 'scores': scores};

//   }

//   String? \_badgeForScore(int best) {

//     if (best >= 90) return 'Expert';

//     if (best >= 80) return 'Très bon';

//     if (best >= 60) return 'Compétent';

//     return null;

//   }

//   Future<void> \_exportCsv(Course module) async {

//     final results = \_quizResultsByModule[module.id] ?? [];

//     if (results.isEmpty) return;

//     final sb = StringBuffer();

//     // On n'inclut que les champs existants : moduleId, moduleTitle, quizId, date, score, total, badge

//     sb.writeln('moduleId,moduleTitle,quizId,date,score,total,badge');

//     for (final r in results) {

//       sb.writeln(

//         '${r.moduleId},"${module.title}",${r.quizId},${r.date.toIso8601String()},${r.score},${r.total},${r.badge}',

//       );

//     }

//     final csv = sb.toString();

//     if (kIsWeb) {

//       await Clipboard.setData(ClipboardData(text: csv));

//       ScaffoldMessenger.of(context).showSnackBar(

//         const SnackBar(content: Text('CSV copié dans le presse-papier.')),

//       );

//     } else {

//       final dir = await getApplicationDocumentsDirectory();

//       final file = File('${dir.path}/${module.id}\_progress.csv');

//       await file.writeAsString(csv, flush: true);

//       ScaffoldMessenger.of(context).showSnackBar(

//         SnackBar(content: Text('CSV exporté dans : ${file.path}')),

//       );

//     }

//   }

//   Widget \_miniSparkline(List<int> scores) {

//     if (scores.isEmpty) return const Text('-');

//     final maxScore = scores.reduce((a, b) => a > b ? a : b);

//     final bars = scores.take(10).toList().reversed.toList();

//     return SizedBox(

//       height: 40,

//       child: Row(

//         crossAxisAlignment: CrossAxisAlignment.end,

//         children: bars.map((s) {

//           final h = (s / (maxScore > 0 ? maxScore : 100)) \* 40;

//           return Expanded(

//             child: Container(

//               margin: const EdgeInsets.symmetric(horizontal: 2),

//               height: h < 4 ? 4 : h,

//               decoration: BoxDecoration(

//                 color: s >= 80

//                     ? Colors.green

//                     : (s >= 50 ? Colors.orange : Colors.red),

//                 borderRadius: BorderRadius.circular(3),

//               ),

//             ),

//           );

//         }).toList(),

//       ),

//     );

//   }

//   @override

//   Widget build(BuildContext context) {

//     return Scaffold(

//       appBar: AppBar(

//         title: const Text('Progression'),

//         actions: [

//           IconButton(

//               onPressed: () => \_loadAll(force: true),

//               icon: const Icon(Icons.refresh)),

//         ],

//       ),

//       body: \_loading

//           ? const Center(child: CircularProgressIndicator())

//           : \_modules.isEmpty

//               ? const Center(child: Text('Aucun module disponible.'))

//               : RefreshIndicator(

//                   onRefresh: () => \_loadAll(force: true),

//                   child: ListView(

//                     padding: const EdgeInsets.all(12),

//                     children: [

//                       if (!\_online)

//                         Container(

//                           padding: const EdgeInsets.all(8),

//                           color: Colors.orange,

//                           child:

//                               const Text('⚠️ Hors ligne — données partielles'),

//                         ),

//                       ...\_modules.map((m) {

//                         final stats = \_computeQuizStats(m.id);

//                         final badge = \_badgeForScore(stats['best']);

//                         final fiches = \_fichesByModule[m.id] ?? [];

//                         final videos = \_videosByModule[m.id] ?? [];

//                         return Card(

//                           child: ListTile(

//                             leading: m.imageUrl != null

//                                 ? Image.network(m.imageUrl!,

//                                     width: 50, height: 50)

//                                 : const Icon(Icons.book, size: 40),

//                             title: Text(m.title),

//                             subtitle: Column(

//                               crossAxisAlignment: CrossAxisAlignment.start,

//                               children: [

//                                 Text(

//                                     'Quiz: ${stats['count']} • Meilleur: ${stats['best']}% • Moyenne: ${stats['avg'].toStringAsFixed(1)}%'),

//                                 \_miniSparkline(List<int>.from(stats['scores'])),

//                                 Text('📘 Fiches vues: ${fiches.length}'),

//                                 Text('🎥 Vidéos vues: ${videos.length}'),

//                               ],

//                             ),

//                             trailing:

//                                 badge != null ? Chip(label: Text(badge)) : null,

//                             onTap: () => Navigator.push(

//                               context,

//                               MaterialPageRoute(

//                                 builder: (\_) => ModuleProgressPage(

//                                   module: m,

//                                   quizResults: \_quizResultsByModule[m.id] ?? [],

//                                   fiches: fiches,

//                                   videos: videos,

//                                   onExportCsv: () => \_exportCsv(m),

//                                 ),

//                               ),

//                             ),

//                           ),

//                         );

//                       }).toList(),

//                     ],

//                   ),

//                 ),

//     );

//   }

// }

// class ModuleProgressPage extends StatelessWidget {

//   final Course module;

//   final List<QuizResult> quizResults;

//   final List<Fiche> fiches;

//   final List<VideoItem> videos;

//   final VoidCallback onExportCsv;

//   const ModuleProgressPage({

//     super.key,

//     required this.module,

//     required this.quizResults,

//     required this.fiches,

//     required this.videos,

//     required this.onExportCsv,

//   });

//   @override

//   Widget build(BuildContext context) {

//     return Scaffold(

//       appBar: AppBar(

//         title: Text('Historique — ${module.title}'),

//         actions: [

//           IconButton(onPressed: onExportCsv, icon: const Icon(Icons.download))

//         ],

//       ),

//       body: ListView(

//         padding: const EdgeInsets.all(12),

//         children: [

//           if (quizResults.isNotEmpty)

//             Column(

//               crossAxisAlignment: CrossAxisAlignment.start,

//               children: [

//                 const Text('📊 Quizzes',

//                     style:

//                         TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

//                 ...quizResults.map((r) {

//                   final percent = r.total > 0 ? (r.score / r.total \* 100) : 0;

//                   return Card(

//                     child: ListTile(

//                       title:

//                           Text('${r.quizId} • ${percent.toStringAsFixed(1)}%'),

//                       subtitle:

//                           Text('${r.date.toLocal()} • ${r.score}/${r.total}'),

//                     ),

//                   );

//                 }).toList(),

//               ],

//             ),

//           if (fiches.isNotEmpty)

//             Column(

//               crossAxisAlignment: CrossAxisAlignment.start,

//               children: [

//                 const SizedBox(height: 12),

//                 const Text('📘 Fiches consultées',

//                     style:

//                         TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

//                 ...fiches.map((f) => ListTile(title: Text(f.title))).toList(),

//               ],

//             ),

//           if (videos.isNotEmpty)

//             Column(

//               crossAxisAlignment: CrossAxisAlignment.start,

//               children: [

//                 const SizedBox(height: 12),

//                 const Text('🎥 Vidéos regardées',

//                     style:

//                         TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

//                 ...videos.map((v) => ListTile(title: Text(v.title))).toList(),

//               ],

//             ),

//         ],

//       ),

//     );

//   }

// }

import 'dart:io';

import 'dart:async';

import 'package:flutter/foundation.dart' show kIsWeb;

import 'package:flutter/material.dart';

import 'package:flutter/services.dart';

import 'package:path\_provider/path\_provider.dart';

import 'package:connectivity\_plus/connectivity\_plus.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import '../../services/firestore\_service.dart';

import '../../models/course\_model.dart';

import '../../models/quiz\_result\_model.dart';

import '../../models/fiche\_model.dart';

import '../../models/video\_model.dart';

class ProgressTab extends StatefulWidget {

  const ProgressTab({*super*.key});

  @override

  State<ProgressTab> *createState*() => \_ProgressTabState();

}

class \_ProgressTabState extends State<ProgressTab> {

  final FirestoreService \_fs = FirestoreService(); // Modules, fiches, vidéos

  final \_auth = FirebaseAuth.instance;

  final \_db = FirebaseFirestore.instance;

  bool \_loading = true;

  bool \_online = true;

  List<Course> \_modules = [];

  final Map<String, List<QuizResult>> \_quizResultsByModule = {};

  final Map<String, List<Fiche>> \_fichesByModule = {};

  final Map<String, List<VideoItem>> \_videosByModule = {};

  StreamSubscription<ConnectivityResult>? \_connectivitySub;

  @override

  void *initState*() {

*super*.*initState*();

    \_connectivitySub = Connectivity().onConnectivityChanged.*listen*((status) {

*setState*(() => \_online = status != ConnectivityResult.none);

    });

*\_loadAll*();

  }

  @override

  void *dispose*() {

    \_connectivitySub?.*cancel*();

*super*.*dispose*();

  }

  Future<void> *\_loadAll*({bool force = false}) **async** {

*setState*(() => \_loading = true);

    final user = \_auth.currentUser;

**if** (user == null) {

*setState*(() {

        \_modules = [];

        \_quizResultsByModule.*clear*();

        \_fichesByModule.*clear*();

        \_videosByModule.*clear*();

        \_loading = false;

      });

**return**;

    }

**try** {

      // Charger les modules

      final modules = **await** \_fs.*getModules*(forceRefresh: force);

      \_modules = modules;

      \_quizResultsByModule.*clear*();

      \_fichesByModule.*clear*();

      \_videosByModule.*clear*();

**for** (final m **in** modules) {

        // 🔹 Charger les résultats de quiz directement depuis Firestore

**try** {

          final snap = **await** \_db

              .*collection*("users")

              .*doc*(user.uid)

              .*collection*("quizResults")

              .*where*("moduleId", isEqualTo: m.id)

              .*get*();

          final results =

              snap.docs.*map*((d) => QuizResult.*fromFirestore*(d)).*toList*();

          results.*sort*((a, b) => b.date.*compareTo*(a.date));

          \_quizResultsByModule[m.id] = results;

        } **catch** (\_) {

          \_quizResultsByModule[m.id] = [];

        }

        // 🔹 Fiches

**try** {

          final fiches = **await** \_fs.*getFichesForModule*(m.id);

          \_fichesByModule[m.id] = fiches;

        } **catch** (\_) {

          \_fichesByModule[m.id] = [];

        }

        // 🔹 Vidéos

**try** {

          final videos = **await** \_fs.*getVideosForModule*(m.id);

          \_videosByModule[m.id] = videos;

        } **catch** (\_) {

          \_videosByModule[m.id] = [];

        }

      }

    } **finally** {

*setState*(() => \_loading = false);

    }

  }

  Map<String, dynamic> *\_computeQuizStats*(String moduleId) {

    final list = \_quizResultsByModule[moduleId] ?? [];

**if** (list.isEmpty) {

**return** {'best': 0, 'avg': 0.0, 'count': 0, 'scores': <int>[]};

    }

    final scores = list

        .*map*((r) => r.total > 0 ? ((r.score / r.total) \* 100).*round*() : 0)

        .*toList*();

    final best = scores.isNotEmpty ? scores.*reduce*((a, b) => a > b ? a : b) : 0;

    final avg = scores.isNotEmpty

        ? scores.*reduce*((a, b) => a + b) / scores.length

        : 0.0;

**return** {'best': best, 'avg': avg, 'count': scores.length, 'scores': scores};

  }

  String? *\_badgeForScore*(int best) {

**if** (best >= 90) **return** 'Expert';

**if** (best >= 80) **return** 'Très bon';

**if** (best >= 60) **return** 'Compétent';

**return** null;

  }

  Future<void> *\_exportCsv*(Course module) **async** {

    final results = \_quizResultsByModule[module.id] ?? [];

**if** (results.isEmpty) **return**;

    final sb = StringBuffer();

    sb.*writeln*('moduleId,moduleTitle,quizId,date,score,total,badge');

**for** (final r **in** results) {

      sb.*writeln*(

        '${*r*.*moduleId*},"${*module*.*title*}",${*r*.*quizId*},${*r*.*date*.*toIso8601String*()},${*r*.*score*},${*r*.*total*},${*r*.*badge*}',

      );

    }

    final csv = sb.*toString*();

**if** (kIsWeb) {

**await** Clipboard.*setData*(ClipboardData(text: csv));

      ScaffoldMessenger.*of*(context).*showSnackBar*(

        const SnackBar(content: Text('CSV copié dans le presse-papier.')),

      );

    } **else** {

      final dir = **await** *getApplicationDocumentsDirectory*();

      final file = File('${*dir*.*path*}/${*module*.*id*}\_progress.csv');

**await** file.*writeAsString*(csv, flush: true);

      ScaffoldMessenger.*of*(context).*showSnackBar*(

        SnackBar(content: Text('CSV exporté dans : ${*file*.*path*}')),

      );

    }

  }

  Widget *\_miniSparkline*(List<int> scores) {

**if** (scores.isEmpty) **return** const Text('-');

    final maxScore = scores.*reduce*((a, b) => a > b ? a : b);

    final bars = scores.*take*(10).*toList*().reversed.*toList*();

**return** SizedBox(

      height: 40,

      child: Row(

        crossAxisAlignment: CrossAxisAlignment.end,

        children: bars.*map*((s) {

          final h = (s / (maxScore > 0 ? maxScore : 100)) \* 40;

**return** Expanded(

            child: Container(

              margin: const EdgeInsets.*symmetric*(horizontal: 2),

              height: h < 4 ? 4 : h,

              decoration: BoxDecoration(

                color: s >= 80

                    ? Colors.green

                    : (s >= 50 ? Colors.orange : Colors.red),

                borderRadius: BorderRadius.*circular*(3),

              ),

            ),

          );

        }).*toList*(),

      ),

    );

  }

  @override

  Widget *build*(BuildContext context) {

**return** Scaffold(

      appBar: AppBar(

        title: const Text('Progression'),

        actions: [

          IconButton(

              onPressed: () => *\_loadAll*(force: true),

              icon: const Icon(Icons.refresh)),

        ],

      ),

      body: \_loading

          ? const Center(child: CircularProgressIndicator())

          : \_modules.isEmpty

              ? const Center(child: Text('Aucun module disponible.'))

              : RefreshIndicator(

                  onRefresh: () => *\_loadAll*(force: true),

                  child: ListView(

                    padding: const EdgeInsets.*all*(12),

                    children: [

**if** (!\_online)

                        Container(

                          padding: const EdgeInsets.*all*(8),

                          color: Colors.orange,

                          child:

                              const Text('⚠️ Hors ligne — données partielles'),

                        ),

                      ...\_modules.*map*((m) {

                        final stats = *\_computeQuizStats*(m.id);

                        final badge = *\_badgeForScore*(stats['best']);

                        final fiches = \_fichesByModule[m.id] ?? [];

                        final videos = \_videosByModule[m.id] ?? [];

**return** Card(

                          child: ListTile(

                            leading: m.imageUrl != null

                                ? Image.*network*(m.imageUrl!,

                                    width: 50, height: 50)

                                : const Icon(Icons.book, size: 40),

                            title: Text(m.title),

                            subtitle: Column(

                              crossAxisAlignment: CrossAxisAlignment.start,

                              children: [

                                Text(

                                    'Quiz: ${*stats*['count']} • Meilleur: ${*stats*['best']}% • Moyenne: ${*stats*['avg'].*toStringAsFixed*(1)}%'),

*\_miniSparkline*(List<int>.*from*(stats['scores'])),

                                Text('📘 Fiches vues: ${*fiches*.*length*}'),

                                Text('🎥 Vidéos vues: ${*videos*.*length*}'),

                              ],

                            ),

                            trailing:

                                badge != null ? Chip(label: Text(badge)) : null,

                            onTap: () => Navigator.*push*(

                              context,

                              MaterialPageRoute(

                                builder: (\_) => ModuleProgressPage(

                                  module: m,

                                  quizResults: \_quizResultsByModule[m.id] ?? [],

                                  fiches: fiches,

                                  videos: videos,

                                  onExportCsv: () => *\_exportCsv*(m),

                                ),

                              ),

                            ),

                          ),

                        );

                      }).*toList*(),

                    ],

                  ),

                ),

    );

  }

}

class ModuleProgressPage extends StatelessWidget {

  final Course module;

  final List<QuizResult> quizResults;

  final List<Fiche> fiches;

  final List<VideoItem> videos;

  final VoidCallback onExportCsv;

  const ModuleProgressPage({

*super*.key,

    required *this*.module,

    required *this*.quizResults,

    required *this*.fiches,

    required *this*.videos,

    required *this*.onExportCsv,

  });

  @override

  Widget *build*(BuildContext context) {

**return** Scaffold(

      appBar: AppBar(

        title: Text('Historique — ${*module*.*title*}'),

        actions: [

          IconButton(onPressed: onExportCsv, icon: const Icon(Icons.download))

        ],

      ),

      body: ListView(

        padding: const EdgeInsets.*all*(12),

        children: [

**if** (quizResults.isNotEmpty)

            Column(

              crossAxisAlignment: CrossAxisAlignment.start,

              children: [

                const Text('📊 Quizzes',

                    style:

                        TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

                ...quizResults.*map*((r) {

                  final percent = r.total > 0 ? (r.score / r.total \* 100) : 0;

**return** Card(

                    child: ListTile(

                      title:

                          Text('${*r*.*quizId*} • ${*percent*.*toStringAsFixed*(1)}%'),

                      subtitle:

                          Text('${*r*.*date*.*toLocal*()} • ${*r*.*score*}/${*r*.*total*}'),

                    ),

                  );

                }).*toList*(),

              ],

            ),

**if** (fiches.isNotEmpty)

            Column(

              crossAxisAlignment: CrossAxisAlignment.start,

              children: [

                const SizedBox(height: 12),

                const Text('📘 Fiches consultées',

                    style:

                        TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

                ...fiches.*map*((f) => ListTile(title: Text(f.title))).*toList*(),

              ],

            ),

**if** (videos.isNotEmpty)

            Column(

              crossAxisAlignment: CrossAxisAlignment.start,

              children: [

                const SizedBox(height: 12),

                const Text('🎥 Vidéos regardées',

                    style:

                        TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

                ...videos.*map*((v) => ListTile(title: Text(v.title))).*toList*(),

              ],

            ),

        ],

      ),

    );

  }

}

// // // lib/screens/quiz/quiz\_detail\_screen.dart

// // import 'package:flutter/material.dart';

// // import '../../models/course\_model.dart';

// // import '../../models/quiz\_model.dart';

// // import '../../models/question\_model.dart';

// // import '../../services/auth\_service.dart';

// // import '../../services/firestore\_service.dart';

// // import '../../models/quiz\_result\_model.dart';

// // class QuizDetailScreen extends StatefulWidget {

// // final Course module;

// // final QuizModel quiz;

// // const QuizDetailScreen({

// // super.key,

// // required this.module,

// // required this.quiz,

// // });

// // @override

// // State<QuizDetailScreen> createState() => \_QuizDetailScreenState();

// // }

// // class \_QuizDetailScreenState extends State<QuizDetailScreen> {

// // int \_currentIndex = 0;

// // final Map<int, int> \_answers = {};

// // DateTime? \_startTime;

// // bool \_quizFinished = false;

// // bool \_loadingQuestions = true;

// // String? \_error;

// // List<QuizResult> \_history = [];

// // final FirestoreService \_firestoreService = FirestoreService();

// // @override

// // void initState() {

// // super.initState();

// // \_initAsync();

// // }

// // Future<void> \_initAsync() async {

// // try {

// // await \_loadQuestionsSafely();

// // await \_loadHistorySafely();

// // if (mounted) {

// // setState(() {

// // \_loadingQuestions = false;

// // \_startTime = DateTime.now();

// // });

// // debugPrint(

// // "🔹 QuizDetailScreen ready: questions=${widget.quiz.questions.length}");

// // }

// // } catch (e, st) {

// // debugPrint("QuizDetailScreen init error: $e\n$st");

// // if (mounted) {

// // setState(() {

// // \_error = "Erreur chargement quiz : $e";

// // \_loadingQuestions = false;

// // });

// // }

// // }

// // }

// // Future<void> \_loadQuestionsSafely() async {

// // try {

// // if (widget.quiz.questions.isNotEmpty) {

// // debugPrint("🔹 Questions déjà présentes, normalisation");

// // for (var i = 0; i < widget.quiz.questions.length; i++) {

// // widget.quiz.questions[i] =

// // \_normalizeQuestion(widget.quiz.questions[i]);

// // }

// // return;

// // }

// // debugPrint("🔹 Chargement questions depuis Firestore...");

// // final questions = await \_firestoreService.getQuestionsForQuiz(

// // widget.module.id,

// // widget.quiz.id,

// // );

// // for (var q in questions) {

// // widget.quiz.questions.add(\_normalizeQuestion(q));

// // }

// // debugPrint("✅ ${widget.quiz.questions.length} questions chargées.");

// // } catch (e, st) {

// // debugPrint("Erreur \_loadQuestionsSafely: $e\n$st");

// // rethrow;

// // }

// // }

// // QuestionModel \_normalizeQuestion(QuestionModel q) {

// // try {

// // final List<String> options = q.options.map((o) => o.toString()).toList();

// // return QuestionModel(

// // question: q.question,

// // options: options,

// // correctIndex: q.correctIndex,

// // explanation: q.explanation,

// // );

// // } catch (e) {

// // debugPrint("normalizeQuestion failed: $e");

// // return q;

// // }

// // }

// // Future<void> \_loadHistorySafely() async {

// // try {

// // final userId = AuthService().currentUserId;

// // if (userId.isEmpty) return;

// // final results =

// // await \_firestoreService.getQuizResults(userId, widget.module.id);

// // if (!mounted) return;

// // setState(() => \_history = results);

// // debugPrint("🔹 Historique chargé : ${results.length} tentatives");

// // } catch (e) {

// // debugPrint("Erreur \_loadHistorySafely: $e");

// // }

// // }

// // void \_goNext() {

// // if (\_currentIndex < widget.quiz.questions.length - 1) {

// // setState(() => \_currentIndex++);

// // }

// // }

// // void \_goPrevious() {

// // if (\_currentIndex > 0) {

// // setState(() => \_currentIndex--);

// // }

// // }

// // int get \_currentScore {

// // int score = 0;

// // for (var entry in \_answers.entries) {

// // final qIndex = entry.key;

// // final selectedIndex = entry.value;

// // if (qIndex >= 0 &&

// // qIndex < widget.quiz.questions.length &&

// // widget.quiz.questions[qIndex].correctIndex == selectedIndex) {

// // score++;

// // }

// // }

// // return score;

// // }

// // Future<void> \_finishQuiz() async {

// // if (widget.quiz.questions.isEmpty) {

// // ScaffoldMessenger.of(context).showSnackBar(

// // const SnackBar(content: Text("Ce quiz ne contient aucune question.")),

// // );

// // return;

// // }

// // final total = widget.quiz.questions.length;

// // final bonnes = \_currentScore;

// // final fautes = total - bonnes;

// // final endTime = DateTime.now();

// // \_startTime ??= endTime;

// // final duree = endTime.difference(\_startTime!);

// // final userId = AuthService().currentUserId;

// // if (userId.isEmpty) {

// // ScaffoldMessenger.of(context).showSnackBar(

// // const SnackBar(

// // content: Text("⚠️ Connectez-vous pour enregistrer vos résultats.")),

// // );

// // return;

// // }

// // try {

// // final answersList = List<int>.generate(

// // total,

// // (i) => \_answers.containsKey(i) ? \_answers[i]! : -1,

// // );

// // // ✅ On utilise FirestoreService au lieu de QuizService

// // await \_firestoreService.saveResult(

// // etudiantId: userId,

// // moduleId: widget.module.id,

// // quizId: widget.quiz.id,

// // score: bonnes,

// // total: total,

// // bonnes: bonnes,

// // fautes: fautes,

// // duree: duree.inSeconds,

// // date: endTime,

// // answers: answersList,

// // );

// // // ✅ On utilise FirestoreService au lieu de BadgeService

// // if ((bonnes / total) >= 0.8) {

// // await \_firestoreService.unlockBadge(

// // "maitrise\_module", "Bravo ! 80%+ atteint 🎯");

// // }

// // if (bonnes == total) {

// // await \_firestoreService.unlockBadge("perfection", "Score parfait 🏆");

// // }

// // setState(() => \_quizFinished = true);

// // await \_loadHistorySafely();

// // if (!mounted) return;

// // showDialog(

// // context: context,

// // builder: (\_) => AlertDialog(

// // title: const Text("Quiz terminé"),

// // content: Text(

// // "Score : $bonnes / $total\n"

// // "✅ Bonnes : $bonnes\n"

// // "❌ Fautes : $fautes\n"

// // "⏱ Temps : ${duree.inMinutes} min",

// // ),

// // actions: [

// // TextButton(

// // onPressed: () => Navigator.pop(context),

// // child: const Text("OK"),

// // ),

// // ],

// // ),

// // );

// // } catch (e, st) {

// // debugPrint("Erreur sauvegarde quiz: $e\n$st");

// // if (!mounted) return;

// // ScaffoldMessenger.of(context).showSnackBar(

// // const SnackBar(

// // content: Text("❌ Erreur lors de l’enregistrement du quiz.")),

// // );

// // }

// // }

// // void \_restartQuiz() {

// // setState(() {

// // \_answers.clear();

// // \_currentIndex = 0;

// // \_quizFinished = false;

// // \_startTime = DateTime.now();

// // });

// // }

// // @override

// // Widget build(BuildContext context) {

// // debugPrint(

// // "🔹 Build: currentIndex=$\_currentIndex, questions=${widget.quiz.questions.length}, loading=$\_loadingQuestions");

// // if (\_loadingQuestions) {

// // return Scaffold(

// // appBar: AppBar(title: Text(widget.module.title)),

// // body: const Center(child: CircularProgressIndicator()),

// // );

// // }

// // if (\_error != null) {

// // return Scaffold(

// // appBar: AppBar(title: Text(widget.module.title)),

// // body: Center(

// // child: Padding(

// // padding: const EdgeInsets.all(16),

// // child: Column(

// // mainAxisSize: MainAxisSize.min,

// // children: [

// // Text(\_error!, style: const TextStyle(color: Colors.red)),

// // const SizedBox(height: 12),

// // ElevatedButton(

// // onPressed: () {

// // setState(() {

// // \_error = null;

// // \_loadingQuestions = true;

// // });

// // \_initAsync();

// // },

// // child: const Text("Réessayer"),

// // ),

// // ],

// // ),

// // ),

// // ),

// // );

// // }

// // if (widget.quiz.questions.isEmpty) {

// // return Scaffold(

// // appBar: AppBar(title: Text(widget.module.title)),

// // body: const Center(

// // child: Text("⚠️ Ce quiz n’a pas encore de questions.")),

// // );

// // }

// // final QuestionModel question = widget.quiz.questions[\_currentIndex];

// // final List<String> options =

// // question.options.map((o) => o.toString()).toList();

// // final int? selectedIndex = \_answers[\_currentIndex];

// // return Scaffold(

// // appBar: AppBar(title: Text(widget.module.title)),

// // body: Padding(

// // padding: const EdgeInsets.all(16.0),

// // child: Column(

// // children: [

// // // Progression

// // Row(

// // mainAxisAlignment: MainAxisAlignment.spaceBetween,

// // children: [

// // Text(

// // "Question ${\_currentIndex + 1}/${widget.quiz.questions.length}",

// // style: const TextStyle(

// // fontSize: 18, fontWeight: FontWeight.bold),

// // ),

// // if (!\_quizFinished)

// // Text("Score : $\_currentScore",

// // style: const TextStyle(

// // fontSize: 16, fontWeight: FontWeight.bold)),

// // ],

// // ),

// // const SizedBox(height: 10),

// // LinearProgressIndicator(

// // value: (\_currentIndex + 1) / widget.quiz.questions.length,

// // ),

// // const SizedBox(height: 20),

// // Text(question.question, style: const TextStyle(fontSize: 20)),

// // const SizedBox(height: 20),

// // // Options

// // Expanded(

// // child: ListView.builder(

// // itemCount: options.length,

// // itemBuilder: (context, index) {

// // final option = options[index];

// // final isSelected = selectedIndex == index;

// // final isCorrect = question.correctIndex == index;

// // Color? cardColor;

// // if (\_quizFinished) {

// // if (isCorrect) {

// // cardColor = Colors.green.shade100;

// // } else if (isSelected && !isCorrect) {

// // cardColor = Colors.red.shade100;

// // }

// // } else if (isSelected) {

// // cardColor = Colors.blue.shade100;

// // }

// // return Card(

// // color: cardColor,

// // child: ListTile(

// // title: Text(option),

// // subtitle: \_quizFinished && isCorrect

// // ? const Text("✅ Bonne réponse",

// // style: TextStyle(color: Colors.green))

// // : (\_quizFinished && isSelected && !isCorrect

// // ? Text(

// // "❌ Mauvaise réponse\nExplication : ${question.explanation ?? "Aucune explication"}",

// // style: const TextStyle(color: Colors.red),

// // )

// // : null),

// // trailing: \_quizFinished

// // ? isCorrect

// // ? const Icon(Icons.check\_circle,

// // color: Colors.green)

// // : (isSelected

// // ? const Icon(Icons.cancel, color: Colors.red)

// // : null)

// // : null,

// // onTap: \_quizFinished

// // ? null

// // : () {

// // setState(() {

// // \_answers[\_currentIndex] = index;

// // });

// // },

// // ),

// // );

// // },

// // ),

// // ),

// // // Navigation

// // Row(

// // mainAxisAlignment: MainAxisAlignment.spaceBetween,

// // children: [

// // ElevatedButton(

// // onPressed: \_currentIndex > 0 ? \_goPrevious : null,

// // child: const Text("⬅️ Précédent"),

// // ),

// // if (!\_quizFinished)

// // \_currentIndex < widget.quiz.questions.length - 1

// // ? ElevatedButton(

// // onPressed: \_goNext, child: const Text("Suivant ➡️"))

// // : ElevatedButton(

// // onPressed: \_finishQuiz,

// // child: const Text("✅ Terminer"))

// // else

// // ElevatedButton(

// // onPressed: \_restartQuiz,

// // child: const Text("🔄 Refaire le quiz")),

// // ],

// // ),

// // // Historique

// // if (\_history.isNotEmpty)

// // Expanded(

// // child: Column(

// // crossAxisAlignment: CrossAxisAlignment.start,

// // children: [

// // const SizedBox(height: 20),

// // const Text("📜 Historique des tentatives",

// // style: TextStyle(

// // fontSize: 16, fontWeight: FontWeight.bold)),

// // Expanded(

// // child: ListView.builder(

// // itemCount: \_history.length,

// // itemBuilder: (context, index) {

// // final result = \_history[index];

// // final percent = result.total > 0

// // ? (result.score / result.total \* 100)

// // : 0.0;

// // return ListTile(

// // leading: const Icon(Icons.history),

// // title:

// // Text("Score: ${result.score}/${result.total}"),

// // subtitle: Text(

// // "✅ Bonnes: ${result.score}, ❌ Fautes: ${result.total - result.score}\nDate: ${result.date.toLocal()} • ${percent.toStringAsFixed(1)}%"),

// // );

// // },

// // ),

// // ),

// // ],

// // ),

// // ),

// // ],

// // ),

// // ),

// // );

// // }

// // }

// // lib/screens/quiz/quiz\_detail\_screen.dart

// import 'package:flutter/material.dart';

// import '../../models/course\_model.dart';

// import '../../models/quiz\_model.dart';

// import '../../models/question\_model.dart';

// import '../../services/firestore\_service.dart';

// import '../../services/auth\_service.dart';

// class QuizDetailScreen extends StatefulWidget {

// final Course module;

// final QuizModel quiz;

// const QuizDetailScreen({

// super.key,

// required this.module,

// required this.quiz,

// });

// @override

// State<QuizDetailScreen> createState() => \_QuizDetailScreenState();

// }

// class \_QuizDetailScreenState extends State<QuizDetailScreen> {

// int \_currentIndex = 0;

// final Map<int, int> \_answers = {};

// bool \_quizFinished = false;

// final FirestoreService \_firestore = FirestoreService();

// QuestionModel get \_currentQuestion => widget.quiz.questions[\_currentIndex];

// void \_selectAnswer(int index) {

// if (\_quizFinished) return;

// setState(() {

// \_answers[\_currentIndex] = index;

// });

// }

// void \_nextQuestion() {

// if (\_currentIndex < widget.quiz.questions.length - 1) {

// setState(() => \_currentIndex++);

// }

// }

// void \_previousQuestion() {

// if (\_currentIndex > 0) {

// setState(() => \_currentIndex--);

// }

// }

// int get \_score {

// int score = 0;

// for (var entry in \_answers.entries) {

// final q = widget.quiz.questions[entry.key];

// if (entry.value == q.correctIndex) score++;

// }

// return score;

// }

// Future<void> \_finishQuiz() async {

// final total = widget.quiz.questions.length;

// final bonnes = \_score;

// final fautes = total - bonnes;

// setState(() => \_quizFinished = true);

// final userId = AuthService().currentUserId;

// if (userId.isNotEmpty) {

// await \_firestore.saveResult(

// etudiantId: userId,

// moduleId: widget.module.id,

// quizId: widget.quiz.id,

// score: bonnes,

// total: total,

// bonnes: bonnes,

// fautes: fautes,

// duree: 0,

// date: DateTime.now(),

// answers: \_answers.values.toList(),

// );

// }

// showDialog(

// context: context,

// builder: (\_) => AlertDialog(

// title: const Text("Quiz terminé"),

// content: Text("Score : $bonnes / $total"),

// actions: [

// TextButton(

// onPressed: () => Navigator.pop(context),

// child: const Text("OK"),

// ),

// ],

// ),

// );

// }

// @override

// Widget build(BuildContext context) {

// final q = \_currentQuestion;

// return Scaffold(

// appBar: AppBar(title: Text(widget.quiz.title)),

// body: Padding(

// padding: const EdgeInsets.all(16.0),

// child: Column(

// children: [

// // Progression

// Text(

// "Question ${\_currentIndex + 1}/${widget.quiz.questions.length}",

// style:

// const TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

// const SizedBox(height: 10),

// LinearProgressIndicator(

// value: (\_currentIndex + 1) / widget.quiz.questions.length,

// ),

// const SizedBox(height: 20),

// // Question

// Text(q.question, style: const TextStyle(fontSize: 20)),

// const SizedBox(height: 20),

// // Options

// Expanded(

// child: ListView.builder(

// itemCount: q.options.length,

// itemBuilder: (context, index) {

// final option = q.options[index];

// final selected = \_answers[\_currentIndex] == index;

// final correct = q.correctIndex == index;

// Color? color;

// if (\_quizFinished) {

// if (correct) {

// color = Colors.green.shade100;

// } else if (selected) {

// color = Colors.red.shade100;

// }

// } else if (selected) {

// color = Colors.blue.shade100;

// }

// return Card(

// color: color,

// child: ListTile(

// title: Text(option),

// subtitle: \_quizFinished && selected && !correct

// ? Text("Explication : ${q.explanation ?? '---'}")

// : null,

// onTap: () => \_selectAnswer(index),

// ),

// );

// },

// ),

// ),

// // Navigation

// Row(

// mainAxisAlignment: MainAxisAlignment.spaceBetween,

// children: [

// ElevatedButton(

// onPressed: \_currentIndex > 0 ? \_previousQuestion : null,

// child: const Text("⬅️ Précédent"),

// ),

// !\_quizFinished

// ? (\_currentIndex < widget.quiz.questions.length - 1

// ? ElevatedButton(

// onPressed: \_nextQuestion,

// child: const Text("Suivant ➡️"),

// )

// : ElevatedButton(

// onPressed: \_finishQuiz,

// child: const Text("✅ Terminer"),

// ))

// : ElevatedButton(

// onPressed: () => Navigator.pop(context),

// child: const Text("↩️ Quitter"),

// ),

// ],

// ),

// ],

// ),

// ),

// );

// }

// }

// // // // lib/screens/tabs/quiz\_tab.dart

// // // import 'package:flutter/material.dart';

// // // import '../../services/firestore\_service.dart';

// // // import '../../models/course\_model.dart';

// // // import '../../models/quiz\_model.dart';

// // // import '../quiz/quiz\_detail\_screen.dart';

// // // class QuizTab extends StatefulWidget {

// // // const QuizTab({super.key});

// // // @override

// // // State<QuizTab> createState() => \_QuizTabState();

// // // }

// // // class \_QuizTabState extends State<QuizTab> {

// // // final FirestoreService \_firestore = FirestoreService();

// // // List<Course> \_modules = [];

// // // List<Course> \_filteredModules = [];

// // // Map<String, List<QuizModel>> \_quizzesMap = {}; // moduleId -> quizzes

// // // bool \_isLoading = true;

// // // @override

// // // void initState() {

// // // super.initState();

// // // \_loadModulesAndQuizzes();

// // // }

// // // Future<void> \_loadModulesAndQuizzes() async {

// // // setState(() => \_isLoading = true);

// // // final modules = await \_firestore.getModules();

// // // final Map<String, List<QuizModel>> quizzesMap = {};

// // // for (final module in modules) {

// // // quizzesMap[module.id] = await \_firestore.getQuizzesForModule(module.id);

// // // }

// // // setState(() {

// // // \_modules = modules;

// // // \_filteredModules = modules;

// // // \_quizzesMap = quizzesMap;

// // // \_isLoading = false;

// // // });

// // // }

// // // void \_filterModules(String query) {

// // // setState(() {

// // // \_filteredModules = \_modules

// // // .where((m) => m.title.toLowerCase().contains(query.toLowerCase()))

// // // .toList();

// // // });

// // // }

// // // Future<void> \_openQuizDetail(Course module, QuizModel quiz) async {

// // // try {

// // // var questions = quiz.questions;

// // // // Si pas encore chargées → on va chercher dans Firestore

// // // if (questions.isEmpty) {

// // // questions = await \_firestore.getQuestionsForQuiz(module.id, quiz.id);

// // // }

// // // if (questions.isEmpty) {

// // // if (!mounted) return;

// // // ScaffoldMessenger.of(context).showSnackBar(

// // // const SnackBar(

// // // content: Text("⚠️ Ce quiz ne contient pas encore de questions."),

// // // ),

// // // );

// // // return;

// // // }

// // // final preparedQuiz = QuizModel(

// // // id: quiz.id,

// // // title: quiz.title,

// // // moduleId: quiz.moduleId,

// // // questions: List.from(questions),

// // // );

// // // if (!mounted) return;

// // // Navigator.push(

// // // context,

// // // MaterialPageRoute(

// // // builder: (\_) => QuizDetailScreen(

// // // module: module,

// // // quiz: preparedQuiz,

// // // ),

// // // ),

// // // );

// // // } catch (e) {

// // // if (!mounted) return;

// // // ScaffoldMessenger.of(context).showSnackBar(

// // // SnackBar(content: Text("❌ Erreur chargement quiz : $e")),

// // // );

// // // }

// // // }

// // // @override

// // // Widget build(BuildContext context) {

// // // return Scaffold(

// // // appBar: AppBar(

// // // title: const Text('📘 Quiz BTS'),

// // // bottom: PreferredSize(

// // // preferredSize: const Size.fromHeight(60),

// // // child: Padding(

// // // padding: const EdgeInsets.all(8.0),

// // // child: TextField(

// // // decoration: InputDecoration(

// // // hintText: 'Rechercher un module...',

// // // prefixIcon: const Icon(Icons.search),

// // // border: OutlineInputBorder(

// // // borderRadius: BorderRadius.circular(10),

// // // ),

// // // filled: true,

// // // fillColor: Colors.white,

// // // ),

// // // onChanged: \_filterModules,

// // // ),

// // // ),

// // // ),

// // // ),

// // // body: \_isLoading

// // // ? const Center(child: CircularProgressIndicator())

// // // : \_filteredModules.isEmpty

// // // ? const Center(child: Text('Aucun module trouvé'))

// // // : ListView.builder(

// // // padding: const EdgeInsets.all(8),

// // // itemCount: \_filteredModules.length,

// // // itemBuilder: (context, index) {

// // // final module = \_filteredModules[index];

// // // final quizzes = \_quizzesMap[module.id] ?? [];

// // // return Card(

// // // margin: const EdgeInsets.symmetric(vertical: 6),

// // // shape: RoundedRectangleBorder(

// // // borderRadius: BorderRadius.circular(10),

// // // ),

// // // child: ExpansionTile(

// // // leading: const Icon(Icons.menu\_book, size: 40),

// // // title: Text(module.title),

// // // subtitle: Text(

// // // "${quizzes.length} quiz disponible${quizzes.length > 1 ? "s" : ""}",

// // // ),

// // // children: quizzes.isEmpty

// // // ? [

// // // const Padding(

// // // padding: EdgeInsets.all(8.0),

// // // child: Text("Aucun quiz disponible."),

// // // )

// // // ]

// // // : quizzes.map((quiz) {

// // // return ListTile(

// // // leading: const Icon(Icons.quiz,

// // // color: Colors.blue),

// // // title: Text(quiz.title),

// // // subtitle: Text(

// // // "${quiz.questions.length} question${quiz.questions.length > 1 ? "s" : ""}",

// // // ),

// // // trailing: const Icon(Icons.arrow\_forward\_ios),

// // // onTap: () => \_openQuizDetail(module, quiz),

// // // );

// // // }).toList(),

// // // ),

// // // );

// // // },

// // // ),

// // // );

// // // }

// // // }

// // // // lib/screens/quiz/quiz\_tab.dart

// // // import 'package:flutter/material.dart';

// // // import '../../services/firestore\_service.dart';

// // // import '../../models/course\_model.dart';

// // // import '../../models/quiz\_model.dart';

// // // import '../quiz/quiz\_detail\_screen.dart';

// // // class QuizTab extends StatefulWidget {

// // // const QuizTab({super.key});

// // // @override

// // // State<QuizTab> createState() => \_QuizTabState();

// // // }

// // // class \_QuizTabState extends State<QuizTab> {

// // // final FirestoreService \_firestore = FirestoreService();

// // // List<Course> \_modules = [];

// // // Map<String, List<QuizModel>> \_quizzesMap = {}; // 🔹 AJOUTÉ

// // // bool \_isLoading = true;

// // // String \_searchQuery = "";

// // // @override

// // // void initState() {

// // // super.initState();

// // // \_loadModules();

// // // }

// // // Future<void> \_loadModules() async {

// // // setState(() => \_isLoading = true);

// // // final modules = await \_firestore.getModules();

// // // final Map<String, List<QuizModel>> quizzesMap = {};

// // // for (final module in modules) {

// // // quizzesMap[module.id] = await \_firestore.getQuizzesForModule(module.id);

// // // }

// // // setState(() {

// // // \_modules = modules;

// // // \_quizzesMap = quizzesMap;

// // // \_isLoading = false;

// // // });

// // // }

// // // Future<void> \_openQuizDetail(Course module, QuizModel quiz) async {

// // // try {

// // // var questions = quiz.questions;

// // // // Si vide → récupérer depuis Firestore

// // // if (questions.isEmpty) {

// // // questions = await \_firestore.getQuestionsForQuiz(module.id, quiz.id);

// // // }

// // // // 🔹 Si toujours vide → afficher message et STOP

// // // if (questions.isEmpty) {

// // // if (!mounted) return;

// // // ScaffoldMessenger.of(context).showSnackBar(

// // // const SnackBar(

// // // content: Text("⚠️ Ce quiz ne contient pas encore de questions.")),

// // // );

// // // return; // ⛔ pas de navigation

// // // }

// // // final preparedQuiz = QuizModel(

// // // id: quiz.id,

// // // title: quiz.title,

// // // moduleId: quiz.moduleId,

// // // questions: List.from(questions),

// // // );

// // // if (!mounted) return;

// // // Navigator.push(

// // // context,

// // // MaterialPageRoute(

// // // builder: (\_) => QuizDetailScreen(

// // // module: module,

// // // quiz: preparedQuiz,

// // // ),

// // // ),

// // // );

// // // } catch (e) {

// // // if (!mounted) return;

// // // ScaffoldMessenger.of(context).showSnackBar(

// // // SnackBar(content: Text("❌ Erreur chargement quiz : $e")),

// // // );

// // // }

// // // }

// // // @override

// // // Widget build(BuildContext context) {

// // // final filteredModules = \_modules

// // // .where(

// // // (m) => m.title.toLowerCase().contains(\_searchQuery.toLowerCase()))

// // // .toList();

// // // return Scaffold(

// // // appBar: AppBar(

// // // title: const Text('📘 Quiz BTS'),

// // // bottom: PreferredSize(

// // // preferredSize: const Size.fromHeight(60),

// // // child: Padding(

// // // padding: const EdgeInsets.all(8.0),

// // // child: TextField(

// // // decoration: InputDecoration(

// // // hintText: 'Rechercher un module...',

// // // prefixIcon: const Icon(Icons.search),

// // // border: OutlineInputBorder(

// // // borderRadius: BorderRadius.circular(10),

// // // ),

// // // filled: true,

// // // fillColor: Colors.white,

// // // ),

// // // onChanged: (value) {

// // // setState(() => \_searchQuery = value);

// // // },

// // // ),

// // // ),

// // // ),

// // // ),

// // // body: \_isLoading

// // // ? const Center(child: CircularProgressIndicator())

// // // : filteredModules.isEmpty

// // // ? const Center(child: Text('Aucun module trouvé'))

// // // : ListView.builder(

// // // padding: const EdgeInsets.all(8),

// // // itemCount: filteredModules.length,

// // // itemBuilder: (context, index) {

// // // final module = filteredModules[index];

// // // final quizzes = \_quizzesMap[module.id] ?? [];

// // // return Card(

// // // child: ExpansionTile(

// // // leading: const Icon(Icons.menu\_book, size: 40),

// // // title: Text(module.title),

// // // subtitle: Text("${quizzes.length} quiz"),

// // // children: quizzes.isEmpty

// // // ? [

// // // const Padding(

// // // padding: EdgeInsets.all(8.0),

// // // child: Text("Aucun quiz disponible."),

// // // )

// // // ]

// // // : quizzes.map((quiz) {

// // // return ListTile(

// // // leading: const Icon(Icons.quiz,

// // // color: Colors.blue),

// // // title: Text(quiz.title),

// // // subtitle: Text(

// // // "${quiz.questions.length} questions"),

// // // onTap: () => \_openQuizDetail(module, quiz),

// // // );

// // // }).toList(),

// // // ),

// // // );

// // // },

// // // ),

// // // );

// // // }

// // // }

// // import 'package:flutter/material.dart';

// // import 'package:cloud\_firestore/cloud\_firestore.dart';

// // import 'package:firebase\_auth/firebase\_auth.dart';

// // import '../../models/course\_model.dart';

// // import '../../models/quiz\_model.dart';

// // import '../../models/question\_model.dart';

// // import '../../models/quiz\_result\_model.dart';

// // /// ==========================

// // /// 👇 Onglet Quiz (liste modules + quiz)

// // /// ==========================

// // class QuizTab extends StatefulWidget {

// // const QuizTab({super.key});

// // @override

// // State<QuizTab> createState() => \_QuizTabState();

// // }

// // class \_QuizTabState extends State<QuizTab> {

// // final \_quizService = \_QuizService();

// // bool \_loading = true;

// // List<Course> \_modules = [];

// // Map<String, List<QuizModel>> \_quizzes = {};

// // String \_search = "";

// // @override

// // void initState() {

// // super.initState();

// // \_load();

// // }

// // Future<void> \_load() async {

// // setState(() => \_loading = true);

// // final modules = await \_quizService.getModules();

// // final map = <String, List<QuizModel>>{};

// // for (var m in modules) {

// // map[m.id] = await \_quizService.getQuizzesForModule(m.id);

// // }

// // setState(() {

// // \_modules = modules;

// // \_quizzes = map;

// // \_loading = false;

// // });

// // }

// // void \_open(Course module, QuizModel quiz) {

// // if (quiz.questions.isEmpty) {

// // ScaffoldMessenger.of(context).showSnackBar(

// // const SnackBar(

// // content: Text("⚠️ Ce quiz n’a pas encore de questions.")),

// // );

// // return;

// // }

// // Navigator.push(

// // context,

// // MaterialPageRoute(

// // builder: (\_) => QuizDetailScreen(

// // module: module,

// // quiz: quiz,

// // service: \_quizService,

// // ),

// // ),

// // );

// // }

// // @override

// // Widget build(BuildContext context) {

// // final filtered = \_modules

// // .where((m) => m.title.toLowerCase().contains(\_search.toLowerCase()))

// // .toList();

// // return Scaffold(

// // appBar: AppBar(

// // title: const Text("📘 Quiz BTS"),

// // bottom: PreferredSize(

// // preferredSize: const Size.fromHeight(50),

// // child: Padding(

// // padding: const EdgeInsets.all(8.0),

// // child: TextField(

// // onChanged: (v) => setState(() => \_search = v),

// // decoration: InputDecoration(

// // hintText: "Rechercher un module...",

// // prefixIcon: const Icon(Icons.search),

// // border: OutlineInputBorder(

// // borderRadius: BorderRadius.circular(10),

// // ),

// // filled: true,

// // fillColor: Colors.white,

// // ),

// // ),

// // ),

// // ),

// // ),

// // body: \_loading

// // ? const Center(child: CircularProgressIndicator())

// // : filtered.isEmpty

// // ? const Center(child: Text("Aucun module trouvé"))

// // : ListView.builder(

// // itemCount: filtered.length,

// // itemBuilder: (\_, i) {

// // final module = filtered[i];

// // final quizzes = \_quizzes[module.id] ?? [];

// // return Card(

// // child: ExpansionTile(

// // title: Text(module.title),

// // subtitle: Text("${quizzes.length} quiz"),

// // leading: const Icon(Icons.menu\_book, size: 40),

// // children: quizzes.isEmpty

// // ? [const ListTile(title: Text("Aucun quiz."))]

// // : quizzes.map((q) {

// // return ListTile(

// // leading: const Icon(Icons.quiz,

// // color: Colors.blue),

// // title: Text(q.title),

// // subtitle:

// // Text("${q.questions.length} questions"),

// // onTap: () => \_open(module, q),

// // );

// // }).toList(),

// // ),

// // );

// // },

// // ),

// // );

// // }

// // }

// // /// ==========================

// // /// 👇 Détail du quiz

// // /// ==========================

// // class QuizDetailScreen extends StatefulWidget {

// // final Course module;

// // final QuizModel quiz;

// // final \_QuizService service;

// // const QuizDetailScreen({

// // super.key,

// // required this.module,

// // required this.quiz,

// // required this.service,

// // });

// // @override

// // State<QuizDetailScreen> createState() => \_QuizDetailScreenState();

// // }

// // class \_QuizDetailScreenState extends State<QuizDetailScreen> {

// // int \_index = 0;

// // final Map<int, int> \_answers = {};

// // bool \_finished = false;

// // QuestionModel get q => widget.quiz.questions[\_index];

// // void \_select(int i) {

// // if (\_finished) return;

// // setState(() => \_answers[\_index] = i);

// // }

// // void \_next() {

// // if (\_index < widget.quiz.questions.length - 1) {

// // setState(() => \_index++);

// // }

// // }

// // void \_prev() {

// // if (\_index > 0) setState(() => \_index--);

// // }

// // int get \_score {

// // int s = 0;

// // for (var e in \_answers.entries) {

// // if (widget.quiz.questions[e.key].correctIndex == e.value) s++;

// // }

// // return s;

// // }

// // Future<void> \_finish() async {

// // final total = widget.quiz.questions.length;

// // final bonnes = \_score;

// // final fautes = total - bonnes;

// // setState(() => \_finished = true);

// // await widget.service.saveResult(

// // moduleId: widget.module.id,

// // quizId: widget.quiz.id,

// // score: bonnes,

// // total: total,

// // bonnes: bonnes,

// // fautes: fautes,

// // duree: 0,

// // date: DateTime.now(),

// // answers: \_answers.values.toList(),

// // );

// // showDialog(

// // context: context,

// // builder: (\_) => AlertDialog(

// // title: const Text("Quiz terminé"),

// // content: Text("✅ Score : $bonnes / $total"),

// // actions: [

// // TextButton(

// // onPressed: () => Navigator.pop(context),

// // child: const Text("OK"),

// // )

// // ],

// // ),

// // );

// // }

// // @override

// // Widget build(BuildContext context) {

// // return Scaffold(

// // appBar: AppBar(title: Text(widget.quiz.title)),

// // body: Padding(

// // padding: const EdgeInsets.all(16),

// // child: Column(

// // children: [

// // Text("Question ${\_index + 1}/${widget.quiz.questions.length}",

// // style:

// // const TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),

// // const SizedBox(height: 10),

// // LinearProgressIndicator(

// // value: (\_index + 1) / widget.quiz.questions.length,

// // ),

// // const SizedBox(height: 20),

// // Text(q.question, style: const TextStyle(fontSize: 20)),

// // const SizedBox(height: 20),

// // Expanded(

// // child: ListView.builder(

// // itemCount: q.options.length,

// // itemBuilder: (\_, i) {

// // final selected = \_answers[\_index] == i;

// // final correct = q.correctIndex == i;

// // Color? color;

// // if (\_finished) {

// // if (correct) color = Colors.green.shade100;

// // if (selected && !correct) color = Colors.red.shade100;

// // } else if (selected) {

// // color = Colors.blue.shade100;

// // }

// // return Card(

// // color: color,

// // child: ListTile(

// // title: Text(q.options[i]),

// // subtitle: \_finished && selected && !correct

// // ? Text("Explication : ${q.explanation ?? ''}")

// // : null,

// // onTap: () => \_select(i),

// // ),

// // );

// // },

// // ),

// // ),

// // Row(

// // mainAxisAlignment: MainAxisAlignment.spaceBetween,

// // children: [

// // ElevatedButton(

// // onPressed: \_index > 0 ? \_prev : null,

// // child: const Text("⬅️ Précédent"),

// // ),

// // !\_finished

// // ? (\_index < widget.quiz.questions.length - 1

// // ? ElevatedButton(

// // onPressed: \_next,

// // child: const Text("Suivant ➡️"),

// // )

// // : ElevatedButton(

// // onPressed: \_finish,

// // child: const Text("✅ Terminer"),

// // ))

// // : ElevatedButton(

// // onPressed: () => Navigator.pop(context),

// // child: const Text("↩️ Quitter"),

// // )

// // ],

// // )

// // ],

// // ),

// // ),

// // );

// // }

// // }

// // /// ==========================

// // /// 👇 Service Quiz

// // /// ==========================

// // class \_QuizService {

// // final \_db = FirebaseFirestore.instance;

// // final \_auth = FirebaseAuth.instance;

// // Future<List<Course>> getModules() async {

// // final snap = await \_db.collection("modules").get();

// // return snap.docs.map((d) => Course.fromMap(d.data(), id: d.id)).toList();

// // }

// // Future<List<QuizModel>> getQuizzesForModule(String moduleId) async {

// // final snap = await \_db

// // .collection("modules")

// // .doc(moduleId)

// // .collection("quizzes")

// // .get();

// // return snap.docs.map((doc) {

// // final data = doc.data();

// // final questions = (data['questions'] as List<dynamic>? ?? [])

// // .map((q) => QuestionModel.fromMap(Map<String, dynamic>.from(q)))

// // .toList();

// // return QuizModel(

// // id: doc.id,

// // moduleId: moduleId,

// // title: (data['title'] ?? "Quiz").toString(),

// // questions: questions,

// // );

// // }).toList();

// // }

// // Future<void> saveResult({

// // required String moduleId,

// // required String quizId,

// // required int score,

// // required int total,

// // required int bonnes,

// // required int fautes,

// // required int duree,

// // required DateTime date,

// // List<int>? answers,

// // }) async {

// // final uid = \_auth.currentUser?.uid ?? "anonymous";

// // final percent = total > 0 ? ((score / total) \* 100).round() : 0;

// // String badge = "bronze";

// // if (percent >= 90)

// // badge = "gold";

// // else if (percent >= 75) badge = "silver";

// // final result = QuizResult(

// // id: "",

// // moduleId: moduleId,

// // quizId: quizId,

// // score: score,

// // total: total,

// // bonnes: bonnes,

// // fautes: fautes,

// // duree: duree,

// // percent: percent,

// // badge: badge,

// // date: date,

// // answers: answers ?? [],

// // ).toMap();

// // await \_db

// // .collection("users")

// // .doc(uid)

// // .collection("quizResults")

// // .add(result);

// // // 🔹 Débloquer badge

// // final badgeRef =

// // \_db.collection("users").doc(uid).collection("badges").doc(badge);

// // if (!(await badgeRef.get()).exists) {

// // await badgeRef.set({

// // "id": badge,

// // "description": "Badge $badge débloqué",

// // "dateUnlocked": DateTime.now().toIso8601String(),

// // });

// // }

// // }

// // }

// // //lib/tabs/quiz\_tab.dart (ou remplace ton fichier existant)

// // //Version minimale à 10% : pas de Firestore, pas d'Auth — juste de l'affichage.

// // import 'package:flutter/material.dart';

// // class QuizTab extends StatelessWidget {

// // const QuizTab({super.key});

// // static const \_modules = <\_ModuleSample>[

// // \_ModuleSample(

// // title: 'Mathématiques BTS',

// // quizzes: [

// // 'Quiz 1 - Fondamentaux',

// // 'Quiz 2 - Dérivées',

// // 'Quiz 3 - Algèbre',

// // ],

// // ),

// // \_ModuleSample(

// // title: 'Comptabilité BTS',

// // quizzes: [

// // 'Quiz 1 - Fondamentaux',

// // 'Quiz 2 - Amortissements',

// // 'Quiz 3 - Stocks',

// // ],

// // ),

// // ];

// // @override

// // Widget build(BuildContext context) {

// // return Scaffold(

// // appBar: AppBar(title: const Text('📘 Quiz (mini)')),

// // body: ListView(

// // padding: const EdgeInsets.all(12),

// // children: [

// // const \_Header(),

// // const SizedBox(height: 12),

// // ...\_modules.map((m) => Card(

// // elevation: 2,

// // child: ExpansionTile(

// // leading: const Icon(Icons.menu\_book, size: 36),

// // title: Text(m.title, style: const TextStyle(fontSize: 16)),

// // subtitle: Text('${m.quizzes.length} quiz'),

// // children: m.quizzes.map((q) {

// // return ListTile(

// // leading: const Icon(Icons.quiz, color: Colors.blue),

// // title: Text(q),

// // trailing: const Icon(Icons.chevron\_right),

// // onTap: () {

// // Navigator.of(context).push(

// // MaterialPageRoute(

// // builder: (\_) => \_QuizWelcomeScreen(title: q),

// // ),

// // );

// // },

// // );

// // }).toList(),

// // ),

// // )),

// // const SizedBox(height: 24),

// // ],

// // ),

// // );

// // }

// // }

// // class \_Header extends StatelessWidget {

// // const \_Header();

// // @override

// // Widget build(BuildContext context) {

// // return Container(

// // padding: const EdgeInsets.all(14),

// // decoration: BoxDecoration(

// // color: Colors.blue.shade50,

// // borderRadius: BorderRadius.circular(12),

// // ),

// // child: const Row(

// // children: [

// // Icon(Icons.school, size: 28),

// // SizedBox(width: 10),

// // Expanded(

// // child: Text(

// // '👋 Bienvenue !\nChoisis un module puis un quiz pour commencer.',

// // style: TextStyle(fontSize: 16, fontWeight: FontWeight.w600),

// // ),

// // ),

// // ],

// // ),

// // );

// // }

// // }

// // class \_QuizWelcomeScreen extends StatelessWidget {

// // final String title;

// // const \_QuizWelcomeScreen({required this.title});

// // @override

// // Widget build(BuildContext context) {

// // return Scaffold(

// // appBar: AppBar(title: Text(title)),

// // body: Center(

// // child: Text(

// // '✅ Bienvenue — $title\n(étape 1: affichage OK)',

// // textAlign: TextAlign.center,

// // style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),

// // ),

// // ),

// // );

// // }

// // }

// // class \_ModuleSample {

// // final String title;

// // final List<String> quizzes;

// // const \_ModuleSample({required this.title, required this.quizzes});

// // }

// // import 'package:flutter/material.dart';

// // class QuizTab extends StatelessWidget {

// // const QuizTab({super.key});

// // static const \_modules = <\_ModuleSample>[

// // \_ModuleSample(

// // title: 'Mathématiques BTS',

// // quizzes: [

// // 'Quiz 1 - Fondamentaux',

// // ],

// // ),

// // \_ModuleSample(

// // title: 'Comptabilité BTS',

// // quizzes: [

// // 'Quiz 1 - Fondamentaux',

// // ],

// // ),

// // ];

// // @override

// // Widget build(BuildContext context) {

// // return Scaffold(

// // appBar: AppBar(title: const Text('📘 Quiz (20%)')),

// // body: ListView(

// // padding: const EdgeInsets.all(12),

// // children: [

// // const \_Header(),

// // const SizedBox(height: 12),

// // ...\_modules.map((m) => Card(

// // elevation: 2,

// // child: ExpansionTile(

// // leading: const Icon(Icons.menu\_book, size: 36),

// // title: Text(m.title, style: const TextStyle(fontSize: 16)),

// // subtitle: Text('${m.quizzes.length} quiz'),

// // children: m.quizzes.map((q) {

// // return ListTile(

// // leading: const Icon(Icons.quiz, color: Colors.blue),

// // title: Text(q),

// // trailing: const Icon(Icons.chevron\_right),

// // onTap: () {

// // Navigator.of(context).push(

// // MaterialPageRoute(

// // builder: (\_) => QuizPlayScreen(title: q),

// // ),

// // );

// // },

// // );

// // }).toList(),

// // ),

// // )),

// // const SizedBox(height: 24),

// // ],

// // ),

// // );

// // }

// // }

// // class \_Header extends StatelessWidget {

// // const \_Header();

// // @override

// // Widget build(BuildContext context) {

// // return Container(

// // padding: const EdgeInsets.all(14),

// // decoration: BoxDecoration(

// // color: Colors.blue.shade50,

// // borderRadius: BorderRadius.circular(12),

// // ),

// // child: const Row(

// // children: [

// // Icon(Icons.school, size: 28),

// // SizedBox(width: 10),

// // Expanded(

// // child: Text(

// // '👋 Bienvenue !\nÉtape 20% : tu peux répondre à une première question.',

// // style: TextStyle(fontSize: 16, fontWeight: FontWeight.w600),

// // ),

// // ),

// // ],

// // ),

// // );

// // }

// // }

// // /// Écran quiz minimal

// // class QuizPlayScreen extends StatefulWidget {

// // final String title;

// // const QuizPlayScreen({required this.title, super.key});

// // @override

// // State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

// // }

// // class \_QuizPlayScreenState extends State<QuizPlayScreen> {

// // int? selectedIndex;

// // bool finished = false;

// // final String question = "Combien font 2 + 2 ?";

// // final List<String> options = ["3", "4"];

// // final int correctIndex = 1; // la bonne réponse est "4"

// // void \_select(int i) {

// // if (finished) return;

// // setState(() => selectedIndex = i);

// // }

// // void \_finish() {

// // setState(() => finished = true);

// // }

// // @override

// // Widget build(BuildContext context) {

// // return Scaffold(

// // appBar: AppBar(title: Text(widget.title)),

// // body: Padding(

// // padding: const EdgeInsets.all(16),

// // child: Column(

// // children: [

// // Text(question, style: const TextStyle(fontSize: 20)),

// // const SizedBox(height: 20),

// // ...List.generate(options.length, (i) {

// // final isSelected = selectedIndex == i;

// // final isCorrect = i == correctIndex;

// // Color? color;

// // if (finished) {

// // if (isCorrect) color = Colors.green.shade200;

// // if (isSelected && !isCorrect) color = Colors.red.shade200;

// // } else if (isSelected) {

// // color = Colors.blue.shade200;

// // }

// // return Card(

// // color: color,

// // child: ListTile(

// // title: Text(options[i]),

// // onTap: () => \_select(i),

// // ),

// // );

// // }),

// // const Spacer(),

// // ElevatedButton(

// // onPressed: finished ? () => Navigator.pop(context) : \_finish,

// // child: Text(finished ? "↩️ Quitter" : "✅ Terminer"),

// // ),

// // ],

// // ),

// // ),

// // );

// // }

// // }

// // class \_ModuleSample {

// // final String title;

// // final List<String> quizzes;

// // const \_ModuleSample({required this.title, required this.quizzes});

// // }

// // import 'package:flutter/material.dart';

// // class QuizTab extends StatelessWidget {

// // const QuizTab({super.key});

// // @override

// // Widget build(BuildContext context) {

// // print("✅ QuizTab affiché"); // debug

// // return Scaffold(

// // appBar: AppBar(title: const Text('📘 Quiz (debug)')),

// // body: ListView(

// // padding: const EdgeInsets.all(12),

// // children: [

// // ListTile(

// // title: const Text("Quiz de test"),

// // onTap: () {

// // print("➡️ Ouverture QuizPlayScreen");

// // Navigator.push(

// // context,

// // MaterialPageRoute(builder: (\_) => const QuizPlayScreen()),

// // );

// // },

// // ),

// // ],

// // ),

// // );

// // }

// // }

// // class QuizPlayScreen extends StatefulWidget {

// // const QuizPlayScreen({super.key});

// // @override

// // State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

// // }

// // class \_QuizPlayScreenState extends State<QuizPlayScreen> {

// // @override

// // void initState() {

// // super.initState();

// // print("✅ QuizPlayScreen init");

// // }

// // @override

// // Widget build(BuildContext context) {

// // print("✅ QuizPlayScreen build()");

// // return Scaffold(

// // appBar: AppBar(title: const Text("Quiz test")),

// // body: const Center(

// // child: Text("Ceci est un test d'affichage"),

// // ),

// // );

// // }

// // }

// import 'package:flutter/material.dart';

// class QuizTab extends StatelessWidget {

// const QuizTab({super.key});

// static const \_modules = <\_ModuleSample>[

// \_ModuleSample(

// title: 'Mathématiques BTS',

// quizzes: [

// 'Quiz 1 - Fondamentaux',

// ],

// ),

// \_ModuleSample(

// title: 'Comptabilité BTS',

// quizzes: [

// 'Quiz 1 - Fondamentaux',

// ],

// ),

// ];

// @override

// Widget build(BuildContext context) {

// print("✅ QuizTab affiché");

// return Scaffold(

// appBar: AppBar(title: const Text('📘 Quiz 30% stable Web')),

// body: ListView(

// padding: const EdgeInsets.all(12),

// children: [

// const \_Header(),

// const SizedBox(height: 12),

// ...\_modules.map((m) => Card(

// elevation: 2,

// child: ExpansionTile(

// leading: const Icon(Icons.menu\_book, size: 36),

// title: Text(m.title, style: const TextStyle(fontSize: 16)),

// subtitle: Text('${m.quizzes.length} quiz'),

// children: m.quizzes.map((q) {

// return ListTile(

// leading: const Icon(Icons.quiz, color: Colors.blue),

// title: Text(q),

// trailing: const Icon(Icons.chevron\_right),

// onTap: () {

// Navigator.of(context).push(

// MaterialPageRoute(

// builder: (\_) => QuizPlayScreen(title: q),

// ),

// );

// },

// );

// }).toList(),

// ),

// )),

// const SizedBox(height: 24),

// ],

// ),

// );

// }

// }

// class \_Header extends StatelessWidget {

// const \_Header();

// @override

// Widget build(BuildContext context) {

// return Container(

// padding: const EdgeInsets.all(14),

// decoration: BoxDecoration(

// color: Colors.blue.shade50,

// borderRadius: BorderRadius.circular(12),

// ),

// child: const Row(

// children: [

// Icon(Icons.school, size: 28),

// SizedBox(width: 10),

// Expanded(

// child: Text(

// '👋 Étape 30% : navigation stable et score.\n'

// '👉 Version Web safe, 2 modules inclus.',

// style: TextStyle(fontSize: 16, fontWeight: FontWeight.w600),

// ),

// ),

// ],

// ),

// );

// }

// }

// /// --------------------- Quiz Screen ---------------------

// class QuizPlayScreen extends StatefulWidget {

// final String title;

// const QuizPlayScreen({required this.title, super.key});

// @override

// State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

// }

// class \_QuizPlayScreenState extends State<QuizPlayScreen> {

// int index = 0;

// final Map<int, int> answers = {};

// bool finished = false;

// final List<\_QuestionSample> questions = const [

// \_QuestionSample(

// text: "Combien font 2 + 2 ?",

// options: ["3", "4", "5"],

// correctIndex: 1,

// ),

// \_QuestionSample(

// text: "La capitale de la France est…",

// options: ["Londres", "Paris", "Berlin"],

// correctIndex: 1,

// ),

// \_QuestionSample(

// text: "Quelle couleur obtient-on en mélangeant bleu et jaune ?",

// options: ["Vert", "Orange", "Violet"],

// correctIndex: 0,

// ),

// ];

// void \_select(int choice) {

// if (finished) return;

// setState(() => answers[index] = choice);

// }

// void \_next() {

// if (index < questions.length - 1) setState(() => index++);

// }

// void \_prev() {

// if (index > 0) setState(() => index--);

// }

// int get score {

// int s = 0;

// for (var entry in answers.entries) {

// if (questions[entry.key].correctIndex == entry.value) s++;

// }

// return s;

// }

// void \_finish() {

// if (!mounted) return;

// setState(() => finished = true);

// final total = questions.length;

// final bonnes = score;

// showDialog(

// context: context,

// builder: (\_) => AlertDialog(

// title: const Text("Quiz terminé"),

// content: Text("✅ Score : $bonnes / $total"),

// actions: [

// TextButton(

// onPressed: () => Navigator.pop(context),

// child: const Text("OK"),

// )

// ],

// ),

// );

// }

// @override

// Widget build(BuildContext context) {

// final q = questions[index];

// final int? selected = answers[index];

// return Scaffold(

// appBar: AppBar(title: Text(widget.title)),

// body: Padding(

// padding: const EdgeInsets.all(16),

// child: SingleChildScrollView(

// child: Column(

// crossAxisAlignment: CrossAxisAlignment.start,

// children: [

// Text(

// "Question ${index + 1}/${questions.length}",

// style:

// const TextStyle(fontSize: 18, fontWeight: FontWeight.bold),

// ),

// const SizedBox(height: 10),

// LinearProgressIndicator(value: (index + 1) / questions.length),

// const SizedBox(height: 20),

// Text(q.text, style: const TextStyle(fontSize: 20)),

// const SizedBox(height: 20),

// // Options

// ...List.generate(q.options.length, (i) {

// final isCorrect = i == q.correctIndex;

// Color? color;

// if (finished) {

// if (isCorrect) color = Colors.green.shade200;

// if (selected == i && !isCorrect) color = Colors.red.shade200;

// } else if (selected == i) {

// color = Colors.blue.shade200;

// }

// return Card(

// color: color,

// child: ListTile(

// title: Text(q.options[i]),

// onTap: () => \_select(i),

// ),

// );

// }),

// const SizedBox(height: 20),

// // Buttons - Web-safe

// Row(

// children: [

// Expanded(

// child: ElevatedButton(

// onPressed: index > 0 ? \_prev : null,

// child: const Text("⬅️ Précédent"),

// ),

// ),

// const SizedBox(width: 12),

// Expanded(

// child: !finished

// ? (index < questions.length - 1

// ? ElevatedButton(

// onPressed: \_next,

// child: const Text("Suivant ➡️"),

// )

// : ElevatedButton(

// onPressed: \_finish,

// child: const Text("✅ Terminer"),

// ))

// : ElevatedButton(

// onPressed: () => Navigator.pop(context),

// child: const Text("↩️ Quitter"),

// ),

// ),

// ],

// ),

// ],

// ),

// ),

// ),

// );

// }

// }

// class \_ModuleSample {

// final String title;

// final List<String> quizzes;

// const \_ModuleSample({required this.title, required this.quizzes});

// }

// class \_QuestionSample {

// final String text;

// final List<String> options;

// final int correctIndex;

// const \_QuestionSample({

// required this.text,

// required this.options,

// required this.correctIndex,

// });

// }

// import 'package:flutter/material.dart';

// import 'package:cloud\_firestore/cloud\_firestore.dart';

// import 'package:firebase\_auth/firebase\_auth.dart';

// class QuizTab extends StatefulWidget {

// const QuizTab({super.key});

// @override

// State<QuizTab> createState() => \_QuizTabState();

// }

// class \_QuizTabState extends State<QuizTab> {

// final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

// late Future<List<Module>> \_modulesFuture;

// @override

// void initState() {

// super.initState();

// \_modulesFuture = \_loadModules();

// }

// Future<List<Module>> \_loadModules() async {

// final modulesSnapshot = await \_firestore.collection('modules').get();

// List<Module> modules = [];

// for (var doc in modulesSnapshot.docs) {

// String moduleId = doc.id;

// String title = doc['title'] ?? 'Module';

// final quizzesSnapshot = await \_firestore

// .collection('modules/$moduleId/quizzes')

// .orderBy('order')

// .get();

// List<Quiz> quizzes = quizzesSnapshot.docs.map((qDoc) {

// final data = qDoc.data();

// return Quiz(

// id: data['id'] ?? qDoc.id,

// title: data['title'] ?? 'Quiz',

// badgeThresholds: Map<String, int>.from(data['badgeThresholds'] ??

// {'gold': 90, 'silver': 75, 'bronze': 50}),

// questions: (data['questions'] as List<dynamic>? ?? []).map((qq) {

// return Question(

// question: qq['question'] ?? '',

// options: List<String>.from(qq['options'] ?? []),

// correctIndex: qq['correctIndex'] ?? 0,

// explanation: qq['explanation'] ?? '',

// );

// }).toList(),

// );

// }).toList();

// modules.add(Module(title: title, quizzes: quizzes));

// }

// return modules;

// }

// @override

// Widget build(BuildContext context) {

// return Scaffold(

// appBar: AppBar(title: const Text('📘 Quiz 100% Firebase')),

// body: FutureBuilder<List<Module>>(

// future: \_modulesFuture,

// builder: (context, snapshot) {

// if (snapshot.connectionState == ConnectionState.waiting) {

// return const Center(child: CircularProgressIndicator());

// }

// if (snapshot.hasError) {

// return Center(child: Text('Erreur: ${snapshot.error}'));

// }

// final modules = snapshot.data ?? [];

// if (modules.isEmpty)

// return const Center(child: Text('Aucun module trouvé'));

// return ListView(

// padding: const EdgeInsets.all(12),

// children: [

// const \_Header(),

// const SizedBox(height: 12),

// ...modules.map((m) => Card(

// elevation: 2,

// child: ExpansionTile(

// leading: const Icon(Icons.menu\_book, size: 36),

// title:

// Text(m.title, style: const TextStyle(fontSize: 16)),

// subtitle: Text('${m.quizzes.length} quiz'),

// children: m.quizzes.map((q) {

// return FutureBuilder<int?>(

// future: \_getUserBestScore(q.id),

// builder: (context, scoreSnap) {

// String subtitle = scoreSnap.data != null

// ? 'Meilleur score: ${scoreSnap.data}'

// : '';

// return ListTile(

// leading:

// const Icon(Icons.quiz, color: Colors.blue),

// title: Text(q.title),

// subtitle: Text(subtitle),

// trailing: const Icon(Icons.chevron\_right),

// onTap: () {

// Navigator.of(context).push(

// MaterialPageRoute(

// builder: (\_) => QuizPlayScreen(quiz: q),

// ),

// );

// },

// );

// });

// }).toList(),

// ),

// )),

// const SizedBox(height: 24),

// ],

// );

// },

// ),

// );

// }

// Future<int?> \_getUserBestScore(String quizId) async {

// final uid = FirebaseAuth.instance.currentUser!.uid;

// final doc =

// await \_firestore.collection('users/$uid/quizHistory').doc(quizId).get();

// if (doc.exists) {

// return doc.data()?['score'] as int?;

// }

// return null;

// }

// }

// /// ================= Header

// class \_Header extends StatelessWidget {

// const \_Header();

// @override

// Widget build(BuildContext context) {

// return Container(

// padding: const EdgeInsets.all(14),

// decoration: BoxDecoration(

// color: Colors.blue.shade50,

// borderRadius: BorderRadius.circular(12),

// ),

// child: const Row(

// children: [

// Icon(Icons.school, size: 28),

// SizedBox(width: 10),

// Expanded(

// child: Text(

// '👋 Étape 100% : score sauvegardé + badges.\n'

// '👉 Historique et meilleur score utilisateur.',

// style: TextStyle(fontSize: 16, fontWeight: FontWeight.w600),

// ),

// ),

// ],

// ),

// );

// }

// }

// /// ================= Models

// class Module {

// final String title;

// final List<Quiz> quizzes;

// Module({required this.title, required this.quizzes});

// }

// class Quiz {

// final String id;

// final String title;

// final List<Question> questions;

// final Map<String, int> badgeThresholds;

// Quiz(

// {required this.id,

// required this.title,

// required this.questions,

// required this.badgeThresholds});

// }

// class Question {

// final String question;

// final List<String> options;

// final int correctIndex;

// final String explanation;

// Question({

// required this.question,

// required this.options,

// required this.correctIndex,

// required this.explanation,

// });

// }

// /// ================= QuizPlayScreen

// class QuizPlayScreen extends StatefulWidget {

// final Quiz quiz;

// const QuizPlayScreen({super.key, required this.quiz});

// @override

// State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

// }

// class \_QuizPlayScreenState extends State<QuizPlayScreen> {

// int index = 0;

// int score = 0;

// int? selectedIndex;

// bool answered = false;

// final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

// final String uid = FirebaseAuth.instance.currentUser!.uid;

// void \_select(int i) {

// if (answered) return;

// setState(() {

// selectedIndex = i;

// answered = true;

// if (i == widget.quiz.questions[index].correctIndex) score++;

// });

// }

// void \_next() {

// if (index < widget.quiz.questions.length - 1) {

// setState(() {

// index++;

// selectedIndex = null;

// answered = false;

// });

// }

// }

// void \_prev() {

// if (index > 0) {

// setState(() {

// index--;

// selectedIndex = null;

// answered = false;

// });

// }

// }

// String \_calculateBadge(int score, int total, Map<String, int> thresholds) {

// double percent = score / total \* 100;

// if (percent >= thresholds['gold']!) return 'Gold 🥇';

// if (percent >= thresholds['silver']!) return 'Silver 🥈';

// if (percent >= thresholds['bronze']!) return 'Bronze 🥉';

// return 'Aucun';

// }

// Future<void> \_saveScore() async {

// final badge = \_calculateBadge(

// score, widget.quiz.questions.length, widget.quiz.badgeThresholds);

// await \_firestore

// .collection('users/$uid/quizHistory')

// .doc(widget.quiz.id)

// .set({

// 'score': score,

// 'total': widget.quiz.questions.length,

// 'badge': badge,

// 'updatedAt': FieldValue.serverTimestamp(),

// }, SetOptions(merge: true));

// }

// void \_finish() async {

// await \_saveScore();

// showDialog(

// context: context,

// builder: (\_) => AlertDialog(

// title: const Text('Quiz terminé 🎉'),

// content: Text('Ton score : $score / ${widget.quiz.questions.length}\n'

// 'Badge : ${\_calculateBadge(score, widget.quiz.questions.length, widget.quiz.badgeThresholds)}'),

// actions: [

// TextButton(

// onPressed: () {

// Navigator.pop(context);

// setState(() {

// index = 0;

// score = 0;

// selectedIndex = null;

// answered = false;

// });

// },

// child: const Text('Rejouer'),

// ),

// TextButton(

// onPressed: () => Navigator.pop(context),

// child: const Text('Fermer'),

// ),

// ],

// ),

// );

// }

// @override

// Widget build(BuildContext context) {

// final question = widget.quiz.questions[index];

// return Scaffold(

// appBar: AppBar(title: Text(widget.quiz.title)),

// body: Padding(

// padding: const EdgeInsets.all(16),

// child: Column(

// children: [

// Text(

// 'Question ${index + 1} / ${widget.quiz.questions.length} | Score: $score',

// style: const TextStyle(fontSize: 18),

// ),

// const SizedBox(height: 10),

// LinearProgressIndicator(

// value: (index + 1) / widget.quiz.questions.length),

// const SizedBox(height: 20),

// Text(

// question.question,

// style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),

// ),

// const SizedBox(height: 20),

// Expanded(

// child: ListView.builder(

// itemCount: question.options.length,

// itemBuilder: (\_, i) {

// Color color = Colors.grey.shade200;

// if (answered) {

// if (i == question.correctIndex)

// color = Colors.green.shade300;

// else if (i == selectedIndex) color = Colors.red.shade300;

// }

// return Card(

// color: color,

// child: ListTile(

// title: Text(question.options[i]),

// onTap: () => \_select(i),

// ),

// );

// },

// ),

// ),

// const SizedBox(height: 20),

// Row(

// children: [

// Expanded(

// child: ElevatedButton(

// onPressed: index > 0 ? \_prev : null,

// child: const Text('⬅️ Précédent'),

// ),

// ),

// const SizedBox(width: 12),

// Expanded(

// child: index < widget.quiz.questions.length - 1

// ? ElevatedButton(

// onPressed: answered ? \_next : null,

// child: const Text('Suivant ➡️'),

// )

// : ElevatedButton(

// onPressed: \_finish,

// child: const Text('✅ Terminer'),

// ),

// ),

// ],

// ),

// ],

// ),

// ),

// );

// }

// }

// import 'package:flutter/material.dart';

// import 'package:cloud\_firestore/cloud\_firestore.dart';

// class QuizTab extends StatefulWidget {

// const QuizTab({super.key});

// @override

// State<QuizTab> createState() => \_QuizTabState();

// }

// class \_QuizTabState extends State<QuizTab> {

// final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

// late Future<List<Module>> \_modulesFuture;

// @override

// void initState() {

// super.initState();

// \_modulesFuture = \_loadModules();

// }

// Future<List<Module>> \_loadModules() async {

// final modulesSnapshot = await \_firestore.collection('modules').get();

// List<Module> modules = [];

// for (var doc in modulesSnapshot.docs) {

// String moduleId = doc.id;

// String title = doc['title'] ?? 'Module';

// // Charger les quizzes depuis la sous-collection

// final quizzesSnapshot = await \_firestore

// .collection('modules/$moduleId/quizzes')

// .orderBy('order')

// .get();

// List<Quiz> quizzes = quizzesSnapshot.docs.map((qDoc) {

// final data = qDoc.data();

// return Quiz(

// id: data['id'] ?? qDoc.id,

// title: data['title'] ?? 'Quiz',

// questions: (data['questions'] as List<dynamic>? ?? []).map((qq) {

// return Question(

// question: qq['question'] ?? '',

// options: List<String>.from(qq['options'] ?? []),

// correctIndex: qq['correctIndex'] ?? 0,

// explanation: qq['explanation'] ?? '',

// );

// }).toList(),

// );

// }).toList();

// modules.add(Module(title: title, quizzes: quizzes));

// }

// return modules;

// }

// @override

// Widget build(BuildContext context) {

// return Scaffold(

// appBar: AppBar(title: const Text('📘 Quiz 50% Firebase')),

// body: FutureBuilder<List<Module>>(

// future: \_modulesFuture,

// builder: (context, snapshot) {

// if (snapshot.connectionState == ConnectionState.waiting) {

// return const Center(child: CircularProgressIndicator());

// }

// if (snapshot.hasError) {

// return Center(child: Text('Erreur: ${snapshot.error}'));

// }

// final modules = snapshot.data ?? [];

// if (modules.isEmpty) return const Center(child: Text('Aucun module trouvé'));

// return ListView(

// padding: const EdgeInsets.all(12),

// children: [

// const \_Header(),

// const SizedBox(height: 12),

// ...modules.map((m) => Card(

// elevation: 2,

// child: ExpansionTile(

// leading: const Icon(Icons.menu\_book, size: 36),

// title: Text(m.title, style: const TextStyle(fontSize: 16)),

// subtitle: Text('${m.quizzes.length} quiz'),

// children: m.quizzes.map((q) {

// return ListTile(

// leading: const Icon(Icons.quiz, color: Colors.blue),

// title: Text(q.title),

// trailing: const Icon(Icons.chevron\_right),

// onTap: () {

// Navigator.of(context).push(

// MaterialPageRoute(

// builder: (\_) => QuizPlayScreen(quiz: q),

// ),

// );

// },

// );

// }).toList(),

// ),

// )),

// const SizedBox(height: 24),

// ],

// );

// },

// ),

// );

// }

// }

// /// ================= Header

// class \_Header extends StatelessWidget {

// const \_Header();

// @override

// Widget build(BuildContext context) {

// return Container(

// padding: const EdgeInsets.all(14),

// decoration: BoxDecoration(

// color: Colors.blue.shade50,

// borderRadius: BorderRadius.circular(12),

// ),

// child: const Row(

// children: [

// Icon(Icons.school, size: 28),

// SizedBox(width: 10),

// Expanded(

// child: Text(

// '👋 Étape 50% : données Firebase dynamiques.\n'

// '👉 Modules + quizzes récupérés depuis Firestore.',

// style: TextStyle(fontSize: 16, fontWeight: FontWeight.w600),

// ),

// ),

// ],

// ),

// );

// }

// }

// /// ================= Models

// class Module {

// final String title;

// final List<Quiz> quizzes;

// Module({required this.title, required this.quizzes});

// }

// class Quiz {

// final String id;

// final String title;

// final List<Question> questions;

// Quiz({required this.id, required this.title, required this.questions});

// }

// class Question {

// final String question;

// final List<String> options;

// final int correctIndex;

// final String explanation;

// Question({

// required this.question,

// required this.options,

// required this.correctIndex,

// required this.explanation,

// });

// }

// /// ================= QuizPlayScreen

// class QuizPlayScreen extends StatefulWidget {

// final Quiz quiz;

// const QuizPlayScreen({super.key, required this.quiz});

// @override

// State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

// }

// class \_QuizPlayScreenState extends State<QuizPlayScreen> {

// int index = 0;

// int score = 0;

// int? selectedIndex;

// bool answered = false;

// void \_select(int i) {

// if (answered) return;

// setState(() {

// selectedIndex = i;

// answered = true;

// if (i == widget.quiz.questions[index].correctIndex) score++;

// });

// }

// void \_next() {

// if (index < widget.quiz.questions.length - 1) {

// setState(() {

// index++;

// selectedIndex = null;

// answered = false;

// });

// }

// }

// void \_prev() {

// if (index > 0) {

// setState(() {

// index--;

// selectedIndex = null;

// answered = false;

// });

// }

// }

// void \_finish() {

// showDialog(

// context: context,

// builder: (\_) => AlertDialog(

// title: const Text('Quiz terminé 🎉'),

// content: Text('Ton score : $score / ${widget.quiz.questions.length}'),

// actions: [

// TextButton(

// onPressed: () {

// Navigator.pop(context);

// setState(() {

// index = 0;

// score = 0;

// selectedIndex = null;

// answered = false;

// });

// },

// child: const Text('Rejouer'),

// ),

// TextButton(

// onPressed: () => Navigator.pop(context),

// child: const Text('Fermer'),

// ),

// ],

// ),

// );

// }

// @override

// Widget build(BuildContext context) {

// final question = widget.quiz.questions[index];

// return Scaffold(

// appBar: AppBar(title: Text(widget.quiz.title)),

// body: Padding(

// padding: const EdgeInsets.all(16),

// child: Column(

// children: [

// Text(

// 'Question ${index + 1} / ${widget.quiz.questions.length} | Score: $score',

// style: const TextStyle(fontSize: 18),

// ),

// const SizedBox(height: 10),

// LinearProgressIndicator(value: (index + 1) / widget.quiz.questions.length),

// const SizedBox(height: 20),

// Text(

// question.question,

// style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),

// ),

// const SizedBox(height: 20),

// Expanded(

// child: ListView.builder(

// itemCount: question.options.length,

// itemBuilder: (\_, i) {

// Color color = Colors.grey.shade200;

// if (answered) {

// if (i == question.correctIndex) color = Colors.green.shade300;

// else if (i == selectedIndex) color = Colors.red.shade300;

// }

// return Card(

// color: color,

// child: ListTile(

// title: Text(question.options[i]),

// onTap: () => \_select(i),

// ),

// );

// },

// ),

// ),

// const SizedBox(height: 20),

// Row(

// children: [

// Expanded(

// child: ElevatedButton(

// onPressed: index > 0 ? \_prev : null,

// child: const Text('⬅️ Précédent'),

// ),

// ),

// const SizedBox(width: 12),

// Expanded(

// child: index < widget.quiz.questions.length - 1

// ? ElevatedButton(

// onPressed: answered ? \_next : null,

// child: const Text('Suivant ➡️'),

// )

// : ElevatedButton(

// onPressed: \_finish,

// child: const Text('✅ Terminer'),

// ),

// ),

// ],

// ),

// ],

// ),

// ),

// );

// }

// }

// import 'package:flutter/material.dart';

// import 'package:cloud\_firestore/cloud\_firestore.dart';

// import 'package:firebase\_auth/firebase\_auth.dart';

// /// ================= QuizTab

// class QuizTab extends StatefulWidget {

// const QuizTab({super.key});

// @override

// State<QuizTab> createState() => \_QuizTabState();

// }

// class \_QuizTabState extends State<QuizTab> {

// final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

// late Future<List<Module>> \_modulesFuture;

// @override

// void initState() {

// super.initState();

// \_modulesFuture = \_loadModules();

// }

// Future<List<Module>> \_loadModules() async {

// final modulesSnapshot = await \_firestore.collection('modules').get();

// List<Module> modules = [];

// for (var doc in modulesSnapshot.docs) {

// String moduleId = doc.id;

// String title = doc['title'] ?? 'Module';

// final quizzesSnapshot = await \_firestore

// .collection('modules/$moduleId/quizzes')

// .orderBy('order', descending: false)

// .get();

// List<Quiz> quizzes = quizzesSnapshot.docs.map((qDoc) {

// final data = qDoc.data();

// return Quiz(

// id: data['id'] ?? qDoc.id,

// title: data['title'] ?? 'Quiz',

// badgeThresholds: Map<String, int>.from(data['badgeThresholds'] ??

// {

// 'gold': 90,

// 'silver': 75,

// 'bronze': 50,

// }),

// questions: (data['questions'] as List<dynamic>? ?? []).map((qq) {

// return Question(

// question: qq['question'] ?? '',

// options: List<String>.from(qq['options'] ?? []),

// correctIndex: qq['correctIndex'] ?? 0,

// explanation: qq['explanation'] ?? '',

// );

// }).toList(),

// );

// }).toList();

// modules.add(Module(title: title, quizzes: quizzes));

// }

// return modules;

// }

// Future<int?> \_getUserBestScore(String quizId) async {

// final user = FirebaseAuth.instance.currentUser;

// if (user == null) return null; // pas connecté → pas d'historique

// final doc = await \_firestore

// .collection('users/${user.uid}/quizHistory')

// .doc(quizId)

// .get();

// if (doc.exists) {

// return doc.data()?['score'] as int?;

// }

// return null;

// }

// @override

// Widget build(BuildContext context) {

// return Scaffold(

// appBar: AppBar(title: const Text('📘 Quiz Firebase sécurisé')),

// body: FutureBuilder<List<Module>>(

// future: \_modulesFuture,

// builder: (context, snapshot) {

// if (snapshot.connectionState == ConnectionState.waiting) {

// return const Center(child: CircularProgressIndicator());

// }

// if (snapshot.hasError) {

// return Center(child: Text('Erreur: ${snapshot.error}'));

// }

// final modules = snapshot.data ?? [];

// if (modules.isEmpty) {

// return const Center(child: Text('Aucun module trouvé'));

// }

// return ListView(

// padding: const EdgeInsets.all(12),

// children: [

// const \_Header(),

// const SizedBox(height: 12),

// ...modules.map((m) => Card(

// elevation: 2,

// child: ExpansionTile(

// leading: const Icon(Icons.menu\_book, size: 36),

// title:

// Text(m.title, style: const TextStyle(fontSize: 16)),

// subtitle: Text('${m.quizzes.length} quiz'),

// children: m.quizzes.map((q) {

// return FutureBuilder<int?>(

// future: \_getUserBestScore(q.id),

// builder: (context, scoreSnap) {

// String subtitle = '';

// if (scoreSnap.connectionState ==

// ConnectionState.done &&

// scoreSnap.data != null) {

// subtitle = 'Meilleur score: ${scoreSnap.data}';

// }

// return ListTile(

// leading:

// const Icon(Icons.quiz, color: Colors.blue),

// title: Text(q.title),

// subtitle: Text(subtitle),

// trailing: const Icon(Icons.chevron\_right),

// onTap: () {

// Navigator.of(context).push(

// MaterialPageRoute(

// builder: (\_) => QuizPlayScreen(quiz: q),

// ),

// );

// },

// );

// },

// );

// }).toList(),

// ),

// )),

// ],

// );

// },

// ),

// );

// }

// }

// /// ================= Header

// class \_Header extends StatelessWidget {

// const \_Header();

// @override

// Widget build(BuildContext context) {

// return Container(

// padding: const EdgeInsets.all(14),

// decoration: BoxDecoration(

// color: Colors.blue.shade50,

// borderRadius: BorderRadius.circular(12),

// ),

// child: const Row(

// children: [

// Icon(Icons.school, size: 28),

// SizedBox(width: 10),

// Expanded(

// child: Text(

// '👋 Étape 100% : score sauvegardé + badges.\n'

// '👉 Historique et meilleur score utilisateur.',

// style: TextStyle(fontSize: 16, fontWeight: FontWeight.w600),

// ),

// ),

// ],

// ),

// );

// }

// }

// /// ================= Models

// class Module {

// final String title;

// final List<Quiz> quizzes;

// Module({required this.title, required this.quizzes});

// }

// class Quiz {

// final String id;

// final String title;

// final List<Question> questions;

// final Map<String, int> badgeThresholds;

// Quiz({

// required this.id,

// required this.title,

// required this.questions,

// required this.badgeThresholds,

// });

// }

// class Question {

// final String question;

// final List<String> options;

// final int correctIndex;

// final String explanation;

// Question({

// required this.question,

// required this.options,

// required this.correctIndex,

// required this.explanation,

// });

// }

// /// ================= QuizPlayScreen

// class QuizPlayScreen extends StatefulWidget {

// final Quiz quiz;

// const QuizPlayScreen({super.key, required this.quiz});

// @override

// State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

// }

// class \_QuizPlayScreenState extends State<QuizPlayScreen> {

// int index = 0;

// int score = 0;

// int? selectedIndex;

// bool answered = false;

// final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

// final User? user = FirebaseAuth.instance.currentUser;

// void \_select(int i) {

// if (answered) return;

// setState(() {

// selectedIndex = i;

// answered = true;

// if (i == widget.quiz.questions[index].correctIndex) score++;

// });

// }

// void \_next() {

// if (index < widget.quiz.questions.length - 1) {

// setState(() {

// index++;

// selectedIndex = null;

// answered = false;

// });

// }

// }

// void \_prev() {

// if (index > 0) {

// setState(() {

// index--;

// selectedIndex = null;

// answered = false;

// });

// }

// }

// String \_calculateBadge(int score, int total, Map<String, int> thresholds) {

// double percent = score / total \* 100;

// if (percent >= thresholds['gold']!) return 'Gold 🥇';

// if (percent >= thresholds['silver']!) return 'Silver 🥈';

// if (percent >= thresholds['bronze']!) return 'Bronze 🥉';

// return 'Aucun';

// }

// Future<void> \_saveScore() async {

// if (user == null) return; // sécurité : utilisateur pas connecté

// final badge = \_calculateBadge(

// score, widget.quiz.questions.length, widget.quiz.badgeThresholds);

// await \_firestore

// .collection('users/${user!.uid}/quizHistory')

// .doc(widget.quiz.id)

// .set({

// 'score': score,

// 'total': widget.quiz.questions.length,

// 'badge': badge,

// 'updatedAt': FieldValue.serverTimestamp(),

// }, SetOptions(merge: true));

// }

// void \_finish() async {

// await \_saveScore();

// showDialog(

// context: context,

// builder: (\_) => AlertDialog(

// title: const Text('Quiz terminé 🎉'),

// content: Text('Ton score : $score / ${widget.quiz.questions.length}\n'

// 'Badge : ${\_calculateBadge(score, widget.quiz.questions.length, widget.quiz.badgeThresholds)}'),

// actions: [

// TextButton(

// onPressed: () {

// Navigator.pop(context);

// setState(() {

// index = 0;

// score = 0;

// selectedIndex = null;

// answered = false;

// });

// },

// child: const Text('Rejouer'),

// ),

// TextButton(

// onPressed: () => Navigator.pop(context),

// child: const Text('Fermer'),

// ),

// ],

// ),

// );

// }

// @override

// Widget build(BuildContext context) {

// final question = widget.quiz.questions[index];

// return Scaffold(

// appBar: AppBar(title: Text(widget.quiz.title)),

// body: Padding(

// padding: const EdgeInsets.all(16),

// child: Column(

// children: [

// Text(

// 'Question ${index + 1} / ${widget.quiz.questions.length} | Score: $score',

// style: const TextStyle(fontSize: 18),

// ),

// const SizedBox(height: 10),

// LinearProgressIndicator(

// value: (index + 1) / widget.quiz.questions.length,

// ),

// const SizedBox(height: 20),

// Text(

// question.question,

// style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),

// ),

// const SizedBox(height: 20),

// Expanded(

// child: ListView.builder(

// itemCount: question.options.length,

// itemBuilder: (\_, i) {

// Color color = Colors.grey.shade200;

// if (answered) {

// if (i == question.correctIndex) {

// color = Colors.green.shade300;

// } else if (i == selectedIndex) {

// color = Colors.red.shade300;

// }

// }

// return Card(

// color: color,

// child: ListTile(

// title: Text(question.options[i]),

// onTap: () => \_select(i),

// ),

// );

// },

// ),

// ),

// const SizedBox(height: 20),

// Row(

// children: [

// Expanded(

// child: ElevatedButton(

// onPressed: index > 0 ? \_prev : null,

// child: const Text('⬅️ Précédent'),

// ),

// ),

// const SizedBox(width: 12),

// Expanded(

// child: index < widget.quiz.questions.length - 1

// ? ElevatedButton(

// onPressed: answered ? \_next : null,

// child: const Text('Suivant ➡️'),

// )

// : ElevatedButton(

// onPressed: \_finish,

// child: const Text('✅ Terminer'),

// ),

// ),

// ],

// ),

// ],

// ),

// ),

// );

// }

// }

// // lib/screens/tabs/quiz\_tab.dart

// import 'package:flutter/material.dart';

// import 'package:cloud\_firestore/cloud\_firestore.dart';

// import 'package:firebase\_auth/firebase\_auth.dart';

// // Ajuste ces imports selon ton architecture si nécessaire

// import '../../models/quiz\_model.dart';

// import '../../models/question\_model.dart';

// /// ================= QuizTab: liste modules + quizzes

// class QuizTab extends StatefulWidget {

// const QuizTab({super.key});

// @override

// State<QuizTab> createState() => \_QuizTabState();

// }

// class \_QuizTabState extends State<QuizTab> {

// final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

// late Future<List<\_ModuleSimple>> \_modulesFuture;

// @override

// void initState() {

// super.initState();

// \_modulesFuture = \_loadModulesAndQuizzes();

// }

// // Module simple pour lister (id + titre + quizzes)

// Future<List<\_ModuleSimple>> \_loadModulesAndQuizzes() async {

// final modulesSnap = await \_firestore.collection('modules').get();

// final List<\_ModuleSimple> modules = [];

// for (final mDoc in modulesSnap.docs) {

// final moduleId = mDoc.id;

// final title = (mDoc.data()['title'] ?? 'Module').toString();

// final quizzesSnap = await \_firestore

// .collection('modules/$moduleId/quizzes')

// .orderBy('order', descending: false)

// .get();

// final quizzes = quizzesSnap.docs.map((qDoc) {

// final data = qDoc.data();

// // Utilise ton QuizModel.fromMap qui demande id + moduleId

// try {

// return QuizModel.fromMap(Map<String, dynamic>.from(data),

// id: qDoc.id, moduleId: moduleId);

// } catch (\_) {

// // fallback simple si la factory diffère

// final questions = (data['questions'] as List<dynamic>? ?? [])

// .map((qq) => QuestionModel.fromMap(Map<String, dynamic>.from(qq)))

// .toList();

// return QuizModel(

// id: qDoc.id,

// title: (data['title'] ?? 'Quiz').toString(),

// moduleId: moduleId,

// questions: questions,

// );

// }

// }).toList();

// modules.add(\_ModuleSimple(id: moduleId, title: title, quizzes: quizzes));

// }

// return modules;

// }

// Future<int?> \_getUserBestScore(String quizId) async {

// final user = FirebaseAuth.instance.currentUser;

// if (user == null) return null;

// final doc = await \_firestore

// .collection('users/${user.uid}/quizHistory')

// .doc(quizId)

// .get();

// if (doc.exists) return (doc.data()?['score'] as num?)?.toInt();

// return null;

// }

// @override

// Widget build(BuildContext context) {

// return Scaffold(

// appBar: AppBar(title: const Text('📘 Quiz')),

// body: FutureBuilder<List<\_ModuleSimple>>(

// future: \_modulesFuture,

// builder: (context, snap) {

// if (snap.connectionState == ConnectionState.waiting) {

// return const Center(child: CircularProgressIndicator());

// }

// if (snap.hasError) {

// return Center(child: Text('Erreur: ${snap.error}'));

// }

// final modules = snap.data ?? [];

// if (modules.isEmpty) return const Center(child: Text('Aucun module'));

// return ListView(

// padding: const EdgeInsets.all(12),

// children: [

// Container(

// padding: const EdgeInsets.all(12),

// decoration: BoxDecoration(

// color: Colors.blue.shade50,

// borderRadius: BorderRadius.circular(10),

// ),

// child: const Text(

// 'Choisis un module puis un quiz. Les réponses sont verrouillées une fois choisies. Le score est calculé uniquement à la fin.',

// ),

// ),

// const SizedBox(height: 12),

// ...modules.map((m) => Card(

// child: ExpansionTile(

// leading: const Icon(Icons.menu\_book),

// title: Text(m.title),

// subtitle: Text('${m.quizzes.length} quiz'),

// children: m.quizzes.map((q) {

// return FutureBuilder<int?>(

// future: \_getUserBestScore(q.id),

// builder: (context, s) {

// final best =

// (s.connectionState == ConnectionState.done)

// ? s.data

// : null;

// return ListTile(

// leading:

// const Icon(Icons.quiz, color: Colors.blue),

// title: Text(q.title),

// subtitle:

// best != null ? Text('Meilleur: $best') : null,

// trailing: const Icon(Icons.chevron\_right),

// onTap: () => Navigator.push(

// context,

// MaterialPageRoute(

// builder: (\_) => QuizPlayScreen(quiz: q),

// ),

// ),

// );

// },

// );

// }).toList(),

// ),

// )),

// ],

// );

// },

// ),

// );

// }

// }

// class \_ModuleSimple {

// final String id;

// final String title;

// final List<QuizModel> quizzes;

// \_ModuleSimple({required this.id, required this.title, required this.quizzes});

// }

// /// ================= QuizPlayScreen : toute la logique du quiz dans une page

// class QuizPlayScreen extends StatefulWidget {

// final QuizModel quiz;

// const QuizPlayScreen({super.key, required this.quiz});

// @override

// State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

// }

// class \_QuizPlayScreenState extends State<QuizPlayScreen> {

// int currentIndex = 0;

// late List<int> userAnswers; // -1 = pas répondu, sinon index choisi

// bool finished = false;

// bool saving = false;

// final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

// final User? user = FirebaseAuth.instance.currentUser;

// @override

// void initState() {

// super.initState();

// userAnswers = List<int>.filled(widget.quiz.questions.length, -1);

// }

// bool get allAnswered =>

// !userAnswers.any((ans) => ans == -1); // vrai si tout répondu

// void \_selectAnswer(int idx) {

// // verrouille la réponse : on ne peut modifier une fois choisie

// if (userAnswers[currentIndex] != -1) return;

// setState(() {

// userAnswers[currentIndex] = idx;

// });

// }

// void \_next() {

// if (currentIndex < widget.quiz.questions.length - 1) {

// setState(() => currentIndex++);

// }

// }

// void \_prev() {

// if (currentIndex > 0) {

// setState(() => currentIndex--);

// }

// }

// int \_computeScore() {

// int s = 0;

// for (var i = 0; i < widget.quiz.questions.length; i++) {

// final q = widget.quiz.questions[i];

// final ans = userAnswers[i];

// if (ans != -1 && ans == q.correctIndex) s++;

// }

// return s;

// }

// String \_computeBadge(int score, int total) {

// final thresholds = widget.quiz.badgeThresholds;

// final percent = total == 0 ? 0 : (score \* 100) / total;

// if (percent >= (thresholds['gold'] ?? 90)) return 'Gold 🥇';

// if (percent >= (thresholds['silver'] ?? 75)) return 'Silver 🥈';

// if (percent >= (thresholds['bronze'] ?? 50)) return 'Bronze 🥉';

// return 'Aucun';

// }

// Future<void> \_saveResultToFirestore({

// required int score,

// required int total,

// required String badge,

// }) async {

// if (user == null) return; // pas connecté -> on ne sauve pas

// final docRef = \_firestore

// .collection('users/${user!.uid}/quizHistory')

// .doc(widget.quiz.id);

// final bonnes = score;

// final fautes = total - score;

// final percent = total == 0 ? 0 : ((score \* 100) / total).round();

// final payload = {

// 'moduleId': widget.quiz.moduleId,

// 'quizId': widget.quiz.id,

// 'score': score,

// 'total': total,

// 'bonnes': bonnes,

// 'fautes': fautes,

// 'percent': percent,

// 'badge': badge,

// 'answers': userAnswers,

// 'updatedAt': FieldValue.serverTimestamp(),

// };

// await docRef.set(payload, SetOptions(merge: true));

// }

// Future<void> \_finishQuiz() async {

// setState(() => finished = true);

// final total = widget.quiz.questions.length;

// final score = \_computeScore();

// final badge = \_computeBadge(score, total);

// // sauvegarde si connecté (opération async)

// setState(() => saving = true);

// try {

// await \_saveResultToFirestore(score: score, total: total, badge: badge);

// } catch (e) {

// // ignore save error but inform user

// ScaffoldMessenger.of(context).showSnackBar(

// SnackBar(content: Text('Erreur sauvegarde: $e')),

// );

// } finally {

// setState(() => saving = false);

// }

// // on affiche le récap directement sur la page (pas seulement un dialog)

// // tu peux aussi ouvrir un dialog, mais l'UI en bas affiche le résumé

// // pour plus de visibilité, ouvrons un dialog + possibilité de review sur la même page

// showDialog(

// context: context,

// builder: (\_) => AlertDialog(

// title: const Text('Quiz terminé 🎉'),

// content: Text(

// 'Score : $score / $total\nBadge : $badge\n\nTu peux relire les réponses ou rejouer.'),

// actions: [

// TextButton(

// onPressed: () {

// Navigator.pop(context);

// // reste sur la page pour relecture (finished == true)

// },

// child: const Text('OK'),

// ),

// ],

// ),

// );

// }

// void \_replay() {

// setState(() {

// userAnswers = List<int>.filled(widget.quiz.questions.length, -1);

// currentIndex = 0;

// finished = false;

// });

// }

// Widget \_buildQuestionView() {

// final q = widget.quiz.questions[currentIndex];

// final answer = userAnswers[currentIndex];

// return Column(

// crossAxisAlignment: CrossAxisAlignment.start,

// children: [

// Text(

// 'Question ${currentIndex + 1} / ${widget.quiz.questions.length}',

// style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),

// ),

// const SizedBox(height: 8),

// LinearProgressIndicator(

// value: (currentIndex + 1) / widget.quiz.questions.length,

// ),

// const SizedBox(height: 16),

// Text(q.question, style: const TextStyle(fontSize: 20)),

// const SizedBox(height: 12),

// ...List.generate(q.options.length, (i) {

// final isSelected = answer == i;

// // si la question a été répondue -> on verrouille la modification

// final disabled = userAnswers[currentIndex] != -1;

// Color? color;

// if (disabled) {

// if (i == q.correctIndex)

// color = Colors.green.shade200;

// else if (isSelected)

// color = Colors.red.shade200;

// else

// color = Colors.grey.shade100;

// } else if (isSelected) {

// color = Colors.blue.shade100;

// } else {

// color = Colors.white;

// }

// return Card(

// color: color,

// child: ListTile(

// title: Text(q.options[i]),

// onTap: disabled ? null : () => \_selectAnswer(i),

// ),

// );

// }),

// const SizedBox(height: 8),

// if (userAnswers[currentIndex] != -1)

// Padding(

// padding: const EdgeInsets.symmetric(vertical: 8.0),

// child: Text(

// userAnswers[currentIndex] == q.correctIndex

// ? '✅ Bonne réponse'

// : '❌ Mauvaise réponse — Réponse correcte: ${q.options[q.correctIndex]}',

// style: TextStyle(

// color: userAnswers[currentIndex] == q.correctIndex

// ? Colors.green

// : Colors.red,

// fontWeight: FontWeight.w700),

// ),

// ),

// if (userAnswers[currentIndex] != -1 &&

// (q.explanation?.isNotEmpty ?? false))

// Padding(

// padding: const EdgeInsets.only(top: 6),

// child: Text(

// 'Explication: ${q.explanation}',

// style: TextStyle(color: Colors.black87),

// ),

// ),

// ],

// );

// }

// Widget \_buildResultSummary() {

// final total = widget.quiz.questions.length;

// final score = \_computeScore();

// final badge = \_computeBadge(score, total);

// final bonnes = score;

// final fautes = total - score;

// final percent = total == 0 ? 0 : ((score \* 100) / total).round();

// return SingleChildScrollView(

// child: Column(

// crossAxisAlignment: CrossAxisAlignment.stretch,

// children: [

// Card(

// color: Colors.blue.shade50,

// child: Padding(

// padding: const EdgeInsets.all(12),

// child: Column(

// children: [

// Text('Résultat',

// style: Theme.of(context).textTheme.titleLarge),

// const SizedBox(height: 8),

// Text('Score : $score / $total',

// style: const TextStyle(

// fontSize: 18, fontWeight: FontWeight.bold)),

// const SizedBox(height: 6),

// Text('Badge : $badge', style: const TextStyle(fontSize: 16)),

// const SizedBox(height: 6),

// Text('Bonnes : $bonnes — Fautes : $fautes — $percent%'),

// if (saving)

// const Padding(

// padding: EdgeInsets.only(top: 8.0),

// child: LinearProgressIndicator(),

// ),

// ],

// ),

// ),

// ),

// const SizedBox(height: 12),

// // Relecture détaillée

// Card(

// child: Padding(

// padding: const EdgeInsets.all(12),

// child: Column(

// crossAxisAlignment: CrossAxisAlignment.start,

// children: [

// Text('Relecture',

// style: Theme.of(context).textTheme.titleMedium),

// const SizedBox(height: 8),

// ...List.generate(widget.quiz.questions.length, (i) {

// final q = widget.quiz.questions[i];

// final ans = userAnswers[i];

// final isCorrect = ans != -1 && ans == q.correctIndex;

// return ListTile(

// tileColor:

// isCorrect ? Colors.green.shade50 : Colors.red.shade50,

// title: Text('Q${i + 1}: ${q.question}'),

// subtitle: Column(

// crossAxisAlignment: CrossAxisAlignment.start,

// children: [

// const SizedBox(height: 4),

// Text(

// 'Ta réponse: ${ans == -1 ? "—" : q.options[ans]}'),

// Text('Bonne réponse: ${q.options[q.correctIndex]}'),

// if ((q.explanation?.isNotEmpty ?? false))

// Padding(

// padding: const EdgeInsets.only(top: 6),

// child: Text('Explication: ${q.explanation}'),

// ),

// ],

// ),

// );

// }),

// ],

// ),

// ),

// ),

// const SizedBox(height: 12),

// Row(

// children: [

// Expanded(

// child: ElevatedButton(

// onPressed: \_replay,

// child: const Text('Rejouer'),

// ),

// ),

// const SizedBox(width: 12),

// Expanded(

// child: OutlinedButton(

// onPressed: () => Navigator.pop(context),

// child: const Text('Fermer'),

// ),

// ),

// ],

// ),

// ],

// ),

// );

// }

// @override

// Widget build(BuildContext context) {

// // sécurité : si pas de questions

// if (widget.quiz.questions.isEmpty) {

// return Scaffold(

// appBar: AppBar(title: Text(widget.quiz.title)),

// body:

// const Center(child: Text('Ce quiz ne contient pas de questions.')),

// );

// }

// return Scaffold(

// appBar: AppBar(title: Text(widget.quiz.title)),

// body: Padding(

// padding: const EdgeInsets.all(12),

// child: finished

// ? \_buildResultSummary()

// : Column(

// children: [

// Expanded(

// child:

// SingleChildScrollView(child: \_buildQuestionView())),

// const SizedBox(height: 8),

// Row(

// children: [

// Expanded(

// child: ElevatedButton(

// onPressed: currentIndex > 0 ? \_prev : null,

// child: const Text('⬅️ Précédent'),

// ),

// ),

// const SizedBox(width: 8),

// Expanded(

// child: (currentIndex < widget.quiz.questions.length - 1)

// ? ElevatedButton(

// onPressed: userAnswers[currentIndex] != -1

// ? \_next

// : null, // n'avance que si la question courante est répondue

// child: const Text('Suivant ➡️'),

// )

// : ElevatedButton(

// onPressed: allAnswered ? \_finishQuiz : null,

// child: const Text('✅ Terminer'),

// ),

// ),

// ],

// ),

// ],

// ),

// ),

// );

// }

// }

// lib/screens/tabs/quiz\_tab.dart

import 'package:flutter/material.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

// Ajuste ces imports selon ton architecture si nécessaire

import '../../models/quiz\_model.dart';

import '../../models/question\_model.dart';

import '../../models/quiz\_result\_model.dart';

/// ================= QuizTab: liste modules + quizzes + historique

class QuizTab extends StatefulWidget {

const QuizTab({super.key});

@override

State<QuizTab> createState() => \_QuizTabState();

}

class \_QuizTabState extends State<QuizTab> {

final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

late Future<List<\_ModuleSimple>> \_modulesFuture;

final User? user = FirebaseAuth.instance.currentUser;

@override

void initState() {

super.initState();

\_modulesFuture = \_loadModulesAndQuizzes();

}

// Module simple pour lister (id + titre + quizzes)

Future<List<\_ModuleSimple>> \_loadModulesAndQuizzes() async {

final modulesSnap = await \_firestore.collection('modules').get();

final List<\_ModuleSimple> modules = [];

for (final mDoc in modulesSnap.docs) {

final moduleId = mDoc.id;

final title = (mDoc.data()['title'] ?? 'Module').toString();

final quizzesSnap = await \_firestore

.collection('modules/$moduleId/quizzes')

.orderBy('order', descending: false)

.get();

final quizzes = quizzesSnap.docs.map((qDoc) {

final data = qDoc.data();

try {

return QuizModel.fromMap(Map<String, dynamic>.from(data),

id: qDoc.id, moduleId: moduleId);

} catch (\_) {

final questions = (data['questions'] as List<dynamic>? ?? [])

.map((qq) => QuestionModel.fromMap(Map<String, dynamic>.from(qq)))

.toList();

return QuizModel(

id: qDoc.id,

title: (data['title'] ?? 'Quiz').toString(),

moduleId: moduleId,

questions: questions,

);

}

}).toList();

modules.add(\_ModuleSimple(id: moduleId, title: title, quizzes: quizzes));

}

return modules;

}

Future<int?> \_getUserBestScore(String quizId) async {

if (user == null) return null;

final doc = await \_firestore

.collection('users/${user!.uid}/quizHistory')

.doc(quizId)

.get();

if (doc.exists) return (doc.data()?['score'] as num?)?.toInt();

return null;

}

/// 🔹 Historique en dessous

Widget \_buildHistory(String userId) {

return StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(

stream: \_firestore

.collection("users/$userId/quizHistory")

.orderBy("date", descending: true)

.snapshots(),

builder: (context, snapshot) {

if (!snapshot.hasData) return const SizedBox();

final results = snapshot.data!.docs

.map((doc) => QuizResult.fromFirestore(doc))

.toList();

if (results.isEmpty) {

return const SizedBox();

}

return Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

const Divider(),

const Text(

"📜 Historique de mes quiz",

style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),

),

...results.map((r) => Card(

child: ListTile(

title: Text("${r.quizId} : ${r.score}/${r.total}"),

subtitle: Text(

"Badge: ${r.badge} | ${r.percent}% | ${r.date.toLocal().toString().split(' ')[0]}"),

trailing: const Icon(Icons.history),

onTap: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (\_) => QuizReviewScreen(result: r),

),

);

},

),

)),

],

);

},

);

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: const Text('📘 Quiz')),

body: FutureBuilder<List<\_ModuleSimple>>(

future: \_modulesFuture,

builder: (context, snap) {

if (snap.connectionState == ConnectionState.waiting) {

return const Center(child: CircularProgressIndicator());

}

if (snap.hasError) {

return Center(child: Text('Erreur: ${snap.error}'));

}

final modules = snap.data ?? [];

if (modules.isEmpty) return const Center(child: Text('Aucun module'));

return ListView(

padding: const EdgeInsets.all(12),

children: [

Container(

padding: const EdgeInsets.all(12),

decoration: BoxDecoration(

color: Colors.blue.shade50,

borderRadius: BorderRadius.circular(10),

),

child: const Text(

'Choisis un module puis un quiz. Les réponses sont verrouillées une fois choisies. Le score est calculé uniquement à la fin.',

),

),

const SizedBox(height: 12),

...modules.map((m) => Card(

child: ExpansionTile(

leading: const Icon(Icons.menu\_book),

title: Text(m.title),

subtitle: Text('${m.quizzes.length} quiz'),

children: m.quizzes.map((q) {

return FutureBuilder<int?>(

future: \_getUserBestScore(q.id),

builder: (context, s) {

final best =

(s.connectionState == ConnectionState.done)

? s.data

: null;

return ListTile(

leading:

const Icon(Icons.quiz, color: Colors.blue),

title: Text(q.title),

subtitle:

best != null ? Text('Meilleur: $best') : null,

trailing: const Icon(Icons.chevron\_right),

onTap: () => Navigator.push(

context,

MaterialPageRoute(

builder: (\_) => QuizPlayScreen(quiz: q),

),

),

);

},

);

}).toList(),

),

)),

if (user != null) \_buildHistory(user!.uid),

],

);

},

),

);

}

}

class \_ModuleSimple {

final String id;

final String title;

final List<QuizModel> quizzes;

\_ModuleSimple({required this.id, required this.title, required this.quizzes});

}

/// ================= QuizPlayScreen

class QuizPlayScreen extends StatefulWidget {

final QuizModel quiz;

const QuizPlayScreen({super.key, required this.quiz});

@override

State<QuizPlayScreen> createState() => \_QuizPlayScreenState();

}

class \_QuizPlayScreenState extends State<QuizPlayScreen> {

int currentIndex = 0;

late List<int> userAnswers; // -1 = pas répondu

bool finished = false;

bool saving = false;

final FirebaseFirestore \_firestore = FirebaseFirestore.instance;

final User? user = FirebaseAuth.instance.currentUser;

@override

void initState() {

super.initState();

userAnswers = List<int>.filled(widget.quiz.questions.length, -1);

}

bool get allAnswered => !userAnswers.any((ans) => ans == -1);

int \_computeScore() {

int s = 0;

for (var i = 0; i < widget.quiz.questions.length; i++) {

final q = widget.quiz.questions[i];

final ans = userAnswers[i];

if (ans != -1 && ans == q.correctIndex) s++;

}

return s;

}

String \_computeBadge(int score, int total) {

final thresholds = widget.quiz.badgeThresholds;

final percent = total == 0 ? 0 : (score \* 100) / total;

if (percent >= (thresholds['Or'] ?? 90)) return 'Or 🥇';

if (percent >= (thresholds['Argent'] ?? 70)) return 'Argent 🥈';

if (percent >= (thresholds['Bronze'] ?? 50)) return 'Bronze 🥉';

return 'Aucun';

}

Future<void> \_saveResult(int score, int total, String badge) async {

if (user == null) return;

final result = QuizResult(

id: widget.quiz.id,

moduleId: widget.quiz.moduleId,

quizId: widget.quiz.id,

score: score,

total: total,

bonnes: score,

fautes: total - score,

duree: 0,

percent: total == 0 ? 0 : ((score \* 100) / total).round(),

badge: badge,

date: DateTime.now(),

answers: userAnswers,

);

await \_firestore

.collection('users/${user!.uid}/quizHistory')

.doc(widget.quiz.id)

.set(result.toMap(), SetOptions(merge: true));

}

Future<void> \_finishQuiz() async {

setState(() => finished = true);

final total = widget.quiz.questions.length;

final score = \_computeScore();

final badge = \_computeBadge(score, total);

setState(() => saving = true);

try {

await \_saveResult(score, total, badge);

} catch (e) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text('Erreur sauvegarde: $e')),

);

} finally {

setState(() => saving = false);

}

}

void \_replay() {

setState(() {

userAnswers = List<int>.filled(widget.quiz.questions.length, -1);

currentIndex = 0;

finished = false;

});

}

@override

Widget build(BuildContext context) {

if (widget.quiz.questions.isEmpty) {

return Scaffold(

appBar: AppBar(title: Text(widget.quiz.title)),

body:

const Center(child: Text('Ce quiz ne contient pas de questions.')),

);

}

return Scaffold(

appBar: AppBar(title: Text(widget.quiz.title)),

body: Padding(

padding: const EdgeInsets.all(12),

child: finished

? \_buildResultSummary()

: Column(

children: [

Expanded(child: \_buildQuestionView()),

const SizedBox(height: 8),

Row(

children: [

Expanded(

child: ElevatedButton(

onPressed: currentIndex > 0

? () => setState(() => currentIndex--)

: null,

child: const Text('⬅️ Précédent'),

),

),

const SizedBox(width: 8),

Expanded(

child: (currentIndex < widget.quiz.questions.length - 1)

? ElevatedButton(

onPressed: userAnswers[currentIndex] != -1

? () => setState(() => currentIndex++)

: null,

child: const Text('Suivant ➡️'),

)

: ElevatedButton(

onPressed: allAnswered ? \_finishQuiz : null,

child: const Text('✅ Terminer'),

),

),

],

),

],

),

),

);

}

Widget \_buildQuestionView() {

final q = widget.quiz.questions[currentIndex];

final answer = userAnswers[currentIndex];

return SingleChildScrollView(

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text('Question ${currentIndex + 1} / ${widget.quiz.questions.length}',

style:

const TextStyle(fontSize: 16, fontWeight: FontWeight.bold)),

const SizedBox(height: 8),

LinearProgressIndicator(

value: (currentIndex + 1) / widget.quiz.questions.length,

),

const SizedBox(height: 16),

Text(q.question, style: const TextStyle(fontSize: 20)),

const SizedBox(height: 12),

...List.generate(q.options.length, (i) {

final isSelected = answer == i;

final disabled = userAnswers[currentIndex] != -1;

Color? color;

if (disabled) {

if (i == q.correctIndex)

color = Colors.green.shade200;

else if (isSelected)

color = Colors.red.shade200;

else

color = Colors.grey.shade100;

} else if (isSelected) {

color = Colors.blue.shade100;

}

return Card(

color: color,

child: ListTile(

title: Text(q.options[i]),

onTap: disabled

? null

: () => setState(() {

userAnswers[currentIndex] = i;

}),

),

);

}),

],

),

);

}

Widget \_buildResultSummary() {

final total = widget.quiz.questions.length;

final score = \_computeScore();

final badge = \_computeBadge(score, total);

return Column(

crossAxisAlignment: CrossAxisAlignment.stretch,

children: [

Card(

color: Colors.blue.shade50,

child: Padding(

padding: const EdgeInsets.all(12),

child: Column(

children: [

Text('Résultat', style: Theme.of(context).textTheme.titleLarge),

Text('Score : $score / $total'),

Text('Badge : $badge'),

],

),

),

),

ElevatedButton(onPressed: \_replay, child: const Text('Rejouer')),

OutlinedButton(

onPressed: () => Navigator.pop(context),

child: const Text('Fermer')),

],

);

}

}

/// ================= Relecture d’un quiz sauvegardé

class QuizReviewScreen extends StatelessWidget {

final QuizResult result;

const QuizReviewScreen({super.key, required this.result});

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text("Relecture ${result.quizId}")),

body: ListView.builder(

itemCount: result.answers.length,

itemBuilder: (context, i) {

return ListTile(

title: Text("Question ${i + 1}"),

subtitle: Text("Réponse donnée: ${result.answers[i]}"),

);

},

),

);

}

}  
  
// lib/screens/tabs/revision\_tab.dart

import 'package:flutter/material.dart';

import '../../services/firestore\_service.dart';

import '../../models/course\_model.dart';

import '../revision/module\_detail\_screen.dart';

class RevisionTab extends StatefulWidget {

  const RevisionTab({*super*.key});

  @override

  State<RevisionTab> *createState*() => \_RevisionTabState();

}

class \_RevisionTabState extends State<RevisionTab> {

  final TextEditingController \_searchController = TextEditingController();

  String \_searchQuery = "";

  final FirestoreService \_firestore = FirestoreService();

  List<Course> \_modules = [];

  bool \_loading = true;

  String? \_error;

  @override

  void *initState*() {

*super*.*initState*();

*\_loadModules*();

  }

  Future<void> *\_loadModules*() **async** {

*setState*(() {

      \_loading = true;

      \_error = null;

    });

**try** {

      final mods = **await** \_firestore.*getModules*();

*setState*(() {

        \_modules = mods;

        \_loading = false;

      });

    } **catch** (e) {

*setState*(() {

        \_error = "Erreur chargement modules : $*e*";

        \_loading = false;

      });

    }

  }

  @override

  Widget *build*(BuildContext context) {

    final filtered = \_modules.*where*((m) {

      final q = \_searchQuery.*trim*().*toLowerCase*();

      final title = m.title.*toLowerCase*();

      final desc = m.description.*toLowerCase*();

      final tags = m.tags.*join*(' ').*toLowerCase*();

**return** q.isEmpty ||

          title.*contains*(q) ||

          desc.*contains*(q) ||

          tags.*contains*(q);

    }).*toList*();

**return** Scaffold(

      appBar: AppBar(

        title: const Text("Révisions"),

        centerTitle: true,

        actions: [

          IconButton(

            icon: const Icon(Icons.refresh),

            tooltip: 'Rafraîchir',

            onPressed: \_loadModules,

          ),

        ],

      ),

      body: Column(

        children: [

          // Recherche

          Padding(

            padding: const EdgeInsets.*all*(8.0),

            child: TextField(

              controller: \_searchController,

              decoration: InputDecoration(

                hintText: "Rechercher un module, matière ou mot-clé...",

                prefixIcon: const Icon(Icons.search),

                suffixIcon: \_searchQuery.isNotEmpty

                    ? IconButton(

                        icon: const Icon(Icons.clear),

                        onPressed: () {

                          \_searchController.*clear*();

*setState*(() => \_searchQuery = "");

                        },

                      )

                    : null,

                border:

                    OutlineInputBorder(borderRadius: BorderRadius.*circular*(12)),

              ),

              onChanged: (val) => *setState*(() => \_searchQuery = val),

            ),

          ),

**if** (\_loading) const LinearProgressIndicator(),

**if** (\_error != null)

            Padding(

              padding: const EdgeInsets.*all*(12.0),

              child: Text(\_error!, style: const TextStyle(color: Colors.red)),

            ),

          Expanded(

            child: \_loading && \_modules.isEmpty

                ? const Center(child: CircularProgressIndicator())

                : filtered.isEmpty

                    ? const Center(child: Text('Aucun module trouvé'))

                    : ListView.*builder*(

                        itemCount: filtered.length,

                        itemBuilder: (context, index) {

                          final Course module = filtered[index];

                          final displayTitle = module.title.isNotEmpty

                              ? module.title

                              : (module.description.isNotEmpty

                                  ? module.description

                                  : module.id);

                          final fichesCount = module.fiches.length;

                          final videosCount = module.videos.length;

**return** Card(

                            margin: const EdgeInsets.*symmetric*(

                                horizontal: 12, vertical: 6),

                            shape: RoundedRectangleBorder(

                                borderRadius: BorderRadius.*circular*(12)),

                            child: ListTile(

                              leading: module.imageUrl != null &&

                                      module.imageUrl!.isNotEmpty

                                  ? ClipRRect(

                                      borderRadius: BorderRadius.*circular*(10),

                                      child: Image.*network*(module.imageUrl!,

                                          width: 52,

                                          height: 52,

                                          fit: BoxFit.cover),

                                    )

                                  : Container(

                                      width: 52,

                                      height: 52,

                                      decoration: BoxDecoration(

                                        color: Colors.blue.shade50,

                                        borderRadius: BorderRadius.*circular*(10),

                                      ),

                                      child: const Icon(Icons.book,

                                          size: 30, color: Colors.blue),

                                    ),

                              title: Text(displayTitle,

                                  style: const TextStyle(

                                      fontWeight: FontWeight.bold,

                                      fontSize: 16)),

                              subtitle: Text(

                                  "$*fichesCount* fiches • $*videosCount* vidéos",

                                  style: const TextStyle(color: Colors.grey)),

                              trailing:

                                  const Icon(Icons.arrow\_forward\_ios, size: 18),

                              onTap: () {

                                // Passer toutes les fiches et vidéos pour le détail

                                Navigator.*push*(

                                  context,

                                  MaterialPageRoute(

                                    builder: (\_) => ModuleDetailScreen(

                                      moduleData: {

                                        ...module.*toMap*(),

                                        'fiches': module.fiches

                                            .*map*((f) => f.*toMap*())

                                            .*toList*(),

                                        'videos': module.videos

                                            .*map*((v) => v.*toMap*())

                                            .*toList*(),

                                      },

                                    ),

                                  ),

                                );

                              },

                            ),

                          );

                        },

                      ),

          ),

        ],

      ),

    );

  }

}

// 📁 lib/screens/dashboard.dart

import 'dart:ui';

import 'package:flutter/material.dart';

import 'package:provider/provider.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

import 'package:reussirbts/provider/user\_provider.dart';

import 'package:reussirbts/utils/connectivity\_service.dart';

import 'package:reussirbts/widgets/network\_banner.dart';

import 'tabs/home\_tab.dart';

import 'tabs/quiz\_tab.dart';

import 'tabs/revision\_tab.dart';

import 'tabs/progress\_tab.dart';

import 'tabs/profile\_tab.dart';

import 'admin/admin\_panel.dart';

class Dashboard extends StatefulWidget {

  const Dashboard({*super*.key});

  @override

  State<Dashboard> *createState*() => \_DashboardState();

}

class \_DashboardState extends State<Dashboard> {

  int \_currentIndex = 0;

  bool \_isOnline = true;

  final List<Widget> \_tabs = const [

    HomeTab(),

    QuizTab(),

    RevisionTab(),

    ProgressTab(),

    ProfileTab(),

  ];

  @override

  void *initState*() {

*super*.*initState*();

*\_loadLastIndex*();

*\_listenConnectivity*();

  }

  Future<void> *\_loadLastIndex*() **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

*setState*(() {

      \_currentIndex = prefs.*getInt*('lastTabIndex') ?? 0;

    });

  }

  Future<void> *\_saveLastIndex*(int index) **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

**await** prefs.*setInt*('lastTabIndex', index);

  }

  void *\_listenConnectivity*() {

    ConnectivityService.connectivityStream.*listen*((status) {

*setState*(() {

        \_isOnline = status;

      });

    });

  }

  Future<bool> *\_onWillPop*() **async** {

**if** (\_currentIndex != 0) {

*setState*(() => \_currentIndex = 0);

**return** false;

    }

**return** true;

  }

  @override

  Widget *build*(BuildContext context) {

    final userProvider = Provider.*of*<UserProvider>(context);

**return** WillPopScope(

      onWillPop: \_onWillPop,

      child: Scaffold(

        extendBody: true,

        body: Column(

          children: [

**if** (!\_isOnline) const NetworkBanner(),

            Expanded(

              child: AnimatedSwitcher(

                duration: const Duration(milliseconds: 400),

                transitionBuilder: (child, animation) {

                  final offsetAnim = Tween<Offset>(

                    begin: const Offset(0.05, 0.05),

                    end: Offset.zero,

                  ).*animate*(animation);

**return** FadeTransition(

                    opacity: animation,

                    child: SlideTransition(position: offsetAnim, child: child),

                  );

                },

                child: \_tabs[\_currentIndex],

              ),

            ),

          ],

        ),

        bottomNavigationBar: \_GlassBottomNavBar(

          currentIndex: \_currentIndex,

          onTap: (i) {

*setState*(() => \_currentIndex = i);

*\_saveLastIndex*(i);

          },

        ),

        floatingActionButton: userProvider.user?.role == 'admin'

            ? FloatingActionButton.*extended*(

                backgroundColor: Colors.blueAccent,

                onPressed: () {

                  Navigator.*push*(

                    context,

                    MaterialPageRoute(builder: (\_) => const AdminPanel()),

                  );

                },

                icon: const Icon(Icons.admin\_panel\_settings),

                label: const Text("Admin"),

              )

            : null,

        floatingActionButtonLocation: FloatingActionButtonLocation.endDocked,

      ),

    );

  }

}

class \_GlassBottomNavBar extends StatelessWidget {

  final int currentIndex;

  final Function(int) onTap;

  const \_GlassBottomNavBar({

    required *this*.currentIndex,

    required *this*.onTap,

*super*.key,

  });

  @override

  Widget *build*(BuildContext context) {

    final items = const [

      {'icon': Icons.home, 'label': 'Accueil'},

      {'icon': Icons.quiz, 'label': 'Quiz'},

      {'icon': Icons.book, 'label': 'Révision'},

      {'icon': Icons.bar\_chart, 'label': 'Progression'},

      {'icon': Icons.person, 'label': 'Profil'},

    ];

**return** Container(

      decoration: BoxDecoration(

        borderRadius: const BorderRadius.*only*(

          topLeft: Radius.*circular*(24),

          topRight: Radius.*circular*(24),

        ),

        boxShadow: [

          BoxShadow(

            color: Colors.black.*withOpacity*(0.1),

            blurRadius: 12,

            offset: const Offset(0, -2),

          )

        ],

      ),

      child: ClipRRect(

        borderRadius: const BorderRadius.*only*(

          topLeft: Radius.*circular*(24),

          topRight: Radius.*circular*(24),

        ),

        child: BackdropFilter(

          filter: ImageFilter.*blur*(sigmaX: 20, sigmaY: 20),

          child: BottomNavigationBar(

            type: BottomNavigationBarType.fixed,

            backgroundColor: Colors.white.*withOpacity*(0.75),

            elevation: 0,

            currentIndex: currentIndex,

            selectedItemColor: Colors.blueAccent, // ✅ lisible sur fond clair

            unselectedItemColor: Colors.black54,

            showUnselectedLabels: true,

            onTap: onTap,

            items: [

**for** (int i = 0; i < items.length; i++)

                BottomNavigationBarItem(

                  icon: \_AnimatedNavIcon(

                    icon: items[i]['icon'] as IconData,

                    isActive: currentIndex == i,

                  ),

                  label: items[i]['label'] as String,

                )

            ],

          ),

        ),

      ),

    );

  }

}

class \_AnimatedNavIcon extends StatefulWidget {

  final IconData icon;

  final bool isActive;

  const \_AnimatedNavIcon({

    required *this*.icon,

    required *this*.isActive,

*super*.key,

  });

  @override

  State<\_AnimatedNavIcon> *createState*() => \_AnimatedNavIconState();

}

class \_AnimatedNavIconState extends State<\_AnimatedNavIcon>

    with SingleTickerProviderStateMixin {

  late AnimationController \_controller;

  @override

  void *initState*() {

*super*.*initState*();

    \_controller = AnimationController(

        vsync: *this*, duration: const Duration(milliseconds: 400));

**if** (widget.isActive) {

      \_controller.*forward*();

    }

  }

  @override

  void *didUpdateWidget*(covariant \_AnimatedNavIcon oldWidget) {

*super*.*didUpdateWidget*(oldWidget);

**if** (widget.isActive && !oldWidget.isActive) {

      \_controller.*forward*(from: 0.0);

    } **else** **if** (!widget.isActive && oldWidget.isActive) {

      \_controller.*reverse*();

    }

  }

  @override

  Widget *build*(BuildContext context) {

**return** ScaleTransition(

      scale: Tween<double>(begin: 1.0, end: 1.3).*animate*(

        CurvedAnimation(parent: \_controller, curve: Curves.elasticOut),

      ),

      child: RotationTransition(

        turns: Tween<double>(begin: 0.0, end: 0.1).*animate*(

          CurvedAnimation(parent: \_controller, curve: Curves.easeOutBack),

        ),

        child: Container(

          padding: const EdgeInsets.*all*(8),

          decoration: BoxDecoration(

            color: widget.isActive

                ? Colors.blueAccent.*withOpacity*(0.85)

                : Colors.transparent,

            borderRadius: BorderRadius.*circular*(16),

          ),

          child: Icon(

            widget.icon,

            color: widget.isActive ? Colors.white : Colors.black87,

          ),

        ),

      ),

    );

  }

  @override

  void *dispose*() {

    \_controller.*dispose*();

*super*.*dispose*();

  }

}

// lib/services/firestore\_service.dart

import 'dart:convert';

import 'dart:io';

import 'package:cloud\_firestore/cloud\_firestore.dart';

// import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:flutter/foundation.dart' show kIsWeb;

import 'package:path\_provider/path\_provider.dart';

import '../models/user\_model.dart';

import '../models/course\_model.dart';

import '../models/fiche\_model.dart';

import '../models/video\_model.dart';

// import '../models/quiz\_model.dart';

// import '../models/quiz\_result\_model.dart';

// import '../models/question\_model.dart';

class FirestoreService {

  final FirebaseFirestore \_db = FirebaseFirestore.instance;

  // ===================== HELPERS =====================

  int *\_toInt*(dynamic v, [int fallback = 0]) {

**if** (v == null) **return** fallback;

**if** (v is int) **return** v;

**if** (v is double) **return** v.*toInt*();

**if** (v is num) **return** v.*toInt*();

**if** (v is String) **return** int.*tryParse*(v) ?? fallback;

**return** fallback;

  }

  int *\_extractOrderFromId*(String id) {

**if** (id.isEmpty) **return** 0;

    final m = RegExp(r'\d+').*firstMatch*(id);

**if** (m != null) **return** int.*tryParse*(m.*group*(0)!) ?? 0;

**return** 0;

  }

  int *\_orderFromDocData*(Map<String, dynamic> data, String id) {

**if** (data.*containsKey*('order')) {

**return** *\_toInt*(data['order'], *\_extractOrderFromId*(id));

    }

**return** *\_extractOrderFromId*(id);

  }

  // ===================== USERS =====================

  Future<AppUser?> *getUser*(String uid) **async** {

**try** {

      final doc = **await** \_db.*collection*('users').*doc*(uid).*get*();

**if** (doc.exists && doc.*data*() != null) {

**return** AppUser.*fromMap*(doc.*data*()!);

      }

    } **catch** (e) {

*print*('FirestoreService.getUser firestore error: $*e*');

    }

**if** (!kIsWeb) {

      final users = **await** *\_getAllUsersLocal*();

**try** {

**return** users.*firstWhere*((u) => u.uid == uid);

      } **catch** (\_) {

**return** null;

      }

    }

**return** null;

  }

  Future<void> *saveUser*(AppUser user) **async** {

    // Sauvegarde locale

**if** (!kIsWeb) {

**try** {

        final dir = **await** *getApplicationDocumentsDirectory*();

        final file = File('${*dir*.*path*}/users.json');

        List<AppUser> users = **await** *\_getAllUsersLocal*();

        users.*removeWhere*((u) => u.uid == user.uid);

        users.*add*(user);

**await** file

            .*writeAsString*(*jsonEncode*(users.*map*((u) => u.*toMap*()).*toList*()));

      } **catch** (e) {

*print*('FirestoreService.saveUser local error: $*e*');

      }

    }

    // Sauvegarde Firestore

**try** {

**await** \_db.*collection*('users').*doc*(user.uid).*set*(user.*toMap*());

    } **catch** (e) {

*print*('FirestoreService.saveUser firestore error: $*e*');

    }

  }

  Future<void> *updateUser*(AppUser user) **async** => **await** *saveUser*(user);

  Future<void> *deleteUserData*(String uid) **async** {

**try** {

      // Supprimer les résultats de quiz liés à l'utilisateur

      final resultsSnap = **await** \_db

          .*collection*('users')

          .*doc*(uid)

          .*collection*('quizResults')

          .*get*();

      final batch = \_db.*batch*();

**for** (final doc **in** resultsSnap.docs) {

        batch.*delete*(doc.reference);

      }

      // Supprimer l'utilisateur

      batch.*delete*(\_db.*collection*('users').*doc*(uid));

**await** batch.*commit*();

    } **catch** (e) {

*print*('FirestoreService.deleteUserData error: $*e*');

**rethrow**;

    }

  }

  Future<List<AppUser>> *\_getAllUsersLocal*() **async** {

**if** (kIsWeb) **return** [];

**try** {

      final dir = **await** *getApplicationDocumentsDirectory*();

      final file = File('${*dir*.*path*}/users.json');

**if** (!**await** file.*exists*()) **return** [];

      final jsonStr = **await** file.*readAsString*();

      final List<dynamic> jsonList = *jsonDecode*(jsonStr);

**return** jsonList.*map*((e) => AppUser.*fromMap*(e)).*toList*();

    } **catch** (e) {

*print*('FirestoreService.\_getAllUsersLocal error: $*e*');

**return** [];

    }

  }

  Future<bool> *isAdmin*(String uid) **async** {

**try** {

      final doc = **await** \_db.*collection*('users').*doc*(uid).*get*();

**return** doc.exists && (doc.*data*()?['role'] as String?) == 'admin';

    } **catch** (e) {

*print*('FirestoreService.isAdmin error: $*e*');

**return** false;

    }

  }

  // ===================== MODULES =====================

  Future<List<Course>> *getModules*({bool forceRefresh = false}) **async** {

    List<Course> cachedModules = [];

    // Lire depuis le cache local

**if** (!kIsWeb) {

**try** {

        final dir = **await** *getApplicationDocumentsDirectory*();

        final file = File('${*dir*.*path*}/modules.json');

**if** (**await** file.*exists*()) {

          final jsonStr = **await** file.*readAsString*();

          final List<dynamic> jsonList = *jsonDecode*(jsonStr);

          cachedModules =

              jsonList.*map*((e) => Course.*fromMap*(e, id: e['id'])).*toList*();

**if** (!forceRefresh) **return** cachedModules;

        }

      } **catch** (e) {

*print*('FirestoreService.getModules cache read error: $*e*');

      }

    }

    // Lire depuis Firestore

**try** {

      final snap = **await** \_db.*collection*('modules').*get*();

      List<Course> modules = [];

**for** (final doc **in** snap.docs) {

        final data = Map<String, dynamic>.*from*(doc.*data*());

        final moduleId = doc.id;

        // Fiches

        final fiches = **await** *getFichesForModule*(moduleId);

        // Videos

        final videos = **await** *getVideosForModule*(moduleId);

        // // Quizzes

        // final quizzes = await getQuizzesForModule(moduleId);

        // Build Course

        final moduleMap = Map<String, dynamic>.*from*(data);

        moduleMap['fiches'] =

            fiches.*map*((f) => f.*toMap*()..['id'] = f.id).*toList*();

        moduleMap['videos'] =

            videos.*map*((v) => v.*toMap*()..['id'] = v.id).*toList*();

        // moduleMap['countQuizzes'] = quizzes.length;

        modules.*add*(Course.*fromMap*(moduleMap, id: moduleId));

      }

      // Écrire le cache local

**if** (!kIsWeb) {

**try** {

          final dir = **await** *getApplicationDocumentsDirectory*();

          final file = File('${*dir*.*path*}/modules.json');

**await** file.*writeAsString*(*jsonEncode*(

              modules.*map*((m) => m.*toMap*()..['id'] = m.id).*toList*()));

        } **catch** (e) {

*print*('FirestoreService.getModules cache write error: $*e*');

        }

      }

**return** modules;

    } **catch** (e) {

*print*('FirestoreService.getModules firestore error: $*e*');

**return** cachedModules;

    }

  }

  // ===================== MODULE CONTENT =====================

  Future<List<Fiche>> *getFichesForModule*(String moduleId) **async** {

**try** {

      final snap = **await** \_db

          .*collection*('modules')

          .*doc*(moduleId)

          .*collection*('fichesSynthese')

          .*get*();

      final pairs = snap.docs.*map*((d) {

        final m = Map<String, dynamic>.*from*(d.*data*());

        final order = *\_orderFromDocData*(m, d.id);

        final fiche = Fiche.*fromMap*(m, id: d.id);

**return** {'order': order, 'fiche': fiche};

      }).*toList*();

      pairs.*sort*((a, b) => (a['order'] as int).*compareTo*(b['order'] as int));

**return** pairs.*map*((p) => p['fiche'] as Fiche).*toList*();

    } **catch** (e) {

*print*("FirestoreService.getFichesForModule error: $*e*");

**return** [];

    }

  }

  Future<List<VideoItem>> *getVideosForModule*(String moduleId) **async** {

**try** {

      final snap = **await** \_db

          .*collection*('modules')

          .*doc*(moduleId)

          .*collection*('videos')

          .*get*();

      final pairs = snap.docs.*map*((d) {

        final m = Map<String, dynamic>.*from*(d.*data*());

        final order = *\_orderFromDocData*(m, d.id);

        final video = VideoItem.*fromMap*(m, id: d.id);

**return** {'order': order, 'video': video};

      }).*toList*();

      pairs.*sort*((a, b) => (a['order'] as int).*compareTo*(b['order'] as int));

**return** pairs.*map*((p) => p['video'] as VideoItem).*toList*();

    } **catch** (e) {

*print*("FirestoreService.getVideosForModule error: $*e*");

**return** [];

    }

  }

  // // ===================== QUIZZES =====================

  // Future<List<QuestionModel>> getQuestionsForQuiz(

  //     String moduleId, String quizId) async {

  //   try {

  //     final snap = await \_db

  //         .collection('modules')

  //         .doc(moduleId)

  //         .collection('quizzes')

  //         .doc(quizId)

  //         .collection('questions')

  //         .get();

  //     return snap.docs

  //         .map(

  //             (d) => QuestionModel.fromMap(Map<String, dynamic>.from(d.data())))

  //         .toList();

  //   } catch (e) {

  //     print("FirestoreService.getQuestionsForQuiz error: $e");

  //     return [];

  //   }

  // }

  // Future<List<QuizModel>> getQuizzesForModule(String moduleId) async {

  //   try {

  //     final snapshot = await \_db

  //         .collection('modules')

  //         .doc(moduleId)

  //         .collection('quizzes')

  //         .get();

  //     List<QuizModel> quizzes = [];

  //     for (final doc in snapshot.docs) {

  //       final data = doc.data();

  //       List<QuestionModel> questions = [];

  //       if (data['questions'] != null && data['questions'] is List) {

  //         final List<dynamic> questionsList = data['questions'];

  //         for (var q in questionsList) {

  //           questions.add(QuestionModel.fromMap(Map<String, dynamic>.from(q)));

  //         }

  //       } else {

  //         questions = await getQuestionsForQuiz(moduleId, doc.id);

  //       }

  //       // 🔹 Sécurité : éviter null ou crash

  //       questions = questions.where((q) => q.question.isNotEmpty).toList();

  //       quizzes.add(QuizModel(

  //         id: doc.id,

  //         moduleId: moduleId,

  //         title: (data['title'] ?? doc.id).toString(),

  //         questions: questions,

  //       ));

  //     }

  //     return quizzes;

  //   } catch (e) {

  //     print("FirestoreService.getQuizzesForModule error: $e");

  //     return [];

  //   }

  // }

  // /// 🔹 Mock pour tester sans Firestore

  // Future<List<QuizModel>> getQuizzesMock() async {

  //   const moduleId = "mockModule";

  //   return [

  //     QuizModel(

  //       id: '1',

  //       moduleId: moduleId,

  //       title: 'Quiz Test',

  //       questions: [

  //         QuestionModel(

  //           question: 'Exemple question ?',

  //           options: ['A', 'B', 'C', 'D'],

  //           correctIndex: 0,

  //         ),

  //         QuestionModel(

  //           question: 'Le BTS est un diplôme de… ?',

  //           options: ['Licence', 'Brevet', 'Technicien Supérieur', 'Master'],

  //           correctIndex: 2,

  //         ),

  //       ],

  //     ),

  //   ];

  // }

  // // ===================== QUIZ RESULTS =====================

  // Future<List<QuizResult>> getQuizResults(String uid, String moduleId) async {

  //   try {

  //     final snap = await \_db

  //         .collection('users')

  //         .doc(uid)

  //         .collection('quizResults')

  //         .where('moduleId', isEqualTo: moduleId)

  //         .get();

  //     return snap.docs.map((d) => QuizResult.fromFirestore(d)).toList();

  //   } catch (e) {

  //     print('Erreur getQuizResults: $e');

  //     return [];

  //   }

  // }

  // Future<void> saveResult({

  //   String? etudiantId,

  //   required String moduleId,

  //   required String quizId,

  //   required int score,

  //   required int total,

  //   required int bonnes,

  //   required int fautes,

  //   required int duree,

  //   required DateTime date,

  //   List<int>? answers,

  // }) async {

  //   final uid =

  //       etudiantId ?? FirebaseAuth.instance.currentUser?.uid ?? "UNKNOWN\_USER";

  //   try {

  //     final percent = total > 0 ? ((score / total) \* 100).round() : 0;

  //     // Déterminer le badge

  //     String badge = 'bronze';

  //     if (percent >= 90) {

  //       badge = 'gold';

  //     } else if (percent >= 75) {

  //       badge = 'silver';

  //     }

  //     final data = {

  //       "etudiantId": uid,

  //       "moduleId": moduleId,

  //       "quizId": quizId,

  //       "score": score,

  //       "total": total,

  //       "bonnes": bonnes,

  //       "fautes": fautes,

  //       "duree": duree,

  //       "durationSeconds": duree,

  //       "percent": percent,

  //       "badge": badge,

  //       "answers": answers ?? [],

  //       "date": date.toIso8601String(),

  //     };

  //     await \_db

  //         .collection("users")

  //         .doc(uid)

  //         .collection("quizResults")

  //         .add(data);

  //   } catch (e) {

  //     print("FirestoreService.saveResult error: $e");

  //   }

  // }

  // // ===================== BADGES =====================

  // Future<void> unlockBadge(String badgeId, String description) async {

  //   final uid = FirebaseAuth.instance.currentUser?.uid;

  //   if (uid == null || uid.isEmpty) return;

  //   final badgeRef =

  //       \_db.collection("users").doc(uid).collection("badges").doc(badgeId);

  //   final snap = await badgeRef.get();

  //   if (!snap.exists) {

  //     await badgeRef.set({

  //       "id": badgeId,

  //       "description": description,

  //       "dateUnlocked": DateTime.now().toIso8601String(),

  //     });

  //   }

  // }

  // ===================== CRUD (Modules / Fiches / Videos / Quizzes) =====================

  // ---- MODULES ----

  Future<String> *addModule*(Course course) **async** {

    final ref = \_db.*collection*('modules').*doc*(course.id);

**await** ref.*set*(course.*toMap*());

**return** ref.id;

  }

  Future<void> *updateModule*(Course course) **async** {

**await** \_db.*collection*('modules').*doc*(course.id).*update*(course.*toMap*());

  }

  Future<void> *deleteModule*(String moduleId) **async** {

**await** \_db.*collection*('modules').*doc*(moduleId).*delete*();

  }

  // ---- FICHES ----

  Future<void> *addFiche*(String moduleId, Fiche fiche) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('fichesSynthese')

        .*doc*(fiche.id)

        .*set*(fiche.*toMap*());

  }

  Future<void> *updateFiche*(String moduleId, Fiche fiche) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('fichesSynthese')

        .*doc*(fiche.id)

        .*update*(fiche.*toMap*());

  }

  Future<void> *deleteFiche*(String moduleId, String ficheId) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('fichesSynthese')

        .*doc*(ficheId)

        .*delete*();

  }

  // ---- VIDEOS ----

  Future<void> *addVideo*(String moduleId, VideoItem video) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('videos')

        .*doc*(video.id)

        .*set*(video.*toMap*());

  }

  Future<void> *updateVideo*(String moduleId, VideoItem video) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('videos')

        .*doc*(video.id)

        .*update*(video.*toMap*());

  }

  Future<void> *deleteVideo*(String moduleId, String videoId) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('videos')

        .*doc*(videoId)

        .*delete*();

  }

  // // ---- QUIZZES ----

  // Future<void> addQuiz(String moduleId, QuizModel quiz) async {

  //   await \_db

  //       .collection('modules')

  //       .doc(moduleId)

  //       .collection('quizzes')

  //       .doc(quiz.id)

  //       .set(quiz.toMap());

  // }

  // Future<void> updateQuiz(String moduleId, QuizModel quiz) async {

  //   await \_db

  //       .collection('modules')

  //       .doc(moduleId)

  //       .collection('quizzes')

  //       .doc(quiz.id)

  //       .update(quiz.toMap());

  // }

  // Future<void> deleteQuiz(String moduleId, String quizId) async {

  //   await \_db

  //       .collection('modules')

  //       .doc(moduleId)

  //       .collection('quizzes')

  //       .doc(quizId)

  //       .delete();

  // }

}

// lib/services/firestore\_service.dart

import 'dart:convert';

import 'dart:io';

import 'package:cloud\_firestore/cloud\_firestore.dart';

// import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:flutter/foundation.dart' show kIsWeb;

import 'package:path\_provider/path\_provider.dart';

import '../models/user\_model.dart';

import '../models/course\_model.dart';

import '../models/fiche\_model.dart';

import '../models/video\_model.dart';

// import '../models/quiz\_model.dart';

// import '../models/quiz\_result\_model.dart';

// import '../models/question\_model.dart';

class FirestoreService {

  final FirebaseFirestore \_db = FirebaseFirestore.instance;

  // ===================== HELPERS =====================

  int *\_toInt*(dynamic v, [int fallback = 0]) {

**if** (v == null) **return** fallback;

**if** (v is int) **return** v;

**if** (v is double) **return** v.*toInt*();

**if** (v is num) **return** v.*toInt*();

**if** (v is String) **return** int.*tryParse*(v) ?? fallback;

**return** fallback;

  }

  int *\_extractOrderFromId*(String id) {

**if** (id.isEmpty) **return** 0;

    final m = RegExp(r'\d+').*firstMatch*(id);

**if** (m != null) **return** int.*tryParse*(m.*group*(0)!) ?? 0;

**return** 0;

  }

  int *\_orderFromDocData*(Map<String, dynamic> data, String id) {

**if** (data.*containsKey*('order')) {

**return** *\_toInt*(data['order'], *\_extractOrderFromId*(id));

    }

**return** *\_extractOrderFromId*(id);

  }

  // ===================== USERS =====================

  Future<AppUser?> *getUser*(String uid) **async** {

**try** {

      final doc = **await** \_db.*collection*('users').*doc*(uid).*get*();

**if** (doc.exists && doc.*data*() != null) {

**return** AppUser.*fromMap*(doc.*data*()!);

      }

    } **catch** (e) {

*print*('FirestoreService.getUser firestore error: $*e*');

    }

**if** (!kIsWeb) {

      final users = **await** *\_getAllUsersLocal*();

**try** {

**return** users.*firstWhere*((u) => u.uid == uid);

      } **catch** (\_) {

**return** null;

      }

    }

**return** null;

  }

  Future<void> *saveUser*(AppUser user) **async** {

    // Sauvegarde locale

**if** (!kIsWeb) {

**try** {

        final dir = **await** *getApplicationDocumentsDirectory*();

        final file = File('${*dir*.*path*}/users.json');

        List<AppUser> users = **await** *\_getAllUsersLocal*();

        users.*removeWhere*((u) => u.uid == user.uid);

        users.*add*(user);

**await** file

            .*writeAsString*(*jsonEncode*(users.*map*((u) => u.*toMap*()).*toList*()));

      } **catch** (e) {

*print*('FirestoreService.saveUser local error: $*e*');

      }

    }

    // Sauvegarde Firestore

**try** {

**await** \_db.*collection*('users').*doc*(user.uid).*set*(user.*toMap*());

    } **catch** (e) {

*print*('FirestoreService.saveUser firestore error: $*e*');

    }

  }

  Future<void> *updateUser*(AppUser user) **async** => **await** *saveUser*(user);

  Future<void> *deleteUserData*(String uid) **async** {

**try** {

      // Supprimer les résultats de quiz liés à l'utilisateur

      final resultsSnap = **await** \_db

          .*collection*('users')

          .*doc*(uid)

          .*collection*('quizResults')

          .*get*();

      final batch = \_db.*batch*();

**for** (final doc **in** resultsSnap.docs) {

        batch.*delete*(doc.reference);

      }

      // Supprimer l'utilisateur

      batch.*delete*(\_db.*collection*('users').*doc*(uid));

**await** batch.*commit*();

    } **catch** (e) {

*print*('FirestoreService.deleteUserData error: $*e*');

**rethrow**;

    }

  }

  Future<List<AppUser>> *\_getAllUsersLocal*() **async** {

**if** (kIsWeb) **return** [];

**try** {

      final dir = **await** *getApplicationDocumentsDirectory*();

      final file = File('${*dir*.*path*}/users.json');

**if** (!**await** file.*exists*()) **return** [];

      final jsonStr = **await** file.*readAsString*();

      final List<dynamic> jsonList = *jsonDecode*(jsonStr);

**return** jsonList.*map*((e) => AppUser.*fromMap*(e)).*toList*();

    } **catch** (e) {

*print*('FirestoreService.\_getAllUsersLocal error: $*e*');

**return** [];

    }

  }

  Future<bool> *isAdmin*(String uid) **async** {

**try** {

      final doc = **await** \_db.*collection*('users').*doc*(uid).*get*();

**return** doc.exists && (doc.*data*()?['role'] as String?) == 'admin';

    } **catch** (e) {

*print*('FirestoreService.isAdmin error: $*e*');

**return** false;

    }

  }

  // ===================== MODULES =====================

  Future<List<Course>> *getModules*({bool forceRefresh = false}) **async** {

    List<Course> cachedModules = [];

    // Lire depuis le cache local

**if** (!kIsWeb) {

**try** {

        final dir = **await** *getApplicationDocumentsDirectory*();

        final file = File('${*dir*.*path*}/modules.json');

**if** (**await** file.*exists*()) {

          final jsonStr = **await** file.*readAsString*();

          final List<dynamic> jsonList = *jsonDecode*(jsonStr);

          cachedModules =

              jsonList.*map*((e) => Course.*fromMap*(e, id: e['id'])).*toList*();

**if** (!forceRefresh) **return** cachedModules;

        }

      } **catch** (e) {

*print*('FirestoreService.getModules cache read error: $*e*');

      }

    }

    // Lire depuis Firestore

**try** {

      final snap = **await** \_db.*collection*('modules').*get*();

      List<Course> modules = [];

**for** (final doc **in** snap.docs) {

        final data = Map<String, dynamic>.*from*(doc.*data*());

        final moduleId = doc.id;

        // Fiches

        final fiches = **await** *getFichesForModule*(moduleId);

        // Videos

        final videos = **await** *getVideosForModule*(moduleId);

        // // Quizzes

        // final quizzes = await getQuizzesForModule(moduleId);

        // Build Course

        final moduleMap = Map<String, dynamic>.*from*(data);

        moduleMap['fiches'] =

            fiches.*map*((f) => f.*toMap*()..['id'] = f.id).*toList*();

        moduleMap['videos'] =

            videos.*map*((v) => v.*toMap*()..['id'] = v.id).*toList*();

        // moduleMap['countQuizzes'] = quizzes.length;

        modules.*add*(Course.*fromMap*(moduleMap, id: moduleId));

      }

      // Écrire le cache local

**if** (!kIsWeb) {

**try** {

          final dir = **await** *getApplicationDocumentsDirectory*();

          final file = File('${*dir*.*path*}/modules.json');

**await** file.*writeAsString*(*jsonEncode*(

              modules.*map*((m) => m.*toMap*()..['id'] = m.id).*toList*()));

        } **catch** (e) {

*print*('FirestoreService.getModules cache write error: $*e*');

        }

      }

**return** modules;

    } **catch** (e) {

*print*('FirestoreService.getModules firestore error: $*e*');

**return** cachedModules;

    }

  }

  // ===================== MODULE CONTENT =====================

  Future<List<Fiche>> *getFichesForModule*(String moduleId) **async** {

**try** {

      final snap = **await** \_db

          .*collection*('modules')

          .*doc*(moduleId)

          .*collection*('fichesSynthese')

          .*get*();

      final pairs = snap.docs.*map*((d) {

        final m = Map<String, dynamic>.*from*(d.*data*());

        final order = *\_orderFromDocData*(m, d.id);

        final fiche = Fiche.*fromMap*(m, id: d.id);

**return** {'order': order, 'fiche': fiche};

      }).*toList*();

      pairs.*sort*((a, b) => (a['order'] as int).*compareTo*(b['order'] as int));

**return** pairs.*map*((p) => p['fiche'] as Fiche).*toList*();

    } **catch** (e) {

*print*("FirestoreService.getFichesForModule error: $*e*");

**return** [];

    }

  }

  Future<List<VideoItem>> *getVideosForModule*(String moduleId) **async** {

**try** {

      final snap = **await** \_db

          .*collection*('modules')

          .*doc*(moduleId)

          .*collection*('videos')

          .*get*();

      final pairs = snap.docs.*map*((d) {

        final m = Map<String, dynamic>.*from*(d.*data*());

        final order = *\_orderFromDocData*(m, d.id);

        final video = VideoItem.*fromMap*(m, id: d.id);

**return** {'order': order, 'video': video};

      }).*toList*();

      pairs.*sort*((a, b) => (a['order'] as int).*compareTo*(b['order'] as int));

**return** pairs.*map*((p) => p['video'] as VideoItem).*toList*();

    } **catch** (e) {

*print*("FirestoreService.getVideosForModule error: $*e*");

**return** [];

    }

  }

  // // ===================== QUIZZES =====================

  // Future<List<QuestionModel>> getQuestionsForQuiz(

  //     String moduleId, String quizId) async {

  //   try {

  //     final snap = await \_db

  //         .collection('modules')

  //         .doc(moduleId)

  //         .collection('quizzes')

  //         .doc(quizId)

  //         .collection('questions')

  //         .get();

  //     return snap.docs

  //         .map(

  //             (d) => QuestionModel.fromMap(Map<String, dynamic>.from(d.data())))

  //         .toList();

  //   } catch (e) {

  //     print("FirestoreService.getQuestionsForQuiz error: $e");

  //     return [];

  //   }

  // }

  // Future<List<QuizModel>> getQuizzesForModule(String moduleId) async {

  //   try {

  //     final snapshot = await \_db

  //         .collection('modules')

  //         .doc(moduleId)

  //         .collection('quizzes')

  //         .get();

  //     List<QuizModel> quizzes = [];

  //     for (final doc in snapshot.docs) {

  //       final data = doc.data();

  //       List<QuestionModel> questions = [];

  //       if (data['questions'] != null && data['questions'] is List) {

  //         final List<dynamic> questionsList = data['questions'];

  //         for (var q in questionsList) {

  //           questions.add(QuestionModel.fromMap(Map<String, dynamic>.from(q)));

  //         }

  //       } else {

  //         questions = await getQuestionsForQuiz(moduleId, doc.id);

  //       }

  //       // 🔹 Sécurité : éviter null ou crash

  //       questions = questions.where((q) => q.question.isNotEmpty).toList();

  //       quizzes.add(QuizModel(

  //         id: doc.id,

  //         moduleId: moduleId,

  //         title: (data['title'] ?? doc.id).toString(),

  //         questions: questions,

  //       ));

  //     }

  //     return quizzes;

  //   } catch (e) {

  //     print("FirestoreService.getQuizzesForModule error: $e");

  //     return [];

  //   }

  // }

  // /// 🔹 Mock pour tester sans Firestore

  // Future<List<QuizModel>> getQuizzesMock() async {

  //   const moduleId = "mockModule";

  //   return [

  //     QuizModel(

  //       id: '1',

  //       moduleId: moduleId,

  //       title: 'Quiz Test',

  //       questions: [

  //         QuestionModel(

  //           question: 'Exemple question ?',

  //           options: ['A', 'B', 'C', 'D'],

  //           correctIndex: 0,

  //         ),

  //         QuestionModel(

  //           question: 'Le BTS est un diplôme de… ?',

  //           options: ['Licence', 'Brevet', 'Technicien Supérieur', 'Master'],

  //           correctIndex: 2,

  //         ),

  //       ],

  //     ),

  //   ];

  // }

  // // ===================== QUIZ RESULTS =====================

  // Future<List<QuizResult>> getQuizResults(String uid, String moduleId) async {

  //   try {

  //     final snap = await \_db

  //         .collection('users')

  //         .doc(uid)

  //         .collection('quizResults')

  //         .where('moduleId', isEqualTo: moduleId)

  //         .get();

  //     return snap.docs.map((d) => QuizResult.fromFirestore(d)).toList();

  //   } catch (e) {

  //     print('Erreur getQuizResults: $e');

  //     return [];

  //   }

  // }

  // Future<void> saveResult({

  //   String? etudiantId,

  //   required String moduleId,

  //   required String quizId,

  //   required int score,

  //   required int total,

  //   required int bonnes,

  //   required int fautes,

  //   required int duree,

  //   required DateTime date,

  //   List<int>? answers,

  // }) async {

  //   final uid =

  //       etudiantId ?? FirebaseAuth.instance.currentUser?.uid ?? "UNKNOWN\_USER";

  //   try {

  //     final percent = total > 0 ? ((score / total) \* 100).round() : 0;

  //     // Déterminer le badge

  //     String badge = 'bronze';

  //     if (percent >= 90) {

  //       badge = 'gold';

  //     } else if (percent >= 75) {

  //       badge = 'silver';

  //     }

  //     final data = {

  //       "etudiantId": uid,

  //       "moduleId": moduleId,

  //       "quizId": quizId,

  //       "score": score,

  //       "total": total,

  //       "bonnes": bonnes,

  //       "fautes": fautes,

  //       "duree": duree,

  //       "durationSeconds": duree,

  //       "percent": percent,

  //       "badge": badge,

  //       "answers": answers ?? [],

  //       "date": date.toIso8601String(),

  //     };

  //     await \_db

  //         .collection("users")

  //         .doc(uid)

  //         .collection("quizResults")

  //         .add(data);

  //   } catch (e) {

  //     print("FirestoreService.saveResult error: $e");

  //   }

  // }

  // // ===================== BADGES =====================

  // Future<void> unlockBadge(String badgeId, String description) async {

  //   final uid = FirebaseAuth.instance.currentUser?.uid;

  //   if (uid == null || uid.isEmpty) return;

  //   final badgeRef =

  //       \_db.collection("users").doc(uid).collection("badges").doc(badgeId);

  //   final snap = await badgeRef.get();

  //   if (!snap.exists) {

  //     await badgeRef.set({

  //       "id": badgeId,

  //       "description": description,

  //       "dateUnlocked": DateTime.now().toIso8601String(),

  //     });

  //   }

  // }

  // ===================== CRUD (Modules / Fiches / Videos / Quizzes) =====================

  // ---- MODULES ----

  Future<String> *addModule*(Course course) **async** {

    final ref = \_db.*collection*('modules').*doc*(course.id);

**await** ref.*set*(course.*toMap*());

**return** ref.id;

  }

  Future<void> *updateModule*(Course course) **async** {

**await** \_db.*collection*('modules').*doc*(course.id).*update*(course.*toMap*());

  }

  Future<void> *deleteModule*(String moduleId) **async** {

**await** \_db.*collection*('modules').*doc*(moduleId).*delete*();

  }

  // ---- FICHES ----

  Future<void> *addFiche*(String moduleId, Fiche fiche) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('fichesSynthese')

        .*doc*(fiche.id)

        .*set*(fiche.*toMap*());

  }

  Future<void> *updateFiche*(String moduleId, Fiche fiche) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('fichesSynthese')

        .*doc*(fiche.id)

        .*update*(fiche.*toMap*());

  }

  Future<void> *deleteFiche*(String moduleId, String ficheId) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('fichesSynthese')

        .*doc*(ficheId)

        .*delete*();

  }

  // ---- VIDEOS ----

  Future<void> *addVideo*(String moduleId, VideoItem video) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('videos')

        .*doc*(video.id)

        .*set*(video.*toMap*());

  }

  Future<void> *updateVideo*(String moduleId, VideoItem video) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('videos')

        .*doc*(video.id)

        .*update*(video.*toMap*());

  }

  Future<void> *deleteVideo*(String moduleId, String videoId) **async** {

**await** \_db

        .*collection*('modules')

        .*doc*(moduleId)

        .*collection*('videos')

        .*doc*(videoId)

        .*delete*();

  }

  // // ---- QUIZZES ----

  // Future<void> addQuiz(String moduleId, QuizModel quiz) async {

  //   await \_db

  //       .collection('modules')

  //       .doc(moduleId)

  //       .collection('quizzes')

  //       .doc(quiz.id)

  //       .set(quiz.toMap());

  // }

  // Future<void> updateQuiz(String moduleId, QuizModel quiz) async {

  //   await \_db

  //       .collection('modules')

  //       .doc(moduleId)

  //       .collection('quizzes')

  //       .doc(quiz.id)

  //       .update(quiz.toMap());

  // }

  // Future<void> deleteQuiz(String moduleId, String quizId) async {

  //   await \_db

  //       .collection('modules')

  //       .doc(moduleId)

  //       .collection('quizzes')

  //       .doc(quizId)

  //       .delete();

  // }

}

// // ========== lib/services/local\_user\_service.dart ==========

import 'dart:convert';

import 'package:shared\_preferences/shared\_preferences.dart';

class LocalUserService {

  static const \_pendingKey = 'pending\_users\_queue';

  Future<List<Map<String, dynamic>>> *loadPendingUsers*() **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

    final raw = prefs.*getString*(\_pendingKey);

**if** (raw == null || raw.isEmpty) **return** [];

    final list = *jsonDecode*(raw) as List<dynamic>;

**return** list.*cast*<Map<String, dynamic>>();

    // ignore: avoid\_print

  }

  Future<void> *savePendingUserMap*(Map<String, dynamic> userMap) **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

    final list = **await** *loadPendingUsers*();

    list.*add*(userMap);

**await** prefs.*setString*(\_pendingKey, *jsonEncode*(list));

  }

  Future<void> *replacePendingUsers*(List<Map<String, dynamic>> users) **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

**await** prefs.*setString*(\_pendingKey, *jsonEncode*(users));

  }

  Future<void> *clearPendingUsers*() **async** {

    final prefs = **await** SharedPreferences.*getInstance*();

**await** prefs.*remove*(\_pendingKey);

  }

}

// lib/utils/app\_constants.dart

class AppConstants {

  static const appName = 'Réussir BTS';

  static const splashImage = 'assets/illustrations/splash.png';

}

// lib/utils/connectivity\_service.dart

import 'package:connectivity\_plus/connectivity\_plus.dart';

import 'dart:async';

class ConnectivityService {

  static final \_controller = StreamController<bool>.*broadcast*();

  static Stream<bool> get connectivityStream => \_controller.stream;

  static void *initialize*() {

    Connectivity().onConnectivityChanged.*listen*((result) {

      \_controller.*add*(result != ConnectivityResult.none);

    });

  }

  static Future<bool> *isOnline*() **async** {

    final result = **await** Connectivity().*checkConnectivity*();

**return** result != ConnectivityResult.none;

  }

}

// lib/utils/validators.dart

class Validators {

  /// Champ obligatoire

  static String? *notEmpty*(String? v, {String msg = "Champ requis"}) {

**if** (v == null || v.*trim*().isEmpty) **return** msg;

**return** null;

  }

  /// Email format standard

  static String? *email*(String? v) {

**if** (v == null || v.*trim*().isEmpty) **return** "Email requis";

    final re = RegExp(r"^[^\s@]+@[^\s@]+\.[^\s@]+$");

**if** (!re.*hasMatch*(v.*trim*())) **return** "Email invalide";

**return** null;

  }

  /// Mot de passe min 8, au moins une lettre et un chiffre

  static String? *password*(String? v) {

**if** (v == null || v.isEmpty) **return** "Mot de passe requis";

**if** (v.length < 8) **return** "Min. 8 caractères";

    final hasLetter = RegExp(r"[A-Za-z]").*hasMatch*(v);

    final hasDigit = RegExp(r"\d").*hasMatch*(v);

**if** (!hasLetter || !hasDigit) {

**return** "Doit contenir lettres et chiffres";

    }

**return** null;

  }

  /// Nom complet : deux mots, ≥4 lettres chacun, pas de répétition simple

  static String? *fullName*(String? v) {

**if** (v == null || v.*trim*().isEmpty) **return** "Nom requis";

    final parts = v.*trim*().*split*(' ');

**if** (parts.length < 2) **return** "Veuillez saisir prénom et nom";

**for** (var p **in** parts) {

**if** (p.length < 4) **return** "Chaque nom doit avoir ≥4 lettres";

**if** (*\_hasSimpleRepetition*(p)) **return** "Nom invalide (répétition)";

    }

**return** null;

  }

  /// Vérifie le téléphone : chiffres et + seulement, 7-15 caractères

  static String? *phone*(String? v) {

**if** (v == null || v.*trim*().isEmpty) **return** "Téléphone requis";

    final re = RegExp(r'^\+?\d{7,15}$');

**if** (!re.*hasMatch*(v.*trim*())) **return** "Téléphone invalide";

**return** null;

  }

  /// Texte avec minLength et pas de répétition simple

  static String? *minLengthNoRepeat*(String? v, int minLength) {

**if** (v == null || v.*trim*().isEmpty) **return** "Champ requis";

    final t = v.*trim*();

**if** (t.length < minLength) **return** "Min. $*minLength* caractères";

**if** (*\_hasSimpleRepetition*(t)) **return** "Texte invalide (répétition)";

**return** null;

  }

  /// Helper pour détecter répétitions simples comme AAAA, TTTT, AAAB, etc.

  static bool *\_hasSimpleRepetition*(String s) {

**if** (s.isEmpty) **return** false;

    final lower = s.*toLowerCase*();

    final firstChar = lower[0];

**if** (lower.*split*('').*every*((c) => c == firstChar)) **return** true; // AAAA

    final set = lower.*split*('').*toSet*();

**if** (set.length <= 2 && lower.length >= 4) **return** true; // AAAB, AABB

**return** false;

  }

}

// lib/widgets/custom\_button.dart

import 'package:flutter/material.dart';

class CustomButton extends StatelessWidget {

  final String text;

  final VoidCallback onPressed;

  const CustomButton({*super*.key, required *this*.text, required *this*.onPressed});

  @override

  Widget *build*(BuildContext context) {

**return** ElevatedButton(

      onPressed: onPressed,

      child: Text(text),

    );

  }

}

// lib/widgets/custom\_input.dart

import 'package:flutter/material.dart';

class CustomInput extends StatelessWidget {

  final String label;

  final TextEditingController controller;

  final String? errorText;

  final bool isPassword;

  final bool enabled;

  final IconData? icon;

  const CustomInput({

*super*.key,

    required *this*.label,

    required *this*.controller,

*this*.errorText,

*this*.isPassword = false,

*this*.enabled = true,

*this*.icon,

  });

  @override

  Widget *build*(BuildContext context) {

**return** Padding(

      padding: const EdgeInsets.*only*(bottom: 16),

      child: TextFormField(

        controller: controller,

        obscureText: isPassword,

        enabled: enabled,

        decoration: InputDecoration(

          labelText: label,

          prefixIcon: icon != null ? Icon(icon) : null,

          border: OutlineInputBorder(

              borderRadius: BorderRadius.*circular*(12),

              borderSide: const BorderSide()),

          errorText: errorText,

          filled: !enabled,

          fillColor: enabled ? null : Colors.grey.shade200,

        ),

      ),

    );

  }

}

// lib/widgets/microlearning\_card.dart

import 'package:flutter/material.dart';

class MicrolearningCard extends StatelessWidget {

  final String title;

  final int duration;

  final VoidCallback onTap;

  const MicrolearningCard({

*super*.key,

    required *this*.title,

    required *this*.duration,

    required *this*.onTap,

  });

  @override

  Widget *build*(BuildContext context) {

**return** SizedBox(

      width: 200,

      child: Card(

        shape: RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(16)),

        elevation: 2,

        child: InkWell(

          onTap: onTap,

          child: Padding(

            padding: const EdgeInsets.*all*(12),

            child: Column(

              crossAxisAlignment: CrossAxisAlignment.start,

              children: [

                const Icon(Icons.flash\_on, color: Colors.orange, size: 32),

                const SizedBox(height: 12),

                Text(title,

                    style: const TextStyle(

                        fontWeight: FontWeight.bold, fontSize: 16)),

                const Spacer(),

                Text("$*duration* min",

                    style: TextStyle(color: Colors.grey[600], fontSize: 14)),

              ],

            ),

          ),

        ),

      ),

    );

  }

}

// lib/widgets/module\_card.dart

import 'package:flutter/material.dart';

import '../models/course\_model.dart';

class ModuleCard extends StatelessWidget {

  final Course course;

  final VoidCallback onTap;

  const ModuleCard({*super*.key, required *this*.course, required *this*.onTap});

  @override

  Widget *build*(BuildContext context) {

**return** Card(

      shape: RoundedRectangleBorder(borderRadius: BorderRadius.*circular*(16)),

      elevation: 3,

      child: InkWell(

        onTap: onTap,

        child: Padding(

          padding: const EdgeInsets.*all*(12),

          child: Row(

            children: [

              Icon(Icons.book, size: 40, color: Colors.blue[700]),

              const SizedBox(width: 12),

              Expanded(

                child: Column(

                  crossAxisAlignment: CrossAxisAlignment.start,

                  children: [

                    Text(course.title,

                        style: const TextStyle(

                            fontWeight: FontWeight.bold, fontSize: 16)),

                    const SizedBox(height: 4),

                    Text(

                        "${*course*.*countFiches*} fiches • ${*course*.*countVideos*} vidéos",

                        style: TextStyle(color: Colors.grey[600])),

                  ],

                ),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

// lib/widgets/network\_banner.dart

import 'package:flutter/material.dart';

class NetworkBanner extends StatelessWidget {

  const NetworkBanner({*super*.key});

  @override

  Widget *build*(BuildContext context) {

**return** Container(

      color: Colors.red,

      padding: const EdgeInsets.*all*(8),

      width: double.infinity,

      child: const Text(

        'Vous êtes hors ligne',

        style: TextStyle(color: Colors.white, fontWeight: FontWeight.bold),

        textAlign: TextAlign.center,

      ),

    );

  }

}

// lib/firebase\_options.dart

import 'package:firebase\_core/firebase\_core.dart' show FirebaseOptions;

import 'package:flutter/foundation.dart'

    show defaultTargetPlatform, kIsWeb, TargetPlatform;

class DefaultFirebaseOptions {

  static FirebaseOptions get currentPlatform {

**if** (kIsWeb) **return** web;

**switch** (defaultTargetPlatform) {

**case** TargetPlatform.android:

**return** android;

**case** TargetPlatform.iOS:

**return** ios;

**case** TargetPlatform.macOS:

**return** macos;

**case** TargetPlatform.windows:

**return** windows;

**case** TargetPlatform.linux:

**throw** UnsupportedError(

            'Linux non configuré. Reconfigurer avec FlutterFire CLI.');

**default**:

**throw** UnsupportedError('Plateforme non supportée');

    }

  }

  static const FirebaseOptions web = FirebaseOptions(

    apiKey: 'AIzaSyDVPG6iS88SMttGhK6oB-yMzIeip1qLNbE',

    appId: '1:807140710167:web:5bdca195a2577bc4450952',

    messagingSenderId: '807140710167',

    projectId: 'reussirbts',

    authDomain: 'reussirbts.firebaseapp.com',

    storageBucket: 'reussirbts.firebasestorage.app',

  );

  static const FirebaseOptions android = FirebaseOptions(

    apiKey: 'AIzaSyBq-sDrk9GDjKVwB-kbWxWYADQedTPgXno',

    appId: '1:807140710167:android:d89ce773d1aefe13450952',

    messagingSenderId: '807140710167',

    projectId: 'reussirbts',

    storageBucket: 'reussirbts.firebasestorage.app',

  );

  static const FirebaseOptions ios = FirebaseOptions(

    apiKey: 'AIzaSyAbc9n1cmPGZDrIs1q6wy15SOfeLSL22SY',

    appId: '1:807140710167:ios:440ea7369515507e450952',

    messagingSenderId: '807140710167',

    projectId: 'reussirbts',

    storageBucket: 'reussirbts.firebasestorage.app',

    iosBundleId: 'com.example.reussirbts',

  );

  static const FirebaseOptions macos = FirebaseOptions(

    apiKey: 'AIzaSyAbc9n1cmPGZDrIs1q6wy15SOfeLSL22SY',

    appId: '1:807140710167:ios:440ea7369515507e450952',

    messagingSenderId: '807140710167',

    projectId: 'reussirbts',

    storageBucket: 'reussirbts.firebasestorage.app',

    iosBundleId: 'com.example.reussirbts',

  );

  static const FirebaseOptions windows = FirebaseOptions(

    apiKey: 'AIzaSyDVPG6iS88SMttGhK6oB-yMzIeip1qLNbE',

    appId: '1:807140710167:web:c43a142858692d80450952',

    messagingSenderId: '807140710167',

    projectId: 'reussirbts',

    authDomain: 'reussirbts.firebaseapp.com',

    storageBucket: 'reussirbts.firebasestorage.app',

  );

}

// lib/main.dart

import 'package:flutter/material.dart';

import 'package:firebase\_core/firebase\_core.dart';

import 'package:flutter/foundation.dart' show kIsWeb;

import 'package:provider/provider.dart'; // ✅ Nécessaire pour Provider

import 'app/app\_routes.dart';

import 'app/app\_theme.dart';

import 'firebase\_options.dart';

import 'provider/user\_provider.dart'; // ✅ Ton UserProvider

import 'provider/theme\_provider.dart'; // ✅ Nouveau ThemeProvider

/// Flag global pour savoir si on est en mode hors-ligne

bool isOfflineMode = false;

void *main*() **async** {

  WidgetsFlutterBinding.*ensureInitialized*();

**try** {

**if** (kIsWeb) {

**await** Firebase.*initializeApp*(

        options: DefaultFirebaseOptions.web,

      );

    } **else** {

**await** Firebase.*initializeApp*(

        options: DefaultFirebaseOptions.currentPlatform,

      );

    }

*debugPrint*('✅ Firebase initialisé avec succès !');

  } **catch** (e, s) {

*debugPrint*('❌ Erreur initialisation Firebase : $*e*');

*debugPrint*('$*s*');

    // Active le fallback hors-ligne

    isOfflineMode = true;

*debugPrint*('⚠️ Mode hors-ligne activé (fallback local JSON)');

  }

*runApp*(

    MultiProvider(

      providers: [

        ChangeNotifierProvider(

            create: (\_) => UserProvider()), // ✅ UserProvider global

        ChangeNotifierProvider(

            create: (\_) => ThemeProvider()), // ✅ ThemeProvider global

      ],

      child: const ReussirBtsApp(),

    ),

  );

}

class ReussirBtsApp extends StatelessWidget {

  const ReussirBtsApp({*super*.key});

  @override

  Widget *build*(BuildContext context) {

    final themeProvider = Provider.*of*<ThemeProvider>(context);

**return** MaterialApp(

      title: 'Réussir BTS',

      theme: AppTheme.lightTheme,

      darkTheme: AppTheme.darkTheme,

      themeMode: themeProvider.isDark ? ThemeMode.dark : ThemeMode.light,

      initialRoute: AppRoutes.splash,

      routes: AppRoutes.routes,

      debugShowCheckedModeBanner: false,

      builder: (context, child) {

        // Message global si mode hors-ligne

**if** (isOfflineMode) {

**return** Stack(

            children: [

              child ?? const SizedBox.*shrink*(),

              Positioned(

                bottom: 10,

                left: 10,

                right: 10,

                child: Container(

                  padding: const EdgeInsets.*all*(12),

                  decoration: BoxDecoration(

                    color: Colors.redAccent.*withOpacity*(0.9),

                    borderRadius: BorderRadius.*circular*(12),

                  ),

                  child: const Text(

                    '⚠️ Vous êtes en mode hors-ligne. Certaines fonctionnalités peuvent être limitées.',

                    style: TextStyle(color: Colors.white),

                    textAlign: TextAlign.center,

                  ),

                ),

              ),

            ],

          );

        }

**return** child ?? const SizedBox.*shrink*();

      },

    );

  }

}

<!**DOCTYPE** html>

<**html**>

  <**head**>

    <**link** rel="manifest" href="manifest.json" />

    <**meta** name="theme-color" content="#2563EB" />

    <**link** rel="apple-touch-icon" href="icons/Icon-192.png" />

    <**meta** name="apple-mobile-web-app-capable" content="yes" />

    <**meta**

      name="apple-mobile-web-app-status-bar-style"

      content="black-translucent"

    />

    <!--

    If you are serving your web app in a path other than the root, change the

    href value below to reflect the base path you are serving from.

    The path provided below has to start and end with a slash "/" in order for

    it to work correctly.

    For more details:

    \* https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base

    This is a placeholder for base href that will be replaced by the value of

    the `--base-href` argument provided to `flutter build`.

  -->

    <**base** href="$FLUTTER\_BASE\_HREF" />

    <**meta** charset="UTF-8" />

    <**meta** content="IE=Edge" http-equiv="X-UA-Compatible" />

    <**meta** name="description" content="A new Flutter project." />

    <!-- iOS meta tags & icons -->

    <**meta** name="apple-mobile-web-app-capable" content="yes" />

    <**meta** name="apple-mobile-web-app-status-bar-style" content="black" />

    <**meta** name="apple-mobile-web-app-title" content="reussirbts" />

    <**link** rel="apple-touch-icon" href="icons/Icon-192.png" />

    <!-- Favicon -->

    <**link** rel="icon" type="image/png" href="favicon.png" />

    <**title**>reussirbts</**title**>

    <**link** rel="manifest" href="manifest.json" />

  </**head**>

  <**body**>

    <**script** src="flutter\_bootstrap.js" async></**script**>

    <**script**>

      if ("serviceWorker" in navigator) {

        window.addEventListener("flutter-first-frame", function () {

          navigator.serviceWorker.register("flutter\_service\_worker.js");

        });

      }

    </script>

  </body>

</html>

{

  "name": "Réussir BTS",

  "short\_name": "RéussirBTS",

  "description": "Ton hub d’apprentissage moderne : révisions intelligentes, quiz, suivi.",

  "start\_url": "/",

  "display": "standalone",

  "background\_color": "#ffffff",

  "theme\_color": "#2563EB",

  "orientation": "portrait-primary",

  "icons": [

    {

      "src": "icons/Icon-192.png",

      "sizes": "192x192",

      "type": "image/png"

    },

    {

      "src": "icons/Icon-512.png",

      "sizes": "512x512",

      "type": "image/png"

    }

  ]

}

name: reussirbts

description: Une application de révision pour le BTS au Sénégal

publish\_to: "none"

version: 1.0.0+1

environment:

  sdk: ">=3.4.0 <4.0.0"

dependencies:

  flutter:

    sdk: flutter

  # Firebase

  firebase\_core: ^2.27.0

  firebase\_auth: ^4.18.0

  cloud\_firestore: ^4.16.0

  firebase\_storage: ^11.6.9

  # State management

  provider: ^6.1.2

  # Polices et styles

  google\_fonts: ^6.1.0

  cupertino\_icons: ^1.0.8 # <- Ajout obligatoire pour iOS/Flutter icons

  # Connectivité (online/offline)

  connectivity\_plus: ^4.0.2

  # Stockage local

  shared\_preferences: ^2.2.3

  path\_provider: ^2.1.4

  # Upload fichiers / images

  file\_picker: ^8.0.0+1

  # Liens externes

  url\_launcher: ^6.3.0

  # SVG (optionnel)

  flutter\_svg: ^2.0.9

  # Téléchargement & ouverture fichiers

  dio: ^5.7.0

  open\_file: ^3.3.2

dev\_dependencies:

  flutter\_test:

    sdk: flutter

  flutter\_lints: ^2.0.1

flutter:

  uses-material-design: true

  assets:

    - assets/images/bts\_logo.png

    - assets/illustrations/splash.png

methode avant de reecrire un fichier demande le code actuel et on reecrire tout le fichier en prenant en comptant les nouvelles solutions eglant les anciens problemes