

Discussion avec ChatGPT

Les réponses ne correspondent pas exactement à ce qui a été obtenu, j'ai omi les parties qui ne m'intéressaient pas et les erreurs que j'ai fixé à la main en quelques secondes.

Utilisateur: Voici les spécifications pour mon projet, liste moi déjà dans un premier les technologies que tu me recommandes:

Présentation de l'application. Une fois l'application installée et sans inscription, elle permet à son utilisateur de bénéficier de :

- Présentation claire et ergonomique de l'application sous différents formats : vidéo, texte, etc.
- Une démonstration (utilisation) réelle mais temporaire des services de l'application.
- Une interface de connexion
- Etc.

Inscription. L'application permet de s'inscrire selon trois profils :

- Parent : Dans ce cas, l'application lui propose :
- De saisir les informations du parent : nom, prénom, choix login, choix mot de passe, adresse mail, compte Tweeter, Facebook, etc. Permettre de vérifier

l'adresse mail (Par exemple, activation du compte via une lien envoyé à l'adresse mail en renseignée).

- De choisir le nombre d'élèves concernés par cette inscription et de saisir leurs

profils (nom, prénom, lien de parenté, niveau scolaire, etc.)

- De choisir l'année ou les années scolaires concernées.
- De choisir une formule parmi un ensemble de formules proposées. Par exemple : ...

ChatGPT: Pour ce projet d'application mobile avec des fonctionnalités spécifiques à plusieurs profils (parent et élève), ainsi qu'une partie serveur robuste, voici les technologies et outils que je te recommande pour chaque partie :

Frontend Mobile (Android) :

Pour l'interface utilisateur, l'accessibilité et l'expérience utilisateur, il est important d'utiliser des outils et frameworks adaptés au développement Android.

Langage : Kotlin Kotlin est maintenant le langage recommandé pour le développement Android, car il offre une syntaxe plus simple et plus concise que Java.

UI Design : Material Design Utilisation des principes de Material Design pour offrir une interface ergonomique et fluide, conforme aux standards modernes.

Android Jetpack Components : Fragments : Pour gérer la navigation entre les différentes vues

(parent, élève, etc.) dans l'application. ViewModel & LiveData : Pour gérer les données en temps réel et la persistance à travers les cycles de vie des activités. Room : Pour la gestion de la base de données locale, permettant le mode offline. DataBinding : Pour lier les composants de l'interface utilisateur aux données en toute simplicité.

Librairies pour la gestion multimédia : ExoPlayer : Pour la gestion et l’affichage des vidéos (utilisé pour les cours en format vidéo). Glide : Pour le chargement et l’affichage optimisé des images (fiches de cours, avatars, etc.).

Backend (Serveur) :

La partie serveur doit être capable de gérer des inscriptions, stocker des données, exécuter des algorithmes, et supporter la synchronisation offline/online.

- Langage et Framework Backend :
- Node.js avec Express.js : Idéal pour un backend léger et rapide, avec des bibliothèques riches pour le développement d’API RESTful.
- Python avec Django ou Flask : Si tu prévois des algorithmes plus complexes (pour les recommandations ou la gestion des courbes de progression), Python peut être un bon choix grâce à ses bibliothèques d’intelligence artificielle et de machine learning.
- Base de données :
 - PostgreSQL : Pour une base de données relationnelle robuste, avec une bonne gestion des relations entre les différentes entités (utilisateurs, cours, exercices, etc.).
 - MongoDB : Si tu préfères une base de données NoSQL, plus flexible pour stocker des documents (cours, activités des élèves, etc.).
- Authentification & Sécurité :
 - OAuth2 pour l’authentification via des services externes (Google, Facebook, Twitter).
 - JWT (JSON Web Token) : Pour gérer les sessions d’utilisateurs de manière sécurisée.

Synchronisation Online/Offline :

Pour permettre un mode offline avec synchronisation en ligne lorsque l’utilisateur repasse en mode connecté :

Realm : Base de données mobile qui permet une synchronisation facile entre le mode offline et online. WorkManager : Pour gérer les tâches de fond et la synchronisation des données en mode offline/online.

Paielements en ligne :

Pour la gestion des paiements des parents pour les formules choisies :

- Stripe : Pour les paiements sécurisés par carte bancaire.
- PayPal SDK : Comme alternative de paiement en ligne.

Notifications et Rappels :

Pour la gestion des notifications à propos des cours ou des rappels d’activité :

- Firebase Cloud Messaging (FCM) : Pour l’envoi de notifications push.
- AlarmManager : Pour planifier des notifications locales ou rappels même en mode offline.

Intelligence artificielle pour recommandations :

Pour les recommandations de progression aux élèves :

- TensorFlow Lite : Si tu envisages de faire de l’IA embarquée sur mobile pour personnaliser les recommandations.
- Python (Scikit-Learn ou TensorFlow) : Si l’intelligence artificielle est gérée côté serveur.

Tests et Qualité de Code :

- JUnit & Espresso : Pour tester les fonctionnalités et l'interface utilisateur.
- SonarQube : Pour l'analyse de la qualité du code.

Autres outils à envisager :

- Retrofit : Pour les appels réseau, notamment la communication entre l'application et le serveur.
- Coroutines (Kotlin) : Pour la gestion asynchrone, en particulier pour les appels réseau et la gestion de tâches de fond. SpeechRecognizer API : Pour l'activation de fonctionnalités par la voix.

Gestion des événements en temps réel :

WebSockets : Pour la communication en temps réel dans le cadre des cours en direct ou des chats.

Hébergement Backend :

- AWS (Amazon Web Services) ou Google Cloud : Pour l'hébergement des services backend et la gestion de la base de données.
- Heroku : Option plus simple pour des déploiements rapides avec une gestion facile des bases de données et des environnements.

Résumé :

- Frontend : Kotlin, Android Jetpack, Material Design, ExoPlayer, Glide.
- Backend : Node.js/Express ou Python/Django, PostgreSQL/MongoDB, OAuth2, JWT.
- Sync : Realm, WorkManager.
- Paiement : Stripe, PayPal SDK.
- Notifications : Firebase Cloud Messaging.
- IA : TensorFlow Lite (mobile) ou Python (backend).
- Tests : JUnit, Espresso.

Cela te donnera une architecture performante et moderne pour ton application.

Utilisateur: Génère un morceau de code qui va afficher un écran de sélection entre les différents types d'utilisateur avec Kotlin et Jetpack Compose.

Pas de bla-bla, uniquement le code

ChatGPT:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            UserSelectionScreen(onUserSelected = { userType ->
                // Handle user selection (Parent, Student, etc.)
            })
        }
    }
}
```

@Composable

```
fun UserSelectionScreen(onUserSelected: (String) -> Unit) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Center,
```

```

        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(text = "Choisissez votre profil", fontSize = 24.sp, color = Color.Black)

        Button(
            modifier = Modifier.padding(top = 16.dp),
            onClick = { onUserSelected("Parent") }
        ) {
            Text(text = "Parent")
        }

        Button(
            modifier = Modifier.padding(top = 16.dp),
            onClick = { onUserSelected("Élève") }
        ) {
            Text(text = "Élève")
        }
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewUserSelectionScreen() {
    UserSelectionScreen(onUserSelected = {})
}

```

Utilisateur: Parfait, quelle serait l'étape d'après, d'après toi, liste en plusieurs 2-3 plutôt courte que je choisisse

ChatGPT: Voici 3 prochaines étapes possibles après l'écran de sélection des utilisateurs :

1. Création des écrans d'inscription : Développer les écrans d'inscription pour chaque type d'utilisateur (Parent, Élève) avec les champs requis (nom, email, mot de passe, etc.) et la validation des formulaires.
2. Implémentation de la logique de navigation : Utiliser Jetpack Navigation pour gérer la navigation entre les différents écrans (sélection, inscription, connexion, etc.).
3. Connexion au backend : Mettre en place les appels réseau (via Retrofit par exemple) pour envoyer les données d'inscription/connexion au serveur.

Choisis selon la priorité de ton projet !

Utilisateur: Faisons le 1, post-inscription, on va afficher le choix du type d'utilisateur. Pas de bla-bla

ChatGPT:

```

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            RegistrationScreen()
        }
    }
}

```

```
}
```

```
@Composable
```

```
fun RegistrationScreen() {  
    var name by remember { mutableStateOf("") }  
    var email by remember { mutableStateOf("") }  
    var password by remember { mutableStateOf("") }  
    var isRegistered by remember { mutableStateOf(false) }  
  
    if (isRegistered) {  
        UserSelectionScreen(onUserSelected = { userType ->  
            // Handle user type after registration  
        })  
    } else {  
        Column(  
            modifier = Modifier  
                .fillMaxSize()  
                .padding(16.dp),  
            verticalArrangement = Arrangement.Center,  
            horizontalAlignment = Alignment.CenterHorizontally  
        ) {  
            Text(text = "Inscription", fontSize = 24.sp, color = Color.Black)  
  
            BasicTextField(  
                value = name,  
                onChange = { name = it },  
                modifier = Modifier.padding(top = 16.dp),  
                decorationBox = { innerTextField ->  
                    if (name.isEmpty()) Text("Nom") else innerTextField()  
                }  
            )  
  
            BasicTextField(  
                value = email,  
                onChange = { email = it },  
                modifier = Modifier.padding(top = 16.dp),  
                decorationBox = { innerTextField ->  
                    if (email.isEmpty()) Text("Email") else innerTextField()  
                }  
            )  
  
            BasicTextField(  
                value = password,  
                onChange = { password = it },  
                modifier = Modifier.padding(top = 16.dp),  
                decorationBox = { innerTextField ->  
                    if (password.isEmpty()) Text("Mot de passe") else  
innerTextField()  
                }  
            )  
  
            Button(  
                modifier = Modifier.padding(top = 16.dp),  
                onClick = { isRegistered = true }  
            ) {  
                Text(text = "S'inscrire")  
            }  
        }  
    }  
}
```

```

    }
  }
}

```

@Composable

```

fun UserSelectionScreen(onUserSelected: (String) -> Unit) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(text = "Choisissez votre profil", fontSize = 24.sp, color = Color.Black)

        Button(
            modifier = Modifier.padding(top = 16.dp),
            onClick = { onUserSelected("Parent") }
        ) {
            Text(text = "Parent")
        }

        Button(
            modifier = Modifier.padding(top = 16.dp),
            onClick = { onUserSelected("Élève") }
        ) {
            Text(text = "Élève")
        }
    }
}

```

@Preview(showBackground = true)

@Composable

```

fun PreviewRegistrationScreen() {
    RegistrationScreen()
}

```

Utilisateur:

```
[versions]
agp = "8.6.0"
kotlin = "1.9.0"
coreKtx = "1.10.1"
junit = "4.13.2"
junitVersion = "1.1.5"
espressoCore = "3.5.1"
appcompat = "1.6.1"
material = "1.10.0"

[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
junit = { group = "junit", name = "junit", version.ref = "junit" }
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
androidx-appcompat = { group = "androidx.appcompat", name = "appcompat", version.ref = "appcompat" }
material = { group = "com.google.android.material", name = "material", version.ref = "material" }

[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
```

Ajoute les dépendances pour la navigation

ChatGPT:

```
[libraries]
androidx-navigation-fragment-ktx = {
    group = "androidx.navigation",
    name = "navigation-fragment-ktx",
    version = "2.7.3"
}
androidx-navigation-ui-ktx = {
    group = "androidx.navigation",
    name = "navigation-ui-ktx",
    version = "2.7.3"
}
```

Utilisateur: Bien, il me manque compose maintenant, ajoute le. Pas de blabla.

ChatGPT:

```
[versions]
compose = "1.5.0"

[libraries]
androidx-compose-ui = { group = "androidx.compose.ui", name = "ui", version.ref = "compose" }
```

```

androidx-compose-material = { group = "androidx.compose.material", name = "material",
version.ref = "compose" }
androidx-compose-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling",
version.ref = "compose" }
androidx-compose-runtime = { group = "androidx.compose.runtime", name = "runtime",
version.ref = "compose" }
androidx-compose-foundation = { group = "androidx.compose.foundation", name =
"foundation", version.ref = "compose" }
androidx-compose-navigation = { group = "androidx.navigation", name = "navigation-
compose", version = "2.7.3" }

```

Utilisateur: Parfait, maintenant créer des structures de données avec Kotlin pour représenter les données dans les pré-requis. Pas de blabla

ChatGPT:

```

data class Parent(
    val id: String,
    val name: String,
    val email: String,
    val password: String,
    val children: List<Student>
)

data class Student(
    val id: String,
    val name: String,
    val relationshipToParent: String,
    val gradeLevel: String,
    val schoolYear: List<String>
)

data class Course(
    val id: String,
    val title: String,
    val description: String,
    val format: CourseFormat,
    val year: String
)

enum class CourseFormat {
    DOCUMENTS_ONLY,
    TEACHER_SUPPORT
}

data class PaymentOption(
    val id: String,
    val type: PaymentType,
    val details: String
)

enum class PaymentType {
    CREDIT_CARD,
    DIRECT_DEBIT
}

```



```

data class User(
    val id: String,
    val userType: UserType,
    val login: String,
    val password: String
)

enum class UserType {
    PARENT,
    STUDENT
}

data class ActivityLog(
    val studentId: String,
    val activityType: ActivityType,
    val timestamp: Long
)

enum class ActivityType {
    COURSE_VIEW,
    EXERCISE_COMPLETION,
    LOGIN
}

data class Recommendation(
    val studentId: String,
    val advice: String,
    val timestamp: Long
)

```

Utilisateur: Parfait, pour le moment tu vas mettre des valeurs par défaut au lieu de travailler avec une base de donnée, tu vas aussi ajouter des boutons à la page de sélection le temps que j'implémente l'authentification.

Fais une page qui liste les cours d'un étudiant.

Pas de blabla, que le code du composant. Oublie pas le jeu de donnée dans la preview en paramètre du composant

ChatGPT:

```

@Composable
fun StudentCoursesScreen(courses: List<Course>) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.Start
    ) {
        Text(text = "Cours de l'Étudiant", fontSize = 24.sp)

        Spacer(modifier = Modifier.height(16.dp))

        for (course in courses) {

```

```

        CourseItem(course = course)
        Spacer(modifier = Modifier.height(8.dp))
    }
}

@Composable
fun CourseItem(course: Course) {
    Column(modifier = Modifier.padding(8.dp)) {
        Text(text = course.title, fontSize = 20.sp)
        Text(text = course.description)
        Text(text = "Format: ${course.format}", fontSize = 12.sp)
        Text(text = "Année: ${course.year}", fontSize = 12.sp)
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewStudentCoursesScreen() {
    val sampleCourses = listOf(
        Course("1", "Mathématiques", "Cours sur les nombres et les opérations.",
        CourseFormat.DOCUMENTS_ONLY, "2024"),
        Course("2", "Physique", "Introduction à la physique moderne.",
        CourseFormat.TEACHER_SUPPORT, "2024"),
        Course("3", "Chimie", "Principes de base de la chimie.",
        CourseFormat.DOCUMENTS_ONLY, "2024")
    )

    StudentCoursesScreen(courses = sampleCourses)
}

```

Utilisateur: Une idée d'autre page qu'on peut déjà créer ? Enumère en 2-3

ChatGPT: Voici quelques idées de pages que tu pourrais créer :

1. Page de Détails du Cours : Affiche les détails d'un cours spécifique, y compris le contenu, les exercices associés, et les ressources supplémentaires.
2. Page de Suivi de Progression : Montre une vue d'ensemble des progrès de l'étudiant, avec des graphiques et des statistiques sur les cours suivis et les performances.
3. Page de Profil de l'Étudiant : Permet à l'étudiant de voir et de modifier ses informations personnelles, comme le nom, le niveau scolaire, et les préférences de communication.

Ces pages peuvent aider à enrichir l'expérience utilisateur avant l'implémentation complète de l'authentification et des données en temps réel.

Utilisateur: Créer la page de détail de cours avec un jeu de donnée en preview. Pas de blabla que du code

ChatGPT:

```

@Composable
fun CourseDetailScreen(courseDetail: CourseDetail) {
    Column(

```

```

        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.Start
    ) {
        Text(text = courseDetail.title, fontSize = 24.sp)

        Spacer(modifier = Modifier.height(8.dp))

        Text(text = "Description:", fontSize = 18.sp)
        Text(text = courseDetail.description)

        Spacer(modifier = Modifier.height(16.dp))

        Text(text = "Contenu:", fontSize = 18.sp)
        courseDetail.content.forEach { item ->
            Text(text = "- $item")
        }

        Spacer(modifier = Modifier.height(16.dp))

        Text(text = "Exercices:", fontSize = 18.sp)
        courseDetail.exercises.forEach { exercise ->
            Text(text = "- $exercise")
        }

        Spacer(modifier = Modifier.height(16.dp))

        Text(text = "Format: ${courseDetail.format}", fontSize = 12.sp)
        Text(text = "Année: ${courseDetail.year}", fontSize = 12.sp)
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewCourseDetailScreen() {
    val sampleCourseDetail = CourseDetail(
        id = "1",
        title = "Mathématiques",
        description = "Cours sur les nombres et les opérations.",
        content = listOf("Nombres entiers", "Nombres rationnels", "Opérations de
base"),
        exercises = listOf("Exercice 1", "Exercice 2", "Exercice 3"),
        format = CourseFormat.DOCUMENTS_ONLY,
        year = "2024"
    )

    CourseDetailScreen(courseDetail = sampleCourseDetail)
}

```