Projet : Mini-projet de réservation pour un resto fictif en Flutter

Introduction

Pour bien démarrer ce **mini-projet de réservation pour un resto fictif**, vous pouvez reprendre (cloner) votre projet de menu de restaurant comme base de votre application (voici mon code sur GitHub : https://github.com/ariatsi/flutter_restaurant_app) ou bien commencer un nouveau projet.

Voici quelques étapes et livrables que vous allez mettre en place **en équipe** de 4 étudiants.

1. Rédiger un mini cahier des charges

1. **Contexte & objectifs**
    - Présenter rapidement le restaurant (nom, concept, type de clientèle).
    - Définit l'objectif de l'application (prendre et gérer des réservations, consulter le menu, etc.).
2. **Cibles & rôles utilisateurs**
    - **Client** : peut voir la dispo des tables, réserver, modifier ou annuler une réservation.
    - **Hôte/Serveur** (back-office) : peut consulter la liste des réservations, valider ou refuser, gérer le plan de salle.
    - (***Éventuellement***) **Admin** : gérer les menus, les plages horaires, ajouter/supprimer des restaurants.
3. **Fonctionnalités principales**
    - **Front-end Flutter (mobile)**
        - Écran d'accueil + menu de restaurant.
        - Recherche de date/heure pour réservation.
        - Formulaire de réservation (nom, téléphone, nombre de couverts).
        - Confirmation + rappel (notifications push ou e-mail).
        - (***Éventuellement***) Si le temps permet, système de création de compte / connexion avant de réserver (email/mot de passe).
    - **API**
        - Endpoints CRUD pour les réservations (GET /reservations, POST /reservations, etc.).
        - Gestion de la disponibilité des créneaux.
        - (***Éventuellement***) Si le temps permet, Endpoint pour connexion et création de compte.
    - **(Optionnel si temps) Web/App**
        - Interface responsive pour les mêmes fonctionnalités qu'en mobile.

4. **Contraintes techniques & choix de stack**
    - **Mobile** : Flutter (Dart, version ≥ 3.0)
    - **API** : Node.js/Express ou PHP (ou équivalent)
    - **Base de données** : MySQL (ou équivalent)
    - **Authentification** : simple token (JWT) (ou équivalent)
5. **Livrables attendus pour cette étape 1**
    - Maquettes ou wireframes (Figma ou équivalent)
    - Back-log de user stories
    - Repo GitHub avec README et premiers commits
    - Board Trello/Notion avec tâches assignées (**important !**)

2. Mettre en place les outils de suivi

- **Gestion des tâches**
    - **Trello**
        - Colonnes typiques : Backlog | À faire | En cours | Code review | Terminé
        - Chaque carte = une user story ou tâche technique (ex. « Créer l'écran de sélection de date »).
    - **Notion**
        - Table Kanban intégrée ou page « Cahier des charges » + suivis de tâches en checklist.
- **Versionning**
    - Créer un repo GitHub/GitLab dès le départ.
- **Communication**
    - Notre Canal Microsoft Teams pour le chat rapide.
    - Réunions courtes (stand-up) 10 min max, par demi-journée.

3. Définir un découpage en user stories

Formule chaque besoin du client sous forme de user story :

En tant que [rôle], je veux [action] afin de [bénéfice].

Par exemple :

- **US-001** : « En tant que client, je veux sélectionner une date et un créneau horaire pour voir les disponibilités. »
- **US-002** : « En tant que client, je veux saisir mes coordonnées pour réserver une table. »

- **US-010** : « En tant qu'hôte, je veux valider ou refuser une réservation pour tenir à jour le planning. »

Attribue à chaque story :

- Une estimation (petit/moyen/large).
- Un responsable.
- Un critère d'acceptation (ex. « Après réservation, l'utilisateur reçoit un écran de confirmation »).

4. Configurer l'environnement de dev

1. **Flutter**
    - Installer Flutter SDK, configurer un émulateur Android.
    - Créer un nouveau projet flutter `create resto_app`.
2. **API**
    - Initialiser un projet Node.js ou PHP ou équivalent.
    - Ajouter une route pour tester le serveur (serveur node / XAMPP / etc.).
3. **Base de données**
    - Installer le client (ou package) MySQL/phpMyAdmin ou équivalent.

5. Premier sprint : MVP (Version Minimale Viable)

1. **Sprint planning (1 demi-journée)**
    - Sélectionner 3–4 US prioritaires (ex. écrans de date, formulaire de résa, stub API).
2. **Sprint review**
    - Démontrer l'écran de sélection de date et la création d'une réservation (même sans persistance).
    - Mettre à jour le board, retro (ce qui a été facile/difficile).

6. Second Sprint : Fonctionnalités avancées (après-midi)

L'objectif de ce sprint est d'enrichir l'application avec des fonctionnalités avancées et de se rapprocher d'un produit utilisable. Chaque groupe devra au minimum respecter les exigences **obligatoires** pour obtenir la moyenne.

Fonctionnalités obligatoires (pour valider le projet / atteindre la moyenne) :

- **Affichage du menu** (accessible sans connexion, réutilisable depuis un TP précédent).
- **Ajout d'une réservation** avec un formulaire fonctionnel.
- **Connexion / inscription utilisateur** (email + mot de passe minimum).

Fonctionnalités pour monter la note :

- **Stockage réel des réservations** en base de données.
- **Vérification de la disponibilité par créneau horaire** (ex. 14h = 7 places restantes).
- **Modification / suppression d'une réservation** par l'utilisateur (via interface + API).
- **Écran back-office pour l'hôte** avec validation / refus des réservations.

Fonctionnalités bonus (optionnelles) :

- **Gestion des tables** (ex. 1 personne → occupe une table de 2, la place est décomptée).
- **Notifications locales ou par e-mail** (confirmation envoyée à l'utilisateur).
- **Notifications locales ou par e-mail** (confirmation envoyée à l'hôte).
- **Intégration d'une carte Google Maps** (localisation du restaurant ou sélection).

Livrables finaux attendus

Chaque groupe devra remettre les livrables suivants **à la fin de la journée** :

1. Dépôt GitHub (public) – à transmettre via devoir Teams

Le dépôt GitHub **doit être public** et contenir :

- Le code Flutter dans un dossier `frontend/` (ou à la racine).
- Le back-end/API dans un dossier `backend/`.
- Un fichier `README.md` contenant :
    - Une **description claire du projet**.
    - Les **instructions de lancement** (Flutter + API si présente).
    - Un résumé des **fonctionnalités réalisées**.
    - **Aucune donnée personnelle ni confidentielle** ne doit y figurer.
- Le **fichier `.sql` du dump de la base de données** doit être **ajouté dans le dépôt**.

2. Document PDF de documentation (à déposer sur Teams)

Ce document est **obligatoire** et devra inclure :

- Le **nom du groupe** + **noms des 4 membres**.
- Le **rôle de chacun** (**obligatoire** – qui a fait quoi) pour les notes individuelles.
- Les **identifiants de connexion** pour tester l'application (ex. : utilisateur test, admin, etc.).
- Des **captures d'écran** de :
    - L'application (fonctionnalités principales visibles).

- Le **Board Trello ou Notion** avec les user stories (partage optionnel, mais les **screens sont obligatoires**).

3. Vidéo de démonstration

Une courte vidéo (~1 minute) montrant les fonctionnalités principales de l'application.

- Vous pouvez filmer l'écran avec l'outils de capture de vidéo intégré dans le **AVD de l'Android Studio** un avec un autre outil simple (ex : OBS, outil de capture Windows/Mac, Loom, etc.).
- La vidéo peut être ajoutée dans GitHub, le PDF (lien Drive) ou transmise séparément dans le devoir Teams.

Modalités de notation (sur 20 points)

Critère	Détail	Points
Fonctionnalités obligatoires	Affichage du menu, ajout de réservation, connexion/inscription utilisateur	**/8**
Fonctionnalités avancées	Stockage en BDD, vérification des créneaux, édition réservation, back-office hôte	**/5**
Interface Flutter	Design clair, responsive, navigation fluide	**/2**
Structure du projet et code	Organisation du dépôt, clarté du code, séparation front/back, usage des composants	**/2**
Documentation et livrables	GitHub public, README propre, dump SQL, PDF complet avec rôles, captures, accès comptes	**/2**
Travail d'équipe et méthode	Répartition des tâches claire ("qui a fait quoi"), méthode agile, board Trello ou Notion	**/1**
Bonus techniques (jusqu'à +3 pts)	Fonctionnalités supplémentaires : notifications, gestion des tables, carte, effort notable	**+0 → +3**
Appréciation personnelle (bonus/malus)	Qualité et complexité des tâches accomplies, implication individuelle dans le projet	**± variable**

Remarques importantes :

- **La note finale ne pourra pas dépasser 20.** Par exemple, un groupe ayant obtenu 19 pourra recevoir au maximum +1 en bonus.
- **L'appréciation personnelle du professeur** pourra valoriser ou minorer un travail selon la qualité de l'implication individuelle, le sérieux, l'autonomie ou au contraire les retards, erreurs majeures ou absentéismes.
- Tous les critères sont évalués à partir des **livrables rendus** et de la **qualité visible du travail en groupe.**