Projet Final : API Node / Express

Application de réservation pour un restaurant fictif

Objectif général

Vous allez créer une **API REST complète en Node.js + Express** avec base de données **MySQL**, capable de gérer :

- Les **réservations** de clients
- Le **menu du restaurant** (visible sans compte)
- La **gestion des tables et des créneaux horaires**
- Et une **authentification sécurisée**

> *Cette API sera utilisée **dans un second temps** par une application **Frontend** qui consommera vos routes. Elle doit donc être complète, propre, bien documentée et maintenable.*

Organisation du projet

- Travail en **groupe de 3-4 étudiants**.
- Chacun doit **participer à au moins une fonctionnalité complète**, et indiquer clairement dans le `README.md` (ou dans un fichier PDF de documentation) qui a fait quoi (par ex. routes, middleware, CRUD...).
- Le **code sera évalué collectivement**, mais la participation individuelle sera vérifiée.

Partie 1 – 60 % de la note

Réalisable en **une demi-journée** par groupe de **3-4 étudiants**.

Fonctionnalités à implémenter

1. Authentification

- **POST** `/signup` : créer un compte utilisateur (côté client)
- **POST** `/login` : authentification avec email / mot de passe et retour d'un **JWT**
- Middleware Express (authMiddleware) pour protéger toutes les routes privées

2. Gestion des Réservations (/reservations)

- **GET** `/reservations` : récupérer toutes les réservations (admin uniquement)
- **GET** `/my-reservations` : récupérer **ses propres** réservations (client connecté)
- **POST** `/reservations` : créer une réservation (**nécessite un compte client**)
  - Champs requis : **name**, **phone**, (ou juste **user_id**) **number_of_people**, **date**, **time**, **note**
  - Vérifier que le créneau est disponible, impossible de réserver pour plus que la capacité disponible
  - L'attribution automatique d'une ou plusieurs tables selon la capacité
  - La réservation est enregistrée en "**en attente**"

- **PUT** `/reservations/:id` : modifier une réservation (si encore en attente)
- **DELETE** `/reservations/:id` : annuler une réservation (toujours possible)
- **PATCH** `/reservations/:id/validate` : l'admin valide la réservation (statut devient confirmée et plus modifiable)

Une réservation a donc toujours un **status** :

- "`pending`" par défaut
- "`confirmed`" (valable uniquement par un admin)
- "`cancelled`" (client ou admin)

3. Menu du restaurant (/menu)

- **GET** `/menu` : **consultable librement**, accessible sans authentification
- Données à structurer en catégories : *Entrées*, *plats*, *desserts*, *boissons*, etc.
- Chaque plat possède : **id**, **name**, **description**, **price**, **category** et éventuellement une **image**

4. Gestion des tables

- Tables fixes de 2, 4 et 6 places (admin a la possibilité d'ajouter / modifier les tables disponibles au restaurent),
- Une réservation utilise 1 ou plusieurs tables selon le nombre de personnes,
- Doit vérifier la **disponibilité par créneau**
  - Exemple : 7 personnes → 1 table de 6 + 1 table de 2,
  - Ne pas dépasser la capacité des tables disponibles.

5. Documentation API (obligatoire)

- Fournir un fichier `README.md` et un document PDF (plus une page `docs.json` - optionnelle)
  - Liste des **routes disponibles**
  - Exemples de **requêtes JSON** (input/output)
  - Les méthodes attendues (**GET**, **POST**, etc.)
  - **Statuts HTTP utilisés**
  - Les conditions d'accès (auth requise ou non)

Partie 2 – Jusqu'à 40 % de la note

Réalisable en **une demi-journée** supplémentaire.

Fonctionnalités valorisées. Choisissez celles qui vous intéressent ou que vous êtes capables de réaliser.

Suggestions de fonctionnalités

1. Créneaux personnalisés et gestion des horaires

- Table `opening_slots` définissant les créneaux disponibles **par jour**
  - Chaque jour peut avoir plusieurs créneaux : 12h, 13h, 19h, 20h30
- Gestion **d'exceptions d'ouverture/fermeture**
  - Fermeture exceptionnelle - aucune réservation possible ce jour

- Ajout de créneaux exceptionnels par admin

- Par exemple Endpoint **GET** `/availability?date=2026-06-20` pour afficher les créneaux disponibles ce jour-là.

2. Rôles utilisateur

- Champ `role` dans `users` : "**admin**" ou "**client**"
- L'admin peut :
  - Confirmer ou annuler toute réservation
  - Gérer les horaires
  - Accéder à réservations et tables
  - Ajout/suppression de créneaux
  - Ajout/suppression de tables
  - Voir les statistiques : nb de clients / jour, etc.
- Le client peut :
  - Réserver un créneau,
  - Voir ses réservations,
  - Annulé une réservation.

3. Gestion des tables de restaurant

- Types de tables fixes (2, 4, 6 places, etc…)
- Une réservation peut mobiliser plusieurs tables (Exemple : 8 personnes = 1 table de 6 + 1 table de 2)
- Si aucune combinaison possible → refus de la réservation

4. Filtres dynamiques

- Filtrage sur les réservations : `GET /reservations?date=...&status=...&sort=...`
- Recherche de plats dans le menu : `GET /menu?category=desserts&max_price=15`

5. Notifications simulées

- Message console à chaque action et/ou les logs consultable :

```
[NOTIF] Réservation "confirmée" pour Marie DURND à 20h le 15/06.
```

6. Proposition d'une structure basique de la base de données (MySQL)

Exemples de tables à inclure (évolutives) :

- `users(id, email, password_hash, fname, lname, phone, role)`,
- `reservations(id, user_id, number_of_people, date, time, status)`,
- `tables(id, seats)`,
- `reservation_tables(reservation_id, table_id)`,
- `menu_items(id, name, description, price, category)`,
- `opening_slots(id, date_time, duration, available, comment)`.

Livrables

Organisation du travail

- Vous travaillez **en groupe de 3 à 4 personnes**.
- Il est **obligatoire d'indiquer clairement qui fait quoi** :
  - Dans le README du projet,
  - Ou / et dans le fichier PDF de documentation.
- Utilisez **Git** pour suivre les contributions.
- Utilisation d'un outil de gestion des taches / de projet et fortement souhaitable (la capture d'écran à joindre).

Ce que vous devez rendre

- Tous les **endpoints** doivent être testables via **Postman** ou **ThunderClient**.
- Code complet (server.js, routes/, controllers/, models/, etc.) via repo publique **GitHub** / GitLab.
- Le schéma de la BDD (MCD ou MLD ou MPD),
- Le fichier `.sql` pour créer les tables doit être inclut.
- Un fichier `README.md` clair avec :
  - Explication du projet
  - Documentation des routes
  - Instructions d'installation et de test
- Token JWT de test ou compte déjà enregistré
  - Les comptes clients / admin avec des email demo type `jean@example.com` et des mots de passe pour tester l'API.

Présentation orale et démonstration de l'API

Chaque groupe devra présenter son projet à l'oral lors de la séance prévue.

Recommandation : **15 minutes max par groupe**, la démo **doit être préparée** et fluide (tokens prêts, requêtes enregistrées, …).

Cette présentation devra inclure :

- Une **présentation technique de l'API** (architecture, choix techniques, sécurité, base de données),
- Une **démonstration fonctionnelle en direct** des endpoints principaux à l'aide d'un outil de test d'API (Postman, Thunder Client, Insomnia ou équivalent).

La démonstration devra montrer que l'API est **fonctionnelle, sécurisée et conforme au cahier des charges**, avec des requêtes réelles et des réponses exploitables.

Chaque membre du groupe devra **prendre la parole** et être capable d'expliquer les parties techniques sur lesquelles il ou elle a travaillé.

Caractéristiques techniques à présenter à l'oral

1. Architecture générale

- Stack utilisée : Node.js, Express, MySQL
- Organisation du projet : routes / controllers / models / middlewares
- Principe REST appliqué (ressources, verbes HTTP)

2. Base de données

- Modèle de données (MCD / MLD / MPD)
- Tables principales (users, reservations, tables, menu, etc.)
- Relations entre les tables
- Gestion des capacités et des créneaux

3. Authentification & sécurité

- Mécanisme d'authentification (JWT)
- Rôles utilisateurs (client / admin si implémenté)
- Middleware de protection des routes
- Gestion des accès (routes publiques vs privées)

4. Endpoints clés de l'API

- Création de compte et login
- Création, modification, annulation et validation des réservations
- Consultation du menu
- Gestion des tables et/ou créneaux (si bonus)
- Codes HTTP retournés (200, 201, 400, 401, 403, 404…)

5. Logique métier

- Vérification de la disponibilité des tables
- Attribution automatique des tables
- Gestion des statuts de réservation
- Règles bloquantes (créneau plein, réservation confirmée non modifiable, etc.)

6. Démonstration via un outil de test d'API

- Présentation via Postman (ou équivalent)
- Exemple de requêtes : sans authentification et avec token JWT
- Lecture et compréhension des réponses JSON
- Démonstration d'un cas valide et d'un cas d'erreur

Barème final du Projet (/100)

PARTIE OBLIGATOIRE – 60 points	Note
API fonctionnelle (routes CRUD, DB, auth, menu, logique de réservation)	/40
Implication personnelle sur le socle commun (code écrit, qualité, clarté des tâches réalisées)	/20

PARTIE COMPLEMENTAIRE – 40 points	Note
Fonctions avancées réalisées en groupe : **créneaux dynamiques, gestion fine des tables, rôles utilisateurs, filtres...**	/30

Valorisation individuelle : **prise d'initiative, autonomie technique, difficulté des fonctions traitées**	/10

PARTIE ORALE – 30 points	**Note**
Présentation de projet avec des critères mentionnées plus haut	/15
Démonstration	/15

Total général : /130

Détails et conseils pour les étudiants

Partie obligatoire (60pts) :

- Doit être **fonctionnelle** à 100% pour valider le projet.
- Toute personne **non investie** peut perdre jusqu'à 20 points (note individuelle).
- Attendu : **routes**, **auth**, **BDD**, **menu**, **réservations**, **documentation**, **test** via Postman.

Partie complémentaire (40pts) :

- Bonus valorisé fortement si le socle est propre.
- Les groupes motivés peuvent viser des **fonctionnalités avancées** (gestion de disponibilité, annulations, rôles, gestion multiple de tables…).
- L'individu qui **porte** une fonctionnalité technique peut être **valorisé personnellement**.

Partie orale (30pts) :

- **Chaque membre du groupe doit prendre la parole** et être capable de présenter la partie technique sur laquelle il ou elle a travaillé.
- **Présentation du projet et des choix techniques :** Présentation claire de l'architecture de l'API, de l'organisation du code, de la BDD, justification des choix techniques.
- **Démonstration de l'API :** Démonstration fluide et préparée de l'API via Postman (ou équivalent), incluant l'authentification JWT, les endpoints clés, la gestion des erreurs et l'analyse des réponses JSON.