Contrat d'API pour la Plateforme de Mobilité Distribuée

1. **API Externe** : Interfaces exposées par l'API Gateway au Front-end Angular. Toutes les requêtes du front passent par le Gateway, qui gère l'authentification, les autorisations et le routage. Les endpoints sont préfixés par `/api` pour une normalisation.
2. **API Interne** : Interfaces des microservices métier exposées au Gateway uniquement. Chaque microservice écoute sur un port distinct (ex. : users sur 3001, trips sur 3002, reservations sur 3003). Pas d'authentification locale dans les microservices ; le Gateway passe un token interne ou des headers validés.

Conventions générales :

- **Format des requêtes/réponses** : JSON pour les bodies. Codes HTTP standards (200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error).
- **Authentification** : JWT (JSON Web Token) généré par le Gateway après login. Inclus dans l'header `Authorization: Bearer <token>`. Le Gateway valide le token et ajoute des claims (userId, role) aux requêtes internes.
- **Gestion des erreurs** : Réponses d'erreur normalisées au format `{ "error": "message", "code": "ERR_CODE" }`. Le Gateway normalise les erreurs des microservices.
- **Pagination et filtres** : Pour les listes, utiliser des query params comme `page`, `limit`, `sort`.
- **Rôles** : `client` (utilisateur standard), `gestionnaire` (admin pour trajets/réservations).
- **Validation** : Schémas basés sur Joi ou équivalent dans chaque service, mais définis ici pour clarté.
- **Documentation** : Utiliser Swagger ou Postman pour tester, mais ce contrat est la référence.
- **Modifications** : Toute changement doit être validé par les trois équipes. Pas de breaking changes sans versionning (ex. : /v1/).

1. API Externe (Front-end <-> API Gateway)

Le Gateway est le point d'entrée unique. Base URL : `http://localhost:3000/api` (ou équivalent en prod). Il route vers les microservices internes sans exposer leurs URLs.

Endpoints pour Authentification (Gérés par le Gateway, avec appel au service users)

- **POST /auth/register**
  - Description : Créer un compte utilisateur.
  - Body : `{ "username": string (required, unique), "password": string (required, min 8 chars), "email": string (required, valid email), "role": string (optional, default "client") }`
  - Response : 201 - `{ "userId": string, "username": string, "role": string }`
  - Erreurs : 400 (validation), 409 (username/email exists).
  - Autorisation : Aucune (public).
- **POST /auth/login**
  - Description : Authentifier un utilisateur et obtenir un token.
  - Body : `{ "username": string (required), "password": string (required) }`
  - Response : 200 - `{ "token": string (JWT), "userId": string, "role": string }`
  - Erreurs : 401 (invalid credentials).
  - Autorisation : Aucune (public).
- **GET /auth/me**
  - Description : Obtenir les détails de l'utilisateur connecté.
  - Response : 200 - `{ "userId": string, "username": string, "email": string, "role": string }`
  - Erreurs : 401 (no token).
  - Autorisation : Token requis.

Endpoints pour Users (Routés vers service users)

- **GET /users/:userId**
  - Description : Consulter un compte utilisateur (seulement le sien pour clients).
  - Params : userId (string, required).
  - Response : 200 - `{ "userId": string, "username": string, "email": string, "role": string }`
  - Erreurs : 403 (access denied), 404 (not found).
  - Autorisation : Token requis ; gestionnaires peuvent voir tous.
- **PUT /users/:userId/role**
  - Description : Mettre à jour le rôle d'un utilisateur (gestionnaires only).
  - Params : userId (string, required).
  - Body : `{ "role": string (required, "client" ou "gestionnaire") }`
  - Response : 200 - `{ "userId": string, "role": string }`
  - Erreurs : 403 (not gestionnaire).
  - Autorisation : Token requis, rôle gestionnaire.

Endpoints pour Trips (Routés vers service trips)

- **POST /trips**
  - Description : Créer un trajet (gestionnaires only).
  - Body : `{ "departure": string (required, e.g. "Paris"), "arrival": string (required), "date": date (ISO, required), "placesAvailable": number (required, >0), "price": number (optional) }`
  - Response : 201 - `{ "tripId": string, ... (full trip details) }`
  - Erreurs : 400 (validation), 403 (not gestionnaire).
  - Autorisation : Token requis, rôle gestionnaire.
- **GET /trips**
  - Description : Rechercher des trajets disponibles (public).
  - Query params : `departure` (string), `arrival` (string), `date` (date), `page` (number, default 1), `limit` (number, default 10).
  - Response : 200 - `{ "trips": array of { "tripId": string, "departure": string, "arrival": string, "date": date,`

`"placesAvailable": number }, "total": number }`
    - Erreurs : 400 (invalid filters).
    - Autorisation : Aucune (public pour recherche).

- **GET /trips/:tripId**
    - Description : Détails d'un trajet.
    - Params : tripId (string, required).
    - Response : 200 - `{ "tripId": string, "departure": string, "arrival": string, "date": date, "placesAvailable": number, "reservationsCount": number }`
    - Erreurs : 404 (not found).
    - Autorisation : Aucune (public).

- **PUT /trips/:tripId**
    - Description : Modifier un trajet (places, etc., gestionnaires only).
    - Params : tripId (string, required).
    - Body : `{ "placesAvailable": number (optional), ... }`
    - Response : 200 - Updated trip.
    - Erreurs : 403 (not gestionnaire), 404.
    - Autorisation : Token requis, rôle gestionnaire.

Endpoints pour Reservations (Routés vers service reservations)

- **POST /reservations**
    - Description : Créer une réservation.
    - Body : `{ "tripId": string (required), "userId": string (from token), "places": number (required, >0) }`
    - Response : 201 - `{ "reservationId": string, "status": "pending", "tripId": string, "places": number }`
    - Erreurs : 400 (places exceed available), 404 (trip not found).
    - Autorisation : Token requis (client ou gestionnaire).

- **GET /reservations**
    - Description : Liste des réservations (pour utilisateur ou par trajet pour gestionnaires).
    - Query params : `userId` (string, optional pour gestionnaires), `tripId` (string, optional), `status` (string), `page` (number), `limit` (number).
    - Response : 200 - `{ "reservations": array of { "reservationId": string, "tripId": string, "userId": string, "places": number, "status": string }, "total": number }`
    - Erreurs : 403 (access denied).
    - Autorisation : Token requis.

- **PUT /reservations/:reservationId/cancel**
    - Description : Annuler une réservation (libère places).
    - Params : reservationId (string, required).
    - Response : 200 - `{ "reservationId": string, "status": "canceled" }`
    - Erreurs : 404, 403 (not owner or gestionnaire), 400 (already canceled).
    - Autorisation : Token requis.

- **PUT /reservations/:reservationId/status**
    - Description : Mettre à jour le statut (gestionnaires only, ex. confirm/expire).
    - Params : reservationId (string, required).
    - Body : `{ "status": string (required, "confirmed" | "expired") }`
    - Response : 200 - Updated reservation.
    - Erreurs : 403, 400 (invalid status transition).
    - Autorisation : Token requis, rôle gestionnaire.

2. API Interne (API Gateway <-> Microservices)

Ces APIs sont internes, non exposées. Le Gateway ajoute des headers comme `X-User-Id` et `X-User-Role` après validation du token. Base URLs : ex. `http://users-service:3001/` pour users.

Service Users (Port 3001)

- **POST /users** : Même que externe /auth/register, mais sans password hashing (Gateway hash avant).
- **GET /users/:userId** : Même que externe.
- **PUT /users/:userId/role** : Même que externe.
- **POST /users/authenticate** : Interne pour login - Body: `{ "username": string, "password": string }` , Response: `{ "userId": string, "role": string }` ou 401.

Service Trips (Port 3002)

- **POST /trips** : Même que externe.
- **GET /trips** : Même que externe.
- **GET /trips/:tripId** : Même que externe, plus interne: inclut `reservationsCount` via appel à reservations si needed (mais éviter couplage ; peut-être event sourcing plus tard).
- **PUT /trips/:tripId** : Même que externe.
- **PUT /trips/:tripId/reserve** : Interne pour réserver places - Body: `{ "places": number }` , Response: 200 si OK, 400 si insuffisant. (Appelé par reservations service si couplage sync).

Service Reservations (Port 3003)

- **POST /reservations** : Même que externe, mais appelle trips pour vérifier/verrouiller places (transaction simple, pas distribuée).
- **GET /reservations** : Même que externe.
- **PUT /reservations/:reservationId/cancel** : Même que externe, appelle trips pour libérer places.
- **PUT /reservations/:reservationId/status** : Même que externe, peut notifier trips si needed.

Flux Exemple : Création de Réservation

1. Front -> Gateway: POST /reservations avec token.
2. Gateway valide token, ajoute X-User-Id, route à reservations-service.
3. Reservations-service -> Trips-service (interne): Vérifie places via GET /trips/:tripId, puis PUT /trips/:tripId/reserve.
4. Si OK, persiste réservation, retourne à Gateway.
5. Gateway normalise et retourne au Front.