User Management Service – Microservice Documentation

1. Overview

The **User Management Service** is responsible for:

- User registration and login (phone, email, social login)
- OTP verification (optional)
- Profile management (details, emergency contacts, documents)
- Role-based access control (Passenger, Driver, Admin, Dispatcher, Finance)
- Multi-language support

This service is a standalone microservice and communicates with other services (Ride Booking, Payment, etc.) through **REST APIs**.

2. Backend

2.1 Responsibilities

- Handle user CRUD operations
- Authenticate users via JWT
- Role-based access control
- Password hashing and verification
- Provide APIs for frontend consumption

2.2 Tech Stack

- **Node.js + Express**
- **Sequelize ORM** with **MySQL**
- **JWT** for authentication
- **Bcrypt** for password hashing
- **Dotenv** for configuration

2.3 Project Structure

```
user-management-service-backend/
├── src
│   ├── config
│   │   └── database.js          # Sequelize DB connection
│   ├── controllers
│   │   └── userController.js    # Request handlers
│   ├── models
│   │   └── user.js              # Sequelize User model
│   ├── routes
│   │   └── userRoutes.js        # REST API routes
│   ├── services
│   │   └── userService.js       # Business logic
│   ├── middleware
│   │   └── authMiddleware.js    # JWT authentication
│   ├── utils
│   │   └── password.js          # Password hash/compare helper
│   └── app.js                   # Express app entry
├── .env                         # Environment variables
```

```
├── package.json
└── server.js                # Start server
```

2.4 Example APIs

Method	Endpoint	Description
POST	/api/users/register	Register a new user
POST	/api/users/login	Login user, return JWT
GET	/api/users/:id	Get user profile
PUT	/api/users/:id	Update user profile

---

3. Frontend

3.1 Responsibilities

- Render UI for registration, login, and profile management
- Validate user input
- Consume backend APIs for CRUD operations
- Handle JWT token in local storage or cookies
- Provide role-based rendering

3.2 Tech Stack

- React.js
- Axios or Fetch API for HTTP requests
- Context/Redux for global state (optional)
- i18n library for multi-language support

3.3 Project Structure

user-management-service-frontend/

```
├── public/
│   └── index.html
├── src
│   ├── app
│   │   └── store.js             # Redux store configuration
│   ├── features
│   │   └── auth
│   │       ├── authSlice.js       # Redux slice for authentication
│   │       └── authThunks.js      # Async thunks for login/register/profile
```

```
│   ├── components
│   │   ├── RegisterForm.jsx
│   │   ├── LoginForm.jsx
│   │   ├── Profile.jsx
│   │   └── Navbar.jsx
│   ├── pages
│   │   ├── RegisterPage.jsx
│   │   ├── LoginPage.jsx
│   │   └── ProfilePage.jsx
│   ├── services
│   │   └── api.js              # Axios or fetch API calls to backend
│   ├── utils
│   │   └── validators.js       # Form validation functions
│   ├── App.jsx
│   └── index.js                # Entry point (Provider wrapping store)
├── package.json
└── .env
```

4. Relation Between Backend & Frontend

- **Frontend** sends HTTP requests to **backend API endpoints**.
- **Backend** validates requests, interacts with **MySQL via Sequelize**, and returns responses (JSON).
- JWT tokens are used for authentication; the frontend stores the token in **local storage** or **cookies** and sends it in headers for protected routes.
- Backend handles **role-based access** for Admin, Passenger, Driver, etc., while frontend conditionally renders UI components based on user role.

5. Database

- **MySQL** database named user_management.
- **Users table** (Sequelize model) with fields:

- id (INT, PK)
- username (VARCHAR)
- email (VARCHAR, unique)
- password (VARCHAR)
- role (ENUM: user, admin)
- created_at (TIMESTAMP)

```
mysql>  DESCRIBE users;
+-----------------+-----------------------------------------------------------------+------+-----+-----------+----------------+
| Field           | Type                                                            | Null | Key | Default   | Extra          |
+-----------------+-----------------------------------------------------------------+------+-----+-----------+----------------+
| id              | bigint                                                          | NO   | PRI | NULL      | auto_increment |
| phone           | varchar(20)                                                     | YES  | UNI | NULL      |                |
| email           | varchar(100)                                                    | YES  | UNI | NULL      |                |
| password_hash   | varchar(255)                                                    | YES  |     | NULL      |                |
| social_provider | enum('google','facebook','apple','none')                        | NO   |     | none      |                |
| role            | enum('passenger','driver','dispatcher','admin','finance')       | NO   |     | passenger |                |
| language        | enum('am','en','om','ti','af')                                  | NO   |     | en        |                |
| is_active       | tinyint(1)                                                      | NO   |     | 1         |                |
| created_at      | datetime                                                        | NO   |     | NULL      |                |
| updated_at      | datetime                                                        | NO   |     | NULL      |                |
+-----------------+-----------------------------------------------------------------+------+-----+-----------+----------------+
10 rows in set (0.00 sec)

mysql> DESCRIBE otps;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | bigint      | NO   | PRI | NULL    | auto_increment |
| user_id    | bigint      | NO   | MUL | NULL    |                |
| code       | varchar(10) | NO   |     | NULL    |                |
| expires_at | datetime    | NO   |     | NULL    |                |
| verified   | tinyint(1)  | NO   |     | 0       |                |
+------------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> DESCRIBE profiles;
+------------------+----------------------+------+-----+---------+----------------+
| Field            | Type                 | Null | Key | Default | Extra          |
+------------------+----------------------+------+-----+---------+----------------+
| id               | bigint               | NO   | PRI | NULL    | auto_increment |
| user_id          | bigint               | NO   | MUL | NULL    |                |
| first_name       | varchar(50)          | YES  |     | NULL    |                |
| last_name        | varchar(50)          | YES  |     | NULL    |                |
| gender           | enum('male','female') | YES  |     | NULL    |                |
| dob              | date                 | YES  |     | NULL    |                |
| emergency_contact | json                 | YES  |     | NULL    |                |
| documents        | json                 | YES  |     | NULL    |                |
| created_at       | datetime             | NO   |     | NULL    |                |
| updated_at       | datetime             | NO   |     | NULL    |                |
```

Base

- Base URL: http://localhost:4000
- Prefix: /api
- Auth: Bearer JWT on protected endpoints

Content-Type: application/json

Public

- POST /api/auth/register: Create user (email, optional password, role, language). Also creates empty
- profile. Returns user + JWT.
- POST /api/auth/login: Login with email/password. Returns user + JWT.
- Protected (Bearer token)
- GET /api/profiles/me: Fetch the authenticated user’s profile.

- PUT /api/profiles: Update profile fields (first_name, last_name, gender, dob).

OTP (dev/testing)

- POST /api/otps/request: Generate OTP for a user. Body: { user_id }. Returns OTP code (dev only).
- POST /api/otps/request: Generate OTP for a user. Body: { user_id }. Returns OTP code (dev only).
- POST /api/otps/verify: Verify OTP. Body: { user_id, code }. Returns { verified: true }.

.env file

```
# General

NODE_ENV=development

PORT=4000

# JWT

JWT_SECRET=

JWT_EXPIRES_IN=7d

# Security

BCRYPT_SALT_ROUNDS=10

# Database

DB_HOST=127.0.0.1

DB_PORT=3306

DB_USER=root

DB_PASSWORD=

DB_NAME=user_service

DB_LOGGING=false
```