

# ParkEasy Backend Project Documentation

## Project Overview

ParkEasy is a backend API server for a smart parking management system. It manages users, buildings, parking spots, and spot reservations with authentication and admin controls. The backend is built with Node.js, Express, and MongoDB, using JWT for authentication and role-based access control.

## File Structure and Purpose

ParkEasy/	
├─ server.js	# Entry point; sets up Express app, routes, DB connection, and starts server
├─ routes/	# Express route definitions
│   ├─ authRoutes.js	# Routes for user signup, login, and profile (authentication)
│   ├─ userRoutes.js	# Routes related to user data (currently profile)
│   ├─ buildingRoutes.js	# Routes for managing buildings (CRUD, admin only for some)
│   ├─ spotRoutes.js	# Routes for managing parking spots by building (CRUD, admin only)
│   └─ reservationRoutes.js	# Routes for parking spot reservations (reserve spot, get reservations)
├─ controllers/	# Controller functions implementing the logic for each route
│   ├─ authController.js	# Signup, login, and get profile with JWT token creation
│   ├─ buildingController.js	# Create, read, update, delete buildings
│   ├─ spotController.js	# Create, read, delete parking spots linked to buildings
│   └─ reservationController.js	# Reserve spots, get reservation info with user and spot details
├─ middleware/	# Middleware for authentication and authorization
│   └─ auth.js	# JWT token verification and admin-only route guard
├─ models/	# Mongoose schema models representing database structure
│   ├─ User.js	# User schema with fields like name, email, password (hashed), role, etc.
│   ├─ Building.js	# Building schema with name and abbreviation
│   └─ Spot.js	# Spot schema linked to a building, with spot number and availability
│   └─ Reservation.js	# Reservation schema linked to user and spot, with reservation period
└─ .env	# Environment variables (MongoDB URI, JWT secret, port, etc.)

# Functionalities

## Authentication & User Management

- Signup: Registers users with hashed passwords, prevents signup as admin.
- Login: Authenticates users, validates roles, returns JWT tokens.
- Get Profile: Returns user profile except password (protected route).
- Role-based access: "user" and "admin" roles with admin-only middleware protection.

## Building Management

- Admins can add, edit, fetch, and delete buildings.
- Buildings have unique abbreviations.

## Parking Spot Management

- Admins can add and delete parking spots linked to buildings.
- Spots have unique numbers per building and availability status.
- Any authorized user can fetch spots by building.

## Reservation Management

- Users can reserve available spots for a time range; spot availability toggled.
  - Reservations store user, spot, start and end times.
  - Users can fetch reservations filtered by building (with user and spot details).
- 

## How It Works: Overview

1. Server Initialization:
  - `server.js` sets up Express with JSON parsing, CORS, connects to MongoDB using MONGO\_URI from `.env`.
  - Registers all routes under `/api/...` prefixes.
  - Starts listening on configured PORT.
2. Authentication:
  - Signup and login routes create and return JWT tokens with user id and role.
  - `auth.js` middleware verifies tokens on protected routes and allows only admins for restricted routes.
3. Data Models:
  - MongoDB stores users, buildings, spots, and reservations.
  - Schema fields maintain reference integrity (e.g., spot references building, reservation references spot and user).
4. Route-Controller Flow:
  - User requests hit routes in `routes/`.
  - Middleware like auth verify tokens and permissions.
  - Controller logic performs database queries and returns JSON responses.

---

## Environment Variables ( `.env` )

- `MONGO_URI`: MongoDB database connection string.
  - `JWT_SECRET`: Secret key to sign JWT tokens.
  - `JWT_EXPIRES_IN`: Token expiry duration (e.g., "7d").
  - `PORT`: Server port (default 5000).
- 

## Development Notes

- Use nodemon for live server reload during development.
  - Passwords are securely hashed with bcrypt.
  - JWT tokens are used for stateless authentication.
  - Admin routes are protected by role check middleware.
  - Errors generally return HTTP status codes with JSON messages.
  - Spot reservations update the spot availability to prevent double booking.
- 

## How Frontend Can Integrate

- Base API URL: `http://localhost:5000/api`
- Use `/auth/signup` and `/auth/login` for user auth workflows.
- Use the returned JWT in Authorization header: `Bearer <token>` for protected calls.
- Fetch buildings, spots, and reservations to display data.
- Post reservation requests with valid JWT and spot IDs.