EV Co-ownership & Cost-sharing System

Tài liệu Kỹ thuật Đầy đủ

---

1. TỔNG QUAN DỰ ÁN

1.1 Thông tin dự án

- **Tên dự án**: EV Co-ownership & Cost-sharing System
- **Mô tả**: Hệ thống quản lý đồng sở hữu và chia sẻ chi phí xe điện
- **Thời gian**: 8 tuần (400+ giờ)
- **Team size**: 4 sinh viên
- **Architecture**: Microservices

1.2 Actors

- **Co-owner**: Chủ xe đồng sở hữu
- **Staff**: Nhân viên vận hành
- **Admin**: Quản trị viên hệ thống

1.3 Công nghệ sử dụng

Frontend

- React.js + Vite
- Tailwind CSS
- Axios (HTTP client)
- React Router v6
- React Query (data fetching)
- Zustand (state management)

Backend

- Node.js + Express.js
- Sequelize ORM + Sequelize CLI
- JWT Authentication
- Socket.io (real-time)

Infrastructure

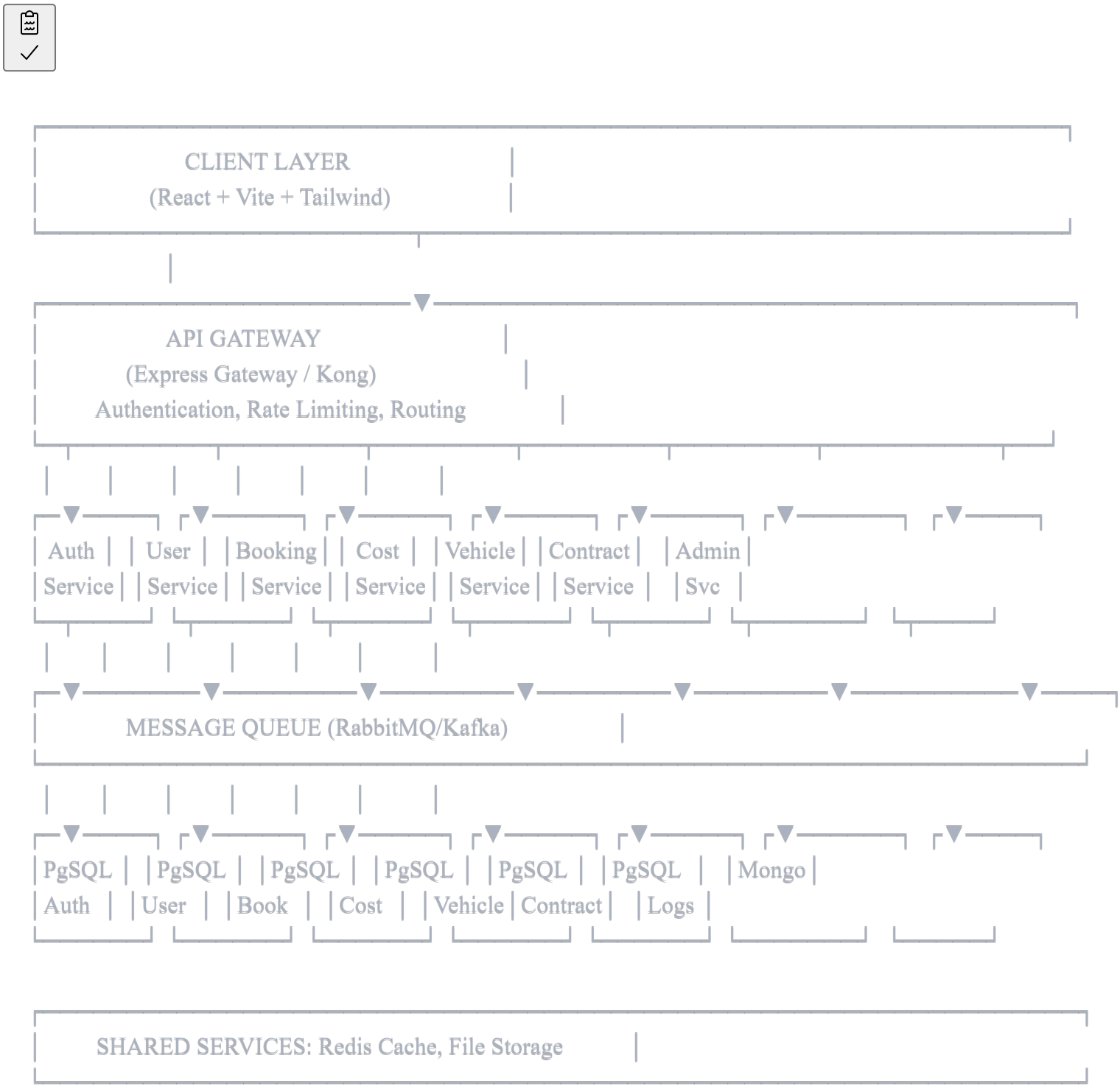
- Docker & Docker Compose
- API Gateway (Express Gateway hoặc Kong)
- Redis (Caching & Session)
- RabbitMQ/Kafka (Message Queue)
- Nginx (Reverse Proxy)

Databases

- PostgreSQL (User, Auth, Contract)
- PostgreSQL (Booking, Schedule)
- PostgreSQL (Payment, Cost)
- MongoDB (Logs, Analytics)

2. KIẾN TRÚC HỆ THỐNG MICROSERVICES

2.1 Sơ đồ tổng quan



2.2 Các Microservices

1. Auth Service (Port: 3001)

- Đăng ký, đăng nhập
- JWT token generation/validation
- Password reset
- KYC verification (CMND/CCCD, GPLX)

2. User Service (Port: 3002)

- Quản lý profile user
- Quản lý ownership ratio
- Quản lý group membership
- User preferences

3. Booking Service (Port: 3003)

- Đặt lịch sử dụng xe
- Calendar management
- Conflict resolution
- Priority algorithm
- Check-in/Check-out

4. Cost Service (Port: 3004)

- Chi phí tracking
- Cost splitting algorithms
- Payment processing
- Invoice generation
- Monthly/Quarterly reports

5. Vehicle Service (Port: 3005)

- Quản lý xe
- Maintenance scheduling
- Vehicle status
- Service history

6. Contract Service (Port: 3006)

- E-contract management
- Legal document storage
- Co-ownership agreements
- Digital signatures

7. Admin Service (Port: 3007)

- Dashboard & Analytics
- Dispute management
- System configuration
- Staff management

8. Notification Service (Port: 3008)

- Email notifications
- Push notifications
- SMS (optional)
- In-app notifications

---

3. THIẾT KẾ DATABASE

3.1 Database Strategy

Mỗi service có database riêng (Database per Service Pattern)

Service	Database	Purpose
Auth Service	PostgreSQL	Users, Tokens, Sessions
User Service	PostgreSQL	Profiles, Groups, Ownership
Booking Service	PostgreSQL	Bookings, Schedules
Cost Service	PostgreSQL	Costs, Payments, Invoices
Vehicle Service	PostgreSQL	Vehicles, Maintenance
Contract Service	PostgreSQL	Contracts, Documents
Admin Service	PostgreSQL	Staff, Disputes
Logs/Analytics	MongoDB	System logs, Analytics

3.2 Schema chi tiết

AUTH DATABASE

sql

```sql
-- users table (core authentication)
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  email VARCHAR(255) UNIQUE NOT NULL,
  phone VARCHAR(20) UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('co_owner', 'staff', 'admin') NOT NULL,
  is_verified BOOLEAN DEFAULT FALSE,
  is_active BOOLEAN DEFAULT TRUE,
  last_login_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- kyc_verifications
CREATE TABLE kyc_verifications (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id),
  id_card_number VARCHAR(20) UNIQUE,
  id_card_front_url VARCHAR(500),
  id_card_back_url VARCHAR(500),
  driver_license_number VARCHAR(20),
  driver_license_url VARCHAR(500),
  verification_status ENUM('pending', 'approved', 'rejected') DEFAULT 'pending',
  verified_by UUID REFERENCES users(id),
  verified_at TIMESTAMP,
  rejection_reason TEXT,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- refresh_tokens
CREATE TABLE refresh_tokens (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id),
  token VARCHAR(500) NOT NULL,
  expires_at TIMESTAMP NOT NULL,
  is_revoked BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP DEFAULT NOW()
);

-- password_resets
CREATE TABLE password_resets (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
```

```
    user_id UUID REFERENCES users(id),
    reset_token VARCHAR(255) NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    used BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);
```

USER DATABASE

sql

```sql
-- user_profiles
CREATE TABLE user_profiles (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID UNIQUE NOT NULL, -- Reference to auth.users.id
  full_name VARCHAR(255) NOT NULL,
  date_of_birth DATE,
  address TEXT,
  avatar_url VARCHAR(500),
  bio TEXT,
  preferences JSONB,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- co_ownership_groups
CREATE TABLE co_ownership_groups (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_name VARCHAR(255) NOT NULL,
  description TEXT,
  created_by UUID NOT NULL,
  group_fund_balance DECIMAL(15, 2) DEFAULT 0,
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- group_members
CREATE TABLE group_members (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID REFERENCES co_ownership_groups(id),
  user_id UUID NOT NULL,
  ownership_percentage DECIMAL(5, 2) NOT NULL CHECK (ownership_percentage > 0 AND ownership_percentag
  role ENUM('admin', 'member') DEFAULT 'member',
  joined_at TIMESTAMP DEFAULT NOW(),
  left_at TIMESTAMP,
  is_active BOOLEAN DEFAULT TRUE,
  UNIQUE(group_id, user_id)
);

-- group_fund_transactions
CREATE TABLE group_fund_transactions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID REFERENCES co_ownership_groups(id),
  transaction_type ENUM('deposit', 'withdrawal', 'allocation') NOT NULL,
```

```
  amount DECIMAL(15, 2) NOT NULL,
  description TEXT,
  created_by UUID NOT NULL,
  created_at TIMESTAMP DEFAULT NOW()
);

-- group_votes
CREATE TABLE group_votes (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID REFERENCES co_ownership_groups(id),
  title VARCHAR(255) NOT NULL,
  description TEXT,
  vote_type ENUM('upgrade', 'maintenance', 'insurance', 'sell_vehicle', 'other'),
  status ENUM('open', 'closed', 'executed') DEFAULT 'open',
  deadline TIMESTAMP,
  created_by UUID NOT NULL,
  created_at TIMESTAMP DEFAULT NOW(),
  closed_at TIMESTAMP
);

-- vote_options
CREATE TABLE vote_options (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  vote_id UUID REFERENCES group_votes(id) ON DELETE CASCADE,
  option_text VARCHAR(255) NOT NULL,
  vote_count INTEGER DEFAULT 0
);

-- user_votes
CREATE TABLE user_votes (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  vote_id UUID REFERENCES group_votes(id),
  user_id UUID NOT NULL,
  option_id UUID REFERENCES vote_options(id),
  voted_at TIMESTAMP DEFAULT NOW(),
  UNIQUE(vote_id, user_id)
);
```

BOOKING DATABASE

sql

```sql
-- vehicles (replicated data for booking context)
CREATE TABLE vehicles (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID NOT NULL,
  vehicle_name VARCHAR(255) NOT NULL,
  license_plate VARCHAR(20) UNIQUE NOT NULL,
  status ENUM('available', 'in_use', 'maintenance', 'unavailable') DEFAULT 'available',
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- bookings
CREATE TABLE bookings (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  vehicle_id UUID REFERENCES vehicles(id),
  user_id UUID NOT NULL,
  group_id UUID NOT NULL,
  start_time TIMESTAMP NOT NULL,
  end_time TIMESTAMP NOT NULL,
  status ENUM('pending', 'confirmed', 'in_progress', 'completed', 'cancelled') DEFAULT 'pending',
  priority_score INTEGER DEFAULT 0,
  purpose TEXT,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- check_in_out_logs
CREATE TABLE check_in_out_logs (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  booking_id UUID REFERENCES bookings(id),
  action_type ENUM('check_in', 'check_out') NOT NULL,
  odometer_reading INTEGER,
  fuel_level DECIMAL(5, 2), -- battery percentage for EV
  images JSONB, -- array of image URLs
  notes TEXT,
  qr_code VARCHAR(255),
  digital_signature TEXT,
  performed_by UUID NOT NULL,
  performed_at TIMESTAMP DEFAULT NOW()
);

-- booking_conflicts
CREATE TABLE booking_conflicts (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
```

```
  booking_id_1 UUID REFERENCES bookings(id),
  booking_id_2 UUID REFERENCES bookings(id),
  conflict_type ENUM('time_overlap', 'vehicle_unavailable') NOT NULL,
  resolved BOOLEAN DEFAULT FALSE,
  resolution_note TEXT,
  resolved_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW()
);
```

COST DATABASE

sql

```sql
-- cost_categories
CREATE TABLE cost_categories (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  category_name VARCHAR(100) NOT NULL UNIQUE,
  description TEXT,
  is_recurring BOOLEAN DEFAULT FALSE
);

-- costs
CREATE TABLE costs (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID NOT NULL,
  vehicle_id UUID NOT NULL,
  category_id UUID REFERENCES cost_categories(id),
  cost_name VARCHAR(255) NOT NULL,
  total_amount DECIMAL(15, 2) NOT NULL,
  split_type ENUM('ownership_ratio', 'usage_based', 'equal', 'custom') DEFAULT 'ownership_ratio',
  cost_date DATE NOT NULL,
  description TEXT,
  receipt_url VARCHAR(500),
  created_by UUID NOT NULL,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- cost_splits
CREATE TABLE cost_splits (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  cost_id UUID REFERENCES costs(id) ON DELETE CASCADE,
  user_id UUID NOT NULL,
  split_amount DECIMAL(15, 2) NOT NULL,
  payment_status ENUM('pending', 'paid', 'overdue') DEFAULT 'pending',
  paid_at TIMESTAMP
);

-- payments
CREATE TABLE payments (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  cost_split_id UUID REFERENCES cost_splits(id),
  user_id UUID NOT NULL,
  amount DECIMAL(15, 2) NOT NULL,
  payment_method ENUM('e_wallet', 'bank_transfer', 'credit_card', 'cash') NOT NULL,
  transaction_id VARCHAR(255),
  payment_status ENUM('pending', 'completed', 'failed', 'refunded') DEFAULT 'pending',
```

```
  payment_date TIMESTAMP DEFAULT NOW(),
  notes TEXT
);

-- invoices
CREATE TABLE invoices (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID NOT NULL,
  invoice_number VARCHAR(50) UNIQUE NOT NULL,
  invoice_period_start DATE NOT NULL,
  invoice_period_end DATE NOT NULL,
  total_amount DECIMAL(15, 2) NOT NULL,
  generated_at TIMESTAMP DEFAULT NOW(),
  pdf_url VARCHAR(500)
);

-- invoice_items
CREATE TABLE invoice_items (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  invoice_id UUID REFERENCES invoices(id) ON DELETE CASCADE,
  cost_id UUID REFERENCES costs(id),
  item_description VARCHAR(255),
  amount DECIMAL(15, 2) NOT NULL
);
```

VEHICLE DATABASE

sql

```sql
-- vehicles
CREATE TABLE vehicles (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID NOT NULL,
  vehicle_name VARCHAR(255) NOT NULL,
  brand VARCHAR(100),
  model VARCHAR(100),
  year INTEGER,
  license_plate VARCHAR(20) UNIQUE NOT NULL,
  vin VARCHAR(17) UNIQUE,
  color VARCHAR(50),
  battery_capacity_kwh DECIMAL(6, 2),
  current_odometer INTEGER DEFAULT 0,
  status ENUM('available', 'in_use', 'maintenance', 'unavailable') DEFAULT 'available',
  purchase_date DATE,
  purchase_price DECIMAL(15, 2),
  images JSONB,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- maintenance_schedules
CREATE TABLE maintenance_schedules (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  vehicle_id UUID REFERENCES vehicles(id),
  maintenance_type VARCHAR(100) NOT NULL,
  scheduled_date DATE NOT NULL,
  odometer_at_schedule INTEGER,
  status ENUM('scheduled', 'in_progress', 'completed', 'cancelled') DEFAULT 'scheduled',
  cost DECIMAL(15, 2),
  notes TEXT,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- maintenance_history
CREATE TABLE maintenance_history (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  vehicle_id UUID REFERENCES vehicles(id),
  maintenance_schedule_id UUID REFERENCES maintenance_schedules(id),
  maintenance_type VARCHAR(100) NOT NULL,
  performed_date DATE NOT NULL,
  odometer_reading INTEGER,
  cost DECIMAL(15, 2),
```

```sql
  service_provider VARCHAR(255),
  description TEXT,
  receipt_url VARCHAR(500),
  performed_by UUID NOT NULL,
  created_at TIMESTAMP DEFAULT NOW()
);

-- vehicle_insurance
CREATE TABLE vehicle_insurance (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  vehicle_id UUID REFERENCES vehicles(id),
  insurance_provider VARCHAR(255) NOT NULL,
  policy_number VARCHAR(100) UNIQUE NOT NULL,
  coverage_type VARCHAR(100),
  premium_amount DECIMAL(15, 2) NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  is_active BOOLEAN DEFAULT TRUE,
  document_url VARCHAR(500),
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- charging_sessions
CREATE TABLE charging_sessions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  vehicle_id UUID REFERENCES vehicles(id),
  user_id UUID NOT NULL,
  charging_station_location VARCHAR(255),
  start_time TIMESTAMP NOT NULL,
  end_time TIMESTAMP,
  energy_consumed_kwh DECIMAL(8, 2),
  cost DECIMAL(10, 2),
  payment_method VARCHAR(50),
  created_at TIMESTAMP DEFAULT NOW()
);
```

CONTRACT DATABASE

sql

```sql
-- contracts
CREATE TABLE contracts (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID NOT NULL,
  contract_type ENUM('co_ownership', 'amendment', 'termination') NOT NULL,
  contract_number VARCHAR(50) UNIQUE NOT NULL,
  title VARCHAR(255) NOT NULL,
  content TEXT NOT NULL,
  status ENUM('draft', 'pending_signatures', 'active', 'expired', 'terminated') DEFAULT 'draft',
  created_by UUID NOT NULL,
  effective_date DATE,
  expiry_date DATE,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- contract_parties
CREATE TABLE contract_parties (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  contract_id UUID REFERENCES contracts(id) ON DELETE CASCADE,
  user_id UUID NOT NULL,
  party_role ENUM('owner', 'co_owner', 'witness') NOT NULL,
  ownership_percentage DECIMAL(5, 2),
  has_signed BOOLEAN DEFAULT FALSE,
  signed_at TIMESTAMP,
  signature_data TEXT,
  UNIQUE(contract_id, user_id)
);

-- contract_documents
CREATE TABLE contract_documents (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  contract_id UUID REFERENCES contracts(id) ON DELETE CASCADE,
  document_name VARCHAR(255) NOT NULL,
  document_type VARCHAR(50),
  file_url VARCHAR(500) NOT NULL,
  uploaded_by UUID NOT NULL,
  uploaded_at TIMESTAMP DEFAULT NOW()
);

-- contract_amendments
CREATE TABLE contract_amendments (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  original_contract_id UUID REFERENCES contracts(id),
```

```
    amendment_contract_id UUID REFERENCES contracts(id),
    amendment_reason TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);
```

ADMIN DATABASE

sql

```sql
-- staff_profiles
CREATE TABLE staff_profiles (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID UNIQUE NOT NULL,
  employee_id VARCHAR(50) UNIQUE NOT NULL,
  position VARCHAR(100),
  department VARCHAR(100),
  hire_date DATE,
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- disputes
CREATE TABLE disputes (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  group_id UUID NOT NULL,
  dispute_type ENUM('booking_conflict', 'cost_dispute', 'damage_claim', 'other') NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT NOT NULL,
  filed_by UUID NOT NULL,
  against_user UUID,
  status ENUM('open', 'investigating', 'resolved', 'closed') DEFAULT 'open',
  priority ENUM('low', 'medium', 'high', 'critical') DEFAULT 'medium',
  assigned_to UUID,
  resolution TEXT,
  resolved_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- dispute_messages
CREATE TABLE dispute_messages (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  dispute_id UUID REFERENCES disputes(id) ON DELETE CASCADE,
  sender_id UUID NOT NULL,
  message TEXT NOT NULL,
  attachments JSONB,
  sent_at TIMESTAMP DEFAULT NOW()
);

-- system_settings
CREATE TABLE system_settings (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
```

```
  setting_key VARCHAR(100) UNIQUE NOT NULL,
  setting_value TEXT NOT NULL,
  data_type ENUM('string', 'number', 'boolean', 'json') DEFAULT 'string',
  description TEXT,
  updated_by UUID NOT NULL,
  updated_at TIMESTAMP DEFAULT NOW()
);

-- audit_logs
CREATE TABLE audit_logs (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID,
  action VARCHAR(100) NOT NULL,
  entity_type VARCHAR(100),
  entity_id UUID,
  old_values JSONB,
  new_values JSONB,
  ip_address VARCHAR(45),
  user_agent TEXT,
  created_at TIMESTAMP DEFAULT NOW()
);
```

MONGODB - Logs & Analytics

javascript

```
// analytics_events collection
{
  _id: ObjectId,
  event_type: String, // 'booking_created', 'cost_added', 'login', etc.
  user_id: String,
  group_id: String,
  metadata: Object,
  timestamp: Date,
  ip_address: String,
  user_agent: String
}

// system_logs collection
{
  _id: ObjectId,
  service_name: String,
  log_level: String, // 'info', 'warn', 'error', 'debug'
  message: String,
  stack_trace: String,
  metadata: Object,
  timestamp: Date
}

// usage_statistics collection (aggregated data)
{
  _id: ObjectId,
  group_id: String,
  user_id: String,
  vehicle_id: String,
  period_start: Date,
  period_end: Date,
  total_bookings: Number,
  total_hours_used: Number,
  total_distance_km: Number,
  total_cost_paid: Number,
  usage_percentage: Number
}
```

---

4. CẤU TRÚC DỰ ÁN

4.1 Tổng quan thư mục

```
ev-coownership-system/
├── docs/                       # Tài liệu
│   ├── confluence/             # Tài liệu Confluence
│   ├── api/                    # API documentation
│   └── diagrams/               # Sơ đồ kiến trúc
├── frontend/                   # React Frontend
├── backend/                    # Backend services
│   ├── api-gateway/            # API Gateway
│   ├── auth-service/           # Auth Service
│   ├── user-service/           # User Service
│   ├── booking-service/        # Booking Service
│   ├── cost-service/           # Cost Service
│   ├── vehicle-service/        # Vehicle Service
│   ├── contract-service/       # Contract Service
│   ├── admin-service/          # Admin Service
│   ├── notification-service/   # Notification Service
│   └── shared/                 # Shared libraries
│       ├── utils/
│       ├── middleware/
│       └── config/
├── infrastructure/             # Infrastructure configs
│   ├── docker/
│   ├── kubernetes/
│   └── nginx/
├── scripts/                    # Deployment & utility scripts
├── .github/                    # GitHub Actions & templates
│   └── workflows/
├── docker-compose.yml
├── docker-compose.dev.yml
├── .env.example
├── .gitignore
└── README.md
```

4.2 Frontend Structure (React + Vite)

```
frontend/
├── public/
│   ├── favicon.ico
│   └── assets/
├── src/
│   ├── assets/              # Images, fonts, etc.
│   ├── components/          # Reusable components
│   │   ├── common/          # Button, Input, Modal, etc.
│   │   ├── layout/          # Header, Sidebar, Footer
│   │   ├── booking/         # Booking related components
│   │   ├── cost/            # Cost related components
│   │   └── dashboard/       # Dashboard widgets
│   ├── pages/               # Page components
│   │   ├── auth/
│   │   │   ├── Login.jsx
│   │   │   ├── Register.jsx
│   │   │   └── KYCVerification.jsx
│   │   ├── dashboard/
│   │   │   └── Dashboard.jsx
│   │   ├── booking/
│   │   │   ├── BookingList.jsx
│   │   │   ├── CreateBooking.jsx
│   │   │   └── BookingCalendar.jsx
│   │   ├── cost/
│   │   │   ├── CostList.jsx
│   │   │   ├── AddCost.jsx
│   │   │   └── CostReport.jsx
│   │   ├── group/
│   │   │   ├── GroupDetail.jsx
│   │   │   ├── GroupMembers.jsx
│   │   │   └── GroupVoting.jsx
│   │   ├── vehicle/
│   │   │   ├── VehicleList.jsx
│   │   │   └── VehicleDetail.jsx
│   │   ├── contract/
│   │   │   ├── ContractList.jsx
│   │   │   └── ContractDetail.jsx
│   │   └── admin/
│   │       ├── AdminDashboard.jsx
│   │       ├── DisputeManagement.jsx
│   │       └── StaffManagement.jsx
│   ├── services/            # API services
│   │   ├── api.js           # Axios instance
│   │   ├── authService.js
```

```
│   │   ├── bookingService.js
│   │   ├── costService.js
│   │   └── ...
│   ├── store/              # State management (Zustand)
│   │   ├── authStore.js
│   │   ├── bookingStore.js
│   │   └── ...
│   ├── hooks/              # Custom React hooks
│   │   ├── useAuth.js
│   │   ├── useBooking.js
│   │   └── ...
│   ├── utils/              # Utility functions
│   │   ├── formatters.js
│   │   ├── validators.js
│   │   └── constants.js
│   ├── routes/             # Route configuration
│   │   ├── index.jsx
│   │   ├── PrivateRoute.jsx
│   │   └── RoleBasedRoute.jsx
│   ├── styles/             # Global styles
│   │   └── index.css
│   ├── App.jsx
│   ├── main.jsx
│   └── vite-env.d.ts
├── .env.development
├── .env.production
├── .eslintrc.cjs
├── .prettierrc
├── index.html
├── package.json
├── postcss.config.js
├── tailwind.config.js
├── vite.config.js
└── README.md
```

4.3 Backend Service Structure (Ví dụ: Auth Service)

```
auth-service/
├── src/
│   ├── config/
│   │   ├── database.js          # Sequelize config
│   │   ├── redis.js             # Redis config
│   │   └── env.js               # Environment variables
│   ├── models/                  # Sequelize models
│   │   ├── index.js
│   │   ├── User.js
│   │   ├── KYCVerification.js
│   │   ├── RefreshToken.js
│   │   └── PasswordReset.js
│   ├── controllers/             # Request handlers
│   │   ├── authController.js
│   │   ├── kycController.js
│   │   └── tokenController.js
│   ├── services/                # Business logic
│   │   ├── authService.js
│   │   ├── kycService.js
│   │   ├── tokenService.js
│   │   └── emailService.js
│   ├── middlewares/             # Express middlewares
│   │   ├── authMiddleware.js
│   │   ├── validationMiddleware.js
│   │   ├── errorHandler.js
│   │   └── rateLimiter.js
│   ├── routes/                  # API routes
│   │   ├── index.js
│   │   ├── authRoutes.js
│   │   └── kycRoutes.js
│   ├── validators/              # Request validation schemas
│   │   ├── authValidator.js
│   │   └── kycValidator.js
│   ├── utils/                   # Utility functions
│   │   ├── jwt.js
│   │   ├── bcrypt.js
│   │   ├── logger.js
│   │   └── responseFormatter.js
│   ├── events/                  # Event publishers/subscribers
│   │   ├── publishers/
│   │   │   └── userEventPublisher.js
│   │   └── subscribers/
│   │       └── userEventSubscriber.js
│   ├── migrations/              # Sequelize migrations
```

```
│   │   └── YYYYMMDDHHMMSS-create-users.js
│   ├── seeders/              # Database seeders
│   │   └── demo-users.js
│   └── app.js                # Express app setup
├── tests/                    # Unit & integration tests
│   ├── unit/
│   ├── integration/
│   └── e2e/
├── .env.example
├── .dockerignore
├── Dockerfile
├── package.json
├── .sequelizerc
└── README.md
```

4.4 API Gateway Structure

```
api-gateway/
├── src/
│   ├── config/
│   │   ├── env.js
│   │   ├── redis.js
│   │   └── serviceRegistry.js   # Service discovery
│   ├── middlewares/
│   │   ├── authMiddleware.js    # JWT verification
│   │   ├── rateLimiter.js       # Rate limiting per user
│   │   ├── corsMiddleware.js
│   │   ├── requestLogger.js
│   │   └── errorHandler.js
│   ├── routes/
│   │   ├── index.js
│   │   └── proxyRoutes.js       # Route to microservices
│   ├── services/
│   │   ├── proxyService.js      # HTTP proxy logic
│   │   └── loadBalancer.js      # Load balancing logic
│   ├── utils/
│   │   ├── logger.js
│   │   └── circuitBreaker.js    # Circuit breaker pattern
│   └── app.js
├── .env.example
├── Dockerfile
├── package.json
└── README.md
```

4.5 Shared Libraries Structure

```
backend/shared/
├── utils/
│   ├── logger.js              # Winston logger
│   ├── responseFormatter.js   # Standardized API responses
│   ├── errorClasses.js        # Custom error classes
│   └── dateHelpers.js
├── middleware/
│   ├── authMiddleware.js      # JWT validation (reusable)
│   ├── validationMiddleware.js  # Joi validation
│   └── errorHandler.js        # Global error handler
├── config/
│   ├── database.js            # Database config template
│   ├── redis.js               # Redis config template
│   └── rabbitmq.js            # RabbitMQ config
├── events/
│   ├── eventBus.js            # Event bus wrapper
│   └── eventTypes.js          # Event type constants
└── package.json
```

---

5. API DOCUMENTATION

5.1 API Gateway Routes

Base URL: `http://localhost:3000/api/v1`

Authentication Routes

- `POST /auth/register` → Auth Service
- `POST /auth/login` → Auth Service
- `POST /auth/refresh-token` → Auth Service
- `POST /auth/logout` → Auth Service
- `POST /auth/forgot-password` → Auth Service
- `POST /auth/reset-password` → Auth Service
- `POST /auth/verify-email` → Auth Service

KYC Routes

- `POST /kyc/submit` → Auth Service
- `GET /kyc/status` → Auth Service
- `PUT /kyc/verify/:id` → Auth Service (Admin only)

User Routes

- `GET /users/profile` → User Service
- `PUT /users/profile` → User Service
- `GET /users/:id` → User Service

Group Routes

- `POST /groups` → User Service
- `GET /groups` → User Service
- `GET /groups/:id` → User Service
- `PUT /groups/:id` → User Service
- `DELETE /groups/:id` → User Service
- `POST /groups/:id/members` → User Service
- `DELETE /groups/:id/members/:userId` → User Service
- `PUT /groups/:id/members/:userId/ownership` → User Service

Group Voting Routes

- `POST /groups/:id/votes` → User Service
- `GET /groups/:id/votes` → User Service
- `POST /votes/:id/cast` → User Service
- `PUT /votes/:id/close` → User Service

Booking Routes

- `POST /bookings` → Booking Service
- `GET /bookings` → Booking Service
- `GET /bookings/:id` → Booking Service
- `PUT /bookings/:id` → Booking Service
- `DELETE /bookings/:id` → Booking Service
- `GET /bookings/calendar` → Booking Service
- `POST /bookings/:id/check-in` → Booking Service
- `POST /bookings/:id/check-out` → Booking Service

Cost Routes

- `POST /costs` → Cost Service
- `GET /costs` → Cost Service
- `GET /costs/:id` → Cost Service
- `PUT /costs/:id` → Cost Service
- `DELETE /costs/:id` → Cost Service
- `GET /costs/summary` → Cost Service
- `GET /costs/report` → Cost Service
- `POST /costs/:id/split` → Cost Service

Payment Routes

- `POST /payments` → Cost Service
- `GET /payments` → Cost Service
- `GET /payments/:id` → Cost Service
- `POST /payments/:id/confirm` → Cost Service

Vehicle Routes

- `POST /vehicles` → Vehicle Service
- `GET /vehicles` → Vehicle Service
- `GET /vehicles/:id` → Vehicle Service
- `PUT /vehicles/:id` → Vehicle Service
- `DELETE /vehicles/:id` → Vehicle Service
- `GET /vehicles/:id/maintenance-history` → Vehicle Service
- `POST /vehicles/:id/maintenance` → Vehicle Service
- `POST /vehicles/:id/charging` → Vehicle Service

Contract Routes

- `POST /contracts` → Contract Service
- `GET /contracts` → Contract Service
- `GET /contracts/:id` → Contract Service
- `PUT /contracts/:id` → Contract Service
- `POST /contracts/:id/sign` → Contract Service
- `GET /contracts/:id/download` → Contract Service

Admin Routes

- `GET /admin/dashboard` → Admin Service
- `GET /admin/disputes` → Admin Service
- `GET /admin/disputes/:id` → Admin Service
- `PUT /admin/disputes/:id` → Admin Service
- `POST /admin/disputes/:id/messages` → Admin Service
- `GET /admin/staff` → Admin Service
- `POST /admin/staff` → Admin Service
- `GET /admin/audit-logs` → Admin Service

5.2 API Response Format (Standardized)

Success Response

json

```
{
  "success": true,
  "message": "Operation successful",
  "data": {
    // Response data
  },
  "metadata": {
    "timestamp": "2025-10-12T10:30:00Z",
    "requestId": "uuid"
  }
}
```

Error Response

json

```
{
 "success": false,
 "error": {
  "code": "ERROR_CODE",
  "message": "Error description",
  "details": {}
 },
 "metadata": {
  "timestamp": "2025-10-12T10:30:00Z",
  "requestId": "uuid"
 }
}
```

Paginated Response


json

```
{
 "success": true,
 "data": [...],
 "pagination": {
  "page": 1,
  "limit": 20,
  "total": 100,
  "totalPages": 5
 }
}
```

---

6. DOCKER & DEPLOYMENT

6.1 docker-compose.yml (Development)


yaml

```yaml
version: '3.8'

services:
  # PostgreSQL Databases
  postgres-auth:
    image: postgres:15-alpine
    container_name: ev_postgres_auth
    environment:
      POSTGRES_DB: auth_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5432:5432"
    volumes:
      - postgres_auth_data:/var/lib/postgresql/data
    networks:
      - ev_network

  postgres-user:
    image: postgres:15-alpine
    container_name: ev_postgres_user
    environment:
      POSTGRES_DB: user_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5433:5432"
    volumes:
      - postgres_user_data:/var/lib/postgresql/data
    networks:
      - ev_network

  postgres-booking:
    image: postgres:15-alpine
    container_name: ev_postgres_booking
    environment:
      POSTGRES_DB: booking_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5434:5432"
    volumes:
      - postgres_booking_data:/var/lib/postgresql/data
    networks:
```

```
    - ev_network

  postgres-cost:
    image: postgres:15-alpine
    container_name: ev_postgres_cost
    environment:
      POSTGRES_DB: cost_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5435:5432"
    volumes:
      - postgres_cost_data:/var/lib/postgresql/data
    networks:
      - ev_network

  postgres-vehicle:
    image: postgres:15-alpine
    container_name: ev_postgres_vehicle
    environment:
      POSTGRES_DB: vehicle_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5436:5432"
    volumes:
      - postgres_vehicle_data:/var/lib/postgresql/data
    networks:
      - ev_network

  postgres-contract:
    image: postgres:15-alpine
    container_name: ev_postgres_contract
    environment:
      POSTGRES_DB: contract_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5437:5432"
    volumes:
      - postgres_contract_data:/var/lib/postgresql/data
    networks:
      - ev_network
```

```yaml
postgres-admin:
  image: postgres:15-alpine
  container_name: ev_postgres_admin
  environment:
    POSTGRES_DB: admin_db
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres123
  ports:
    - "5438:5432"
  volumes:
    - postgres_admin_data:/var/lib/postgresql/data
  networks:
    - ev_network

# MongoDB
mongodb:
  image: mongo:7
  container_name: ev_mongodb
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin
    MONGO_INITDB_ROOT_PASSWORD: admin123
  ports:
    - "27017:27017"
  volumes:
    - mongodb_data:/data/db
  networks:
    - ev_network

# Redis
redis:
  image: redis:7-alpine
  container_name: ev_redis
  command: redis-server --requirepass redis123
  ports:
    - "6379:6379"
  volumes:
    - redis_data:/data
  networks:
    - ev_network

# RabbitMQ
rabbitmq:
  image: rabbitmq:3-management-alpine
  container_name: ev_rabbitmq
```

```
  environment:
    RABBITMQ_DEFAULT_USER: admin
    RABBITMQ_DEFAULT_PASS: admin123
  ports:
    - "5672:5672"
    - "15672:15672"
  volumes:
    - rabbitmq_data:/var/lib/rabbitmq
  networks:
    - ev_network

# API Gateway
api-gateway:
  build:
    context: ./backend/api-gateway
    dockerfile: Dockerfile
  container_name: ev_api_gateway
  ports:
    - "3000:3000"
  environment:
    NODE_ENV: development
    PORT: 3000
    REDIS_URL: redis://:redis123@redis:6379
    AUTH_SERVICE_URL: http://auth-service:3001
    USER_SERVICE_URL: http://user-service:3002
    BOOKING_SERVICE_URL: http://booking-service:3003
    COST_SERVICE_URL: http://cost-service:3004
    VEHICLE_SERVICE_URL: http://vehicle-service:3005
    CONTRACT_SERVICE_URL: http://contract-service:3006
    ADMIN_SERVICE_URL: http://admin-service:3007
  depends_on:
    - redis
  networks:
    - ev_network
  volumes:
    - ./backend/api-gateway/src:/app/src
  command: npm run dev

# Auth Service
auth-service:
  build:
    context: ./backend/auth-service
    dockerfile: Dockerfile
  container_name: ev_auth_service
```

```
  ports:
    - "3001:3001"
  environment:
    NODE_ENV: development
    PORT: 3001
    DB_HOST: postgres-auth
    DB_PORT: 5432
    DB_NAME: auth_db
    DB_USER: postgres
    DB_PASSWORD: postgres123
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
    JWT_SECRET: your_jwt_secret_key_here
    JWT_EXPIRES_IN: 1d
    JWT_REFRESH_EXPIRES_IN: 7d
  depends_on:
    - postgres-auth
    - redis
    - rabbitmq
  networks:
    - ev_network
  volumes:
    - ./backend/auth-service/src:/app/src
  command: npm run dev

# User Service
user-service:
  build:
    context: ./backend/user-service
    dockerfile: Dockerfile
  container_name: ev_user_service
  ports:
    - "3002:3002"
  environment:
    NODE_ENV: development
    PORT: 3002
    DB_HOST: postgres-user
    DB_PORT: 5432
    DB_NAME: user_db
    DB_USER: postgres
    DB_PASSWORD: postgres123
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
  depends_on:
```

```
    - postgres-user
    - redis
    - rabbitmq
  networks:
    - ev_network
  volumes:
    - ./backend/user-service/src:/app/src
  command: npm run dev

# Booking Service
booking-service:
  build:
    context: ./backend/booking-service
    dockerfile: Dockerfile
  container_name: ev_booking_service
  ports:
    - "3003:3003"
  environment:
    NODE_ENV: development
    PORT: 3003
    DB_HOST: postgres-booking
    DB_PORT: 5432
    DB_NAME: booking_db
    DB_USER: postgres
    DB_PASSWORD: postgres123
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
  depends_on:
    - postgres-booking
    - redis
    - rabbitmq
  networks:
    - ev_network
  volumes:
    - ./backend/booking-service/src:/app/src
  command: npm run dev

# Cost Service
cost-service:
  build:
    context: ./backend/cost-service
    dockerfile: Dockerfile
  container_name: ev_cost_service
  ports:
```

```
    - "3004:3004"
  environment:
    NODE_ENV: development
    PORT: 3004
    DB_HOST: postgres-cost
    DB_PORT: 5432
    DB_NAME: cost_db
    DB_USER: postgres
    DB_PASSWORD: postgres123
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
  depends_on:
    - postgres-cost
    - redis
    - rabbitmq
  networks:
    - ev_network
  volumes:
    - ./backend/cost-service/src:/app/src
  command: npm run dev

# Vehicle Service
vehicle-service:
  build:
    context: ./backend/vehicle-service
    dockerfile: Dockerfile
  container_name: ev_vehicle_service
  ports:
    - "3005:3005"
  environment:
    NODE_ENV: development
    PORT: 3005
    DB_HOST: postgres-vehicle
    DB_PORT: 5432
    DB_NAME: vehicle_db
    DB_USER: postgres
    DB_PASSWORD: postgres123
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
  depends_on:
    - postgres-vehicle
    - redis
    - rabbitmq
  networks:
```

```
    - ev_network
  volumes:
    - ./backend/vehicle-service/src:/app/src
  command: npm run dev

# Contract Service
contract-service:
  build:
    context: ./backend/contract-service
    dockerfile: Dockerfile
  container_name: ev_contract_service
  ports:
    - "3006:3006"
  environment:
    NODE_ENV: development
    PORT: 3006
    DB_HOST: postgres-contract
    DB_PORT: 5432
    DB_NAME: contract_db
    DB_USER: postgres
    DB_PASSWORD: postgres123
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
  depends_on:
    - postgres-contract
    - redis
    - rabbitmq
  networks:
    - ev_network
  volumes:
    - ./backend/contract-service/src:/app/src
  command: npm run dev

# Admin Service
admin-service:
  build:
    context: ./backend/admin-service
    dockerfile: Dockerfile
  container_name: ev_admin_service
  ports:
    - "3007:3007"
  environment:
    NODE_ENV: development
    PORT: 3007
```

```
    DB_HOST: postgres-admin
    DB_PORT: 5432
    DB_NAME: admin_db
    DB_USER: postgres
    DB_PASSWORD: postgres123
    MONGODB_URL: mongodb://admin:admin123@mongodb:27017
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
  depends_on:
    - postgres-admin
    - mongodb
    - redis
    - rabbitmq
  networks:
    - ev_network
  volumes:
    - ./backend/admin-service/src:/app/src
  command: npm run dev

# Notification Service
notification-service:
  build:
    context: ./backend/notification-service
    dockerfile: Dockerfile
  container_name: ev_notification_service
  ports:
    - "3008:3008"
  environment:
    NODE_ENV: development
    PORT: 3008
    REDIS_URL: redis://:redis123@redis:6379
    RABBITMQ_URL: amqp://admin:admin123@rabbitmq:5672
    SMTP_HOST: smtp.gmail.com
    SMTP_PORT: 587
    SMTP_USER: your_email@gmail.com
    SMTP_PASSWORD: your_app_password
  depends_on:
    - redis
    - rabbitmq
  networks:
    - ev_network
  volumes:
    - ./backend/notification-service/src:/app/src
  command: npm run dev
```

```
  # Frontend
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile.dev
    container_name: ev_frontend
    ports:
      - "5173:5173"
    environment:
      VITE_API_BASE_URL: http://localhost:3000/api/v1
    volumes:
      - ./frontend/src:/app/src
      - ./frontend/public:/app/public
    networks:
      - ev_network
    command: npm run dev -- --host

networks:
  ev_network:
    driver: bridge

volumes:
  postgres_auth_data:
  postgres_user_data:
  postgres_booking_data:
  postgres_cost_data:
  postgres_vehicle_data:
  postgres_contract_data:
  postgres_admin_data:
  mongodb_data:
  redis_data:
  rabbitmq_data:
```

6.2 Dockerfile cho Backend Services (Template)

dockerfile

```
# backend/auth-service/Dockerfile
FROM node:18-alpine

WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci --only=production

# Copy source code
COPY . .

# Expose port
EXPOSE 3001

# Start application
CMD ["node", "src/app.js"]
```

6.3 Dockerfile cho Frontend (Development)


dockerfile

```
# frontend/Dockerfile.dev
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 5173

CMD ["npm", "run", "dev", "--", "--host"]
```

---

7. GITHUB PROJECT PLANNING (8 TUẦN)

7.1 Sprint Planning Overview

Sprint	Duration	Focus Area	Estimated Hours
Sprint 1	Week 1	Setup & Infrastructure	48h
Sprint 2	Week 2	Auth & User Services	56h
Sprint 3	Week 3	Booking Service & Frontend Core	56h
Sprint 4	Week 4	Cost Service & Payment	56h
Sprint 5	Week 5	Vehicle & Contract Services	56h
Sprint 6	Week 6	Admin Service & Notifications	48h
Sprint 7	Week 7	Integration & Testing	56h
Sprint 8	Week 8	Bug Fixes, Polish & Deployment	48h
Total	**8 weeks**		**424 hours**

7.2 Detailed Sprint Breakdown

SPRINT 1 - Week 1: Setup & Infrastructure (48h)

Goals:

- Thiết lập môi trường phát triển
- Cấu trúc dự án
- Database setup
- Docker configuration

Tasks:

Task	Assignee	Estimate	Priority
Setup GitHub repository & project board	Team Lead	2h	High
Create project structure (folders)	All	3h	High
Setup Docker Compose file	Backend Dev 1	4h	High
Setup PostgreSQL databases (7 instances)	Backend Dev 1	3h	High
Setup MongoDB & Redis	Backend Dev 2	2h	High
Setup RabbitMQ	Backend Dev 2	2h	High
Create Sequelize models for Auth DB	Backend Dev 1	4h	High
Create Sequelize models for User DB	Backend Dev 2	4h	High
Run migrations for Auth & User DB	Backend Dev 1-2	2h	High
Setup API Gateway basic structure	Backend Dev 3	4h	High
Setup Shared libraries	Backend Dev 3	3h	Medium
Initialize Frontend with Vite + React	Frontend Dev	4h	High
Setup Tailwind CSS	Frontend Dev	2h	High
Create CI/CD pipeline (GitHub Actions)	Team Lead	3h	Medium
Write Confluence documentation - Part 1	All	4h	High
Code review & testing	All	2h	High

Deliverables:

- ✅ Dự án cấu trúc hoàn chỉnh
- ✅ Docker containers running
- ✅ Databases initialized
- ✅ Basic frontend & backend boilerplate
- ✅ Documentation part 1

SPRINT 2 - Week 2: Auth & User Services (56h)

Goals:

- Hoàn thành Authentication Service
- Hoàn thành User Service
- Basic frontend login/register

Tasks:

Task	Assignee	Estimate	Priority
Auth Service Development			
Implement Register endpoint	Backend Dev 1	4h	High
Implement Login endpoint	Backend Dev 1	4h	High
Implement JWT token generation	Backend Dev 1	3h	High
Implement Refresh token logic	Backend Dev 1	3h	High
Implement Forgot/Reset password	Backend Dev 1	4h	Medium
Implement KYC submission endpoint	Backend Dev 1	4h	Medium
KYC verification (admin) endpoint	Backend Dev 1	3h	Medium
User Service Development			
Implement User Profile CRUD	Backend Dev 2	4h	High
Implement Group CRUD	Backend Dev 2	4h	High
Implement Group Members management	Backend Dev 2	4h	High
Implement Ownership ratio updates	Backend Dev 2	3h	Medium
API Gateway Integration			
Configure Auth routes in Gateway	Backend Dev 3	2h	High
Configure User routes in Gateway	Backend Dev 3	2h	High
Implement JWT middleware	Backend Dev 3	3h	High
Frontend Development			
Create Login page	Frontend Dev	3h	High
Create Register page	Frontend Dev	3h	High
Create KYC Verification page	Frontend Dev	4h	Medium
Implement Auth service (API calls)	Frontend Dev	2h	High
Setup React Router	Frontend Dev	2h	High
Testing & Documentation			
Unit tests for Auth service	Backend Dev 1	3h	Medium
Unit tests for User service	Backend Dev 2	3h	Medium
Update Confluence docs	All	2h	Medium

Deliverables:

- ✅ Auth Service fully functional
- ✅ User Service fully functional
- ✅ Login/Register working on frontend
- ✅ JWT authentication working

SPRINT 3 - Week 3: Booking Service & Frontend Core (56h)

Goals:

- Hoàn thành Booking Service
- Calendar & booking UI
- Dashboard layout

Tasks:

Task	Assignee	Estimate	Priority
Booking Service - Backend			
Create Booking models & migrations	Backend Dev 1	3h	High
Implement Create booking endpoint	Backend Dev 1	4h	High
Implement Get bookings (list & detail)	Backend Dev 1	3h	High
Implement Update/Cancel booking	Backend Dev 1	3h	High
Implement Check-in/Check-out	Backend Dev 1	4h	High
Implement Calendar view endpoint	Backend Dev 1	3h	High
Implement conflict detection logic	Backend Dev 1	4h	Medium
Implement priority algorithm	Backend Dev 1	4h	Medium
Frontend - Booking UI			
Create Booking List page	Frontend Dev	4h	High
Create Booking Calendar component	Frontend Dev	5h	High
Create Create Booking form	Frontend Dev	4h	High
Implement Booking service (API)	Frontend Dev	2h	High
Frontend - Core Layout			
Create Dashboard layout	Frontend Dev	4h	High
Create Header component	Frontend Dev	2h	High
Create Sidebar navigation	Frontend Dev	3h	High
Create Dashboard home page	Frontend Dev	3h	Medium
Integration & Testing			
Integrate Booking routes in Gateway	Backend Dev 3	2h	High
Test booking flow end-to-end	All	3h	High
Update documentation	All	2h	Medium

Deliverables:

- ✅ Booking Service hoàn chỉnh
- ✅ Calendar booking UI
- ✅ Dashboard layout
- ✅ Booking flow working

SPRINT 4 - Week 4: Cost Service & Payment (56h)

Goals:

- Hoàn thành Cost Service
- Payment processing
- Cost splitting algorithms
- Reporting

Tasks:

Task	Assignee	Estimate	Priority
Cost Service - Backend			
Create Cost models & migrations	Backend Dev 2	3h	High
Implement Add cost endpoint	Backend Dev 2	4h	High
Implement Cost splitting logic	Backend Dev 2	5h	High
Implement Get costs (list & detail)	Backend Dev 2	3h	High
Implement Update/Delete cost	Backend Dev 2	3h	Medium
Implement Payment endpoints	Backend Dev 2	4h	High
Implement Invoice generation	Backend Dev 2	4h	Medium
Implement Cost summary/report	Backend Dev 2	4h	High
Frontend - Cost Management			
Create Cost List page	Frontend Dev	4h	High
Create Add Cost form	Frontend Dev	4h	High
Create Cost Report page	Frontend Dev	4h	High
Create Payment modal	Frontend Dev	3h	High
Implement Cost service (API)	Frontend Dev	2h	High
Create cost splitting UI	Frontend Dev	3h	Medium
Integration & Testing			
Integrate Cost routes in Gateway	Backend Dev 3	2h	High
Test payment flow	All	3h	High
Test cost splitting accuracy	Backend Dev 2	2h	High
Update documentation	All	2h	Medium

Deliverables:

- ✅ Cost Service hoàn chỉnh
- ✅ Payment processing working
- ✅ Cost splitting algorithms tested
- ✅ Cost reports & invoices

SPRINT 5 - Week 5: Vehicle & Contract Services (56h)

Goals:

- Hoàn thành Vehicle Service
- Hoàn thành Contract Service
- E-signature integration

Tasks:

Task	Assignee	Estimate	Priority
Vehicle Service - Backend			
Create Vehicle models & migrations	Backend Dev 1	3h	High
Implement Vehicle CRUD endpoints	Backend Dev 1	4h	High
Implement Maintenance scheduling	Backend Dev 1	4h	High
Implement Maintenance history	Backend Dev 1	3h	High
Implement Insurance management	Backend Dev 1	3h	Medium
Implement Charging session tracking	Backend Dev 1	4h	Medium
Contract Service - Backend			
Create Contract models & migrations	Backend Dev 2	3h	High
Implement Contract CRUD endpoints	Backend Dev 2	4h	High
Implement Contract signing logic	Backend Dev 2	5h	High
Implement E-signature integration	Backend Dev 2	4h	High
Implement Contract document storage	Backend Dev 2	3h	Medium
Implement Contract amendments	Backend Dev 2	3h	Medium
Frontend - Vehicle Management			
Create Vehicle List page	Frontend Dev	3h	High
Create Vehicle Detail page	Frontend Dev	4h	High
Create Add Vehicle form	Frontend Dev	3h	High
Create Maintenance scheduler	Frontend Dev	4h	Medium
Implement Vehicle service (API)	Frontend Dev	2h	High
Frontend - Contract Management			
Create Contract List page	Frontend Dev	3h	High
Create Contract Detail page	Frontend Dev	4h	High
Create E-signature component	Frontend Dev	5h	High
Implement Contract service (API)	Frontend Dev	2h	High
Integration & Testing			
Integrate Vehicle & Contract routes	Backend Dev 3	2h	High
Test e-signature flow	All	3h	High
Update documentation	All	2h	Medium

Deliverables:

- ✅ Vehicle Service hoàn chỉnh
- ✅ Contract Service hoàn chỉnh
- ✅ E-signature working
- ✅ Vehicle maintenance tracking

SPRINT 6 - Week 6: Admin Service & Notifications (48h)

Goals:

- Hoàn thành Admin Service
- Notification Service
- Dispute management
- Analytics dashboard

Tasks:

Task	Assignee	Estimate	Priority
Admin Service - Backend			
Create Admin models & migrations	Backend Dev 1	3h	High
Implement Staff management	Backend Dev 1	4h	High
Implement Dispute CRUD endpoints	Backend Dev 1	4h	High
Implement Dispute resolution workflow	Backend Dev 1	4h	High
Implement Audit logs	Backend Dev 1	3h	Medium
Implement System settings	Backend Dev 1	2h	Medium
Admin Service - Analytics			
Setup MongoDB analytics schema	Backend Dev 2	2h	High
Implement Analytics data collection	Backend Dev 2	4h	High
Implement Dashboard statistics API	Backend Dev 2	4h	High
Notification Service			
Setup email service (NodeMailer)	Backend Dev 3	3h	High
Implement email templates	Backend Dev 3	3h	Medium
Implement notification queue consumer	Backend Dev 3	4h	High
Implement push notification (optional)	Backend Dev 3	3h	Low
Frontend - Admin Panel			
Create Admin Dashboard page	Frontend Dev	4h	High
Create Dispute Management page	Frontend Dev	4h	High
Create Staff Management page	Frontend Dev	3h	Medium
Create Analytics charts	Frontend Dev	4h	High
Implement Admin service (API)	Frontend Dev	2h	High
Integration & Testing			
Integrate Admin routes in Gateway	Backend Dev 3	2h	High
Test notification delivery	All	2h	Medium
Update documentation	All	2h	Medium

Deliverables:

- ✅ Admin Service hoàn chỉnh
- ✅ Notification system working
- ✅ Dispute management
- ✅ Analytics dashboard

SPRINT 7 - Week 7: Integration & Testing (56h)

Goals:

- End-to-end testing
- Integration testing
- Performance testing
- Bug fixes

Tasks:

Task	Assignee	Estimate	Priority
Integration Testing			
Test complete user journey (co-owner)	All	6h	High
Test complete admin workflow	All	4h	High
Test all API endpoints	Backend Team	6h	High
Test microservice communication	Backend Team	4h	High
Frontend Testing			
Test all UI components	Frontend Dev	4h	High
Test responsive design	Frontend Dev	3h	High
Test form validations	Frontend Dev	3h	High
Cross-browser testing	Frontend Dev	3h	Medium
Performance Testing			
Load testing API Gateway	Backend Dev 3	3h	Medium
Database query optimization	Backend Team	4h	Medium
Frontend performance audit	Frontend Dev	2h	Medium
Security Testing			
Security audit - JWT implementation	Backend Dev 1	2h	High
Test input validation	All	3h	High
Test SQL injection prevention	Backend Team	2h	High
Test XSS prevention	Frontend Dev	2h	High
Bug Fixes			
Fix critical bugs	All	8h	High
Fix medium priority bugs	All	4h	Medium
Code refactoring	All	4h	Medium
Documentation			
Complete API documentation	Backend Team	3h	High
Complete user guide	Frontend Dev	2h	Medium
Update Confluence docs	All	3h	High

Deliverables:

- ✅ All features tested
- ✅ Critical bugs fixed
- ✅ Performance optimized
- ✅ Security verified

SPRINT 8 - Week 8: Bug Fixes, Polish & Deployment (48h)

Goals:

- Final bug fixes
- UI/UX polish
- Production deployment
- Documentation finalization

Tasks:

Task	Assignee	Estimate	Priority
Final Bug Fixes			
Fix remaining bugs	All	8h	High
Edge case handling	All	4h	High
Error message improvements	All	2h	Medium
UI/UX Polish			
UI consistency check	Frontend Dev	3h	High
Improve loading states	Frontend Dev	2h	Medium
Improve error handling UI	Frontend Dev	2h	Medium
Add animations & transitions	Frontend Dev	3h	Low
Accessibility improvements	Frontend Dev	2h	Medium
Production Preparation			
Create production Docker Compose	Backend Dev 3	3h	High
Setup environment variables	Backend Dev 3	2h	High
Database backup strategy	Backend Dev 1	2h	High
Setup logging & monitoring	Backend Dev 2	3h	High
Deployment			
Deploy databases	Team Lead	2h	High
Deploy backend services	Backend Team	4h	High
Deploy frontend	Frontend Dev	2h	High
Configure domain & SSL	Team Lead	2h	High
Documentation Finalization			
Complete Confluence documentation	All	4h	High
Create deployment guide	Team Lead	2h	High
Create user manual	Frontend Dev	3h	Medium
Create video demo	Team Lead	2h	Medium
Final Review			
Final code review	All	3h	High
Final testing in production	All	2h	High
Presentation preparation	All	2h	High

Deliverables:

- ✅ Production-ready application
- ✅ Complete documentation
- ✅ Deployed and accessible
- ✅ Demo & presentation ready

7.3 GitHub Project Board Structure

Columns:

1. **Backlog** - Tất cả tasks chưa bắt đầu
2. **To Do** - Tasks của sprint hiện tại
3. **In Progress** - Đang làm
4. **In Review** - Đang code review
5. **Testing** - Đang test
6. **Done** - Hoàn thành

Labels:

- `priority: high` - Ưu tiên cao
- `priority: medium` - Ưu tiên trung bình
- `priority: low` - Ưu tiên thấp
- `type: feature` - Feature mới

- `type: bug` - Bug fix
- `type: enhancement` - Cải thiện
- `type: documentation` - Tài liệu
- `service: auth` - Auth service
- `service: user` - User service
- `service: booking` - Booking service
- `service: cost` - Cost service
- `service: vehicle` - Vehicle service
- `service: contract` - Contract service
- `service: admin` - Admin service
- `frontend` - Frontend work
- `backend` - Backend work
- `testing` - Testing task

Milestones:

- Sprint 1: Setup & Infrastructure
- Sprint 2: Auth & User Services
- Sprint 3: Booking Service
- Sprint 4: Cost Service
- Sprint 5: Vehicle & Contract
- Sprint 6: Admin & Notifications
- Sprint 7: Integration & Testing
- Sprint 8: Deployment

---

8. CODE EXAMPLES

8.1 Sequelize Model Example (User)

javascript

```
// backend/auth-service/src/models/User.js
const { DataTypes } = require('sequelize');
const bcrypt = require('bcryptjs');

module.exports = (sequelize) => {
  const User = sequelize.define('User', {
    id: {
      type: DataTypes.UUID,
      defaultValue: DataTypes.UUIDV4,
      primaryKey: true
    },
    email: {
      type: DataTypes.STRING(255),
      allowNull: false,
      unique: true,
      validate: {
        isEmail: true
      }
    },
    phone: {
      type: DataTypes.STRING(20),
      unique: true
    },
    passwordHash: {
      type: DataTypes.STRING(255),
      allowNull: false,
      field: 'password_hash'
    },
    role: {
      type: DataTypes.ENUM('co_owner', 'staff', 'admin'),
      allowNull: false,
      defaultValue: 'co_owner'
    },
    isVerified: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
      field: 'is_verified'
    },
    isActive: {
      type: DataTypes.BOOLEAN,
      defaultValue: true,
      field: 'is_active'
    },
    lastLoginAt: {
```

```
      type: DataTypes.DATE,
      field: 'last_login_at'
    }
  }, {
    tableName: 'users',
    timestamps: true,
    underscored: true,
    hooks: {
      beforeCreate: async (user) => {
        if (user.passwordHash) {
          user.passwordHash = await bcrypt.hash(user.passwordHash, 10);
        }
      },
      beforeUpdate: async (user) => {
        if (user.changed('passwordHash')) {
          user.passwordHash = await bcrypt.hash(user.passwordHash, 10);
        }
      }
    }
  });

  User.prototype.validatePassword = async function(password) {
    return await bcrypt.compare(password, this.passwordHash);
  };

  User.prototype.toJSON = function() {
    const values = { ...this.get() };
    delete values.passwordHash;
    return values;
  };

  return User;
};
```

8.2 Express Controller Example

javascript

```
// backend/auth-service/src/controllers/authController.js
const authService = require('../services/authService');
const { successResponse, errorResponse } = require('../utils/responseFormatter');

class AuthController {
  async register(req, res, next) {
    try {
      const { email, phone, password, role } = req.body;

      const result = await authService.register({
        email,
        phone,
        password,
        role
      });

      return successResponse(res, 'Registration successful', result, 201);
    } catch (error) {
      next(error);
    }
  }

  async login(req, res, next) {
    try {
      const { email, password } = req.body;

      const result = await authService.login(email, password);

      return successResponse(res, 'Login successful', result);
    } catch (error) {
      next(error);
    }
  }

  async refreshToken(req, res, next) {
    try {
      const { refreshToken } = req.body;

      const result = await authService.refreshToken(refreshToken);

      return successResponse(res, 'Token refreshed', result);
    } catch (error) {
      next(error);
    }
```

```
  }

  async logout(req, res, next) {
    try {
      const { refreshToken } = req.body;

      await authService.logout(refreshToken);

      return successResponse(res, 'Logout successful');
    } catch (error) {
      next(error);
    }
  }
}

module.exports = new AuthController();
```

8.3 React Component Example

javascript

```jsx
// frontend/src/pages/auth/Login.jsx
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { authService } from '../../services/authService';
import { useAuthStore } from '../../store/authStore';

const Login = () => {
  const navigate = useNavigate();
  const setAuth = useAuthStore((state) => state.setAuth);

  const [formData, setFormData] = useState({
    email: '',
    password: ''
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value
    });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setLoading(true);

    try {
      const response = await authService.login(formData);

      setAuth(response.data.user, response.data.accessToken);
      localStorage.setItem('refreshToken', response.data.refreshToken);

      navigate('/dashboard');
    } catch (err) {
      setError(err.response?.data?.error?.message || 'Login failed');
    } finally {
      setLoading(false);
    }
  };

  return (
```

```
<div className="min-h-screen flex items-center justify-center bg-gray-50 py-12 px-4 sm:px-6 lg:px-8">
 <div className="max-w-md w-full space-y-8">
  <div>
   <h2 className="mt-6 text-center text-3xl font-extrabold text-gray-900">
    Sign in to your account
   </h2>
  </div>
  <form className="mt-8 space-y-6" onSubmit={handleSubmit}>
   {error && (
    <div className="rounded-md bg-red-50 p-4">
     <p className="text-sm text-red-800">{error}</p>
    </div>
   )}

   <div className="rounded-md shadow-sm -space-y-px">
    <div>
     <input
      name="email"
      type="email"
      required
      className="appearance-none rounded-none relative block w-full px-3 py-2 border border-gray-300 placehol
      placeholder="Email address"
      value={formData.email}
      onChange={handleChange}
     />
    </div>
    <div>
     <input
      name="password"
      type="password"
      required
      className="appearance-none rounded-none relative block w-full px-3 py-2 border border-gray-300 placehol
      placeholder="Password"
      value={formData.password}
      onChange={handleChange}
     />
    </div>
   </div>

   <div>
    <button
     type="submit"
     disabled={loading}
     className="group relative w-full flex justify-center py-2 px-4 border border-transparent text-sm font-mediun
```

```
          >
            {loading ? 'Signing in...' : 'Sign in'}
          </button>
        </div>
      </form>
    </div>
  </div>
 );
};

export default Login;
```

---

9. DATABASE INDEXES & OPTIMIZATION

9.1 Recommended Indexes

sql

```sql
-- AUTH DATABASE
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_phone ON users(phone);
CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_kyc_user_id ON kyc_verifications(user_id);
CREATE INDEX idx_kyc_status ON kyc_verifications(verification_status);
CREATE INDEX idx_refresh_tokens_user ON refresh_tokens(user_id);
CREATE INDEX idx_refresh_tokens_token ON refresh_tokens(token);

-- USER DATABASE
CREATE INDEX idx_profiles_user_id ON user_profiles(user_id);
CREATE INDEX idx_groups_created_by ON co_ownership_groups(created_by);
CREATE INDEX idx_members_group_id ON group_members(group_id);
CREATE INDEX idx_members_user_id ON group_members(user_id);
CREATE INDEX idx_votes_group_id ON group_votes(group_id);
CREATE INDEX idx_votes_status ON group_votes(status);

-- BOOKING DATABASE
CREATE INDEX idx_bookings_vehicle_id ON bookings(vehicle_id);
CREATE INDEX idx_bookings_user_id ON bookings(user_id);
CREATE INDEX idx_bookings_group_id ON bookings(group_id);
CREATE INDEX idx_bookings_start_time ON bookings(start_time);
CREATE INDEX idx_bookings_status ON bookings(status);
CREATE INDEX idx_checkin_booking_id ON check_in_out_logs(booking_id);

-- COST DATABASE
CREATE INDEX idx_costs_group_id ON costs(group_id);
CREATE INDEX idx_costs_vehicle_id ON costs(vehicle_id);
CREATE INDEX idx_costs_date ON costs(cost_date);
CREATE INDEX idx_splits_cost_id ON cost_splits(cost_id);
CREATE INDEX idx_splits_user_id ON cost_splits(user_id);
CREATE INDEX idx_payments_user_id ON payments(user_id);
CREATE INDEX idx_invoices_group_id ON invoices(group_id);

-- VEHICLE DATABASE
CREATE INDEX idx_vehicles_group_id ON vehicles(group_id);
CREATE INDEX idx_vehicles_status ON vehicles(status);
CREATE INDEX idx_maintenance_vehicle_id ON maintenance_schedules(vehicle_id);
CREATE INDEX idx_charging_vehicle_id ON charging_sessions(vehicle_id);

-- CONTRACT DATABASE
CREATE INDEX idx_contracts_group_id ON contracts(group_id);
CREATE INDEX idx_contracts_status ON contracts(status);
CREATE INDEX idx_parties_contract_id ON contract_parties(contract_id);
```

```
CREATE INDEX idx_parties_user_id ON contract_parties(user_id);

-- ADMIN DATABASE
CREATE INDEX idx_disputes_group_id ON disputes(group_id);
CREATE INDEX idx_disputes_status ON disputes(status);
CREATE INDEX idx_disputes_assigned_to ON disputes(assigned_to);
CREATE INDEX idx_audit_user_id ON audit_logs(user_id);
CREATE INDEX idx_audit_created_at ON audit_logs(created_at);
```

---

10. ENVIRONMENT VARIABLES

10.1 .env.example (Auth Service)

bash

```
# Server
NODE_ENV=development
PORT=3001

# Database
DB_HOST=localhost
DB_PORT=5432
DB_NAME=auth_db
DB_USER=postgres
DB_PASSWORD=postgres123
DB_POOL_MAX=5
DB_POOL_MIN=0
DB_POOL_ACQUIRE=30000
DB_POOL_IDLE=10000

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=redis123
REDIS_DB=0

# RabbitMQ
RABBITMQ_URL=amqp://admin:admin123@localhost:5672
RABBITMQ_EXCHANGE=ev_exchange
RABBITMQ_QUEUE_AUTH=auth_queue

# JWT
JWT_SECRET=your_super_secret_jwt_key_change_in_production
JWT_EXPIRES_IN=1d
JWT_REFRESH_SECRET=your_refresh_token_secret_key
JWT_REFRESH_EXPIRES_IN=7d

# Email (for verification)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=your_email@gmail.com
SMTP_PASSWORD=your_app_specific_password
EMAIL_FROM="EV Co-ownership <noreply@evcoownership.com>"

# File Upload
MAX_FILE_SIZE=5242880
ALLOWED_FILE_TYPES=image/jpeg,image/png,image/jpg,application/pdf
```

```
# Other Services URLs
USER_SERVICE_URL=http://localhost:3002
NOTIFICATION_SERVICE_URL=http://localhost:3008
```

---

11. TESTING STRATEGY

11.1 Testing Pyramid

```
          /\
         /E2E\
        /______\
       /        \
      /Integration\
     /____________\
    /              \
   /   Unit Tests   \
  /__________________\
```

11.2 Unit Test Example (Jest)

javascript

```
// backend/auth-service/tests/unit/authService.test.js
const authService = require('../../src/services/authService');
const { User } = require('../../src/models');

jest.mock('../../src/models');

describe('AuthService', () => {
  describe('register', () => {
    it('should create a new user successfully', async () => {
      const userData = {
        email: 'test@example.com',
        password: 'password123',
        role: 'co_owner'
      };

      User.create.mockResolvedValue({
        id: 'uuid-123',
        email: userData.email,
        role: userData.role
      });

      const result = await authService.register(userData);

      expect(result).toHaveProperty('id');
      expect(result.email).toBe(userData.email);
      expect(User.create).toHaveBeenCalledWith(
        expect.objectContaining({
          email: userData.email,
          role: userData.role
        })
      );
    });

    it('should throw error if email already exists', async () => {
      User.create.mockRejectedValue({
        name: 'SequelizeUniqueConstraintError'
      });

      await expect(
        authService.register({
          email: 'existing@example.com',
          password: 'pass123'
        })
      ).rejects.toThrow();
```

```
  });
});
});
```

---

12. SECURITY BEST PRACTICES

12.1 Security Checklist

✅ Authentication & Authorization

- JWT với expiration time ngắn
- Refresh token rotation
- Password hashing (bcrypt với salt rounds >= 10)
- Rate limiting trên login endpoint
- Account lockout sau nhiều lần đăng nhập sai

✅ Input Validation

- Validate tất cả input (Joi/Yup)
- Sanitize input để tránh XSS
- Parameterized queries (Sequelize ORM)
- File upload validation (type, size)

✅ API Security

- CORS configuration
- Helmet.js cho security headers
- HTTPS only trong production
- API rate limiting
- Request size limiting

✅ Data Protection

- Mã hóa sensitive data
- Secure session management
- Environment variables cho secrets
- Database backup strategy

✅ Monitoring & Logging

- Log tất cả authentication attempts
- Audit logs cho sensitive operations
- Error logging (không expose stack trace)
- Monitor suspicious activities

---

13. DEPLOYMENT CHECKLIST

13.1 Pre-deployment

- ☐ All tests passing
- ☐ Code reviewed
- ☐ Environment variables configured

- [ ] Database migrations ready
- [ ] Backup strategy in place
- [ ] Monitoring setup
- [ ] SSL certificates ready
- [ ] Domain configured

13.2 Production Configuration

yaml

```yaml
# docker-compose.prod.yml highlights
services:
  api-gateway:
    restart: always
    environment:
      NODE_ENV: production
    deploy:
      replicas: 2
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
```

SUMMARY

Đây là tài liệu đầy đủ cho dự án EV Co-ownership System bao gồm:

✅ **Kiến trúc Microservices hoàn chỉnh** ✅ **Database design chi tiết cho 7 PostgreSQL + 1 MongoDB** ✅ **Cấu trúc dự án Frontend (React) & Backend (Node.js)** ✅ **API Gateway configuration** ✅ **Docker Compose setup** ✅ **GitHub Project Planning 8 tuần (424h)** ✅ **Code examples** ✅ **Testing strategy** ✅ **Security best practices** ✅ **Deployment guide**

Next Steps:

1. Clone repository structure
2. Setup Docker containers
3. Follow Sprint 1 tasks
4. Start coding theo planning

Good luck với dự án! 🚀