PropertyConnect - Real Estate Platform

Project Overview

PropertyConnect is a web-based real estate platform that connects agents with buyers/renters, featuring blockchain verification, AI-driven insights, and comprehensive property management tools.

Project Directory Structure

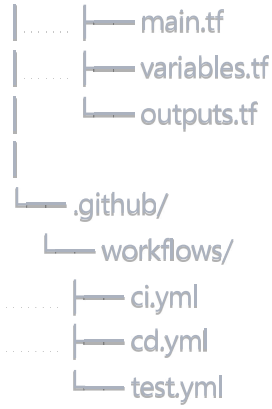
```
propertyconnect/
├── README.md
├── docker-compose.yml
├── .gitignore
├── .env.example
├── package.json
├──
├── frontend/
│   ├── package.json
│   ├── tsconfig.json
│   ├── tailwind.config.js
│   ├── vite.config.ts
│   ├── .env.example
│   ├── public/
│   │   ├── index.html
│   │   ├── favicon.ico
│   │   └── images/
│   │       ├── logo.png
│   │       └── property-placeholder.jpg
│   ├── src/
│   │   ├── main.tsx
│   │   ├── App.tsx
│   │   ├── index.css
│   │   ├── types/
│   │   │   ├── index.ts
│   │   │   ├── property.ts
│   │   │   ├── user.ts
│   │   │   └── agent.ts
│   │   ├── components/
│   │   │   ├── common/
│   │   │   │   ├── Header.tsx
│   │   │   │   ├── Footer.tsx
│   │   │   │   ├── LoadingSpinner.tsx
│   │   │   │   ├── ErrorBoundary.tsx
│   │   │   │   └── Modal.tsx
│   │   │   ├── property/
│   │   │   │   ├── PropertyCard.tsx
│   │   │   │   ├── PropertyList.tsx
│   │   │   │   ├── PropertyDetails.tsx
│   │   │   │   ├── PropertyFilters.tsx
│   │   │   │   ├── PropertySearch.tsx
│   │   │   │   └── VirtualTour.tsx
│   │   │   ├── agent/
```

```
│   │   │   │   ├── AgentCard.tsx
│   │   │   │   ├── AgentProfile.tsx
│   │   │   │   ├── AgentDashboard.tsx
│   │   │   │   ├── AgentMetrics.tsx
│   │   │   │   └── AgentCollaboration.tsx
│   │   │   ├── map/
│   │   │   │   ├── MapView.tsx
│   │   │   │   ├── MapMarker.tsx
│   │   │   │   └── MapFilters.tsx
│   │   │   ├── auth/
│   │   │   │   ├── LoginForm.tsx
│   │   │   │   ├── RegisterForm.tsx
│   │   │   │   └── ProtectedRoute.tsx
│   │   │   ├── chat/
│   │   │   │   ├── ChatBot.tsx
│   │   │   │   ├── ChatMessage.tsx
│   │   │   │   └── SupportChat.tsx
│   │   │   └── blockchain/
│   │   │       ├── VerificationBadge.tsx
│   │   │       └── BlockchainStatus.tsx
│   │   ├── pages/
│   │   │   ├── HomePage.tsx
│   │   │   ├── PropertyListingsPage.tsx
│   │   │   ├── PropertyDetailsPage.tsx
│   │   │   ├── AgentDashboardPage.tsx
│   │   │   ├── BuyerDashboardPage.tsx
│   │   │   ├── LoginPage.tsx
│   │   │   ├── RegisterPage.tsx
│   │   │   └── NotFoundPage.tsx
│   │   ├── store/
│   │   │   ├── index.ts
│   │   │   ├── slices/
│   │   │   │   ├── authSlice.ts
│   │   │   │   ├── propertySlice.ts
│   │   │   │   ├── agentSlice.ts
│   │   │   │   ├── chatSlice.ts
│   │   │   │   └── mapSlice.ts
│   │   │   └── api/
│   │   │       ├── authApi.ts
│   │   │       ├── propertyApi.ts
│   │   │       ├── agentApi.ts
│   │   │       └── chatApi.ts
│   │   ├── services/
│   │   │   ├── api.ts
```

```
│   │   │   ├── auth.ts
│   │   │   ├── mapbox.ts
│   │   │   ├── blockchain.ts
│   │   │   └── chatbot.ts
│   │   ├── utils/
│   │   │   ├── constants.ts
│   │   │   ├── helpers.ts
│   │   │   ├── validation.ts
│   │   │   └── formatters.ts
│   │   ├── hooks/
│   │   │   ├── useAuth.ts
│   │   │   ├── useProperties.ts
│   │   │   ├── useAgents.ts
│   │   │   └── useChat.ts
│   │   └── __tests__/
│   │       ├── components/
│   │       ├── pages/
│   │       └── utils/
│   └── Dockerfile
│
├── backend/
│   ├── package.json
│   ├── tsconfig.json
│   ├── .env.example
│   ├── prisma/
│   │   ├── schema.prisma
│   │   ├── migrations/
│   │   └── seed.ts
│   ├── src/
│   │   ├── server.ts
│   │   ├── app.ts
│   │   ├── config/
│   │   │   ├── database.ts
│   │   │   ├── blockchain.ts
│   │   │   ├── auth.ts
│   │   │   └── redis.ts
│   │   ├── controllers/
│   │   │   ├── authController.ts
│   │   │   ├── propertyController.ts
│   │   │   ├── agentController.ts
│   │   │   ├── userController.ts
│   │   │   ├── chatController.ts
│   │   │   └── blockchainController.ts
│   │   ├── middleware/
```

```
│   │   │   ├── auth.ts
│   │   │   ├── validation.ts
│   │   │   ├── rateLimiter.ts
│   │   │   ├── errorHandler.ts
│   │   │   └── logger.ts
│   │   ├── models/
│   │   │   ├── User.ts
│   │   │   ├── Property.ts
│   │   │   ├── Agent.ts
│   │   │   ├── Transaction.ts
│   │   │   └── Chat.ts
│   │   ├── routes/
│   │   │   ├── index.ts
│   │   │   ├── auth.ts
│   │   │   ├── properties.ts
│   │   │   ├── agents.ts
│   │   │   ├── users.ts
│   │   │   ├── chat.ts
│   │   │   └── blockchain.ts
│   │   ├── services/
│   │   │   ├── authService.ts
│   │   │   ├── propertyService.ts
│   │   │   ├── agentService.ts
│   │   │   ├── blockchainService.ts
│   │   │   ├── aiService.ts
│   │   │   ├── emailService.ts
│   │   │   ├── chatbotService.ts
│   │   │   └── mapService.ts
│   │   ├── utils/
│   │   │   ├── constants.ts
│   │   │   ├── helpers.ts
│   │   │   ├── validation.ts
│   │   │   └── encryption.ts
│   │   ├── types/
│   │   │   ├── index.ts
│   │   │   ├── property.ts
│   │   │   ├── user.ts
│   │   │   └── blockchain.ts
│   │   └── __tests__/
│   │       ├── controllers/
│   │       ├── services/
│   │       └── utils/
│   └── Dockerfile
│
```

```
├── blockchain/
│   ├── package.json
│   ├── truffle-config.js
│   ├── contracts/
│   │   ├── PropertyVerification.sol
│   │   └── Migrations.sol
│   ├── migrations/
│   │   ├── 1_initial_migration.js
│   │   └── 2_deploy_contracts.js
│   ├── test/
│   │   └── PropertyVerification.test.js
│   └── scripts/
│       ├── deploy.js
│       └── verify.js
│
├── ai/
│   ├── package.json
│   ├── requirements.txt
│   ├── chatbot/
│   │   ├── main.py
│   │   ├── intents.json
│   │   └── responses.py
│   ├── rate-analysis/
│   │   ├── model.py
│   │   ├── train.py
│   │   ├── predict.py
│   │   └── data/
│   │       └── sample_data.csv
│   └── api/
│       ├── app.py
│       └── routes.py
│
├── infrastructure/
│   ├── docker/
│   │   ├── nginx/
│   │   │   ├── nginx.conf
│   │   │   └── Dockerfile
│   │   └── redis/
│   │       └── redis.conf
│   ├── k8s/
│   │   ├── deployment.yaml
│   │   ├── service.yaml
│   │   └── ingress.yaml
│   └── terraform/
```



Key Configuration Files

Root Package.json

json

```
{
  "name": "propertyconnect",
  "version": "1.0.0",
  "description": "Real Estate Platform with Blockchain Verification",
  "private": true,
  "workspaces": [
    "frontend",
    "backend",
    "blockchain",
    "ai"
  ],
  "scripts": {
    "dev": "concurrently \"npm run dev:frontend\" \"npm run dev:backend\"",
    "dev:frontend": "cd frontend && npm run dev",
    "dev:backend": "cd backend && npm run dev",
    "build": "npm run build:frontend && npm run build:backend",
    "build:frontend": "cd frontend && npm run build",
    "build:backend": "cd backend && npm run build",
    "test": "npm run test:frontend && npm run test:backend",
    "test:frontend": "cd frontend && npm test",
    "test:backend": "cd backend && npm test",
    "docker:build": "docker-compose build",
    "docker:up": "docker-compose up -d",
    "docker:down": "docker-compose down"
  },
  "devDependencies": {
    "concurrently": "^7.6.0"
  }
}
```

Docker Compose

yaml

```yaml
version: '3.8'

services:
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=development
      - VITE_API_URL=http://localhost:5000
    depends_on:
      - backend
    volumes:
      - ./frontend:/app
      - /app/node_modules

  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - NODE_ENV=development
      - DATABASE_URL=postgresql://user:password@postgres:5432/propertyconnect
      - REDIS_URL=redis://redis:6379
    depends_on:
      - postgres
      - redis
    volumes:
      - ./backend:/app
      - /app/node_modules

  postgres:
    image: postgres:15
    environment:
      - POSTGRES_DB=propertyconnect
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
```

```
    image: redis:7-alpine
    ports:
      - "6379:6379"

  ai-service:
    build: ./ai
    ports:
      - "8000:8000"
    environment:
      - FLASK_ENV=development
    depends_on:
      - redis

volumes:
  postgres_data:
```

Environment Variables (.env.example)

env

```
# Database
DATABASE_URL=postgresql://user:password@localhost:5432/propertyconnect
REDIS_URL=redis://localhost:6379

# Authentication
JWT_SECRET=your-jwt-secret-here
JWT_EXPIRES_IN=7d
AUTH0_DOMAIN=your-auth0-domain
AUTH0_CLIENT_ID=your-auth0-client-id
AUTH0_CLIENT_SECRET=your-auth0-client-secret

# Blockchain
ETHEREUM_RPC_URL=https://mainnet.infura.io/v3/your-project-id
PRIVATE_KEY=your-private-key-here
CONTRACT_ADDRESS=your-contract-address

# APIs
GOOGLE_MAPS_API_KEY=your-google-maps-api-key
MAPBOX_ACCESS_TOKEN=your-mapbox-token
INTERCOM_ACCESS_TOKEN=your-intercom-token

# AI Services
OPENAI_API_KEY=your-openai-api-key
DIALOGFLOW_PROJECT_ID=your-dialogflow-project-id
DIALOGFLOW_CREDENTIALS=path/to/credentials.json

# Email
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASS=your-app-password

# Cloud Storage
AWS_ACCESS_KEY_ID=your-aws-access-key
AWS_SECRET_ACCESS_KEY=your-aws-secret-key
AWS_REGION=us-east-1
AWS_S3_BUCKET=propertyconnect-uploads

# Development
NODE_ENV=development
PORT=5000
FRONTEND_URL=http://localhost:3000
```

Core Implementation Files

Frontend - Main App Component

typescript

```tsx
// frontend/src/App.tsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { Provider } from 'react-redux';
import { store } from './store';
import { AuthProvider } from './contexts/AuthContext';
import { ErrorBoundary } from './components/common/ErrorBoundary';
import { Header } from './components/common/Header';
import { Footer } from './components/common/Footer';
import { ChatBot } from './components/chat/ChatBot';
import { HomePage } from './pages/HomePage';
import { PropertyListingsPage } from './pages/PropertyListingsPage';
import { PropertyDetailsPage } from './pages/PropertyDetailsPage';
import { AgentDashboardPage } from './pages/AgentDashboardPage';
import { BuyerDashboardPage } from './pages/BuyerDashboardPage';
import { LoginPage } from './pages/LoginPage';
import { RegisterPage } from './pages/RegisterPage';
import { NotFoundPage } from './pages/NotFoundPage';
import { ProtectedRoute } from './components/auth/ProtectedRoute';
import 'mapbox-gl/dist/mapbox-gl.css';

function App() {
  return (
    <ErrorBoundary>
      <Provider store={store}>
        <AuthProvider>
          <Router>
            <div className="min-h-screen bg-gray-50">
              <Header />
              <main className="flex-1">
                <Routes>
                  <Route path="/" element={<HomePage />} />
                  <Route path="/properties" element={<PropertyListingsPage />} />
                  <Route path="/properties/:id" element={<PropertyDetailsPage />} />
                  <Route path="/login" element={<LoginPage />} />
                  <Route path="/register" element={<RegisterPage />} />
                  <Route
                    path="/agent/dashboard"
                    element={
                      <ProtectedRoute role="agent">
                        <AgentDashboardPage />
                      </ProtectedRoute>
                    }
```

```
                />
                <Route
                  path="/buyer/dashboard"
                  element={
                    <ProtectedRoute role="buyer">
                      <BuyerDashboardPage />
                    </ProtectedRoute>
                  }
                />
                <Route path="*" element={<NotFoundPage />} />
              </Routes>
            </main>
            <Footer />
            <ChatBot />
          </div>
        </Router>
      </AuthProvider>
    </Provider>
  </ErrorBoundary>
 );
}

export default App;
```

Backend - Main Server

typescript

```typescript
// backend/src/server.ts
import express from 'express';
import cors from 'cors';
import helmet from 'helmet';
import morgan from 'morgan';
import rateLimit from 'express-rate-limit';
import { PrismaClient } from '@prisma/client';
import { errorHandler } from './middleware/errorHandler';
import { logger } from './middleware/logger';
import routes from './routes';

const app = express();
const prisma = new PrismaClient();

// Security middleware
app.use(helmet());
app.use(cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:3000',
  credentials: true
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  message: 'Too many requests from this IP, please try again later.'
});
app.use('/api/', limiter);

// Body parsing
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));

// Logging
app.use(morgan('combined', { stream: { write: (message) => logger.info(message.trim()) } }));

// Health check
app.get('/health', (req, res) => {
  res.json({ status: 'OK', timestamp: new Date().toISOString() });
});

// API routes
app.use('/api', routes);
```

```
// Error handling
app.use(errorHandler);

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
  console.log(`Environment: ${process.env.NODE_ENV}`);
});

// Graceful shutdown
process.on('SIGTERM', async () => {
  console.log('SIGTERM received, shutting down gracefully');
  await prisma.$disconnect();
  process.exit(0);
});
```

Database Schema (Prisma)

prisma

```
// backend/prisma/schema.prisma
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id          String   @id @default(cuid())
  email       String   @unique
  password    String
  firstName   String
  lastName    String
  phone       String?
  avatar      String?
  role        Role     @default(BUYER)
  isVerified  Boolean  @default(false)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  // Relations
  agent       Agent?
  buyer       Buyer?
  messages    Message[]
  transactions Transaction[]

  @@map("users")
}

model Agent {
  id            String   @id @default(cuid())
  userId        String   @unique
  licenseNumber String   @unique
  agency        String
  experience    Int
  rating        Float    @default(0)
  reviewCount   Int      @default(0)
  responseTime  Int      @default(24) // hours
  bio           String?
  specialties   String[]
```

```
  serviceAreas String[]
  isActive     Boolean  @default(true)
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt

  // Relations
  user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  properties   Property[]
  collaborations  Collaboration[]
  metrics      AgentMetric[]

  @@map("agents")
}

model Buyer {
  id           String   @id @default(cuid())
  userId       String   @unique
  preferredAreas String[]
  budgetMin    Int?
  budgetMax    Int?
  propertyTypes  PropertyType[]
  savedProperties String[]
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt

  // Relations
  user         User   @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@map("buyers")
}

model Property {
  id           String       @id @default(cuid())
  title        String
  description  String
  type         PropertyType
  status       PropertyStatus @default(ACTIVE)
  price        Int
  currency     String       @default("USD")
  bedrooms     Int?
  bathrooms    Int?
  area         Int
  areaUnit     String       @default("sqft")
  address      String
```

```
  city          String
  state         String
  zipCode       String
  country       String     @default("US")
  coordinates   Json // {lat: number, lng: number}
  images        String[]
  virtualTourUrl String?
  amenities     String[]
  yearBuilt     Int?
  lotSize       Int?
  isVerified    Boolean    @default(false)
  blockchainTxHash String?
  localInsights Json? // School ratings, crime rates, etc.
  aiAnalysis    Json? // AI-generated insights
  createdAt     DateTime   @default(now())
  updatedAt     DateTime   @updatedAt

  // Relations
  agentId       String
  agent         Agent      @relation(fields: [agentId], references: [id])
  transactions  Transaction[]
  collaborations Collaboration[]

  @@map("properties")
}

model Transaction {
  id            String          @id @default(cuid())
  propertyId    String
  buyerId       String
  agentId       String
  type          TransactionType
  amount        Int
  status        TransactionStatus @default(PENDING)
  contractHash  String?
  completedAt   DateTime?
  createdAt     DateTime        @default(now())
  updatedAt     DateTime        @updatedAt

  // Relations
  property      Property @relation(fields: [propertyId], references: [id])
  buyer         User     @relation(fields: [buyerId], references: [id])

  @@map("transactions")
```

```
}

model Collaboration {
  id              String   @id @default(cuid())
  propertyId      String
  primaryAgentId  String
  secondaryAgentId String
  type            CollaborationType
  splitRatio      Float    @default(0.5)
  status          CollaborationStatus @default(PENDING)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  // Relations
  property        Property @relation(fields: [propertyId], references: [id])
  primaryAgent    Agent    @relation(fields: [primaryAgentId], references: [id])

  @@map("collaborations")
}

model Message {
  id              String      @id @default(cuid())
  senderId        String
  receiverId      String?
  content         String
  type            MessageType @default(TEXT)
  isRead          Boolean     @default(false)
  createdAt       DateTime    @default(now())

  // Relations
  sender          User @relation(fields: [senderId], references: [id])

  @@map("messages")
}

model AgentMetric {
  id              String   @id @default(cuid())
  agentId         String
  month           Int
  year            Int
  propertiesListed Int     @default(0)
  propertiesSold  Int      @default(0)
  totalRevenue    Int      @default(0)
  avgResponseTime Int      @default(0)
```

```
  customerSatisfaction Float @default(0)
  createdAt            DateTime @default(now())

  // Relations
  agent                Agent @relation(fields: [agentId], references: [id])

  @@unique([agentId, month, year])
  @@map("agent_metrics")
}

enum Role {
  BUYER
  AGENT
  ADMIN
}

enum PropertyType {
  HOUSE
  APARTMENT
  CONDO
  TOWNHOUSE
  LAND
  COMMERCIAL
}

enum PropertyStatus {
  ACTIVE
  PENDING
  SOLD
  WITHDRAWN
}

enum TransactionType {
  SALE
  RENT
  LEASE
}

enum TransactionStatus {
  PENDING
  APPROVED
  COMPLETED
  CANCELLED
}
```

```
enum CollaborationType {
  CO_LISTING
  REFERRAL
  JOINT_VENTURE
}

enum CollaborationStatus {
  PENDING
  ACCEPTED
  DECLINED
  COMPLETED
}

enum MessageType {
  TEXT
  IMAGE
  DOCUMENT
  SYSTEM
}
```

Frontend - Property Search Component

typescript

```tsx
// frontend/src/components/property/PropertySearch.tsx
import React, { useState, useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { Search, MapPin, Filter, SlidersHorizontal } from 'lucide-react';
import { setFilters, searchProperties } from '../../store/slices/propertySlice';
import { RootState } from '../../store';
import { PropertyFilters } from './PropertyFilters';
import { MapView } from '../map/MapView';

export const PropertySearch: React.FC = () => {
  const dispatch = useDispatch();
  const { filters, isLoading, properties } = useSelector((state: RootState) => state.property);
  const [searchTerm, setSearchTerm] = useState('');
  const [showFilters, setShowFilters] = useState(false);
  const [viewMode, setViewMode] = useState<'list' | 'map'>('list');

  useEffect(() => {
    dispatch(searchProperties({ ...filters, searchTerm }));
  }, [filters, searchTerm, dispatch]);

  const handleSearch = (e: React.FormEvent) => {
    e.preventDefault();
    dispatch(searchProperties({ ...filters, searchTerm }));
  };

  const handleFilterChange = (newFilters: any) => {
    dispatch(setFilters(newFilters));
  };

  return (
    <div className="w-full">
      {/* Search Bar */}
      <div className="bg-white rounded-lg shadow-md p-4 mb-6">
        <form onSubmit={handleSearch} className="flex gap-4">
          <div className="flex-1 relative">
            <Search className="absolute left-3 top-3 h-5 w-5 text-gray-400" />
            <input
              type="text"
              placeholder="Search by location, property type, or keywords..."
              className="w-full pl-10 pr-4 py-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:
              value={searchTerm}
              onChange={(e) => setSearchTerm(e.target.value)}
            />
```

```
        </div>
        <button
          type="button"
          onClick={() => setShowFilters(!showFilters)}
          className="flex items-center gap-2 px-4 py-2 bg-gray-100 text-gray-700 rounded-lg hover:bg-gray-200"
        >
          <SlidersHorizontal className="h-5 w-5" />
          Filters
        </button>
        <button
          type="submit"
          className="px-6 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 disabled:opacity-50"
          disabled={isLoading}
        >
          {isLoading ? 'Searching...' : 'Search'}
        </button>
      </form>

      {/* Filters Panel */}
      {showFilters && (
        <div className="mt-4 border-t pt-4">
          <PropertyFilters
            filters={filters}
            onFiltersChange={handleFilterChange}
          />
        </div>
      )}
    </div>

    {/* View Mode Toggle */}
    <div className="flex justify-between items-center mb-4">
      <div className="flex items-center gap-2">
        <span className="text-gray-600">
          {properties.length} properties found
        </span>
      </div>
      <div className="flex rounded-lg border border-gray-300 overflow-hidden">
        <button
          className={`px-4 py-2 ${viewMode === 'list' ? 'bg-blue-600 text-white' : 'bg-white text-gray-700'}`}
          onClick={() => setViewMode('list')}
        >
          List
        </button>
        <button
```

```
            className={`px-4 py-2 ${viewMode === 'map' ? 'bg-blue-600 text-white' : 'bg-white text-gray-700'}`}
            onClick={() => setViewMode('map')}
          >
            Map
          </button>
        </div>
      </div>

      {/* Results */}
      {viewMode === 'map' ? (
        <MapView properties={properties} />
      ) : (
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
          {properties.map((property) => (
            <PropertyCard key={property.id} property={property} />
          ))}
        </div>
      )}
    </div>
  );
};
```

Backend - Property Controller

typescript

```typescript
// backend/src/controllers/propertyController.ts
import { Request, Response } from 'express';
import { PrismaClient } from '@prisma/client';
import { blockchainService } from '../services/blockchainService';
import { aiService } from '../services/aiService';
import { mapService } from '../services/mapService';
import { validateProperty } from '../utils/validation';

const prisma = new PrismaClient();

export const createProperty = async (req: Request, res: Response) => {
  try {
    const { error } = validateProperty(req.body);
    if (error) {
      return res.status(400).json({ error: error.details[0].message });
    }

    const { agentId } = req.user;
```