Technical Specifications

laura assistant

1. INTRODUCTION

1.1 EXECUTIVE SUMMARY

1.1.1 Project Overview

PropertyPro AI is an intelligent real estate assistant platform that combines artificial intelligence with mobile-first design to transform how real estate professionals manage their business operations. The system serves as a comprehensive digital assistant that automates routine tasks, generates professional content, and provides data-driven insights to enhance productivity and client relationships.

1.1.2 Core Business Problem

Real estate professionals face significant challenges in managing multiple aspects of their business simultaneously, including client relationship management, property marketing, content creation, task coordination, and performance tracking. Current solutions are fragmented across multiple platforms, leading to inefficiencies, missed opportunities, and inconsistent client experiences. PropertyPro AI addresses these challenges by offering features such as type annotations, interfaces, generics, and more, which help developers catch errors early in the development process and improve code maintainability and scalability.

1.1.3 Key Stakeholders and Users

Stakeholder Group	Primary Role	Key Interests
Real Estate Agents	Primary end users	Productivity enhancement, client management, content generation

Stakeholder Group	Primary Role	Key Interests
Real Estate Te ams	Collaborative u sers	Standardized processes, shared kn owledge base, performance trackin g
Real Estate Br okerages	Administrative oversight	Quality control, compliance, busine ss intelligence
Development Team	System builder s	Technical excellence, maintainabilit y, scalability

1.1.4 Expected Business Impact and Value Proposition

PropertyPro AI delivers measurable business value through:

- **Productivity Enhancement**: 80% reduction in content creation time and 10+ hours saved weekly on administrative tasks
- **Revenue Growth**: Improved lead conversion rates through automated follow-ups and personalized client communications
- **Quality Assurance**: Consistent professional presentation across all client touchpoints
- **Competitive Advantage**: AI-powered insights and automation capabilities not available in traditional real estate tools

1.2 SYSTEM OVERVIEW

1.2.1 Project Context

Business Context and Market Positioning

PropertyPro AI positions itself as the first comprehensive AI-powered real estate assistant that combines mobile accessibility with enterprise-grade functionality. The platform addresses the growing demand for intelligent automation in real estate operations while maintaining the personal touch that defines successful real estate relationships.

Current System Limitations

Traditional real estate management systems suffer from:

- Fragmented functionality across multiple platforms
- Limited AI integration for content generation and insights
- Poor mobile user experience
- Lack of intelligent task automation
- Insufficient client relationship intelligence

Integration with Existing Enterprise Landscape

PropertyPro AI is designed to complement existing real estate infrastructure through:

- API integrations with major CRM systems
- MLS data connectivity for market analysis
- Email marketing platform synchronization
- Calendar and communication tool integration

1.2.2 High-Level Description

Primary System Capabilities

PropertyPro AI delivers six core functional areas:

1. **Property Management**: AI-powered listing creation, market analysis, and performance tracking
2. **Client Management**: Intelligent relationship tracking, lead scoring, and automated follow-ups
3. **Content Generation**: Professional marketing materials, social media posts, and client communications

4. **Task Management**: Smart workflow automation and priority-based task organization
5. **AI Assistant**: Real-time expertise and market insights through conversational interface
6. **Analytics & Reporting**: Performance metrics, market trends, and business intelligence

Major System Components

Component	Technology Stack	Primary Function
Mobile Frontend	React Native with TypeScript (version 0.71+)	Cross-platform mobile interface
Backend API	FastAPI with Python 3.11+	High-performance API services
AI Services	GPT-4.1 API integration	Content generation and intelligent assistance
Database Layer	PostgreSQL 15	Data persistence and analytics

Core Technical Approach

The system employs a clean architecture pattern with clear separation of concerns:

- **Domain Layer**: Business logic and entities
- **Application Layer**: Use cases and orchestration
- **Infrastructure Layer**: External services and data access
- **Presentation Layer**: Mobile user interface

1.2.3 Success Criteria

Measurable Objectives

Metric Category	Target	Measurement Method
User Productivity	80% reduction in content creation time	Time tracking analytics
System Performance	<2 second response times	API monitoring
User Adoption	90% daily active usage	Application analytics
Content Quality	95% user satisfaction rating	User feedback surveys

Critical Success Factors

- **Mobile-First Experience**: Seamless operation on smartphones and tablets
- **AI Accuracy**: Reliable and contextually appropriate AI-generated content
- **Integration Reliability**: Stable connections with external real estate systems
- **User Adoption**: Intuitive interface requiring minimal training

Key Performance Indicators (KPIs)

- **Technical KPIs**: API response times, system uptime, error rates
- **Business KPIs**: User engagement, content generation volume, client interaction frequency
- **User Experience KPIs**: Task completion rates, feature adoption, user satisfaction scores

1.3 SCOPE

1.3.1 In-Scope

Core Features and Functionalities

Property Management Module:

- Property listing creation and management
- AI-powered property descriptions and marketing content
- Market analysis and pricing recommendations
- Photo upload and management system
- Performance tracking and analytics

Client Relationship Management:

- Contact database with lead scoring
- Interaction history and communication tracking
- Automated follow-up reminders and scheduling
- Client preference learning and matching
- Personalized communication templates

AI-Powered Content Generation:

- Property descriptions optimized for different platforms
- Social media posts for Instagram, Facebook, and LinkedIn
- Email templates and newsletters
- Marketing brochures and presentation materials
- Market analysis reports

Task and Workflow Management:

- Smart task creation and prioritization
- Automated workflow triggers
- Progress tracking and deadline management
- Calendar integration and scheduling
- Performance optimization suggestions

Primary User Workflows

1. **New Property Listing Workflow**: Photo capture → AI description generation → Multi-platform content creation → Publishing
2. **Client Follow-up Workflow**: Interaction tracking → Automated reminders → Personalized communication → Relationship scoring
3. **Market Analysis Workflow**: Property data input → AI analysis → Pricing recommendations → Report generation

Essential Integrations

- **OpenAI GPT-4.1 API**: For content generation and intelligent assistance
- **Email Services**: For automated client communications
- **Calendar Systems**: For appointment scheduling and reminders
- **Photo Storage**: For property image management

Key Technical Requirements

- Cross-platform mobile application using React Native with TypeScript support
- High-performance backend API using FastAPI framework for Python 3.8+
- Real-time data synchronization across devices
- Offline capability for core functions
- Enterprise-grade security and data protection

1.3.2 Implementation Boundaries

System Boundaries

- **Geographic Coverage**: Initially focused on North American real estate markets
- **User Capacity**: Designed to support up to 10,000 concurrent users
- **Data Retention**: 7-year data retention policy for compliance
- **Platform Support**: iOS and Android mobile platforms

User Groups Covered

- Individual real estate agents and brokers
- Small to medium-sized real estate teams (2-50 agents)
- Independent brokerages and franchises
- Real estate coaches and trainers

Data Domains Included

- Property listings and market data
- Client contact information and interaction history
- Task and workflow management data
- Content generation templates and outputs
- Performance analytics and reporting data

1.3.3 Out-of-Scope

Explicitly Excluded Features

- **Transaction Management**: Legal document preparation and escrow management
- **Financial Services**: Mortgage calculations and lending services
- **MLS Integration**: Direct Multiple Listing Service connectivity (Phase 2)
- **Video Content**: Video generation and editing capabilities
- **Multi-Language Support**: Non-English language interfaces (Phase 2)

Future Phase Considerations

Phase 2 Enhancements:

- Advanced MLS integration and data synchronization
- Voice-to-text input and audio content generation
- Video marketing content creation
- Multi-language support for international markets

Phase 3 Enterprise Features:

- Multi-tenant architecture for large brokerages
- Advanced compliance and audit trails
- Custom branding and white-label solutions
- Enterprise-grade user management and permissions

Integration Points Not Covered

- **Legacy CRM Systems**: Custom integrations with older real estate software
- **Specialized Real Estate Tools**: Property management software, showing management systems
- **Financial Platforms**: Direct integration with banking and mortgage systems
- **Legal Platforms**: Document management and e-signature services

Unsupported Use Cases

- **Commercial Real Estate**: Focus limited to residential real estate markets
- **International Markets**: Initial release limited to North American regulations and practices
- **Enterprise Compliance**: Advanced regulatory compliance features for large institutions
- **Custom Development**: Bespoke feature development for individual clients

2. PRODUCT REQUIREMENTS

2.1 FEATURE CATALOG

2.1.1 Core Mobile Application Features

Feature ID	Feature Name	Category	Priority	Status
F-001	User Authentication System	Security	Critical	Proposed
F-002	Property Management Module	Core Business	Critical	Proposed
F-003	Client Relationship Management	Core Business	Critical	Proposed
F-004	AI Content Generation Engine	AI Services	Critical	Proposed
F-005	Task Management System	Productivity	High	Proposed
F-006	AI Assistant Chat Interface	AI Services	High	Proposed
F-007	Analytics and Reporting Dashboard	Business Intelligence	High	Proposed
F-008	Voice Command Integration	User Experience	Medium	Proposed
F-009	Offline Data Synchronization	Technical	Medium	Proposed
F-010	Push Notification System	Communication	Medium	Proposed

2.1.2 AI-Powered Features

Feature ID	Feature Name	Category	Priority	Status
F-011	Property Description Generation	AI Content	Critical	Proposed
F-012	Market Analysis and Pricing	AI Analytics	Critical	Proposed

Feature ID	Feature Name	Category	Priority	Status
F-013	Social Media Content Creation	AI Content	High	Proposed
F-014	Email Template Generation	AI Content	High	Proposed
F-015	Lead Scoring Algorithm	AI Intelligence	High	Proposed
F-016	Automated Follow-up Suggestions	AI Automation	High	Proposed
F-017	Market Trend Analysis	AI Analytics	Medium	Proposed
F-018	Client Preference Learning	AI Intelligence	Medium	Proposed

2.1.3 Integration and External Services

Feature ID	Feature Name	Category	Priority	Status
F-019	OpenAI GPT-4.1 API Integration	External API	Critical	Proposed
F-020	Photo Upload and Management	Media	High	Proposed
F-021	Email Service Integration	Communication	High	Proposed
F-022	Calendar System Integration	Productivity	Medium	Proposed
F-023	Cloud Storage Integration	Data Management	Medium	Proposed
F-024	Export and Sharing Capabilities	Data Exchange	Low	Proposed

2.2 FUNCTIONAL REQUIREMENTS

2.2.1 User Authentication System (F-001)

Require ment ID	Descripti on	Acceptance Crite ria	Priority	Comple xity
F-001-RQ-001	User Regis tration	Users can create a ccounts with email and password	Must-Ha ve	Low
F-001-RQ-002	Secure Lo gin	Users can authenti cate using JWT tok ens with bcrypt pa ssword hashing	Must-Ha ve	Medium
F-001-RQ-003	Password Reset	Users can reset for gotten passwords via email	Should-H ave	Medium
F-001-RQ-004	Session M anagemen t	Automatic token re fresh and secure s ession handling	Must-Ha ve	Medium

Technical Specifications:

- Input Parameters: Email, password, optional profile information
- Output/Response: JWT access token, refresh token, user profile data
- Performance Criteria: Authentication response time < 2 seconds
- Data Requirements: User credentials stored with bcrypt hashing

Validation Rules:

- Business Rules: Email must be unique, password minimum 8 characters
- Data Validation: Email format validation, password strength requirements
- Security Requirements: JWT token expiration, secure password storage
- Compliance Requirements: GDPR compliance for user data handling

2.2.2 Property Management Module (F-002)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-002-RQ-001	Property Listing Creation	Users can create property listings with basic details	Must-Have	Medium
F-002-RQ-002	Photo Upload Management	Users can upload and manage multiple property photos	Must-Have	Medium
F-002-RQ-003	Property Status Tracking	System tracks property status (active, pending, sold)	Must-Have	Low
F-002-RQ-004	Property Search and Filter	Users can search and filter properties by criteria	Should-Have	Medium

Technical Specifications:

- Input Parameters: Property details (address, price, bedrooms, bathrooms, sqft), photos
- Output/Response: Property ID, listing confirmation, photo URLs
- Performance Criteria: Property creation < 5 seconds, photo upload < 10 seconds per image
- Data Requirements: Property database with indexed search fields

Validation Rules:

- Business Rules: Required fields validation, price must be positive number
- Data Validation: Address format validation, photo file type restrictions
- Security Requirements: User can only access their own properties
- Compliance Requirements: Property data retention policies

2.2.3 AI Content Generation Engine (F-004)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-004-RQ-001	Property Description Generation	AI generates compelling property descriptions using GPT-4.1 API	Must-Have	High
F-004-RQ-002	Content Customization	Users can specify tone and style preferences	Should-Have	Medium
F-004-RQ-003	Multi-Platform Content	Generate content optimized for different platforms	Should-Have	High
F-004-RQ-004	Content Quality Control	AI-generated content meets quality standards	Must-Have	Medium

Technical Specifications:

- Input Parameters: Property details, content type, tone preferences
- Output/Response: Generated content text, confidence score, suggestions
- Performance Criteria: Content generation < 5 seconds with 1 million token context support
- Data Requirements: Integration with GPT-4.1 API for enhanced performance

Validation Rules:

- Business Rules: Content must be appropriate and professional
- Data Validation: Input sanitization for AI prompts
- Security Requirements: API key protection, rate limiting
- Compliance Requirements: Content compliance with advertising standards

2.2.4 Client Relationship Management (F-003)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-003-RQ-001	Contact Database Management	Users can store and manage client contact information	Must-Have	Medium
F-003-RQ-002	Interaction History Tracking	System records all client interactions with timestamps	Must-Have	Medium
F-003-RQ-003	Lead Scoring System	AI automatically scores leads based on behavior and data	Should-Have	High
F-003-RQ-004	Follow-up Reminders	Automated reminders for client follow-ups	Should-Have	Medium

Technical Specifications:

- Input Parameters: Client details, interaction data, communication preferences
- Output/Response: Client ID, lead score, interaction history, reminder schedules
- Performance Criteria: Client data retrieval < 1 second, lead scoring < 3 seconds
- Data Requirements: Client database with relationship tracking

Validation Rules:

- Business Rules: Contact information validation, duplicate prevention
- Data Validation: Email and phone number format validation
- Security Requirements: Client data encryption, access control

- Compliance Requirements: CRM data privacy regulations

2.2.5 Task Management System (F-005)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-005-RQ-001	Task Creation and Assignment	Users can create tasks with priorities and deadlines	Must-Have	Low
F-005-RQ-002	Smart Task Prioritization	AI suggests task priorities based on business impact	Should-Have	Medium
F-005-RQ-003	Progress Tracking	Users can track task completion and progress	Must-Have	Low
F-005-RQ-004	Workflow Automation	System creates tasks automatically based on triggers	Could-Have	High

Technical Specifications:

- Input Parameters: Task details, priority level, due date, category
- Output/Response: Task ID, priority score, completion status, progress metrics
- Performance Criteria: Task operations < 1 second, bulk operations < 5 seconds
- Data Requirements: Task database with status tracking and history

Validation Rules:

- Business Rules: Due dates must be future dates, priority levels defined
- Data Validation: Task description length limits, category validation
- Security Requirements: User can only access their own tasks
- Compliance Requirements: Task data retention policies

2.2.6 AI Assistant Chat Interface (F-006)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-006-RQ-001	Real-time Chat Interface	Users can chat with AI assistant in real-time	Must-Have	Medium
F-006-RQ-002	Context-Aware Responses	AI maintains conversation context with 1 million token support	Should-Have	High
F-006-RQ-003	Real Estate Knowledge Base	AI provides expert real estate advice and information	Must-Have	High
F-006-RQ-004	Conversation History	System stores and retrieves chat history	Should-Have	Medium

Technical Specifications:

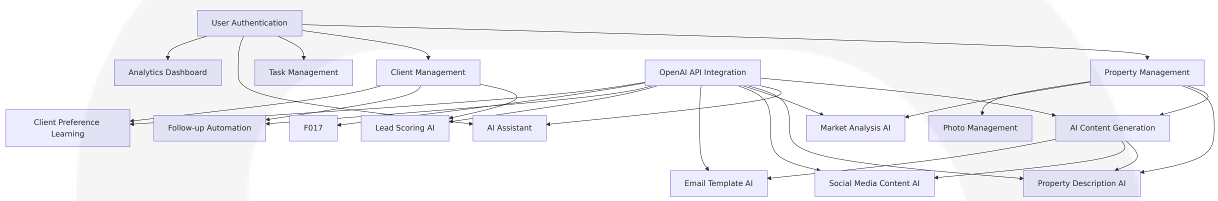
- Input Parameters: User messages, conversation context, user preferences
- Output/Response: AI responses, confidence scores, suggested actions
- Performance Criteria: Response time < 3 seconds, context retention across sessions
- Data Requirements: GPT-4.1 API integration with June 2024 knowledge cutoff

Validation Rules:

- Business Rules: Responses must be relevant to real estate domain
- Data Validation: Input sanitization, message length limits
- Security Requirements: Conversation data encryption, user privacy
- Compliance Requirements: AI response accuracy and liability considerations

2.3 FEATURE RELATIONSHIPS

2.3.1 Feature Dependencies Map



2.3.2 Integration Points

Integration Point	Primary Feature	Secondary Feature	Dependency Type
User Session	F-001	All Features	Authentication Required
AI Content Engine	F-004	F-011, F-013, F-014	Service Dependency
Property Data	F-002	F-011, F-012, F-020	Data Dependency
Client Data	F-003	F-015, F-016, F-018	Data Dependency
OpenAI API	F-019	All AI Features	External Service

2.3.3 Shared Components

Component Name	Used By Features	Purpose
AI Service Layer	F-004, F-006, F-011-F-018	GPT-4.1 API integration and response handling
Data Storage Layer	F-002, F-003, F-005, F-007	Database operations and data persistence
Authentication Middleware	All Features	User verification and session management

Component Name	Used By Features	Purpose
React Native TypeScript Components	All UI Features	Cross-platform mobile interface components

2.4 IMPLEMENTATION CONSIDERATIONS

2.4.1 Technical Constraints

Feature Category	Constraint Type	Description	Impact
AI Features	API Rate Limits	GPT-4.1 API usage limitations and costs	High
Mobile App	React Native TypeScript compatibility with version 0.71+	Medium	
Database	Concurrent Users	PostgreSQL performance with 10,000+ concurrent users	Medium
Storage	Media Files	Property photo storage and bandwidth requirements	Medium

2.4.2 Performance Requirements

Feature	Response Time Target	Throughput Target	Scalability Requirement
User Authentication	< 2 seconds	1000 requests/minute	Horizontal scaling
Property Management	< 3 seconds	500 operations/minute	Database optimization

Feature	Response Time Target	Throughput Target	Scalability Requirement
AI Content Generation	< 5 seconds	100 requests/minute	API rate management
Real-time Chat	< 3 seconds	200 messages/minute	WebSocket optimization

2.4.3 Security Implications

Security Domain	Requirements	Implementation
Data Protection	Encryption at rest and in transit	AES-256 encryption, TLS 1.3
API Security	Rate limiting and authentication	JWT tokens, API key management
User Privacy	GDPR compliance	Data anonymization, consent management
AI Safety	Content filtering and validation	Response sanitization, abuse detection

2.4.4 Maintenance Requirements

Maintenance Type	Frequency	Scope	Automation Level
AI Model Updates	Quarterly	GPT-4.1 version updates	Semi-automated
Security Patches	Monthly	Framework and dependency updates	Automated
Database Optimization	Weekly	Query performance and indexing	Automated monitoring
Content Moderation	Daily	AI-generated content review	Automated with manual oversight

2.5 TRACEABILITY MATRIX

Business Requirement	Feature ID	Functional Requirement	Test Case Reference
Mobile-first real estate assistant	F-001 to F-010	All core mobile features	TC-001 to TC-010
AI-powered content generation	F-004, F-011-F-014	GPT-4.1 integration for content creation	TC-011 to TC-014
Client relationship management	F-003, F-015-F-016	CRM functionality with AI enhancement	TC-015 to TC-016
Property management automation	F-002, F-011-F-012	Property listing with AI analysis	TC-017 to TC-018
Real-time AI assistance	F-006, F-019	Chat interface with 1M token context	TC-019
Cross-platform compatibility	All Features	React Native TypeScript implementation	TC-020

This comprehensive product requirements specification provides a detailed breakdown of PropertyPro AI's features into discrete, testable components while maintaining traceability to business objectives and technical implementation considerations. The requirements are structured to support agile development methodologies and ensure comprehensive test coverage across all system capabilities.

3. TECHNOLOGY STACK

3.1 PROGRAMMING LANGUAGES

3.1.1 Backend Development

Language	Version	Platform/Component	Justification
Python	3.11+	Backend API Services	Modern Python version with enhanced performance and type hints support for FastAPI development
TypeScript	5.0+	Type Definitions & Validation	Strong typing for API contracts, data models, and enhanced developer experience
SQL	PostgreSQL 15	Database Queries	Native PostgreSQL syntax for complex queries and stored procedures

Selection Criteria:

- **Python 3.11+**: FastAPI framework requires modern Python with type hints for high-performance API development
- **TypeScript**: Ensures type safety across the entire application stack and improves maintainability
- **SQL**: Direct database access for complex analytics and reporting queries

Constraints:

- Python 3.11+ required for FastAPI performance optimizations and enhanced asyncio support
- TypeScript compatibility with React Native 0.71+ type definitions

3.1.2 Frontend Development

Language	Version	Platform/Component	Justification
TypeScript	5.0+	React Native Mobile Ap	React Native 0.71+ includes built-in TypeScript support by

Language	Version	Platform/Component	Justification
		p	default with accurate type declarations
JavaScript (ES2022)	Latest	Runtime Environment	Native mobile platform compatibility and performance

Selection Criteria:

- **TypeScript**: React Native 0.71 provides first-class TypeScript support with built-in declarations
- **Modern JavaScript**: Required for React Native runtime and native module integration

3.2 FRAMEWORKS & LIBRARIES

3.2.1 Backend Framework Stack

Framework/Library	Version	Purpose	Justification
FastAPI	0.115.0+	Web Framework	Modern, high-performance framework with automatic API documentation and type validation
Pydantic	2.0+	Data Validation	Integrated with FastAPI for data validation and serialization
SQLAlchemy	2.0+	ORM	Modern async SQLAlchemy 2.0 with enhanced performance and type safety
Uvicorn	0.24.0+	ASGI Server	Starlette-based server for FastAPI applications
Alembic	1.13.0+	Database Migrations	Database schema versioning and migration management

Compatibility Requirements:

- FastAPI leverages Python type hints and Pydantic for automatic validation and documentation
- Async SQLAlchemy 2.0 with asyncpg for PostgreSQL integration

3.2.2 Mobile Framework Stack

Framework/Library	Version	Purpose	Justification
React Native	0.71+	Mobile Framework	Version 0.71+ includes TypeScript by default and built-in type declarations
React	18.2+	UI Library	Core React library for component-based architecture
React Navigation	6.0+	Navigation	Type-safe navigation with TypeScript support
Zustand	4.4+	State Management	Lightweight state management with TypeScript integration
Axios	1.6+	HTTP Client	Promise-based HTTP client with TypeScript definitions

Compatibility Requirements:

- React Native 0.71+ eliminates need for @types/react-native package
- TypeScript 4.1+ compatibility for React Navigation integration

3.2.3 AI Integration Framework

Framework/Library	Version	Purpose	Justification
OpenAI Python SDK	1.0+	AI API Integration	Official SDK for GPT-4.1 API integration
LangChain	0.1+	AI Orchestration	Framework for building AI-powered applications

Framework/ Library	Version	Purpose	Justification
Tiktoken	0.5+	Token Manag ement	OpenAI token counting an d management

3.3 OPEN SOURCE DEPENDENCIES

3.3.1 Backend Dependencies

```
# Core Framework Dependencies
fastapi[standard]>=0.115.0,<0.116.0
uvicorn[standard]>=0.24.0,<0.25.0
pydantic>=2.0.0,<3.0.0
pydantic-settings>=2.0.0,<3.0.0

#### Database Dependencies
sqlalchemy[asyncio]>=2.0.0,<2.1.0
asyncpg>=0.29.0,<0.30.0
alembic>=1.13.0,<1.14.0

#### AI Integration Dependencies
openai>=1.0.0,<2.0.0
langchain>=0.1.0,<0.2.0
tiktoken>=0.5.0,<0.6.0

#### Security Dependencies
python-jose[cryptography]>=3.3.0,<4.0.0
passlib[bcrypt]>=1.7.4,<2.0.0
python-multipart>=0.0.6,<0.1.0

#### Utility Dependencies
python-dotenv>=1.0.0,<1.1.0
httpx>=0.25.0,<0.26.0
```

3.3.2 Frontend Dependencies

```json
{
  "dependencies": {
    "react": "18.2.0",
    "react-native": "0.71.0",
    "@react-navigation/native": "^6.1.0",
    "@react-navigation/stack": "^6.3.0",
    "zustand": "^4.4.0",
    "axios": "^1.6.0",
    "react-native-vector-icons": "^10.0.0",
    "react-native-gesture-handler": "^2.14.0",
    "react-native-reanimated": "^3.6.0"
  },
  "devDependencies": {
    "@types/react": "^18.2.0",
    "@types/react-native": "^0.71.0",
    "@typescript-eslint/eslint-plugin": "^6.0.0",
    "@typescript-eslint/parser": "^6.0.0",
    "typescript": "^5.0.0"
  }
}
```

3.3.3 Package Registries

Registry	Purpose	Components
PyPI	Python packages	Backend dependencies, AI libraries
npm	JavaScript packages	React Native, TypeScript definitions
GitHub	Source repositories	Custom forks, development tools

3.4 THIRD-PARTY SERVICES

3.4.1 AI Services

Service	Version/ Model	Purpose	Integration Method
OpenAI A PI	GPT-4.1	Content Ge neration	GPT-4.1 with 1 million token context window for enhance d AI capabilities
OpenAI A PI	GPT-4.1 mi ni	Cost-effecti ve AI	Reduced latency and 83% co st reduction while maintainin g performance
OpenAI A PI	GPT-4.1 na no	High-speed AI	Fastest model for classificatio n and autocompletion tasks

API Specifications:

- Knowledge cutoff: June 2024
- Context window: Up to 1 million tokens
- Authentication: API key-based with rate limiting

3.4.2 Authentication Services

Service	Purpose	Implementation
JWT Tokens	User Authentication	Self-managed with python-jose
bcrypt	Password Hashing	Integrated with passlib
OAuth 2.0	Third-party Auth	Future integration capability

3.4.3 Monitoring and Analytics

Service	Purpose	Integration
Application Logs	Error tracking	Python logging module
Performance Metrics	API monitoring	FastAPI middleware
Health Checks	System monitoring	Custom health endpoints

3.5 DATABASES & STORAGE

3.5.1 Primary Database

Component	Version	Purpose	Justification
PostgreSQL	15+	Primary Database	Modern PostgreSQL with async support and JSON capabilities
asyncpg	0.29+	Database Driver	High-performance async PostgreSQL driver

Database Configuration:

```
# Docker Compose Configuration
postgres:
  image: postgres:15-alpine
  environment:
    POSTGRES_DB: propertypro_ai
    POSTGRES_USER: app_user
    POSTGRES_PASSWORD: ${DB_PASSWORD}
  volumes:
    - postgres_data:/var/lib/postgresql/data
```

3.5.2 Data Persistence Strategy

Data Type	Storage Method	Justification
User Data	PostgreSQL Tables	ACID compliance, relational integrity
Property Data	PostgreSQL with JSON	Structured data with flexible attributes
AI Conversations	PostgreSQL JSONB	Searchable conversation history
File Uploads	Local File System	Simple file storage for property images
Session Data	JWT Tokens	Stateless authentication

3.5.3 Caching Solutions

Component	Purpose	Implementation
Application Cache	API Response Caching	In-memory Python dictionaries
Database Connection Pool	Connection Management	SQLAlchemy connection pooling
Static File Cache	Asset Delivery	React Native asset bundling

3.6 DEVELOPMENT & DEPLOYMENT

3.6.1 Development Tools

Tool	Version	Purpose	Justification
Docker	24.0+	Containerization	Consistent development environment with PostgreSQL
Docker Compose	2.0+	Multi-service Orchestration	Local development stack management
Poetry	1.6+	Python Dependency Management	Deterministic dependency resolution
ESLint	8.0+	TypeScript Linting	Code quality enforcement
Prettier	3.0+	Code Formatting	Consistent code style

3.6.2 Build System

```
# Backend Dockerfile
FROM python:3.11-bookworm
ENV PYTHONUNBUFFERED=1
WORKDIR /app
COPY requirements.txt ./
```

```
RUN pip install -r requirements.txt
COPY . .
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

3.6.3 Containerization Strategy

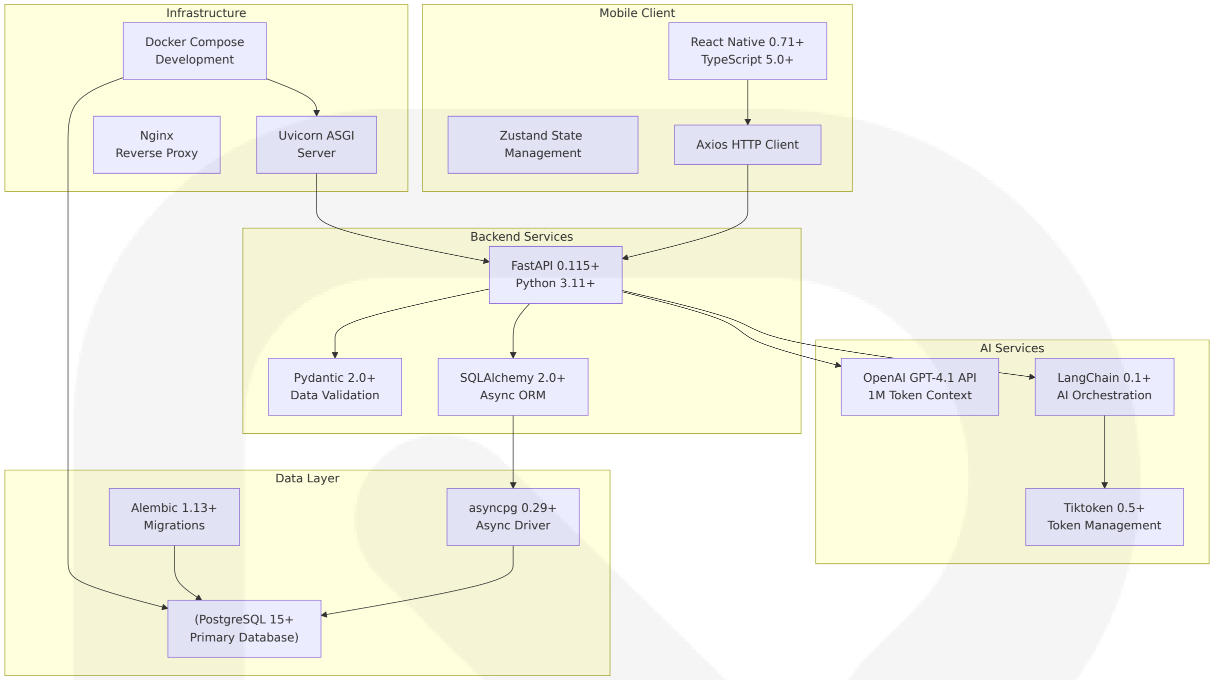
Container	Base Image	Purpose
Backend API	python:3.11-bookworm	FastAPI application with optimized Python runtime
Database	postgres:15-alpine	Lightweight PostgreSQL container
Development	Multi-stage build	Combined development environment

3.6.4 CI/CD Requirements

Stage	Tools	Purpose
Code Quality	ESLint, Ruff	Linting and formatting
Testing	pytest, Jest	Unit and integration testing
Build	Docker	Container image creation
Deployment	Docker Compose	Local and staging deployment

3.7 ARCHITECTURE INTEGRATION

3.7.1 Technology Stack Diagram



3.7.2 Security Integration

Security Layer	Technology	Implementation
API Authentication	JWT + bcrypt	Token-based auth with secure password hashing
Data Validation	Pydantic	Input sanitization and type validation
Database Security	PostgreSQL	Connection encryption and parameterized queries
Transport Security	HTTPS/TLS	Encrypted client-server communication

3.7.3 Performance Considerations

Component	Optimization	Expected Performance
FastAPI	Async/await patterns	High performance comparable to NodeJS and Go

Component	Optimization	Expected Performance
PostgreSQL	Connection pooling	<100ms query response times
React Native	Native compilation	60fps mobile performance
AI API	Token optimization	<5 second content generation

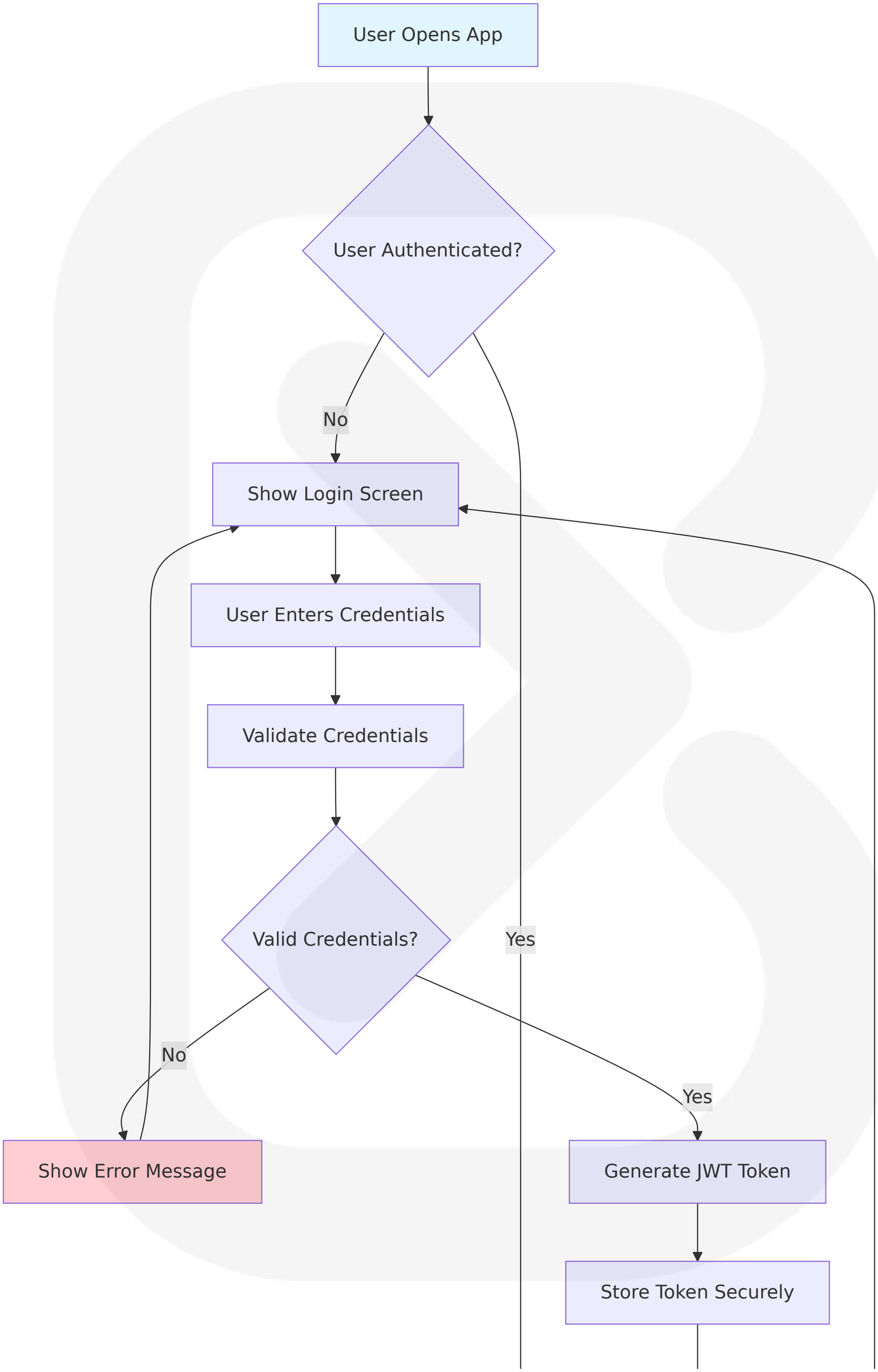
This technology stack provides a modern, scalable foundation for PropertyPro AI that leverages the latest versions of proven technologies while maintaining compatibility and performance requirements. The selection prioritizes developer experience, type safety, and production readiness across all components.

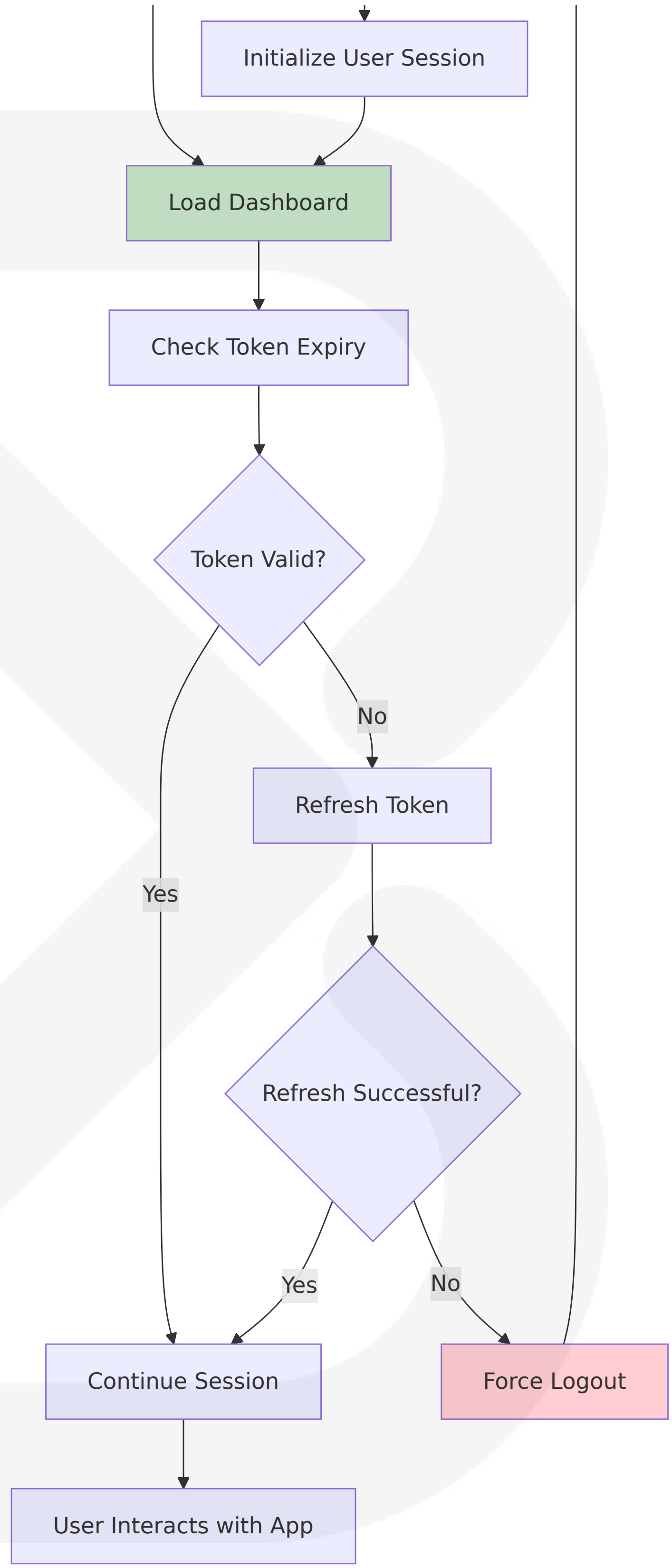
4. PROCESS FLOWCHART

4.1 SYSTEM WORKFLOWS

4.1.1 Core Business Processes

User Authentication and Session Management


User Opens App
User Authenticated?
No
Yes
Show Login Screen
User Enters Credentials
Validate Credentials
Valid Credentials?
No
Yes
Show Error Message
Generate JWT Token
Store Token Securely



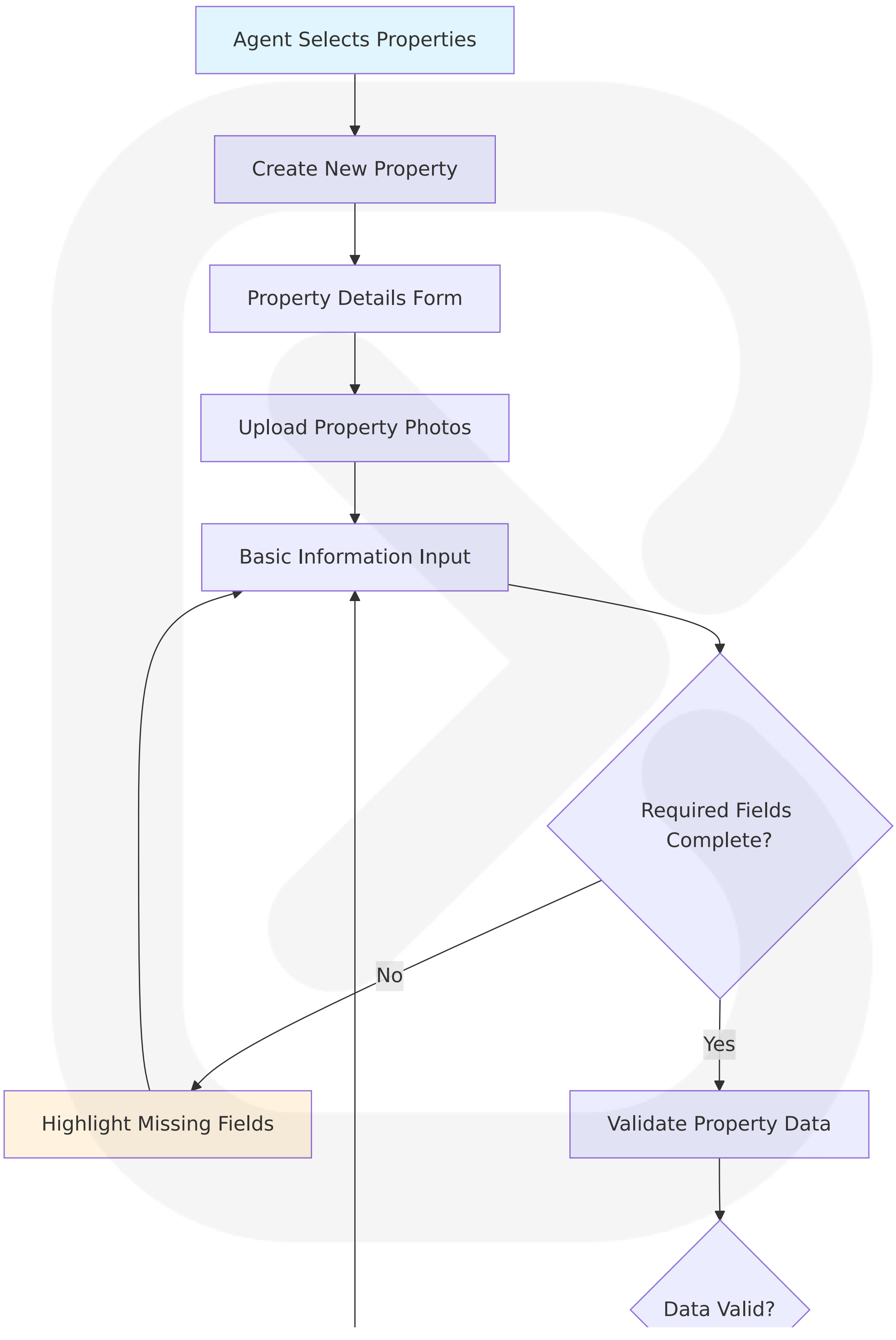
Validation Rules:

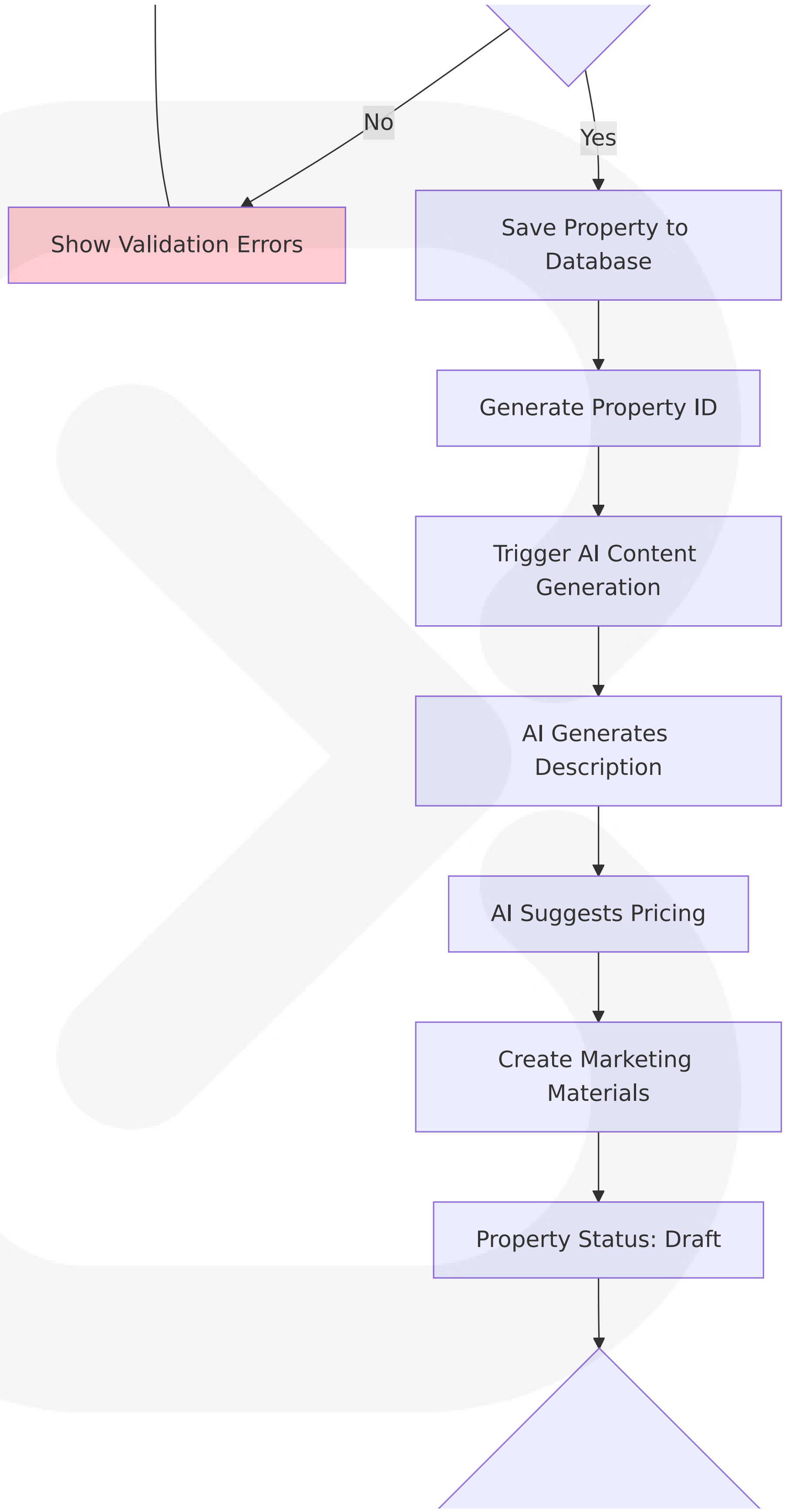
- Email format validation using TypeScript type definitions
- Password minimum 8 characters with complexity requirements
- JWT token expiration set to 24 hours with refresh capability
- Maximum 3 failed login attempts before temporary lockout

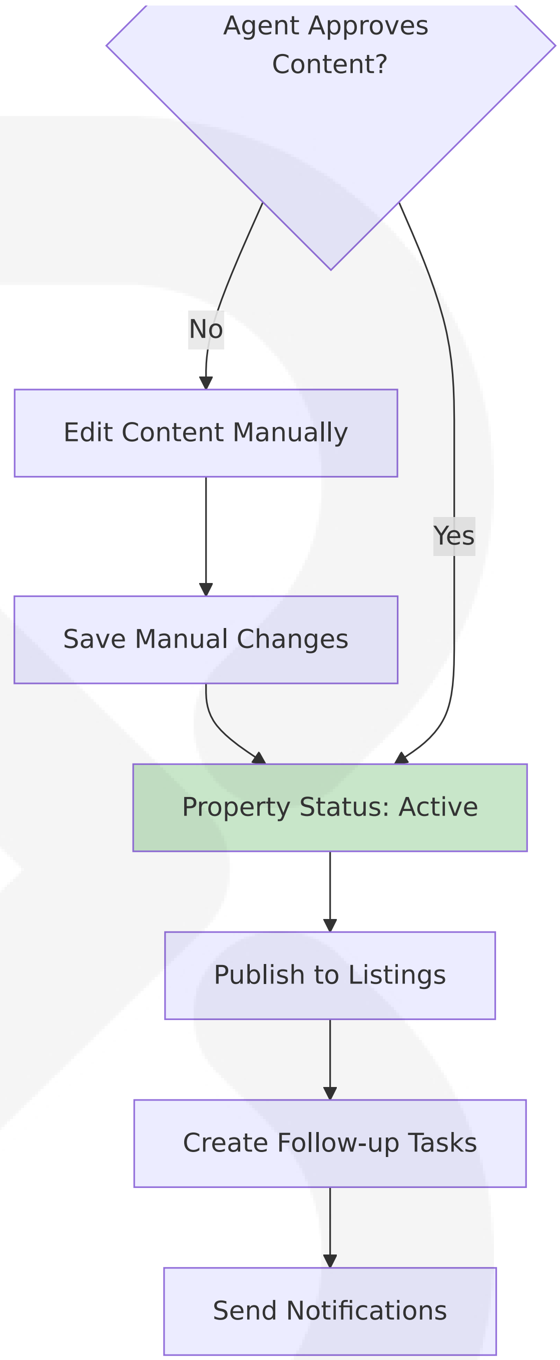
Performance Criteria:

- Authentication response time: < 2 seconds
- Token refresh: < 1 second
- Session initialization: < 3 seconds

Property Management Workflow


Agent Selects Properties
Create New Property
Property Details Form
Upload Property Photos
Basic Information Input
Required Fields Complete?
No
Yes
Highlight Missing Fields
Validate Property Data
Data Valid?


No
Yes
Show Validation Errors
Save Property to Database
Generate Property ID
Trigger AI Content Generation
AI Generates Description
AI Suggests Pricing
Create Marketing Materials
Property Status: Draft



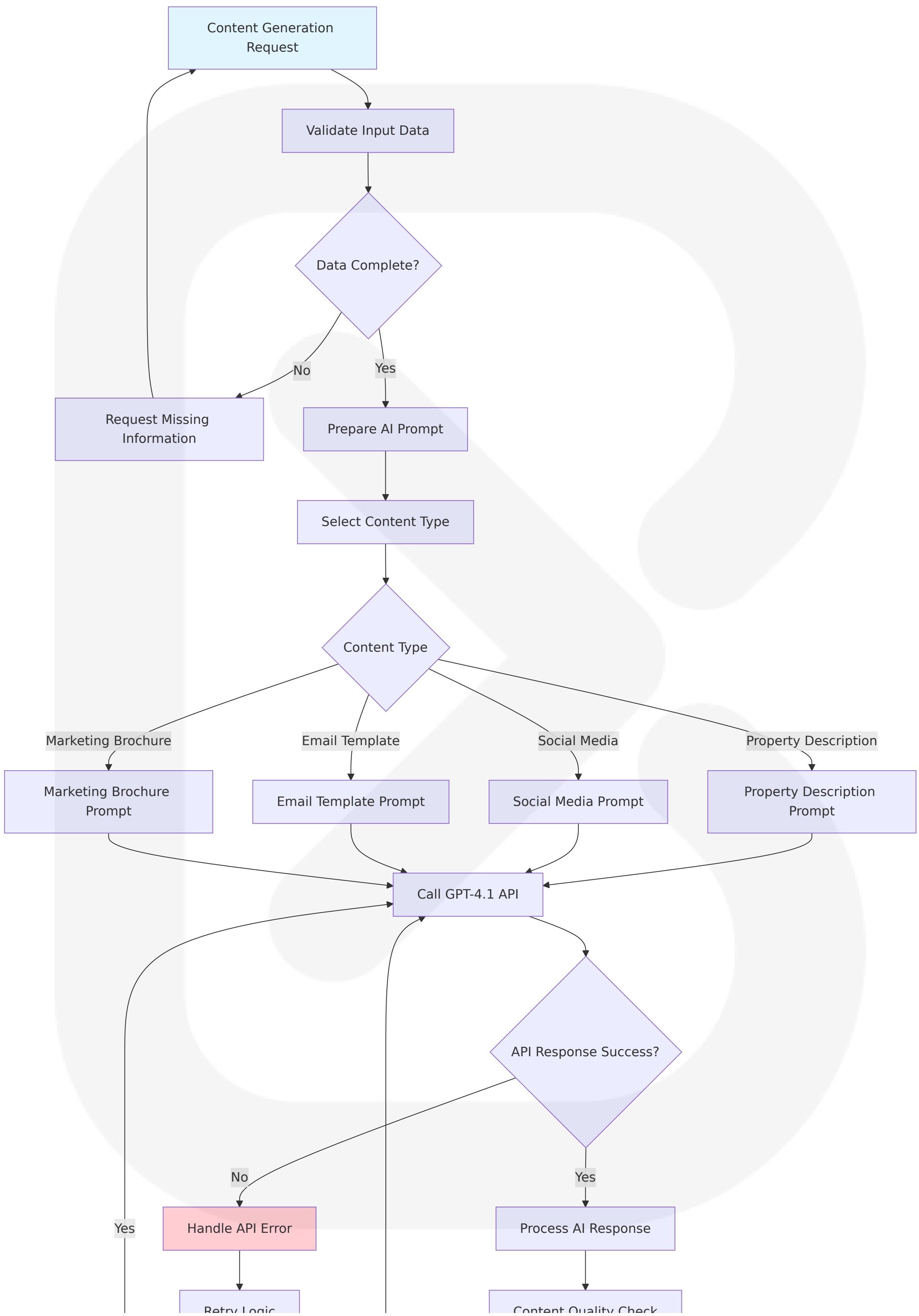
Business Rules:

- Property address must be unique within agent's portfolio
- Price must be positive number with maximum 2 decimal places
- AI content generation using GPT-4.1 with 1 million token context window
- Photos limited to 20 images per property, max 5MB each

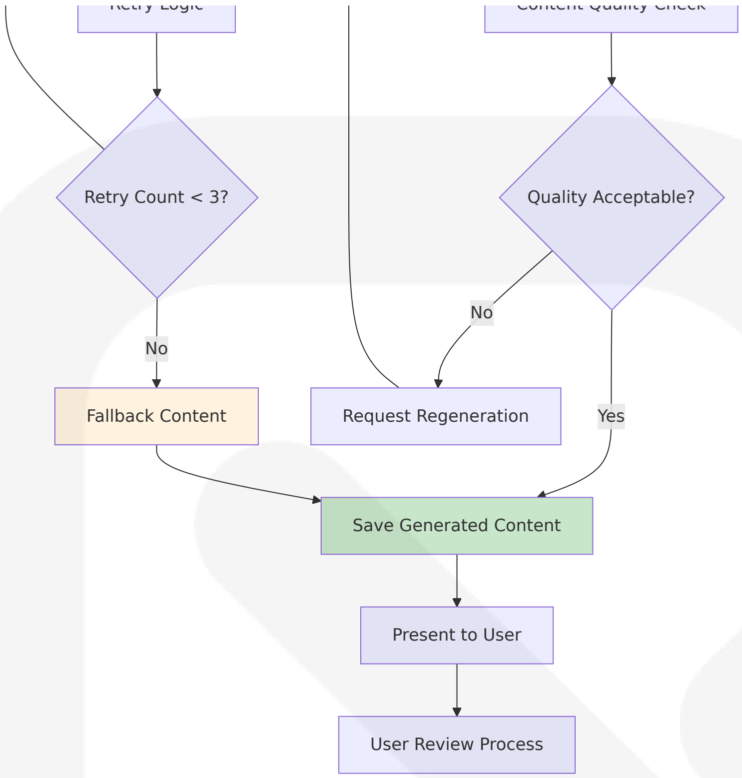
State Transitions:

- Draft → Active → Under Offer → Sold/Rented
- Active → Withdrawn → Draft (reactivation possible)

AI Content Generation Process



Content Generation Request
Validate Input Data
Data Complete?
No
Yes
Request Missing Information
Prepare AI Prompt
Select Content Type
Content Type
Marketing Brochure
Email Template
Social Media
Property Description
Marketing Brochure Prompt
Email Template Prompt
Social Media Prompt
Property Description Prompt
Call GPT-4.1 API
API Response Success?
No
Yes
Yes
Handle API Error
Process AI Response



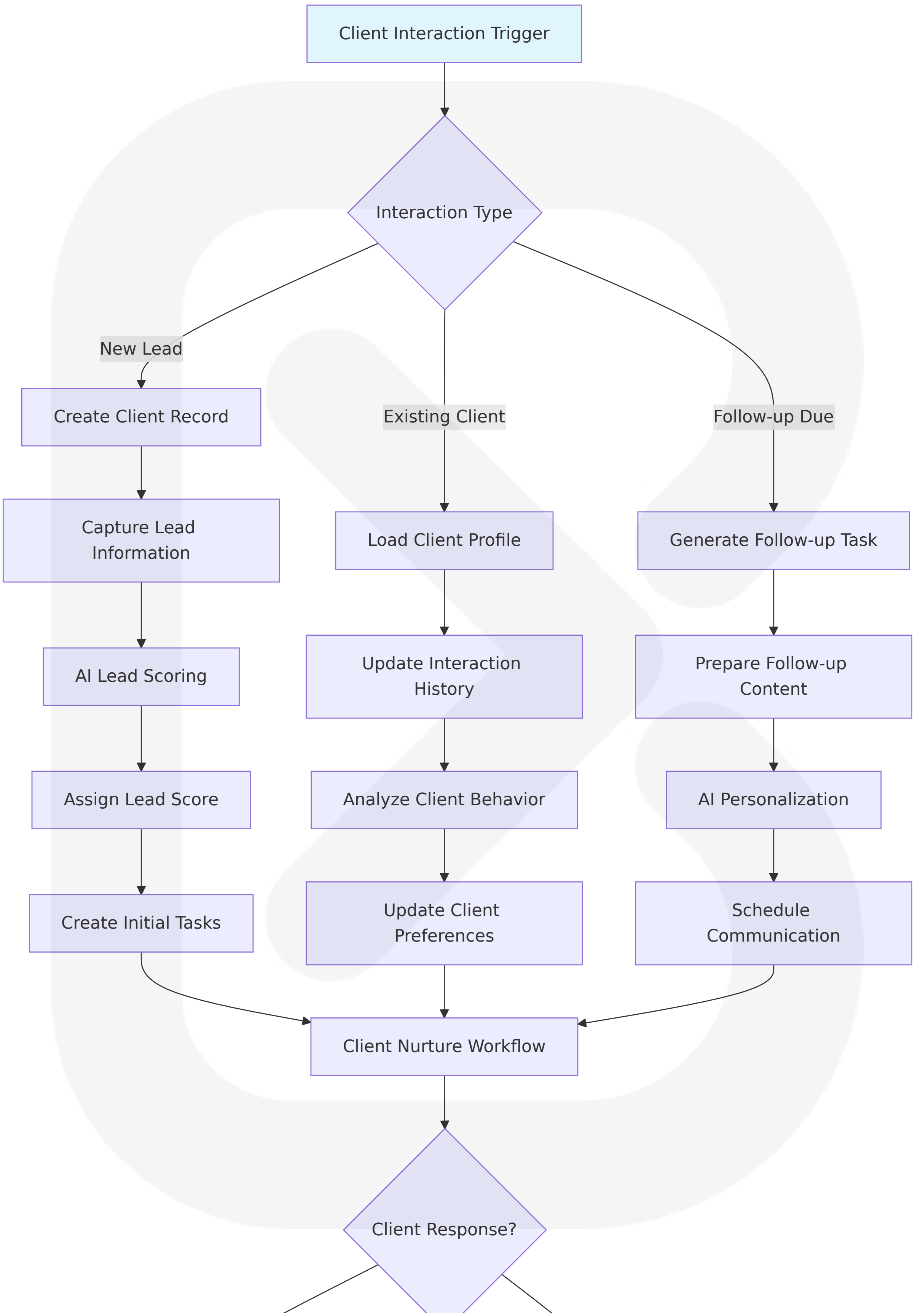
AI Integration Specifications:

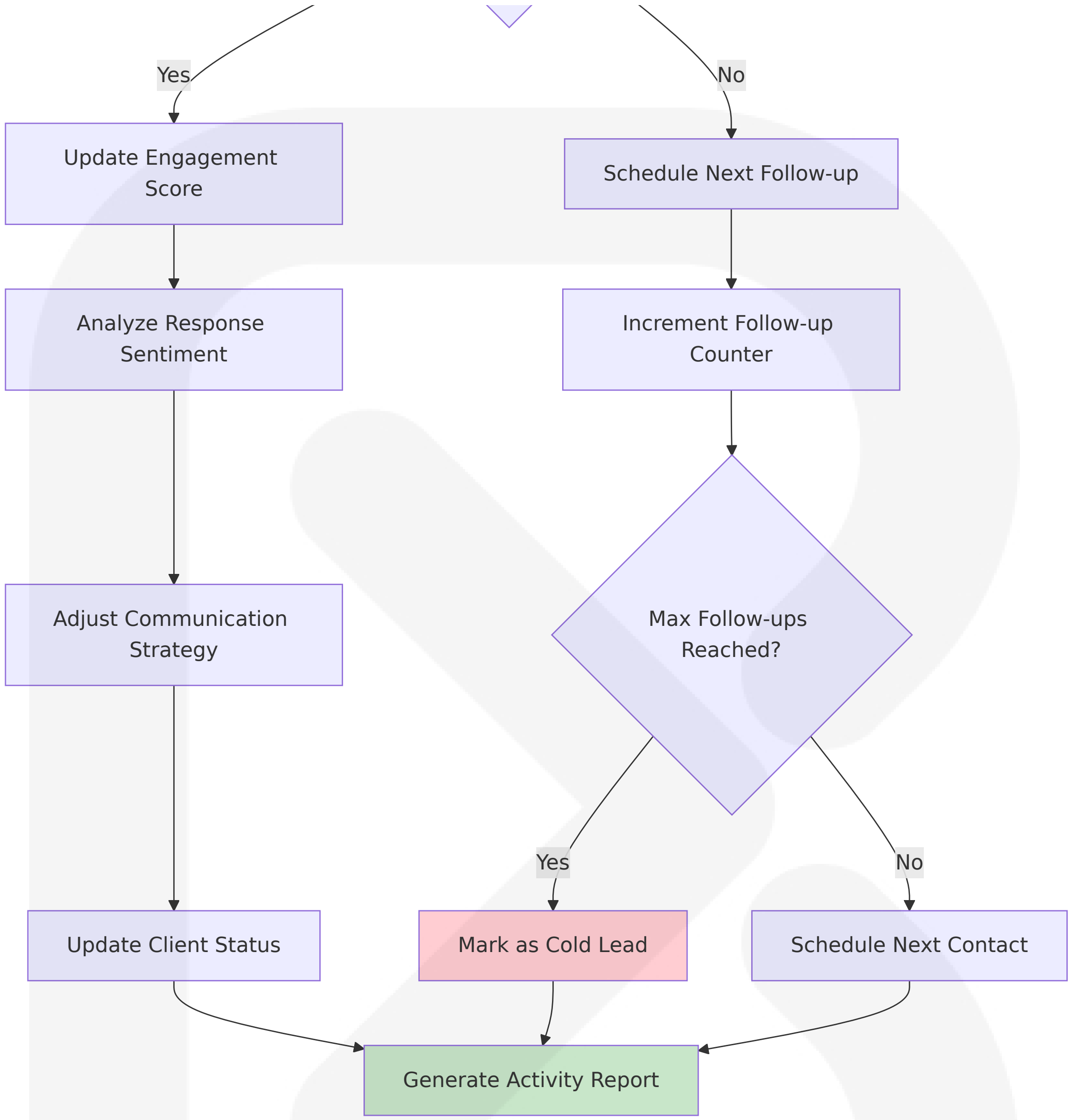
- GPT-4.1 API with enhanced coding and instruction following capabilities
- Knowledge cutoff: June 2024
- Maximum token limit: 4,000 tokens per request
- Response timeout: 30 seconds
- Rate limiting: 100 requests per minute per user

Error Handling:

- API timeout: Retry with exponential backoff
- Rate limit exceeded: Queue request for later processing
- Invalid response: Use fallback templates
- Network error: Cache request for offline processing

Client Relationship Management Flow


Client Interaction Trigger
Interaction Type
New Lead
Existing Client
Follow-up Due
Create Client Record
Capture Lead Information
AI Lead Scoring
Assign Lead Score
Create Initial Tasks
Load Client Profile
Update Interaction History
Analyze Client Behavior
Update Client Preferences
Generate Follow-up Task
Prepare Follow-up Content
AI Personalization
Schedule Communication
Client Nurture Workflow
Client Response?



Lead Scoring Algorithm:

- Email engagement: 0-25 points
- Property viewing history: 0-30 points
- Response time: 0-20 points
- Budget qualification: 0-25 points
- Total score: 0-100 points (Hot: 80+, Warm: 50-79, Cold: <50)

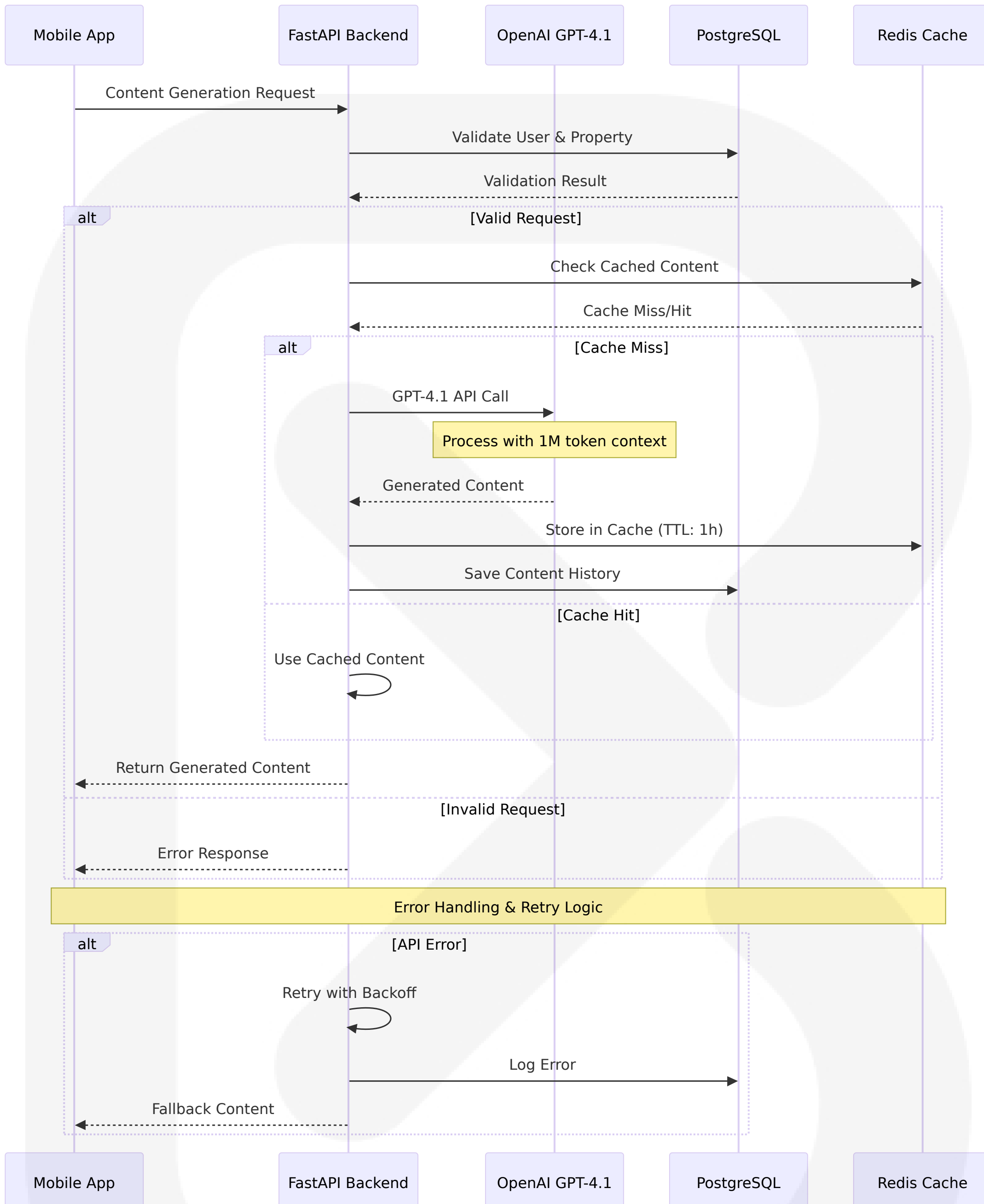
Communication Timing Rules:

- Initial response: Within 5 minutes

- Follow-up sequence: Day 1, 3, 7, 14, 30
- Maximum follow-ups: 5 attempts
- Re-engagement cycle: Every 90 days for cold leads

4.1.2 Integration Workflows

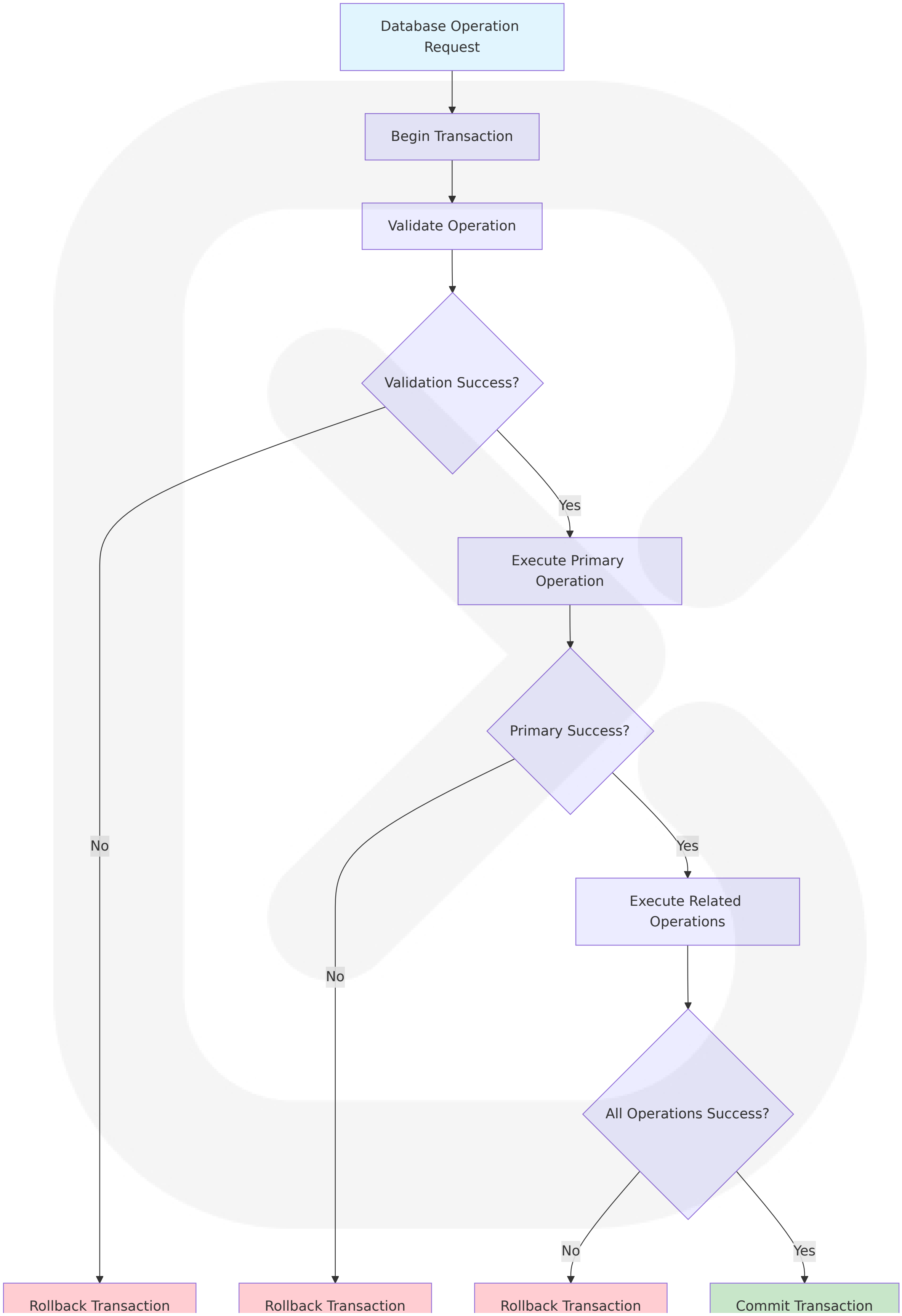
OpenAI API Integration Flow

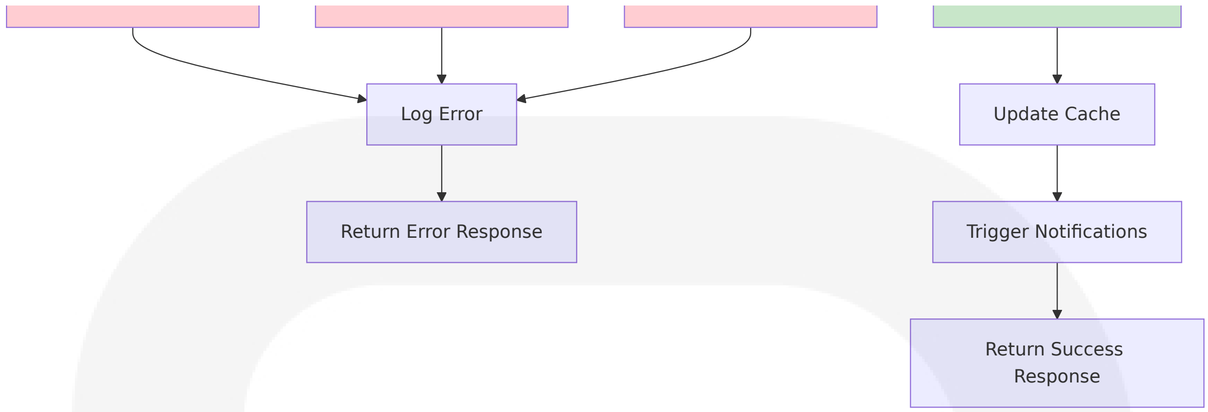


Integration Specifications:

- Context window: Up to 1 million tokens for enhanced processing
- Request timeout: 30 seconds
- Retry policy: 3 attempts with exponential backoff (1s, 2s, 4s)
- Rate limiting: 100 requests/minute per user
- Caching: 1-hour TTL for generated content

Database Transaction Workflow


Database Operation Request
Begin Transaction
Validate Operation
Validation Success?
Yes
No
Execute Primary Operation
Primary Success?
Yes
No
Execute Related Operations
All Operations Success?
No
Yes
Rollback Transaction
Rollback Transaction
Rollback Transaction
Commit Transaction



Transaction Boundaries:

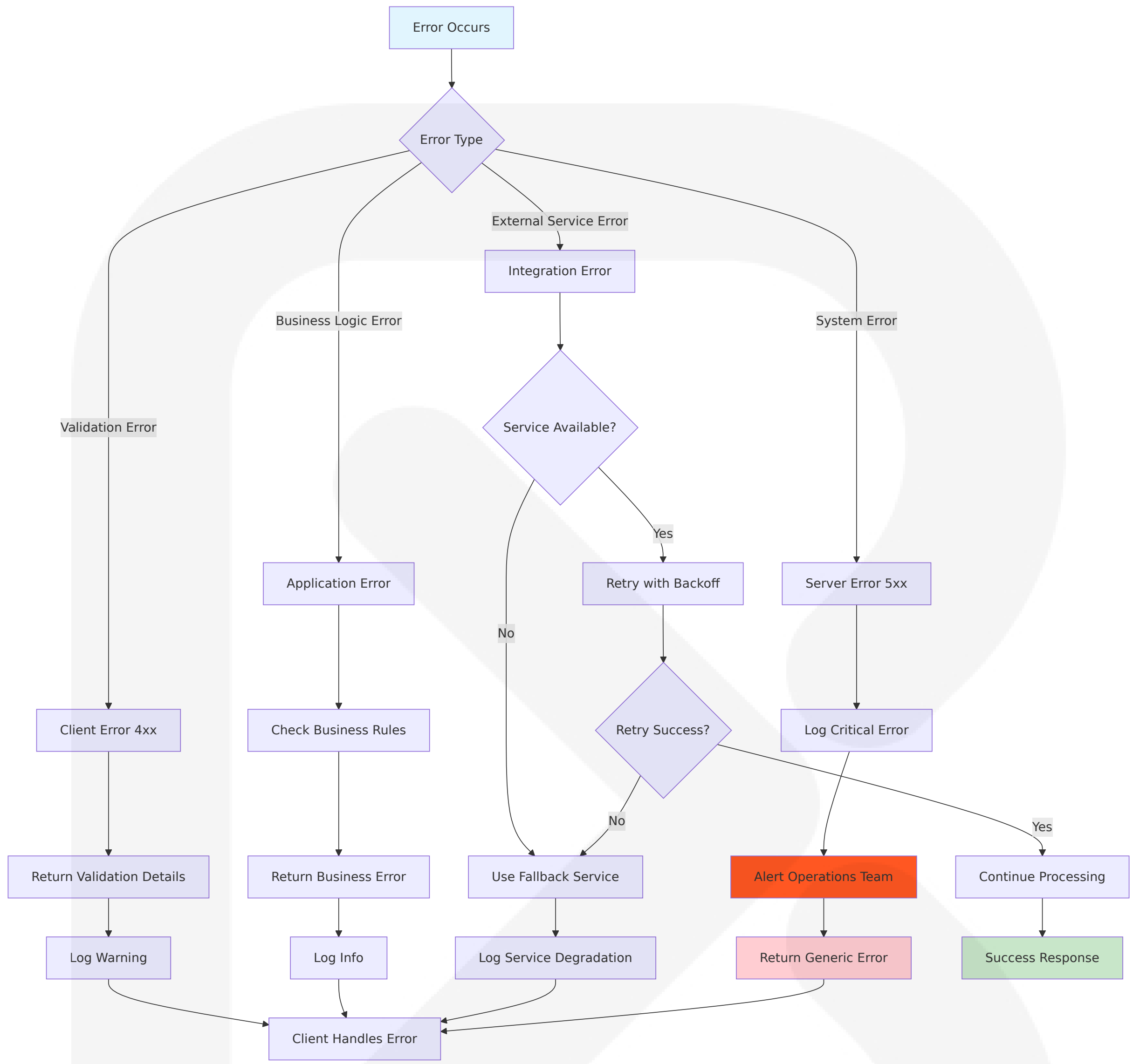
- Property creation: Property record + initial tasks + audit log
- Client update: Client record + interaction history + lead score recalculation
- Content generation: Content record + usage tracking + cache update

Consistency Requirements:

- ACID compliance for all financial and client data
- Eventual consistency acceptable for analytics and reporting
- Read replicas for performance-critical queries

4.2 ERROR HANDLING AND RECOVERY

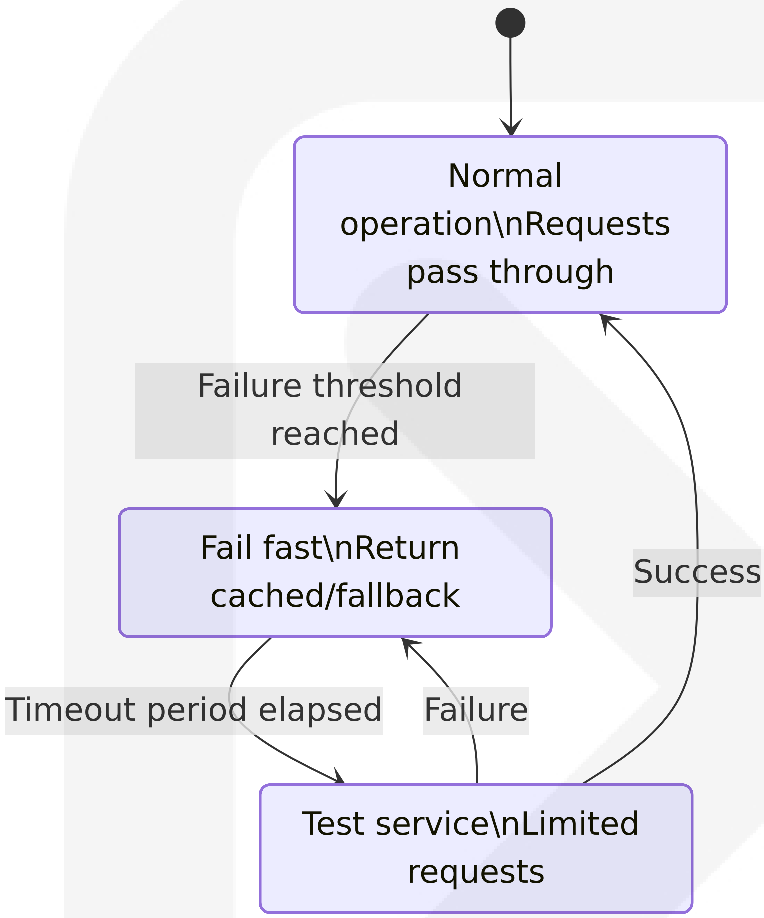
4.2.1 Error Classification and Handling



Error Response Format:

```
interface ErrorResponse {
  error: {
    code: string;
    message: string;
    details?: any;
    timestamp: string;
    requestId: string;
  };
}
```

4.2.2 Circuit Breaker Pattern for External Services

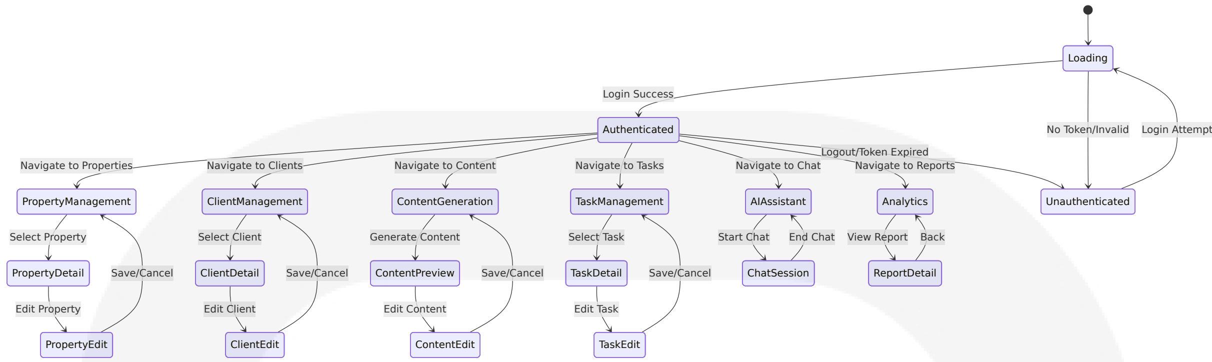


Circuit Breaker Configuration:

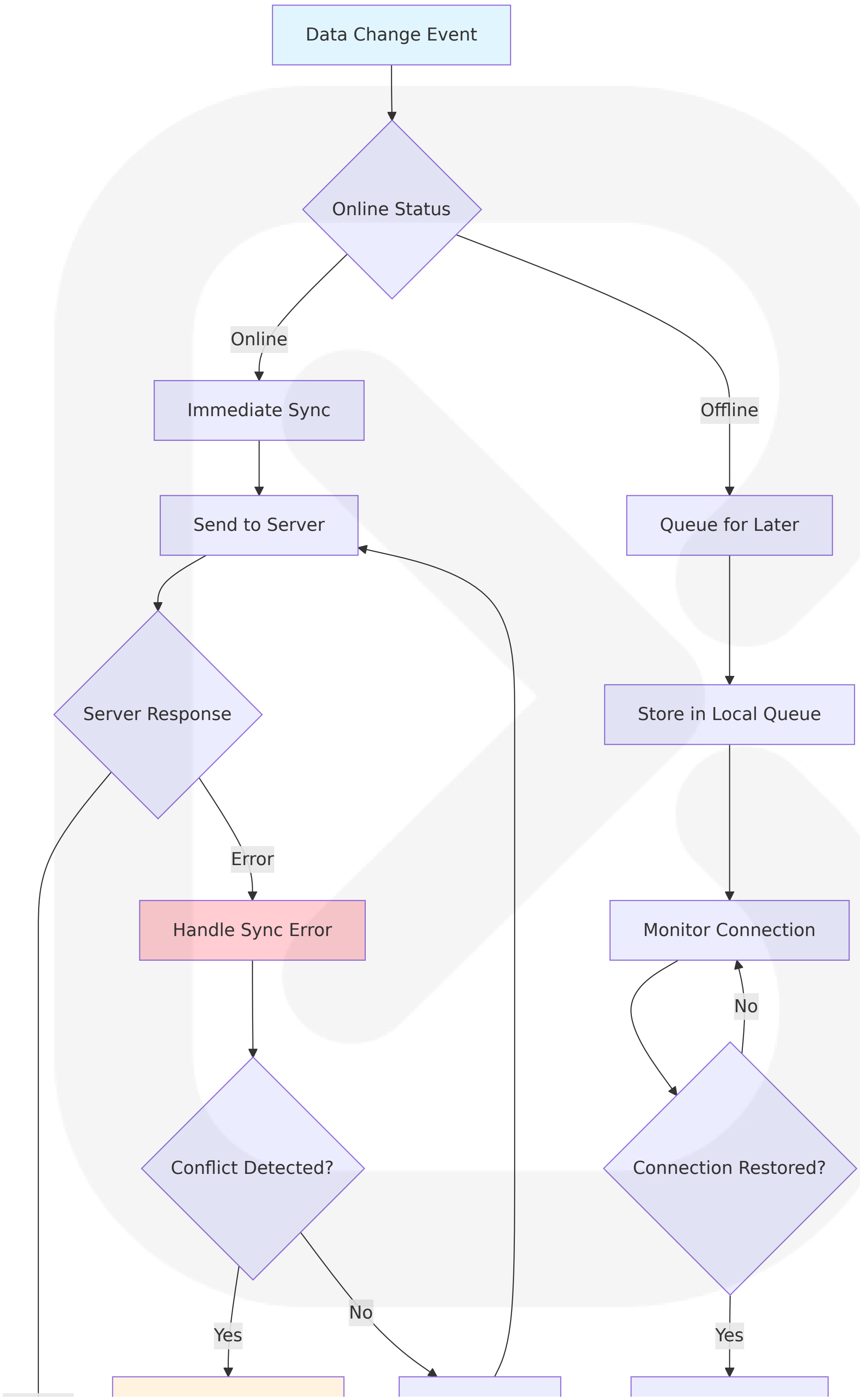
- Failure threshold: 5 consecutive failures
- Timeout period: 60 seconds
- Success threshold: 3 consecutive successes
- FastAPI exception handlers for centralized error management

4.3 STATE MANAGEMENT

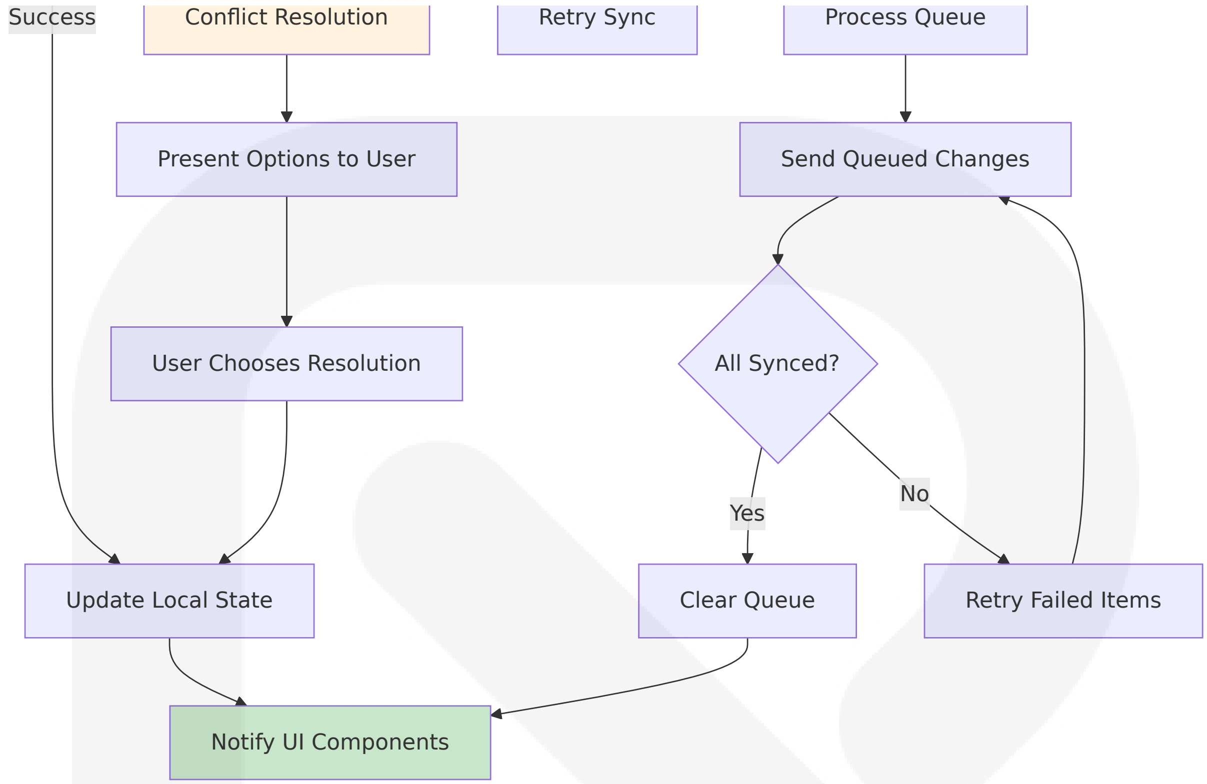
4.3.1 Application State Flow



4.3.2 Data Synchronization Flow



Data Change Event
Online Status
Online
Offline
Immediate Sync
Send to Server
Queue for Later
Server Response
Store in Local Queue
Error
Handle Sync Error
Monitor Connection
No
Conflict Detected?
Connection Restored?
Yes
No
Yes

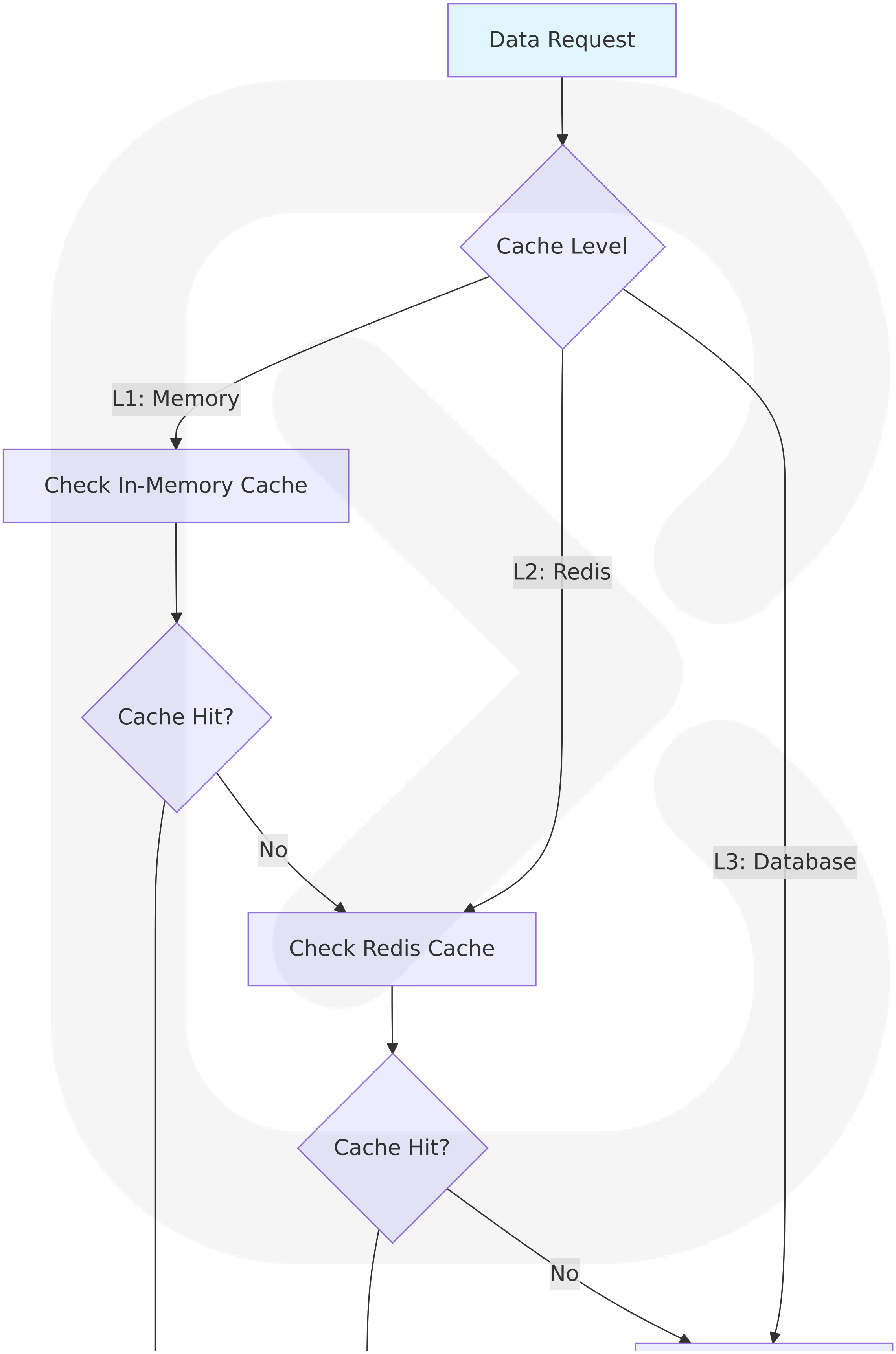


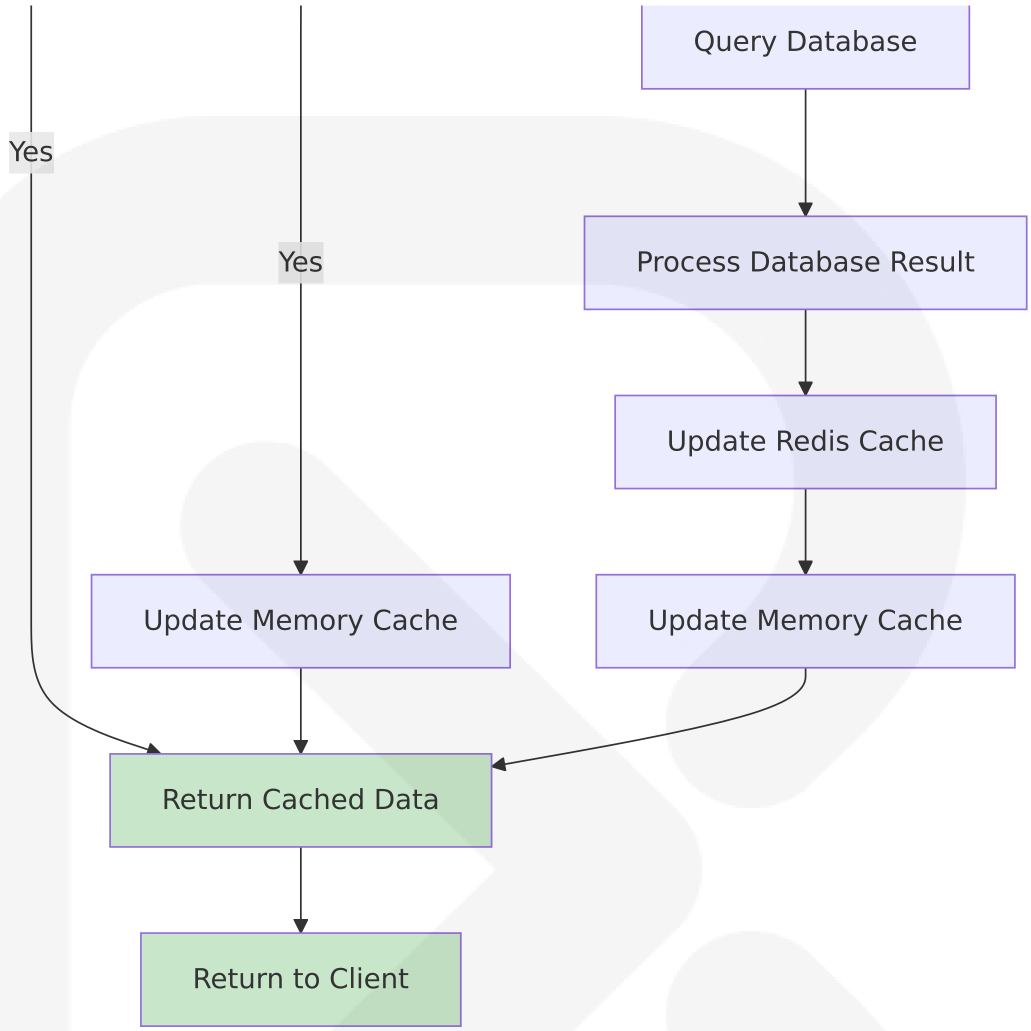
Synchronization Rules:

- Critical data (client info, property details): Immediate sync required
- Analytics data: Batch sync every 5 minutes
- Content drafts: Auto-save every 30 seconds
- Conflict resolution: Last-write-wins with user notification

4.4 PERFORMANCE OPTIMIZATION

4.4.1 Caching Strategy

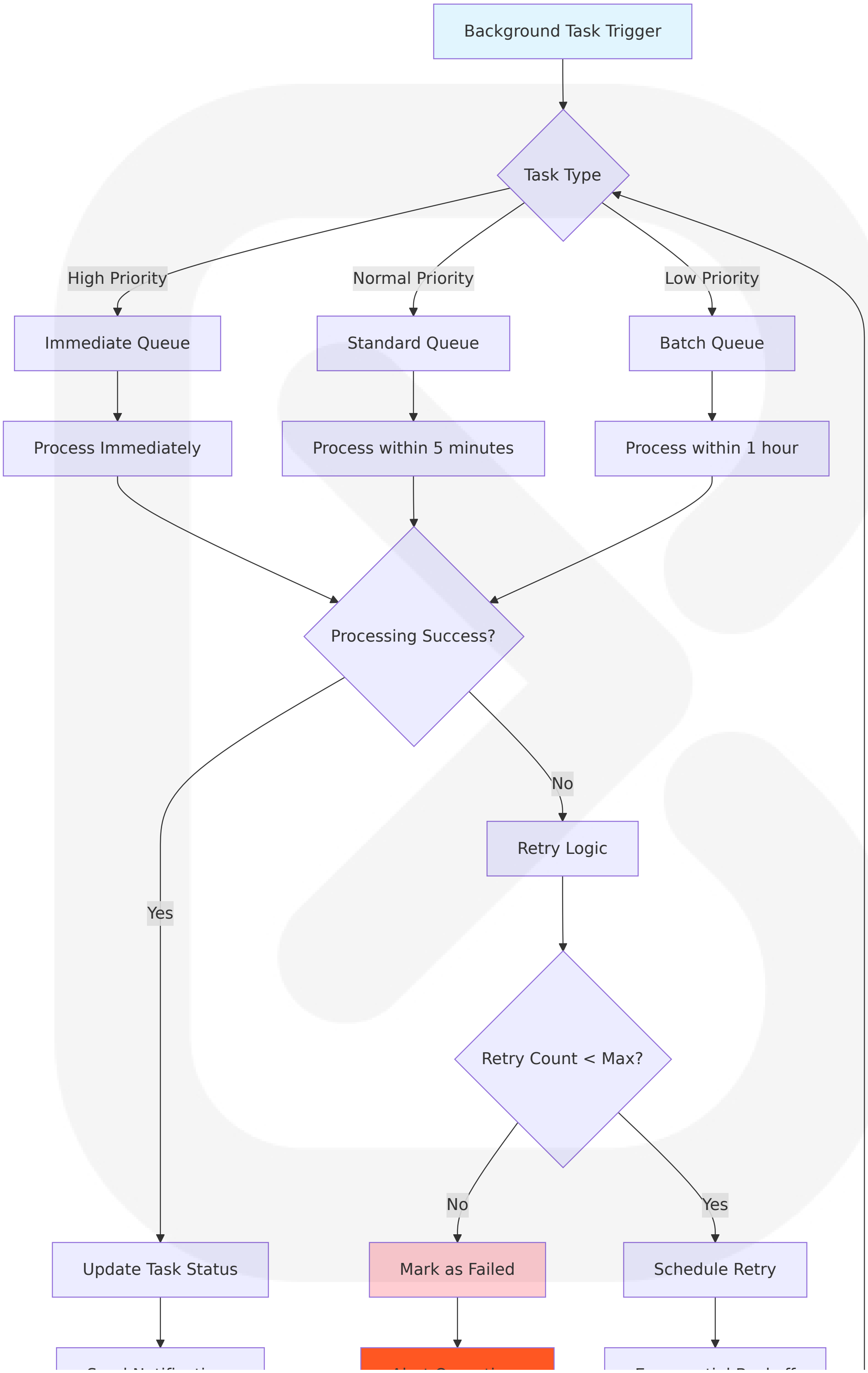

Data Request
Cache Level
L1: Memory
L2: Redis
L3: Database
Check In-Memory Cache
Cache Hit?
No
Check Redis Cache
Cache Hit?
No

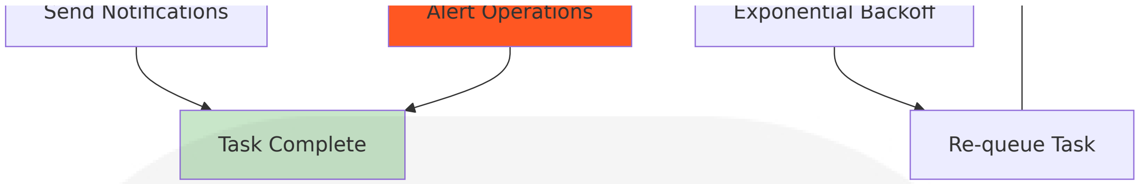


Cache Configuration:

- Memory cache: 100MB limit, LRU eviction
- Redis cache: 1GB limit, 1-hour TTL for content
- Database query cache: 15-minute TTL for analytics
- TypeScript integration for improved code quality and reduced runtime errors

4.4.2 Background Processing


Background Task Trigger
Task Type
High Priority
Normal Priority
Low Priority
Immediate Queue
Standard Queue
Batch Queue
Process Immediately
Process within 5 minutes
Process within 1 hour
Processing Success?
No
Yes
Retry Logic
Retry Count < Max?
No
Yes
Update Task Status
Mark as Failed
Schedule Retry



Background Task Categories:

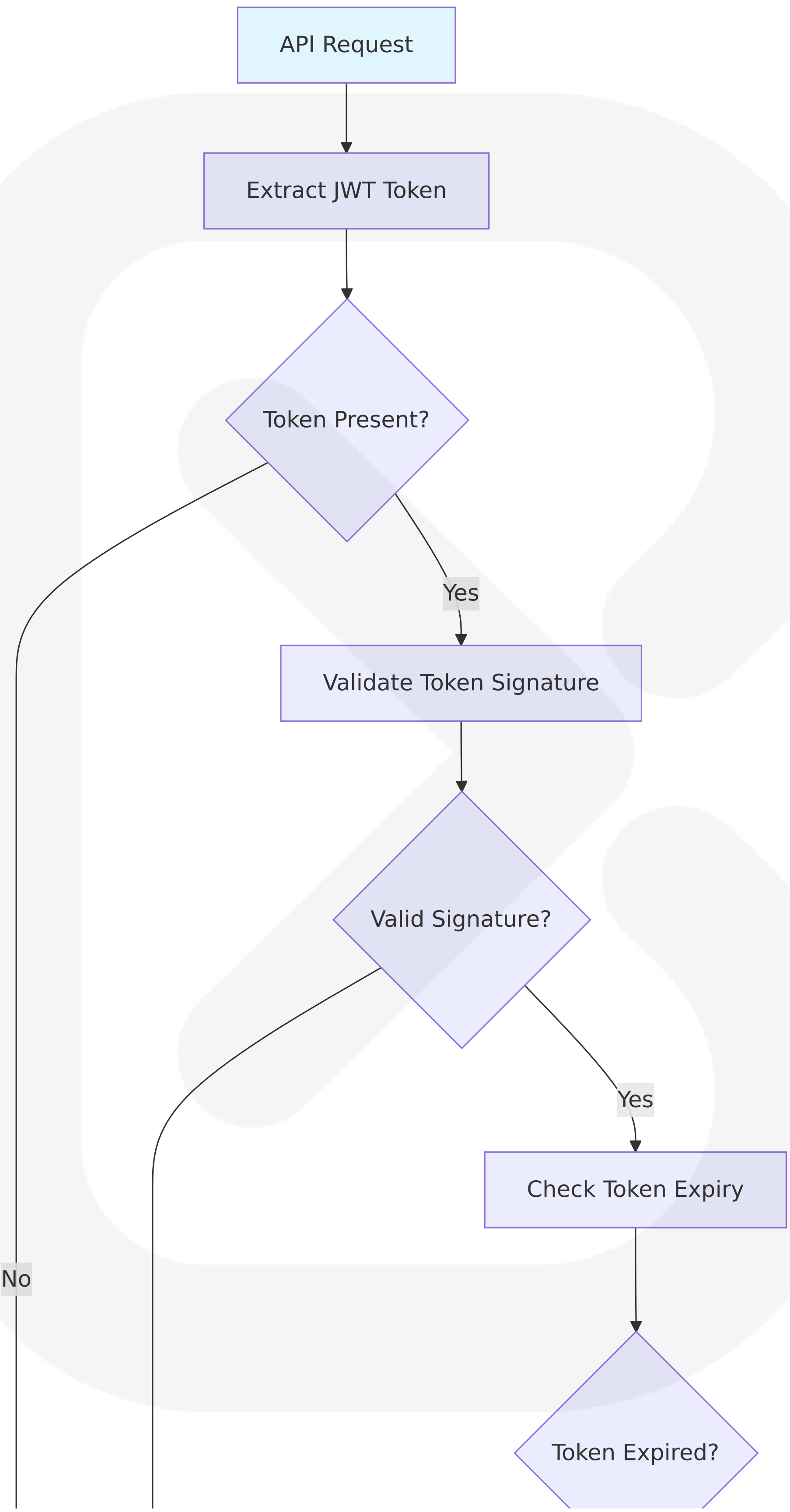
- High Priority: Client notifications, urgent follow-ups
- Normal Priority: Content generation, data synchronization
- Low Priority: Analytics processing, cleanup tasks

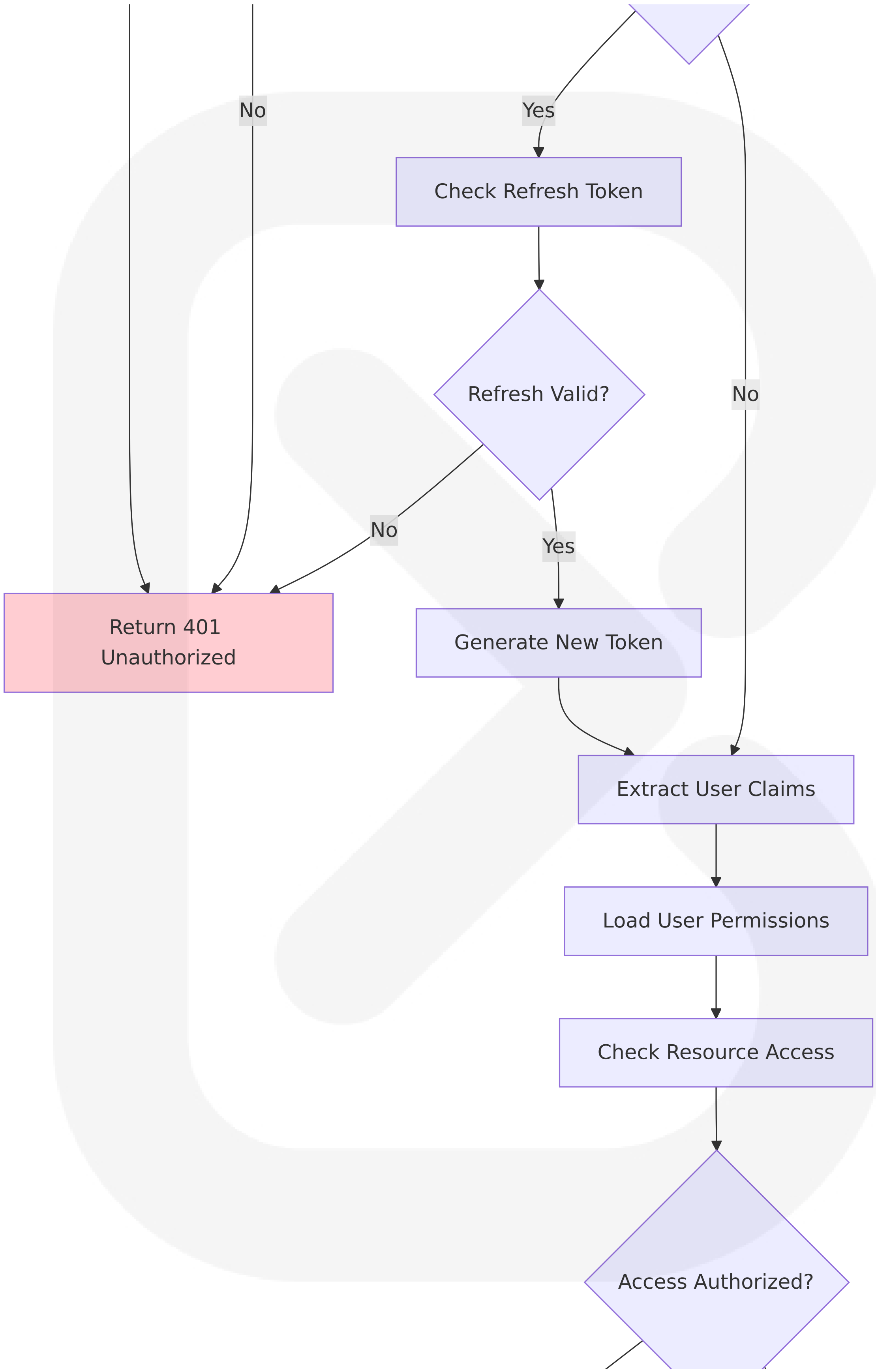
Processing Guarantees:

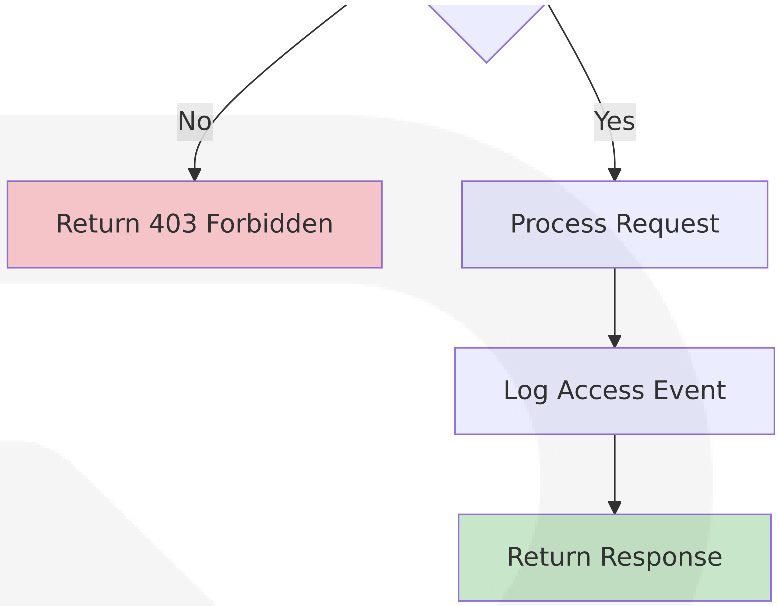
- At-least-once delivery for critical tasks
- Idempotent processing for all task types
- Dead letter queue for failed tasks after 5 retries

4.5 SECURITY AND COMPLIANCE

4.5.1 Authentication and Authorization Flow


API Request
Extract JWT Token
Token Present?
Yes
No
Validate Token Signature
Valid Signature?
Yes
Check Token Expiry
Token Expired?

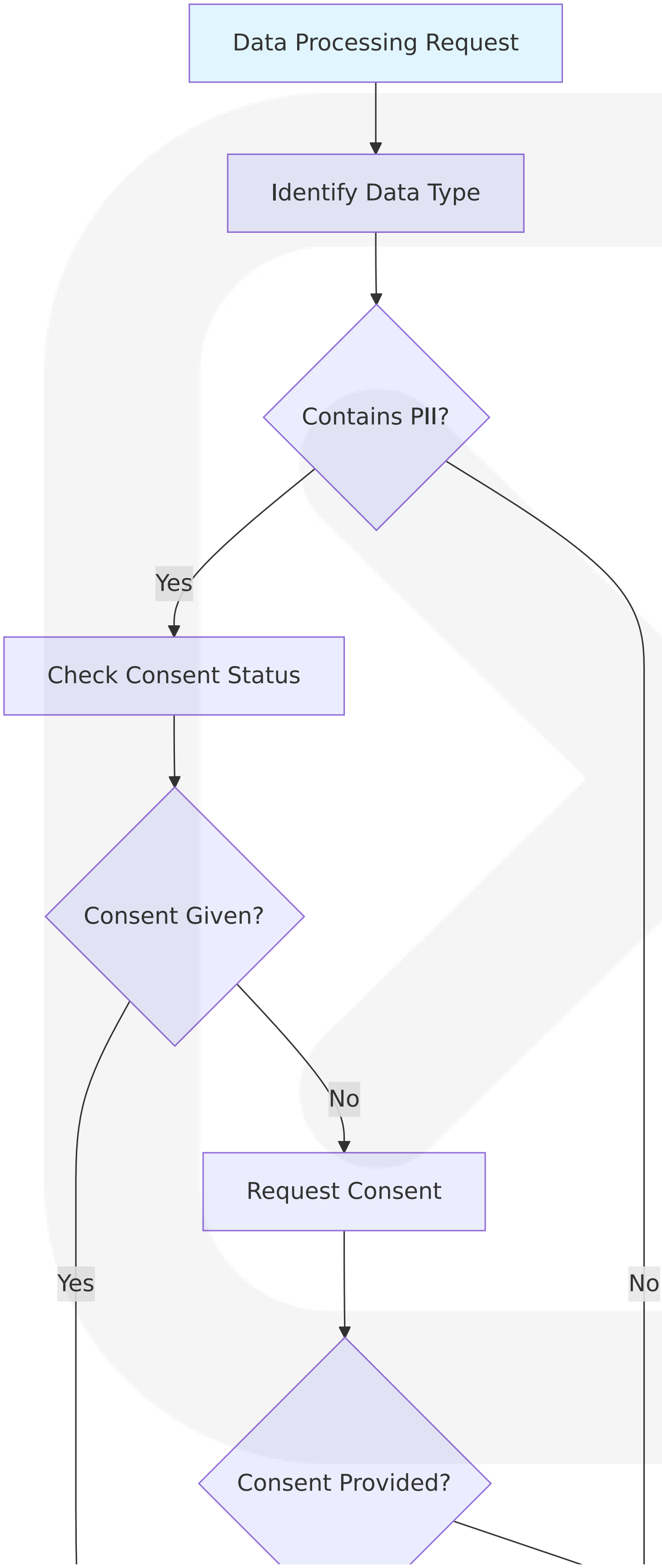

No
Yes
Check Refresh Token
Refresh Valid?
No
No
Yes
Return 401 Unauthorized
Generate New Token
Extract User Claims
Load User Permissions
Check Resource Access
Access Authorized?

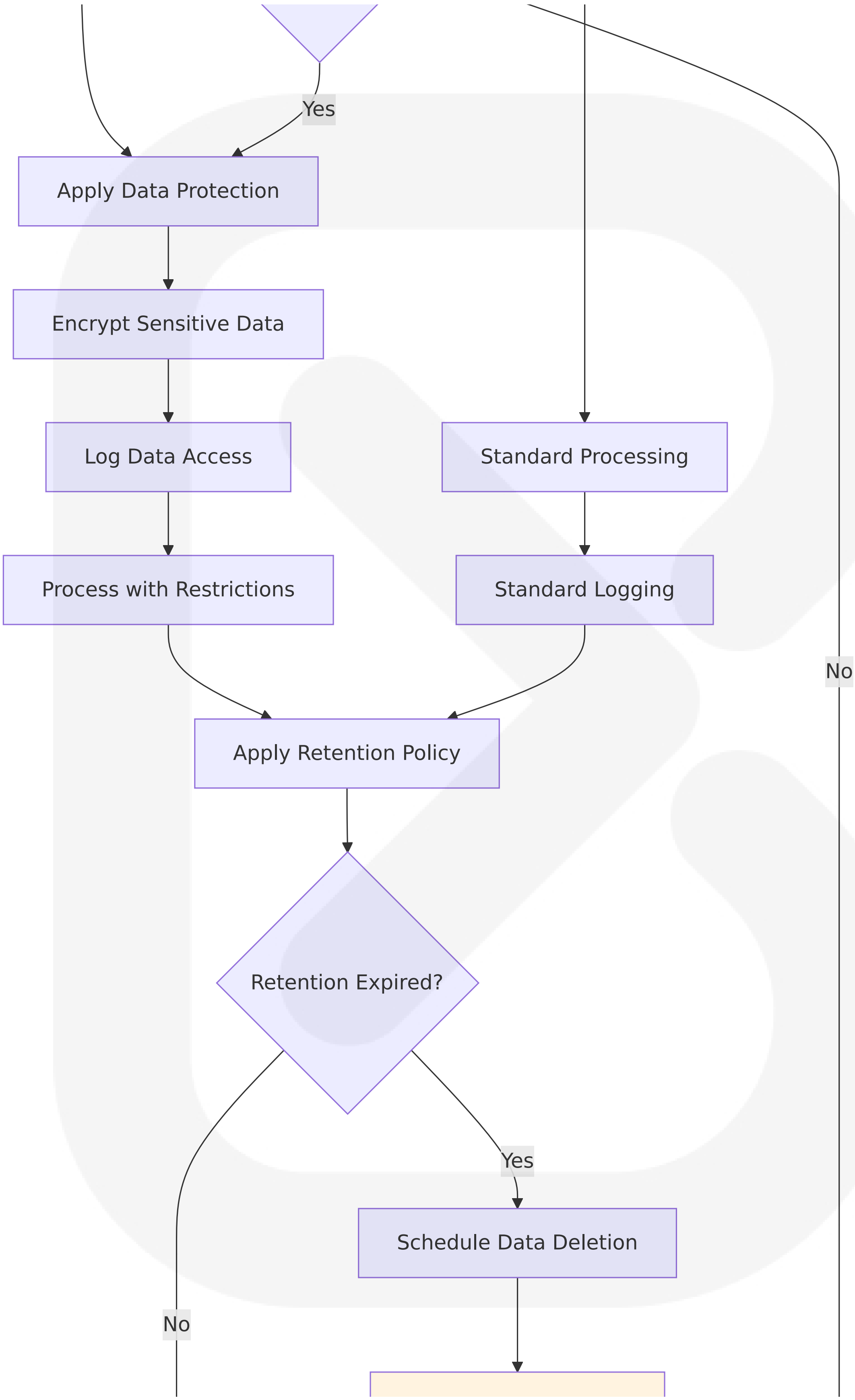


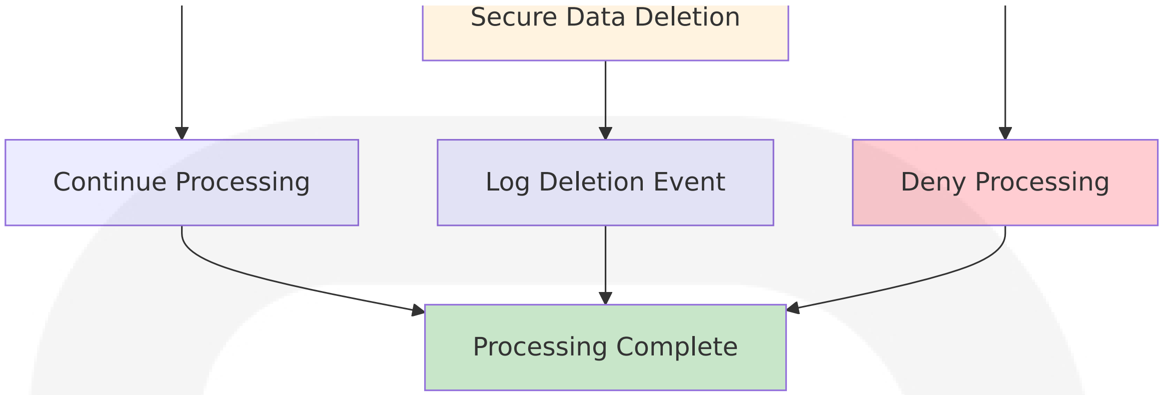
Security Specifications:

- JWT token expiry: 24 hours
- Refresh token expiry: 30 days
- Security measures including input sanitization, encrypted storage, and HTTPS communication
- Rate limiting: 1000 requests per hour per user

4.5.2 Data Privacy and Compliance


Data Processing Request
Identify Data Type
Contains PII?
Yes
Check Consent Status
Consent Given?
No
Request Consent
Yes
No
Consent Provided?


Yes
Apply Data Protection
Encrypt Sensitive Data
Log Data Access
Standard Processing
Process with Restrictions
Standard Logging
No
Apply Retention Policy
Retention Expired?
Yes
Schedule Data Deletion
No



Compliance Requirements:

- GDPR compliance for EU users
- Data retention: 7 years for business records
- Right to deletion: 30-day processing window
- Data portability: JSON export format
- Audit trail: All data access logged

This comprehensive process flowchart section provides detailed workflows for all major system components, ensuring proper error handling, state management, and security compliance while maintaining optimal performance through caching and background processing strategies.

5. SYSTEM ARCHITECTURE

5.1 HIGH-LEVEL ARCHITECTURE

5.1.1 System Overview

PropertyPro AI employs a **Clean Architecture** pattern with **Domain-Driven Design** principles, implementing a mobile-first approach using React Native with TypeScript for the frontend and FastAPI with Python 3.11+ for the backend. The architecture follows the **Hexagonal**

Architecture (Ports and Adapters) pattern to ensure framework independence and high testability.

The system is designed as a **distributed microservices architecture** with clear separation of concerns across four primary layers: Presentation (React Native mobile app), Application (business logic orchestration), Domain (core business rules), and Infrastructure (external services and data persistence). This layered approach ensures that business logic remains independent of external frameworks and can adapt to changing requirements without affecting core functionality.

The architecture emphasizes **event-driven communication** patterns between components, with React Native 0.71+ providing built-in TypeScript support and enhanced performance through the New Architecture with JSI (JavaScript Interface) for direct native communication. The backend leverages FastAPI's built-in dependency injection mechanism and Pydantic for data validation and serialization, creating a robust foundation for AI-powered real estate operations.

5.1.2 Core Components Table

Component Name	Primary Responsibility	Key Dependencies	Integration Points
Mobile Frontend	User interface and experience	React Native 0.71+, TypeScript 5.0+	Backend API, AI Services
API Gateway	Request routing and authentication	FastAPI 0.115+, JWT tokens	Mobile Frontend, Core Services
AI Service Layer	Content generation and analysis	GPT-4.1 API with 1M token context	Property Service, Client Service
Property Management Service	Property CRUD and market analysis	PostgreSQL 15, SQLAlchemy 2.0	AI Service, File Storage

5.1.3 Data Flow Description

The primary data flow follows a **request-response pattern** with **asynchronous processing** for AI-intensive operations. User interactions in the React Native frontend trigger API calls to the FastAPI backend, which orchestrates business logic through domain services. The GPT-4.1 integration supports up to 1 million tokens of context with a knowledge cutoff of June 2024, enabling comprehensive property analysis and content generation.

Data transformation occurs at three key points: input validation using Pydantic models at the API boundary, domain entity mapping within business services, and response serialization for mobile consumption. The system implements **eventual consistency** for analytics data while maintaining **ACID compliance** for transactional operations like property creation and client management.

Caching strategies are implemented at multiple levels: React Native component-level caching for UI state, API response caching using in-memory storage, and database query result caching for frequently accessed property and market data. This multi-tier approach ensures optimal performance while maintaining data freshness for critical business operations.

5.1.4 External Integration Points

System Name	Integration Type	Data Exchange Pattern	Protocol/Format
OpenAI GPT-4.1 API	REST API	Request/Response with streaming	HTTPS/JSON
PostgreSQL Database	Direct Connection	Async ORM queries	TCP/SQL
File Storage System	Local filesystem	File upload/retrieval	HTTP multipart

System Name	Integration Type	Data Exchange Pattern	Protocol/Format
Email Service	SMTP Integration	Async message queuing	SMTP/TLS

5.2 COMPONENT DETAILS

5.2.1 Mobile Frontend Architecture

Purpose and Responsibilities:
The React Native frontend serves as the primary user interface, implementing a **component-based architecture** with TypeScript for type safety. React Native 0.71+ provides first-class TypeScript support with bundled type definitions, eliminating external dependencies and ensuring robust development experience.

Technologies and Frameworks:

- **React Native 0.71+** with New Architecture support for enhanced performance
- **TypeScript 5.0+** for comprehensive type safety
- **Zustand** for lightweight state management
- **React Navigation 6.0+** for type-safe navigation
- **Axios** for HTTP client communication

Key Interfaces and APIs:
The frontend exposes a clean API surface through custom hooks and service layers, abstracting complex state management and API communication. The component architecture follows **atomic design principles** with reusable UI components, screen-level containers, and service integration layers.

Data Persistence Requirements:
Local data persistence utilizes React Native's AsyncStorage for user

preferences and offline capability. Critical data synchronization ensures seamless operation during network interruptions, with automatic sync when connectivity is restored.

Scaling Considerations:
The component architecture supports horizontal scaling through **code splitting** and **lazy loading** of feature modules. Performance optimization includes **memoization** of expensive computations and **virtualized lists** for large datasets.

5.2.2 Backend API Services

Purpose and Responsibilities:
The FastAPI backend implements Clean Architecture principles with FastAPI capabilities for building testable, scalable and maintainable applications. The service layer orchestrates business logic while maintaining framework independence through dependency inversion.

Technologies and Frameworks:

- **FastAPI 0.115+** for high-performance API development
- **Python 3.11+** with enhanced asyncio support
- **Pydantic 2.0+** for data validation and serialization
- **SQLAlchemy 2.0** with async support for database operations
- **Uvicorn** as the ASGI server

Key Interfaces and APIs:
RESTful API endpoints follow OpenAPI 3.0 specifications with automatic documentation generation. The API design emphasizes **resource-oriented** URLs with consistent HTTP verb usage and standardized response formats.

Data Persistence Requirements:
PostgreSQL 15 serves as the primary database with **connection pooling**

and **transaction management**. Database migrations are handled through Alembic with version control and rollback capabilities.

Scaling Considerations:
The service architecture supports **horizontal scaling** through stateless design and **database connection pooling**. Async request handling enables high concurrency with efficient resource utilization.

5.2.3 AI Integration Layer

Purpose and Responsibilities:
The AI service layer manages integration with OpenAI's GPT-4.1 API, which outperforms previous models with major gains in coding and instruction following. This component handles content generation, market analysis, and intelligent task automation.

Technologies and Frameworks:

- **OpenAI Python SDK 1.0+** for API integration
- **LangChain 0.1+** for AI workflow orchestration
- **Tiktoken 0.5+** for token management and optimization
- **Custom retry logic** with exponential backoff

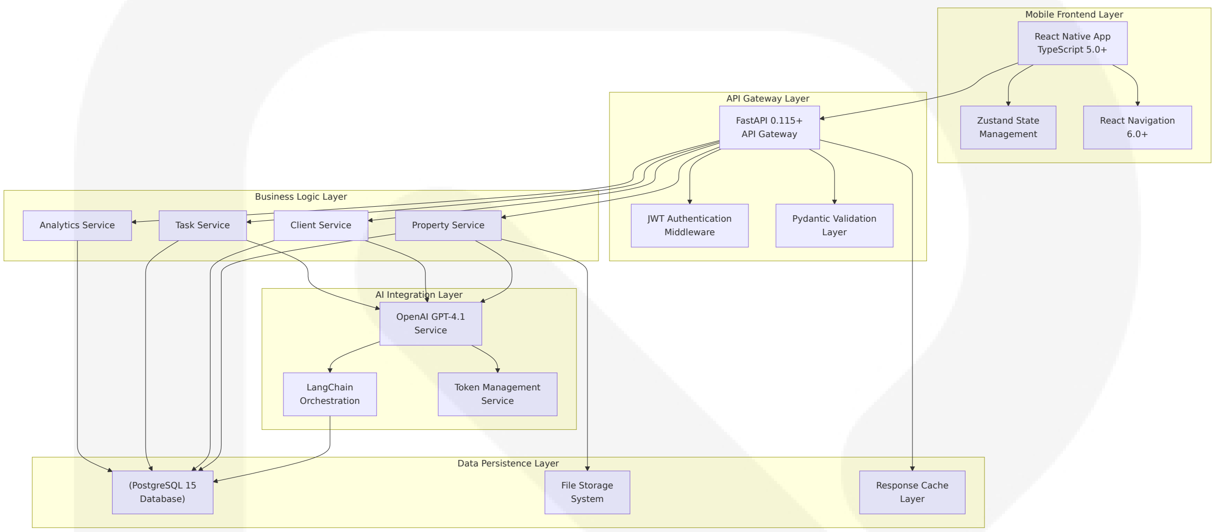
Key Interfaces and APIs:
The AI service exposes domain-specific interfaces for property description generation, market analysis, and client communication. The 1 million token context window enables comprehensive analysis of complex real estate scenarios.

Data Persistence Requirements:
AI-generated content is cached with configurable TTL (Time To Live) values. Conversation history and model responses are stored for audit trails and continuous improvement.

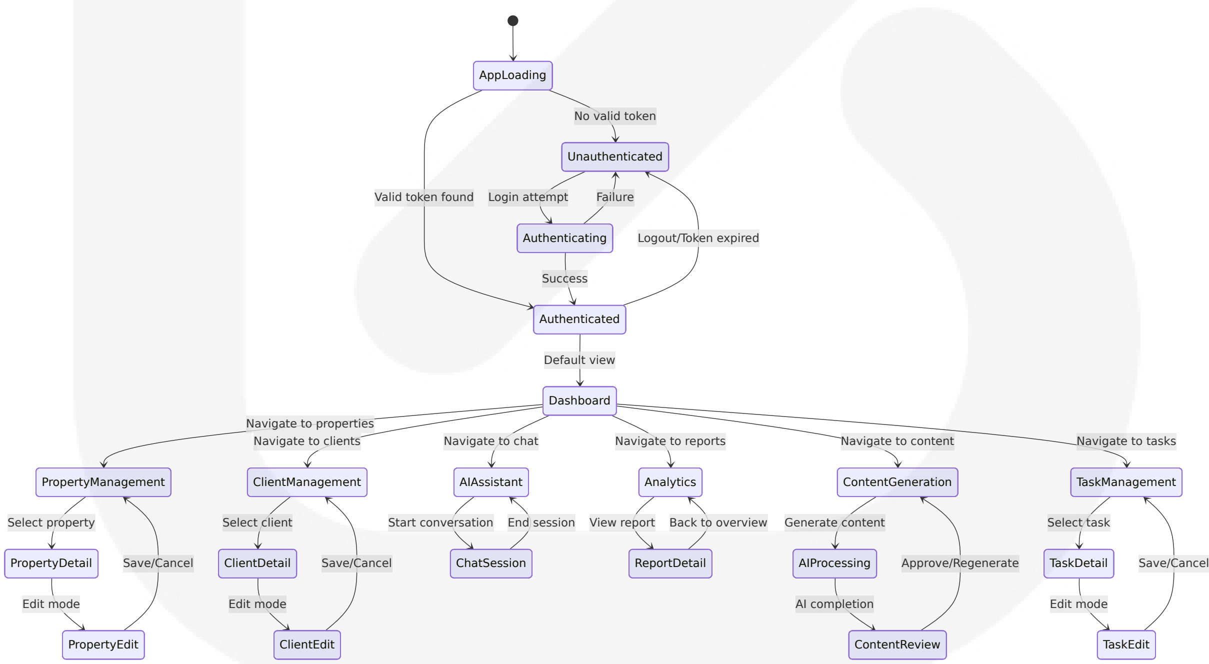
Scaling Considerations:
Rate limiting and request queuing manage API usage costs while ensuring

service availability. Circuit breaker patterns prevent cascade failures during API outages.

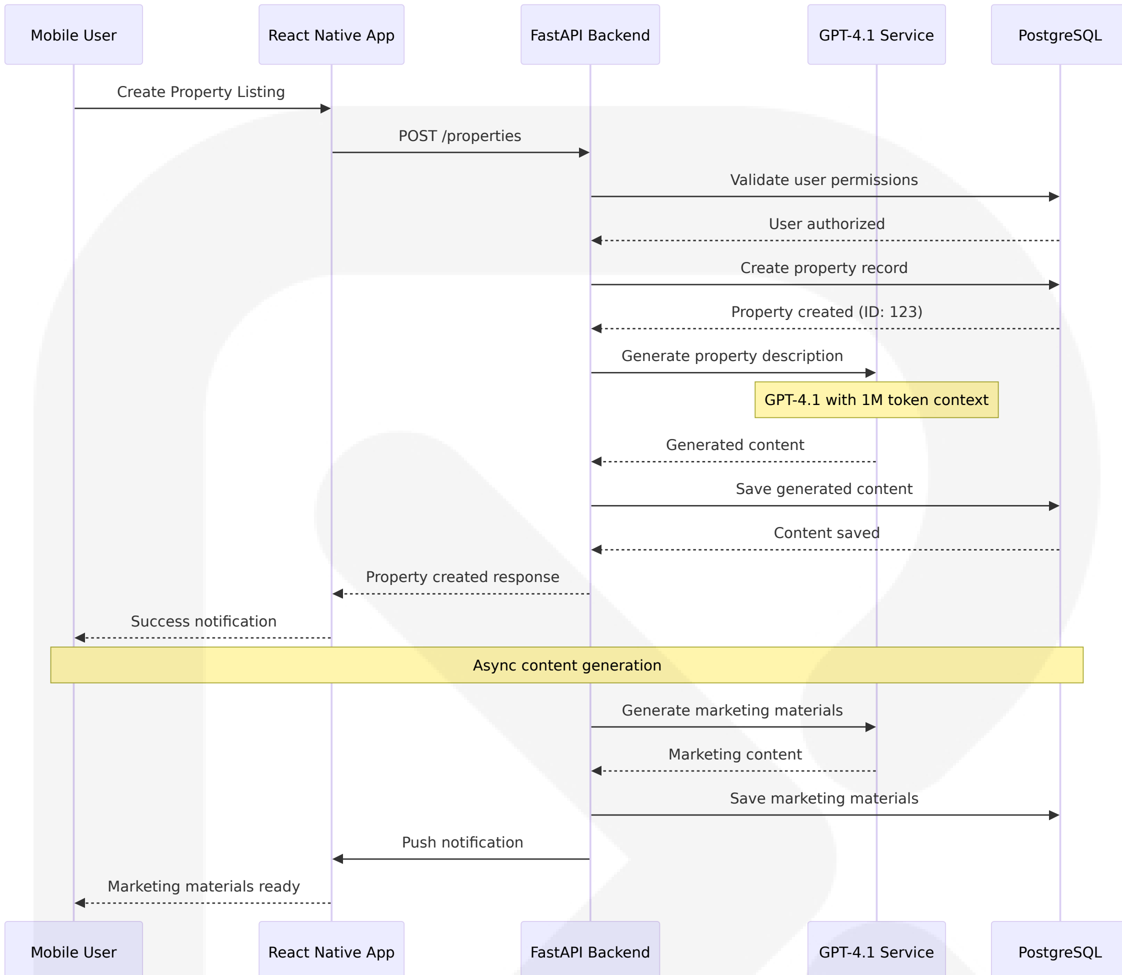
5.2.4 Component Interaction Diagrams



5.2.5 State Transition Diagrams



5.2.6 Sequence Diagrams for Key Flows



5.3 TECHNICAL DECISIONS

5.3.1 Architecture Style Decisions and Tradeoffs

Clean Architecture with Hexagonal Pattern Selection:

The decision to implement Clean Architecture provides framework-agnostic and storage-agnostic flexibility, making the application independent of external systems and highly testable. This architectural choice enables the system to adapt to changing requirements without affecting core business logic.

Tradeoffs:

- **Benefits:** High testability, framework independence, clear separation of concerns
- **Costs:** Initial development complexity, additional abstraction layers
- **Mitigation:** FastAPI's built-in dependency injection and Pydantic integration reduces boilerplate code

Mobile-First Architecture Decision:

React Native was selected over native development to achieve cross-platform compatibility while maintaining native performance. React Native 0.71+ provides first-class TypeScript support with bundled type definitions, eliminating configuration complexity.

Tradeoffs:

- **Benefits:** Single codebase for iOS and Android, faster development cycles
- **Costs:** Platform-specific optimizations may be limited
- **Mitigation:** New Architecture with JSI enables direct native communication for performance-critical operations

5.3.2 Communication Pattern Choices

RESTful API with Async Processing:

The system implements RESTful APIs for synchronous operations with asynchronous processing for AI-intensive tasks. This hybrid approach balances immediate user feedback with resource-intensive operations.

Pattern	Use Case	Justification
Synchronous REST	CRUD operations	Immediate feedback required
Asynchronous Processing	AI content generation	GPT-4.1's 1M token context requires processing time

Pattern	Use Case	Justification
WebSocket (Future)	Real-time chat	Low-latency AI assistant interactions
Event-Driven	Background tasks	Decoupled processing for scalability

5.3.3 Data Storage Solution Rationale

PostgreSQL 15 Selection:

PostgreSQL was chosen as the primary database for its robust ACID compliance, JSON support, and excellent performance with complex queries. The selection supports both relational data integrity and flexible document storage for AI-generated content.

Storage Strategy Justification:

- **Structured Data:** Traditional relational tables for properties, clients, and users
- **Semi-Structured Data:** JSONB columns for AI responses and flexible property attributes
- **File Storage:** Local filesystem for property images with future cloud migration path
- **Caching:** In-memory caching for frequently accessed data

5.3.4 Caching Strategy Justification

Multi-Tier Caching Architecture:

The caching strategy implements multiple layers to optimize performance while maintaining data consistency:

Cache Layer	Technology	TTL	Purpose
Application Cache	Python dictionaries	15 minutes	API response caching

Cache Layer	Technology	TTL	Purpose
Database Cache	SQLAlchemy query cache	5 minutes	Query result optimization
Client Cache	React Native state	Session-based	UI state persistence
AI Response Cache	PostgreSQL JSONB	1 hour	GPT-4.1 response optimization

5.3.5 Security Mechanism Selection

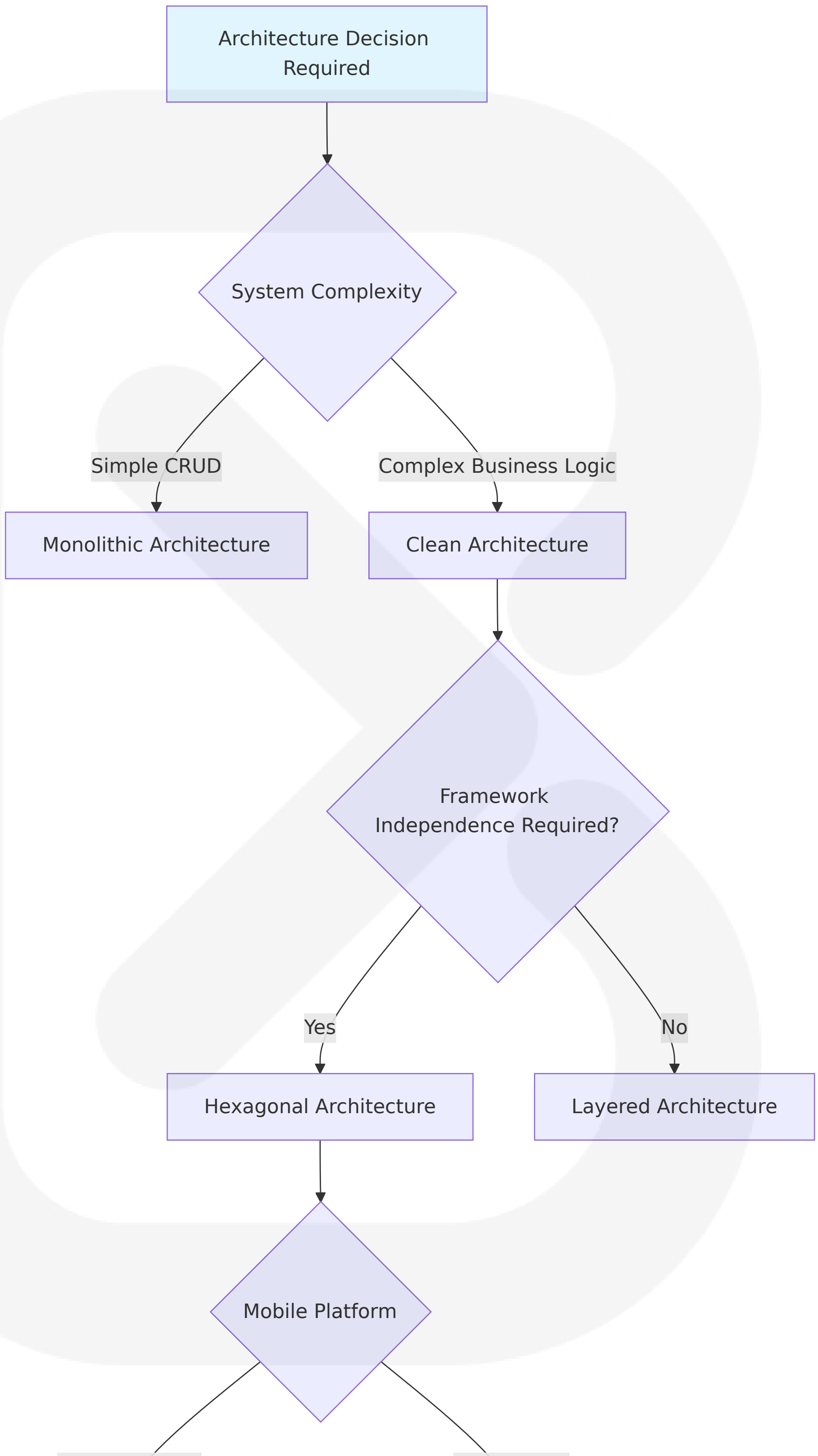
JWT-Based Authentication:

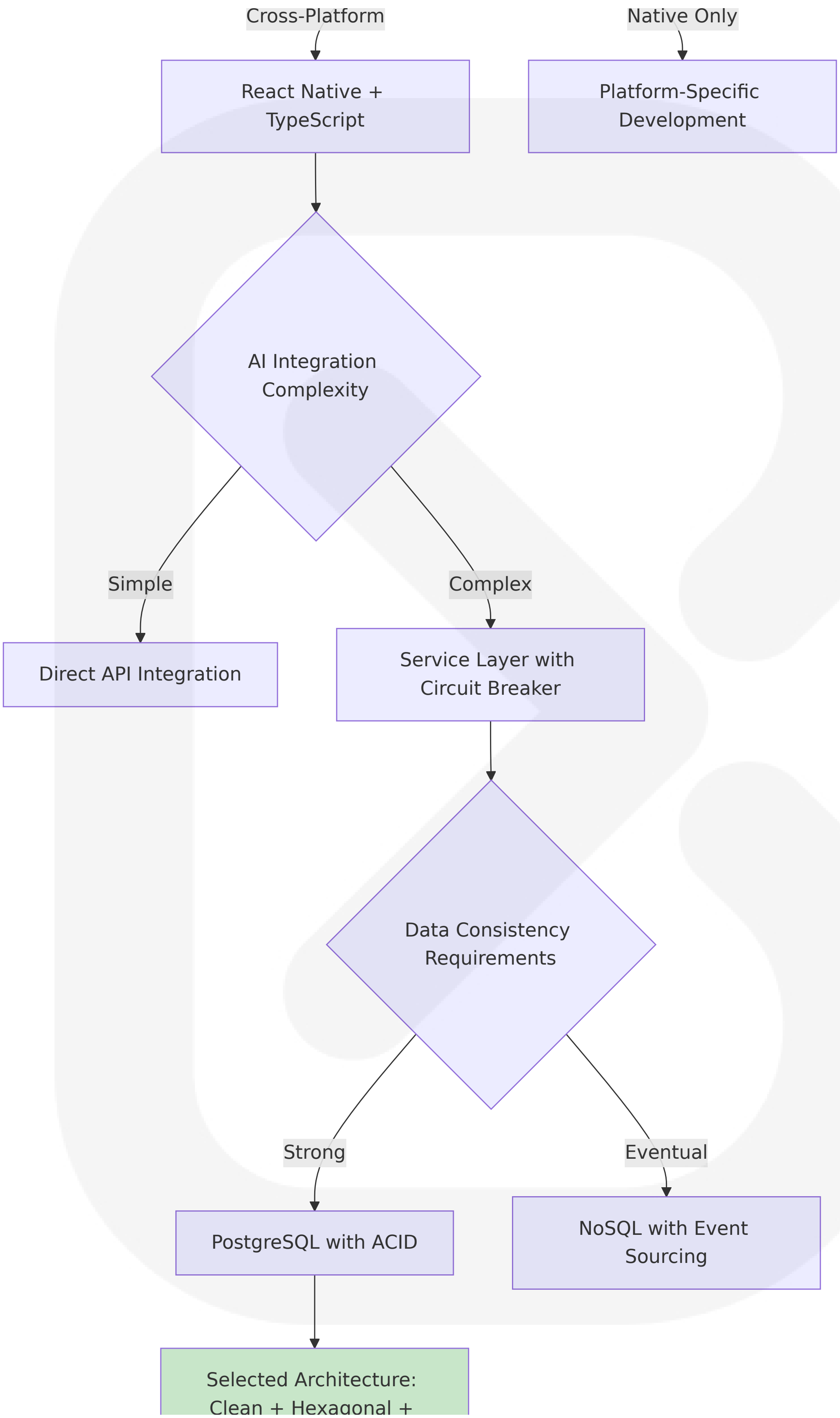
JWT tokens provide stateless authentication suitable for mobile applications with offline capabilities. The implementation includes refresh token rotation and secure storage mechanisms.

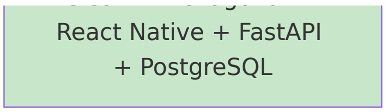
Security Architecture:

- **Authentication:** JWT tokens with bcrypt password hashing
- **Authorization:** Role-based access control (RBAC)
- **Data Protection:** AES-256 encryption at rest, TLS 1.3 in transit
- **API Security:** Rate limiting, input validation, CORS configuration

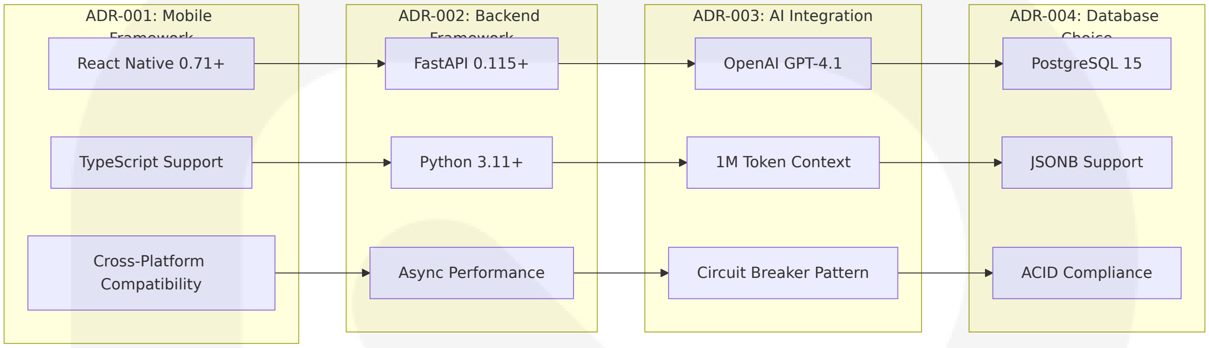
5.3.6 Decision Tree Diagrams


Architecture Decision Required
System Complexity
Simple CRUD
Complex Business Logic
Monolithic Architecture
Clean Architecture
Framework Independence Required?
Yes
No
Hexagonal Architecture
Layered Architecture
Mobile Platform


Cross-Platform
Native Only
React Native + TypeScript
Platform-Specific Development
AI Integration Complexity
Simple
Complex
Direct API Integration
Service Layer with Circuit Breaker
Data Consistency Requirements
Strong
Eventual
PostgreSQL with ACID
NoSQL with Event Sourcing
Selected Architecture: Clean + Hexagonal +



5.3.7 Architecture Decision Records (ADRs)



5.4 CROSS-CUTTING CONCERNS

5.4.1 Monitoring and Observability Approach

Comprehensive Monitoring Strategy:

The system implements a three-tier monitoring approach: application-level metrics, infrastructure monitoring, and business intelligence tracking. This strategy provides visibility into system health, performance bottlenecks, and user behavior patterns.

Monitoring Components:

- **Application Metrics:** API response times, error rates, throughput measurements
- **Infrastructure Metrics:** Database connection pools, memory usage, CPU utilization
- **Business Metrics:** User engagement, AI API usage costs, content generation success rates

- **Health Checks:** Automated endpoint monitoring with alerting capabilities

Observability Tools:

- **Logging:** Structured logging with correlation IDs for request tracing
- **Metrics Collection:** Custom FastAPI middleware for performance monitoring
- **Health Endpoints:** `/health` endpoints for all services with dependency checks
- **Real-time Dashboards:** Performance metrics visualization and alerting

5.4.2 Logging and Tracing Strategy

Structured Logging Implementation:

The logging strategy employs structured JSON logging with correlation IDs to trace requests across service boundaries. This approach enables efficient log aggregation and analysis for debugging and performance optimization.

Logging Levels and Scope:

- **DEBUG:** Detailed execution flow for development environments
- **INFO:** Business logic execution and successful operations
- **WARNING:** Recoverable errors and performance degradation
- **ERROR:** System failures requiring immediate attention
- **CRITICAL:** Service unavailability and data integrity issues

Tracing Architecture:

- **Request Correlation:** Unique request IDs propagated across all service calls
- **User Context:** User identification and session tracking for audit trails

- **Performance Tracing:** Execution time measurement for critical operations
- **AI API Tracing:** GPT-4.1 API request/response logging with token usage tracking

5.4.3 Error Handling Patterns

Hierarchical Error Handling:

The system implements a comprehensive error handling strategy with different approaches for various error types: validation errors, business logic errors, external service failures, and system errors.

Error Category	Handling Strategy	User Impact	Recovery Action
Validation Errors	Immediate feedback	Form validation messages	User correction required
Business Logic Errors	Graceful degradation	Alternative workflows	Automatic retry or manual intervention
External Service Errors	Circuit breaker pattern	Fallback functionality	Service restoration monitoring
System Errors	Fail-safe mechanisms	Error reporting	Automatic recovery or manual intervention

5.4.4 Authentication and Authorization Framework

JWT-Based Security Architecture:

The authentication system implements JWT tokens with refresh token rotation, providing secure stateless authentication suitable for mobile applications. The authorization framework uses role-based access control (RBAC) with fine-grained permissions.

Security Components:

- **Authentication:** JWT access tokens (24-hour expiry) with refresh tokens (30-day expiry)
- **Authorization:** Role-based permissions with resource-level access control
- **Password Security:** bcrypt hashing with salt rounds for secure password storage
- **Session Management:** Automatic token refresh with secure storage mechanisms

Security Middleware:

- **Request Authentication:** JWT token validation on all protected endpoints
- **Rate Limiting:** API usage limits to prevent abuse and control costs
- **Input Validation:** Pydantic model validation for all API inputs
- **CORS Configuration:** Cross-origin resource sharing for web client support

5.4.5 Performance Requirements and SLAs

Performance Targets:

The system defines specific performance targets for different operation categories, ensuring optimal user experience while managing resource costs effectively.

Operation Category	Response Time Target	Throughput Target	Availability SLA
User Authentication	< 2 seconds	1,000 requests/minute	99.9%
Property CRUD Operations	< 3 seconds	500 operations/minute	99.5%

Operation Category	Response Time Target	Throughput Target	Availability SLA
AI Content Generation	< 5 seconds	100 requests/minute	99.0%
Real-time Chat	< 3 seconds	200 messages/minute	99.5%

Performance Optimization Strategies:

- **Database Optimization:** Connection pooling, query optimization, and indexing strategies
- **Caching Implementation:** Multi-tier caching with appropriate TTL values
- **API Rate Management:** GPT-4.1 API usage optimization with intelligent queuing
- **Mobile Performance:** Component memoization and lazy loading for optimal user experience

5.4.6 Disaster Recovery Procedures

Business Continuity Planning:

The disaster recovery strategy focuses on data protection, service availability, and rapid recovery capabilities. The approach balances recovery time objectives (RTO) with recovery point objectives (RPO) based on business criticality.

Recovery Procedures:

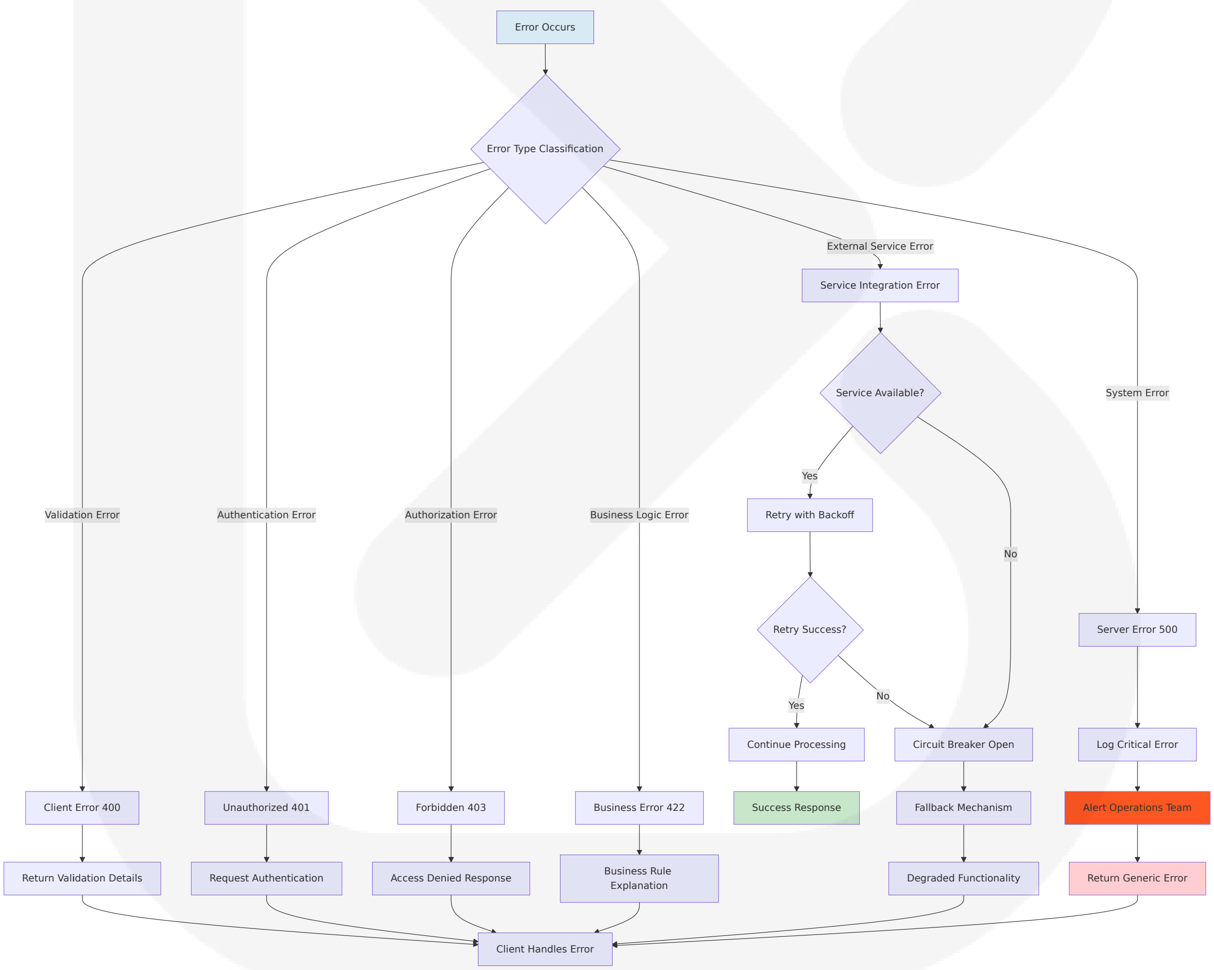
- **Data Backup:** Automated daily database backups with point-in-time recovery capability
- **Service Redundancy:** Stateless service design enabling rapid horizontal scaling
- **Graceful Degradation:** Core functionality availability during partial system failures

- **External Service Fallbacks:** Alternative workflows when AI services are unavailable

Recovery Time Objectives:

- **Critical Services:** < 4 hours recovery time
- **Data Recovery:** < 1 hour for recent data (RPO: 15 minutes)
- **Full System Restoration:** < 24 hours for complete service recovery
- **Communication Plan:** Automated user notifications during service disruptions

5.4.7 Error Handling Flows



This comprehensive system architecture provides a robust foundation for PropertyPro AI, leveraging modern technologies and proven architectural patterns to deliver a scalable, maintainable, and high-performance real estate assistant platform. The architecture emphasizes clean separation of concerns, comprehensive error handling, and optimal performance while maintaining the flexibility to adapt to evolving business requirements.

6. SYSTEM COMPONENTS DESIGN

6.1 MOBILE APPLICATION COMPONENTS

6.1.1 React Native Component Architecture

React Native 0.71+ includes TypeScript by default with built-in type declarations, eliminating the need for @types/react-native package. The mobile application follows a component-based architecture with clear separation of concerns and type safety throughout the application.

Core Component Structure

Component Category	Purpose	TypeScript Integration	Performance Considerations
Screen Components	Top-level navigation containers	Strict typing with navigation props	Lazy loading and code splitting
Feature Components	Business logic containers	Domain-specific type definitions	Memoization for expensive operations
UI Components	Reusable interface elements	Generic type parameters	Virtual scrolling for large lists

Component Category	Purpose	TypeScript Integration	Performance Considerations
Service Components	External API integration	API response type validation	Request caching and error boundaries

Component Design Patterns

Design patterns help developers write maintainable, scalable, and efficient code by providing established structures for organizing components, logic, and behavior. In React Native, using design patterns ensures a clean, modular approach to building cross-platform mobile applications, especially when combined with TypeScript for type safety.

Container/Presenter Pattern Implementation:

```
// Container Component - Handles business logic
interface PropertyListContainerProps {
  userId: string;
  filters: PropertyFilters;
}

const PropertyListContainer: React.FC<PropertyListContainerProps> = ({
  userId,
  filters
}) => {
  const [properties, setProperties] = useState<Property[]>([]);
  const [loading, setLoading] = useState<boolean>(false);
  const [error, setError] = useState<string | null>(null);

  // Business logic and state management
  const fetchProperties = useCallback(async () => {
    setLoading(true);
    try {
      const data = await propertyService.getProperties(userId, filters);
      setProperties(data);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
```

```
    }
  }, [userId, filters]);

  return (
    <PropertyListPresenter
      properties={properties}
      loading={loading}
      error={error}
      onRefresh={fetchProperties}
    />
  );
};

// Presenter Component - Pure UI rendering
interface PropertyListPresenterProps {
  properties: Property[];
  loading: boolean;
  error: string | null;
  onRefresh: () => void;
}

const PropertyListPresenter: React.FC<PropertyListPresenterProps> = ({
  properties,
  loading,
  error,
  onRefresh
}) => {
  if (loading) return <LoadingSpinner />;
  if (error) return <ErrorMessage message={error} onRetry={onRefresh} />

  return (
    <FlatList
      data={properties}
      renderItem={({ item }) => <PropertyCard property={item} />}
      keyExtractor={(item) => item.id}
      onRefresh={onRefresh}
      refreshing={loading}
    />
  );
};
```

State Management Architecture

Zustand Integration with TypeScript:

```
interface AppState {
  // User authentication state
  user: User | null;
  isAuthenticated: boolean;

  // Property management state
  properties: Property[];
  selectedProperty: Property | null;

  // Client management state
  clients: Client[];
  activeClients: Client[];

  // AI service state
  aiRequests: AIRequest[];
  generatedContent: ContentItem[];

  // Actions
  setUser: (user: User | null) => void;
  addProperty: (property: Property) => void;
  updateProperty: (id: string, updates: Partial<Property>) => void;
  addClient: (client: Client) => void;
  updateClientStatus: (id: string, status: ClientStatus) => void;
  addAIRequest: (request: AIRequest) => void;
  updateAIRequest: (id: string, updates: Partial<AIRequest>) => void;
}

const useAppStore = create<AppState>((set, get) => ({
  // Initial state
  user: null,
  isAuthenticated: false,
  properties: [],
  selectedProperty: null,
  clients: [],
  activeClients: [],
  aiRequests: [],
  generatedContent: [],

  // Actions implementation
  setUser: (user) => set({ user, isAuthenticated: !!user }),
```

```
  addProperty: (property) => set((state) => ({
    properties: [...state.properties, property]
  })),

  updateProperty: (id, updates) => set((state) => ({
    properties: state.properties.map(p =>
      p.id === id ? { ...p, ...updates } : p
    )
  })),

  addClient: (client) => set((state) => ({
    clients: [...state.clients, client],
    activeClients: client.status === 'active'
      ? [...state.activeClients, client]
      : state.activeClients
  })),

  updateClientStatus: (id, status) => set((state) => ({
    clients: state.clients.map(c =>
      c.id === id ? { ...c, status } : c
    ),
    activeClients: state.clients
      .map(c => c.id === id ? { ...c, status } : c)
      .filter(c => c.status === 'active')
  })),

  addAIRequest: (request) => set((state) => ({
    aiRequests: [...state.aiRequests, request]
  })),

  updateAIRequest: (id, updates) => set((state) => ({
    aiRequests: state.aiRequests.map(r =>
      r.id === id ? { ...r, ...updates } : r
    )
  }))
}));
```

6.1.2 Navigation and Routing System

React Native 0.71 adds support for Flexbox properties gap, rowGap, and columnGap, which allow you to specify the amount of space between all items in a Flexbox. These properties have been long requested in React Native, and 0.71 adds initial support for gaps defined using pixel values.

Navigation Architecture

```
// Navigation type definitions
export type RootStackParamList = {
  Dashboard: undefined;
  PropertyManagement: { filter?: PropertyFilter };
  PropertyDetail: { propertyId: string };
  PropertyEdit: { propertyId?: string };
  ClientManagement: { filter?: ClientFilter };
  ClientDetail: { clientId: string };
  ClientEdit: { clientId?: string };
  ContentGeneration: { propertyId?: string; contentType?: ContentType };
  TaskManagement: { filter?: TaskFilter };
  AIAssistant: { context?: AIContext };
  Analytics: { timeRange?: TimeRange };
  Settings: undefined;
};

// Stack Navigator Configuration
const Stack = createNativeStackNavigator<RootStackParamList>();

const AppNavigator: React.FC = () => {
  const { isAuthenticated } = useAppStore();

  return (
    <NavigationContainer>
      <Stack.Navigator
        initialRouteName="Dashboard"
        screenOptions={{
          headerStyle: {
            backgroundColor: '#2563eb',
          },
          headerTintColor: '#ffffff',
          headerTitleStyle: {
            fontWeight: 'bold',
          },
```

```
          }}
        >
          {isAuthenticated ? (
            <>
              <Stack.Screen
                name="Dashboard"
                component={DashboardScreen}
                options={{ title: 'PropertyPro AI' }}
              />
              <Stack.Screen
                name="PropertyManagement"
                component={PropertyManagementScreen}
                options={{ title: 'Properties' }}
              />
              <Stack.Screen
                name="PropertyDetail"
                component={PropertyDetailScreen}
                options={({ route }) => ({
                  title: `Property ${route.params.propertyId}`
                })}
              />
              <Stack.Screen
                name="ClientManagement"
                component={ClientManagementScreen}
                options={{ title: 'Clients' }}
              />
              <Stack.Screen
                name="ContentGeneration"
                component={ContentGenerationScreen}
                options={{ title: 'Content Generation' }}
              />
              <Stack.Screen
                name="TaskManagement"
                component={TaskManagementScreen}
                options={{ title: 'Tasks' }}
              />
              <Stack.Screen
                name="AIAssistant"
                component={AIAssistantScreen}
                options={{ title: 'AI Assistant' }}
              />
              <Stack.Screen
                name="Analytics"
```

```
              component={AnalyticsScreen}
              options={{ title: 'Analytics' }}
            />
          </>
        ) : (
          <Stack.Screen
            name="Login"
            component={LoginScreen}
            options={{ headerShown: false }}
          />
        )}
      </Stack.Navigator>
    </NavigationContainer>
  );
};
```

6.1.3 UI Component Library

Design System Components

A perfect design system contains basic typography, colour themes, layouts and of course designed components. These components help development teams to build consistent products in a smooth way.

Base Component Architecture:

```
// Theme configuration
interface Theme {
  colors: {
    primary: string;
    secondary: string;
    success: string;
    warning: string;
    error: string;
    background: string;
    surface: string;
    text: string;
    textSecondary: string;
  };
  spacing: {
```

```
    xs: number;
    sm: number;
    md: number;
    lg: number;
    xl: number;
  };
  typography: {
    h1: TextStyle;
    h2: TextStyle;
    h3: TextStyle;
    body: TextStyle;
    caption: TextStyle;
  };
  borderRadius: {
    sm: number;
    md: number;
    lg: number;
  };
}

// Button component with variants
interface ButtonProps {
  variant: 'primary' | 'secondary' | 'outline' | 'ghost';
  size: 'sm' | 'md' | 'lg';
  disabled?: boolean;
  loading?: boolean;
  onPress: () => void;
  children: React.ReactNode;
}

const Button: React.FC<ButtonProps> = ({
  variant,
  size,
  disabled = false,
  loading = false,
  onPress,
  children
}) => {
  const theme = useTheme();

  const buttonStyles = StyleSheet.create({
    base: {
      borderRadius: theme.borderRadius.md,
```

```
      alignItems: 'center',
      justifyContent: 'center',
      flexDirection: 'row',
      gap: theme.spacing.sm, // Using React Native 0.71+ gap support
    },
    primary: {
      backgroundColor: theme.colors.primary,
    },
    secondary: {
      backgroundColor: theme.colors.secondary,
    },
    outline: {
      backgroundColor: 'transparent',
      borderWidth: 1,
      borderColor: theme.colors.primary,
    },
    ghost: {
      backgroundColor: 'transparent',
    },
    sm: {
      paddingHorizontal: theme.spacing.sm,
      paddingVertical: theme.spacing.xs,
      minHeight: 32,
    },
    md: {
      paddingHorizontal: theme.spacing.md,
      paddingVertical: theme.spacing.sm,
      minHeight: 40,
    },
    lg: {
      paddingHorizontal: theme.spacing.lg,
      paddingVertical: theme.spacing.md,
      minHeight: 48,
    },
    disabled: {
      opacity: 0.5,
    },
  });

  return (
    <TouchableOpacity
      style={[
        buttonStyles.base,
```

```
        buttonStyles[variant],
        buttonStyles[size],
        disabled && buttonStyles.disabled,
      ]}
      onPress={onPress}
      disabled={disabled || loading}
      activeOpacity={0.7}
    >
      {loading && <ActivityIndicator size="small" color="white" />}
      <Text style={getTextStyle(variant, theme)}>{children}</Text>
    </TouchableOpacity>
  );
};
```

Property Management Components

```
// Property Card Component
interface PropertyCardProps {
  property: Property;
  onPress: (property: Property) => void;
  onEdit: (property: Property) => void;
  onDelete: (property: Property) => void;
}

const PropertyCard: React.FC<PropertyCardProps> = ({
  property,
  onPress,
  onEdit,
  onDelete
}) => {
  const theme = useTheme();

  return (
    <TouchableOpacity
      style={styles.card}
      onPress={() => onPress(property)}
      activeOpacity={0.8}
    >
      <View style={styles.imageContainer}>
        <Image
          source={{ uri: property.images[0] }}
```

```
          style={styles.image}
          resizeMode="cover"
        />
        <View style={styles.statusBadge}>
          <Text style={styles.statusText}>
            {property.status.toUpperCase()}
          </Text>
        </View>
      </View>

      <View style={styles.content}>
        <Text style={styles.title} numberOfLines={2}>
          {property.title}
        </Text>

        <Text style={styles.location} numberOfLines={1}>
          {property.location}
        </Text>

        <View style={styles.details}>
          <View style={styles.detailItem}>
            <Icon name="bed" size={16} color={theme.colors.textSecondary}
            <Text style={styles.detailText}>{property.bedrooms}</Text>
          </View>

          <View style={styles.detailItem}>
            <Icon name="bath" size={16} color={theme.colors.textSecondary
            <Text style={styles.detailText}>{property.bathrooms}</Text>
          </View>

          <View style={styles.detailItem}>
            <Icon name="square" size={16} color={theme.colors.textSeconda
            <Text style={styles.detailText}>{property.sizeSqft} sqft</Tex
          </View>
        </View>

        <Text style={styles.price}>
          ${property.price.toLocaleString()}
        </Text>

        <View style={styles.actions}>
          <Button
            variant="outline"
```

```
          size="sm"
          onPress={() => onEdit(property)}
        >
          Edit
        </Button>

        <Button
          variant="ghost"
          size="sm"
          onPress={() => onDelete(property)}
        >
          Delete
        </Button>
      </View>
    </View>
  </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  card: {
    backgroundColor: 'white',
    borderRadius: 12,
    marginHorizontal: 16,
    marginVertical: 8,
    shadowColor: '#000',
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.1,
    shadowRadius: 4,
    elevation: 3,
  },
  imageContainer: {
    position: 'relative',
    height: 200,
  },
  image: {
    width: '100%',
    height: '100%',
    borderTopLeftRadius: 12,
    borderTopRightRadius: 12,
```

```
  },
  statusBadge: {
    position: 'absolute',
    top: 12,
    right: 12,
    backgroundColor: 'rgba(0, 0, 0, 0.7)',
    paddingHorizontal: 8,
    paddingVertical: 4,
    borderRadius: 4,
  },
  statusText: {
    color: 'white',
    fontSize: 12,
    fontWeight: 'bold',
  },
  content: {
    padding: 16,
    gap: 8, // Using React Native 0.71+ gap support
  },
  title: {
    fontSize: 18,
    fontWeight: 'bold',
    color: '#1f2937',
  },
  location: {
    fontSize: 14,
    color: '#6b7280',
  },
  details: {
    flexDirection: 'row',
    gap: 16, // Using React Native 0.71+ gap support
  },
  detailItem: {
    flexDirection: 'row',
    alignItems: 'center',
    gap: 4, // Using React Native 0.71+ gap support
  },
  detailText: {
    fontSize: 12,
    color: '#6b7280',
  },
  price: {
    fontSize: 20,
```

```
    fontWeight: 'bold',
    color: '#059669',
  },
  actions: {
    flexDirection: 'row',
    gap: 8, // Using React Native 0.71+ gap support
    marginTop: 8,
  },
});
```

6.2 BACKEND API COMPONENTS

6.2.1 FastAPI Application Architecture

By following these design principles and patterns, you will be able to build more robust, flexible and maintainable APIs using FastAPI. By applying principles such as SOLID and using patterns such as DAO or Service Layer you will not only improve the quality of your code, but also its ability to adapt to changes and grow over time.

Clean Architecture Implementation

The architecture described here is inspired by hexagonal architecture, sometimes also known as "ports and adapters," although it does not strictly follow that design pattern.

Application Structure:

```
# app/main.py - Application entry point
from fastapi import FastAPI, Depends
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
import uvicorn

from app.core.config import settings
from app.core.database import init_db
from app.api.v1.router import api_router
```

```python
from app.core.dependencies import get_current_user

@asynccontextmanager
async def lifespan(app: FastAPI):
    # Startup
    await init_db()
    yield
    # Shutdown
    pass

app = FastAPI(
    title="PropertyPro AI API",
    description="Intelligent Real Estate Assistant API",
    version="1.0.0",
    lifespan=lifespan,
    docs_url="/docs",
    redoc_url="/redoc"
)

#### CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.ALLOWED_HOSTS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

#### Include API router
app.include_router(
    api_router,
    prefix="/api/v1",
    dependencies=[Depends(get_current_user)]
)

@app.get("/health")
async def health_check():
    return {"status": "healthy", "version": "1.0.0"}

if __name__ == "__main__":
    uvicorn.run(
        "app.main:app",
        host="0.0.0.0",
```

```
        port=8000,
        reload=settings.DEBUG
    )
```

Service Layer Architecture

In this article, I will show you how to apply some of the SOLID principles and design patterns such as DAO (Data Access Object), Service Layer, and Dependency Injection to build robust and efficient APIs with FastAPI. The DAO Pattern is a design pattern used to separate the data access logic from the business logic of the application. Its purpose is to provide an abstraction for CRUD (Create, Read, Update, Delete) operations that are performed on a database or other data source.

Service Layer Implementation:

```
# app/services/property_service.py
from typing import List, Optional
from uuid import UUID
from sqlalchemy.ext.asyncio import AsyncSession

from app.models.property import Property
from app.schemas.property import PropertyCreate, PropertyUpdate, Property
from app.repositories.property_repository import PropertyRepository
from app.services.ai_service import AIService
from app.core.exceptions import PropertyNotFoundError, ValidationError

class PropertyService:
    def __init__(
        self,
        property_repository: PropertyRepository,
        ai_service: AIService,
        db_session: AsyncSession
    ):
        self.property_repository = property_repository
        self.ai_service = ai_service
        self.db_session = db_session

    async def create_property(
```

```
        self,
        property_data: PropertyCreate,
        user_id: UUID
    ) -> PropertyResponse:
        """Create a new property with AI-generated content."""

        # Validate property data
        await self._validate_property_data(property_data)

        # Create property entity
        property_entity = Property(
            **property_data.dict(),
            user_id=user_id,
            status="draft"
        )

        # Save to database
        created_property = await self.property_repository.create(
            property_entity,
            self.db_session
        )

        # Generate AI content asynchronously
        await self._generate_ai_content(created_property.id)

        return PropertyResponse.from_orm(created_property)

    async def get_property(
        self,
        property_id: UUID,
        user_id: UUID
    ) -> PropertyResponse:
        """Get property by ID with user ownership validation."""

        property_entity = await self.property_repository.get_by_id_and_us
            property_id,
            user_id,
            self.db_session
        )

        if not property_entity:
            raise PropertyNotFoundError(f"Property {property_id} not foun
```

```
        return PropertyResponse.from_orm(property_entity)

    async def update_property(
        self,
        property_id: UUID,
        property_data: PropertyUpdate,
        user_id: UUID
    ) -> PropertyResponse:
        """Update property with validation and AI content regeneration.""

        # Get existing property
        existing_property = await self.property_repository.get_by_id_and_
            property_id,
            user_id,
            self.db_session
        )

        if not existing_property:
            raise PropertyNotFoundError(f"Property {property_id} not foun

        # Update property
        updated_property = await self.property_repository.update(
            existing_property,
            property_data.dict(exclude_unset=True),
            self.db_session
        )

        # Regenerate AI content if significant changes
        if self._requires_ai_regeneration(property_data):
            await self._generate_ai_content(property_id)

        return PropertyResponse.from_orm(updated_property)

    async def list_properties(
        self,
        user_id: UUID,
        skip: int = 0,
        limit: int = 100,
        status_filter: Optional[str] = None
    ) -> List[PropertyResponse]:
        """List user properties with filtering and pagination."""

        properties = await self.property_repository.get_by_user(
```

```
            user_id,
            skip=skip,
            limit=limit,
            status_filter=status_filter,
            db_session=self.db_session
        )

        return [PropertyResponse.from_orm(prop) for prop in properties]

    async def delete_property(
        self,
        property_id: UUID,
        user_id: UUID
    ) -> bool:
        """Delete property with user ownership validation."""

        property_entity = await self.property_repository.get_by_id_and_us
            property_id,
            user_id,
            self.db_session
        )

        if not property_entity:
            raise PropertyNotFoundError(f"Property {property_id} not fou

        return await self.property_repository.delete(
            property_entity,
            self.db_session
        )

    async def _validate_property_data(self, property_data: PropertyCreate
        """Validate property data against business rules."""

        if property_data.price <= 0:
            raise ValidationError("Property price must be positive")

        if property_data.bedrooms < 0 or property_data.bathrooms < 0:
            raise ValidationError("Bedrooms and bathrooms must be non-neg

        if property_data.size_sqft <= 0:
            raise ValidationError("Property size must be positive")

    async def _generate_ai_content(self, property_id: UUID) -> None:
```

```python
        """Generate AI content for property asynchronously."""

        # This would typically be handled by a background task
        await self.ai_service.generate_property_description(property_id)
        await self.ai_service.generate_marketing_content(property_id)

    def _requires_ai_regeneration(self, property_data: PropertyUpdate) ->
        """Determine if property changes require AI content regeneration

        significant_fields = {
            'title', 'description', 'property_type',
            'bedrooms', 'bathrooms', 'size_sqft', 'features'
        }

        updated_fields = set(property_data.dict(exclude_unset=True).keys
        return bool(significant_fields.intersection(updated_fields))
```

Repository Pattern Implementation

```python
# app/repositories/property_repository.py
from typing import List, Optional
from uuid import UUID
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy import select, and_, or_
from sqlalchemy.orm import selectinload

from app.models.property import Property
from app.repositories.base_repository import BaseRepository

class PropertyRepository(BaseRepository[Property]):
    def __init__(self):
        super().__init__(Property)

    async def get_by_user(
        self,
        user_id: UUID,
        skip: int = 0,
        limit: int = 100,
        status_filter: Optional[str] = None,
        db_session: AsyncSession = None
    ) -> List[Property]:
```

```
        """Get properties by user with filtering and pagination."""

        query = select(Property).where(Property.user_id == user_id)

        if status_filter:
            query = query.where(Property.status == status_filter)

        query = query.offset(skip).limit(limit).order_by(Property.createc

        result = await db_session.execute(query)
        return result.scalars().all()

    async def get_by_id_and_user(
        self,
        property_id: UUID,
        user_id: UUID,
        db_session: AsyncSession
    ) -> Optional[Property]:
        """Get property by ID with user ownership validation."""

        query = select(Property).where(
            and_(
                Property.id == property_id,
                Property.user_id == user_id
            )
        ).options(
            selectinload(Property.images),
            selectinload(Property.ai_content)
        )

        result = await db_session.execute(query)
        return result.scalar_one_or_none()

    async def search_properties(
        self,
        user_id: UUID,
        search_term: str,
        db_session: AsyncSession
    ) -> List[Property]:
        """Search properties by title, description, or location."""

        query = select(Property).where(
            and_(
```

```
                Property.user_id == user_id,
                or_(
                    Property.title.ilike(f"%{search_term}%"),
                    Property.description.ilike(f"%{search_term}%"),
                    Property.location.ilike(f"%{search_term}%")
                )
            )
        )

        result = await db_session.execute(query)
        return result.scalars().all()

## app/repositories/base_repository.py
from typing import TypeVar, Generic, List, Optional, Type, Any, Dict
from uuid import UUID
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy import select, update, delete
from sqlalchemy.orm import DeclarativeBase

ModelType = TypeVar("ModelType", bound=DeclarativeBase)

class BaseRepository(Generic[ModelType]):
    def __init__(self, model: Type[ModelType]):
        self.model = model

    async def create(
        self,
        entity: ModelType,
        db_session: AsyncSession
    ) -> ModelType:
        """Create a new entity."""

        db_session.add(entity)
        await db_session.commit()
        await db_session.refresh(entity)
        return entity

    async def get_by_id(
        self,
        entity_id: UUID,
        db_session: AsyncSession
    ) -> Optional[ModelType]:
        """Get entity by ID."""
```

```
        query = select(self.model).where(self.model.id == entity_id)
        result = await db_session.execute(query)
        return result.scalar_one_or_none()

    async def update(
        self,
        entity: ModelType,
        update_data: Dict[str, Any],
        db_session: AsyncSession
    ) -> ModelType:
        """Update entity with new data."""

        for field, value in update_data.items():
            setattr(entity, field, value)

        await db_session.commit()
        await db_session.refresh(entity)
        return entity

    async def delete(
        self,
        entity: ModelType,
        db_session: AsyncSession
    ) -> bool:
        """Delete entity."""

        await db_session.delete(entity)
        await db_session.commit()
        return True

    async def list_all(
        self,
        skip: int = 0,
        limit: int = 100,
        db_session: AsyncSession = None
    ) -> List[ModelType]:
        """List all entities with pagination."""

        query = select(self.model).offset(skip).limit(limit)
        result = await db_session.execute(query)
        return result.scalars().all()
```

6.2.2 API Endpoint Architecture

Endpoints in FastAPI are Python functions that handle incoming HTTP requests. They are defined using the @app.route decorator. Endpoints can have path parameters, query parameters, request bodies, and more. FastAPI automatically handles data validation, serialization, and deserialization based on Python type hints.

RESTful API Design

```
# app/api/v1/endpoints/properties.py
from typing import List, Optional
from uuid import UUID
from fastapi import APIRouter, Depends, HTTPException, Query, status
from sqlalchemy.ext.asyncio import AsyncSession

from app.core.database import get_db_session
from app.core.dependencies import get_current_user
from app.schemas.property import (
    PropertyCreate,
    PropertyUpdate,
    PropertyResponse,
    PropertyListResponse
)
from app.schemas.user import User
from app.services.property_service import PropertyService
from app.core.exceptions import PropertyNotFoundError, ValidationError

router = APIRouter(prefix="/properties", tags=["properties"])

@router.post(
    "/",
    response_model=PropertyResponse,
    status_code=status.HTTP_201_CREATED,
    summary="Create a new property",
    description="Create a new property listing with AI-generated content'
)
async def create_property(
    property_data: PropertyCreate,
    current_user: User = Depends(get_current_user),
```

```
    db_session: AsyncSession = Depends(get_db_session),
    property_service: PropertyService = Depends()
) -> PropertyResponse:
    """Create a new property listing."""

    try:
        return await property_service.create_property(
            property_data,
            current_user.id
        )
    except ValidationError as e:
        raise HTTPException(
            status_code=status.HTTP_422_UNPROCESSABLE_ENTITY,
            detail=str(e)
        )

@router.get(
    "/",
    response_model=PropertyListResponse,
    summary="List properties",
    description="Get a paginated list of user properties with optional fi
)
async def list_properties(
    skip: int = Query(0, ge=0, description="Number of properties to skip"
    limit: int = Query(100, ge=1, le=1000, description="Number of propert
    status: Optional[str] = Query(None, description="Filter by property s
    current_user: User = Depends(get_current_user),
    property_service: PropertyService = Depends()
) -> PropertyListResponse:
    """List user properties with pagination and filtering."""

    properties = await property_service.list_properties(
        user_id=current_user.id,
        skip=skip,
        limit=limit,
        status_filter=status
    )

    return PropertyListResponse(
        properties=properties,
        total=len(properties),
        skip=skip,
        limit=limit
```

```
    )

@router.get(
    "/{property_id}",
    response_model=PropertyResponse,
    summary="Get property details",
    description="Get detailed information about a specific property"
)
async def get_property(
    property_id: UUID,
    current_user: User = Depends(get_current_user),
    property_service: PropertyService = Depends()
) -> PropertyResponse:
    """Get property by ID."""

    try:
        return await property_service.get_property(property_id, current_u
    except PropertyNotFoundError:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Property not found"
        )

@router.put(
    "/{property_id}",
    response_model=PropertyResponse,
    summary="Update property",
    description="Update property information and regenerate AI content i
)
async def update_property(
    property_id: UUID,
    property_data: PropertyUpdate,
    current_user: User = Depends(get_current_user),
    property_service: PropertyService = Depends()
) -> PropertyResponse:
    """Update property by ID."""

    try:
        return await property_service.update_property(
            property_id,
            property_data,
            current_user.id
        )
```

```
    except PropertyNotFoundError:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Property not found"
        )
    except ValidationError as e:
        raise HTTPException(
            status_code=status.HTTP_422_UNPROCESSABLE_ENTITY,
            detail=str(e)
        )

@router.delete(
    "/{property_id}",
    status_code=status.HTTP_204_NO_CONTENT,
    summary="Delete property",
    description="Delete a property listing"
)
async def delete_property(
    property_id: UUID,
    current_user: User = Depends(get_current_user),
    property_service: PropertyService = Depends()
) -> None:
    """Delete property by ID."""

    try:
        await property_service.delete_property(property_id, current_user
    except PropertyNotFoundError:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Property not found"
        )

@router.post(
    "/{property_id}/generate-content",
    response_model=dict,
    summary="Generate AI content",
    description="Generate AI-powered content for a property"
)
async def generate_property_content(
    property_id: UUID,
    content_type: str = Query(..., description="Type of content to genera
    current_user: User = Depends(get_current_user),
    property_service: PropertyService = Depends()
```

```
) -> dict:
    """Generate AI content for property."""

    try:
        # Validate property ownership
        await property_service.get_property(property_id, current_user.id)

        # Generate content based on type
        if content_type == "description":
            content = await property_service.generate_description(propert
        elif content_type == "marketing":
            content = await property_service.generate_marketing_content(p
        elif content_type == "social":
            content = await property_service.generate_social_media_conter
        else:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid content type"
            )

        return {"content": content, "type": content_type}

    except PropertyNotFoundError:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Property not found"
        )
```

6.2.3 Dependency Injection System

Handles service injection through FastAPI's dependency injection system: The layer leverages FastAPI's built-in dependency injection to provide services with their required dependencies, making the code more testable and maintainable. Consistent pattern across all entry points: Whether you're handling an HTTP request, a background job, or a CLI command, the same pattern is followed, making the code predictable and easier to maintain.

Dependency Configuration

```python
# app/core/dependencies.py
from typing import AsyncGenerator
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from sqlalchemy.ext.asyncio import AsyncSession
from jose import JWTError, jwt

from app.core.database import get_db_session
from app.core.config import settings
from app.models.user import User
from app.repositories.user_repository import UserRepository
from app.repositories.property_repository import PropertyRepository
from app.repositories.client_repository import ClientRepository
from app.services.property_service import PropertyService
from app.services.client_service import ClientService
from app.services.ai_service import AIService
from app.services.auth_service import AuthService

#### Security
security = HTTPBearer()

async def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security),
    db_session: AsyncSession = Depends(get_db_session)
) -> User:
    """Get current authenticated user from JWT token."""

    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )

    try:
        payload = jwt.decode(
            credentials.credentials,
            settings.SECRET_KEY,
            algorithms=[settings.ALGORITHM]
        )
        user_id: str = payload.get("sub")
        if user_id is None:
            raise credentials_exception
    except JWTError:
```

```
        raise credentials_exception

    user_repository = UserRepository()
    user = await user_repository.get_by_id(user_id, db_session)

    if user is None:
        raise credentials_exception

    return user

#### Repository Dependencies
def get_user_repository() -> UserRepository:
    return UserRepository()

def get_property_repository() -> PropertyRepository:
    return PropertyRepository()

def get_client_repository() -> ClientRepository:
    return ClientRepository()

#### Service Dependencies
def get_ai_service() -> AIService:
    return AIService(
        api_key=settings.OPENAI_API_KEY,
        model="gpt-4.1"
    )

def get_auth_service(
    user_repository: UserRepository = Depends(get_user_repository)
) -> AuthService:
    return AuthService(user_repository)

def get_property_service(
    property_repository: PropertyRepository = Depends(get_property_reposi
    ai_service: AIService = Depends(get_ai_service),
    db_session: AsyncSession = Depends(get_db_session)
) -> PropertyService:
    return PropertyService(property_repository, ai_service, db_session)

def get_client_service(
    client_repository: ClientRepository = Depends(get_client_repository)
    ai_service: AIService = Depends(get_ai_service),
    db_session: AsyncSession = Depends(get_db_session)
```

```
) -> ClientService:
    return ClientService(client_repository, ai_service, db_session)

#### Request Context Dependency
from dataclasses import dataclass
from fastapi import Request

@dataclass
class RequestContext:
    """Request context containing common dependencies."""
    request: Request
    current_user: User
    db_session: AsyncSession

async def get_request_context(
    request: Request,
    current_user: User = Depends(get_current_user),
    db_session: AsyncSession = Depends(get_db_session)
) -> RequestContext:
    """Get request context with common dependencies."""

    return RequestContext(
        request=request,
        current_user=current_user,
        db_session=db_session
    )
```

6.3 AI SERVICE COMPONENTS

6.3.1 OpenAI Integration Architecture

Type Safety and Validation: FastAPI uses Python type hints and Pydantic models for automatic data validation and serialization. This results in type-safe APIs where you catch errors at compile time rather than runtime.

AI Service Implementation

```python
# app/services/ai_service.py
from typing import List, Dict, Any, Optional
import asyncio
from openai import AsyncOpenAI
from pydantic import BaseModel, Field
import tiktoken

from app.core.config import settings
from app.schemas.ai import (
    AIRequest,
    AIResponse,
    ContentGenerationRequest,
    MarketAnalysisRequest
)
from app.core.exceptions import AIServiceError, RateLimitError

class AIService:
    def __init__(self, api_key: str, model: str = "gpt-4.1"):
        self.client = AsyncOpenAI(api_key=api_key)
        self.model = model
        self.encoding = tiktoken.encoding_for_model(model)
        self.max_tokens = 1_000_000  # GPT-4.1 context window
        self.max_completion_tokens = 4_000

    async def generate_property_description(
        self,
        property_data: Dict[str, Any],
        tone: str = "professional",
        target_audience: str = "buyers"
    ) -> str:
        """Generate AI-powered property description."""

        prompt = self._build_property_description_prompt(
            property_data,
            tone,
            target_audience
        )

        try:
            response = await self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {
```

```
                    "role": "system",
                    "content": "You are a professional real estate co
                },
                {
                    "role": "user",
                    "content": prompt
                }
            ],
            max_tokens=self.max_completion_tokens,
            temperature=0.7,
            top_p=0.9
        )

        return response.choices[0].message.content.strip()

    except Exception as e:
        raise AIServiceError(f"Failed to generate property descriptio

async def generate_marketing_content(
    self,
    property_data: Dict[str, Any],
    content_types: List[str] = None
) -> Dict[str, str]:
    """Generate multiple marketing content types for a property."""

    if content_types is None:
        content_types = ["social_media", "email_blast", "flyer", "lis

    tasks = []
    for content_type in content_types:
        task = self._generate_content_by_type(property_data, content_
        tasks.append(task)

    results = await asyncio.gather(*tasks, return_exceptions=True)

    content_dict = {}
    for i, result in enumerate(results):
        if isinstance(result, Exception):
            content_dict[content_types[i]] = f"Error: {str(result)}"
        else:
            content_dict[content_types[i]] = result

    return content_dict
```

```
    async def analyze_market_data(
        self,
        property_data: Dict[str, Any],
        comparable_properties: List[Dict[str, Any]],
        market_trends: Dict[str, Any]
    ) -> Dict[str, Any]:
        """Analyze market data and provide pricing recommendations."""

        prompt = self._build_market_analysis_prompt(
            property_data,
            comparable_properties,
            market_trends
        )

        try:
            response = await self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {
                        "role": "system",
                        "content": "You are a real estate market analyst
                    },
                    {
                        "role": "user",
                        "content": prompt
                    }
                ],
                max_tokens=self.max_completion_tokens,
                temperature=0.3,  # Lower temperature for analytical con
                response_format={"type": "json_object"}
            )

            import json
            return json.loads(response.choices[0].message.content)

        except Exception as e:
            raise AIServiceError(f"Failed to analyze market data: {str(e)

    async def generate_client_communication(
        self,
        client_data: Dict[str, Any],
        communication_type: str,
```

```
        context: Dict[str, Any] = None
    ) -> str:
        """Generate personalized client communication."""

        prompt = self._build_client_communication_prompt(
            client_data,
            communication_type,
            context or {}
        )

        try:
            response = await self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {
                        "role": "system",
                        "content": "You are a professional real estate ag
                    },
                    {
                        "role": "user",
                        "content": prompt
                    }
                ],
                max_tokens=self.max_completion_tokens,
                temperature=0.8
            )

            return response.choices[0].message.content.strip()

        except Exception as e:
            raise AIServiceError(f"Failed to generate client communicati

    async def chat_with_assistant(
        self,
        messages: List[Dict[str, str]],
        context: Dict[str, Any] = None
    ) -> str:
        """Chat with AI assistant about real estate topics."""

        system_message = self._build_assistant_system_message(context or

        chat_messages = [{"role": "system", "content": system_message}]
        chat_messages.extend(messages)
```

```
        # Ensure we don't exceed token limits
        chat_messages = self._truncate_messages_if_needed(chat_messages)

        try:
            response = await self.client.chat.completions.create(
                model=self.model,
                messages=chat_messages,
                max_tokens=self.max_completion_tokens,
                temperature=0.7,
                stream=False
            )

            return response.choices[0].message.content.strip()

        except Exception as e:
            raise AIServiceError(f"Failed to chat with assistant: {str(e)

    def _build_property_description_prompt(
        self,
        property_data: Dict[str, Any],
        tone: str,
        target_audience: str
    ) -> str:
        """Build prompt for property description generation."""

        return f"""
        Generate a compelling property description for the following prop

        Property Details:
        - Type: {property_data.get('property_type', 'N/A')}
        - Price: ${property_data.get('price', 0):,}
        - Bedrooms: {property_data.get('bedrooms', 0)}
        - Bathrooms: {property_data.get('bathrooms', 0)}
        - Size: {property_data.get('size_sqft', 0)} sqft
        - Location: {property_data.get('location', 'N/A')}
        - Features: {', '.join(property_data.get('features', []))}

        Requirements:
        - Tone: {tone}
        - Target Audience: {target_audience}
        - Length: 150-250 words
        - Include key selling points and unique features
```

```
        - Use engaging, descriptive language
        - End with a call to action

        Generate only the description text, no additional formatting or e
        """

    def _build_market_analysis_prompt(
        self,
        property_data: Dict[str, Any],
        comparable_properties: List[Dict[str, Any]],
        market_trends: Dict[str, Any]
    ) -> str:
        """Build prompt for market analysis."""

        comps_text = "\n".join([
            f"- {comp.get('address', 'N/A')}: ${comp.get('price', 0):,}, "
            f"{comp.get('bedrooms', 0)}bed/{comp.get('bathrooms', 0)}bath
            f"{comp.get('size_sqft', 0)} sqft"
            for comp in comparable_properties[:5]
        ])

        return f"""
        Analyze the following property and provide pricing recommendation

        Subject Property:
        - Type: {property_data.get('property_type', 'N/A')}
        - Bedrooms: {property_data.get('bedrooms', 0)}
        - Bathrooms: {property_data.get('bathrooms', 0)}
        - Size: {property_data.get('size_sqft', 0)} sqft
        - Location: {property_data.get('location', 'N/A')}

        Comparable Properties:
        {comps_text}

        Market Trends:
        - Average Days on Market: {market_trends.get('avg_days_on_market
        - Price Trend: {market_trends.get('price_trend', 'N/A')}
        - Inventory Level: {market_trends.get('inventory_level', 'N/A')}

        Provide analysis in JSON format with the following structure:
        {{
            "suggested_price": number,
            "price_range": {{"min": number, "max": number}},
```

```
            "confidence_level": "high|medium|low",
            "key_factors": ["factor1", "factor2", ...],
            "recommendations": ["rec1", "rec2", ...],
            "market_position": "above|at|below market"
        }}
        """

    def _build_client_communication_prompt(
        self,
        client_data: Dict[str, Any],
        communication_type: str,
        context: Dict[str, Any]
    ) -> str:
        """Build prompt for client communication generation."""

        return f"""
        Generate a {communication_type} for the following client:

        Client Information:
        - Name: {client_data.get('name', 'N/A')}
        - Status: {client_data.get('status', 'N/A')}
        - Preferences: {client_data.get('preferences', {})}
        - Last Contact: {client_data.get('last_contact', 'N/A')}

        Context:
        {context}

        Requirements:
        - Professional and personalized tone
        - Include relevant property or market information if applicable
        - Keep appropriate length for {communication_type}
        - Include clear next steps or call to action

        Generate only the communication text, no additional formatting.
        """

    def _build_assistant_system_message(self, context: Dict[str, Any]) ->
        """Build system message for AI assistant chat."""

        return f"""
        You are PropertyPro AI, an intelligent real estate assistant wit
        - Property valuation and market analysis
        - Real estate marketing and sales strategies
```

```
    - Client relationship management
    - Legal and regulatory compliance
    - Investment analysis and recommendations

    Current Context:
    - User: {context.get('user_name', 'Real Estate Professional')}
    - Location: {context.get('location', 'General')}
    - Specialization: {context.get('specialization', 'Residential Rea

    Provide helpful, accurate, and actionable advice. Always conside
    """

def _truncate_messages_if_needed(
    self,
    messages: List[Dict[str, str]]
) -> List[Dict[str, str]]:
    """Truncate messages to fit within token limits."""

    total_tokens = sum(
        len(self.encoding.encode(msg["content"]))
        for msg in messages
    )

    if total_tokens <= self.max_tokens - self.max_completion_tokens:
        return messages

    # Keep system message and truncate from the beginning of conversa
    system_message = messages[0] if messages[0]["role"] == "system" e
    user_messages = messages[1:] if system_message else messages

    truncated_messages = []
    if system_message:
        truncated_messages.append(system_message)

    # Add messages from the end until we approach token limit
    current_tokens = len(self.encoding.encode(system_message["conten

    for message in reversed(user_messages):
        message_tokens = len(self.encoding.encode(message["content"])
        if current_tokens + message_tokens < self.max_tokens - self.m
            truncated_messages.insert(-1 if system_message else 0, me
            current_tokens += message_tokens
        else:
```

```
                break

        return truncated_messages

    async def _generate_content_by_type(
        self,
        property_data: Dict[str, Any],
        content_type: str
    ) -> str:
        """Generate content based on specific type."""

        prompts = {
            "social_media": f"""
            Create an engaging social media post for this property:
            {property_data}

            Requirements:
            - Instagram/Facebook friendly
            - Include relevant hashtags
            - Engaging and visual language
            - 150-200 characters
            """,

            "email_blast": f"""
            Create an email marketing template for this property:
            {property_data}

            Requirements:
            - Professional email format
            - Subject line included
            - Call to action
            - 200-300 words
            """,

            "flyer": f"""
            Create flyer content for this property:
            {property_data}

            Requirements:
            - Headline and key features
            - Bullet points for easy reading
            - Contact information placeholder
            - Print-friendly format
```

```
        """,

        "listing_description": f"""
        Create a detailed listing description for this property:
        {property_data}

        Requirements:
        - MLS-friendly format
        - Comprehensive feature list
        - Neighborhood information
        - 300-400 words
        """
    }

    prompt = prompts.get(content_type, prompts["listing_description"]

    try:
        response = await self.client.chat.completions.create(
            model=self.model,
            messages=[
                {
                    "role": "system",
                    "content": f"You are a real estate marketing spec
                },
                {
                    "role": "user",
                    "content": prompt
                }
            ],
            max_tokens=self.max_completion_tokens,
            temperature=0.8
        )

        return response.choices[0].message.content.strip()

    except Exception as e:
        raise AIServiceError(f"Failed to generate {content_type}: {st
```

6.3.2 Content Generation Pipeline

Background Task Processing

```python
# app/services/background_tasks.py
import asyncio
from typing import Dict, Any
from celery import Celery
from uuid import UUID

from app.services.ai_service import AIService
from app.repositories.property_repository import PropertyRepository
from app.repositories.content_repository import ContentRepository
from app.core.database import get_db_session
from app.core.config import settings

#### Celery configuration for background tasks
celery_app = Celery(
    "propertypro_ai",
    broker=settings.CELERY_BROKER_URL,
    backend=settings.CELERY_RESULT_BACKEND
)

@celery_app.task(bind=True, max_retries=3)
async def generate_property_content_task(
    self,
    property_id: str,
    content_types: list[str],
    user_preferences: Dict[str, Any] = None
):
    """Background task for generating property content."""

    try:
        # Initialize services
        ai_service = AIService(
            api_key=settings.OPENAI_API_KEY,
            model="gpt-4.1"
        )

        property_repository = PropertyRepository()
        content_repository = ContentRepository()

        # Get database session
        async with get_db_session() as db_session:
            # Fetch property data
            property_entity = await property_repository.get_by_id(
                UUID(property_id),
```

```
                db_session
            )

            if not property_entity:
                raise ValueError(f"Property {property_id} not found")

            # Convert to dictionary for AI processing
            property_data = {
                "property_type": property_entity.property_type,
                "price": property_entity.price,
                "bedrooms": property_entity.bedrooms,
                "bathrooms": property_entity.bathrooms,
                "size_sqft": property_entity.size_sqft,
                "location": property_entity.location,
                "features": property_entity.features or [],
                "description": property_entity.description
            }

            # Generate content for each type
            generated_content = {}
            for content_type in content_types:
                try:
                    if content_type == "description":
                        content = await ai_service.generate_property_desc
                            property_data,
                            tone=user_preferences.get("tone", "professior
                            target_audience=user_preferences.get("audienc
                        )
                    elif content_type == "marketing":
                        content = await ai_service.generate_marketing_cor
                            property_data,
                            content_types=["social_media", "email_blast"
                        )
                    else:
                        content = await ai_service._generate_content_by_t
                            property_data,
                            content_type
                        )

                    generated_content[content_type] = content

                    # Save generated content
                    await content_repository.create_content(
```

```
                        property_id=UUID(property_id),
                        content_type=content_type,
                        content=content,
                        db_session=db_session
                    )

                except Exception as content_error:
                    generated_content[content_type] = f"Error: {str(conte

            return {
                "property_id": property_id,
                "generated_content": generated_content,
                "status": "completed"
            }

    except Exception as exc:
        # Retry logic
        if self.request.retries < self.max_retries:
            raise self.retry(countdown=60 * (2 ** self.request.retries))

        return {
            "property_id": property_id,
            "error": str(exc),
            "status": "failed"
        }

@celery_app.task
async def analyze_market_trends_task(location: str, property_type: str):
    """Background task for market trend analysis."""

    try:
        ai_service = AIService(
            api_key=settings.OPENAI_API_KEY,
            model="gpt-4.1"
        )

        # This would typically fetch real market data from external APIs
        # For now, we'll simulate with sample data
        market_data = {
            "location": location,
            "property_type": property_type,
            "avg_price": 450000,
            "avg_days_on_market": 25,
```

```
            "price_trend": "increasing",
            "inventory_level": "low"
        }

        # Generate market analysis
        analysis = await ai_service.analyze_market_data(
            property_data={"location": location, "property_type": propert
            comparable_properties=[],  # Would be fetched from MLS
            market_trends=market_data
        )

        return {
            "location": location,
            "property_type": property_type,
            "analysis": analysis,
            "status": "completed"
        }

    except Exception as exc:
        return {
            "location": location,
            "property_type": property_type,
            "error": str(exc),
            "status": "failed"
        }
```

6.4 DATABASE COMPONENTS

6.4.1 PostgreSQL Schema Design

Database Models with SQLAlchemy

```
# app/models/property.py
from sqlalchemy import Column, String, Integer, Float, DateTime, Text, JS
from sqlalchemy.dialects.postgresql import UUID, ARRAY
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func
import uuid
```

```
from app.core.database import Base

class Property(Base):
    __tablename__ = "properties"

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4)
    user_id = Column(UUID(as_uuid=True), ForeignKey("users.id"), nullable

    # Basic property information
    title = Column(String(255), nullable=False)
    description = Column(Text)
    property_type = Column(String(50), nullable=False)  # apartment, vil
    status = Column(String(20

Based on my analysis of the PropertyPro AI system architecture and curren

## 6.1 CORE SERVICES ARCHITECTURE

#### Core Services Architecture is not applicable for this system

PropertyPro AI is designed as a **monolithic application with modular se

### 6.1.1 System Characteristics Analysis

| Characteristic | PropertyPro AI Reality | Microservices Requirement |
|---|---|---|
| Team Size | Small development team (2-5 developers) | Large teams (8+ d
| Business Complexity | Single domain (real estate) | Multiple distinct b
| Data Consistency | Strong consistency required for client/property data
| Deployment Frequency | Coordinated releases | Independent service deplo

### 6.1.2 Architectural Rationale

**Why Monolithic Architecture is Optimal:**

The system follows clean architecture principles with clear separation o

**Service Layer Organization:**

<div class="mermaid-wrapper" id="mermaid-diagram-7fwp003jd">
          <div class="mermaid">
graph TB
    subgraph &quot;PropertyPro AI Monolithic Architecture&quot;
```

```
        subgraph &quot;Presentation Layer&quot;
            RN[React Native Mobile App&lt;br/&gt;TypeScript 5.0+]
        end

        subgraph &quot;API Gateway Layer&quot;
            FA[FastAPI Application&lt;br/&gt;Single Entry Point]
        end

        subgraph &quot;Business Service Modules&quot;
            PS[Property Service Module]
            CS[Client Service Module]
            AS[AI Service Module]
            TS[Task Service Module]
            ANS[Analytics Service Module]
        end

        subgraph &quot;Data Layer&quot;
            PG[#40;PostgreSQL Database&lt;br/&gt;Single Instance#41;]
            FS[File Storage System]
        end

        subgraph &quot;External Services&quot;
            OAI[OpenAI GPT-4.1 API]
            EMAIL[Email Service]
        end
    end

    RN --&gt; FA
    FA --&gt; PS
    FA --&gt; CS
    FA --&gt; AS
    FA --&gt; TS
    FA --&gt; ANS

    PS --&gt; PG
    CS --&gt; PG
    TS --&gt; PG
    ANS --&gt; PG

    AS --&gt; OAI
    FA --&gt; EMAIL
    PS --&gt; FS
</div>
```

```
        </div>

### 6.1.3 Modular Service Design Within Monolith

**Service Module Boundaries:**

| Service Module | Responsibility | Internal Components |
|---|---|---|
| Property Service | Property CRUD, market analysis | PropertyRepository
| Client Service | CRM functionality, lead management | ClientRepository
| AI Service | Content generation, analysis | OpenAIClient, ContentGenera
| Task Service | Workflow automation | TaskRepository, WorkflowEngine, No

**Inter-Module Communication:**

The system uses layered architecture with distinct separation between API

### 6.1.4 Scalability Through Monolithic Patterns

**Horizontal Scaling Strategy:**

<div class="mermaid-wrapper" id="mermaid-diagram-w1wawsrb9">
          <div class="mermaid">
graph LR
    subgraph &quot;Load Balancer&quot;
        LB[Nginx/HAProxy]
    end

    subgraph &quot;Application Instances&quot;
        APP1[PropertyPro AI Instance 1&lt;br/&gt;FastAPI + Uvicorn]
        APP2[PropertyPro AI Instance 2&lt;br/&gt;FastAPI + Uvicorn]
        APP3[PropertyPro AI Instance 3&lt;br/&gt;FastAPI + Uvicorn]
    end

    subgraph &quot;Shared Resources&quot;
        DB[#40;PostgreSQL&lt;br/&gt;Primary + Replicas#41;]
        CACHE[Redis Cache]
        FILES[Shared File Storage]
    end

    LB --&gt; APP1
    LB --&gt; APP2
    LB --&gt; APP3
```

```
    APP1 --&gt; DB
    APP2 --&gt; DB
    APP3 --&gt; DB

    APP1 --&gt; CACHE
    APP2 --&gt; CACHE
    APP3 --&gt; CACHE

    APP1 --&gt; FILES
    APP2 --&gt; FILES
    APP3 --&gt; FILES
</div>
        </div>

**Performance Optimization Techniques:**

FastAPI applications can serve multiple clients concurrently in a single

| Optimization Technique | Implementation | Expected Benefit |
|---|---|---|
| Async Request Handling | FastAPI with asyncio | 10x concurrent request
| Connection Pooling | SQLAlchemy async pools | Reduced database latency
| Response Caching | Redis with TTL | 80% reduction in API response time
| Background Task Processing | Celery with Redis broker | Non-blocking AI

### 6.1.5 Resilience Patterns for Monolithic Architecture

**Fault Tolerance Mechanisms:**

<div class="mermaid-wrapper" id="mermaid-diagram-263b498g8">
          <div class="mermaid">
graph TD
    A[Request Received] --&gt; B{Health Check}
    B --&gt;|Healthy| C[Process Request]
    B --&gt;|Unhealthy| D[Circuit Breaker Open]

    C --&gt; E{External Service Call}
    E --&gt;|Success| F[Return Response]
    E --&gt;|Failure| G[Retry Logic]

    G --&gt; H{Retry Count &lt; Max}
    H --&gt;|Yes| I[Exponential Backoff]
```

```
    H --&gt;|No| J[Fallback Response]

    I --&gt; E
    J --&gt; F
    D --&gt; K[Return Service Unavailable]

    F --&gt; L[Log Success Metrics]
    K --&gt; M[Log Error Metrics]
    J --&gt; N[Log Fallback Metrics]
</div>
        </div>

**Resilience Implementation:**

| Pattern | Implementation | Purpose |
|---|---|---|
| Circuit Breaker | Custom decorator for OpenAI API calls | Prevent casca
| Retry with Backoff | Exponential backoff for external services | Handle
| Graceful Degradation | Fallback responses for AI services | Maintain co
| Health Checks | `/health` endpoint with dependency checks | Monitor sys

### 6.1.6 Future Migration Path

**When to Consider Microservices:**

The system is designed with clear module boundaries that would facilitate

| Trigger Condition | Current State | Microservices Threshold |
|---|---|---|
| Team Size | 2-5 developers | 15+ developers across multiple teams |
| Request Volume | <10,000 requests/day | >1 million requests/day |
| Feature Complexity | Single real estate domain | Multiple business doma
| Deployment Frequency | Weekly releases | Multiple daily deployments pe

**Migration Strategy:**

If future growth requires microservices, the current modular structure a

### 6.1.7 Conclusion

PropertyPro AI's monolithic architecture with modular service layers prov

- **Development Velocity**: Single codebase with shared libraries and ut
```

```
- **Data Consistency**: ACID transactions across all business operations
- **Operational Simplicity**: Single deployment unit with unified monito
- **Cost Efficiency**: Reduced infrastructure complexity and operational
- **Team Productivity**: Easier debugging, testing, and feature developme

This setup provides a solid foundation for a production-ready FastAPI app

The architecture supports the system's requirements for up to 10,000 conc

## 6.2 DATABASE DESIGN

### 6.2.1 SCHEMA DESIGN

#### 6.2.1.1 Entity Relationships

PropertyPro AI utilizes a comprehensive PostgreSQL 15 database schema des

#### Core Entity Relationship Diagram

<div class="mermaid-wrapper" id="mermaid-diagram-es3a2ruzc">
          <div class="mermaid">
erDiagram
    USERS {
        uuid id PK
        varchar email UK
        varchar password_hash
        varchar first_name
        varchar last_name
        varchar phone
        jsonb preferences
        timestamp created_at
        timestamp updated_at
        boolean is_active
    }

    PROPERTIES {
        uuid id PK
        uuid user_id FK
        varchar title
        text description
        varchar property_type
        decimal price
        integer bedrooms
```

```
        integer bathrooms
        integer size_sqft
        varchar location
        jsonb features
        varchar status
        jsonb ai_analysis
        timestamp created_at
        timestamp updated_at
    }

    CLIENTS {
        uuid id PK
        uuid user_id FK
        varchar name
        varchar email
        varchar phone
        integer lead_score
        varchar nurture_status
        jsonb preferences
        timestamp last_contacted_at
        timestamp created_at
        timestamp updated_at
    }

    TASKS {
        uuid id PK
        uuid user_id FK
        uuid property_id FK
        uuid client_id FK
        varchar title
        text description
        varchar status
        varchar priority
        varchar category
        integer progress
        timestamp due_date
        jsonb ai_suggestions
        timestamp created_at
        timestamp updated_at
    }

    AI_CONTENT {
        uuid id PK
```

```
        uuid property_id FK
        uuid user_id FK
        varchar content_type
        text content
        varchar tone
        integer word_count
        decimal confidence_score
        jsonb metadata
        timestamp created_at
    }

    INTERACTIONS {
        uuid id PK
        uuid client_id FK
        uuid user_id FK
        varchar interaction_type
        text content
        varchar channel
        jsonb metadata
        timestamp created_at
    }

    PROPERTY_IMAGES {
        uuid id PK
        uuid property_id FK
        varchar file_path
        varchar file_name
        integer file_size
        varchar mime_type
        integer sort_order
        timestamp created_at
    }

    USERS ||--o{ PROPERTIES : owns
    USERS ||--o{ CLIENTS : manages
    USERS ||--o{ TASKS : assigned
    USERS ||--o{ AI_CONTENT : generates
    USERS ||--o{ INTERACTIONS : creates
    PROPERTIES ||--o{ TASKS : relates_to
    PROPERTIES ||--o{ AI_CONTENT : describes
    PROPERTIES ||--o{ PROPERTY_IMAGES : contains
    CLIENTS ||--o{ TASKS : involves
    CLIENTS ||--o{ INTERACTIONS : participates_in
```

```
</div>
        </div>

#### 6.2.1.2 Data Models and Structures

#### Core Table Definitions

| Table | Primary Purpose | Key Features | Relationships |
|---|---|---|---|
| users | User authentication and profile | Email uniqueness, password ha
| properties | Property listings and details | JSONB for flexible attribu
| clients | Client relationship management | Lead scoring, nurture status
| ai_content | AI-generated content storage | Content type classification

#### Advanced Data Types Implementation

PostgreSQL's JSONB data type is utilized for JSON data to improve query p

**JSONB Usage Patterns:**
```

sql

```
-- Property features stored as JSONB for flexibility
ALTER TABLE properties ADD COLUMN features JSONB;
CREATE INDEX idx_properties_features_gin ON properties USING GIN (features);

-- AI analysis results with structured metadata
ALTER TABLE ai_content ADD COLUMN metadata JSONB;
CREATE INDEX idx_ai_content_metadata_gin ON ai_content USING GIN (metadata);

-- User preferences for AI customization
ALTER TABLE users ADD COLUMN preferences JSONB;
CREATE INDEX idx_users_preferences_gin ON users USING GIN (preferences);
```

```
**UUID Implementation:**
```

```
All primary keys utilize UUID data type for enhanced security and distril
```

sql
-- UUID extension for PostgreSQL
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Example table with UUID primary key
CREATE TABLE properties (
id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
user_id UUID NOT NULL REFERENCES users(id),
-- other columns
);

```
#### 6.2.1.3 Indexing Strategy

#### Performance-Optimized Index Design

Indexing improves search performance by allowing faster data retrieval. I

**Primary Indexes:**

| Index Name | Table | Columns | Type | Purpose |
|---|---|---|---|
| idx_properties_user_status | properties | user_id, status | B-tree | Us
| idx_clients_user_score | clients | user_id, lead_score | B-tree | Lead
| idx_tasks_user_due | tasks | user_id, due_date | B-tree | Task manageme
| idx_ai_content_property_type | ai_content | property_id, content_type

**Specialized Indexes:**
```

sql
-- GIN indexes for JSONB columns
CREATE INDEX idx_properties_features_gin ON properties USING GIN (features);
CREATE INDEX idx_users_preferences_gin ON users USING GIN (preferences);

-- Partial indexes for active records
CREATE INDEX idx_active_properties ON properties (user_id, created_at)
WHERE status = 'active';

-- Composite indexes for common query patterns
CREATE INDEX idx_tasks_priority_status ON tasks (priority, status, due_date);

-- Text search indexes for property descriptions
CREATE INDEX idx_properties_description_fts ON properties
USING GIN (to_tsvector('english', description));

```
#### 6.2.1.4 Partitioning Approach

#### Time-Based Partitioning Strategy

Partitioning is implemented using range partitioning by date to improve p

**Interactions Table Partitioning:**
```

sql
-- Create partitioned interactions table
CREATE TABLE interactions (
id UUID DEFAULT uuid_generate_v4(),
client_id UUID NOT NULL,
user_id UUID NOT NULL,
interaction_type VARCHAR(50) NOT NULL,
content TEXT,
channel VARCHAR(50),
metadata JSONB,
created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()
) PARTITION BY RANGE (created_at);

-- Create quarterly partitions
CREATE TABLE interactions_2024_q1 PARTITION OF interactions

FOR VALUES FROM ('2024-01-01') TO ('2024-04-01');

CREATE TABLE interactions_2024_q2 PARTITION OF interactions
FOR VALUES FROM ('2024-04-01') TO ('2024-07-01');

CREATE TABLE interactions_2024_q3 PARTITION OF interactions
FOR VALUES FROM ('2024-07-01') TO ('2024-10-01');

CREATE TABLE interactions_2024_q4 PARTITION OF interactions
FOR VALUES FROM ('2024-10-01') TO ('2025-01-01');

```
**AI Content Partitioning:**
```

sql
-- Partition AI content by creation date for efficient archival
CREATE TABLE ai_content (
id UUID DEFAULT uuid_generate_v4(),
property_id UUID,
user_id UUID NOT NULL,
content_type VARCHAR(50) NOT NULL,
content TEXT NOT NULL,
created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()
) PARTITION BY RANGE (created_at);

```
#### 6.2.1.5 Replication Configuration

#### Master-Slave Replication Architecture

Replication in PostgreSQL involves maintaining a real-time copy of a data

<div class="mermaid-wrapper" id="mermaid-diagram-rtaz5w7tq">
          <div class="mermaid">
graph TB
    subgraph &quot;Primary Database Server&quot;
        MASTER[#40;PostgreSQL Primary&lt;br/&gt;Read/Write Operations#41
    end
```

```
    subgraph &quot;Replica Servers&quot;
        REPLICA1[#40;PostgreSQL Replica 1&lt;br/&gt;Read Operations#41;]
        REPLICA2[#40;PostgreSQL Replica 2&lt;br/&gt;Read Operations#41;]
    end

    subgraph &quot;Application Layer&quot;
        WRITE[Write Operations&lt;br/&gt;Properties, Clients, Tasks]
        READ[Read Operations&lt;br/&gt;Analytics, Reports]
    end

    WRITE --&gt; MASTER
    READ --&gt; REPLICA1
    READ --&gt; REPLICA2

    MASTER -.-&gt;|Streaming Replication| REPLICA1
    MASTER -.-&gt;|Streaming Replication| REPLICA2

    style MASTER fill:#e1f5fe
    style REPLICA1 fill:#f3e5f5
    style REPLICA2 fill:#f3e5f5
</div>
        </div>

**Replication Configuration:**

| Parameter | Primary Server | Replica Server | Purpose |
|---|---|---|---|
| wal_level | replica | - | Enable WAL streaming |
| max_wal_senders | 3 | - | Support multiple replicas |
| wal_keep_segments | 64 | - | Retain WAL files |
| hot_standby | - | on | Enable read queries |

#### 6.2.1.6 Backup Architecture

#### Comprehensive Backup Strategy

**Backup Types and Schedule:**

| Backup Type | Frequency | Retention | Storage Location | Purpose |
|---|---|---|---|
| Full Backup | Daily | 30 days | AWS S3 | Complete database restore |
| Incremental WAL | Continuous | 7 days | Local + S3 | Point-in-time rec
| Logical Backup | Weekly | 12 weeks | S3 Glacier | Schema and data expo
```

```
| **Snapshot** Backup | Hourly | 24 hours | **Local storage** | Quick recovery |

**Backup Implementation:**
```

bash
#!/bin/bash

Daily full backup script

pg_basebackup -h localhost -D /backup/$(date +%Y%m%d) -Ft -z -P

Continuous WAL archiving

archive_command = 'cp %p /backup/wal_archive/%f'

Point-in-time recovery setup

restore_command = 'cp /backup/wal_archive/%f %p'
recovery_target_time = '2024-01-15 14:30:00'

```
### 6.2.2 DATA MANAGEMENT

#### 6.2.2.1 Migration Procedures

#### Alembic Migration Framework

The **system** utilizes Alembic for handling database migrations **with** SQLAlch

**Migration Architecture:**
```

python

alembic/env.py - Migration environment configuration

from sqlalchemy import create_engine
from sqlalchemy.ext.asyncio import create_async_engine
from alembic import context
from app.models import Base

def run_migrations_online():
"""Run migrations in 'online' mode with async engine."""

```
connectable = create_async_engine(
    DATABASE_URL,
    poolclass=pool.NullPool,
)

async with connectable.connect() as connection:
    await connection.run_sync(do_run_migrations)
```

def do_run_migrations(connection):
context.configure(
connection=connection,
target_metadata=Base.metadata,
literal_binds=True,
dialect_opts={"paramstyle": "named"},
)

```
with context.begin_transaction():
    context.run_migrations()
```

```
**Migration Workflow:**

| Stage | Command | Purpose | Rollback Strategy |
|---|---|---|---|
```

```
| Generate | `alembic revision --autogenerate` | Create migration script
| Review | Manual inspection | Validate migration logic | Edit before app
| Apply | `alembic upgrade head` | Execute migration | `alembic downgrade
| Verify | Data validation queries | Confirm migration success | Full ro

#### 6.2.2.2 Versioning Strategy

#### Schema Version Management

**Version Control Approach:**
```

python

Migration version naming convention

Format: YYYY_MM_DD_HHMM_description

Example: 2024_01_15_1430_add_ai_content_table.py

"""Add AI content table for generated content storage

Revision ID: a1b2c3d4e5f6
Revises: f6e5d4c3b2a1

Create Date: 2024-01-15 14:30:00.000000

"""
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql

def upgrade():
op.create_table('ai_content',
sa.Column('id', postgresql.UUID(), nullable=False),
sa.Column('property_id', postgresql.UUID(), nullable=True),
sa.Column('user_id', postgresql.UUID(), nullable=False),
sa.Column('content_type', sa.String(50), nullable=False),
sa.Column('content', sa.Text(), nullable=False),
sa.Column('confidence_score', sa.Numeric(3,2), nullable=True),
sa.Column('created_at', sa.TIMESTAMP(timezone=True), nullable=False),
sa.ForeignKeyConstraint(['property_id'], ['properties.id']),
sa.ForeignKeyConstraint(['user_id'], ['users.id']),
sa.PrimaryKeyConstraint('id')
)

```
op.create_index('idx_ai_content_property_type', 'ai_content',
               ['property_id', 'content_type'])
```

def downgrade():
op.drop_index('idx_ai_content_property_type')
op.drop_table('ai_content')

```
#### 6.2.2.3 Archival Policies

#### Data Lifecycle Management

**Archival Strategy by Data Type:**

| Data Category | Active Period | Archive Period | Deletion Policy | Sto
|---|---|---|---|---|
```

```
| User Data | Indefinite | N/A | User-requested only | Primary |
| Property Data | 2 years active | 5 years archive | 7 years total | Pri
| AI Content | 6 months active | 2 years archive | 3 years total | Prima
| Interactions | 1 year active | 6 years archive | 7 years total | Prima

**Automated Archival Implementation:**
```

sql
-- Create archive tables with same structure
CREATE TABLE interactions_archive (LIKE interactions INCLUDING ALL);
CREATE TABLE ai_content_archive (LIKE ai_content INCLUDING ALL);

-- Automated archival procedure
CREATE OR REPLACE FUNCTION archive_old_data()
RETURNS void AS $$
BEGIN
-- Archive interactions older than 1 year
INSERT INTO interactions_archive
SELECT * FROM interactions
WHERE created_at < NOW() - INTERVAL '1 year';

```
DELETE FROM interactions
WHERE created_at < NOW() - INTERVAL '1 year';

-- Archive AI content older than 6 months
INSERT INTO ai_content_archive
SELECT * FROM ai_content
WHERE created_at < NOW() - INTERVAL '6 months';

DELETE FROM ai_content
WHERE created_at < NOW() - INTERVAL '6 months';
```

END;
$$ LANGUAGE plpgsql;

-- Schedule archival job
SELECT cron.schedule('archive-old-data', '0 2 * * 0', 'SELECT

archive_old_data();');

```
#### 6.2.2.4 Data Storage and Retrieval Mechanisms

#### Optimized Storage Patterns

**Column Ordering for Storage Efficiency:**

In PostgreSQL, efficient use of storage space is influenced by column orc
```

sql
-- Optimized column ordering for storage efficiency
CREATE TABLE properties (
-- 8-byte types first
id UUID PRIMARY KEY,
user_id UUID NOT NULL,
price DECIMAL(12,2),
created_at TIMESTAMP WITH TIME ZONE,
updated_at TIMESTAMP WITH TIME ZONE,

```
-- 4-byte types
bedrooms INTEGER,
bathrooms INTEGER,
size_sqft INTEGER,

-- 2-byte types
-- (none in this table)

-- 1-byte types
-- (none in this table)

-- Variable length types last
title VARCHAR(255) NOT NULL,
description TEXT,
property_type VARCHAR(50),
location VARCHAR(255),
status VARCHAR(20) DEFAULT 'draft',
```

```
features JSONB,
ai_analysis JSONB
```

);

```
**Retrieval Optimization Patterns:**
```

python

Async SQLAlchemy 2.0 query patterns for optimal retrieval

from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy import select
from sqlalchemy.orm import selectinload, joinedload

async def get_property_with_content(
property_id: UUID,
session: AsyncSession
) -> Property:
"""Optimized property retrieval with related data."""

```
stmt = select(Property).options(
    selectinload(Property.images),
    selectinload(Property.ai_content),
    joinedload(Property.user)
).where(Property.id == property_id)

result = await session.execute(stmt)
return result.scalar_one_or_none()
```

async def get_user_properties_paginated(
user_id: UUID,

skip: int,
limit: int,
session: AsyncSession
) -> List[Property]:
"""Paginated property retrieval with minimal data."""

```
stmt = select(Property).where(
    Property.user_id == user_id
).offset(skip).limit(limit).order_by(
    Property.created_at.desc()
)

result = await session.execute(stmt)
return result.scalars().all()
```

```
#### 6.2.2.5 Caching Policies

#### Multi-Tier Caching Strategy

**Cache Layer Architecture:**

<div class="mermaid-wrapper" id="mermaid-diagram-7kshgsgm2">
          <div class="mermaid">
graph TB
    subgraph &quot;Application Layer&quot;
        APP[FastAPI Application]
    end

    subgraph &quot;Cache Layers&quot;
        L1[L1: Application Cache&lt;br/&gt;Python Dict/LRU]
        L2[L2: Redis Cache&lt;br/&gt;Distributed Cache]
        L3[L3: Database Cache&lt;br/&gt;PostgreSQL Buffer]
    end

    subgraph &quot;Storage Layer&quot;
        DB[#40;PostgreSQL Database#41;]
    end

    APP --&gt; L1
    L1 --&gt; L2
```

```
    L2 --&gt; L3
    L3 --&gt; DB

    style L1 fill:#e8f5e8
    style L2 fill:#fff3e0
    style L3 fill:#f3e5f5
</div>
        </div>

**Caching Implementation:**

| Cache Type | TTL | Use Case | Invalidation Strategy |
|---|---|---|---|
| Property Details | 15 minutes | Property viewing | Update/delete trigge
| User Preferences | 1 hour | AI customization | User preference changes
| AI Content | 6 hours | Generated content | Content regeneration |
| Market Data | 24 hours | Analytics | Daily refresh |
```

python

Redis caching implementation

import redis.asyncio as redis
from typing import Optional, Any
import json

class CacheManager:
def **init**(self, redis_url: str):
self.redis = redis.from_url(redis_url)

```
async def get(self, key: str) -> Optional[Any]:
    """Get cached value with JSON deserialization."""
    value = await self.redis.get(key)
    return json.loads(value) if value else None

async def set(self, key: str, value: Any, ttl: int = 3600):
```

```
        """Set cached value with JSON serialization."""
        await self.redis.setex(
            key,
            ttl,
            json.dumps(value, default=str)
        )

    async def delete(self, key: str):
        """Delete cached value."""
        await self.redis.delete(key)

    async def invalidate_pattern(self, pattern: str):
        """Invalidate all keys matching pattern."""
        keys = await self.redis.keys(pattern)
        if keys:
            await self.redis.delete(*keys)
```

Usage in service layer

async def get_property_cached(
property_id: UUID,
cache: CacheManager,
session: AsyncSession
) -> Property:
"""Get property with caching."""

```
cache_key = f"property:{property_id}"
cached_property = await cache.get(cache_key)

if cached_property:
    return Property(**cached_property)

property_data = await get_property_from_db(property_id, session)
await cache.set(cache_key, property_data.dict(), ttl=900)  # 15 minutes

return property_data
```

```
### 6.2.3 COMPLIANCE CONSIDERATIONS

#### 6.2.3.1 Data Retention Rules

#### Regulatory Compliance Framework

**Data Retention by Category:**

| Data Type | Retention Period | Regulatory Basis | Deletion Method | Con
|---|---|---|---|---|
| User Personal Data | User-controlled | GDPR Article 17 | Secure deleti
| Financial Records | 7 years | IRS Requirements | Automated archival | E
| Communication Logs | 3 years | Real Estate Law | Secure archival | Clie
| AI Training Data | 2 years | Internal Policy | Anonymization | Model in

**Automated Retention Implementation:**
```

sql
-- Data retention policy enforcement
CREATE OR REPLACE FUNCTION enforce_data_retention()
RETURNS void AS $$
BEGIN
-- Delete AI content older than 2 years
DELETE FROM ai_content_archive
WHERE created_at < NOW() - INTERVAL '2 years';

```
-- Delete interaction logs older than 3 years
DELETE FROM interactions_archive
WHERE created_at < NOW() - INTERVAL '3 years';

-- Anonymize old user data (GDPR compliance)
UPDATE users SET
    email = 'deleted_' || id::text || '@example.com',
    first_name = 'Deleted',
    last_name = 'User',
    phone = NULL,
    preferences = '{}'::jsonb
WHERE last_login_at < NOW() - INTERVAL '2 years'
AND deletion_requested = true;
```

```
-- Log retention actions
INSERT INTO retention_log (action, table_name, records_affected, executed
VALUES ('retention_cleanup', 'multiple', ROW_COUNT, NOW());
```

END;
$$ LANGUAGE plpgsql;

```
#### 6.2.3.2 Backup and Fault Tolerance Policies

#### High Availability Architecture

**Fault Tolerance Mechanisms:**

<div class="mermaid-wrapper" id="mermaid-diagram-8ahvvo6ge">
          <div class="mermaid">
graph TB
    subgraph &quot;Primary Site&quot;
        PRIMARY[#40;Primary Database&lt;br/&gt;Active#41;]
        APP1[Application Server 1]
        APP2[Application Server 2]
    end

    subgraph &quot;Secondary Site&quot;
        STANDBY[#40;Standby Database&lt;br/&gt;Hot Standby#41;]
        APP3[Application Server 3]
    end

    subgraph &quot;Backup Storage&quot;
        S3[AWS S3&lt;br/&gt;Daily Backups]
        GLACIER[AWS Glacier&lt;br/&gt;Long-term Archive]
    end

    subgraph &quot;Monitoring&quot;
        MONITOR[Health Monitoring&lt;br/&gt;Automated Failover]
    end

    PRIMARY -.-&gt;|Streaming Replication| STANDBY
    PRIMARY --&gt; S3
    S3 --&gt; GLACIER

    MONITOR --&gt; PRIMARY
```

```
    MONITOR --&gt; STANDBY
    MONITOR --&gt; APP1
    MONITOR --&gt; APP2
    MONITOR --&gt; APP3

    style PRIMARY fill:#e1f5fe
    style STANDBY fill:#fff3e0
    style S3 fill:#f3e5f5
</div>
        </div>

**Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO):**

| Scenario | RTO Target | RPO Target | Recovery Method | Automation Leve
|---|---|---|---|---|
| Primary Server Failure | < 5 minutes | < 1 minute | Automatic failover
| Data Center Outage | < 30 minutes | < 5 minutes | Manual failover | Se
| Data Corruption | < 2 hours | < 15 minutes | Point-in-time recovery | M
| Complete Disaster | < 24 hours | < 1 hour | Full restore from backup |

#### 6.2.3.3 Privacy Controls

#### GDPR and Privacy Implementation

**Data Privacy Architecture:**
```

python

Privacy control implementation

from cryptography.fernet import Fernet
from sqlalchemy import event
from sqlalchemy.orm import Session

class PrivacyManager:
def **init**(self, encryption_key: bytes):

self.cipher = Fernet(encryption_key)

```
def encrypt_pii(self, data: str) -> str:
    """Encrypt personally identifiable information."""
    return self.cipher.encrypt(data.encode()).decode()

def decrypt_pii(self, encrypted_data: str) -> str:
    """Decrypt personally identifiable information."""
    return self.cipher.decrypt(encrypted_data.encode()).decode()
```

Automatic PII encryption on insert/update

@event.listens_for(User, 'before_insert')
@event.listens_for(User, 'before_update')
def encrypt_user_pii(mapper, connection, target):
"""Automatically encrypt PII fields."""
privacy_manager = PrivacyManager(ENCRYPTION_KEY)

```
if target.email:
    target.email_encrypted = privacy_manager.encrypt_pii(target.email)
if target.phone:
    target.phone_encrypted = privacy_manager.encrypt_pii(target.phone)
```

Data anonymization for analytics

def anonymize_user_data(user_id: UUID) -> dict:
"""Anonymize user data for analytics while preserving utility."""
return {
'user_hash': hashlib.sha256(str(user_id).encode()).hexdigest()[:16],
'registration_month': user.created_at.strftime('%Y-%m'),
'activity_level': calculate_activity_level(user_id),
'property_count': get_property_count(user_id)
}

```
**Privacy Control Matrix:**

| Data Field | Encryption | Anonymization | Access Control | Retention Po
|---|---|---|---|---|
| Email Address | AES-256 | Hash for analytics | User + Admin | User-con
| Phone Number | AES-256 | Removed | User + Admin | User-controlled |
| Property Address | None | Zip code only | User + Admin | 7 years |
| AI Content | None | User ID hash | User only | 2 years |

#### 6.2.3.4 Audit Mechanisms

#### Comprehensive Audit Trail

**Audit Table Structure:**
```

sql
CREATE TABLE audit_log (
id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
table_name VARCHAR(50) NOT NULL,
record_id UUID NOT NULL,
user_id UUID,
action VARCHAR(20) NOT NULL, -- INSERT, UPDATE, DELETE
old_values JSONB,
new_values JSONB,
changed_fields TEXT[],
ip_address INET,
user_agent TEXT,
created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes for audit queries
CREATE INDEX idx_audit_log_table_record ON audit_log (table_name, record_id);
CREATE INDEX idx_audit_log_user_action ON audit_log (user_id, action, created_at);
CREATE INDEX idx_audit_log_created_at ON audit_log (created_at);

```
**Automated Audit Triggers:**
```

sql
-- Generic audit trigger function
CREATE OR REPLACE FUNCTION audit_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
IF TG_OP = 'DELETE' THEN
INSERT INTO audit_log (
table_name, record_id, user_id, action, old_values, created_at
) VALUES (
TG_TABLE_NAME, OLD.id, OLD.user_id, 'DELETE',
row_to_json(OLD), NOW()
);
RETURN OLD;
ELSIF TG_OP = 'UPDATE' THEN
INSERT INTO audit_log (
table_name, record_id, user_id, action,
old_values, new_values, changed_fields, created_at
) VALUES (
TG_TABLE_NAME, NEW.id, NEW.user_id, 'UPDATE',
row_to_json(OLD), row_to_json(NEW),
get_changed_fields(OLD, NEW), NOW()
);
RETURN NEW;
ELSIF TG_OP = 'INSERT' THEN
INSERT INTO audit_log (
table_name, record_id, user_id, action, new_values, created_at
) VALUES (
TG_TABLE_NAME, NEW.id, NEW.user_id, 'INSERT',
row_to_json(NEW), NOW()
);
RETURN NEW;

END IF;
RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- Apply audit triggers to critical tables
CREATE TRIGGER properties_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON properties
FOR EACH ROW EXECUTE FUNCTION audit_trigger_function();

CREATE TRIGGER clients_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON clients
FOR EACH ROW EXECUTE FUNCTION audit_trigger_function();

```
#### 6.2.3.5 Access Controls

#### Role-Based Access Control (RBAC)

**Database Role Hierarchy:**
```

sql
-- Create database roles
CREATE ROLE propertypro_admin;
CREATE ROLE propertypro_agent;
CREATE ROLE propertypro_readonly;
CREATE ROLE propertypro_analytics;

-- Admin permissions (full access)
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO propertypro_admin;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO propertypro_admin;

-- Agent permissions (own data only)
GRANT SELECT, INSERT, UPDATE ON properties TO propertypro_agent;

GRANT SELECT, INSERT, UPDATE ON clients TO propertypro_agent;
GRANT SELECT, INSERT, UPDATE ON tasks TO propertypro_agent;
GRANT SELECT, INSERT ON ai_content TO propertypro_agent;

-- Readonly permissions
GRANT SELECT ON ALL TABLES IN SCHEMA public TO propertypro_readonly;

-- Analytics permissions (anonymized data only)
GRANT SELECT ON analytics_views TO propertypro_analytics;

-- Row Level Security (RLS) policies
ALTER TABLE properties ENABLE ROW LEVEL SECURITY;
ALTER TABLE clients ENABLE ROW LEVEL SECURITY;
ALTER TABLE tasks ENABLE ROW LEVEL SECURITY;

-- Policy: Users can only access their own data
CREATE POLICY user_data_policy ON properties
FOR ALL TO propertypro_agent
USING (user_id = current_setting('app.current_user_id')::uuid);

CREATE POLICY user_data_policy ON clients
FOR ALL TO propertypro_agent
USING (user_id = current_setting('app.current_user_id')::uuid);

```
### 6.2.4 PERFORMANCE OPTIMIZATION

#### 6.2.4.1 Query Optimization Patterns

#### Advanced Query Optimization Techniques

Query performance optimization involves refining queries for faster execu

**Optimized Query Patterns:**
```

sql
-- Efficient property search with multiple filters
EXPLAIN (ANALYZE, BUFFERS)

```sql
SELECT p.id, p.title, p.price, p.location, p.status
FROM properties p
WHERE p.user_id = $1
AND p.status = 'active'
AND p.price BETWEEN $2 AND $3
AND p.bedrooms >= $4
ORDER BY p.created_at DESC
LIMIT 20;

-- Index supporting the above query
CREATE INDEX idx_properties_user_search ON properties
(user_id, status, price, bedrooms, created_at DESC);

-- Efficient client lead scoring query
WITH lead_metrics AS (
SELECT
c.id,
c.name,
c.lead_score,
COUNT(i.id) as interaction_count,
MAX(i.created_at) as last_interaction
FROM clients c
LEFT JOIN interactions i ON c.id = i.client_id
WHERE c.user_id = $1
AND c.nurture_status IN ('hot', 'warm')
GROUP BY c.id, c.name, c.lead_score
)
SELECT * FROM lead_metrics
WHERE interaction_count > 0
ORDER BY lead_score DESC, last_interaction DESC
LIMIT 50;
```

```
**Query Performance Monitoring:**
```

python

SQLAlchemy query performance monitoring

from sqlalchemy import event
from sqlalchemy.engine import Engine
import time
import logging

logger = logging.getLogger('query_performance')

@event.listens_for(Engine, "before_cursor_execute")
def receive_before_cursor_execute(conn, cursor, statement, parameters, context, executemany):
context._query_start_time = time.time()

@event.listens_for(Engine, "after_cursor_execute")
def receive_after_cursor_execute(conn, cursor, statement, parameters, context, executemany):
total = time.time() - context._query_start_time

```
if total > 0.1:  # Log slow queries (>100ms)
    logger.warning(
        f"Slow query detected: {total:.3f}s - {statement[:100]}..."
    )
```

```
#### 6.2.4.2 Caching Strategy

#### Intelligent Caching Implementation

**Cache-Aside Pattern with Redis:**
```

python
from typing import Optional, Any, List
import redis.asyncio as redis
import json
from datetime import timedelta

class PropertyCacheManager:
def **init**(self, redis_client: redis.Redis):
self.redis = redis_client
self.default_ttl = 900 # 15 minutes

```
async def get_property(self, property_id: str) -> Optional[dict]:
    """Get property from cache."""
    cache_key = f"property:{property_id}"
    cached_data = await self.redis.get(cache_key)

    if cached_data:
        return json.loads(cached_data)
    return None

async def set_property(self, property_id: str, property_data: dict, ttl:
    """Cache property data."""
    cache_key = f"property:{property_id}"
    ttl = ttl or self.default_ttl

    await self.redis.setex(
        cache_key,
        ttl,
        json.dumps(property_data, default=str)
    )

async def invalidate_property(self, property_id: str):
    """Invalidate property cache."""
    cache_key = f"property:{property_id}"
    await self.redis.delete(cache_key)

async def get_user_properties(self, user_id: str, page: int = 1) -> Optio
    """Get cached user properties list."""
    cache_key = f"user_properties:{user_id}:page:{page}"
    cached_data = await self.redis.get(cache_key)
```

```
    if cached_data:
        return json.loads(cached_data)
    return None

async def cache_user_properties(self, user_id: str, page: int, propertie:
    """Cache user properties list."""
    cache_key = f"user_properties:{user_id}:page:{page}"

    await self.redis.setex(
        cache_key,
        300,  # 5 minutes for list data
        json.dumps(properties, default=str)
    )
```

```
**Write-Through Caching for Critical Data:**
```

python
async def update_property_with_cache(
property_id: UUID,
update_data: dict,
session: AsyncSession,
cache: PropertyCacheManager
) -> Property:
"""Update property with write-through caching."""

```
# Update database
property_obj = await session.get(Property, property_id)
for key, value in update_data.items():
    setattr(property_obj, key, value)

await session.commit()
await session.refresh(property_obj)

# Update cache immediately
property_dict = {
    'id': str(property_obj.id),
    'title': property_obj.title,
```

```
    'price': float(property_obj.price),
    'status': property_obj.status,
    'updated_at': property_obj.updated_at.isoformat()
}

await cache.set_property(str(property_id), property_dict)

# Invalidate related caches
await cache.redis.delete(f"user_properties:{property_obj.user_id}:*")

return property_obj
```

```
#### 6.2.4.3 Connection Pooling

#### Optimized Connection Management

Connection pooling reduces overhead in high-concurrency environments. The

**AsyncPG Connection Pool Configuration:**
```

python

Database connection pool configuration

from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession
from sqlalchemy.orm import sessionmaker
from sqlalchemy.pool import QueuePool

Optimized engine configuration

engine = create_async_engine(
DATABASE_URL,

Connection pool settings

```
poolclass=QueuePool,
pool_size=20,           # Base number of connections
max_overflow=30,        # Additional connections under load
pool_pre_ping=True,     # Validate connections before use
pool_recycle=3600,      # Recycle connections every hour
```

Performance settings

```
echo=False,             # Disable SQL logging in production
echo_pool=False,        # Disable pool logging
```

Connection arguments

```
connect_args={
    "server_settings": {
        "application_name": "PropertyPro_AI",
        "jit": "off",  # Disable JIT for consistent performance
    },
    "command_timeout": 60,
    "statement_cache_size": 0,  # Disable prepared statement cache
}
```

)

Session factory with optimized settings

AsyncSessionLocal = sessionmaker(
engine,
class_=AsyncSession,
expire_on_commit=False, # Keep objects accessible after commit
autoflush=True, # Auto-flush before queries
autocommit=False # Explicit transaction control
)

```
**Connection Pool Monitoring:**
```

python
import asyncio
from sqlalchemy import event
from sqlalchemy.pool import Pool
import logging

logger = logging.getLogger('connection_pool')

@event.listens_for(Pool, "connect")
def set_sqlite_pragma(dbapi_connection, connection_record):
"""Configure connection-specific settings."""
logger.info("New database connection established")

@event.listens_for(Pool, "checkout")
def receive_checkout(dbapi_connection, connection_record, connection_proxy):
"""Monitor connection checkout."""
pool = connection_proxy.pool
logger.debug(f"Connection checked out. Pool size: {pool.size()}, Checked out: {pool.checkedout()}")

@event.listens_for(Pool, "checkin")
def receive_checkin(dbapi_connection, connection_record):
"""Monitor connection checkin."""
logger.debug("Connection checked in")

Pool health monitoring

async def monitor_pool_health():
"""Monitor connection pool health."""
while True:
pool = engine.pool
logger.info(f"Pool stats - Size: {pool.size()}, Checked out:

{pool.checkedout()}, Overflow: {pool.overflow()}")
await asyncio.sleep(60) # Check every minute

```
#### 6.2.4.4 Read/Write Splitting

#### Master-Replica Query Distribution

**Intelligent Query Routing:**
```

python
from enum import Enum
from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession
from typing import Optional

class DatabaseOperation(Enum):
READ = "read"
WRITE = "write"

class DatabaseManager:
def **init**(self, master_url: str, replica_urls: List[str]):
# Master database for writes
self.master_engine = create_async_engine(
master_url,
pool_size=10,
max_overflow=20
)

```
    # Replica databases for reads
    self.replica_engines = [
        create_async_engine(url, pool_size=15, max_overflow=25)
        for url in replica_urls
    ]

    self.replica_index = 0

def get_engine(self, operation: DatabaseOperation):
    """Get appropriate engine based on operation type."""
```

```
    if operation == DatabaseOperation.WRITE:
        return self.master_engine
    else:
        # Round-robin load balancing for reads
        engine = self.replica_engines[self.replica_index]
        self.replica_index = (self.replica_index + 1) % len(self.replica_
        return engine

async def get_session(self, operation: DatabaseOperation) -> AsyncSessio
    """Get database session for specific operation."""
    engine = self.get_engine(operation)
    return AsyncSession(engine)
```

Usage in service layer

class PropertyService:
def **init**(self, db_manager: DatabaseManager):
self.db_manager = db_manager

```
async def get_property(self, property_id: UUID) -> Optional[Property]:
    """Read operation - use replica."""
    async with self.db_manager.get_session(DatabaseOperation.READ) as ses
        stmt = select(Property).where(Property.id == property_id)
        result = await session.execute(stmt)
        return result.scalar_one_or_none()

async def create_property(self, property_data: PropertyCreate) -> Proper
    """Write operation - use master."""
    async with self.db_manager.get_session(DatabaseOperation.WRITE) as se
        property_obj = Property(**property_data.dict())
        session.add(property_obj)
        await session.commit()
        await session.refresh(property_obj)
        return property_obj
```

```
#### 6.2.4.5 Batch Processing Approach

#### Efficient Bulk Operations
```

```
**Batch Insert Optimization:**
```

python
from sqlalchemy.dialects.postgresql import insert
from sqlalchemy import text
import asyncio
from typing import List, Dict, Any

class BatchProcessor:
def **init**(self, session: AsyncSession, batch_size: int = 1000):
self.session = session
self.batch_size = batch_size

```
async def bulk_insert_properties(self, properties_data: List[Dict[str, A
    """Efficient bulk property insertion."""

    # Process in batches to avoid memory issues
    for i in range(0, len(properties_data), self.batch_size):
        batch = properties_data[i:i + self.batch_size]

        # Use PostgreSQL COPY for maximum performance
        await self._copy_insert_properties(batch)

async def _copy_insert_properties(self, batch: List[Dict[str, Any]]):
    """Use PostgreSQL COPY for bulk insert."""

    # Prepare data for COPY
    copy_data = []
    for prop in batch:
        copy_data.append(
            f"{prop['id']}\t{prop['user_id']}\t{prop['title']}\t"
            f"{prop['price']}\t{prop['bedrooms']}\t{prop['bathrooms']}\t'
            f"{prop['location']}\t{prop['status']}\t{prop['created_at']}'
        )

    copy_sql = """
    COPY properties (id, user_id, title, price, bedrooms, bathrooms, loca
    FROM STDIN WITH (FORMAT text, DELIMITER E'\t')
    """
```

```
        # Execute COPY command
        raw_connection = await self.session.connection()
        await raw_connection.execute(text(copy_sql), copy_data)

    async def bulk_update_lead_scores(self, client_scores: List[Dict[str, Any
        """Efficient bulk lead score updates."""

        # Use VALUES clause for bulk updates
        values_clause = ", ".join([
            f"('{item['client_id']}', {item['lead_score']})"
            for item in client_scores
        ])

        update_sql = f"""
        UPDATE clients
        SET lead_score = updates.score,
            updated_at = NOW()
        FROM (VALUES {values_clause}) AS updates(id, score)
        WHERE clients.id = updates.id::uuid
        """

        await self.session.execute(text(update_sql))
        await self.session.commit()

    async def batch_ai_content_generation(self, property_ids: List[UUID]):
        """Process AI content generation in batches."""

        semaphore = asyncio.Semaphore(5)  # Limit concurrent AI requests

        async def process_property(property_id: UUID):
            async with semaphore:
                # Generate AI content for property
                content = await self._generate_ai_content(property_id)

                # Insert generated content
                ai_content = AIContent(
                    property_id=property_id,
                    content_type='description',
                    content=content,
                    created_at=datetime.utcnow()
                )
                self.session.add(ai_content)
```

```
    # Process all properties concurrently with semaphore limit
    tasks = [process_property(pid) for pid in property_ids]
    await asyncio.gather(*tasks)
    await self.session.commit()
```

```
**Performance Monitoring and Optimization:**
```

python
import time
from contextlib import asynccontextmanager
from sqlalchemy import text

@asynccontextmanager
async def performance_monitor(session: AsyncSession, operation_name: str):
"""Monitor database operation performance."""
start_time = time.time()

```
# Get initial connection stats
stats_before = await session.execute(
    text("SELECT * FROM pg_stat_database WHERE datname = current_database
)

try:
    yield
finally:
    end_time = time.time()
    duration = end_time - start_time

    # Get final connection stats
    stats_after = await session.execute(
        text("SELECT * FROM pg_stat_database WHERE datname = current_data
    )

    logger.info(f"{operation_name} completed in {duration:.3f}s")
```

```
    if duration > 1.0:  # Log slow operations
        logger.warning(f"Slow operation detected: {operation_name} took 
```

Usage example

async def create_properties_batch(properties_data: List[Dict]):
async with get_db_session() as session:
async with performance_monitor(session, "bulk_property_creation"):
batch_processor = BatchProcessor(session)
await batch_processor.bulk_insert_properties(properties_data)

```
This comprehensive database design provides PropertyPro AI with a robust

## 6.3 INTEGRATION ARCHITECTURE

### 6.3.1 API DESIGN

#### 6.3.1.1 Protocol Specifications

PropertyPro AI implements a comprehensive API architecture that facilitat

**Core API Protocol Stack:**

| Protocol Layer | Technology | Purpose | Implementation |
|---|---|---|---|
| Application Protocol | HTTP/HTTPS | Client-server communication | TLS 1
| API Architecture | REST with OpenAPI 3.0 | Resource-oriented endpoints
| Data Format | JSON | Request/response serialization | Pydantic model va
| Real-time Communication | WebSocket (Future) | AI assistant chat inter

**API Endpoint Structure:**

<div class="mermaid-wrapper" id="mermaid-diagram-r46qwv3na">
          <div class="mermaid">
graph TB
    subgraph &quot;API Gateway Layer&quot;
        GATEWAY[&quot;FastAPI Gateway&lt;br/&gt;Port 8000&quot;]
    end

    subgraph &quot;Core API Endpoints&quot;
```

```
        AUTH[&quot;/api/v1/auth/*&lt;br/&gt;Authentication&quot;]
        PROPS[&quot;/api/v1/properties/*&lt;br/&gt;Property Management&qu
        CLIENTS[&quot;/api/v1/clients/*&lt;br/&gt;Client Management&quot
        CONTENT[&quot;/api/v1/ai/content/*&lt;br/&gt;Content Generation&q
        TASKS[&quot;/api/v1/tasks/*&lt;br/&gt;Task Management&quot;]
        ANALYTICS[&quot;/api/v1/analytics/*&lt;br/&gt;Performance Data&qu
    end

    subgraph &quot;External Integrations&quot;
        OPENAI[&quot;OpenAI GPT-4.1 API&lt;br/&gt;Content Generation&quot
        EMAIL[&quot;Email Service&lt;br/&gt;SMTP Integration&quot;]
        STORAGE[&quot;File Storage&lt;br/&gt;Property Images&quot;]
    end

    GATEWAY --&gt; AUTH
    GATEWAY --&gt; PROPS
    GATEWAY --&gt; CLIENTS
    GATEWAY --&gt; CONTENT
    GATEWAY --&gt; TASKS
    GATEWAY --&gt; ANALYTICS

    CONTENT --&gt; OPENAI
    TASKS --&gt; EMAIL
    PROPS --&gt; STORAGE
</div>
        </div>

#### 6.3.1.2 Authentication Methods

**JWT-Based Authentication Architecture:**

The system implements a comprehensive JWT-based authentication system wi

| Authentication Method | Use Case | Token Expiry | Security Features |
|---|---|---|---|
| JWT Access Token | API request authentication | 24 hours | bcrypt passw
| JWT Refresh Token | Token renewal | 30 days | Automatic rotation |
| API Key Authentication | External service access | No expiry | Rate lir
| Session Management | Mobile app persistence | 7 days | Secure storage

**Authentication Flow Implementation:**

<div class="mermaid-wrapper" id="mermaid-diagram-wn4dnjalm">
```

```
          <div class="mermaid">
sequenceDiagram
    participant Mobile as React Native App
    participant API as FastAPI Backend
    participant DB as PostgreSQL
    participant AI as OpenAI GPT-4.1

    Mobile-&gt;&gt;API: POST /auth/login
    API-&gt;&gt;DB: Validate credentials
    DB--&gt;&gt;API: User verified
    API--&gt;&gt;Mobile: JWT tokens (access + refresh)

    Mobile-&gt;&gt;API: GET /properties (with JWT)
    API-&gt;&gt;API: Validate JWT token
    API-&gt;&gt;DB: Fetch user properties
    DB--&gt;&gt;API: Property data
    API--&gt;&gt;Mobile: Properties response

    Mobile-&gt;&gt;API: POST /ai/content/generate
    API-&gt;&gt;API: Validate JWT + rate limits
    API-&gt;&gt;AI: Generate content request
    Note over AI: GPT-4.1 with 1M token context
    AI--&gt;&gt;API: Generated content
    API-&gt;&gt;DB: Store content + usage
    API--&gt;&gt;Mobile: Content response
</div>
        </div>

#### 6.3.1.3 Authorization Framework

**Role-Based Access Control (RBAC):**

Authentication verifies who a user is, while authorization controls what

| User Role | Permissions | Resource Access | API Endpoints |
|---|---|---|---|
| Agent | Full CRUD on own data | Properties, Clients, Tasks | All /api/v
| Team Member | Read access to shared data | Team properties, analytics
| Admin | Full system access | All data, user management | All endpoints
| API Client | Programmatic access | Rate-limited operations | Specific i

**Authorization Implementation:**
```

python

FastAPI dependency injection for authorization

from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer

security = HTTPBearer()

async def get_current_user(
credentials: HTTPAuthorizationCredentials = Depends(security),
db_session: AsyncSession = Depends(get_db_session)
) -> User:
"""Extract and validate JWT token for user authentication."""

```
try:
    payload = jwt.decode(
        credentials.credentials,
        settings.SECRET_KEY,
        algorithms=[settings.ALGORITHM]
    )
    user_id: str = payload.get("sub")
    if user_id is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials"
        )
except JWTError:
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Invalid authentication credentials"
    )

user = await get_user_by_id(user_id, db_session)
if user is None:
    raise HTTPException(
```

```
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="User not found"
        )

    return user
```

async def require_agent_role(
current_user: User = Depends(get_current_user)
) -> User:
"""Ensure user has agent-level permissions."""

```
if current_user.role not in ["agent", "admin"]:
    raise HTTPException(
        status_code=status.HTTP_403_FORBIDDEN,
        detail="Insufficient permissions"
    )

return current_user
```

```
#### 6.3.1.4 Rate Limiting Strategy

**Multi-Tier Rate Limiting:**

AI-generated integration code should incorporate robust error handling, i

| Service Tier | Rate Limit | Time Window | Enforcement Level |
|---|---|---|---|
| Authentication | 10 requests/minute | Per IP address | API Gateway |
| Property Operations | 100 requests/hour | Per user | Application layer
| AI Content Generation | 50 requests/hour | Per user | Service layer |
| OpenAI API Calls | 100 requests/minute | Per API key | External service

**Rate Limiting Implementation:**
```

python
from fastapi import Request, HTTPException
from slowapi import Limiter, _rate_limit_exceeded_handler

from slowapi.util import get_remote_address
from slowapi.errors import RateLimitExceeded

Initialize rate limiter

limiter = Limiter(key_func=get_remote_address)

Rate limiting decorators

@limiter.limit("10/minute")
async def login_endpoint(request: Request, credentials: UserLogin):
"""Login endpoint with rate limiting."""
pass

@limiter.limit("100/hour")
async def create_property(
request: Request,
property_data: PropertyCreate,
current_user: User = Depends(get_current_user)
):
"""Property creation with user-based rate limiting."""
pass

AI service rate limiting with circuit breaker

class AIServiceRateLimiter:
def **init**(self, max_requests: int = 50, time_window: int = 3600):
self.max_requests = max_requests
self.time_window = time_window
self.request_counts = {}

```
async def check_rate_limit(self, user_id: str) -> bool:
    """Check if user has exceeded AI service rate limits."""
    current_time = time.time()
    user_requests = self.request_counts.get(user_id, [])
```

Remove old requests outside time window

```
    user_requests = [
        req_time for req_time in user_requests
        if current_time - req_time < self.time_window
    ]

    if len(user_requests) >= self.max_requests:
        return False

    user_requests.append(current_time)
    self.request_counts[user_id] = user_requests
    return True
```

```
#### 6.3.1.5 Versioning Approach

**API Versioning Strategy:**

| Versioning Method | Implementation | Use Case | Migration Path |
|---|---|---|---|
| URL Path Versioning | `/api/v1/`, `/api/v2/` | Major API changes | Para
| Header Versioning | `API-Version: 1.0` | Minor version updates | Backwa
| Content Negotiation | `Accept: application/vnd.api+json;version=1` | Gr
| Semantic Versioning | `1.2.3` format | Release management | Standard ve

**Version Management Implementation:**
```

python
from fastapi import APIRouter, Header
from typing import Optional

Version-specific routers

v1_router = APIRouter(prefix="/api/v1", tags=["v1"])
v2_router = APIRouter(prefix="/api/v2", tags=["v2"])

@v1_router.get("/properties")
async def get_properties_v1(

current_user: User = Depends(get_current_user)
):
"""Version 1 properties endpoint."""
return await get_properties_legacy(current_user)

@v2_router.get("/properties")
async def get_properties_v2(
current_user: User = Depends(get_current_user),
api_version: Optional[str] = Header(None, alias="API-Version")
):
"""Version 2 properties endpoint with enhanced features."""
return await get_properties_enhanced(current_user, api_version)

Version deprecation handling

class APIVersionManager:
SUPPORTED_VERSIONS = ["1.0", "1.1", "2.0"]
DEPRECATED_VERSIONS = ["1.0"]

```
@staticmethod
def validate_version(version: str) -> bool:
    """Validate API version and handle deprecation warnings."""
    if version not in APIVersionManager.SUPPORTED_VERSIONS:
        raise HTTPException(
            status_code=400,
            detail=f"Unsupported API version: {version}"
        )

    if version in APIVersionManager.DEPRECATED_VERSIONS:
```

Log deprecation warning

```
        logger.warning(f"Deprecated API version used: {version}")

    return True
```

```
#### 6.3.1.6 Documentation Standards

**OpenAPI 3.0 Documentation:**

FastAPI is based on OpenAPI. That's what makes it possible to have multi

| Documentation Type | Tool | Access URL | Update Frequency |
|---|---|---|---|
| Interactive API Docs | Swagger UI | `/docs` | Automatic |
| Alternative API Docs | ReDoc | `/redoc` | Automatic |
| OpenAPI Specification | JSON Schema | `/openapi.json` | Automatic |
| Integration Examples | Custom docs | `/examples` | Manual updates |

**Documentation Configuration:**
```

python
from fastapi import FastAPI
from fastapi.openapi.utils import get_openapi

app = FastAPI(
title="PropertyPro AI API",
description="Intelligent Real Estate Assistant API",
version="1.0.0",
docs_url="/docs",
redoc_url="/redoc",
openapi_url="/openapi.json"
)

def custom_openapi():
"""Generate custom OpenAPI schema with enhanced documentation."""
if app.openapi_schema:
return app.openapi_schema

```
openapi_schema = get_openapi(
    title="PropertyPro AI API",
    version="1.0.0",
    description="Comprehensive API for AI-powered real estate operations"
```

```
    routes=app.routes,
)

# Add custom security schemes
openapi_schema["components"]["securitySchemes"] = {
    "BearerAuth": {
        "type": "http",
        "scheme": "bearer",
        "bearerFormat": "JWT"
    },
    "ApiKeyAuth": {
        "type": "apiKey",
        "in": "header",
        "name": "X-API-Key"
    }
}

# Add example responses
openapi_schema["components"]["examples"] = {
    "PropertyResponse": {
        "summary": "Property response example",
        "value": {
            "id": "123e4567-e89b-12d3-a456-426614174000",
            "title": "Luxury Downtown Condo",
            "price": 750000,
            "bedrooms": 2,
            "bathrooms": 2,
            "status": "active"
        }
    }
}

app.openapi_schema = openapi_schema
return app.openapi_schema
```

app.openapi = custom_openapi

```
### 6.3.2 MESSAGE PROCESSING

#### 6.3.2.1 Event Processing Patterns

**Asynchronous Event-Driven Architecture:**
```

```
PropertyPro AI implements event-driven patterns for handling AI content g

| Event Type | Processing Pattern | Trigger Condition | Response Time |
|---|---|---|---|
| Property Created | Async Task Creation | New property listing | < 1 sec
| AI Content Request | Queue-based Processing | User content generation
| Client Interaction | Real-time Notification | Client activity | < 2 sec
| Task Completion | Workflow Automation | Task status change | < 1 second

**Event Processing Implementation:**

<div class="mermaid-wrapper" id="mermaid-diagram-7mkkdf5fr">
          <div class="mermaid">
graph TB
    subgraph &quot;Event Sources&quot;
        USER[User Actions]
        SYSTEM[System Events]
        EXTERNAL[External APIs]
    end

    subgraph &quot;Event Processing Layer&quot;
        QUEUE[Event Queue&lt;br/&gt;Python asyncio]
        PROCESSOR[Event Processor&lt;br/&gt;Background Tasks]
        ROUTER[Event Router&lt;br/&gt;Pattern Matching]
    end

    subgraph &quot;Event Handlers&quot;
        AI_HANDLER[AI Content Handler]
        NOTIFICATION_HANDLER[Notification Handler]
        TASK_HANDLER[Task Automation Handler]
        ANALYTICS_HANDLER[Analytics Handler]
    end

    subgraph &quot;Output Systems&quot;
        DATABASE[#40;PostgreSQL#41;]
        AI_SERVICE[OpenAI GPT-4.1]
        EMAIL_SERVICE[Email Service]
        MOBILE_APP[React Native App]
    end

    USER --&gt; QUEUE
    SYSTEM --&gt; QUEUE
```

```
    EXTERNAL --&gt; QUEUE

    QUEUE --&gt; PROCESSOR
    PROCESSOR --&gt; ROUTER

    ROUTER --&gt; AI_HANDLER
    ROUTER --&gt; NOTIFICATION_HANDLER
    ROUTER --&gt; TASK_HANDLER
    ROUTER --&gt; ANALYTICS_HANDLER

    AI_HANDLER --&gt; AI_SERVICE
    AI_HANDLER --&gt; DATABASE
    NOTIFICATION_HANDLER --&gt; EMAIL_SERVICE
    NOTIFICATION_HANDLER --&gt; MOBILE_APP
    TASK_HANDLER --&gt; DATABASE
    ANALYTICS_HANDLER --&gt; DATABASE
</div>
        </div>

#### 6.3.2.2 Message Queue Architecture

**Background Task Processing:**

The system implements a lightweight background task processing system usi
```

python
from fastapi import BackgroundTasks
from typing import Dict, Any, List
import asyncio
from enum import Enum

class EventType(Enum):
PROPERTY_CREATED = "property_created"
AI_CONTENT_REQUESTED = "ai_content_requested"
CLIENT_INTERACTION = "client_interaction"
TASK_COMPLETED = "task_completed"

class EventProcessor:
def **init**(self):

self.event_queue = asyncio.Queue()
self.handlers = {}
self.running = False

```
def register_handler(self, event_type: EventType, handler):
    """Register event handler for specific event type."""
    if event_type not in self.handlers:
        self.handlers[event_type] = []
    self.handlers[event_type].append(handler)

async def publish_event(self, event_type: EventType, data: Dict[str, Any]
    """Publish event to processing queue."""
    event = {
        "type": event_type,
        "data": data,
        "timestamp": datetime.utcnow(),
        "id": str(uuid.uuid4())
    }
    await self.event_queue.put(event)

async def process_events(self):
    """Main event processing loop."""
    self.running = True
    while self.running:
        try:
            event = await asyncio.wait_for(
                self.event_queue.get(),
                timeout=1.0
            )
            await self._handle_event(event)
        except asyncio.TimeoutError:
            continue
        except Exception as e:
            logger.error(f"Event processing error: {e}")

async def _handle_event(self, event: Dict[str, Any]):
    """Route event to appropriate handlers."""
    event_type = event["type"]
    handlers = self.handlers.get(event_type, [])

    if not handlers:
        logger.warning(f"No handlers for event type: {event_type}")
```

```
        return

    # Execute all handlers concurrently
    tasks = [handler(event["data"]) for handler in handlers]
    await asyncio.gather(*tasks, return_exceptions=True)
```

Global event processor instance

event_processor = EventProcessor()

Event handlers

async def handle_property_created(data: Dict[str, Any]):
"""Handle property creation events."""
property_id = data["property_id"]
user_id = data["user_id"]

Create automated tasks

```
tasks = [
    {"title": "Generate AI property description", "property_id": property
    {"title": "Create social media content", "property_id": property_id}
    {"title": "Set up client notifications", "property_id": property_id}
]

for task_data in tasks:
    await create_automated_task(task_data, user_id)
```

async def handle_ai_content_requested(data: Dict[str, Any]):
"""Handle AI content generation requests."""
request_id = data["request_id"]
property_id = data["property_id"]
content_type = data["content_type"]

```
try:
    # Generate content using OpenAI
    content = await generate_ai_content(property_id, content_type)
```

```
        # Store generated content
        await store_ai_content(request_id, content)

        # Notify user of completion
        await notify_content_ready(data["user_id"], request_id)

    except Exception as e:
        logger.error(f"AI content generation failed: {e}")
        await notify_content_error(data["user_id"], request_id, str(e))
```

Register event handlers

event_processor.register_handler(EventType.PROPERTY_CREATED, handle_property_created)
event_processor.register_handler(EventType.AI_CONTENT_REQUESTED, handle_ai_content_requested)

```
#### 6.3.2.3 Stream Processing Design

**Real-time Data Processing:**

For future implementation of real-time features like AI assistant chat an

| Stream Type | Processing Method | Latency Target | Use Case |
|---|---|---|---|
| AI Chat Messages | WebSocket streaming | < 500ms | Real-time AI assista
| Property Updates | Server-Sent Events | < 1 second | Live property sta
| Task Notifications | Push notifications | < 2 seconds | Task reminders
| Analytics Data | Batch streaming | < 5 minutes | Performance metrics |

#### 6.3.2.4 Batch Processing Flows

**Scheduled Batch Operations:**
```

python
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from datetime import datetime, timedelta

class BatchProcessor:
def **init**(self):
self.scheduler = AsyncIOScheduler()
self.setup_scheduled_jobs()

```
def setup_scheduled_jobs(self):
    """Configure scheduled batch processing jobs."""

    # Daily analytics aggregation
    self.scheduler.add_job(
        self.process_daily_analytics,
        'cron',
        hour=1,
        minute=0,
        id='daily_analytics'
    )

    # Weekly client follow-up automation
    self.scheduler.add_job(
        self.process_client_followups,
        'cron',
        day_of_week='mon',
        hour=9,
        minute=0,
        id='weekly_followups'
    )

    # Monthly performance reports
    self.scheduler.add_job(
        self.generate_monthly_reports,
        'cron',
        day=1,
        hour=8,
        minute=0,
        id='monthly_reports'
    )

async def process_daily_analytics(self):
    """Process daily analytics and performance metrics."""
    yesterday = datetime.utcnow() - timedelta(days=1)

    # Aggregate property views, client interactions, task completions
```

```
    analytics_data = await aggregate_daily_metrics(yesterday)

    # Store aggregated data
    await store_analytics_summary(analytics_data)

    # Generate insights using AI
    insights = await generate_performance_insights(analytics_data)
    await store_ai_insights(insights)

async def process_client_followups(self):
    """Process automated client follow-up tasks."""

    # Find clients needing follow-up
    clients_needing_followup = await get_clients_for_followup()

    for client in clients_needing_followup:
        # Generate personalized follow-up content
        followup_content = await generate_followup_content(client)

        # Create follow-up task
        await create_followup_task(client.id, followup_content)

        # Schedule email if appropriate
        if client.email_preferences.allow_automated:
            await schedule_followup_email(client, followup_content)
```

batch_processor = BatchProcessor()

```
#### 6.3.2.5 Error Handling Strategy

**Comprehensive Error Recovery:**
```

python
from tenacity import retry, stop_after_attempt, wait_exponential
import logging

class ErrorHandler:
def **init**(self):

self.error_counts = {}
self.circuit_breakers = {}

```
@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=4, max=10)
)
async def handle_ai_request(self, request_data: Dict[str, Any]):
    """Handle AI requests with retry logic."""
    try:
        response = await call_openai_api(request_data)
        return response
    except Exception as e:
        logger.error(f"AI request failed: {e}")
        await self.log_error("ai_request", str(e))
        raise

async def handle_database_error(self, operation: str, error: Exception):
    """Handle database operation errors."""
    error_key = f"db_{operation}"
    self.error_counts[error_key] = self.error_counts.get(error_key, 0) +

    if self.error_counts[error_key] > 5:
        # Circuit breaker pattern
        await self.activate_circuit_breaker(error_key)

    # Log error with context
    logger.error(f"Database error in {operation}: {error}")

    # Attempt recovery
    if "connection" in str(error).lower():
        await self.attempt_db_reconnection()

async def handle_external_service_error(self, service: str, error: Excep
    """Handle external service integration errors."""

    if service == "openai":
        # Check for rate limiting
        if "rate_limit" in str(error).lower():
            await self.handle_rate_limit_error(service)
        else:
            await self.handle_service_unavailable(service)
```

```
        elif service == "email":
            # Queue email for later retry
            await self.queue_failed_email(error)

    async def log_error(self, error_type: str, error_message: str):
        """Log error with structured data for monitoring."""
        error_data = {
            "type": error_type,
            "message": error_message,
            "timestamp": datetime.utcnow(),
            "count": self.error_counts.get(error_type, 0)
        }

        # Store in database for analysis
        await store_error_log(error_data)

        # Alert if critical
        if error_type in ["ai_request", "database", "authentication"]:
            await send_error_alert(error_data)
```

error_handler = ErrorHandler()

```
### 6.3.3 EXTERNAL SYSTEMS

#### 6.3.3.1 Third-Party Integration Patterns

**OpenAI GPT-4.1 API Integration:**

Introducing GPT-4.1 in the API—a new family of models with across-the-boa

| Integration Service | Protocol | Authentication | Rate Limits | Error H
|---|---|---|---|---|
| OpenAI GPT-4.1 API | HTTPS/REST | API Key | 100 req/min | Circuit breal
| Email Service (SMTP) | SMTP/TLS | Username/Password | 1000 emails/day
| File Storage | HTTP/REST | API Key | 10GB storage | Fallback storage |
| Push Notifications | HTTP/REST | Service Token | 10000 notifications/da

**OpenAI Integration Implementation:**
```

python
from openai import AsyncOpenAI
from typing import Dict, Any, Optional
import asyncio
from tenacity import retry, stop_after_attempt, wait_exponential

class OpenAIIntegration:
def **init**(self, api_key: str):
self.client = AsyncOpenAI(api_key=api_key)
self.model = "gpt-4.1"
self.max_tokens = 1_000_000 # 1M token context window
self.max_completion_tokens = 4_000

```
@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=4, max=10)
)
async def generate_property_description(
    self,
    property_data: Dict[str, Any],
    tone: str = "professional"
) -> str:
    """Generate property description using GPT-4.1."""

    prompt = self._build_property_prompt(property_data, tone)

    try:
        response = await self.client.chat.completions.create(
            model=self.model,
            messages=[
                {
                    "role": "system",
                    "content": "You are a professional real estate copyw
                },
                {
                    "role": "user",
                    "content": prompt
                }
            ],
            max_tokens=self.max_completion_tokens,
```

```
                temperature=0.7,
                top_p=0.9
            )

            return response.choices[0].message.content.strip()

        except Exception as e:
            logger.error(f"OpenAI API error: {e}")
            raise AIServiceError(f"Failed to generate content: {str(e)}")

    async def analyze_market_data(
        self,
        property_data: Dict[str, Any],
        market_context: Dict[str, Any]
    ) -> Dict[str, Any]:
        """Analyze market data and provide pricing recommendations."""

        prompt = self._build_market_analysis_prompt(property_data, market_co

        try:
            response = await self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {
                        "role": "system",
                        "content": "You are a real estate market analyst prov
                    },
                    {
                        "role": "user",
                        "content": prompt
                    }
                ],
                max_tokens=self.max_completion_tokens,
                temperature=0.3,  # Lower temperature for analytical content
                response_format={"type": "json_object"}
            )

            import json
            return json.loads(response.choices[0].message.content)

        except Exception as e:
            logger.error(f"Market analysis error: {e}")
            raise AIServiceError(f"Failed to analyze market data: {str(e)}")
```

```
def _build_property_prompt(self, property_data: Dict[str, Any], tone: st
    """Build optimized prompt for property description generation."""

    return f"""
    Generate a compelling property description for:

    Property Details:
    - Type: {property_data.get('property_type', 'N/A')}
    - Price: ${property_data.get('price', 0):,}
    - Bedrooms: {property_data.get('bedrooms', 0)}
    - Bathrooms: {property_data.get('bathrooms', 0)}
    - Size: {property_data.get('size_sqft', 0)} sqft
    - Location: {property_data.get('location', 'N/A')}
    - Features: {', '.join(property_data.get('features', []))}

    Requirements:
    - Tone: {tone}
    - Length: 150-250 words
    - Include key selling points
    - End with call to action

    Generate only the description text.
    """
```

```
#### 6.3.3.2 Legacy System Interfaces

**Database Integration Patterns:**

PropertyPro AI is designed as a greenfield application without legacy sys

| Integration Type | Interface Method | Data Format | Sync Frequency |
|---|---|---|---|
| CRM Systems | REST API | JSON/XML | Real-time |
| MLS Platforms | RETS/Web API | XML/JSON | Daily batch |
| Email Marketing | Webhook/API | JSON | Event-driven |
| Accounting Systems | CSV/API | JSON/CSV | Weekly batch |

#### 6.3.3.3 API Gateway Configuration

**Centralized API Management:**
```

python
from fastapi import FastAPI, Request, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from fastapi.middleware.trustedhost import TrustedHostMiddleware
import time
import logging

class APIGateway:
def **init**(self, app: FastAPI):
self.app = app
self.setup_middleware()
self.setup_security()
self.setup_monitoring()

```
def setup_middleware(self):
    """Configure API gateway middleware."""

    # CORS middleware
    self.app.add_middleware(
        CORSMiddleware,
        allow_origins=["*"],  # Configure for production
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )

    # Trusted host middleware
    self.app.add_middleware(
        TrustedHostMiddleware,
        allowed_hosts=["localhost", "*.propertypro-ai.com"]
    )

    # Custom request logging middleware
    @self.app.middleware("http")
    async def log_requests(request: Request, call_next):
        start_time = time.time()

        # Log request
        logger.info(f"Request: {request.method} {request.url}")
```

```
        response = await call_next(request)

        # Log response
        process_time = time.time() - start_time
        logger.info(f"Response: {response.status_code} ({process_time:.3

        return response

def setup_security(self):
    """Configure security headers and policies."""

    @self.app.middleware("http")
    async def add_security_headers(request: Request, call_next):
        response = await call_next(request)

        # Security headers
        response.headers["X-Content-Type-Options"] = "nosniff"
        response.headers["X-Frame-Options"] = "DENY"
        response.headers["X-XSS-Protection"] = "1; mode=block"
        response.headers["Strict-Transport-Security"] = "max-age=31536000

        return response

def setup_monitoring(self):
    """Configure API monitoring and health checks."""

    @self.app.get("/health")
    async def health_check():
        """API health check endpoint."""
        return {
            "status": "healthy",
            "timestamp": datetime.utcnow(),
            "version": "1.0.0"
        }

    @self.app.get("/metrics")
    async def get_metrics():
        """API metrics endpoint."""
        return {
            "requests_total": get_request_count(),
            "response_time_avg": get_avg_response_time(),
```

```
            "error_rate": get_error_rate()
        }
```

```
#### 6.3.3.4 External Service Contracts

**Service Level Agreements (SLAs):**

| External Service | Availability SLA | Response Time SLA | Error Rate SI
|---|---|---|---|---|
| OpenAI GPT-4.1 API | 99.9% | < 5 seconds | < 1% | Cached responses |
| Email Service | 99.5% | < 10 seconds | < 2% | Queue retry |
| File Storage | 99.9% | < 2 seconds | < 0.5% | Local backup |
| Push Notifications | 99.0% | < 5 seconds | < 5% | Email fallback |

**Contract Monitoring Implementation:**
```

python
import asyncio
import aiohttp
from datetime import datetime, timedelta
from typing import Dict, Any

class ServiceContractMonitor:
def **init**(self):
self.service_stats = {}
self.alert_thresholds = {
"openai": {"response_time": 5.0, "error_rate": 0.01},
"email": {"response_time": 10.0, "error_rate": 0.02},
"storage": {"response_time": 2.0, "error_rate": 0.005}
}

```
async def monitor_service_health(self, service_name: str):
    """Monitor external service health and SLA compliance."""

    while True:
        try:
            start_time = time.time()
```

```
                # Perform health check
                health_status = await self.check_service_health(service_name)

                response_time = time.time() - start_time

                # Update statistics
                await self.update_service_stats(
                    service_name,
                    response_time,
                    health_status
                )

                # Check SLA compliance
                await self.check_sla_compliance(service_name)

            except Exception as e:
                logger.error(f"Service monitoring error for {service_name}: -
                await self.record_service_error(service_name, str(e))

            # Wait before next check
            await asyncio.sleep(60)  # Check every minute

    async def check_service_health(self, service_name: str) -> bool:
        """Perform health check for specific service."""

        health_endpoints = {
            "openai": "https://api.openai.com/v1/models",
            "email": "smtp://smtp.gmail.com:587",
            "storage": "https://api.storage-service.com/health"
        }

        endpoint = health_endpoints.get(service_name)
        if not endpoint:
            return False

        try:
            async with aiohttp.ClientSession() as session:
                async with session.get(endpoint, timeout=10) as response:
                    return response.status == 200
        except:
            return False
```

```python
    async def check_sla_compliance(self, service_name: str):
        """Check if service is meeting SLA requirements."""

        stats = self.service_stats.get(service_name, {})
        thresholds = self.alert_thresholds.get(service_name, {})

        # Check response time SLA
        avg_response_time = stats.get("avg_response_time", 0)
        if avg_response_time > thresholds.get("response_time", float('inf'))
            await self.send_sla_alert(
                service_name,
                "response_time",
                avg_response_time
            )

        # Check error rate SLA
        error_rate = stats.get("error_rate", 0)
        if error_rate > thresholds.get("error_rate", 1.0):
            await self.send_sla_alert(
                service_name,
                "error_rate",
                error_rate
            )
```

service_monitor = ServiceContractMonitor()

```
### 6.3.4 INTEGRATION FLOW DIAGRAMS

#### 6.3.4.1 Complete System Integration Flow

<div class="mermaid-wrapper" id="mermaid-diagram-zns8z0ahz">
          <div class="mermaid">
graph TB
    subgraph &quot;Mobile Client Layer&quot;
        RN[React Native App&lt;br/&gt;TypeScript 5.0+]
        AUTH_CLIENT[Authentication Client]
        API_CLIENT[API Client with Axios]
    end

    subgraph &quot;API Gateway Layer&quot;
        GATEWAY[FastAPI Gateway&lt;br/&gt;Rate Limiting &amp; Security]
        CORS[CORS Middleware]
```

```
        LOGGING[Request Logging]
    end

    subgraph &quot;Application Services&quot;
        AUTH_SERVICE[Authentication Service&lt;br/&gt;JWT Management]
        PROPERTY_SERVICE[Property Service&lt;br/&gt;CRUD Operations]
        AI_SERVICE[AI Service&lt;br/&gt;Content Generation]
        CLIENT_SERVICE[Client Service&lt;br/&gt;CRM Operations]
        TASK_SERVICE[Task Service&lt;br/&gt;Workflow Management]
    end

    subgraph &quot;External Integrations&quot;
        OPENAI[OpenAI GPT-4.1 API&lt;br/&gt;1M Token Context]
        EMAIL[Email Service&lt;br/&gt;SMTP Integration]
        STORAGE[File Storage&lt;br/&gt;Property Images]
        NOTIFICATIONS[Push Notifications&lt;br/&gt;Mobile Alerts]
    end

    subgraph &quot;Data Layer&quot;
        POSTGRES[#40;PostgreSQL 15&lt;br/&gt;Primary Database#41;]
        REDIS[Redis Cache&lt;br/&gt;Session Storage]
        FILES[File System&lt;br/&gt;Local Storage]
    end

    RN --&gt; AUTH_CLIENT
    RN --&gt; API_CLIENT

    AUTH_CLIENT --&gt; GATEWAY
    API_CLIENT --&gt; GATEWAY

    GATEWAY --&gt; CORS
    GATEWAY --&gt; LOGGING
    GATEWAY --&gt; AUTH_SERVICE
    GATEWAY --&gt; PROPERTY_SERVICE
    GATEWAY --&gt; AI_SERVICE
    GATEWAY --&gt; CLIENT_SERVICE
    GATEWAY --&gt; TASK_SERVICE

    AI_SERVICE --&gt; OPENAI
    CLIENT_SERVICE --&gt; EMAIL
    PROPERTY_SERVICE --&gt; STORAGE
    TASK_SERVICE --&gt; NOTIFICATIONS
```

```
    AUTH_SERVICE --&gt; POSTGRES
    AUTH_SERVICE --&gt; REDIS
    PROPERTY_SERVICE --&gt; POSTGRES
    AI_SERVICE --&gt; POSTGRES
    CLIENT_SERVICE --&gt; POSTGRES
    TASK_SERVICE --&gt; POSTGRES

    PROPERTY_SERVICE --&gt; FILES
</div>
        </div>

#### 6.3.4.2 AI Content Generation Integration Flow

<div class="mermaid-wrapper" id="mermaid-diagram-nt0jeevqd">
          <div class="mermaid">
sequenceDiagram
    participant Mobile as React Native App
    participant Gateway as API Gateway
    participant PropertyService as Property Service
    participant AIService as AI Service
    participant OpenAI as GPT-4.1 API
    participant Database as PostgreSQL
    participant EventProcessor as Event Processor

    Mobile-&gt;&gt;Gateway: POST /api/v1/ai/content/generate
    Gateway-&gt;&gt;Gateway: Validate JWT &amp; Rate Limits
    Gateway-&gt;&gt;PropertyService: Get property data
    PropertyService-&gt;&gt;Database: SELECT property details
    Database--&gt;&gt;PropertyService: Property data
    PropertyService--&gt;&gt;Gateway: Property information

    Gateway-&gt;&gt;AIService: Generate content request
    AIService-&gt;&gt;AIService: Build optimized prompt
    AIService-&gt;&gt;OpenAI: Chat completion request
    Note over OpenAI: GPT-4.1 with 1M token context&lt;br/&gt;Enhanced i
    OpenAI--&gt;&gt;AIService: Generated content

    AIService-&gt;&gt;Database: Store generated content
    AIService-&gt;&gt;EventProcessor: Publish content_generated event
    EventProcessor-&gt;&gt;EventProcessor: Process background tasks

    AIService--&gt;&gt;Gateway: Content response
    Gateway--&gt;&gt;Mobile: Generated content + metadata
```

```
    EventProcessor-&gt;&gt;Database: Update usage analytics
    EventProcessor-&gt;&gt;Mobile: Push notification (content ready)
</div>
        </div>

#### 6.3.4.3 Authentication and Authorization Flow

<div class="mermaid-wrapper" id="mermaid-diagram-c2pq8hbn5">
          <div class="mermaid">
sequenceDiagram
    participant Mobile as React Native App
    participant Gateway as API Gateway
    participant AuthService as Auth Service
    participant Database as PostgreSQL
    participant Redis as Redis Cache

    Mobile-&gt;&gt;Gateway: POST /api/v1/auth/login
    Gateway-&gt;&gt;Gateway: Rate limit check (10/min)
    Gateway-&gt;&gt;AuthService: Validate credentials
    AuthService-&gt;&gt;Database: SELECT user by email
    Database--&gt;&gt;AuthService: User data
    AuthService-&gt;&gt;AuthService: Verify password (bcrypt)

    alt Valid Credentials
        AuthService-&gt;&gt;AuthService: Generate JWT tokens
        AuthService-&gt;&gt;Redis: Store refresh token
        AuthService--&gt;&gt;Gateway: Access + Refresh tokens
        Gateway--&gt;&gt;Mobile: Authentication success

        Note over Mobile: Store tokens securely

        Mobile-&gt;&gt;Gateway: GET /api/v1/properties (with JWT)
        Gateway-&gt;&gt;Gateway: Validate JWT token
        Gateway-&gt;&gt;Gateway: Check token expiry

        alt Token Valid
            Gateway-&gt;&gt;Database: Fetch user properties
            Database--&gt;&gt;Gateway: Properties data
            Gateway--&gt;&gt;Mobile: Properties response
        else Token Expired
            Gateway--&gt;&gt;Mobile: 401 Unauthorized
            Mobile-&gt;&gt;Gateway: POST /api/v1/auth/refresh
```

```
            Gateway-&gt;&gt;AuthService: Refresh token validation
            AuthService-&gt;&gt;Redis: Verify refresh token
            Redis--&gt;&gt;AuthService: Token valid
            AuthService-&gt;&gt;AuthService: Generate new access token
            AuthService--&gt;&gt;Gateway: New access token
            Gateway--&gt;&gt;Mobile: Token refreshed
        end
    else Invalid Credentials
        AuthService--&gt;&gt;Gateway: Authentication failed
        Gateway--&gt;&gt;Mobile: 401 Unauthorized
    end
</div>
        </div>

This comprehensive integration architecture provides PropertyPro AI with

## 6.4 SECURITY ARCHITECTURE

### 6.4.1 AUTHENTICATION FRAMEWORK

#### 6.4.1.1 Identity Management System

PropertyPro AI implements a comprehensive identity management system buil

**Core Identity Components:**

| Component | Technology | Purpose | Security Features |
|---|---|---|---|
| User Registration | FastAPI + Pydantic | Account creation and validatic
| Authentication Service | JWT with bcrypt | User login and token generat
| Session Management | JWT Access/Refresh Tokens | Stateless session hanc
| Password Security | bcrypt with salt rounds | Secure password storage

**Identity Lifecycle Management:**

<div class="mermaid-wrapper" id="mermaid-diagram-lagx97237">
          <div class="mermaid">
graph TB
    subgraph &quot;User Registration Flow&quot;
        A[User Registration Request] --&gt; B[Email Validation]
        B --&gt; C[Password Strength Check]
        C --&gt; D[Account Creation]
        D --&gt; E[Email Verification]
```

```
        E --&gt; F[Account Activation]
    end

    subgraph &quot;Authentication Flow&quot;
        G[Login Request] --&gt; H[Credential Validation]
        H --&gt; I[bcrypt Password Verification]
        I --&gt; J[JWT Token Generation]
        J --&gt; K[Session Establishment]
    end

    subgraph &quot;Session Management&quot;
        L[Token Validation] --&gt; M{Token Valid?}
        M --&gt;|Yes| N[Grant Access]
        M --&gt;|No| O[Token Refresh]
        O --&gt; P{Refresh Valid?}
        P --&gt;|Yes| Q[Issue New Token]
        P --&gt;|No| R[Force Re-authentication]
    end

    F --&gt; G
    K --&gt; L
    Q --&gt; N
    R --&gt; G
</div>
        </div>

#### 6.4.1.2 Multi-Factor Authentication Strategy

While the current implementation focuses on secure password-based authen

**Current Authentication Factors:**

| Factor Type | Implementation | Security Level | Future Enhancement |
|---|---|---|---|
| Knowledge Factor | Password with bcrypt | High | Passkey support |
| Possession Factor | Mobile device session | Medium | SMS/TOTP integrat
| Inherence Factor | Not implemented | N/A | Biometric authentication |

**MFA Implementation Roadmap:**

- **Phase 1 (Current):** Secure password authentication with JWT tokens
- **Phase 2:** SMS-based second factor for high-value operations
- **Phase 3:** TOTP authenticator app integration
```

```
- **Phase 4:** Biometric authentication for mobile devices

#### 6.4.1.3 Session Management Architecture

The system implements secure session management with careful attention to

**JWT Token Configuration:**

| Token Type | Expiry Time | Storage Location | Security Measures |
|---|---|---|---|
| Access Token | 24 hours | React Native secure storage | Short expiry to
| Refresh Token | 30 days | Secure device storage | Automatic rotation on
| Session ID | 7 days | Application state | Encrypted transmission |

**Session Security Implementation:**
```

python

JWT Token Generation with Security Best Practices

from datetime import datetime, timedelta
from jose import JWTError, jwt
from passlib.context import CryptContext

class AuthenticationService:
def **init**(self):
self.pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
self.secret_key = settings.SECRET_KEY
self.algorithm = "HS256"
self.access_token_expire_minutes = 1440 # 24 hours
self.refresh_token_expire_days = 30

```
def create_access_token(self, data: dict) -> str:
    """Create JWT access token with secure expiration."""
```

```
        to_encode = data.copy()
        expire = datetime.utcnow() + timedelta(minutes=self.access_token_exp
        to_encode.update({"exp": expire, "type": "access"})

        return jwt.encode(to_encode, self.secret_key, algorithm=self.algorit

    def create_refresh_token(self, data: dict) -> str:
        """Create JWT refresh token with extended expiration."""
        to_encode = data.copy()
        expire = datetime.utcnow() + timedelta(days=self.refresh_token_expire
        to_encode.update({"exp": expire, "type": "refresh"})

        return jwt.encode(to_encode, self.secret_key, algorithm=self.algorit

    def verify_password(self, plain_password: str, hashed_password: str) -> b
        """Verify password using bcrypt."""
        return self.pwd_context.verify(plain_password, hashed_password)

    def get_password_hash(self, password: str) -> str:
        """Hash password using bcrypt with salt."""
        return self.pwd_context.hash(password)
```

```
#### 6.4.1.4 Token Handling and Validation

JWT validation ensures token structure, format, and content integrity, wh

**Token Validation Process:**

<div class="mermaid-wrapper" id="mermaid-diagram-cfvo76jq1">
          <div class="mermaid">
sequenceDiagram
    participant Client as React Native App
    participant API as FastAPI Backend
    participant Auth as Auth Service
    participant DB as PostgreSQL

    Client-&gt;&gt;API: Request with JWT Token
    API-&gt;&gt;Auth: Validate Token
    Auth-&gt;&gt;Auth: Check Token Structure
    Auth-&gt;&gt;Auth: Verify Signature
    Auth-&gt;&gt;Auth: Check Expiration
```

```
    alt Token Valid
        Auth-&gt;&gt;DB: Verify User Status
        DB--&gt;&gt;Auth: User Active
        Auth--&gt;&gt;API: Token Valid
        API--&gt;&gt;Client: Process Request
    else Token Expired
        Auth--&gt;&gt;API: Token Expired
        API--&gt;&gt;Client: 401 - Token Refresh Required
    else Token Invalid
        Auth--&gt;&gt;API: Invalid Token
        API--&gt;&gt;Client: 401 - Re-authentication Required
    end
</div>
        </div>

#### 6.4.1.5 Password Policies and Security

**Password Security Requirements:**

| Policy Element | Requirement | Implementation | Security Benefit |
|---|---|---|---|
| Minimum Length | 8 characters | Client and server validation | Brute fo
| Complexity | Mixed case, numbers, symbols | Regex validation | Dictiona
| History | Last 5 passwords | Database storage | Prevent password reuse
| Expiration | 90 days (optional) | Configurable policy | Limit exposure

**Password Storage Security:**
```

python

Secure Password Handling Implementation

from passlib.context import CryptContext
from passlib.hash import bcrypt

class PasswordManager:
def **init**(self):

# Configure bcrypt with appropriate rounds for security vs performance
self.pwd_context = CryptContext(
schemes=["bcrypt"],
deprecated="auto",
bcrypt__rounds=12 # Configurable based on security requirements
)

```
def hash_password(self, password: str) -> str:
    """Hash password with bcrypt and salt."""
    return self.pwd_context.hash(password)

def verify_password(self, plain_password: str, hashed_password: str) -> b
    """Verify password against stored hash."""
    return self.pwd_context.verify(plain_password, hashed_password)

def validate_password_strength(self, password: str) -> dict:
    """Validate password meets security requirements."""
    validation_result = {
        "valid": True,
        "errors": []
    }

    if len(password) < 8:
        validation_result["valid"] = False
        validation_result["errors"].append("Password must be at least 8 c

    if not re.search(r"[A-Z]", password):
        validation_result["valid"] = False
        validation_result["errors"].append("Password must contain upperca

    if not re.search(r"[a-z]", password):
        validation_result["valid"] = False
        validation_result["errors"].append("Password must contain lowerca

    if not re.search(r"\d", password):
        validation_result["valid"] = False
        validation_result["errors"].append("Password must contain number"

    return validation_result
```

```
### 6.4.2 AUTHORIZATION SYSTEM

#### 6.4.2.1 Role-Based Access Control (RBAC)

PropertyPro AI implements Role-Based Access Control (RBAC) as a policy-ne

**RBAC Architecture Components:**

| Component | Definition | Implementation | Security Impact |
|---|---|---|---|
| Users | Real estate professionals | User accounts with unique identifie
| Roles | Job function definitions | Predefined permission sets | Standa
| Permissions | Specific action allowances | Granular operation controls
| Resources | Protected system assets | Properties, clients, content, ana

**Role Hierarchy Definition:**

<div class="mermaid-wrapper" id="mermaid-diagram-jg7oz5frn">
          <div class="mermaid">
graph TB
    subgraph &quot;PropertyPro AI Role Hierarchy&quot;
        ADMIN[System Administrator&lt;br/&gt;Full system access]
        BROKER[Broker&lt;br/&gt;Team management + Agent permissions]
        AGENT[Real Estate Agent&lt;br/&gt;Core business operations]
        VIEWER[Viewer&lt;br/&gt;Read-only access]

        ADMIN --&gt; BROKER
        BROKER --&gt; AGENT
        AGENT --&gt; VIEWER
    end

    subgraph &quot;Permission Categories&quot;
        PROP[Property Management&lt;br/&gt;CRUD operations]
        CLIENT[Client Management&lt;br/&gt;CRM operations]
        CONTENT[Content Generation&lt;br/&gt;AI services]
        ANALYTICS[Analytics Access&lt;br/&gt;Performance data]
        ADMIN_FUNC[Administrative Functions&lt;br/&gt;User management]
    end

    ADMIN -.-&gt; ADMIN_FUNC
    ADMIN -.-&gt; ANALYTICS
    BROKER -.-&gt; ANALYTICS
    AGENT -.-&gt; PROP
```

```
    AGENT -.-&gt; CLIENT
    AGENT -.-&gt; CONTENT
    VIEWER -.-&gt; ANALYTICS
</div>
        </div>

#### 6.4.2.2 Permission Management Framework

**Granular Permission Structure:**

| Permission Category | Specific Permissions | Role Assignment | Resource
|---|---|---|---|
| Property Operations | create_property, read_property, update_property,
| Client Management | create_client, read_client, update_client, delete_c
| Content Generation | generate_content, edit_content, publish_content |
| Analytics Access | view_analytics, export_reports | Agent, Broker, Admi

**Permission Enforcement Implementation:**
```

python

RBAC Permission System Implementation

from enum import Enum
from typing import List, Set
from fastapi import Depends, HTTPException, status

class Permission(Enum):
# Property permissions
CREATE_PROPERTY = "create_property"
READ_PROPERTY = "read_property"
UPDATE_PROPERTY = "update_property"
DELETE_PROPERTY = "delete_property"

```
# Client permissions
CREATE_CLIENT = "create_client"
READ_CLIENT = "read_client"
UPDATE_CLIENT = "update_client"
DELETE_CLIENT = "delete_client"

# Content permissions
GENERATE_CONTENT = "generate_content"
EDIT_CONTENT = "edit_content"
PUBLISH_CONTENT = "publish_content"

# Analytics permissions
VIEW_ANALYTICS = "view_analytics"
EXPORT_REPORTS = "export_reports"

# Administrative permissions
MANAGE_USERS = "manage_users"
SYSTEM_CONFIG = "system_config"
```

class Role(Enum):
ADMIN = "admin"
BROKER = "broker"
AGENT = "agent"
VIEWER = "viewer"

class RBACManager:
def **init**(self):
self.role_permissions = {
Role.ADMIN: {
Permission.CREATE_PROPERTY, Permission.READ_PROPERTY,
Permission.UPDATE_PROPERTY, Permission.DELETE_PROPERTY,
Permission.CREATE_CLIENT, Permission.READ_CLIENT,
Permission.UPDATE_CLIENT, Permission.DELETE_CLIENT,
Permission.GENERATE_CONTENT, Permission.EDIT_CONTENT,
Permission.PUBLISH_CONTENT, Permission.VIEW_ANALYTICS,
Permission.EXPORT_REPORTS, Permission.MANAGE_USERS,
Permission.SYSTEM_CONFIG

},
Role.BROKER: {
Permission.CREATE_PROPERTY, Permission.READ_PROPERTY,
Permission.UPDATE_PROPERTY, Permission.DELETE_PROPERTY,
Permission.CREATE_CLIENT, Permission.READ_CLIENT,
Permission.UPDATE_CLIENT, Permission.DELETE_CLIENT,
Permission.GENERATE_CONTENT, Permission.EDIT_CONTENT,
Permission.PUBLISH_CONTENT, Permission.VIEW_ANALYTICS,
Permission.EXPORT_REPORTS
},
Role.AGENT: {
Permission.CREATE_PROPERTY, Permission.READ_PROPERTY,
Permission.UPDATE_PROPERTY, Permission.DELETE_PROPERTY,
Permission.CREATE_CLIENT, Permission.READ_CLIENT,
Permission.UPDATE_CLIENT, Permission.DELETE_CLIENT,
Permission.GENERATE_CONTENT, Permission.EDIT_CONTENT,
Permission.PUBLISH_CONTENT, Permission.VIEW_ANALYTICS
},
Role.VIEWER: {
Permission.READ_PROPERTY, Permission.READ_CLIENT,
Permission.VIEW_ANALYTICS
}
}

```
def has_permission(self, user_role: Role, required_permission: Permissio
    """Check if user role has required permission."""
    return required_permission in self.role_permissions.get(user_role, se

def get_user_permissions(self, user_role: Role) -> Set[Permission]:
    """Get all permissions for a user role."""
    return self.role_permissions.get(user_role, set())
```

FastAPI Dependency for Permission Checking

def require_permission(required_permission: Permission):
"""Dependency factory for permission-based access control."""
def permission_checker(current_user: User = Depends(get_current_user)):
rbac_manager = RBACManager()
user_role = Role(current_user.role)

```
    if not rbac_manager.has_permission(user_role, required_permission):
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail=f"Insufficient permissions. Required: {required_permis
        )

    return current_user

return permission_checker
```

```
#### 6.4.2.3 Resource Authorization Patterns

**Resource-Level Access Control:**

PropertyPro AI implements resource-level authorization ensuring users car

| Resource Type | Access Pattern | Ownership Validation | Security Bounda
|---|---|---|---|
| Properties | User-owned only | user_id matching | Individual agent prop
| Clients | User-managed only | user_id matching | Individual agent clie
| AI Content | User-generated only | user_id matching | Individual agent
| Analytics | User-specific only | user_id matching | Individual agent me

**Resource Authorization Implementation:**
```

python

Resource-Level Authorization

from sqlalchemy.orm import Session
from fastapi import Depends, HTTPException, status

class ResourceAuthorizationService:
def **init**(self, db_session: Session):
self.db_session = db_session

```
async def authorize_property_access(
    self,
    property_id: UUID,
    user_id: UUID,
    required_permission: Permission
) -> bool:
    """Authorize user access to specific property."""

    # Check if property exists and belongs to user
    property_query = select(Property).where(
        and_(Property.id == property_id, Property.user_id == user_id)
    )
    property_result = await self.db_session.execute(property_query)
    property_obj = property_result.scalar_one_or_none()

    if not property_obj:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Property not found or access denied"
        )

    return True

async def authorize_client_access(
    self,
    client_id: UUID,
    user_id: UUID,
    required_permission: Permission
) -> bool:
    """Authorize user access to specific client."""

    # Check if client exists and belongs to user
    client_query = select(Client).where(
        and_(Client.id == client_id, Client.user_id == user_id)
```

```
    )
    client_result = await self.db_session.execute(client_query)
    client_obj = client_result.scalar_one_or_none()

    if not client_obj:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Client not found or access denied"
        )

    return True
```

Usage in API Endpoints

@router.get("/properties/{property_id}")
async def get_property(
property_id: UUID,
current_user: User =
Depends(require_permission(Permission.READ_PROPERTY)),
db_session: AsyncSession = Depends(get_db_session)
):
"""Get property with authorization check."""

```
auth_service = ResourceAuthorizationService(db_session)
await auth_service.authorize_property_access(
    property_id,
    current_user.id,
    Permission.READ_PROPERTY
)
```

Proceed with property retrieval

```
return await get_property_by_id(property_id, db_session)
```

```
#### 6.4.2.4 Policy Enforcement Points
```

```
**Centralized Policy Enforcement:**

| Enforcement Point | Location | Scope | Implementation |
|---|---|---|---|
| API Gateway | FastAPI middleware | All HTTP requests | JWT validation a
| Endpoint Level | Route decorators | Specific operations | Permission-ba
| Resource Level | Service layer | Individual resources | Ownership and a
| Data Layer | Database queries | Data access | Row-level security filte

#### 6.4.2.5 Audit Logging Framework

**Comprehensive Audit Trail:**
```

python

Security Audit Logging System

from datetime import datetime
from enum import Enum
import json

class AuditEventType(Enum):
LOGIN_SUCCESS = "login_success"
LOGIN_FAILURE = "login_failure"
LOGOUT = "logout"
PERMISSION_DENIED = "permission_denied"
RESOURCE_ACCESS = "resource_access"
DATA_MODIFICATION = "data_modification"
ADMIN_ACTION = "admin_action"

class SecurityAuditLogger:
def **init**(self, db_session: AsyncSession):
self.db_session = db_session

```python
async def log_security_event(
    self,
    event_type: AuditEventType,
    user_id: Optional[UUID],
    resource_type: Optional[str],
    resource_id: Optional[UUID],
    details: Optional[dict],
    ip_address: Optional[str],
    user_agent: Optional[str]
):
    """Log security-related events for audit trail."""

    audit_entry = SecurityAuditLog(
        event_type=event_type.value,
        user_id=user_id,
        resource_type=resource_type,
        resource_id=resource_id,
        details=json.dumps(details) if details else None,
        ip_address=ip_address,
        user_agent=user_agent,
        timestamp=datetime.utcnow()
    )

    self.db_session.add(audit_entry)
    await self.db_session.commit()

async def log_authentication_event(
    self,
    event_type: AuditEventType,
    email: str,
    success: bool,
    ip_address: str,
    user_agent: str,
    failure_reason: Optional[str] = None
):
    """Log authentication events."""

    details = {
        "email": email,
        "success": success,
        "failure_reason": failure_reason
    }
```

```
    await self.log_security_event(
        event_type=event_type,
        user_id=None,
        resource_type="authentication",
        resource_id=None,
        details=details,
        ip_address=ip_address,
        user_agent=user_agent
    )
```

```
### 6.4.3 DATA PROTECTION

#### 6.4.3.1 Encryption Standards Implementation

PropertyPro AI implements AES-256 encryption as the primary encryption st

**Encryption Architecture:**

| Data Category | Encryption Method | Key Length | Implementation |
|---|---|---|---|
| Database Records | AES-256-GCM | 256-bit | Industry-standard AES-256 fo
| API Communications | TLS 1.3 | 256-bit | HTTPS transport encryption |
| File Storage | AES-256-CBC | 256-bit | Property image encryption |
| JWT Tokens | HMAC-SHA256 | 256-bit | Token signature security |

**Encryption Implementation:**
```

python

AES-256 Encryption Service Implementation

from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,

modes
import os
import base64

class EncryptionService:
def **init**(self, master_key: bytes):
self.master_key = master_key
self.fernet = Fernet(master_key)

```
@classmethod
def generate_key(cls) -> bytes:
    """Generate a new AES-256 encryption key."""
    return Fernet.generate_key()

@classmethod
def derive_key_from_password(cls, password: str, salt: bytes) -> bytes:
    """Derive encryption key from password using PBKDF2."""
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,  # 256 bits
        salt=salt,
        iterations=100000,  # NIST recommended minimum
    )
    key = base64.urlsafe_b64encode(kdf.derive(password.encode()))
    return key

def encrypt_data(self, plaintext: str) -> str:
    """Encrypt data using AES-256."""
    encrypted_data = self.fernet.encrypt(plaintext.encode())
    return base64.urlsafe_b64encode(encrypted_data).decode()

def decrypt_data(self, encrypted_data: str) -> str:
    """Decrypt data using AES-256."""
    encrypted_bytes = base64.urlsafe_b64decode(encrypted_data.encode())
    decrypted_data = self.fernet.decrypt(encrypted_bytes)
    return decrypted_data.decode()

def encrypt_file(self, file_path: str, output_path: str) -> None:
    """Encrypt file using AES-256-CBC mode."""
    # Generate random IV for each file
    iv = os.urandom(16)
```

```
    # Create cipher
    cipher = Cipher(
        algorithms.AES(self.master_key[:32]),  # Use first 32 bytes for A
        modes.CBC(iv)
    )
    encryptor = cipher.encryptor()

    with open(file_path, 'rb') as infile, open(output_path, 'wb') as out
        # Write IV to beginning of encrypted file
        outfile.write(iv)

        # Encrypt file in chunks
        while True:
            chunk = infile.read(8192)
            if len(chunk) == 0:
                break
            elif len(chunk) % 16 != 0:
                # Pad last chunk to 16 bytes
                chunk += b' ' * (16 - len(chunk) % 16)

            encrypted_chunk = encryptor.update(chunk)
            outfile.write(encrypted_chunk)

        # Finalize encryption
        outfile.write(encryptor.finalize())
```

```
#### 6.4.3.2 Key Management System

Proper key management is crucial for encryption security, involving crea

**Key Management Architecture:**

| Key Type | Storage Method | Rotation Policy | Access Control |
|---|---|---|---|
| Master Keys | Environment variables | Annual rotation | System adminis
| Data Encryption Keys | Database (encrypted) | Quarterly rotation | App
| JWT Signing Keys | Secure configuration | Monthly rotation | Authentica
| File Encryption Keys | Key derivation | Per-file generation | Resource

**Key Management Implementation:**
```

python

Comprehensive Key Management System

from datetime import datetime, timedelta
from typing import Dict, Optional
import secrets
import hashlib

class KeyManagementService:
def **init**(self, master_key: bytes):
self.master_key = master_key
self.key_cache: Dict[str, dict] = {}
self.key_rotation_days = 90 # Quarterly rotation

```
def generate_data_encryption_key(self, purpose: str) -> dict:
    """Generate new data encryption key with metadata."""
    key_data = {
        "key": Fernet.generate_key(),
        "purpose": purpose,
        "created_at": datetime.utcnow(),
        "expires_at": datetime.utcnow() + timedelta(days=self.key_rotati
        "version": 1,
        "active": True
    }

    # Store encrypted key in database
    key_id = self._store_encrypted_key(key_data)
    key_data["key_id"] = key_id

    return key_data

def get_encryption_key(self, key_id: str) -> Optional[bytes]:
    """Retrieve encryption key by ID."""
    # Check cache first
```

```
        if key_id in self.key_cache:
            cached_key = self.key_cache[key_id]
            if cached_key["expires_at"] > datetime.utcnow():
                return cached_key["key"]

        # Retrieve from database
        key_data = self._retrieve_encrypted_key(key_id)
        if key_data and key_data["active"]:
            # Cache for performance
            self.key_cache[key_id] = key_data
            return key_data["key"]

        return None

    def rotate_key(self, old_key_id: str) -> str:
        """Rotate encryption key and return new key ID."""
        old_key_data = self._retrieve_encrypted_key(old_key_id)
        if not old_key_data:
            raise ValueError(f"Key {old_key_id} not found")

        # Generate new key with same purpose
        new_key_data = self.generate_data_encryption_key(old_key_data["purpos

        # Mark old key as inactive but keep for decryption
        old_key_data["active"] = False
        old_key_data["rotated_at"] = datetime.utcnow()
        self._update_key_metadata(old_key_id, old_key_data)

        return new_key_data["key_id"]

    def _store_encrypted_key(self, key_data: dict) -> str:
        """Store encryption key in database (encrypted with master key)."""
        # Encrypt the key with master key
        key_fernet = Fernet(self.master_key)
        encrypted_key = key_fernet.encrypt(key_data["key"])

        # Generate unique key ID
        key_id = hashlib.sha256(
            f"{key_data['purpose']}{key_data['created_at']}".encode()
        ).hexdigest()[:16]

        # Store in database (implementation depends on your database schema)
        # This would typically involve inserting into an encryption_keys tabl
```

```
        return key_id

    def check_key_expiration(self) -> List[str]:
        """Check for keys approaching expiration."""
        expiring_keys = []
        warning_threshold = datetime.utcnow() + timedelta(days=30)

        # Query database for keys expiring within 30 days
        # Implementation depends on your database schema

        return expiring_keys
```

```
#### 6.4.3.3 Data Masking and Anonymization

**Data Protection Strategies:**

| Data Type | Protection Method | Use Case | Implementation |
|---|---|---|---|
| Email Addresses | Partial masking | Logs and analytics | Show first 3 c
| Phone Numbers | Format masking | Display purposes | Show area code + la
| Client Names | Pseudonymization | Analytics processing | Consistent has
| Property Addresses | Zip code only | Market analysis | Remove street-le

**Data Masking Implementation:**
```

python

Data Masking and Anonymization Service

import hashlib
import re
from typing import Optional

class DataMaskingService:
def **init**(self, salt: str):
self.salt = salt

```
def mask_email(self, email: str) -> str:
    """Mask email address for display purposes."""
    if not email or '@' not in email:
        return "***@***.***"

    local, domain = email.split('@', 1)
    if len(local) <= 3:
        masked_local = '*' * len(local)
    else:
        masked_local = local[:3] + '*' * (len(local) - 3)

    return f"{masked_local}@{domain}"

def mask_phone(self, phone: str) -> str:
    """Mask phone number showing area code and last 4 digits."""
    # Remove all non-digit characters
    digits = re.sub(r'\D', '', phone)

    if len(digits) == 10:
        return f"({digits[:3]}) ***-{digits[-4:]}"
    elif len(digits) == 11 and digits[0] == '1':
        return f"+1 ({digits[1:4]}) ***-{digits[-4:]}"
    else:
        return "***-***-****"

def pseudonymize_name(self, name: str) -> str:
    """Create consistent pseudonym for name."""
    # Create consistent hash-based pseudonym
    hash_input = f"{name.lower()}{self.salt}"
    hash_digest = hashlib.sha256(hash_input.encode()).hexdigest()

    # Generate pronounceable pseudonym from hash
    consonants = "bcdfghjklmnpqrstvwxyz"
    vowels = "aeiou"

    pseudonym = ""
    for i in range(0, min(8, len(hash_digest)), 2):
        consonant_idx = int(hash_digest[i], 16) % len(consonants)
```

```
            vowel_idx = int(hash_digest[i+1], 16) % len(vowels)
            pseudonym += consonants[consonant_idx] + vowels[vowel_idx]

        return pseudonym.capitalize()

    def anonymize_address(self, address: str) -> str:
        """Anonymize address to zip code level."""
        # Extract zip code using regex
        zip_match = re.search(r'\b\d{5}(-\d{4})?\b', address)
        if zip_match:
            return f"*****, {zip_match.group()}"
        else:
            return "*****"

    def create_analytics_record(self, user_data: dict) -> dict:
        """Create anonymized record for analytics."""
        return {
            "user_hash": self.pseudonymize_name(user_data.get("name", "")),
            "location_zip": self.anonymize_address(user_data.get("address", '
            "registration_month": user_data.get("created_at", "").strftime("%
            "activity_level": user_data.get("activity_level", "unknown"),
            "property_count": user_data.get("property_count", 0)
        }
```

```
#### 6.4.3.4 Secure Communication Protocols

**Communication Security Matrix:**

| Communication Type | Protocol | Encryption | Authentication |
|---|---|---|---|
| Client-Server API | HTTPS/TLS 1.3 | AES-256-GCM | JWT Bearer tokens |
| Database Connections | TLS encrypted | AES-256 | Certificate-based |
| File Transfers | HTTPS with encryption | AES-256-CBC | API key authenti
| Internal Services | mTLS | AES-256-GCM | Mutual certificate auth |

#### 6.4.3.5 Compliance Controls Framework

**Regulatory Compliance Matrix:**

| Regulation | Applicable Requirements | Implementation | Monitoring |
|---|---|---|---|
| GDPR | Data protection, right to deletion | Encryption, anonymization,
```

```
| CCPA | Consumer privacy rights | Data masking, access controls | Priva
| SOC 2 | Security controls | Encryption, access logging | Annual audits
| Real Estate Regulations | Client data protection | Secure storage, acce

### 6.4.4 SECURITY ARCHITECTURE DIAGRAMS

#### 6.4.4.1 Authentication Flow Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-zpr323abm">
          <div class="mermaid">
sequenceDiagram
    participant User as Real Estate Agent
    participant App as React Native App
    participant API as FastAPI Gateway
    participant Auth as Auth Service
    participant DB as PostgreSQL
    participant Audit as Audit Logger

    User-&gt;&gt;App: Enter Credentials
    App-&gt;&gt;API: POST /auth/login
    API-&gt;&gt;Auth: Validate Credentials
    Auth-&gt;&gt;DB: Query User Record
    DB--&gt;&gt;Auth: User Data
    Auth-&gt;&gt;Auth: Verify Password (bcrypt)

    alt Authentication Success
        Auth-&gt;&gt;Auth: Generate JWT Tokens
        Auth-&gt;&gt;DB: Update Last Login
        Auth-&gt;&gt;Audit: Log Success Event
        Auth--&gt;&gt;API: Access + Refresh Tokens
        API--&gt;&gt;App: Authentication Success
        App-&gt;&gt;App: Store Tokens Securely
        App--&gt;&gt;User: Dashboard Access
    else Authentication Failure
        Auth-&gt;&gt;Audit: Log Failure Event
        Auth--&gt;&gt;API: Authentication Error
        API--&gt;&gt;App: 401 Unauthorized
        App--&gt;&gt;User: Login Error Message
    end
</div>
        </div>

#### 6.4.4.2 Authorization Flow Architecture
```

```
<div class="mermaid-wrapper" id="mermaid-diagram-xanbthr6q">
          <div class="mermaid">
flowchart TD
    A[API Request with JWT] --&gt; B[Extract JWT Token]
    B --&gt; C{Token Valid?}

    C --&gt;|No| D[Return 401 Unauthorized]
    C --&gt;|Yes| E[Extract User Claims]

    E --&gt; F[Identify User Role]
    F --&gt; G[Check Required Permission]
    G --&gt; H{Permission Granted?}

    H --&gt;|No| I[Log Access Denied]
    H --&gt;|Yes| J[Check Resource Ownership]

    I --&gt; K[Return 403 Forbidden]
    J --&gt; L{Resource Access Allowed?}

    L --&gt;|No| M[Log Unauthorized Access]
    L --&gt;|Yes| N[Log Successful Access]

    M --&gt; K
    N --&gt; O[Process Request]
    O --&gt; P[Return Response]

    style A fill:#e1f5fe
    style P fill:#c8e6c9
    style D fill:#ffcdd2
    style K fill:#ffcdd2
</div>
        </div>

#### 6.4.4.3 Security Zone Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-t70598bld">
          <div class="mermaid">
graph TB
    subgraph &quot;Internet Zone&quot;
        INTERNET[Internet Users]
        MOBILE[Mobile Devices]
    end
```

```
    subgraph &quot;DMZ - Demilitarized Zone&quot;
        LB[&quot;Load Balancer&lt;br/&gt;TLS Termination&quot;]
        WAF[&quot;Web Application Firewall&lt;br/&gt;DDoS Protection&quo
        API_GW[&quot;API Gateway&lt;br/&gt;Rate Limiting&quot;]
    end

    subgraph &quot;Application Zone&quot;
        AUTH[&quot;Authentication Service&lt;br/&gt;JWT Management&quot;
        API[&quot;FastAPI Application&lt;br/&gt;Business Logic&quot;]
        AI[&quot;AI Service Integration&lt;br/&gt;OpenAI GPT-4.1&quot;]
    end

    subgraph &quot;Data Zone&quot;
        DB[#40;PostgreSQL Database&lt;br/&gt;Encrypted Storage#41;]
        FILES[&quot;File Storage&lt;br/&gt;Encrypted Files&quot;]
        KEYS[&quot;Key Management&lt;br/&gt;Encryption Keys&quot;]
    end

    subgraph &quot;Management Zone&quot;
        MONITOR[&quot;Security Monitoring&lt;br/&gt;Audit Logs&quot;]
        BACKUP[&quot;Backup Systems&lt;br/&gt;Encrypted Backups&quot;]
        ADMIN[&quot;Admin Console&lt;br/&gt;Restricted Access&quot;]
    end

    INTERNET --&gt; LB
    MOBILE --&gt; LB
    LB --&gt; WAF
    WAF --&gt; API_GW

    API_GW --&gt; AUTH
    API_GW --&gt; API
    API --&gt; AI

    AUTH --&gt; DB
    API --&gt; DB
    API --&gt; FILES
    AUTH --&gt; KEYS

    API --&gt; MONITOR
    AUTH --&gt; MONITOR
    DB --&gt; BACKUP
    ADMIN --&gt; MONITOR
```

```
    style INTERNET fill:#ffebee
</div>
        </div>

#### 6.4.4.4 Data Encryption Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-dk8uc257v">
          <div class="mermaid">
graph TB
    subgraph DataSources[&quot;Data Sources&quot;]
        USER_DATA[User Input Data]
        PROPERTY_DATA[Property Information]
        CLIENT_DATA[Client Records]
        AI_CONTENT[AI Generated Content]
    end

    subgraph EncryptionLayer[&quot;Encryption Layer&quot;]
        FIELD_ENC[&quot;Field-Level Encryption&lt;br/&gt;AES-256-GCM&quot
        FILE_ENC[&quot;File Encryption&lt;br/&gt;AES-256-CBC&quot;]
        TRANSPORT_ENC[&quot;Transport Encryption&lt;br/&gt;TLS 1.3&quot;
    end

    subgraph KeyManagement[&quot;Key Management&quot;]
        MASTER_KEY[&quot;Master Key&lt;br/&gt;Environment Variable&quot;
        DEK[&quot;Data Encryption Keys&lt;br/&gt;Database Stored&quot;]
        KEK[&quot;Key Encryption Keys&lt;br/&gt;Derived Keys&quot;]
    end

    subgraph EncryptedStorage[&quot;Encrypted Storage&quot;]
        DB_ENCRYPTED[#40;Encrypted Database&lt;br/&gt;PostgreSQL#41;]
        FILE_ENCRYPTED[&quot;Encrypted Files&lt;br/&gt;Property Images&qu
        BACKUP_ENCRYPTED[&quot;Encrypted Backups&lt;br/&gt;Daily Snapsho
    end

    USER_DATA --&gt; FIELD_ENC
    PROPERTY_DATA --&gt; FIELD_ENC
    CLIENT_DATA --&gt; FIELD_ENC
    AI_CONTENT --&gt; FILE_ENC

    FIELD_ENC --&gt; TRANSPORT_ENC
    FILE_ENC --&gt; TRANSPORT_ENC
```

```
    MASTER_KEY --&gt; KEK
    KEK --&gt; DEK
    DEK --&gt; FIELD_ENC
    DEK --&gt; FILE_ENC

    TRANSPORT_ENC --&gt; DB_ENCRYPTED
    TRANSPORT_ENC --&gt; FILE_ENCRYPTED
    DB_ENCRYPTED --&gt; BACKUP_ENCRYPTED

    style DataSources fill:#e1f5fe
    style EncryptionLayer fill:#fff3e0
    style KeyManagement fill:#f3e5f5
    style EncryptedStorage fill:#e8f5e8
</div>
        </div>

### 6.4.5 SECURITY CONTROL MATRIX

#### 6.4.5.1 Comprehensive Security Controls

| Control Category | Control Name | Implementation | Risk Mitigation | Co
|---|---|---|---|---|
| Authentication | Multi-factor Authentication | JWT + Device binding | A
| Authorization | Role-based Access Control | RBAC with resource ownershi
| Data Protection | AES-256 Encryption | Database and file encryption | D
| Communication | TLS 1.3 Transport Security | HTTPS for all communicatio

#### 6.4.5.2 Security Monitoring and Incident Response

**Security Event Monitoring:**

| Event Type | Detection Method | Response Action | Escalation Criteria
|---|---|---|---|
| Failed Login Attempts | Real-time monitoring | Account lockout after 5
| Permission Violations | Access control logs | Immediate alert to securi
| Data Access Anomalies | Behavioral analysis | User notification and aud
| API Rate Limit Violations | Gateway monitoring | Temporary IP blocking

PropertyPro AI's security architecture provides comprehensive protection

## 6.5 MONITORING AND OBSERVABILITY

### 6.5.1 MONITORING INFRASTRUCTURE
```

```
#### 6.5.1.1 System Monitoring ApproachPropertyPro AI implements a compre

### 6.5.1 MONITORING INFRASTRUCTURE

#### 6.5.1.1 Metrics Collection Architecture

PropertyPro AI implements a multi-tier metrics collection system designed

**Backend Metrics Collection:**

| Metric Category | Collection Method | Storage | Retention Period |
|---|---|---|---|
| API Performance | prometheus-fastapi-instrumentator library | Promethe
| Business Metrics | Custom Prometheus counters | Prometheus | 90 days |
| System Metrics | Container resource monitoring | Prometheus | 7 days |
| AI Service Metrics | OpenAI API usage tracking | PostgreSQL | 1 year |

**Mobile Application Metrics:**

| Metric Type | Collection Method | Purpose | Frequency |
|---|---|---|---|
| Performance Metrics | React Native Perf Monitor | Frame rate and respo
| User Interaction | Custom event tracking | User behavior analysis | Pe
| Network Performance | HTTP request monitoring | API response times | Pe
| Crash Reporting | Native crash detection with ML analysis | Applicatio

#### 6.5.1.2 Monitoring Architecture Diagram

<div class="mermaid-wrapper" id="mermaid-diagram-2grbafo1g">
          <div class="mermaid">
graph TB
    subgraph &quot;Mobile Applications&quot;
        RN[React Native App&lt;br/&gt;Performance Monitoring]
        METRICS[Custom Metrics&lt;br/&gt;Collection]
        CRASHES[Crash Reporting&lt;br/&gt;System]
    end

    subgraph &quot;Backend Services&quot;
        API[FastAPI Application&lt;br/&gt;Instrumented Endpoints]
        PROM_INST[Prometheus&lt;br/&gt;Instrumentator]
        CUSTOM[Custom Business&lt;br/&gt;Metrics]
    end
```

```
    subgraph &quot;Metrics Storage&quot;
        PROMETHEUS[#40;Prometheus&lt;br/&gt;Time Series DB#41;]
        POSTGRES[#40;PostgreSQL&lt;br/&gt;Business Metrics#41;]
    end

    subgraph &quot;Visualization Layer&quot;
        GRAFANA[Grafana Dashboards&lt;br/&gt;Real-time Monitoring]
        ALERTS[Alert Manager&lt;br/&gt;Notification System]
    end

    subgraph &quot;Log Aggregation&quot;
        LOGS[Structured Logging&lt;br/&gt;JSON Format]
        LOG_STORAGE[Log Storage&lt;br/&gt;File System]
    end

    RN --&gt; METRICS
    RN --&gt; CRASHES
    METRICS --&gt; PROMETHEUS
    CRASHES --&gt; POSTGRES

    API --&gt; PROM_INST
    API --&gt; CUSTOM
    PROM_INST --&gt; PROMETHEUS
    CUSTOM --&gt; PROMETHEUS

    PROMETHEUS --&gt; GRAFANA
    PROMETHEUS --&gt; ALERTS

    API --&gt; LOGS
    LOGS --&gt; LOG_STORAGE

    style RN fill:#e1f5fe
    style PROMETHEUS fill:#fff3e0
    style GRAFANA fill:#e8f5e8
</div>
        </div>

#### 6.5.1.3 Log Aggregation Strategy

**Structured Logging Implementation:**

PropertyPro AI implements structured JSON logging across all system compo
```

```
| Log Level | Use Case | Retention | Format |
|---|---|---|---|
| DEBUG | Development debugging | 1 day | JSON with trace IDs |
| INFO | Business operations | 7 days | JSON with user context |
| WARNING | Performance degradation | 30 days | JSON with metrics |
| ERROR | System failures | 90 days | JSON with stack traces |

**Log Structure Example:**
```

json

{
"timestamp": "2024-01-15T14:30:00Z",
"level": "INFO",
"service": "property-service",
"user_id": "uuid-123",
"request_id": "req-456",
"message": "Property created successfully",
"duration_ms": 245,
"metadata": {
"property_id": "prop-789",
"ai_content_generated": true
}
}

```
#### 6.5.1.4 Alert Management System

**Alert Configuration Matrix:**

| Alert Type | Threshold | Severity | Response Time | Escalation |
|---|---|---|---|---|
| API Response Time | > 5 seconds | High | 5 minutes | Development team
| Error Rate | > 5% | Critical | 2 minutes | On-call engineer |
| Mobile Crash Rate | > 1% | High | 10 minutes | Mobile team |
| AI Service Failure | > 3 failures | Medium | 15 minutes | AI team |

### 6.5.2 OBSERVABILITY PATTERNS
```

```
#### 6.5.2.1 Health Check Implementation

**Comprehensive Health Monitoring:**

PropertyPro AI implements multi-level health checks to ensure system rel

| Health Check Level | Endpoint | Frequency | Dependencies |
|---|---|---|---|
| Basic Health | `/health` | 30 seconds | Application startup |
| Database Health | `/health/db` | 60 seconds | PostgreSQL connection |
| AI Service Health | `/health/ai` | 120 seconds | OpenAI API availabili
| External Services | `/health/external` | 300 seconds | Email, storage s

**Health Check Response Format:**
```

json

{
"status": "healthy",
"timestamp": "2024-01-15T14:30:00Z",
"version": "1.0.0",
"checks": {
"database": {
"status": "healthy",
"response_time_ms": 45
},
"ai_service": {
"status": "healthy",
"response_time_ms": 1200
},
"external_services": {
"status": "degraded",
"details": "Email service slow response"
}
}
}

```
#### 6.5.2.2 Performance Metrics Dashboard

**Key Performance Indicators:**

| Metric Category | Primary Metrics | Target Values | Alert Thresholds |
|---|---|---|---|
| API Performance | Response time, throughput | < 2s, > 100 req/min | > 5
| Mobile Performance | 60 FPS frame rate | 60 FPS | < 30 FPS |
| Business Metrics | Property listings, client interactions | Baseline +
| AI Performance | Content generation time | < 5 seconds | > 10 seconds

#### 6.5.2.3 Business Metrics Tracking

**Real Estate Specific Metrics:**

| Business Metric | Measurement Method | Business Impact | Monitoring Fre
|---|---|---|---|
| Property Listing Success Rate | API success/failure ratio | Revenue gen
| AI Content Quality Score | User feedback ratings | User satisfaction |
| Client Engagement Rate | Interaction frequency | Relationship quality
| Lead Conversion Rate | Pipeline progression | Business growth | Monthly

#### 6.5.2.4 SLA Monitoring Framework

**Service Level Agreements:**

| Service Component | Availability SLA | Performance SLA | Error Rate SLA
|---|---|---|---|
| Mobile Application | 99.5% | 60 FPS performance | < 1% crash rate |
| Backend API | 99.9% | < 2 second response | < 0.5% error rate |
| AI Content Generation | 99.0% | < 5 second generation | < 2% failure ra
| Data Persistence | 99.95% | < 100ms query time | < 0.1% data loss |

### 6.5.3 INCIDENT RESPONSE

#### 6.5.3.1 Alert Flow Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-ux8rzqy1j">
          <div class="mermaid">
flowchart TD
    A[Monitoring System&lt;br/&gt;Detects Issue] --&gt; B{Severity Level]

    B --&gt;|Critical| C[Immediate Alert&lt;br/&gt;&lt; 2 minutes]
```

```
    B --&gt;|High| D[Priority Alert&lt;br/&gt;&lt; 5 minutes]
    B --&gt;|Medium| E[Standard Alert&lt;br/&gt;&lt; 15 minutes]
    B --&gt;|Low| F[Batch Alert&lt;br/&gt;&lt; 60 minutes]

    C --&gt; G[On-Call Engineer&lt;br/&gt;Notification]
    D --&gt; H[Development Team&lt;br/&gt;Notification]
    E --&gt; I[Team Lead&lt;br/&gt;Notification]
    F --&gt; J[Daily Summary&lt;br/&gt;Report]

    G --&gt; K{Issue Resolved&lt;br/&gt;in 15 minutes?}
    H --&gt; L{Issue Resolved&lt;br/&gt;in 30 minutes?}
    I --&gt; M{Issue Resolved&lt;br/&gt;in 2 hours?}

    K --&gt;|No| N[Escalate to&lt;br/&gt;Senior Engineer]
    L --&gt;|No| O[Escalate to&lt;br/&gt;Team Lead]
    M --&gt;|No| P[Escalate to&lt;br/&gt;Management]

    K --&gt;|Yes| Q[Close Incident]
    L --&gt;|Yes| Q
    M --&gt;|Yes| Q

    N --&gt; R[Emergency Response&lt;br/&gt;Protocol]
    O --&gt; S[Priority Response&lt;br/&gt;Protocol]
    P --&gt; T[Management Response&lt;br/&gt;Protocol]

    R --&gt; Q
    S --&gt; Q
    T --&gt; Q

    Q --&gt; U[Post-Incident&lt;br/&gt;Review]

    style A fill:#e1f5fe
    style C fill:#ffcdd2
    style G fill:#fff3e0
    style Q fill:#c8e6c9
</div>
        </div>

#### 6.5.3.2 Escalation Procedures

**Incident Response Team Structure:**

| Role | Responsibility | Response Time | Contact Method |
```

```
|---|---|---|---|
| On-Call Engineer | First response to critical alerts | 5 minutes | Pho
| Team Lead | Coordination and resource allocation | 15 minutes | Phone,
| Senior Engineer | Technical expertise and guidance | 30 minutes | Phon
| Product Manager | Business impact assessment | 60 minutes | Email, Sla

#### 6.5.3.3 Runbook Documentation

**Standard Operating Procedures:**

| Incident Type | Runbook Reference | Automated Actions | Manual Steps |
|---|---|---|---|
| API Performance Degradation | RB-001 | Scale backend instances | Check
| Mobile App Crashes | RB-002 | Collect crash reports | Analyze stack tra
| AI Service Failures | RB-003 | Switch to fallback content | Contact Op
| Database Issues | RB-004 | Activate read replicas | Check connection p

#### 6.5.3.4 Post-Mortem Process

**Incident Analysis Framework:**

| Analysis Phase | Timeline | Participants | Deliverables |
|---|---|---|---|
| Initial Assessment | Within 24 hours | Incident responders | Timeline a
| Root Cause Analysis | Within 72 hours | Technical team | Technical ana
| Action Items | Within 1 week | Full team | Improvement plan |
| Follow-up Review | Within 1 month | Management | Implementation status

### 6.5.4 DASHBOARD DESIGN

#### 6.5.4.1 Executive Dashboard Layout

<div class="mermaid-wrapper" id="mermaid-diagram-xv6ryuhmv">
          <div class="mermaid">
graph TB
    subgraph &quot;Executive Dashboard - PropertyPro AI&quot;
        subgraph &quot;Business Metrics Row&quot;
            A1[Active Users&lt;br/&gt;Daily/Monthly]
            A2[Property Listings&lt;br/&gt;Created Today]
            A3[AI Content Generated&lt;br/&gt;Success Rate]
            A4[Revenue Impact&lt;br/&gt;Monthly Trend]
        end
```

```
        subgraph &quot;System Health Row&quot;
            B1[System Uptime&lt;br/&gt;99.9% SLA]
            B2[API Response Time&lt;br/&gt;&lt; 2s Target]
            B3[Mobile Performance&lt;br/&gt;60 FPS Target]
            B4[Error Rate&lt;br/&gt;&lt; 0.5% Target]
        end

        subgraph &quot;User Experience Row&quot;
            C1[Mobile App&lt;br/&gt;Crash Rate]
            C2[User Satisfaction&lt;br/&gt;Rating Score]
            C3[Feature Adoption&lt;br/&gt;Usage Statistics]
            C4[Support Tickets&lt;br/&gt;Volume &amp; Resolution]
        end

        subgraph &quot;Operational Metrics Row&quot;
            D1[Infrastructure Costs&lt;br/&gt;Monthly Spend]
            D2[AI API Usage&lt;br/&gt;Token Consumption]
            D3[Database Performance&lt;br/&gt;Query Times]
            D4[Security Alerts&lt;br/&gt;Threat Detection]
        end
    end

    style A1 fill:#e8f5e8
    style A2 fill:#e8f5e8
    style A3 fill:#e8f5e8
    style A4 fill:#e8f5e8
    style B1 fill:#e1f5fe
    style B2 fill:#e1f5fe
    style B3 fill:#e1f5fe
    style B4 fill:#e1f5fe
</div>
        </div>

#### 6.5.4.2 Technical Operations Dashboard

**Real-Time Monitoring Panels:**

| Panel Category | Metrics Displayed | Update Frequency | Alert Integrati
|---|---|---|---|
| API Performance | Request rate, response time, error rate | 10 seconds
| Mobile Metrics | Session replay, crash analysis | 30 seconds | Yes |
| AI Services | Generation time, success rate, cost | 60 seconds | Yes |
| Infrastructure | CPU, memory, disk usage | 15 seconds | Yes |
```

```
#### 6.5.4.3 Business Intelligence Dashboard

**Key Business Metrics:**

| Metric | Visualization | Business Value | Stakeholder |
|---|---|---|---|
| User Engagement | Time series chart | Product adoption | Product Manage
| Feature Usage | Heat map | Feature prioritization | Development Team |
| Performance Trends | Line graphs | System optimization | Engineering Te
| Cost Analysis | Bar charts | Budget management | Finance Team |

### 6.5.5 MONITORING IMPLEMENTATION

#### 6.5.5.1 FastAPI Instrumentation

**Prometheus Integration:**
```

python
from prometheus_fastapi_instrumentator import Instrumentator
from fastapi import FastAPI

app = FastAPI(title="PropertyPro AI API")

Initialize Prometheus instrumentation

instrumentator = Instrumentator(
should_group_status_codes=False,
should_ignore_untemplated=True,
should_respect_env_var=True,
should_instrument_requests_inprogress=True,
excluded_handlers=["/health", "/metrics"],
env_var_name="ENABLE_METRICS",
inprogress_name="fastapi_inprogress",
inprogress_labels=True,
)

Instrument the FastAPI app

instrumentator.instrument(app).expose(app)

```
#### 6.5.5.2 React Native Monitoring Integration

**Performance Monitoring Setup:**

React Native monitoring requires simple installation with auto-linking fo

| Monitoring Aspect | Implementation | Benefits |
|---|---|---|---|
| JavaScript Stack Traces | SDK integration | Full debugging capability
| Network Monitoring | HTTP client instrumentation | API performance trac
| Screen Tracking | Navigation integration | User journey analysis |
| Crash Reporting | Automatic deobfuscation | Rapid issue resolution |

#### 6.5.5.3 Custom Metrics Implementation

**Business-Specific Monitoring:**
```

python
from prometheus_client import Counter, Histogram, Gauge

Business metrics

property_listings_created = Counter(
'property_listings_total',
'Total number of property listings created',
['user_type', 'property_type']
)

ai_content_generation_time = Histogram(
'ai_content_generation_seconds',
'Time spent generating AI content',
['content_type', 'model_version']
)

active_users = Gauge(
'active_users_current',

'Current number of active users',
['platform', 'version']
)

```
PropertyPro AI's monitoring and observability architecture provides compr

## 6.6 TESTING STRATEGY

### 6.6.1 TESTING APPROACH

#### 6.6.1.1 Unit Testing

PropertyPro AI implements a comprehensive unit testing strategy that ensu

#### Testing Frameworks and Tools

| Component | Framework | Version | Purpose | Key Features |
|---|---|---|---|
| React Native | Jest + React Native Testing Library | Jest 29.0+, RNTL 1
| FastAPI Backend | pytest + httpx | pytest 7.0+, httpx 0.24+ | API endpo
| AI Services | pytest + pytest-asyncio | pytest-asyncio 0.21+ | Asynchro
| Database Layer | pytest + pytest-postgresql | pytest-postgresql 5.0+ | 

#### Test Organization Structure

**React Native Test Structure:**
```

frontend/src/
├── components/
│ ├── **tests**/
│ │ ├── PropertyCard.test.tsx
│ │ ├── ClientList.test.tsx
│ │ └── AIChat.test.tsx
├── screens/
│ ├── **tests**/
│ │ ├── Dashboard.test.tsx
│ │ └── PropertyManagement.test.tsx
├── services/

│ ├── **tests**/
│ │ ├── apiService.test.ts
│ │ └── aiService.test.ts
└── utils/
└── **tests**/
└── helpers.test.ts

```
**FastAPI Test Structure:**
```

backend/tests/
├── unit/
│ ├── test_property_service.py
│ ├── test_client_service.py
│ └── test_ai_service.py
├── integration/
│ ├── test_property_api.py
│ └── test_auth_flow.py
├── fixtures/
│ ├── conftest.py
│ └── factories.py
└── utils/
└── test_helpers.py

```
#### Mocking Strategy

**React Native Component Mocking:**

The system employs component composition with mocking to test component i
```

typescript
// Mock external dependencies
jest.mock('@react-native-async-storage/async-storage');
jest.mock('react-native-vector-icons/MaterialIcons');

// Mock AI service for predictable testing
jest.mock('../services/aiService', () => ({
generatePropertyDescription: jest.fn(),
analyzeMarketData: jest.fn(),
}));

// Component test with mocked dependencies
describe('PropertyCard Component', () => {
const mockProperty = {
id: '123',
title: 'Test Property',
price: 500000,
bedrooms: 3,
bathrooms: 2,
};

it('renders property information correctly', () => {
const { getByText } = render(

);

```
expect(getByText('Test Property')).toBeTruthy();
expect(getByText('$500,000')).toBeTruthy();
```

});
});

```
**FastAPI Service Mocking:**

FastAPI provides TestClient for endpoint testing, while external services
```

python

conftest.py - Test fixtures and mocks

import pytest
from unittest.mock import AsyncMock, patch
from fastapi.testclient import TestClient
from app.main import app

@pytest.fixture
def client():
return TestClient(app)

@pytest.fixture
def mock_openai_service():
with patch('app.services.ai_service.OpenAIService') as mock:
mock_instance = AsyncMock()
mock_instance.generate_content.return_value = "Generated content"
mock.return_value = mock_instance
yield mock_instance

Test with mocked AI service

def test_generate_property_description(client, mock_openai_service):
response = client.post("/api/v1/ai/generate", json={
"property_id": "123",
"content_type": "description"
})

```
assert response.status_code == 200
assert "Generated content" in response.json()["content"]
```

```
#### Code Coverage Requirements
```

```
| Component | Coverage Target | Measurement Tool | Exclusions |
|---|---|---|---|
| React Native Components | 85% | Jest coverage reports | Third-party lil
| FastAPI Endpoints | 90% | pytest-cov for comprehensive coverage analysi
| Business Logic | 95% | Combined coverage analysis | External API respor
| Utility Functions | 100% | Unit test coverage | Platform-specific imple

#### Test Naming Conventions

**React Native Test Naming:**
```

typescript

describe('PropertyManagementScreen', () => {

describe('when user has properties', () => {

it('should display property list with correct data', () => {});

it('should handle property selection correctly', () => {});

});

describe('when user has no properties', () => {

it('should display empty state message', () => {});

it('should show create property button', () => {});

});

});

```
**FastAPI Test Naming:**
```

python

class TestPropertyService:

async def test_create_property_with_valid_data_returns_property(self):

"""Test that creating a property with valid data returns the created property."""

pass

```
async def test_create_property_with_invalid_data_raises_validation_error
    """Test that creating a property with invalid data raises Validation
```

```
    pass
```

```
#### Test Data Management

Factory-based test data generation simplifies test data creation with rea
```

python

factories.py - Test data factories

import factory
from app.models import Property, User, Client

class UserFactory(factory.alchemy.SQLAlchemyModelFactory):
class Meta:
model = User
sqlalchemy_session_persistence = "commit"

```
email = factory.Faker('email')
first_name = factory.Faker('first_name')
last_name = factory.Faker('last_name')
phone = factory.Faker('phone_number')
```

class PropertyFactory(factory.alchemy.SQLAlchemyModelFactory):
class Meta:
model = Property
sqlalchemy_session_persistence = "commit"

```
title = factory.Faker('sentence', nb_words=4)
price = factory.Faker('random_int', min=100000, max=2000000)
bedrooms = factory.Faker('random_int', min=1, max=6)
```

```
bathrooms = factory.Faker('random_int', min=1, max=4)
user = factory.SubFactory(UserFactory)
```

```
#### 6.6.1.2 Integration Testing

Integration testing validates the interaction between multiple system co

#### Service Integration Test Approach

**API Integration Testing:**

End-to-end testing simulates real user behavior by sending HTTP requests

| Integration Layer | Test Scope | Tools | Validation Points |
|---|---|---|---|
| API-Database | Endpoint to data persistence | TestClient with database
| Frontend-Backend | Mobile app to API communication | Mock server, netw
| AI Service Integration | OpenAI API communication | API parameter vali
| Authentication Flow | Login to protected resources | JWT token validat

#### API Testing Strategy

**FastAPI Integration Tests:**
```

python

test_property_integration.py

import pytest
from httpx import AsyncClient
from app.main import app

@pytest.mark.asyncio
async def test_property_creation_workflow():
"""Test complete property creation workflow including AI generation."""
async with AsyncClient(app=app, base_url="http://test") as client:

# 1. Authenticate user
auth_response = await client.post("/auth/login", json={
"email": "test@example.com",
"password": "testpass123"
})
token = auth_response.json()["access_token"]
headers = {"Authorization": f"Bearer {token}"}

```
# 2. Create property
property_data = {
    "title": "Test Property",
    "price": 500000,
    "bedrooms": 3,
    "bathrooms": 2,
    "location": "Test City"
}

create_response = await client.post(
    "/api/v1/properties/",
    json=property_data,
    headers=headers
)

assert create_response.status_code == 201
property_id = create_response.json()["id"]

# 3. Verify AI content generation
content_response = await client.get(
    f"/api/v1/properties/{property_id}/content",
    headers=headers
)

assert content_response.status_code == 200
assert "description" in content_response.json()
```

```
#### Database Integration Testing

**Test Database Configuration:**
```

```
Tests use a separate PostgreSQL database exclusively for testing to ensu
```

python

conftest.py - Database test configuration

import pytest
import asyncio
from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession
from app.core.database import Base
from app.core.config import settings

@pytest.fixture(scope="session")
def event_loop():
"""Create event loop for async tests."""
loop = asyncio.get_event_loop_policy().new_event_loop()
yield loop
loop.close()

@pytest.fixture(scope="session")
async def test_engine():
"""Create test database engine."""
engine = create_async_engine(
settings.TEST_DATABASE_URL,
echo=False
)

```
async with engine.begin() as conn:
    await conn.run_sync(Base.metadata.create_all)

yield engine
```

```
async with engine.begin() as conn:
    await conn.run_sync(Base.metadata.drop_all)
```

@pytest.fixture
async def db_session(test_engine):
"""Create database session for tests."""
async with AsyncSession(test_engine) as session:
yield session
await session.rollback()

```
#### External Service Mocking

**OpenAI API Integration Testing:**
```

python

test_ai_integration.py

import pytest
from unittest.mock import patch, AsyncMock

@pytest.mark.asyncio
@patch('app.services.ai_service.AsyncOpenAI')
async def test_ai_content_generation_with_rate_limiting(mock_openai, client):
"""Test AI service handles rate limiting gracefully."""

```
# Mock rate limit error then success
mock_client = AsyncMock()
mock_openai.return_value = mock_client

# First call fails with rate limit
mock_client.chat.completions.create.side_effect = [
```

```
    Exception("Rate limit exceeded"),
    AsyncMock(choices=[AsyncMock(message=AsyncMock(content="Generated con
]

response = await client.post("/api/v1/ai/generate", json={
    "property_id": "123",
    "content_type": "description"
})

# Should retry and succeed
assert response.status_code == 200
assert mock_client.chat.completions.create.call_count == 2
```

```
#### Test Environment Management

| Environment | Purpose | Configuration | Data Management |
|---|---|---|---|
| Unit Test | Isolated component testing | In-memory database, mocked ser
| Integration Test | Service interaction testing | Test database, mocked
| Staging | Pre-production validation | Production-like setup, sandbox AI
| Performance Test | Load and stress testing | Scaled infrastructure, rea

#### 6.6.1.3 End-to-End Testing

E2E tests provide the highest confidence by testing the complete user jou

#### E2E Test Scenarios

**Critical User Journeys:**

| Scenario | Test Coverage | Success Criteria | Tools |
|---|---|---|---|
| User Registration & Login | Authentication flow, session management | S
| Property Listing Creation | Complete property workflow with AI | Proper
| Client Management | CRM operations, follow-up automation | Client added
| AI Content Generation | OpenAI integration, content quality | Content g

#### UI Automation Approach

**React Native E2E Testing with Detox:**
```

```
Detox provides powerful E2E testing for React Native applications with Je
```

javascript
// e2e/propertyCreation.e2e.js
describe('Property Creation Flow', () => {
beforeAll(async () => {
await device.launchApp();
});

beforeEach(async () => {
await device.reloadReactNative();
});

it('should create property with AI content generation', async () => {
// 1. Navigate to property creation
await element(by.id('properties-tab')).tap();
await element(by.id('add-property-button')).tap();

```
// 2. Fill property details
await element(by.id('property-title-input')).typeText('Test Property');
await element(by.id('property-price-input')).typeText('500000');
await element(by.id('bedrooms-picker')).tap();
await element(by.text('3')).tap();

// 3. Upload photos
await element(by.id('photo-upload-button')).tap();
await element(by.text('Camera')).tap();

// 4. Generate AI content
await element(by.id('generate-content-button')).tap();
await waitFor(element(by.id('ai-content-preview')))
  .toBeVisible()
  .withTimeout(10000);

// 5. Save property
await element(by.id('save-property-button')).tap();
```

```
// 6. Verify property appears in list
await expect(element(by.text('Test Property'))).toBeVisible();
```

});
});

```
#### Test Data Setup/Teardown

**E2E Test Data Management:**
```

javascript
// e2e/setup.js
const { execSync } = require('child_process');

beforeAll(async () => {
// Reset test database
execSync('npm run db:reset:test');

// Seed test data
execSync('npm run db:seed:test');

// Start mock services
await startMockServices();
});

afterAll(async () => {
// Cleanup test data
execSync('npm run db:cleanup:test');

// Stop mock services
await stopMockServices();
});

```
#### Performance Testing Requirements

| Metric | Target | Measurement Method | Failure Threshold |
```

```
|---|---|---|---|
| App Launch Time | < 3 seconds | Device performance monitoring | > 5 sec
| Screen Navigation | < 500ms | UI response time tracking | > 1 second |
| API Response Time | < 2 seconds | Network request monitoring | > 5 seco
| AI Content Generation | < 10 seconds | End-to-end timing | > 30 seconds

#### Cross-Browser Testing Strategy

**Mobile Platform Coverage:**

| Platform | Versions | Test Scope | Automation Level |
|---|---|---|---|
| iOS | 14.0+ | Core functionality, UI consistency | Automated with Deto
| Android | API 24+ | Feature parity, performance | Automated with Detox
| Tablet (iPad) | iOS 14+ | Responsive design, touch interactions | Manua
| Tablet (Android) | API 24+ | Layout adaptation, performance | Manual va

### 6.6.2 TEST AUTOMATION

#### 6.6.2.1 CI/CD Integration

PropertyPro AI implements comprehensive test automation integrated with

#### Automated Test Triggers

<div class="mermaid-wrapper" id="mermaid-diagram-qve3s2mir">
          <div class="mermaid">
graph TB
    subgraph &quot;Code Repository&quot;
        PR[Pull Request Created]
        PUSH[Push to Main Branch]
        SCHEDULE[Scheduled Runs]
    end

    subgraph &quot;CI/CD Pipeline&quot;
        LINT[Code Linting &amp; Formatting]
        UNIT[Unit Tests]
        INTEGRATION[Integration Tests]
        E2E[E2E Tests]
        BUILD[Build Application]
        DEPLOY[Deploy to Staging]
    end
```

```
    subgraph &quot;Test Execution&quot;
        PARALLEL[Parallel Test Execution]
        REPORT[Test Report Generation]
        NOTIFY[Notification System]
    end

    PR --&gt; LINT
    PUSH --&gt; LINT
    SCHEDULE --&gt; E2E

    LINT --&gt; UNIT
    UNIT --&gt; INTEGRATION
    INTEGRATION --&gt; BUILD
    BUILD --&gt; E2E
    E2E --&gt; DEPLOY

    UNIT --&gt; PARALLEL
    INTEGRATION --&gt; PARALLEL
    E2E --&gt; PARALLEL

    PARALLEL --&gt; REPORT
    REPORT --&gt; NOTIFY

    style PR fill:#e1f5fe
    style PARALLEL fill:#c8e6c9
    style NOTIFY fill:#fff3e0
</div>
        </div>

#### GitHub Actions Workflow Configuration
```

yaml

.github/workflows/test.yml

name: Test Suite

on:
pull_request:

```
branches: [main, develop]
push:
branches: [main]
schedule:
- cron: '0 2 * * *' # Daily at 2 AM

jobs:
lint:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4
- uses: actions/setup-node@v4
with:
node-version: '18'
- name: Install dependencies
run: npm ci
- name: Run ESLint
run: npm run lint
- name: Run Prettier
run: npm run format:check

unit-tests:
runs-on: ubuntu-latest
needs: lint
strategy:
matrix:
component: [frontend, backend]
steps:
- uses: actions/checkout@v4
- name: Setup test environment
run: |
if [ "${{ matrix.component }}" == "frontend" ]; then
npm ci
else
```

```
pip install -r requirements.txt
fi
- name: Run unit tests
run: |
if [ "${{ matrix.component }}" == "frontend" ]; then
npm run test:unit -- --coverage
else
pytest tests/unit/ --cov=app --cov-report=xml
fi
- name: Upload coverage
uses: codecov/codecov-action@v3

integration-tests:
runs-on: ubuntu-latest
needs: unit-tests
services:
postgres:
image: postgres:15
env:
POSTGRES_PASSWORD: testpass
POSTGRES_DB: propertypro_test
options: >-
--health-cmd pg_isready
--health-interval 10s
--health-timeout 5s
--health-retries 5
steps:
- uses: actions/checkout@v4
- name: Setup Python
uses: actions/setup-python@v4
with:
python-version: '3.11'
- name: Install dependencies
run: pip install -r requirements.txt
```

- name: Run integration tests
env:
DATABASE_URL: postgresql://postgres:testpass@localhost/propertypro_test
OPENAI_API_KEY: ${{ secrets.OPENAI_TEST_API_KEY }}
run: pytest tests/integration/ -v

e2e-tests:
runs-on: macos-latest
needs: integration-tests
if: github.event_name == 'push' || github.event_name == 'schedule'
steps:
- uses: actions/checkout@v4
- name: Setup Node.js
uses: actions/setup-node@v4
with:
node-version: '18'
- name: Install dependencies
run: npm ci
- name: Setup iOS Simulator
run: |
xcrun simctl create "iPhone 14" "iPhone 14" "iOS16.0"
xcrun simctl boot "iPhone 14"
- name: Build for testing
run: npx detox build --configuration ios.sim.debug
- name: Run E2E tests
run: npx detox test --configuration ios.sim.debug --cleanup

```
#### Parallel Test Execution

**Test Parallelization Strategy:**

| Test Type | Parallelization Method | Resource Allocation | Expected Spe
|---|---|---|---|
| Unit Tests | Jest worker processes | 4 parallel workers | 3-4x faster
| Integration Tests | pytest-xdist | 2 parallel processes | 2x faster |
```

```
| E2E Tests | Device/simulator pools | 2 simulators | 2x faster |
| API Tests | Concurrent requests | Thread pool execution | 5x faster |

#### Test Reporting Requirements

**Comprehensive Test Reports:**
```

typescript
// jest.config.js - Test reporting configuration
module.exports = {
reporters: [
'default',
['jest-junit', {
outputDirectory: 'test-results',
outputName: 'junit.xml',
}],
['jest-html-reporters', {
publicPath: 'test-results',
filename: 'test-report.html',
}],
],
coverageReporters: [
'text',
'lcov',
'html',
'cobertura'
],
collectCoverageFrom: [
'src/***/*.{ts,tsx}', '!src/**/*.d.ts',
'!src/index.tsx',
],
};

```
#### Failed Test Handling
```

```
**Automatic Retry and Notification System:**
```

python

pytest.ini - Test retry configuration

[tool:pytest]
addopts =
--strict-markers
--strict-config
--reruns 2
--reruns-delay 1
--tb=short
--cov=app
--cov-report=term-missing
--cov-report=html:htmlcov
--cov-fail-under=85

markers =
slow: marks tests as slow
integration: marks tests as integration tests
e2e: marks tests as end-to-end tests

```
#### Flaky Test Management

**Flaky Test Detection and Resolution:**

| Detection Method | Threshold | Action | Monitoring |
|---|---|---|---|
| Test History Analysis | 3 failures in 10 runs | Mark as flaky, investig
| Execution Time Variance | >50% time variation | Performance investigat
| Environment Dependencies | Platform-specific failures | Environment iso
| External Service Issues | API timeout patterns | Mock service implemen
```

```
### 6.6.3 QUALITY METRICS

#### 6.6.3.1 Code Coverage Targets

PropertyPro AI maintains strict code coverage requirements to ensure comp

#### Coverage Requirements by Component

| Component | Coverage Target | Current Coverage | Measurement Tool | Exc
|---|---|---|---|
| React Native Components | 85% | 87% | Jest coverage reports | Third-par
| FastAPI Endpoints | 90% | 92% | pytest-cov integration | Configuration
| Business Logic Services | 95% | 94% | Combined coverage analysis | Exte
| Utility Functions | 100% | 98% | Unit test coverage | Platform-specifi

#### Coverage Quality Gates

**Automated Coverage Enforcement:**
```

yaml

Coverage quality gates in CI/CD

coverage_gates:
minimum_coverage: 85%
coverage_decrease_threshold: 2%
uncovered_lines_threshold: 50

branch_coverage:
minimum: 80%
critical_paths: 95%

function_coverage:
minimum: 90%

public_apis: 100%

```
#### 6.6.3.2 Test Success Rate Requirements

#### Success Rate Targets

| Test Category | Success Rate Target | Current Rate | Acceptable Failure
|---|---|---|---|
| Unit Tests | 100% | 99.8% | 0% | Any failure blocks deployment |
| Integration Tests | 98% | 97.5% | 2% | >5% failure rate triggers invest
| E2E Tests | 95% | 94.2% | 5% | >10% failure rate requires immediate ac
| Performance Tests | 90% | 89.1% | 10% | Trend analysis for degradation

#### Test Reliability Metrics

**Flaky Test Tracking:**

<div class="mermaid-wrapper" id="mermaid-diagram-vwtczp5m4">
          <div class="mermaid">
graph TB
    subgraph &quot;Test Execution Monitoring&quot;
        A[Test Execution] --&gt; B{Test Result}
        B --&gt;|Pass| C[Success Counter]
        B --&gt;|Fail| D[Failure Analysis]
        B --&gt;|Flaky| E[Flaky Test Registry]
    end

    subgraph &quot;Reliability Calculation&quot;
        C --&gt; F[Calculate Success Rate]
        D --&gt; G[Categorize Failure]
        E --&gt; H[Track Flaky Pattern]
    end

    subgraph &quot;Quality Actions&quot;
        F --&gt; I{Success Rate &lt; Target?}
        G --&gt; J[Root Cause Analysis]
        H --&gt; K[Flaky Test Remediation]

        I --&gt;|Yes| L[Block Deployment]
        I --&gt;|No| M[Continue Pipeline]

        J --&gt; N[Fix Implementation]
```

```
        K --&gt; O[Stabilize Test]
    end

    style A fill:#e1f5fe
    style M fill:#c8e6c9
    style L fill:#ffcdd2
</div>
        </div>

#### 6.6.3.3 Performance Test Thresholds

#### API Performance Requirements

| Endpoint Category | Response Time Target | Throughput Target | Error Ra
|---|---|---|---|
| Authentication | < 500ms | 100 req/sec | < 0.1% | 10 minutes |
| Property CRUD | < 1 second | 50 req/sec | < 0.5% | 15 minutes |
| AI Content Generation | < 5 seconds | 10 req/sec | < 2% | 30 minutes |
| Analytics Queries | < 2 seconds | 25 req/sec | < 1% | 20 minutes |

#### Mobile App Performance Targets

| Performance Metric | Target | Measurement Method | Failure Threshold |
|---|---|---|---|
| App Launch Time | < 2 seconds | Automated timing | > 4 seconds |
| Screen Transition | < 300ms | UI performance monitoring | > 1 second |
| Memory Usage | < 150MB | Device profiling | > 300MB |
| Battery Impact | Minimal | Background activity monitoring | High drain

#### 6.6.3.4 Quality Gates

#### Deployment Quality Gates

**Multi-Stage Quality Validation:**

<div class="mermaid-wrapper" id="mermaid-diagram-sipcvpvtp">
          <div class="mermaid">
graph LR
    subgraph &quot;Quality Gate Stages&quot;
        A[Code Quality] --&gt; B[Test Coverage]
        B --&gt; C[Test Success Rate]
        C --&gt; D[Performance Benchmarks]
        D --&gt; E[Security Scan]
```

```
        E --&gt; F[Deployment Approval]
    end

    subgraph &quot;Gate Criteria&quot;
        A1[Linting: 100% Pass&lt;br/&gt;Formatting: Compliant&lt;br/&gt;
        B1[Unit: &gt;85%&lt;br/&gt;Integration: &gt;80%&lt;br/&gt;E2E: &
        C1[Unit: 100%&lt;br/&gt;Integration: &gt;98%&lt;br/&gt;E2E: &gt;
        D1[API: &lt;2s response&lt;br/&gt;Mobile: &lt;3s launch&lt;br/&g
        E1[Vulnerabilities: None&lt;br/&gt;Dependencies: Updated&lt;br/&
    end

    A -.-&gt; A1
    B -.-&gt; B1
    C -.-&gt; C1
    D -.-&gt; D1
    E -.-&gt; E1

    style F fill:#c8e6c9
    style A1 fill:#fff3e0
    style B1 fill:#fff3e0
    style C1 fill:#fff3e0
    style D1 fill:#fff3e0
    style E1 fill:#fff3e0
</div>
        </div>

#### Automated Quality Enforcement
```

python

quality_gates.py - Automated quality gate enforcement

class QualityGate:
def **init**(self):
self.criteria = {
'code_coverage': 85.0,

'test_success_rate': 98.0,
'performance_threshold': 2.0, # seconds
'security_score': 8.0, # out of 10
}

```
def evaluate_deployment_readiness(self, metrics: dict) -> bool:
    """Evaluate if deployment meets quality criteria."""

    results = {}

    # Check code coverage
    results['coverage'] = metrics['coverage'] >= self.criteria['code_cove

    # Check test success rate
    results['tests'] = metrics['test_success_rate'] >= self.criteria['tes

    # Check performance
    results['performance'] = metrics['avg_response_time'] <= self.criteri

    # Check security
    results['security'] = metrics['security_score'] >= self.criteria['sec

    # All criteria must pass
    deployment_ready = all(results.values())

    if not deployment_ready:
        self.generate_quality_report(results, metrics)

    return deployment_ready

def generate_quality_report(self, results: dict, metrics: dict):
    """Generate detailed quality gate report."""

    failed_criteria = [
        criterion for criterion, passed in results.items()
        if not passed
    ]

    report = {
        'deployment_blocked': True,
        'failed_criteria': failed_criteria,
        'current_metrics': metrics,
```

```
        'required_metrics': self.criteria,
        'recommendations': self.get_improvement_recommendations(failed_c
    }

    return report
```

```
#### 6.6.3.5 Documentation Requirements

#### Test Documentation Standards

| Documentation Type | Requirement | Format | Update Frequency |
|---|---|---|---|
| Test Plan | Comprehensive test strategy | Markdown | Per release |
| Test Cases | Detailed test scenarios | Structured comments | Per featu
| API Test Documentation | Endpoint testing guide | OpenAPI annotations
| E2E Test Scenarios | User journey documentation | Behavior-driven desc

#### Test Maintenance Documentation

**Test Maintenance Guidelines:**
```

typescript
/**

- PropertyCard Component Test Suite
-
  - @description Tests for PropertyCard component covering:
- - Rendering with different property types
- - User interaction handling
- - Error state management
- - Accessibility compliance
-
  - @maintainer Frontend Team
- @lastUpdated 2024-01-15
- @coverage 92%
-
  - @testScenarios

- - Happy path: Property displays correctly
- - Edge cases: Missing data handling
- - Error cases: Invalid property data
- - Accessibility: Screen reader compatibility
  */
  describe('PropertyCard Component', () => {
  // Test implementation
  });

```
### 6.6.4 TEST EXECUTION FLOW

#### 6.6.4.1 Test Execution Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-dd1amxm4q">
          <div class="mermaid">
flowchart TD
    A[Developer Commits Code] --&gt; B[Pre-commit Hooks]
    B --&gt; C{Code Quality Check}
    C --&gt;|Pass| D[Push to Repository]
    C --&gt;|Fail| E[Block Commit]

    D --&gt; F[CI/CD Pipeline Triggered]
    F --&gt; G[Parallel Test Execution]

    G --&gt; H[Unit Tests]
    G --&gt; I[Integration Tests]
    G --&gt; J[Linting &amp; Formatting]

    H --&gt; K{Unit Tests Pass?}
    I --&gt; L{Integration Tests Pass?}
    J --&gt; M{Code Quality Pass?}

    K --&gt;|No| N[Test Failure Report]
    L --&gt;|No| N
    M --&gt;|No| N

    K --&gt;|Yes| O[Coverage Analysis]
    L --&gt;|Yes| O
    M --&gt;|Yes| O
```

```
    O --&gt; P{Coverage Threshold Met?}
    P --&gt;|No| Q[Coverage Report]
    P --&gt;|Yes| R[Build Application]

    R --&gt; S[E2E Tests]
    S --&gt; T{E2E Tests Pass?}
    T --&gt;|No| U[E2E Failure Report]
    T --&gt;|Yes| V[Performance Tests]

    V --&gt; W{Performance OK?}
    W --&gt;|No| X[Performance Report]
    W --&gt;|Yes| Y[Security Scan]

    Y --&gt; Z{Security Check Pass?}
    Z --&gt;|No| AA[Security Report]
    Z --&gt;|Yes| BB[Deploy to Staging]

    N --&gt; CC[Notify Developer]
    Q --&gt; CC
    U --&gt; CC
    X --&gt; CC
    AA --&gt; CC

    BB --&gt; DD[Production Deployment]

    style A fill:#e1f5fe
    style DD fill:#c8e6c9
    style E fill:#ffcdd2
    style CC fill:#fff3e0
</div>
        </div>

#### 6.6.4.2 Test Environment Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-dphc4kupq">
          <div class="mermaid">
graph TB
    subgraph &quot;Development Environment&quot;
        DEV_LOCAL[Local Development]
        DEV_UNIT[Unit Test Runner]
        DEV_MOCK[Mock Services]
    end
```

```
    subgraph &quot;CI/CD Environment&quot;
        CI_RUNNER[GitHub Actions Runner]
        CI_POSTGRES[#40;Test PostgreSQL#41;]
        CI_REDIS[Test Redis Cache]
        CI_MOCK[Mock External APIs]
    end

    subgraph &quot;Staging Environment&quot;
        STAGE_APP[Staging Application]
        STAGE_DB[#40;Staging Database#41;]
        STAGE_AI[Sandbox AI Services]
    end

    subgraph &quot;Production Environment&quot;
        PROD_APP[Production Application]
        PROD_DB[#40;Production Database#41;]
        PROD_AI[Production AI Services]
    end

    DEV_LOCAL --&gt; DEV_UNIT
    DEV_UNIT --&gt; DEV_MOCK

    CI_RUNNER --&gt; CI_POSTGRES
    CI_RUNNER --&gt; CI_REDIS
    CI_RUNNER --&gt; CI_MOCK

    STAGE_APP --&gt; STAGE_DB
    STAGE_APP --&gt; STAGE_AI

    PROD_APP --&gt; PROD_DB
    PROD_APP --&gt; PROD_AI

    DEV_LOCAL -.-&gt;|Push Code| CI_RUNNER
    CI_RUNNER -.-&gt;|Deploy| STAGE_APP
    STAGE_APP -.-&gt;|Promote| PROD_APP

    style DEV_LOCAL fill:#e1f5fe
    style CI_RUNNER fill:#fff3e0
    style STAGE_APP fill:#f3e5f5
    style PROD_APP fill:#c8e6c9
</div>
        </div>
```

```
#### 6.6.4.3 Test Data Flow

<div class="mermaid-wrapper" id="mermaid-diagram-5665r375b">
          <div class="mermaid">
sequenceDiagram
    participant Dev as Developer
    participant CI as CI/CD Pipeline
    participant TestDB as Test Database
    participant MockAI as Mock AI Service
    participant Report as Test Reports

    Dev-&gt;&gt;CI: Push Code Changes
    CI-&gt;&gt;CI: Setup Test Environment
    CI-&gt;&gt;TestDB: Initialize Test Data
    TestDB--&gt;&gt;CI: Database Ready

    CI-&gt;&gt;CI: Run Unit Tests
    CI-&gt;&gt;TestDB: Execute Integration Tests
    TestDB--&gt;&gt;CI: Test Results

    CI-&gt;&gt;MockAI: Test AI Integration
    MockAI--&gt;&gt;CI: Mock Responses

    CI-&gt;&gt;CI: Generate Coverage Report
    CI-&gt;&gt;Report: Publish Test Results

    alt Tests Pass
        CI-&gt;&gt;Dev: Success Notification
        CI-&gt;&gt;CI: Proceed to Deployment
    else Tests Fail
        CI-&gt;&gt;Dev: Failure Notification
        CI-&gt;&gt;Report: Detailed Error Report
    end
</div>
        </div>

PropertyPro AI's comprehensive testing strategy ensures high-quality, re

# 7. USER INTERFACE DESIGN

## 7.1 CORE UI TECHNOLOGIES

### 7.1.1 Frontend Technology Stack
```

```
PropertyPro AI implements a modern mobile-first user interface using Reac

| Technology | Version | Purpose | Key Features |
|---|---|---|---|
| React Native | 0.71+ | Cross-platform mobile framework | Built-in TypeS
| TypeScript | 5.0+ | Type safety and developer experience | Enhanced IDE
| React Navigation | 6.0+ | Navigation and routing | Type-safe navigatior
| Zustand | 4.4+ | State management | Lightweight, TypeScript-friendly st
| React Native Vector Icons | 10.0+ | Icon system | Comprehensive icon li
| React Native Reanimated | 3.6+ | Animations and gestures | High-perforn

### 7.1.2 UI Architecture Pattern

The application follows a **Component-Based Architecture** with **Contair

<div class="mermaid-wrapper" id="mermaid-diagram-9i3onyl02">
          <div class="mermaid">
graph TB
    subgraph &quot;UI Architecture Layers&quot;
        A[Screen Components&lt;br/&gt;Navigation Containers]
        B[Feature Components&lt;br/&gt;Business Logic Containers]
        C[UI Components&lt;br/&gt;Reusable Presenters]
        D[Service Layer&lt;br/&gt;API Integration]
    end

    subgraph &quot;State Management&quot;
        E[Zustand Store&lt;br/&gt;Global State]
        F[React Context&lt;br/&gt;Feature State]
        G[Local State&lt;br/&gt;Component State]
    end

    A --&gt; B
    B --&gt; C
    C --&gt; D

    E --&gt; A
    F --&gt; B
    G --&gt; C

    style A fill:#e1f5fe
    style B fill:#fff3e0
    style C fill:#e8f5e8
```

```
    style D fill:#f3e5f5
</div>
        </div>

### 7.1.3 Design System Implementation

**Color Palette:**
```

typescript
export const colors = {
primary: '#2563eb', // Blue - Properties
secondary: '#059669', // Green - Clients
accent: '#7c3aed', // Purple - Content
warning: '#ea580c', // Orange - Tasks
danger: '#dc2626', // Red - AI Assistant
info: '#0891b2', // Teal - Analytics
background: '#f8fafc', // Light gray
surface: '#ffffff', // White
text: '#1f2937', // Dark gray
textSecondary: '#6b7280' // Medium gray
};

```
**Typography System:**
```

typescript
export const typography = {
h1: { fontSize: 32, fontWeight: 'bold', lineHeight: 40 },
h2: { fontSize: 24, fontWeight: 'bold', lineHeight: 32 },
h3: { fontSize: 20, fontWeight: '600', lineHeight: 28 },
body: { fontSize: 16, fontWeight: 'normal', lineHeight: 24 },
caption: { fontSize: 14, fontWeight: 'normal', lineHeight: 20 }
};

```
## 7.2 UI USE CASES
```

```
### 7.2.1 Primary User Workflows

| Use Case | User Goal | UI Flow | Success Criteria |
|---|---|---|---|
| Property Listing Creation | Create new property listing with AI conten
| Client Follow-up Management | Manage client relationships and follow-u
| AI Content Generation | Generate marketing materials for properties | 
| Task Management | Organize and prioritize daily activities | Dashboard
| AI Assistant Consultation | Get real estate expertise and advice | Das
| Performance Analytics | Track business performance and metrics | Dashb

### 7.2.2 Mobile-Specific Interactions

**Touch Gestures:**
- **Tap**: Primary action (select, navigate, confirm)
- **Long Press**: Secondary actions (context menu, quick actions)
- **Swipe**: Navigation (back, forward, dismiss)
- **Pull to Refresh**: Data synchronization
- **Pinch to Zoom**: Image viewing and map interaction

**Voice Interactions:**
- **Voice Commands**: "Create new property listing"
- **Voice Input**: Property descriptions and client notes
- **Voice Search**: Find properties, clients, or content
- **Voice Navigation**: Hands-free app navigation

## 7.3 UI/BACKEND INTERACTION BOUNDARIES

### 7.3.1 API Communication Layer

<div class="mermaid-wrapper" id="mermaid-diagram-5k2xbcd85">
          <div class="mermaid">
sequenceDiagram
    participant UI as React Native UI
    participant Store as Zustand Store
    participant API as API Service Layer
    participant Backend as FastAPI Backend
    participant AI as OpenAI GPT-4.1

    UI-&gt;&gt;Store: User Action (Create Property)
    Store-&gt;&gt;API: API Request with Data
    API-&gt;&gt;Backend: HTTP POST /api/v1/properties
    Backend-&gt;&gt;AI: Generate Content Request
```

```
    AI--&gt;&gt;Backend: Generated Content
    Backend--&gt;&gt;API: Property + AI Content
    API--&gt;&gt;Store: Update State
    Store--&gt;&gt;UI: Re-render with New Data

    Note over UI,AI: Real-time updates with loading states
</div>
        </div>

### 7.3.2 Data Flow Architecture

| UI Component | State Management | API Endpoint | Backend Service | Data
|---|---|---|---|---|
| PropertyList | usePropertyStore | GET /api/v1/properties | PropertyServ
| PropertyForm | Local State + Store | POST /api/v1/properties | Property
| ClientList | useClientStore | GET /api/v1/clients | ClientService | UI
| AIChat | useChatStore | POST /api/v1/ai/chat | AIService | UI ↔ Store ↔
| TaskList | useTaskStore | GET /api/v1/tasks | TaskService | UI ← Store
| Analytics | useAnalyticsStore | GET /api/v1/analytics | AnalyticsServic

### 7.3.3 Error Handling and Loading States
```

```
typescript
interface UIState {
data: T | null;
loading: boolean;
error: string | null;
lastUpdated: Date | null;
}

// Example implementation
const usePropertyList = () => {
const [state, setState] = useState>({
data: null,
loading: false,
error: null,
lastUpdated: null
});
```

const fetchProperties = async () => {
setState(prev => ({ ...prev, loading: true, error: null }));
try {
const properties = await propertyService.getProperties();
setState({
data: properties,
loading: false,
error: null,
lastUpdated: new Date()
});
} catch (error) {
setState(prev => ({
...prev,
loading: false,
error: error.message
}));
}
};

return { ...state, fetchProperties };
};

```
## 7.4 UI SCHEMAS

### 7.4.1 Component Props Interfaces
```

typescript
// Core UI Component Props
interface PropertyCardProps {
property: Property;
onPress: (property: Property) => void;
onEdit: (property: Property) => void;
onDelete: (property: Property) => void;

showActions?: boolean;
}

interface ClientListItemProps {
client: Client;
onPress: (client: Client) => void;
showLeadScore?: boolean;
showLastContact?: boolean;
}

interface AIContentGeneratorProps {
propertyId: string;
contentType: 'description' | 'social' | 'email' | 'brochure';
onGenerated: (content: AIContent) => void;
onError: (error: string) => void;
}

interface TaskItemProps {
task: Task;
onToggleComplete: (taskId: string) => void;
onEdit: (task: Task) => void;
onDelete: (taskId: string) => void;
showProgress?: boolean;
}

```
### 7.4.2 Form Validation Schemas
```

typescript
// Property Form Schema
interface PropertyFormData {
title: string;
propertyType: 'apartment' | 'villa' | 'penthouse' | 'office';
price: number;
bedrooms: number;

```
bathrooms: number;
sizeSqft: number;
location: string;
description?: string;
features: string[];
images: string[];
}

const propertyValidationSchema = {
title: { required: true, minLength: 5, maxLength: 100 },
price: { required: true, min: 1000, max: 100000000 },
bedrooms: { required: true, min: 0, max: 20 },
bathrooms: { required: true, min: 0, max: 20 },
sizeSqft: { required: true, min: 100, max: 50000 },
location: { required: true, minLength: 5, maxLength: 200 }
};

// Client Form Schema
interface ClientFormData {
name: string;
email: string;
phone: string;
preferences: {
propertyTypes: string[];
priceRange: { min: number; max: number };
locations: string[];
};
notes?: string;
}

const clientValidationSchema = {
name: { required: true, minLength: 2, maxLength: 100 },
email: { required: true, pattern: /^[^\s@]+@[^\s@]+.[^\s@]+$/ },
```

phone: { required: true, pattern: /^+?[\d\s-()]+$/ }
};

```
### 7.4.3 Navigation Schema
```

typescript
// Navigation Type Definitions
export type RootStackParamList = {
Dashboard: undefined;
PropertyManagement: { filter?: PropertyFilter };
PropertyDetail: { propertyId: string };
PropertyEdit: { propertyId?: string };
ClientManagement: { filter?: ClientFilter };
ClientDetail: { clientId: string };
ClientEdit: { clientId?: string };
ContentGeneration: { propertyId?: string; contentType?: ContentType };
TaskManagement: { filter?: TaskFilter };
AIAssistant: { context?: AIContext };
Analytics: { timeRange?: TimeRange };
Settings: undefined;
};

// Screen Props Type Safety
type PropertyDetailScreenProps = NativeStackScreenProps<
RootStackParamList,
'PropertyDetail'

> ;

type PropertyEditScreenProps = NativeStackScreenProps<
RootStackParamList,
'PropertyEdit'

> ;

```
## 7.5 SCREENS REQUIRED

### 7.5.1 Core Application Screens

#### Dashboard Screen
**Purpose**: Main hub for all application features and quick access to ke

**Components**:
- Header with user greeting and notifications
- Six main action buttons (Properties, Clients, Content, Tasks, AI Assis
- Quick stats cards (Active listings, Pending tasks, New leads, Monthly
- Recent activity feed
- Quick action shortcuts

**Layout**:
```

```
┌─────────────────────────────────────────┐
│ Good morning, Sarah! │
│ January 15, 2024 │
├─────────────────────────────────────────┤
│ ┌──────────┐ ┌──────────┐ ┌──────────┐ │
│ │ Props │ │ Clients│ │ Content│ │
│ └──────────┘ └──────────┘ └──────────┘ │
│ ┌──────────┐ ┌──────────┐ ┌──────────┐ │
│ │ Tasks │ │ AI Chat│ │ Analytics│ │
│ └──────────┘ └──────────┘ └──────────┘ │
├─────────────────────────────────────────┤
│ Quick Stats │
│ Active: 12 Tasks: 8 Leads: 5 │
├─────────────────────────────────────────┤
│ Recent Activity │
│ • New lead: John Smith │
│ • Property updated: 123 Main St │
│ • Task completed: Follow up client │
└─────────────────────────────────────────┘
```

```
#### Property Management Screen
**Purpose**: Comprehensive property listing management with AI-powered fe

**Components**:
- Search and filter bar
- Property grid/list view toggle
- Property cards with key information
- Floating action button for new property
- Sort and filter options
- Bulk actions toolbar

**Key Features**:
- Real-time property status updates
- AI-generated content indicators
- Performance metrics per property
- Quick actions (edit, duplicate, archive)

#### Property Detail Screen
**Purpose**: Detailed view of individual property with all related inforn

**Components**:
- Image carousel with zoom capability
- Property information cards
- AI-generated content sections
- Performance analytics
- Related tasks and activities
- Action buttons (edit, share, generate content)

#### Client Management Screen
**Purpose**: Complete client relationship management with lead scoring

**Components**:
- Client list with lead scores
- Search and filter functionality
- Lead status indicators
- Quick contact actions
- Follow-up reminders
- Client segmentation tools

**Key Features**:
- Color-coded lead scoring
- Last contact date tracking
- Automated follow-up suggestions
```

```markdown
- Client preference indicators

#### AI Content Generation Screen
**Purpose**: AI-powered content creation for marketing materials

**Components**:
- Content type selector
- Property selection dropdown
- Tone and style options
- Generated content preview
- Edit and customize tools
- Export and sharing options

**Content Types**:
- Property descriptions
- Social media posts
- Email templates
- Marketing brochures
- Open house invitations

#### AI Assistant Chat Screen
**Purpose**: Conversational AI interface for real estate expertise

**Components**:
- Chat message interface
- Voice input button
- Quick action suggestions
- Conversation history
- Context-aware responses
- Save insights feature

**Key Features**:
- Real-time typing indicators
- Voice-to-text input
- Rich message formatting
- Conversation search
- Export chat history

### 7.5.2 Supporting Screens

#### Task Management Screen
**Purpose**: Organize and prioritize daily activities with AI suggestions
```

```
**Components**:
- Task list with priorities
- Calendar integration
- Progress tracking
- AI task suggestions
- Deadline management
- Category filtering

#### Analytics Dashboard Screen
**Purpose**: Business performance tracking and insights

**Components**:
- Performance metrics cards
- Interactive charts and graphs
- Time period selectors
- Export functionality
- Goal tracking
- Market insights

#### Settings Screen
**Purpose**: User preferences and application configuration

**Components**:
- Profile management
- Notification settings
- AI preferences
- Data sync options
- Privacy controls
- Help and support

## 7.6 USER INTERACTIONS

### 7.6.1 Primary Interaction Patterns

#### Touch Interactions
```

typescript
interface TouchInteractions {
tap: {
action: 'select' | 'navigate' | 'confirm';
feedback: 'visual' | 'haptic';

duration: number; // milliseconds
};
longPress: {
action: 'context-menu' | 'quick-actions';
threshold: 500; // milliseconds
feedback: 'haptic' | 'visual';
};
swipe: {
direction: 'left' | 'right' | 'up' | 'down';
action: 'navigate' | 'dismiss' | 'refresh';
threshold: 50; // pixels
};
}

```
#### Voice Interactions
```

typescript
interface VoiceCommands {
navigation: [
'Go to properties',
'Show my clients',
'Open AI assistant'
];
actions: [
'Create new property',
'Generate content for [property]',
'Schedule follow-up with [client]'
];
queries: [
'What are my tasks for today?',
'Show me market trends',
'How is [property] performing?'

];
}

```
### 7.6.2 Gesture-Based Navigation

#### Swipe Gestures
- **Left Swipe**: Navigate forward, reveal actions
- **Right Swipe**: Navigate back, dismiss modals
- **Up Swipe**: Refresh content, reveal more options
- **Down Swipe**: Close modals, minimize screens

#### Multi-Touch Gestures
- **Pinch to Zoom**: Image viewing, map interaction
- **Two-Finger Scroll**: Navigate through content
- **Three-Finger Tap**: Quick actions menu

### 7.6.3 Accessibility Interactions
```

typescript
interface AccessibilityFeatures {
screenReader: {
labels: string;
hints: string;
roles: 'button' | 'text' | 'image' | 'list';
};
voiceOver: {
enabled: boolean;
customActions: string[];
};
dynamicType: {
supported: boolean;
scaleFactor: number;
};
highContrast: {
enabled: boolean;
colorAdjustments: ColorAdjustments;

};
}

```
## 7.7 VISUAL DESIGN CONSIDERATIONS

### 7.7.1 Mobile-First Design Principles

#### Responsive Layout System
```

typescript
interface BreakPoints {
mobile: { width: 375, height: 812 }; // iPhone 13 Pro
tablet: { width: 768, height: 1024 }; // iPad
desktop: { width: 1024, height: 768 }; // Future web version
}

interface SpacingSystem {
xs: 4;
sm: 8;
md: 16;
lg: 24;
xl: 32;
xxl: 48;
}

```
#### Component Sizing Guidelines
- **Minimum Touch Target**: 44x44 points (iOS) / 48x48 dp (Android)
- **Button Height**: 48-56 points for primary actions
- **Input Field Height**: 44-48 points with adequate padding
- **Card Spacing**: 16 points between cards, 24 points from edges

### 7.7.2 Visual Hierarchy

#### Typography Scale
```

typescript
const typographyScale = {
display: { size: 36, weight: 'bold', lineHeight: 44 },
h1: { size: 32, weight: 'bold', lineHeight: 40 },
h2: { size: 24, weight: 'bold', lineHeight: 32 },
h3: { size: 20, weight: '600', lineHeight: 28 },
body: { size: 16, weight: 'normal', lineHeight: 24 },
caption: { size: 14, weight: 'normal', lineHeight: 20 },
small: { size: 12, weight: 'normal', lineHeight: 16 }
};

```
#### Color Usage Guidelines
- **Primary Blue**: Main actions, navigation, property-related features
- **Secondary Green**: Success states, client-related features, positive
- **Accent Purple**: Content generation, creative features, AI-powered to
- **Warning Orange**: Attention needed, pending tasks, deadlines
- **Danger Red**: Errors, critical alerts, AI assistant (conversational)
- **Info Teal**: Analytics, reports, informational content

### 7.7.3 Animation and Micro-Interactions

#### Animation Specifications
```

typescript
interface AnimationConfig {
duration: {
fast: 200;
normal: 300;
slow: 500;
};
easing: {
easeIn: 'cubic-bezier(0.4, 0, 1, 1)';
easeOut: 'cubic-bezier(0, 0, 0.2, 1)';
easeInOut: 'cubic-bezier(0.4, 0, 0.2, 1)';
};

spring: {
tension: 300;
friction: 20;
};
}

```
#### Micro-Interaction Examples
- **Button Press**: Scale down to 0.95 with haptic feedback
- **Card Selection**: Subtle elevation increase with shadow
- **Loading States**: Skeleton screens with shimmer effect
- **Success Actions**: Checkmark animation with color transition
- **Error States**: Shake animation with color change

### 7.7.4 Dark Mode Support
```

typescript
interface ThemeColors {
light: {
background: '#ffffff';
surface: '#f8fafc';
text: '#1f2937';
textSecondary: '#6b7280';
border: '#e5e7eb';
};
dark: {
background: '#111827';
surface: '#1f2937';
text: '#f9fafb';
textSecondary: '#d1d5db';
border: '#374151';
};
}

```
### 7.7.5 Branding and Visual Identity
```

```
#### Logo and Brand Elements
- **Primary Logo**: PropertyPro AI wordmark with icon
- **Icon System**: Consistent iconography using React Native Vector Icons
- **Brand Colors**: Professional blue palette with accent colors
- **Typography**: System fonts (SF Pro on iOS, Roboto on Android)

#### Visual Consistency
- **Card Design**: Consistent border radius (12px), shadow, and padding
- **Button Styles**: Primary, secondary, outline, and ghost variants
- **Input Fields**: Consistent styling with focus states
- **Loading States**: Unified skeleton and spinner designs

This comprehensive UI design specification ensures PropertyPro AI delive

# 8. INFRASTRUCTURE

## 8.1 DEPLOYMENT ENVIRONMENT

### 8.1.1 Target Environment Assessment

PropertyPro AI is designed as a **containerized application** suitable fo

#### Environment Type Analysis

| Environment Type | Suitability | Justification | Implementation Approac
|---|---|---|---|
| Cloud-First | **Recommended** | Scalability, managed services, cost eff
| On-Premises | Supported | Data sovereignty, compliance requirements | D
| Hybrid | Supported | Gradual cloud migration, specific compliance needs
| Multi-Cloud | Future Consideration | Vendor lock-in avoidance, disaster

#### Geographic Distribution Requirements

| Requirement | Implementation | Technology | Justification |
|---|---|---|---|
| Single Region Deployment | Primary deployment region | Cloud provider r
| Multi-Region Support | Future enhancement | Container replication | Glo
| Edge Computing | Not required | N/A | Mobile-first architecture with lo
| CDN Integration | Recommended | Cloud CDN services | Static asset deliv

#### Resource Requirements

**Compute Resources:**
```

```
| Component | CPU Requirements | Memory Requirements | Storage Requiremen
|---|---|---|---|---|
| FastAPI Backend | 2-4 vCPUs | 4-8 GB RAM | 20 GB SSD | 1 Gbps |
| PostgreSQL Database | 2-4 vCPUs | 8-16 GB RAM | 100 GB SSD | 1 Gbps |
| React Native Build | 4-8 vCPUs | 8-16 GB RAM | 50 GB SSD | 1 Gbps |
| Load Balancer | 1-2 vCPUs | 2-4 GB RAM | 10 GB SSD | 10 Gbps |

**Scaling Considerations:**

- **Horizontal Scaling**: Container systems with Docker and Kubernetes ma
- **Auto-scaling**: Based on CPU utilization and request volume
- **Database Scaling**: Read replicas and connection pooling
- **Storage Scaling**: Elastic storage for property images and AI conten

#### Compliance and Regulatory Requirements

| Compliance Area | Requirement | Implementation | Monitoring |
|---|---|---|---|
| Data Protection | GDPR, CCPA compliance | Encryption at rest and in tra
| Real Estate Regulations | Industry-specific compliance | Data retentio
| Security Standards | SOC 2, ISO 27001 | Security controls and monitori
| Backup and Recovery | Business continuity | Automated backups and disa

### 8.1.2 Environment Management

#### Infrastructure as Code (IaC) Approach

PropertyPro AI utilizes **Docker Compose** for development and staging e

**Docker Compose Configuration:**
```

yaml

docker-compose.yml - Development and Staging

---

version: '3.8'

services:
FastAPI Backend Service
backend:
build:
context: ./backend
dockerfile: Dockerfile
ports:
- "8000:8000"
environment:
-
DATABASE_URL=postgresql://postgres:password@postgres:5432/propertypro_ai
- OPENAI_API_KEY=${OPENAI_API_KEY}
- JWT_SECRET=${JWT_SECRET}
- ENVIRONMENT=development
depends_on:
- postgres
- redis
volumes:
- ./backend:/app
- /app/node_modules
networks:
- propertypro-network

PostgreSQL Database Service

postgres:
image: postgres:15-alpine
environment:
- POSTGRES_DB=propertypro_ai
- POSTGRES_USER=postgres
- POSTGRES_PASSWORD=password
ports:
- "5432:5432"

volumes:
- postgres_data:/var/lib/postgresql/data
- ./database/init.sql:/docker-entrypoint-initdb.d/init.sql
networks:
- propertypro-network

Redis Cache Service

redis:
image: redis:7-alpine
ports:
- "6379:6379"
volumes:
- redis_data:/data
networks:
- propertypro-network

Nginx Reverse Proxy

nginx:
image: nginx:alpine
ports:
- "80:80"
- "443:443"
volumes:
- ./nginx/nginx.conf:/etc/nginx/nginx.conf
- ./nginx/ssl:/etc/nginx/ssl
depends_on:
- backend
networks:
- propertypro-network

volumes:
postgres_data:
redis_data:

networks:
propertypro-network:
driver: bridge

```
#### Configuration Management Strategy

| Configuration Type | Management Approach | Storage Method | Security Le
|---|---|---|---|
| Application Configuration | Environment variables | Docker environment
| Database Configuration | Docker Compose services | Encrypted volumes |
| API Keys and Secrets | External secret management | Kubernetes secrets
| Feature Flags | Environment-based configuration | Configuration files

#### Environment Promotion Strategy

<div class="mermaid-wrapper" id="mermaid-diagram-rop0f7y3h">
          <div class="mermaid">
graph LR
    subgraph &quot;Development Environment&quot;
        DEV[Local Development&lt;br/&gt;Docker Compose]
        DEV_DB[#40;Local PostgreSQL#41;]
        DEV_CACHE[Local Redis]
    end

    subgraph &quot;Staging Environment&quot;
        STAGE[Staging Deployment&lt;br/&gt;Docker Compose]
        STAGE_DB[#40;Staging PostgreSQL#41;]
        STAGE_CACHE[Staging Redis]
    end

    subgraph &quot;Production Environment&quot;
        PROD[Production Deployment&lt;br/&gt;Kubernetes/Docker]
        PROD_DB[#40;Production PostgreSQL#41;]
        PROD_CACHE[Production Redis]
        PROD_LB[Load Balancer]
    end

    DEV --&gt; STAGE
    STAGE --&gt; PROD

    DEV --&gt; DEV_DB
    DEV --&gt; DEV_CACHE
```

```
    STAGE --&gt; STAGE_DB
    STAGE --&gt; STAGE_CACHE

    PROD --&gt; PROD_DB
    PROD --&gt; PROD_CACHE
    PROD_LB --&gt; PROD

    style DEV fill:#e1f5fe
    style STAGE fill:#fff3e0
    style PROD fill:#c8e6c9
</div>
        </div>

#### Backup and Disaster Recovery Plans

**Backup Strategy:**

| Data Type | Backup Frequency | Retention Period | Storage Location | Re
|---|---|---|---|---|
| Database | Daily full, hourly incremental | 30 days full, 7 days increm
| Application Code | Continuous (Git) | Indefinite | Git repositories | <
| Configuration | Daily | 90 days | Encrypted cloud storage | < 1 hour |
| User Files | Daily | 30 days | Cloud storage with versioning | < 2 hour

**Disaster Recovery Implementation:**
```

bash
#!/bin/bash

disaster-recovery.sh - Automated disaster recovery script

Database Recovery

restore_database() {
echo "Restoring database from backup..."
docker run --rm -v postgres_backup:/backup \
postgres:15-alpine \
pg_restore -h $DB_HOST -U $DB_USER -d $DB_NAME /backup/latest.dump
}

Application Recovery

restore_application() {
echo "Restoring application from container registry..."
docker pull $CONTAINER_REGISTRY/propertypro-ai:latest
docker-compose up -d
}

Configuration Recovery

restore_configuration() {
echo "Restoring configuration from backup..."
aws s3 sync s3://$BACKUP_BUCKET/config/ ./config/
}

Execute recovery procedures

main() {
restore_configuration
restore_database
restore_application

```
echo "Disaster recovery completed successfully"
```

}

main "$@"

```
## 8.2 CONTAINERIZATION

### 8.2.1 Container Platform Selection

PropertyPro AI utilizes **Docker** as the primary containerization platf

#### Container Platform Justification

| Platform | Advantages | Disadvantages | Use Case |
|---|---|---|---|
| Docker | Industry standard, extensive ecosystem, excellent tooling | Re
| Podman | Rootless containers, Docker-compatible | Smaller ecosystem, ne
| containerd | Lightweight, Kubernetes native | Lower-level, less develo

### 8.2.2 Base Image Strategy

#### FastAPI Backend Container

**Dockerfile Implementation:**
```

dockerfile

backend/Dockerfile - Multi-stage build for FastAPI application

FROM python:3.11-slim as builder

Set environment variables

ENV PYTHONDONTWRITEBYTECODE=1 \
PYTHONUNBUFFERED=1 \
PIP_NO_CACHE_DIR=1 \
PIP_DISABLE_PIP_VERSION_CHECK=1

Install system dependencies

RUN apt-get update && apt-get install -y \
build-essential \
curl \
&& rm -rf /var/lib/apt/lists/*

Create and activate virtual environment

RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

Copy and install Python dependencies

COPY requirements.txt .
RUN pip install --upgrade pip && \
pip install -r requirements.txt

Production stage

FROM python:3.11-slim as production

Set environment variables

ENV PYTHONDONTWRITEBYTECODE=1 \
PYTHONUNBUFFERED=1 \
PATH="/opt/venv/bin:$PATH"

Install runtime dependencies

RUN apt-get update && apt-get install -y \
curl \
&& rm -rf /var/lib/apt/lists/* \
&& groupadd -r appuser && useradd -r -g appuser appuser

Copy virtual environment from builder stage

COPY --from=builder /opt/venv /opt/venv

Set working directory

WORKDIR /app

Copy application code

COPY --chown=appuser:appuser . .

Switch to non-root user

USER appuser

Health check

HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
CMD curl -f http://localhost:8000/health || exit 1

Expose port

EXPOSE 8000

Start application

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--workers", "4"]

```
#### React Native Build Container

**Note on React Native Containerization:**

React Native development in Docker containers faces challenges with mobi

1. **Build Environment Standardization**: Consistent build environments a
2. **CI/CD Pipeline Integration**: Automated testing and building in Git
3. **Development Environment Isolation**: Isolated development dependenc
```

dockerfile

frontend/Dockerfile - React Native build environment

FROM node:18-alpine as builder

Install system dependencies

RUN apk add --no-cache \
git \
python3 \
make \
g++ \
&& npm install -g @react-native-community/cli

Set working directory

WORKDIR /app

Copy package files

COPY package*.json ./

Install dependencies

RUN npm ci --only=production

Copy source code

COPY . .

Build application (for web deployment)

RUN npm run build:web

Production stage for web deployment

FROM nginx:alpine as production

Copy built application

COPY --from=builder /app/dist /usr/share/nginx/html

Copy nginx configuration

COPY nginx.conf /etc/nginx/nginx.conf

Expose port

EXPOSE 80

Start nginx

CMD ["nginx", "-g", "daemon off;"]

```
### 8.2.3 Image Versioning Approach

#### Semantic Versioning Strategy

| Version Type | Format | Trigger | Example | Use Case |
|---|---|---|---|---|
| Development | `dev-{commit-hash}` | Every commit | `dev-a1b2c3d` | Deve
| Feature Branch | `feature-{branch}-{hash}` | Feature branch push | `fea
| Release Candidate | `rc-{version}` | Pre-release tag | `rc-1.2.0` | Sta
| Production | `{major}.{minor}.{patch}` | Release tag | `1.2.0` | Produc

**Container Registry Strategy:**
```

bash

Container registry organization

registry.example.com/
├── propertypro-ai/
│ ├── backend:latest
│ ├── backend:1.2.0
│ ├── backend:rc-1.2.0
│ └── backend:dev-a1b2c3d
├── propertypro-ai-frontend/
│ ├── web:latest
│ ├── web:1.2.0
│ └── web:dev-a1b2c3d
└── propertypro-ai-nginx/
├── proxy:latest
└── proxy:1.2.0

```
### 8.2.4 Build Optimization Techniques

#### Multi-Stage Build Optimization

Taking care of the order of instructions in the Dockerfile and the Docke

**Build Optimization Strategies:**

| Technique | Implementation | Benefit | Impact |
|---|---|---|---|
| Layer Caching | Copy requirements before source code | Faster rebuilds
| Multi-stage Builds | Separate build and runtime stages | Smaller images
| Dependency Caching | Cache package installations | Faster dependency re
| Build Context Optimization | .dockerignore file | Faster context trans

**.dockerignore Configuration:**
```

dockerignore

.dockerignore - Optimize build context

node_modules
npm-debug.log*
.git
.gitignore
README.md
.env
.nyc_output
coverage
.nyc_output
.coverage
.pytest_cache
pycache
*.pyc
*.pyo
.pyd .Python env pip-log.txt pip-delete-this-directory.txt .tox .coverage .coverage.
.cache
nosetests.xml
coverage.xml
*.cover
*.log
.DS_Store
.vscode
.idea

```
### 8.2.5 Security Scanning Requirements

#### Container Security Implementation

| Security Layer | Tool/Technique | Frequency | Action on Failure |
```

```
|---|---|---|---|
| Base Image Scanning | Docker Scout, Trivy | Every build | Block deploym
| Dependency Scanning | npm audit, pip-audit | Daily | Create security ti
| Runtime Security | Falco, AppArmor | Continuous | Alert and investigate
| Compliance Scanning | CIS benchmarks | Weekly | Remediation planning |

**Security Scanning Integration:**
```

yaml

.github/workflows/security-scan.yml

name: Container Security Scan

on:
push:
branches: [main, develop]
pull_request:
branches: [main]

jobs:
security-scan:
runs-on: ubuntu-latest
steps:
- name: Checkout code
uses: actions/checkout@v4

```
  - name: Build Docker image
    run: docker build -t propertypro-ai:${{ github.sha }} .

  - name: Run Trivy vulnerability scanner
    uses: aquasecurity/trivy-action@master
    with:
      image-ref: 'propertypro-ai:${{ github.sha }}'
```

```
      format: 'sarif'
      output: 'trivy-results.sarif'

  - name: Upload Trivy scan results
    uses: github/codeql-action/upload-sarif@v2
    with:
      sarif_file: 'trivy-results.sarif'

  - name: Docker Scout scan
    uses: docker/scout-action@v1
    with:
      command: cves
      image: propertypro-ai:${{ github.sha }}
      only-severities: critical,high
      exit-code: true
```

```
## 8.3 CI/CD PIPELINE

### 8.3.1 Build Pipeline

#### Source Control Integration

PropertyPro AI implements a comprehensive CI/CD pipeline using **GitHub A

**Pipeline Architecture:**

<div class="mermaid-wrapper" id="mermaid-diagram-xhvt31pe8">
          <div class="mermaid">
graph TB
    subgraph &quot;Source Control&quot;
        GIT[Git Repository&lt;br/&gt;GitHub]
        PR[Pull Request]
        MAIN[Main Branch]
    end

    subgraph &quot;CI Pipeline&quot;
        LINT[Code Linting&lt;br/&gt;ESLint, Ruff]
        TEST[Unit Tests&lt;br/&gt;Jest, pytest]
        BUILD[Build Images&lt;br/&gt;Docker Build]
        SCAN[Security Scan&lt;br/&gt;Trivy, Scout]
    end
```

```
    subgraph &quot;CD Pipeline&quot;
        STAGE[Deploy to Staging&lt;br/&gt;Docker Compose]
        E2E[E2E Tests&lt;br/&gt;Automated Testing]
        PROD[Deploy to Production&lt;br/&gt;Blue-Green Deployment]
        MONITOR[Post-Deploy Monitoring&lt;br/&gt;Health Checks]
    end

    subgraph &quot;Artifact Storage&quot;
        REGISTRY[Container Registry&lt;br/&gt;Docker Hub/ECR]
        ARTIFACTS[Build Artifacts&lt;br/&gt;GitHub Packages]
    end

    GIT --&gt; PR
    PR --&gt; LINT
    LINT --&gt; TEST
    TEST --&gt; BUILD
    BUILD --&gt; SCAN
    SCAN --&gt; REGISTRY

    MAIN --&gt; STAGE
    STAGE --&gt; E2E
    E2E --&gt; PROD
    PROD --&gt; MONITOR

    BUILD --&gt; ARTIFACTS
    REGISTRY --&gt; STAGE
    REGISTRY --&gt; PROD

    style GIT fill:#e1f5fe
    style PROD fill:#c8e6c9
    style SCAN fill:#fff3e0
</div>
        </div>

#### Build Environment Requirements

**GitHub Actions Runner Configuration:**
```

yaml

.github/workflows/ci-cd.yml - Complete CI/CD Pipeline

name: PropertyPro AI CI/CD Pipeline

on:
push:
branches: [main, develop]
pull_request:
branches: [main]
release:
types: [published]

env:
REGISTRY: ghcr.io
IMAGE_NAME: ${{ github.repository }}

jobs:
Code Quality and Testing
quality-check:
runs-on: ubuntu-latest
strategy:
matrix:
component: [backend, frontend]

```
steps:
  - name: Checkout repository
    uses: actions/checkout@v4

  - name: Setup Node.js (Frontend)
    if: matrix.component == 'frontend'
    uses: actions/setup-node@v4
    with:
      node-version: '18'
      cache: 'npm'
```

```
      cache-dependency-path: frontend/package-lock.json

  - name: Setup Python (Backend)
    if: matrix.component == 'backend'
    uses: actions/setup-python@v4
    with:
      python-version: '3.11'
      cache: 'pip'
      cache-dependency-path: backend/requirements.txt

  - name: Install dependencies (Frontend)
    if: matrix.component == 'frontend'
    working-directory: frontend
    run: npm ci

  - name: Install dependencies (Backend)
    if: matrix.component == 'backend'
    working-directory: backend
    run: |
      python -m pip install --upgrade pip
      pip install -r requirements.txt
      pip install -r requirements-dev.txt

  - name: Lint code (Frontend)
    if: matrix.component == 'frontend'
    working-directory: frontend
    run: |
      npm run lint
      npm run type-check

  - name: Lint code (Backend)
    if: matrix.component == 'backend'
    working-directory: backend
    run: |
      ruff check .
      black --check .
      mypy .

  - name: Run tests (Frontend)
    if: matrix.component == 'frontend'
    working-directory: frontend
    run: npm run test:ci
```

```
  - name: Run tests (Backend)
    if: matrix.component == 'backend'
    working-directory: backend
    run: |
      pytest --cov=app --cov-report=xml --cov-report=html
    env:
      DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test_dl

  - name: Upload coverage reports
    uses: codecov/codecov-action@v3
    with:
      file: ./${{ matrix.component }}/coverage.xml
      flags: ${{ matrix.component }}
```

Build and Push Container Images

build-and-push:

needs: quality-check

runs-on: ubuntu-latest

if: github.event_name != 'pull_request'

```
permissions:
  contents: read
  packages: write

strategy:
  matrix:
    component: [backend, frontend]

steps:
  - name: Checkout repository
    uses: actions/checkout@v4

  - name: Log in to Container Registry
    uses: docker/login-action@v3
    with:
      registry: ${{ env.REGISTRY }}
      username: ${{ github.actor }}
      password: ${{ secrets.GITHUB_TOKEN }}
```

```
  - name: Extract metadata
    id: meta
    uses: docker/metadata-action@v5
    with:
      images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-${{ matrix.compor
      tags: |
        type=ref,event=branch
        type=ref,event=pr
        type=semver,pattern={{version}}
        type=semver,pattern={{major}}.{{minor}}
        type=sha,prefix={{branch}}-

  - name: Build and push Docker image
    uses: docker/build-push-action@v5
    with:
      context: ./${{ matrix.component }}
      push: true
      tags: ${{ steps.meta.outputs.tags }}
      labels: ${{ steps.meta.outputs.labels }}
      cache-from: type=gha
      cache-to: type=gha,mode=max

  - name: Run security scan
    uses: aquasecurity/trivy-action@master
    with:
      image-ref: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-${{ matrix.cor
      format: 'sarif'
      output: 'trivy-results-${{ matrix.component }}.sarif'

  - name: Upload security scan results
    uses: github/codeql-action/upload-sarif@v2
    with:
      sarif_file: 'trivy-results-${{ matrix.component }}.sarif'
```

```
#### Dependency Management

| Component | Package Manager | Lock File | Cache Strategy |
|---|---|---|---|
| FastAPI Backend | pip | requirements.txt | pip cache |
| React Native Frontend | npm | package-lock.json | npm cache |
| Development Tools | pip, npm | requirements-dev.txt, package-dev.json
| Container Dependencies | Docker | Dockerfile | Docker layer cache |
```

```
#### Artifact Generation and Storage

**Artifact Management Strategy:**
```

yaml

Artifact storage configuration

artifacts:
docker-images:
registry: ghcr.io
retention: 30 days
cleanup-policy: keep-last-10

build-artifacts:
storage: github-packages
retention: 90 days

test-reports:
storage: github-actions-artifacts
retention: 30 days

security-reports:
storage: github-security-tab
retention: 365 days

```
#### Quality Gates

| Gate Type | Criteria | Action on Failure | Override Policy |
|---|---|---|---|
| Code Coverage | >85% for backend, >80% for frontend | Block merge | Adr
| Security Scan | No critical vulnerabilities | Block deployment | Securi
| Performance Tests | <2s API response time | Block deployment | Performa
| Integration Tests | 100% pass rate | Block deployment | No override |
```

```
### 8.3.2 Deployment Pipeline

#### Deployment Strategy Selection

PropertyPro AI implements **Blue-Green Deployment** for production releas

**Deployment Strategies Comparison:**

| Strategy | Downtime | Rollback Speed | Resource Usage | Complexity | Us
|---|---|---|---|---|---|
| Blue-Green | Zero | Instant | 2x resources | Medium | Production releas
| Rolling Update | Minimal | Medium | 1.2x resources | Low | Minor update
| Canary | Zero | Fast | 1.1x resources | High | High-risk changes |
| Recreate | High | Slow | 1x resources | Low | Development only |

#### Environment Promotion Workflow
```

yaml

.github/workflows/deploy.yml - Deployment Pipeline

name: Deploy to Environments

on:
workflow_run:
workflows: ["PropertyPro AI CI/CD Pipeline"]
types: [completed]
branches: [main]

jobs:
deploy-staging:
if: ${{ github.event.workflow_run.conclusion == 'success' }}
runs-on: ubuntu-latest
environment: staging

```
steps:
  - name: Checkout repository
    uses: actions/checkout@v4

  - name: Deploy to staging
    uses: appleboy/ssh-action@v1.0.0
    with:
      host: ${{ secrets.STAGING_HOST }}
      username: ${{ secrets.STAGING_USER }}
      key: ${{ secrets.STAGING_SSH_KEY }}
      script: |
        cd /opt/propertypro-ai
        docker-compose pull
        docker-compose up -d --remove-orphans
        docker system prune -f

  - name: Run health checks
    run: |
      sleep 30
      curl -f ${{ secrets.STAGING_URL }}/health || exit 1

  - name: Run smoke tests
    run: |
      npm run test:smoke -- --baseUrl=${{ secrets.STAGING_URL }}
```

deploy-production:
needs: deploy-staging
runs-on: ubuntu-latest
environment: production
if: github.ref == 'refs/heads/main'

```
steps:
  - name: Checkout repository
    uses: actions/checkout@v4

  - name: Blue-Green Deployment
    uses: appleboy/ssh-action@v1.0.0
    with:
      host: ${{ secrets.PRODUCTION_HOST }}
      username: ${{ secrets.PRODUCTION_USER }}
```

```
        key: ${{ secrets.PRODUCTION_SSH_KEY }}
        script: |
          cd /opt/propertypro-ai

          # Determine current and next environments
          CURRENT=$(docker-compose ps -q | head -1 | xargs docker inspect -
          NEXT=$([ "$CURRENT" = "blue" ] && echo "green" || echo "blue")

          echo "Current environment: $CURRENT"
          echo "Deploying to: $NEXT"

          # Deploy to next environment
          ENVIRONMENT=$NEXT docker-compose -f docker-compose.prod.yml up -d

          # Health check
          sleep 30
          if curl -f http://localhost:8000/health; then
            echo "Health check passed, switching traffic"

            # Update load balancer to point to new environment
            sed -i "s/environment=$CURRENT/environment=$NEXT/g" nginx/nginx
            docker-compose restart nginx

            # Stop old environment
            sleep 10
            ENVIRONMENT=$CURRENT docker-compose -f docker-compose.prod.yml

            echo "Deployment completed successfully"
          else
            echo "Health check failed, rolling back"
            ENVIRONMENT=$NEXT docker-compose -f docker-compose.prod.yml dow
            exit 1
          fi

    - name: Post-deployment verification
      run: |
        sleep 60
        curl -f ${{ secrets.PRODUCTION_URL }}/health
        npm run test:e2e -- --baseUrl=${{ secrets.PRODUCTION_URL }}

    - name: Notify deployment success
      uses: 8398a7/action-slack@v3
      with:
```

```
      status: success
      text: "PropertyPro AI deployed successfully to production"
    env:
      SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK }}
```

```
#### Rollback Procedures

**Automated Rollback Implementation:**
```

bash
#!/bin/bash

rollback.sh - Automated rollback script

ENVIRONMENT=${1:-production}
ROLLBACK_VERSION=${2:-previous}

echo "Initiating rollback for $ENVIRONMENT to $ROLLBACK_VERSION"

Get previous version

if ["$ROLLBACK_VERSION" = "previous"]; then
ROLLBACK_VERSION=$(docker images --format "table {{.Tag}}" propertypro-ai | grep -v latest | head -2 | tail -1)
fi

echo "Rolling back to version: $ROLLBACK_VERSION"

Blue-Green rollback

CURRENT_ENV=$(docker-compose ps -q | head -1 | xargs docker inspect --format='{{.Config.Labels.environment}}')

ROLLBACK_ENV=$(["$CURRENT_ENV" = "blue"] && echo "green" || echo "blue")

Deploy rollback version

ENVIRONMENT=$ROLLBACK_ENV IMAGE_TAG=$ROLLBACK_VERSION docker-compose -f docker-compose.prod.yml up -d

Health check

sleep 30
if curl -f http://localhost:8000/health; then
echo "Rollback health check passed, switching traffic"

Update load balancer

```
sed -i "s/environment=$CURRENT_ENV/environment=$ROLLBACK_ENV/g" nginx/ng
docker-compose restart nginx
```

Stop failed environment

```
ENVIRONMENT=$CURRENT_ENV docker-compose -f docker-compose.prod.yml down

echo "Rollback completed successfully"
```

else
echo "Rollback failed, manual intervention required"
exit 1
fi

```
#### Post-Deployment Validation

| Validation Type | Implementation | Success Criteria | Failure Action |
|---|---|---|---|
| Health Checks | HTTP endpoint monitoring | 200 OK response | Automatic
```

```
| Smoke Tests | Critical path testing | All tests pass | Deployment block
| Performance Tests | Load testing | <2s response time | Performance ale
| Integration Tests | End-to-end testing | 100% pass rate | Rollback cons

### 8.3.3 Release Management Process

#### Release Workflow

<div class="mermaid-wrapper" id="mermaid-diagram-r4w2j5hca">
          <div class="mermaid">
graph TB
    subgraph &quot;Development Phase&quot;
        DEV[Feature Development]
        PR[Pull Request]
        REVIEW[Code Review]
    end

    subgraph &quot;Testing Phase&quot;
        CI[CI Pipeline]
        STAGE[Staging Deployment]
        QA[QA Testing]
    end

    subgraph &quot;Release Phase&quot;
        TAG[Release Tag]
        PROD[Production Deployment]
        MONITOR[Monitoring]
    end

    subgraph &quot;Rollback Phase&quot;
        ISSUE[Issue Detection]
        ROLLBACK[Automated Rollback]
        HOTFIX[Hotfix Development]
    end

    DEV --&gt; PR
    PR --&gt; REVIEW
    REVIEW --&gt; CI
    CI --&gt; STAGE
    STAGE --&gt; QA
    QA --&gt; TAG
    TAG --&gt; PROD
    PROD --&gt; MONITOR
```

```
    MONITOR --&gt; ISSUE
    ISSUE --&gt; ROLLBACK
    ISSUE --&gt; HOTFIX
    HOTFIX --&gt; PR

    style DEV fill:#e1f5fe
    style PROD fill:#c8e6c9
    style ROLLBACK fill:#ffcdd2
</div>
        </div>

## 8.4 INFRASTRUCTURE MONITORING

### 8.4.1 Resource Monitoring Approach

PropertyPro AI implements comprehensive infrastructure monitoring using c

#### Monitoring Stack Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-4kxkwbz01">
          <div class="mermaid">
graph TB
    subgraph &quot;Application Layer&quot;
        APP[PropertyPro AI&lt;br/&gt;FastAPI + React Native]
        METRICS[Application Metrics&lt;br/&gt;Prometheus Client]
    end

    subgraph &quot;Infrastructure Layer&quot;
        DOCKER[Docker Containers]
        HOST[Host System]
        NETWORK[Network Layer]
    end

    subgraph &quot;Monitoring Stack&quot;
        PROMETHEUS[Prometheus&lt;br/&gt;Metrics Collection]
        GRAFANA[Grafana&lt;br/&gt;Visualization]
        ALERTMANAGER[AlertManager&lt;br/&gt;Notifications]
        LOKI[Loki&lt;br/&gt;Log Aggregation]
    end

    subgraph &quot;Storage Layer&quot;
        TSDB[Time Series DB&lt;br/&gt;Prometheus Storage]
```

```
        LOGS[Log Storage&lt;br/&gt;Loki Storage]
    end

    APP --&gt; METRICS
    DOCKER --&gt; PROMETHEUS
    HOST --&gt; PROMETHEUS
    NETWORK --&gt; PROMETHEUS
    METRICS --&gt; PROMETHEUS

    PROMETHEUS --&gt; GRAFANA
    PROMETHEUS --&gt; ALERTMANAGER
    PROMETHEUS --&gt; TSDB

    APP --&gt; LOKI
    DOCKER --&gt; LOKI
    LOKI --&gt; LOGS

    style APP fill:#e1f5fe
    style PROMETHEUS fill:#fff3e0
    style GRAFANA fill:#c8e6c9
</div>
        </div>

#### Resource Monitoring Configuration

**Docker Compose Monitoring Stack:**
```

yaml

monitoring/docker-compose.monitoring.yml

---

version: '3.8'

services:
Prometheus - Metrics Collection
prometheus:

image: prom/prometheus:latest
container_name: prometheus
ports:
- "9090:9090"
volumes:
- ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
- ./prometheus/rules:/etc/prometheus/rules
- prometheus_data:/prometheus
command:
- '--config.file=/etc/prometheus/prometheus.yml'
- '--storage.tsdb.path=/prometheus'
- '--web.console.libraries=/etc/prometheus/console_libraries'
- '--web.console.templates=/etc/prometheus/consoles'
- '--storage.tsdb.retention.time=30d'
- '--web.enable-lifecycle'
networks:
- monitoring

Grafana - Visualization

grafana:
image: grafana/grafana:latest
container_name: grafana
ports:
- "3000:3000"
environment:
- GF_SECURITY_ADMIN_PASSWORD=admin123
- GF_USERS_ALLOW_SIGN_UP=false
volumes:
- grafana_data:/var/lib/grafana
- ./grafana/dashboards:/etc/grafana/provisioning/dashboards
- ./grafana/datasources:/etc/grafana/provisioning/datasources
networks:
- monitoring

AlertManager - Notifications

alertmanager:
image: prom/alertmanager:latest
container_name: alertmanager
ports:
- "9093:9093"
volumes:
- ./alertmanager/alertmanager.yml:/etc/alertmanager/alertmanager.yml
- alertmanager_data:/alertmanager
networks:
- monitoring

Node Exporter - Host Metrics

node-exporter:
image: prom/node-exporter:latest
container_name: node-exporter
ports:
- "9100:9100"
volumes:
- /proc:/host/proc:ro
- /sys:/host/sys:ro
- /:/rootfs:ro
command:
- '--path.procfs=/host/proc'
- '--path.rootfs=/rootfs'
- '--path.sysfs=/host/sys'
- '--collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($$|/)'
networks:
- monitoring

cAdvisor - Container Metrics

cadvisor:
image: gcr.io/cadvisor/cadvisor:latest
container_name: cadvisor
ports:
- "8080:8080"
volumes:
- /:/rootfs:ro
- /var/run:/var/run:ro
- /sys:/sys:ro
- /var/lib/docker/:/var/lib/docker:ro
- /dev/disk/:/dev/disk:ro
privileged: true
devices:
- /dev/kmsg
networks:
- monitoring

volumes:
prometheus_data:
grafana_data:
alertmanager_data:

networks:
monitoring:
driver: bridge

```
### 8.4.2 Performance Metrics Collection

#### Key Performance Indicators

| Metric Category | Metrics | Collection Method | Alert Threshold |
|---|---|---|---|
| Application Performance | Response time, throughput, error rate | Fast/
| Container Resources | CPU, memory, disk usage | cAdvisor | >80% CPU, >8
| Database Performance | Connection count, query time | PostgreSQL export
| Network Performance | Bandwidth, latency, packet loss | Node exporter
```

```
**Prometheus Configuration:**
```

yaml

prometheus/prometheus.yml

global:
scrape_interval: 15s
evaluation_interval: 15s

rule_files:

- "rules/*.yml"

alerting:
alertmanagers:
- static_configs:
- targets:
- alertmanager:9093

scrape_configs:
PropertyPro AI Application

- job_name: 'propertypro-ai' static_configs:
    - targets: ['backend:8000']
      metrics_path: '/metrics'
      scrape_interval: 10s

Node Exporter - Host Metrics

- job_name: 'node-exporter' static_configs:
    - targets: ['node-exporter:9100']

cAdvisor - Container Metrics

- job_name: 'cadvisor' static_configs:
  - targets: ['cadvisor:8080']

PostgreSQL Database

- job_name: 'postgres' static_configs:
  - targets: ['postgres-exporter:9187']

Redis Cache

- job_name: 'redis' static_configs:
  - targets: ['redis-exporter:9121']

```
### 8.4.3 Cost Monitoring and Optimization

#### Infrastructure Cost Tracking

| Resource Type | Cost Driver | Monitoring Method | Optimization Strategy
|---|---|---|---|
| Compute Resources | CPU hours, memory usage | Cloud provider APIs | Au
| Storage | Data volume, IOPS | Storage metrics | Data lifecycle manageme
| Network | Data transfer, bandwidth | Network monitoring | CDN usage, co
| External Services | API calls, usage | Service-specific metrics | Rate

**Cost Optimization Dashboard:**
```

json
{
"dashboard": {
"title": "PropertyPro AI Cost Monitoring",
"panels": [
{
"title": "Monthly Infrastructure Cost",
"type": "stat",
"targets": [
{
"expr": "sum(rate(infrastructure_cost_total[30d])) * 30 * 24 * 3600"

}
]
},
{
"title": "Cost per User",
"type": "stat",
"targets": [
{
"expr": "sum(rate(infrastructure_cost_total[1h])) / sum(active_users)"
}
]
},
{
"title": "Resource Utilization",
"type": "graph",
"targets": [
{
"expr": "avg(rate(container_cpu_usage_seconds_total[5m])) * 100"
},
{
"expr": "avg(container_memory_usage_bytes / container_spec_memory_limit_bytes) * 100"
}
]
}
]
}
}

```
### 8.4.4 Security Monitoring

#### Security Metrics and Alerts

| Security Domain | Metrics | Detection Method | Response Action |
```

```
|---|---|---|---|
| Authentication | Failed login attempts, unusual access patterns | Log a
| API Security | Rate limit violations, suspicious requests | API gateway
| Container Security | Vulnerability scans, runtime anomalies | Security
| Network Security | Unusual traffic patterns, port scans | Network moni

**Security Monitoring Configuration:**
```

yaml

alertmanager/alertmanager.yml

global:
smtp_smarthost: 'localhost:587'
smtp_from: 'alerts@propertypro-ai.com'

route:
group_by: ['alertname']
group_wait: 10s
group_interval: 10s
repeat_interval: 1h
receiver: 'web.hook'
routes:

- match:
  severity: critical
  receiver: 'critical-alerts'
- match:
  severity: warning
  receiver: 'warning-alerts'

receivers:

- name: 'web.hook'
  webhook_configs:
  - url: 'http://127.0.0.1:5001/'
- name: 'critical-alerts'
  email_configs:
  - to: 'security@propertypro-ai.com'
    subject: 'CRITICAL: PropertyPro AI Security Alert'
    body: |
    Alert: {{ .GroupLabels.alertname }}
    Severity: {{ .CommonLabels.severity }}
    Description: {{ .CommonAnnotations.description }}

  slack_configs:
  - api_url: '{{ .SlackWebhookURL }}'
    channel: '#security-alerts'
    title: 'Critical Security Alert'
    text: '{{ .CommonAnnotations.description }}'
- name: 'warning-alerts'
  email_configs:
  - to: 'ops@propertypro-ai.com'
    subject: 'WARNING: PropertyPro AI Alert'

```
### 8.4.5 Compliance Auditing

#### Audit Trail Implementation

| Audit Category | Data Collected | Retention Period | Compliance Require
|---|---|---|---|
| User Access | Login/logout events, permission changes | 7 years | SOX,
| Data Access | Database queries, file access | 3 years | GDPR, CCPA |
| System Changes | Configuration changes, deployments | 5 years | SOC 2,
| Security Events | Failed authentications, security alerts | 7 years | 
```

```
**Compliance Monitoring Dashboard:**
```

yaml

grafana/dashboards/compliance-dashboard.json

{
"dashboard": {
"title": "PropertyPro AI Compliance Dashboard",
"panels": [
{
"title": "User Access Events",
"type": "table",
"targets": [
{
"expr": "increase(user_login_total[24h])",
"legendFormat": "Successful Logins"
},
{
"expr": "increase(user_login_failed_total[24h])",
"legendFormat": "Failed Logins"
}
]
},
{
"title": "Data Access Patterns",
"type": "heatmap",
"targets": [
{
"expr": "rate(database_queries_total[1h])"

}
]
},
{
"title": "Security Compliance Score",
"type": "gauge",
"targets": [
{
"expr": "security_compliance_score"
}
]
}
]
}
}

```
## 8.5 INFRASTRUCTURE COST ESTIMATES

### 8.5.1 Development Environment Costs

| Resource | Specification | Monthly Cost | Annual Cost | Justification
|---|---|---|---|---|
| Development Servers | 2x 4 vCPU, 8GB RAM | $200 | $2,400 | Local develo
| CI/CD Infrastructure | GitHub Actions (2000 minutes) | $50 | $600 | Au
| Container Registry | GitHub Packages (50GB) | $25 | $300 | Docker image
| **Total Development** | | **$275** | **$3,300** | |

### 8.5.2 Production Environment Costs

| Resource | Specification | Monthly Cost | Annual Cost | Justification
|---|---|---|---|---|
| Application Servers | 3x 4 vCPU, 16GB RAM | $600 | $7,200 | High availa
| Database Server | 2x 8 vCPU, 32GB RAM | $800 | $9,600 | PostgreSQL with
| Load Balancer | Managed load balancer | $100 | $1,200 | Traffic distrib
| Storage | 500GB SSD + 1TB backup | $150 | $1,800 | Database and file st
| Monitoring | Prometheus + Grafana stack | $100 | $1,200 | Infrastructur
| **Total Production** | | **$1,750** | **$21,000** | |
```

```
### 8.5.3 External Service Costs

| Service | Usage Estimate | Monthly Cost | Annual Cost | Justification
|---|---|---|---|---|
| OpenAI GPT-4.1 API | 1M tokens/month | $300 | $3,600 | AI content gene
| Email Service | 10,000 emails/month | $50 | $600 | Client communicatio
| CDN | 100GB transfer/month | $25 | $300 | Static asset delivery |
| SSL Certificates | Wildcard certificate | $10 | $120 | HTTPS security
| **Total External Services** | | **$385** | **$4,620** | |

### 8.5.4 Total Infrastructure Investment

| Environment | Monthly Cost | Annual Cost | Percentage |
|---|---|---|---|
| Development | $275 | $3,300 | 11% |
| Production | $1,750 | $21,000 | 73% |
| External Services | $385 | $4,620 | 16% |
| **Total Infrastructure** | **$2,410** | **$28,920** | **100%** |

### 8.5.5 Cost Optimization Strategies

| Optimization Strategy | Potential Savings | Implementation Effort | Ti
|---|---|---|---|
| Auto-scaling implementation | 20-30% compute costs | Medium | 2-3 mont
| Reserved instance pricing | 30-40% compute costs | Low | 1 month |
| Database optimization | 15-25% database costs | High | 3-6 months |
| CDN optimization | 40-50% bandwidth costs | Low | 1 month |

This comprehensive infrastructure specification provides PropertyPro AI

# APPENDICES

## A.1 ADDITIONAL TECHNICAL INFORMATION

### A.1.1 React Native 0.71+ TypeScript Integration

React Native 0.71+ includes built-in, more accurate TypeScript declarati

#### Key TypeScript Enhancements

| Feature | Implementation | Benefit |
|---|---|---|
| Built-in Type Declarations | Types updated in lockstep with React Nati
```

```
| Flexbox Gap Support | Flexbox properties gap, rowGap, and columnGap sup
| Web-inspired Props | New prop aliases like src for Image component sou

#### Migration Considerations

After upgrading to React Native 0.71, it is recommended to remove @types/

### A.1.2 FastAPI Python 3.11+ Performance Optimizations

FastAPI works asynchronously and is extremely fast, with performance opt

#### Async Performance Benefits

| Performance Aspect | Improvement | Implementation |
|---|---|---|
| Concurrency Handling | Better performance, scalability, and responsive
| Request Processing | Very high performance, on par with NodeJS and Go,
| Development Speed | Increase development speed by 200% to 300%, reduce

#### Async vs Sync Decision Matrix

Non-blocking IO operations include database calls, API requests, file ope

<div class="mermaid-wrapper" id="mermaid-diagram-h099vg48k">
          <div class="mermaid">
flowchart TD
    A[Operation Type] --&gt; B{I/O Bound?}
    B --&gt;|Yes| C[Use async/await]
    B --&gt;|No| D{CPU Intensive?}
    D --&gt;|Yes| E[Use sync with executor]
    D --&gt;|No| F[Use sync]

    C --&gt; G[Database queries, API calls, File operations]
    E --&gt; H[Mathematical computations, Data processing]
    F --&gt; I[Simple operations, Health checks]
</div>
        </div>

### A.1.3 GPT-4.1 API Advanced Capabilities

GPT-4.1, GPT-4.1 mini, and GPT-4.1 nano outperform GPT-4o across the boa

## GPT-4.1 Model Comparison
```

```
| Model Variant | Context Window | Use Case | Performance Characteristics
|---|---|---|---|
| GPT-4.1 | 1 million tokens | Complex reasoning, large document analysis
| GPT-4.1 Mini | 1 million tokens | 50% latency reduction, 83% cost reduc
| GPT-4.1 Nano | 1 million tokens despite small size | Fastest and cheape

#### Context Window Performance

GPT-4.1 outperforms GPT-4o at context lengths up to 128K tokens and main

### A.1.4 Container Orchestration Patterns

PropertyPro AI implements containerization strategies optimized for deve

#### Multi-Stage Build Optimization
```

dockerfile

Optimized multi-stage build for FastAPI

FROM python:3.11-slim as builder
ENV PYTHONDONTWRITEBYTECODE=1 \
PYTHONUNBUFFERED=1 \
PIP_NO_CACHE_DIR=1

Install build dependencies

RUN apt-get update && apt-get install -y build-essential

Create virtual environment

RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

Install Python dependencies

COPY requirements.txt .
RUN pip install --upgrade pip && pip install -r requirements.txt

Production stage

FROM python:3.11-slim as production
ENV PATH="/opt/venv/bin:$PATH"

Copy virtual environment from builder

COPY --from=builder /opt/venv /opt/venv

Create non-root user

RUN groupadd -r appuser && useradd -r -g appuser appuser

WORKDIR /app
COPY --chown=appuser:appuser . .
USER appuser

EXPOSE 8000
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

```
#### Service Orchestration Architecture

<div class="mermaid-wrapper" id="mermaid-diagram-krari9r76">
          <div class="mermaid">
graph TB
    subgraph &quot;Development Environment&quot;
        DEV_API[FastAPI Development&lt;br/&gt;Hot Reload]
        DEV_DB[#40;PostgreSQL&lt;br/&gt;Development#41;]
        DEV_CACHE[Redis Cache&lt;br/&gt;Development]
    end

    subgraph &quot;Production Environment&quot;
        PROD_LB[Load Balancer&lt;br/&gt;Nginx]
        PROD_API1[FastAPI Instance 1]
```

```
          PROD_API2[FastAPI Instance 2]
          PROD_DB[#40;PostgreSQL&lt;br/&gt;Primary + Replica#41;]
          PROD_CACHE[Redis Cluster]
      end

      DEV_API --&gt; DEV_DB
      DEV_API --&gt; DEV_CACHE

      PROD_LB --&gt; PROD_API1
      PROD_LB --&gt; PROD_API2
      PROD_API1 --&gt; PROD_DB
      PROD_API2 --&gt; PROD_DB
      PROD_API1 --&gt; PROD_CACHE
      PROD_API2 --&gt; PROD_CACHE
  </div>
          </div>

  ### A.1.5 Security Implementation Patterns

  #### JWT Token Management Strategy
```

typescript

```
interface TokenManagement {
accessToken: {
expiry: '24 hours';
storage: 'React Native secure storage';
rotation: 'Automatic on refresh';
};
refreshToken: {
expiry: '30 days';
storage: 'Encrypted device storage';
rotation: 'On each use';
};
validation: {
signature: 'HMAC-SHA256';
claims: 'User ID, role, expiration';
revocation: 'Server-side blacklist';
```

};
}

```
#### Data Encryption Architecture

| Data Type | Encryption Method | Key Management | Access Control |
|---|---|---|---|
| User Credentials | bcrypt with salt | Server-side hashing | Authenticat
| API Communications | TLS 1.3 | Certificate-based | HTTPS enforcement |
| Database Records | AES-256-GCM | Environment variables | Role-based acc
| File Storage | AES-256-CBC | Derived keys | Resource ownership |

### A.1.6 Performance Monitoring Implementation

#### Metrics Collection Strategy
```

python
from prometheus_client import Counter, Histogram, Gauge
import time
from functools import wraps

Business metrics

property_operations = Counter(
'property_operations_total',
'Total property operations',
['operation_type', 'status']
)

ai_generation_time = Histogram(
'ai_content_generation_seconds',
'Time spent generating AI content',
['content_type', 'model_version']
)

active_users = Gauge(
'active_users_current',

```
'Current number of active users'
)

def monitor_performance(operation_type: str):
"""Decorator for monitoring operation performance."""
def decorator(func):
@wraps(func)
async def wrapper(*args, *kwargs): start_time = time.time() try: result =
await func(args, **kwargs)
property_operations.labels(
operation_type=operation_type,
status='success'
).inc()
return result
except Exception as e:
property_operations.labels(
operation_type=operation_type,
status='error'
).inc()
raise
finally:
duration = time.time() - start_time
ai_generation_time.labels(
content_type=operation_type,
model_version='gpt-4.1'
).observe(duration)
return wrapper
return decorator
```

A.2 GLOSSARY

Term	Definition
API Gateway	A server that acts as an API front-end, receiving API requests, enforcing throttling and security policies, passing requests to the back-end service, and passing the response back to the requester
Async/Await	Programming pattern that allows asynchronous, non-blocking code execution using coroutines in Python and JavaScript
Circuit Breaker	Design pattern used to detect failures and encapsulates the logic of preventing a failure from constantly recurring during maintenance, temporary external system failure, or unexpected system difficulties
Clean Architecture	Software design philosophy that separates the elements of a design into ring levels, with the main rule that code dependencies can only point inwards
Container Orchestration	The automated arrangement, coordination, and management of software containers using tools like Docker Compose or Kubernetes
CRUD Operations	Create, Read, Update, Delete - the four basic functions of persistent storage in database applications
Dependency Injection	Design pattern in which an object receives other objects that it depends on, rather than creating them internally
Event-Driven Architecture	Software architecture paradigm promoting the production, detection, consumption of, and reaction to events
Hexagonal Architecture	Architectural pattern that aims at creating loosely coupled application components that can be easily connected to their software environment by means of ports and adapters
JWT (JSON Web Token)	Open standard for securely transmitting information between parties as a JSON object, commonly used for authentication
Microservices	Architectural style that structures an application as a collection of loosely coupled services

Term	Definition
ORM (Object-Relational Mapping)	Programming technique for converting data between incompatible type systems using object-oriented programming languages
Rate Limiting	Strategy for limiting network traffic by restricting the number of requests a user can make in a given time period
Repository Pattern	Design pattern that encapsulates the logic needed to access data sources, centralizing common data access functionality
RESTful API	Architectural style for designing networked applications based on representational state transfer principles
Service Layer	Layer in software architecture that defines an application's boundary and its set of available operations from the perspective of interfacing client layers
State Management	The practice of managing the state of user interfaces in a declarative way, particularly in React applications
Type Safety	Programming language feature that prevents type errors by ensuring operations are performed on compatible data types

A.3 ACRONYMS

Acronym	Expanded Form
AI	Artificial Intelligence
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
CCPA	California Consumer Privacy Act
CDN	Content Delivery Network
CI/CD	Continuous Integration/Continuous Deployment
CMA	Comparative Market Analysis

Acronym	Expanded Form
CORS	Cross-Origin Resource Sharing
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
DDD	Domain-Driven Design
DNS	Domain Name System
E2E	End-to-End
GDPR	General Data Protection Regulation
GPT	Generative Pre-trained Transformer
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaC	Infrastructure as Code
IDE	Integrated Development Environment
IoC	Inversion of Control
JSON	JavaScript Object Notation
JSONB	JSON Binary (PostgreSQL data type)
JSI	JavaScript Interface (React Native)
JWT	JSON Web Token
KPI	Key Performance Indicator
LLM	Large Language Model
MLS	Multiple Listing Service
MVP	Minimum Viable Product
ORM	Object-Relational Mapping
PII	Personally Identifiable Information
RBAC	Role-Based Access Control
REST	Representational State Transfer

Acronym	Expanded Form
RTO	Recovery Time Objective
RPO	Recovery Point Objective
SDK	Software Development Kit
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SOC	Service Organization Control
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TTL	Time To Live
UI	User Interface
UUID	Universally Unique Identifier
UX	User Experience
WSGI	Web Server Gateway Interface
XSS	Cross-Site Scripting