Requirements Specification for InvestRand Platform

Document Control

- **Version:** 1.0
- **Project:** InvestRand Property Investment Platform
- **Geographic Focus:** South Africa
- **Primary Currency:** ZAR (South African Rand)

Executive Summary

Project Overview

InvestRand is a comprehensive property investment platform that enables users to discover, list, manage, and invest in rental properties across South Africa. The platform facilitates the entire property investment lifecycle from initial listing through to portfolio management, with a specific focus on multi-let and student accommodation investment strategies.

Business Context

The platform addresses the need for a centralized system that connects property investors, sourcing agents, service providers, and administrative personnel in managing rental property investments. It streamlines property listing workflows, financial modeling, approval processes, and contract executions while providing transparency and automated calculations for investment returns.

Key Stakeholders

1. **Property Investors** - End users seeking investment opportunities
2. **Sourcing Agents** - Professionals who list and source investment properties
3. **Buyers Representatives (Renovate)** - Professionals managing property modifications
4. **Service Providers** - Third-party contractors and service professionals
5. **Platform Administrators** - System administrators and approvers
6. **Property Owners** - Current property owners listing their assets

Success Criteria

1. **Listing Approval Time:** Reduce listing approval cycle from submission to published state to less than 48 hours
2. **Investment Calculation Accuracy:** Automated ROI calculations with 100% accuracy for bond financing and cash purchases
3. **User Engagement:** 80% of registered users complete their first agreement within 7 days of registration
4. **Platform Uptime:** 99.5% availability during business hours (6 AM - 10 PM SAST)
5. **Document Management:** 100% of property artifacts stored securely with public URLs available within 2 seconds

Scope Statement

In Scope

- Property listing creation, management, and discovery
- Investment financial modeling (bonds, cash purchases, transaction costs)
- Multi-level approval workflows for listings
- Service provider directory and management
- User account and profile management with role-based access
- Contract execution and versioning system
- Multi-channel notification system (email primary)
- File upload and storage (AWS S3)
- Portfolio management for investors
- Multi-let and student accommodation rental models
- Subunit management for complex properties

Out of Scope

- Property transaction processing (escrow, payments)
- Property management operations (maintenance requests, tenant management)
- Multi-currency support beyond ZAR (noted as future enhancement)
- Mobile native applications (web responsive only)
- Real-time chat functionality
- Automated property valuation services
- Integration with property registries or deeds offices

---

1. Stakeholder Analysis

1.1 Stakeholder Registry

Stakeholder Group	Role	Key Interests	Pain Points
Property Investors	Primary Users	Discover profitable investment opportunities, calculate ROI, track portfolio	Lack of transparency, complex financial calculations, limited property options
Sourcing Agents	Content Creators	List properties, earn sourcing fees, build reputation	Time-consuming approval processes, difficulty showcasing properties
Buyers Representatives	Service Professionals	Manage property modifications, coordinate renovations	Unclear modification tracking, lack of integration with listings
Service Providers	Third-party Vendors	Gain visibility, receive inquiries, build portfolio	Limited discovery, no centralized platform for property services
Platform Administrators	System Operators	Ensure listing quality, manage approvals, maintain platform integrity	Manual approval workflows, lack of audit trails
Property Owners	Asset Holders	Maximize property value, find qualified investors	Limited reach, complex listing requirements

1.2 User Personas

Persona 1: The Investment-Focused Investor (Primary)

Name: Thabo Mthembu
Age: 38
Role: Investor
Goals:
- Build a diversified rental property portfolio
- Maximize ROI through multi-let strategies
- Access detailed financial projections before committing
- Track all investments in one dashboard

Pain Points:
- Difficulty finding properties with realistic rental income projections
- Complex bond calculations requiring financial expertise

- Lack of transparency in sourcing fees and transaction costs
- No central platform to manage multiple property investments

Motivation:
"I want to build passive income through rental properties, but I need accurate financial data and professional support to make informed decisions."

Persona 2: The Professional Sourcing Agent

Name: Zanele Khumalo
Age: 32
Role: Sourcing Agent
Goals:
- List high-quality investment properties efficiently
- Earn competitive sourcing fees (5% of purchase price, minimum R30,000)
- Build credibility through successful deals
- Manage multiple listings simultaneously

Pain Points:
- Long approval times delay property visibility
- Need to manually calculate and explain financial models to investors
- Difficulty managing property documentation (photos, reports, leases)
- Limited tools to showcase multi-let potential

Motivation:
"I want to be recognized as a trusted sourcing agent who delivers high-quality investment opportunities with complete transparency."

Persona 3: The Platform Administrator

Name: John van der Merwe
Age: 45
Role: Admin
Goals:
- Ensure only high-quality listings are published
- Maintain platform integrity and reputation
- Track all system activities for compliance
- Manage user agreements and contracts

Pain Points:
- Manual review of each listing submission
- Lack of automated validation for listing completeness
- Difficulty tracking approval status across multiple listings
- Need better tools for user agreement management

Motivation:
"I need efficient tools to review and approve listings quickly while maintaining quality standards and regulatory compliance."

1.3 Stakeholder Communication Matrix

Stakeholder	Communication Method	Frequency	Key Information Needs
Investors	Email notifications, Dashboard	Real-time for status changes	Listing approvals, new opportunities, portfolio updates
Sourcing Agents	Email, Telegram integration	Real-time for submissions	Approval status, feedback, published listings
Administrators	Email, Admin panel	Daily summaries	Pending approvals, system health, user activities
Service Providers	Email, Profile updates	Weekly	New opportunities, profile views, inquiries
Property Owners	Email	On status changes	Listing status, investor inquiries, offers

---

2. Business Requirements

2.1 Strategic Business Goals

1. **Market Leadership:** Establish InvestRand as the leading property investment platform in South Africa within 24 months
2. **Transaction Volume:** Facilitate R500 million in property transactions annually by year 3
3. **User Acquisition:** Onboard 5,000 active investors and 500 sourcing agents within 18 months
4. **Platform Revenue:** Generate revenue through sourcing fees (5% of transaction value, min R30,000)
5. **Quality Assurance:** Maintain 95% listing quality score through robust approval workflows

2.2 Business Objectives

Objective	Measurement	Target	Timeline
Reduce listing approval time	Average hours from submission to published	< 48 hours	Q1 2026
Increase platform engagement	Monthly active users	3,000 MAU	Q2 2026
Improve listing quality	Approval rate (first submission)	> 70%	Q3 2026
Expand service provider network	Registered service providers	200 active providers	Q4 2026
Portfolio growth	Average properties per investor	3.5 properties	Q2 2027

2.3 Success Metrics and KPIs

Platform Metrics

- **Listing Velocity:** Number of listings published per week (Target: 20+)
- **Approval Efficiency:** Percentage of listings approved within 24 hours (Target: 60%)
- **User Retention:** 90-day user retention rate (Target: 65%)
- **Portfolio Value:** Total property value under management (Target: R2 billion by year 2)

Financial Metrics

- **Average Sourcing Fee:** Per transaction (Minimum: R30,000, Target Average: R75,000)
- **Investment ROI:** Average projected ROI for published listings (Target: 10-15% annually)
- **Transaction Completion Rate:** Percentage of listings that result in agreements (Target: 25%)

Quality Metrics

- **Listing Completeness:** Percentage of listings with all required fields (Target: 95%)
- **Data Accuracy:** Accuracy of automated financial calculations (Target: 100%)
- **User Satisfaction:** Net Promoter Score (Target: > 50)

2.4 Business Constraints

Financial Constraints

- Platform operates on sourcing fee revenue model (5% of property purchase price)

- Minimum sourcing fee threshold: R30,000 per transaction
- Infrastructure budget limited to AWS South Africa region (af-south-1)

Regulatory Constraints

- Compliance with South African data protection regulations (POPIA)
- Property listing accuracy requirements
- Financial disclosure requirements for investment properties
- Contract execution must capture location, IP, and timestamp for legal validity

Technical Constraints

- Primary currency: ZAR (multi-currency marked as future enhancement)
- AWS infrastructure locked to South Africa region
- Must support both REST and GraphQL APIs for flexibility
- File storage limited to AWS S3 with public-read access

Business Process Constraints

- All listings must undergo approval workflow before publication
- Sourcing fees automatically calculated based on property purchase price
- Default bond terms: 240 months (20 years) at 9.75% interest rate
- Investment vehicles tied to single property listings

2.5 Key Assumptions

1. **User Access:** All users have reliable internet access and modern web browsers
2. **Property Data:** Sourcing agents provide accurate property information and documentation
3. **Financial Literacy:** Investors understand basic rental property investment concepts
4. **Service Provider Availability:** Sufficient service providers available in target markets
5. **Market Conditions:** Property market remains stable with consistent rental demand
6. **Technology Adoption:** Users comfortable with web-based property discovery and management
7. **Regulatory Stability:** No significant changes to property investment regulations
8. **Currency Stability:** ZAR remains the sole operating currency for initial platform launch

---

3. Functional Requirements

3.1 User Account Management

REQ-UAM-001: User Registration

Type: Functional
Priority: Must Have
Description: The system shall allow new users to register with email, password, first name, and last name.

Rationale: User registration is the entry point for all platform interactions. Email-based registration enables notification delivery and account recovery.

Acceptance Criteria:
- Given a user visits the registration page
- When they provide valid email, password (meeting complexity requirements), first name, and last name
- Then an account is created with a unique Account UID
- And a UserProfile is automatically created with needs_updating=True
- And the user receives a welcome email notification
- And the user is prompted to sign the "Proposal to Service" agreement

Dependencies: Email notification system, Agreement system

REQ-UAM-002: User Authentication (JWT)

Type: Functional
Priority: Must Have
Description: The system shall authenticate users using JWT tokens with access and refresh token mechanism.

Rationale: JWT-based authentication enables stateless authentication suitable for REST and GraphQL APIs with token refresh capability.

Acceptance Criteria:
- Given a registered user provides valid email and password
- When they submit login credentials
- Then the system returns access token (short-lived) and refresh token (long-lived)
- And subsequent API requests include access token in Authorization header
- And tokens can be refreshed without re-authentication
- And failed login attempts are logged and rate-limited (max 5 attempts, 2-hour lockout)

Dependencies: JWT library, Brute-force protection (django-axes)

REQ-UAM-003: User Profile Management

Type: Functional
Priority: Must Have
Description: The system shall maintain extended user profiles with role assignments, contact details, and metadata.

Rationale: User profiles extend base authentication with business-specific data including roles, contact information, and customizable metadata.

Acceptance Criteria:
- Given an authenticated user
- When they access their profile
- Then they can view and edit contact details (phone number, secondary phone number)
- And they can view assigned roles (Sourcing Agent, Investor, Service Provider, Admin, Buyers Representative)
- And they can manage UI settings (stored as JSON configuration)
- And profile changes update the modified timestamp
- And needs_updating flag can be cleared once required information is complete

Dependencies: User account, Role system

REQ-UAM-004: Role-Based Access Control

Type: Functional
Priority: Must Have
Description: The system shall enforce role-based permissions to control user access to features and data.

Rationale: Different user types require different capabilities. Role-based access ensures investors, agents, and admins see only relevant features.

Acceptance Criteria:
- Given a user has assigned roles
- When they access platform features
- Then permissions are checked based on role (Sourcing Agent, Investor, Service Provider, Admin, Buyers Representative)
- And unauthorized access attempts return 403 Forbidden
- And role assignments are auditable with change history
- And users can have multiple roles simultaneously

Dependencies: Permission manager, User profile

REQ-UAM-005: Agreement Execution

Type: Functional
Priority: Must Have
Description: The system shall require users to execute the "Proposal to Service" agreement before accessing platform features.

Rationale: Legal agreements establish terms of service and protect platform from liability. Capturing execution metadata ensures legal enforceability.

Acceptance Criteria:
- Given a new user completes registration
- When they first log in
- Then they are presented with the current version of the "Proposal to Service" agreement
- And they must accept the agreement to proceed
- And the system captures: account ID, user ID, agreement version, timestamp, IP address, user agent, GPS coordinates (if available), platform details
- And agreement execution is immutable and auditable
- And users can view their signed agreements at any time

Dependencies: Contract system, User account

3.2 Property Listing Management

REQ-PLM-001: Create Property Listing

Type: Functional
Priority: Must Have
Description: Sourcing agents and admins shall be able to create new property listings with comprehensive property details.

Rationale: Property listings are the core asset of the platform. Comprehensive data capture enables accurate financial modeling and investor decision-making.

Acceptance Criteria:
- Given an authorized user (Sourcing Agent or Admin)
- When they create a new listing
- Then they must provide: name, listing type, address, buying price, market price, rental approach
- And the system generates a unique listing UID
- And listing status defaults to "Created"
- And lister_user and lister_account are automatically set
- And owner field is optional
- And an Approval record is automatically created with status "Pending Submission"
- And a main Subunit is automatically created for the listing
- And the system captures created and modified timestamps

Dependencies: Approval system, Address management, Subunit system

---

REQ-PLM-002: Listing Types

Type: Functional
Priority: Must Have
Description: The system shall support the following property listing types: Free Standing Unit, Free Standing House, Apartment, Duplex, Attached Duplex, Attached Unit.

Rationale: Different property types have different investment characteristics. Type classification aids in search, filtering, and appropriate financial modeling.

Acceptance Criteria:
- Given a user creates a listing
- When they select listing type
- Then they can choose from: Free Standing Unit, Free Standing House, Apartment, Duplex, Attached Duplex, Attached Unit
- And the selected type is stored and displayed throughout the system
- And listing type is seeded in database during initial setup

Dependencies: Database seeding

REQ-PLM-003: Listing Address Management

Type: Functional
Priority: Must Have
Description: The system shall capture comprehensive address information including Google Places integration and GPS coordinates.

Rationale: Accurate location data enables map-based search, location analysis, and investor due diligence.

Acceptance Criteria:
- Given a user creates or edits a listing
- When they provide address information
- Then the system stores: Google address, Google Place ID, GPS longitude, GPS latitude, street number, street name, suburb, city, country code, postal code, province
- And all address fields are optional to accommodate incomplete data
- And GPS coordinates use decimal format with 6 decimal places
- And country code uses ISO 2-letter format

Dependencies: Google Places API (frontend integration)

REQ-PLM-004: Listing Status Workflow

Type: Functional
Priority: Must Have
Description: The system shall manage listing status through defined states: Created, In Review, Published, Under Offer, Sold, Not Available.

Rationale: Status workflow provides visibility into listing lifecycle and controls when listings appear in public search results.

Acceptance Criteria:
- Given a listing exists
- When the status changes
- Then valid transitions are: Created → In Review → Published → Under Offer → Sold
- And Published → Not Available is allowed
- And status changes are tracked with timestamps
- And only Published listings appear in public search
- And status changes trigger notifications to lister and interested investors

Dependencies: Approval system, Notification system

REQ-PLM-005: Listing Approval Workflow

Type: Functional
Priority: Must Have
Description: Listings shall undergo approval workflow with states: Pending Submission, Submitted, In Review, Needs Review, Approved, Rejected.

Rationale: Quality control through approval ensures only complete, accurate listings are published, maintaining platform reputation.

Acceptance Criteria:
- Given a listing is created
- When the lister submits for approval
- Then approval status changes to "Submitted"
- And an admin can review and update status to: In Review, Needs Review, Approved, or Rejected
- And status updates create StateUpdate records with comment, timestamp, and reviewer user
- And Approved status triggers listing status change to "Published"
- And Rejected status includes mandatory comment explaining rejection reason
- And all state transitions are auditable with full history

Dependencies: Approval system, User roles, Notification system

---

REQ-PLM-006: Subunit Management

Type: Functional
Priority: Must Have
Description: The system shall support hierarchical subunit structures within listings to model multi-let configurations.

Rationale: Multi-let properties (houses split into multiple rental units) require granular unit management for accurate rental income calculations.

Acceptance Criteria:
- Given a listing exists
- When subunits are created
- Then each subunit can have: type, description, size (m²), expected rental income, current rental income, bedroom count, bathroom count, amenities
- And subunits can have parent-child relationships (e.g., house > unit > room)
- And subunit types include: Free Standing Unit, Free Standing Room, Apartment, Room, Duplex, Attached Duplex, Attached Unit, Main
- And a "Main" subunit is automatically created for every listing
- And rental income aggregates from child subunits to parent subunits
- And listing total rental income aggregates from top-level subunits only

Dependencies: Listing, Rental approach

REQ-PLM-007: Rental Approaches

Type: Functional
Priority: Must Have
Description: Subunits shall support different rental strategies: Multi-Let (Multi-Family), Student Accommodation, Single Family, None.

Rationale: Different rental strategies (multi-let vs single-family) significantly impact income and investor appeal.

Acceptance Criteria:
- Given a subunit is configured
- When a rental approach is selected
- Then available options are: Multi-Let (Multi-Family), Student Accommodation, Single Family, None
- And both current rental approach and future rental approach can be specified
- And future rental approach includes transition timeline (in months)
- And rental approach is seeded in database during setup

Dependencies: Subunit management, Database seeding

REQ-PLM-008: Property Amenities

Type: Functional
Priority: Should Have
Description: Listings and subunits shall capture amenities as structured data and free-text descriptions.

Rationale: Amenities influence property value and rental appeal. Detailed amenity data aids investor decision-making.

Acceptance Criteria:
- Given a listing or subunit
- When amenities are specified
- Then the system captures: is_furnished, is_partially_furnished, pets_allowed, allows_subletting, has_private_garden, has_shared_garden, has_private_kitchen, has_shared_kitchen, has_private_bathroom, has_shared_bathroom, has_prepaid_electricity_meter, has_prepaid_water_meter, is_partitioned
- And free-text amenities field allows custom descriptions
- And amenities can be specified at both listing and subunit levels

Dependencies: Listing, Subunit

REQ-PLM-009: Property Modifications

Type: Functional
Priority: Should Have
Description: Listings shall track planned property modifications with expected costs and service provider associations.

Rationale: Many investment properties require modifications (renovations, repairs) before optimal rental. Tracking modifications informs total investment cost.

Acceptance Criteria:
- Given a listing exists
- When modifications are added
- Then each modification captures: description, expected cost (Money field in ZAR), service provider (optional)
- And total modification cost is calculated by summing all modification costs
- And modifications integrate into total investment cost calculations

Dependencies: Listing, Service provider, Investment modeling

REQ-PLM-010: Listing Artifacts (Documents)

Type: Functional
Priority: Must Have
Description: The system shall allow upload and management of property-related documents and images.

Rationale: Visual and documentary evidence (photos, financial reports, leases, offers) builds investor confidence and provides due diligence materials.

Acceptance Criteria:
- Given a listing exists
- When files are uploaded
- Then artifact types include: Listing Photos, Financial Reports, Leases, Offers, Proof of Income, Custom
- And files are uploaded to AWS S3 with UUID-based filenames
- And file URL format: https://investrand-backend-prod.s3.af-south-1.amazonaws.com/property_artifacts/{uuid}.{extension}
- And files are publicly accessible (public-read ACL)
- And artifact metadata (listing association, upload timestamp) is stored in database
- And deleted listings do not delete associated S3 artifacts (manual cleanup)

Dependencies: AWS S3, File upload system

REQ-PLM-011: Listing Discovery and Search

Type: Functional
Priority: Must Have
Description: Users shall be able to discover published listings through search and filtering.

Rationale: Discovery is the primary investor use case. Effective search enables investors to find properties matching their criteria.

Acceptance Criteria:
- Given published listings exist
- When a user searches
- Then they can filter by: location (city, suburb), listing type, price range, rental income range, rental approach
- And only listings with status "Published" appear in search results
- And results display: name, address (suburb and city), listing type, buying price, total expected rental income, primary photo
- And users can view full listing details by clicking
- And unauthenticated users can browse listings (read-only)

Dependencies: Listing status, Search indexing

REQ-PLM-012: Listing Preview

Type: Functional
Priority: Should Have
Description: Listers shall be able to preview how their listing appears to investors before publishing.

Rationale: Preview reduces errors and helps listers optimize listing presentation before submission for approval.

Acceptance Criteria:
- Given a listing is in "Created" status
- When the lister accesses preview
- Then they see the listing rendered exactly as investors will see it
- And preview is accessible via /listing/preview/:listing_id route
- And preview is only accessible to the lister and admins

Dependencies: Listing display views

REQ-PLM-013: Manage My Listings

Type: Functional
Priority: Must Have
Description: Sourcing agents and admins shall have a dashboard to view and manage all their listings.

Rationale: Centralized listing management enables efficient oversight of multiple properties at various approval stages.

Acceptance Criteria:
- Given a sourcing agent or admin logs in
- When they access "My Listings" dashboard
- Then they see all listings they created
- And listings are grouped by status (Created, In Review, Published, Under Offer, Sold, Not Available)
- And each listing shows: name, address, status, approval state, created date, last modified date
- And they can navigate to edit, preview, or view full details for each listing
- And they can submit listings for approval from this dashboard

Dependencies: Listing, User roles, Approval system

3.3 Investment Management

REQ-INV-001: Create Investment Vehicle

Type: Functional
Priority: Must Have

Description: The system shall automatically create an Investment record when a listing is created or when an investor expresses interest.

Rationale: Investment records encapsulate financial modeling separate from property listing data, enabling multiple financing scenarios.

Acceptance Criteria:
- Given a listing exists
- When an Investment is created
- Then it is associated with exactly one listing
- And default transaction costs are automatically initialized (sourcing fee)
- And default financing option (Bank Bond) is created
- And the Investment serves as the container for all financial modeling

Dependencies: Listing, Transaction costs, Financing

REQ-INV-002: Transaction Cost Management

Type: Functional
Priority: Must Have
Description: Investments shall track one-time transaction costs including transfer costs, bond registration, sourcing fees, deposits, and cash payments.

Rationale: Accurate total investment cost calculation requires capturing all one-time costs beyond property purchase price.

Acceptance Criteria:
- Given an Investment exists
- When transaction costs are configured
- Then supported cost types are: Transfer Cost, Bond Registration Fees, Sourcing Fee, Deposit, Cash Payment
- And each cost has: cost type, amount (Money field in ZAR), financing association (optional)
- And sourcing fee is automatically calculated as 5% of buying price (minimum R30,000)
- And sourcing fee is marked as default transaction cost and initialized automatically
- And total transaction cost is sum of all transaction cost amounts

Dependencies: Investment, Financing, Money field

REQ-INV-003: Sourcing Fee Calculation

Type: Functional
Priority: Must Have
Description: The system shall automatically calculate sourcing fees as 5% of property buying price with a minimum of R30,000.

Rationale: Sourcing fees are the primary platform revenue model. Automated calculation ensures consistency and transparency.

Acceptance Criteria:
- Given an Investment is created
- When sourcing fee is calculated
- Then fee = MAX(buying_price * 0.05, R30,000)
- And sourcing fee is stored as a transaction cost of type "Sourcing Fee"
- And sourcing fee updates automatically if buying price changes
- And sourcing fee is displayed clearly to investors in investment summary

Dependencies: Investment, Transaction costs, Listing buying price

---

REQ-INV-004: Financing Types

Type: Functional
Priority: Must Have
Description: Investments shall support two financing types: Bank Bond and Cash.

Rationale: Financing method fundamentally changes investment structure. Bank bond requires interest calculations; cash requires full payment upfront.

Acceptance Criteria:
- Given an Investment exists
- When financing type is selected
- Then available options are: Bank Bond, Cash
- And each Investment can have multiple financing scenarios
- And exactly one financing is marked as "default" (is_default=True)
- And switching financing type recalculates transaction costs and recurring expenses

Dependencies: Investment, Bond, Transaction costs

REQ-INV-005: Bank Bond Modeling

Type: Functional
Priority: Must Have
Description: The system shall calculate bond repayments using standard amortization formulas when financing type is Bank Bond.

Rationale: Accurate bond calculations enable investors to understand total cost of financing and monthly cash flow requirements.

Acceptance Criteria:
- Given financing type is Bank Bond
- When bond parameters are set

- Then default values are: projected_interest_rate=9.75%, term_months=240 (20 years), deposit=R0
- And bond principal = buying_price - deposit
- And monthly repayment calculation uses formula: P * [r(1+r)^n] / [(1+r)^n - 1] where P=principal, r=monthly_rate, n=term_months
- And bond repayment is automatically added as recurring expense
- And deposit (if > 0) is automatically added as transaction cost of type "Bond Deposit"
- And actual_interest_rate can be specified once bond is finalized (defaults to 0)

Dependencies: Investment, Financing, Recurring expenses, Transaction costs

---

REQ-INV-006: Cash Purchase Modeling

Type: Functional
Priority: Must Have
Description: When financing type is Cash, the system shall create a transaction cost equal to the full buying price.

Rationale: Cash purchases eliminate bond interest but require full capital upfront. This must be reflected in transaction cost totals.

Acceptance Criteria:
- Given financing type is Cash
- When financing is saved
- Then a transaction cost of type "Cash Payment" is created with amount = buying_price
- And any existing bond records for this financing are deleted
- And any bond-related recurring expenses are removed
- And switching from Cash to Bank Bond removes the cash payment transaction cost

Dependencies: Investment, Financing, Transaction costs

REQ-INV-007: Recurring Expenses

Type: Functional
Priority: Must Have
Description: Investments shall track monthly and annual recurring expenses with optional cost escalation percentages.

Rationale: Ongoing expenses (rates, levies, maintenance, bond repayments) significantly impact investment ROI and must be modeled accurately.

Acceptance Criteria:
- Given an Investment exists
- When recurring expenses are configured
- Then each expense captures: description, cost (Money field in ZAR), cost_percentage_of_income, is_annual_expense, expected_annual_escalation_percentage, financing association

- And expenses can be fixed amount OR percentage of rental income (mutually exclusive)
- And bond repayments are automatically created as recurring expense when Bank Bond financing is selected
- And recurring expenses are associated with specific financing scenarios
- And annual expenses are flagged separately from monthly expenses

Dependencies: Investment, Financing, Bond

REQ-INV-008: Rental Income Projection

Type: Functional
Priority: Must Have
Description: The system shall calculate total expected and current rental income by aggregating subunit rental values.

Rationale: Accurate rental income projection is critical for ROI calculation and investor decision-making.

Acceptance Criteria:
- Given a listing has subunits configured
- When rental income is calculated
- Then total_expected_rental_income = sum of expected_rental_income from all top-level subunits (no parent_subunit)
- And total_current_rental_income = sum of current_rental_income from all top-level subunits
- And rental income aggregates automatically when subunit values change
- And multi-level subunit hierarchies aggregate upward (children → parent → listing)
- And annual rental escalation percentage is applied for future projections

Dependencies: Listing, Subunit, Rental model

REQ-INV-009: Rental Model Configuration

Type: Functional
Priority: Must Have
Description: Listings shall have associated rental models specifying rental type and annual escalation.

Rationale: Rental escalation projections enable multi-year ROI calculations, essential for long-term investment evaluation.

Acceptance Criteria:
- Given a listing exists
- When a rental model is created
- Then it captures: rental_type (references RentalApproach), annual_rental_escalation (decimal percentage)
- And annual_rental_escalation applies to future rental income projections
- And rental model is associated with the listing (one rental model per listing)

Dependencies: Listing, Rental approach

REQ-INV-010: Investment Summary Calculations

Type: Functional
Priority: Must Have
Description: The system shall provide comprehensive investment summary showing total costs, income, and projected returns.

Rationale: Investors need clear financial summary to make go/no-go decisions. Summary consolidates all financial components.

Acceptance Criteria:
- Given an Investment exists
- When investment summary is displayed
- Then it shows: Buying Price, Market Price, Total Transaction Costs, Total Modification Costs, Total Initial Investment (Buying + Transaction + Modifications), Total Expected Monthly Rental Income, Total Current Monthly Rental Income, Total Monthly Recurring Expenses, Net Monthly Cash Flow (Income - Expenses), Projected Annual Return (Net Monthly * 12 / Total Investment)
- And all amounts are displayed in ZAR currency format
- And calculations update in real-time as underlying values change
- And summary is accessible to lister and investors

Dependencies: Investment, Transaction costs, Recurring expenses, Rental income, Modifications

---

3.4 Service Provider Management

REQ-SPM-001: Service Provider Profiles

Type: Functional
Priority: Should Have
Description: Service providers shall be able to create and manage their professional profiles.

Rationale: Service provider directory adds value for investors and listers who need contractors, property managers, and other professionals.

Acceptance Criteria:
- Given a user registers as a Service Provider
- When they create their profile
- Then they can specify: company name, description, services offered, contact details, coverage areas
- And they can upload profile pictures and portfolio images
- And profile pictures are stored in S3 with prefix: service_provider_pictures/
- And profiles are publicly viewable in service provider directory

Dependencies: User profile, File upload, S3 storage

REQ-SPM-002: Service Provider Discovery

Type: Functional
Priority: Should Have
Description: Users shall be able to search and discover service providers by service type and location.

Rationale: Centralized service provider directory simplifies finding qualified professionals for property services.

Acceptance Criteria:
- Given service provider profiles exist
- When users search the directory
- Then they can filter by: service type, location, rating (if rating system implemented)
- And search results display: company name, services, location, contact information, profile picture
- And users can view full profile details
- And unauthenticated users can browse service provider directory

Dependencies: Service provider profiles, Search functionality

---

REQ-SPM-003: Service Provider-Listing Association

Type: Functional
Priority: Should Have
Description: Listers shall be able to associate service providers with property modifications.

Rationale: Linking service providers to modifications creates transparency and enables provider portfolio building.

Acceptance Criteria:
- Given a listing has modifications
- When a modification is configured
- Then the lister can optionally associate a service provider
- And associated service provider details are visible in modification details
- And service providers can see listings where they are referenced (future: with permission)

Dependencies: Listing modifications, Service provider profiles

---

3.5 Contract and Agreement Management

REQ-CAM-001: Contract Types

Type: Functional
Priority: Must Have
Description: The system shall support versioned contract templates starting with "Proposal to Service" agreement.

Rationale: Legal agreements establish terms of service. Versioning enables contract updates while maintaining historical records.

Acceptance Criteria:
- Given contract types are defined
- When a contract type is created
- Then it has: name (static identifier), friendly_name (display name)
- And "Proposal to Service" is seeded by default
- And additional contract types can be added (e.g., investment agreements, service contracts)

Dependencies: Database seeding

REQ-CAM-002: Contract Versioning

Type: Functional
Priority: Must Have
Description: Contracts shall support multiple versions with automatic version numbering.

Rationale: Contract terms may change over time. Versioning ensures users execute current terms while maintaining audit trail of historical versions.

Acceptance Criteria:
- Given a contract exists
- When a new version is created
- Then version_number auto-increments (1, 2, 3, ...)
- And each version has: html_content (rendered contract text), active (boolean flag)
- And only one version can be active at a time
- And version numbering is unique per contract (enforced by database constraint)

Dependencies: Contract types

REQ-CAM-003: Contract Execution

Type: Functional
Priority: Must Have
Description: The system shall capture comprehensive metadata when users execute (sign) contracts.

Rationale: Legal enforceability requires proof of who signed, when, where, and under what conditions. Metadata provides this evidence.

Acceptance Criteria:
- Given a user is presented with a contract
- When they accept/execute the contract
- Then the system captures: account, user, contract, contract_version, timestamp, IP address, user agent string, platform details, GPS coordinates (if available), location string (if available)
- And execution records are immutable (no updates or deletions)
- And users can view all their historical contract executions
- And admins can audit all contract executions

Dependencies: Contracts, Contract versions, User account

REQ-CAM-004: Agreement Requirement Enforcement

Type: Functional
Priority: Must Have
Description: New users shall be required to execute the "Proposal to Service" agreement before accessing platform features.

Rationale: Legal protection requires all users to agree to terms of service before using platform.

Acceptance Criteria:
- Given a user registers
- When they first log in
- Then they are redirected to agreement acceptance page
- And they cannot access other features until agreement is executed
- And upon execution, they are redirected to dashboard
- And account.needs_updating flag can be used to trigger additional required information collection

Dependencies: User account, Contract execution, Frontend routing

---

3.6 Approval Workflow Engine

REQ-AWE-001: Generic Approval System

Type: Functional
Priority: Must Have
Description: The system shall provide a generic, reusable approval workflow engine supporting multiple approval item types.

Rationale: Multiple entities require approval (listings, modifications, service providers). Generic engine enables consistent approval process across all entity types.

Acceptance Criteria:
- Given an approval item type is registered (e.g., Listing)
- When an approval workflow is created
- Then it supports states: Pending Submission, Submitted, In Review, Needs Review, Approved, Rejected
- And each item type defines which states are supported
- And each item type defines starting state
- And state transitions create StateUpdate records with user, timestamp, and optional comment

Dependencies: State management, User authentication

REQ-AWE-002: Approval Item Types

Type: Functional
Priority: Must Have
Description: The system shall register supported approval item types with their source apps and supported states.

Rationale: Type registration enables the approval engine to work with any model implementing approval interface.

Acceptance Criteria:
- Given a model requires approval
- When the ItemType is registered
- Then it specifies: model_name, source_app, supported_states, starting_state
- And "Listing" item type is registered with source_app="property"
- And supported states for Listing are: Pending Submission, Submitted, Needs Review, Rejected, Approved
- And starting state for Listing is "Pending Submission"

Dependencies: Database seeding, State management

REQ-AWE-003: State Updates with Comments

Type: Functional
Priority: Must Have
Description: All approval state changes shall create audit trail records capturing who changed the state, when, and why.

Rationale: Approval audit trail provides accountability and enables tracking approval bottlenecks.

Acceptance Criteria:
- Given an approval exists
- When state is updated
- Then a StateUpdate record is created with: approval, new_state, comment (optional, mandatory for rejections), created_by (user), timestamp

- And the parent Approval.state is updated to match new_state
- And all state updates are immutable (no edits or deletions)
- And state update history is viewable by lister and admins

Dependencies: Approval, State, User authentication

REQ-AWE-004: Approval Closure

Type: Functional
Priority: Should Have
Description: Approved or Rejected approvals shall be marked as closed to prevent further state changes.

Rationale: Terminal states (Approved, Rejected) should not change once reached. Closure flag prevents accidental or malicious state changes.

Acceptance Criteria:
- Given an approval reaches Approved or Rejected state
- When is_closed is set to True
- Then no further state updates are allowed
- And reopening requires explicit admin action

Dependencies: Approval, State updates

3.7 Notification System

REQ-NOT-001: Multi-Channel Notification Framework

Type: Functional
Priority: Must Have
Description: The system shall provide a flexible notification framework supporting multiple communication mediums (email primary).

Rationale: Event-driven notifications keep users informed of important status changes. Multi-channel design enables future expansion to SMS, push notifications.

Acceptance Criteria:
- Given notification types are defined
- When a notification event occurs
- Then the system determines: notification type, user recipients, communication medium (email), delivery endpoint
- And notifications are queued as Jobs
- And delivery is handled asynchronously
- And delivery success/failure is logged

Dependencies: Email configuration, Job queue

REQ-NOT-002: Notification Templates

Type: Functional
Priority: Must Have
Description: Notifications shall use templates with variable substitution for dynamic content.

Rationale: Templates enable consistent messaging while allowing personalization with user-specific data.

Acceptance Criteria:
- Given a notification type exists
- When a template is created
- Then it specifies: name, content (with placeholders), communication_medium, required_fields (JSON dict), meta_templates (JSON)
- And placeholders are replaced with actual values at send time
- And required fields are validated before sending
- And missing required fields log error and prevent sending

Dependencies: Notification types

REQ-NOT-003: User Notification Subscriptions

Type: Functional
Priority: Must Have
Description: Users shall be able to subscribe and unsubscribe from notification types.

Rationale: User control over notifications prevents spam and improves user experience.

Acceptance Criteria:
- Given notification types exist
- When a user manages subscriptions
- Then they can subscribe or unsubscribe from each notification type
- And new users are auto-subscribed to default notification groups
- And unsubscribed users do not receive notifications
- And subscription status is checked before queuing notification jobs

Dependencies: Notification types, User account

REQ-NOT-004: Notification Groups

Type: Functional
Priority: Should Have
Description: Notifications shall be organized into groups for bulk subscription management.

Rationale: Notification groups (e.g., "Investment Updates", "System Alerts") simplify user subscription management.

Acceptance Criteria:
- Given notification types exist
- When notification groups are created
- Then each group contains multiple notification types
- And users can subscribe/unsubscribe to entire groups
- And new users are auto-subscribed to essential notification groups

Dependencies: Notification types, Subscriptions

REQ-NOT-005: Email Delivery

Type: Functional
Priority: Must Have
Description: The system shall deliver notifications via email using configured SMTP endpoint.

Rationale: Email is the primary communication channel for platform notifications.

Acceptance Criteria:
- Given a notification is queued for email delivery
- When the email handler processes the job
- Then it retrieves template, substitutes variables, sends via SMTP
- And delivery is logged with timestamp
- And failures are logged with error details
- And users receive emails from configured DEFAULT_FROM_EMAIL

Dependencies: Email configuration (SMTP), Notification templates

---

REQ-NOT-006: Notification Job History

Type: Functional
Priority: Should Have
Description: The system shall maintain complete history of all notification deliveries.

Rationale: Notification history enables debugging delivery issues and provides audit trail for communications.

Acceptance Criteria:
- Given notifications are sent
- When jobs are processed
- Then each job records: uid, user, notification, communication_medium, data (JSON payload), delivery_endpoint, started timestamp, completed timestamp, subscribed (boolean)

- And job history is queryable by user, notification type, date range
- And admins can view all job history

Dependencies: Notification system, Job model

3.8 Portfolio Management

REQ-PFM-001: My Portfolio Dashboard

Type: Functional
Priority: Must Have
Description: Investors shall have a dashboard showing all properties they have expressed interest in or invested in.

Rationale: Centralized portfolio view enables investors to track and manage multiple property investments.

Acceptance Criteria:
- Given a user has investor role
- When they access "My Portfolio"
- Then they see all listings where they are marked as owner or have active agreements
- And each listing shows: property name, address, purchase price, expected rental income, status
- And they can navigate to full listing details
- And portfolio summary shows: total invested, total expected monthly income, total properties

Dependencies: Listing, User roles, Investment

REQ-PFM-002: My Agreements

Type: Functional
Priority: Must Have
Description: Users shall be able to view all agreements they have executed.

Rationale: Agreement history provides users with record of all contracts they have signed.

Acceptance Criteria:
- Given a user has executed agreements
- When they access "My Agreements"
- Then they see all contract executions associated with their account
- And each agreement shows: contract type, contract name, version, execution date, status
- And they can view the full contract content they signed
- And agreements are sorted by execution date (newest first)

Dependencies: Contract execution, User account

3.9 Administrative Functions

REQ-ADM-001: Listing Management Dashboard

Type: Functional
Priority: Must Have
Description: Admins shall have a dashboard to view and manage all listings across all users.

Rationale: Admins need oversight of all platform listings to manage approvals and quality control.

Acceptance Criteria:
- Given a user has admin role
- When they access "Manage Listings"
- Then they see all listings regardless of owner
- And they can filter by: status, approval state, lister, date range
- And they can navigate to any listing for review and approval
- And they can change listing status and approval state
- And they can view full listing details and edit if needed

Dependencies: Listing, User roles (Admin), Approval system

REQ-ADM-002: Agreement Dashboard

Type: Functional
Priority: Must Have
Description: Admins shall have a dashboard to view all user agreements and execution status.

Rationale: Legal compliance requires oversight of agreement execution across all users.

Acceptance Criteria:
- Given a user has admin role
- When they access "Agreements Dashboard"
- Then they see summary of agreement execution rates by agreement type
- And they can drill down to see all executions for a specific user
- And they can filter by: agreement type, date range, user
- And they can export agreement execution data for compliance reporting

Dependencies: Contract execution, User roles (Admin)

REQ-ADM-003: User Account Management

Type: Functional
Priority: Must Have
Description: Admins shall be able to view and manage user accounts, roles, and permissions.

Rationale: User administration enables account support, role assignment, and security management.

Acceptance Criteria:
- Given a user has admin role
- When they access user management
- Then they can view all user accounts
- And they can assign/remove roles (Investor, Sourcing Agent, Service Provider, Admin, Buyers Representative)
- And they can view user activity history (listings created, agreements signed, logins)
- And they can deactivate user accounts if needed

Dependencies: User account, User profile, Roles, Permissions

REQ-ADM-004: Audit Trail Access

Type: Functional
Priority: Must Have
Description: Admins shall have access to comprehensive audit logs of all data changes.

Rationale: Audit trail provides accountability and enables investigation of data issues or disputes.

Acceptance Criteria:
- Given audit logging is enabled (django-easy-audit)
- When admins access audit logs
- Then they can view all model create, update, delete operations
- And logs capture: model, operation, user, timestamp, field changes (before/after)
- And logs are searchable by: model, user, date range, operation type
- And audit logs are immutable

Dependencies: django-easy-audit, Admin role

3.10 Lease Management

REQ-LSE-001: Lease Tracking

Type: Functional
Priority: Should Have
Description: Subunits shall support lease records tracking current and historical tenant agreements.

Rationale: Lease records provide proof of current rental income and rental history for investor due diligence.

Acceptance Criteria:
- Given a subunit exists
- When leases are created
- Then each lease captures: subunit, monthly_rent (Money field), start_date, end_date, tenant_details
- And the most recent lease determines subunit.current_rental_income
- And current_rental_income updates automatically when new lease is added
- And lease history is maintained for audit purposes

Dependencies: Subunit, Money field

---

4. Non-Functional Requirements

4.1 Performance Requirements

REQ-NFR-PERF-001: Page Load Time

Type: Non-Functional
Priority: Must Have
Description: Web pages shall load within 3 seconds on standard broadband connections (10 Mbps+).

Rationale: Fast page loads improve user experience and reduce bounce rates, especially important for listing discovery.

Acceptance Criteria:
- Listing discovery page loads in < 3 seconds with 20 listings displayed
- Individual listing detail page loads in < 2 seconds
- Dashboard pages load in < 2 seconds
- Measured on 10 Mbps connection with <100ms latency

Testing: Use Google Lighthouse, WebPageTest, or similar tools

---

REQ-NFR-PERF-002: API Response Time

Type: Non-Functional
Priority: Must Have
Description: REST and GraphQL API endpoints shall respond within 500ms for 95% of requests.

Rationale: Fast API responses enable responsive frontend interactions and good user experience.

Acceptance Criteria:
- 95th percentile response time < 500ms

- 99th percentile response time < 1000ms
- Measured under normal load (100 concurrent users)

Testing: Use load testing tools (Locust, JMeter) to measure under load

---

REQ-NFR-PERF-003: Database Query Optimization

Type: Non-Functional
Priority: Must Have
Description: Database queries shall use indexes and avoid N+1 query problems to maintain performance.

Rationale: Efficient database queries prevent performance degradation as data volume grows.

Acceptance Criteria:
- Listing search queries execute in < 100ms
- Investment calculation queries execute in < 200ms
- No N+1 query patterns in API endpoints
- Database indexes on foreign keys, status fields, UID fields

Testing: Use Django Debug Toolbar, query profiling to identify slow queries

---

REQ-NFR-PERF-004: File Upload Performance

Type: Non-Functional
Priority: Should Have
Description: File uploads to S3 shall complete within 10 seconds for files up to 5MB.

Rationale: Responsive file uploads improve user experience during listing creation.

Acceptance Criteria:
- 5MB file uploads complete in < 10 seconds on 10 Mbps upload connection
- Upload progress indicator displays during upload
- Failed uploads provide clear error messages

Testing: Test with various file sizes (1MB, 3MB, 5MB) and connection speeds

---

REQ-NFR-PERF-005: Scalability

Type: Non-Functional
Priority: Should Have
Description: The system shall support 1,000 concurrent users and 10,000 active listings without performance degradation.

Rationale: Platform growth requires system to scale to handle increased load.

Acceptance Criteria:
- System maintains < 3s page load times with 1,000 concurrent users
- API response times remain < 500ms (95th percentile) under load
- Database can store and query 10,000+ listings efficiently
- AWS Lambda can scale to handle traffic spikes

Testing: Conduct load testing with simulated user traffic

4.2 Security Requirements

REQ-NFR-SEC-001: Authentication Security

Type: Non-Functional
Priority: Must Have
Description: User authentication shall use industry-standard security practices including password hashing, JWT tokens, and brute-force protection.

Rationale: Secure authentication prevents unauthorized access and protects user accounts.

Acceptance Criteria:
- Passwords hashed using Django's PBKDF2 algorithm
- JWT tokens with short-lived access tokens (15 min) and long-lived refresh tokens (7 days)
- Brute-force protection limits login attempts (5 attempts, 2-hour lockout)
- Failed login attempts logged for security monitoring

Dependencies: djangorestframework-simplejwt, django-axes

REQ-NFR-SEC-002: Authorization and Access Control

Type: Non-Functional
Priority: Must Have
Description: All API endpoints shall enforce role-based access control to prevent unauthorized data access.

Rationale: Proper authorization ensures users can only access data and features appropriate to their role.

Acceptance Criteria:
- Listing edit endpoints verify user is lister or admin
- Approval state changes restricted to admin users
- User profile data access restricted to account owner or admin
- Portfolio data restricted to investor or admin
- Unauthorized access returns HTTP 403 Forbidden

Dependencies: Permission manager, User roles

REQ-NFR-SEC-003: Data Protection (HTTPS)

Type: Non-Functional
Priority: Must Have
Description: All data transmission shall use HTTPS (TLS 1.2+) encryption.

Rationale: Encrypted transmission protects sensitive data (credentials, financial information) from interception.

Acceptance Criteria:
- All production domains use HTTPS with valid SSL certificates
- HTTP requests redirect to HTTPS
- TLS 1.2 or higher enforced
- Certificate managed via Let's Encrypt or AWS Certificate Manager

Dependencies: Traefik reverse proxy, Let's Encrypt

REQ-NFR-SEC-004: Input Validation

Type: Non-Functional
Priority: Must Have
Description: All user input shall be validated and sanitized to prevent injection attacks.

Rationale: Input validation prevents SQL injection, XSS, and other injection attacks.

Acceptance Criteria:
- Django ORM prevents SQL injection
- HTML input sanitized to prevent XSS
- File uploads validated for type and size
- GraphQL input validated using schema types
- Invalid input returns clear error messages without exposing system details

Dependencies: Django REST Framework serializers, GraphQL schema validation

---

REQ-NFR-SEC-005: File Upload Security

Type: Non-Functional
Priority: Must Have
Description: File uploads shall be validated for file type, size, and content to prevent malicious uploads.

Rationale: Unrestricted file uploads create security risks including malware upload, DoS attacks, and data breaches.

Acceptance Criteria:
- Maximum file size: 5MB per upload
- Allowed file types: .jpg, .jpeg, .png, .pdf
- MIME type validation (not just extension)
- Files stored with UUID filenames to prevent path traversal
- Uploaded files scanned for malware (future enhancement)

Dependencies: File upload serializers, S3 storage

REQ-NFR-SEC-006: Session Management

Type: Non-Functional
Priority: Must Have
Description: User sessions shall expire after period of inactivity and on logout.

Rationale: Session expiration reduces risk of unauthorized access from unattended devices.

Acceptance Criteria:
- JWT access tokens expire after 15 minutes
- Refresh tokens expire after 7 days
- Logout invalidates tokens (blacklist if implemented)
- Expired token access returns HTTP 401 Unauthorized

Dependencies: JWT configuration

REQ-NFR-SEC-007: CORS Configuration

Type: Non-Functional
Priority: Must Have
Description: CORS shall be configured to allow requests only from authorized frontend domains.

Rationale: CORS configuration prevents unauthorized websites from making API requests on behalf of users.

Acceptance Criteria:
- CORS allows origins: platform.investrand.co.za, uat-platform.investrand.co.za, dev-platform.investrand.co.za, localhost:8080 (dev)
- Credentials (cookies, auth headers) allowed for authorized origins
- Preflight requests handled correctly

Dependencies: django-cors-headers

REQ-NFR-SEC-008: Secrets Management

Type: Non-Functional
Priority: Must Have
Description: Secrets (API keys, database credentials, AWS keys) shall be externalized from source code.

Rationale: Hardcoded secrets in source code create security vulnerabilities if code is exposed.

Acceptance Criteria:
- Secrets stored in baseapp/_secrets/ directory (excluded from git)
- Environment-specific secrets for dev, uat, prod
- AWS credentials use IAM roles (Lambda) or access keys (externalized)
- Third-party API keys (ActiveCampaign, Telegram) externalized
- No secrets committed to version control

Dependencies: _secrets module, .gitignore

REQ-NFR-SEC-009: Audit Logging

Type: Non-Functional
Priority: Must Have
Description: All data modifications shall be logged with user, timestamp, and changes for audit trail.

Rationale: Audit trail enables investigation of security incidents and data disputes.

Acceptance Criteria:
- All model create, update, delete operations logged
- Logs capture: model, user, timestamp, operation, before/after values
- Logs immutable (no edits or deletions)
- Logs retained for minimum 12 months

Dependencies: django-easy-audit

4.3 Reliability Requirements

REQ-NFR-REL-001: System Availability

Type: Non-Functional
Priority: Must Have
Description: The system shall maintain 99.5% uptime during business hours (6 AM - 10 PM SAST, Monday-Sunday).

Rationale: High availability ensures investors and agents can access platform when needed, especially during peak usage hours.

Acceptance Criteria:
- Maximum downtime: 3.6 hours per month during business hours
- Planned maintenance scheduled during off-peak hours (10 PM - 6 AM)
- Uptime monitored using health check endpoints
- Incident response within 1 hour for critical outages

Testing: Monitor uptime using external monitoring service (e.g., UptimeRobot, Pingdom)

REQ-NFR-REL-002: Error Handling

Type: Non-Functional
Priority: Must Have
Description: Application errors shall be handled gracefully with user-friendly messages and logged for debugging.

Rationale: Graceful error handling improves user experience and enables rapid issue resolution.

Acceptance Criteria:
- Unhandled exceptions caught by global error handler
- User sees friendly error message (not stack traces)
- Errors logged to Sentry with full context (user, request, stack trace)
- 500 errors automatically alert development team

Dependencies: Sentry integration

REQ-NFR-REL-003: Data Backup

Type: Non-Functional
Priority: Must Have
Description: Database shall be backed up daily with 30-day retention.

Rationale: Regular backups enable recovery from data loss due to hardware failure, software bugs, or user errors.

Acceptance Criteria:
- Automated daily backups at 2 AM SAST
- Backup retention: 30 days
- Backups stored in geographically separate location from primary database
- Backup restoration tested quarterly

Testing: Test backup restoration on non-production environment

---

REQ-NFR-REL-004: Disaster Recovery

Type: Non-Functional
Priority: Should Have
Description: System shall be recoverable within 4 hours in case of catastrophic failure.

Rationale: Disaster recovery plan minimizes business impact from major system failures.

Acceptance Criteria:
- Recovery Time Objective (RTO): 4 hours
- Recovery Point Objective (RPO): 24 hours (daily backup)
- Documented disaster recovery procedures
- Annual disaster recovery drill

Testing: Annual disaster recovery simulation

4.4 Usability Requirements

REQ-NFR-USA-001: User Interface Responsiveness

Type: Non-Functional
Priority: Must Have
Description: User interface shall be fully responsive and functional on desktop, tablet, and mobile devices.

Rationale: Users access platform from various devices. Responsive design ensures usability across all screen sizes.

Acceptance Criteria:
- Functional on screen sizes from 320px (mobile) to 2560px (desktop)
- Touch-friendly controls on mobile (minimum 44px tap targets)
- Forms usable on mobile devices
- Navigation adapts to mobile (hamburger menu)

Testing: Test on real devices (iPhone, Android phone, iPad, desktop) and browser dev tools

---

REQ-NFR-USA-002: Browser Compatibility

Type: Non-Functional
Priority: Must Have
Description: System shall function correctly on modern browsers: Chrome, Firefox, Safari, Edge (latest 2 versions).

Rationale: Cross-browser compatibility ensures all users can access platform regardless of browser choice.

Acceptance Criteria:
- Full functionality on Chrome, Firefox, Safari, Edge (latest 2 versions)
- Graceful degradation on older browsers (display message recommending upgrade)
- No browser-specific bugs in core workflows

Testing: Manual testing on target browsers, automated cross-browser testing

---

REQ-NFR-USA-003: Accessibility

Type: Non-Functional
Priority: Should Have
Description: User interface shall follow WCAG 2.1 Level AA accessibility guidelines.

Rationale: Accessible design ensures platform usable by users with disabilities, expanding potential user base.

Acceptance Criteria:
- Semantic HTML with proper heading hierarchy
- ARIA labels on interactive elements
- Keyboard navigation functional for all features
- Sufficient color contrast (4.5:1 for normal text)
- Form fields have associated labels

Testing: Automated accessibility testing (axe, WAVE), manual keyboard navigation testing

---

REQ-NFR-USA-004: User Feedback and Validation

Type: Non-Functional
Priority: Must Have
Description: Forms shall provide immediate validation feedback and clear error messages.

Rationale: Immediate feedback helps users correct errors quickly, improving form completion rates.

Acceptance Criteria:
- Inline validation on form fields (real-time or on blur)
- Clear, specific error messages (not generic)
- Success messages for completed actions
- Loading indicators during async operations
- Disabled buttons during processing to prevent double-submission

Dependencies: Frontend form validation

4.5 Maintainability Requirements

REQ-NFR-MNT-001: Code Quality Standards

Type: Non-Functional
Priority: Should Have
Description: Code shall follow PEP 8 style guide (Python) and ESLint rules (JavaScript).

Rationale: Consistent code style improves readability and maintainability.

Acceptance Criteria:
- Python code passes PEP 8 linting (flake8 or pylint)
- JavaScript code passes ESLint with configured rules
- Code review required before merging to main branch
- Automated linting in CI/CD pipeline

Dependencies: Linting tools, CI/CD pipeline

REQ-NFR-MNT-002: Documentation

Type: Non-Functional
Priority: Should Have
Description: Code shall include docstrings, inline comments, and API documentation.

Rationale: Good documentation enables faster onboarding and easier maintenance.

Acceptance Criteria:
- Python functions have docstrings explaining purpose, parameters, return values
- Complex logic includes inline comments
- API endpoints documented with drf-spectacular (OpenAPI/Swagger)
- GraphQL schema includes field descriptions

Dependencies: drf-spectacular

REQ-NFR-MNT-003: Modular Architecture

Type: Non-Functional
Priority: Must Have
Description: System shall maintain modular architecture with clear separation of concerns.

Rationale: Modular design enables independent development, testing, and deployment of components.

Acceptance Criteria:
- Django apps focused on single business domain (property, investment, user_account, approvals)
- Business logic in models/services, not views

- Reusable components (approval engine, notification system)
- Minimal coupling between apps

Dependencies: Django app structure

REQ-NFR-MNT-004: Database Migrations

Type: Non-Functional
Priority: Must Have
Description: Database schema changes shall use Django migrations for version control and deployment.

Rationale: Migrations enable controlled, reversible database changes across environments.

Acceptance Criteria:
- All schema changes implemented as Django migrations
- Migrations tested on dev/uat before production deployment
- Migrations include data migrations when needed
- Migration files committed to version control

Dependencies: Django migrations

4.6 Compliance Requirements

REQ-NFR-CMP-001: Data Privacy (POPIA Compliance)

Type: Non-Functional
Priority: Must Have
Description: System shall comply with South African Protection of Personal Information Act (POPIA).

Rationale: Legal requirement for processing personal information of South African citizens.

Acceptance Criteria:
- Privacy policy published and accessible
- User consent obtained for data processing (agreement execution)
- Users can request data deletion (right to be forgotten)
- Data minimization (only collect necessary information)
- Data retention policies defined and enforced

Dependencies: Legal review, Privacy policy

REQ-NFR-CMP-002: Terms of Service

Type: Non-Functional
Priority: Must Have

Description: Users shall be required to accept Terms of Service (Proposal to Service agreement) before platform use.

Rationale: Terms of Service establish legal relationship and protect platform from liability.

Acceptance Criteria:
- Terms of Service displayed during registration/first login
- User acceptance captured with timestamp, IP, user agent
- Terms of Service accessible at all times (footer link)
- Updated terms require re-acceptance

Dependencies: Agreement system, Contract execution

REQ-NFR-CMP-003: Financial Disclosure

Type: Non-Functional
Priority: Must Have
Description: Investment calculations shall include clear disclaimers that projections are estimates, not guarantees.

Rationale: Financial projections are estimates; disclaimers protect platform from liability for investment losses.

Acceptance Criteria:
- Investment summary includes disclaimer: "Projections are estimates only. Actual returns may vary."
- Sourcing fee calculation clearly displayed
- All assumptions (interest rate, rental escalation) clearly stated

Dependencies: Investment summary display

---

5. Data Requirements

5.1 Core Data Entities

Entity: User

Description: Platform user authentication and basic information
Attributes:
- id (Integer, PK)
- username (String, unique, max 150 chars) - stores email address
- email (String, unique, max 254 chars)
- first_name (String, max 150 chars)

- last_name (String, max 150 chars)
- password (String, hashed)
- is_active (Boolean)
- date_joined (DateTime)
- last_login (DateTime)

Relationships:
- Has one Account
- Has one or more UserProfile
- Has many ContactDetail
- Has many Agreement executions

Volumes: Estimated 5,000 users in first 18 months, growth to 50,000 users in 5 years

---

Entity: Account

Description: Business account container for user, extends Django User model
Attributes:
- id (Integer, PK)
- uid (UUID, unique)
- owner (FK to User)
- needs_updating (Boolean, default True)
- created (DateTime)
- modified (DateTime)

Relationships:
- Belongs to one User
- Has many Listings
- Has many Contract executions

Volumes: 1:1 with User, ~5,000 accounts

Entity: UserProfile

Description: Extended user information including roles and metadata
Attributes:
- id (Integer, PK)
- uid (UUID, unique)
- user (FK to User)
- needs_updating (Boolean)
- created (DateTime)
- modified (DateTime)

Relationships:
- Belongs to one User
- Has many Roles (many-to-many)
- Has many MetaData
- Has many MetaDataTypes needing updating (many-to-many)

Volumes: 1:1 with User, ~5,000 profiles

Entity: Listing

Description: Core property listing entity
Attributes:
- id (Integer, PK)
- uid (UUID, unique)
- name (String, max 4096 chars)
- lister_user (FK to User, nullable)
- lister_account (FK to Account)
- owner (FK to User, nullable)
- listing_type (FK to ListingType)
- status (FK to ListingStatus)
- approval (FK to Approval, one-to-one)
- description (Text, nullable)
- amenities (Text, nullable)
- address (FK to ListingAddress)
- allows_subletting (Boolean, default False)
- max_occupant_count (Integer, nullable)
- total_expected_rental_income (Money, ZAR)
- total_current_rental_income (Money, ZAR)
- market_price (Money, ZAR)
- buying_price (Money, ZAR)
- created (DateTime)
- modified (DateTime)

Relationships:
- Belongs to one Account (lister_account)
- Has one Approval
- Has one Address
- Has many Subunits
- Has many Artifacts
- Has many Modifications
- Has one or more Investments
- Has one RentalModel

Volumes: 10,000 listings by year 2, 100,000 listings by year 5

Entity: Subunit

Description: Rentable unit within a listing (supports hierarchical multi-let structures)
Attributes:
- id (Integer, PK)
- listing (FK to Listing)
- description (Text, nullable)
- amenities (Text, nullable)
- subunit_type (FK to SubunitType)
- parent_subunit (FK to self, nullable)
- current_rental_approach (FK to RentalApproach)
- future_rental_approach (FK to RentalApproach)
- future_rental_approach_months (Integer, default 12)
- size_m2 (Decimal, nullable)
- expected_rental_income (Money, ZAR)
- current_rental_income (Money, ZAR)
- to_be_created (Boolean, default False)
- bathroom_count (Decimal, nullable)
- bedroom_count (Decimal, nullable)
- [Boolean amenity fields: is_partitioned, has_private_kitchen, has_shared_kitchen, etc.]
- floor_number (Integer, default 0)
- max_occupant_count (Integer, nullable)
- created (DateTime)
- modified (DateTime)

Relationships:
- Belongs to one Listing
- Can have one parent Subunit
- Can have many child Subunits
- Has many Leases

Volumes: Average 3 subunits per listing, ~30,000 subunits by year 2

Entity: Investment

Description: Financial investment vehicle for a property listing
Attributes:
- id (Integer, PK)
- listing (FK to Listing)
- created (DateTime)
- modified (DateTime)

Relationships:
- Belongs to one Listing
- Has many TransactionCosts

- Has many RecurringExpenses
- Has many Financings

Volumes: 1-2 per listing (default + investor scenarios), ~15,000 investment records by year 2

Entity: Financing

Description: Financing scenario for an investment (Bond or Cash)
Attributes:
- id (Integer, PK)
- investment (FK to Investment)
- financing_type (FK to FinancingType)
- is_default (Boolean, default False)
- created (DateTime)
- modified (DateTime)

Relationships:
- Belongs to one Investment
- Has zero or one Bond (if type is Bank Bond)
- Has many TransactionCosts
- Has many RecurringExpenses

Volumes: 1-3 per investment (exploring scenarios), ~20,000 financing records by year 2

Entity: Bond

Description: Bond financing details and calculations
Attributes:
- id (Integer, PK)
- financing (FK to Financing)
- projected_interest_rate_percentage (Decimal, default 9.75)
- actual_interest_rate_percentage (Decimal, default 0)
- term_months (Integer, default 240)
- monthly_repayments (Money, ZAR)
- total_bond_principal_amount (Money, ZAR)
- deposit (Money, ZAR)
- created (DateTime)
- modified (DateTime)

Relationships:
- Belongs to one Financing

Volumes: ~15,000 bonds by year 2 (most investments use bond financing)

Entity: Approval

Description: Generic approval workflow container
Attributes:
- id (Integer, PK)
- uid (UUID, unique)
- item_type (FK to ItemType)
- is_closed (Boolean, default False)
- state (FK to State)
- created (DateTime)
- modified (DateTime)

Relationships:
- Has many StateUpdates

Volumes: 1:1 with Listing for listing approvals, ~10,000 approvals by year 2

Entity: Contract

Description: Legal agreement template
Attributes:
- id (Integer, PK)
- name (String, max 1024 chars)
- contract_type (FK to ContractType)
- created (DateTime)
- modified (DateTime)

Relationships:
- Has many Versions
- Has many Executions

Volumes: <10 contract templates (Proposal to Service primary)

Entity: Execution

Description: Record of user executing (signing) a contract
Attributes:
- id (Integer, PK)
- account (FK to Account)
- user (FK to User)
- contract (FK to Contract)
- contract_version (FK to Version)
- location (String, max 2048 chars, nullable)
- gps_long (Decimal, 9 digits, 6 decimal places, nullable)

- gps_lat (Decimal, 9 digits, 6 decimal places, nullable)
- ip (String, max 1024 chars, nullable)
- platform (String, max 4096 chars, nullable)
- user_agent (String, max 4096 chars, nullable)
- created (DateTime)

Relationships:
- Belongs to one Account
- Belongs to one User
- Belongs to one Contract
- Belongs to one Version

Volumes: 5,000 executions (1 per user for Proposal to Service)

Entity: Notification

Description: Notification type definition
Attributes:
- id (Integer, PK)
- app_name (String, max 1024 chars)
- name (String, max 1024 chars)
- communication_medium (FK to CommunicationMedium)
- template (FK to Template, nullable)
- delivery_endpoint (FK to DeliveryEndpoint)
- created (DateTime)
- modified (DateTime)

Relationships:
- Has many Subscriptions
- Has many Jobs

Volumes: ~50 notification types across all modules

5.2 Entity Relationships

Listing Hierarchy:

```
Account → Listing → Subunit (hierarchical) → Lease
Listing → Investment → Financing → Bond
Listing → Approval → StateUpdate
```

User Hierarchy:

```
User → Account → Agreement Execution
User → UserProfile → Roles
User → ContactDetail
```

Notification Flow:

```
NotificationGroup → Notification → Subscription (per user)
Notification → Job (delivery record)
```

5.3 Data Volumes and Growth

Entity	Initial (6 months)	Year 1	Year 2	Year 5
Users	500	2,000	5,000	50,000
Listings	100	1,000	10,000	100,000
Subunits	300	3,000	30,000	300,000
Investments	150	1,500	15,000	150,000
Approvals	100	1,000	10,000	100,000
StateUpdates	300	3,000	30,000	300,000
Artifacts	500	5,000	50,000	500,000
Notification Jobs	5,000	50,000	500,000	5,000,000

Storage Projections:
- Database: 100 MB (initial) → 1 GB (year 1) → 10 GB (year 2) → 100 GB (year 5)
- S3 Storage (artifacts): 1 GB (initial) → 50 GB (year 1) → 500 GB (year 2) → 5 TB (year 5)

5.4 Data Quality Requirements

Accuracy

- Financial calculations (bond repayments, ROI) must be 100% accurate
- GPS coordinates accurate to 6 decimal places

- Money fields store amounts with 2 decimal precision

Completeness

- Listings require: name, listing type, address, buying price before approval submission
- User profiles require: email, first_name, last_name, phone_number
- Contract executions capture complete metadata (timestamp, IP, user agent)

Consistency

- Rental income aggregations consistent across subunit hierarchy
- Status and approval state remain synchronized
- Only one default financing per investment enforced

Timeliness

- Real-time updates for approval state changes
- Rental income aggregations update within 1 second of subunit changes
- Notification jobs processed within 5 minutes of creation

5.5 Data Lifecycle Policies

Data Retention

- User accounts: Retained indefinitely while active; 7-year retention after account closure (legal requirement)
- Listings: Retained indefinitely (historical record)
- Audit logs: 12-month retention minimum, 7-year retention recommended
- Notification jobs: 90-day retention (purge older records)
- S3 artifacts: Retained indefinitely (manual cleanup for deleted listings)

Data Archival

- Listings with status "Sold" for >2 years archived to cold storage
- Notification jobs >90 days archived and purged from active database

Data Deletion

- User-requested deletion: Anonymize user data, retain audit trail with anonymized user ID
- GDPR/POPIA right to be forgotten: Anonymize personal data, maintain business transaction records

---

6. Integration Requirements

6.1 External Systems and APIs

INT-001: Active Campaign CRM Integration

Description: Integration with Active Campaign for marketing automation and CRM
Integration Type: REST API (outbound)
Purpose:
- Sync user registrations to CRM
- Track user engagement for marketing campaigns
- Send targeted email campaigns

Data Exchange:
- User registration data (email, name, registration date)
- Listing view events
- Agreement execution events

Frequency: Real-time via event triggers
Authentication: API Key (stored in _secrets)
Dependencies: activecampaign-python library
Error Handling: Failed syncs logged; retry mechanism for transient failures

Configuration:

```
# baseapp/_secrets/_active_campaign_secrets.py
ACTIVE_CAMPAIGN_API_KEY = "secret_key"
ACTIVE_CAMPAIGN_BASE_URL = "https://account.api-us1.com"
```

INT-002: Telegram Bot Integration

Description: Telegram bot for real-time notifications and alerts
Integration Type: Telegram Bot API (outbound)
Purpose:
- Send approval status notifications to admins
- Alert sourcing agents of listing status changes
- Provide real-time platform alerts

Data Exchange:
- Listing approval state changes
- System alerts and errors
- User activity notifications

Frequency: Real-time via event triggers
Authentication: Bot Token (stored in _secrets)
Dependencies: python-telegram-bot library
Error Handling: Failed message sends logged; no retry (non-critical)

Configuration:

```
# baseapp/_secrets/_telegram_secrets.py
TELEGRAM_BOT_TOKEN = "secret_token"
TELEGRAM_CHAT_ID = "chat_id"
```

INT-003: AWS S3 File Storage

Description: AWS S3 for property artifact and service provider picture storage
Integration Type: AWS SDK (boto3)
Purpose:
- Store property listing photos
- Store property documents (leases, financial reports, offers)
- Store service provider pictures
- Serve files via public HTTPS URLs

Data Exchange:
- File uploads (multipart/form-data)
- File metadata (filename, size, upload timestamp)

Frequency: On-demand (user-initiated uploads)
Authentication: AWS Access Key ID + Secret Access Key (stored in _secrets)
Region: af-south-1 (South Africa)
Buckets:
- investrand-backend-dev (development)
- investrand-backend-prod (production/UAT)

Dependencies: boto3, django-s3-storage, django-storages

File Naming Convention:
- Property artifacts: property_artifacts/{uuid}.{extension}
- Service provider pictures: service_provider_pictures/{uuid}.{extension}

ACL: public-read (all files publicly accessible)

Configuration:

```
# baseapp/_secrets/_aws_secrets.py
AWS_ACCESS_KEY_ID = "access_key"
AWS_SECRET_ACCESS_KEY = "secret_key"
AWS_REGION = "af-south-1"
AWS_S3_BUCKET_NAME = "investrand-backend-prod"
```

Future Enhancement: Migrate to private S3 buckets with pre-signed URLs for enhanced security

---

INT-004: Sentry Error Tracking

Description: Sentry for real-time error tracking and monitoring
Integration Type: Sentry SDK (outbound)
Purpose:
- Capture unhandled exceptions
- Track application errors
- Monitor performance issues
- Alert developers of critical errors

Data Exchange:
- Exception stack traces
- Request context (URL, user, headers)
- Performance metrics

Frequency: Real-time on error occurrence
Authentication: DSN (currently hardcoded, should be moved to _secrets)
Dependencies: sentry-sdk

Current DSN (SECURITY ISSUE):

```
# baseapp/settings/components/sentry_settings.py
# TODO: Move to _secrets
dsn="https://dc6338422fe34d9fa9857289eefe6e7a@o4505420590678016.ingest.sentry.io/4505420592644096"
```

Recommendation: Externalize DSN to _secrets immediately

INT-005: Email Delivery (SMTP)

Description: Email delivery for notifications
Integration Type: SMTP (outbound)
Purpose:
- Send user registration confirmation emails
- Send approval status notifications

- Send password reset emails
- Send marketing emails (via Active Campaign)

Data Exchange:
- Email content (HTML and plain text)
- Recipient addresses
- Subject lines

Frequency: Real-time via notification system
Authentication: SMTP credentials (stored in _secrets)
Dependencies: Django email backend

Configuration:

```
# baseapp/_secrets/_email_secrets.py
EMAIL_HOST = "smtp.example.com"
EMAIL_PORT = 587
EMAIL_HOST_USER = "user@example.com"
EMAIL_HOST_PASSWORD = "password"
EMAIL_USE_TLS = True
DEFAULT_FROM_EMAIL = "noreply@investrand.co.za"
```

INT-006: Google Places API (Frontend)

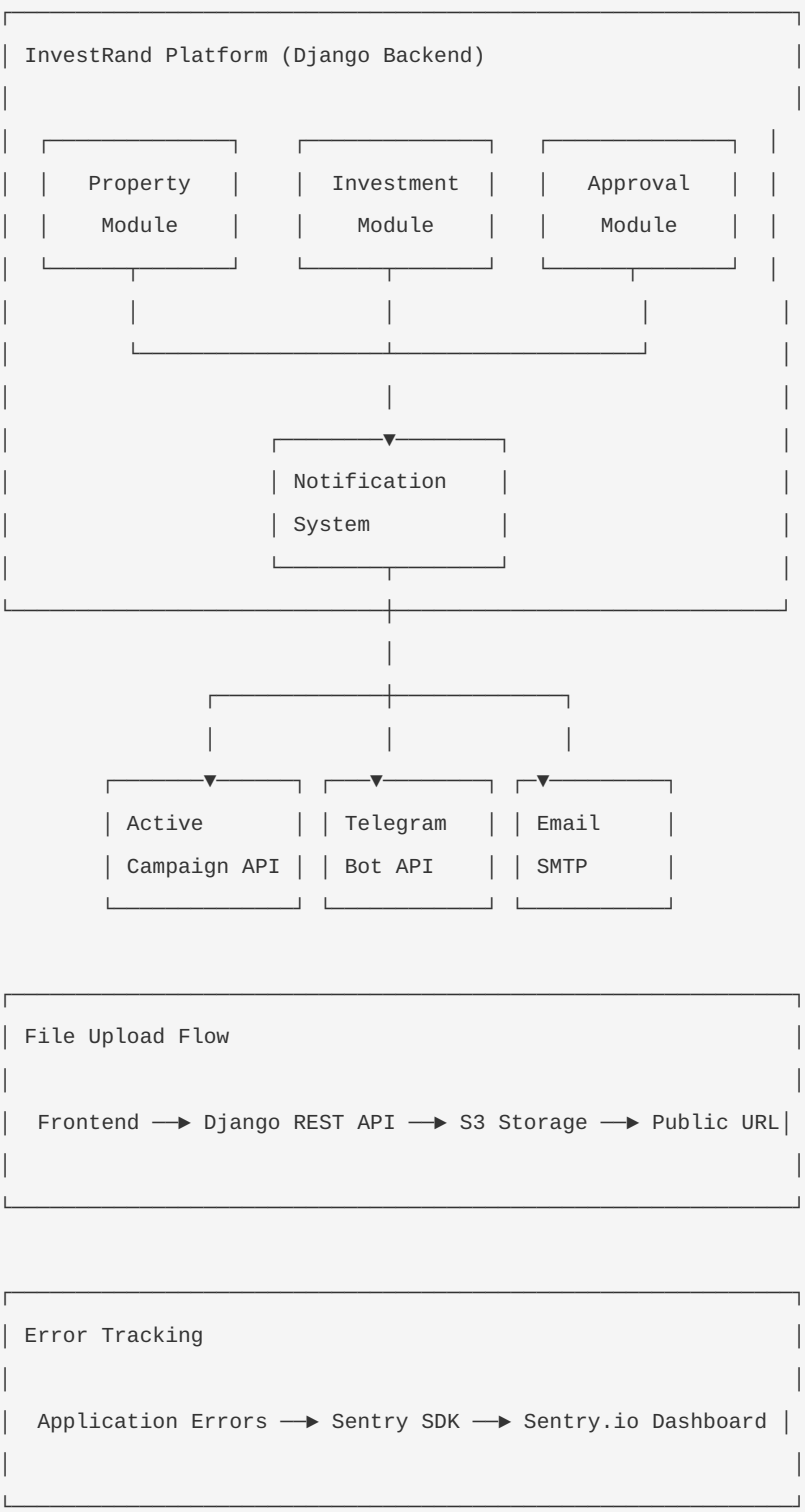
Description: Google Places API for address autocomplete (frontend integration)
Integration Type: JavaScript API (frontend only, no backend integration)
Purpose:
- Address autocomplete during listing creation
- GPS coordinate lookup
- Standardized address formatting

Data Exchange:
- Address search queries
- Place IDs, GPS coordinates, formatted addresses

Frequency: On-demand (user typing in address field)
Authentication: Google API Key (frontend configuration)
Dependencies: Google Maps JavaScript API

Note: Backend stores Google address data (g_address, g_place_id, gps_lat, gps_long) but does not directly integrate with Google API

6.2 Integration Architecture



6.3 Integration Protocols and Data Formats

Integration	Protocol	Data Format	Authentication
Active Campaign	HTTPS/REST	JSON	API Key (Header)
Telegram Bot	HTTPS/REST	JSON	Bot Token (URL param)
AWS S3	HTTPS/REST	Binary (files)	AWS Signature v4
Sentry	HTTPS	JSON	DSN
Email SMTP	SMTP/TLS	MIME (HTML + plain text)	Username/Password
Google Places	N/A (frontend)	JSON	API Key

6.4 Integration Security

1. **API Key Management:**
   - All API keys stored in _secrets module (excluded from git)
   - Environment-specific keys for dev, uat, prod
   - Keys rotated annually or on suspected compromise

2. **AWS Security:**
   - S3 bucket access restricted by IAM policy
   - Access keys never committed to version control
   - Future: Migrate to IAM roles for Lambda execution

3. **Email Security:**
   - SMTP over TLS enforced
   - SPF and DKIM records configured for domain
   - DMARC policy enforced

4. **HTTPS Enforcement:**
   - All external API calls use HTTPS
   - SSL certificate validation enforced (not disabled)

6.5 Integration Monitoring and Error Handling

Active Campaign:
- Log all sync attempts (success/failure)

- Retry failed syncs up to 3 times with exponential backoff
- Alert admins if sync fails for 24+ hours

Telegram:
- Log all message sends
- No retry on failure (non-critical notifications)
- Fallback to email if Telegram unavailable

S3 Uploads:
- Retry failed uploads up to 2 times
- Return clear error message to user on failure
- Monitor S3 upload success rate (target: >99%)

Email Delivery:
- Queue emails for retry if SMTP server unavailable
- Log all send attempts with delivery status
- Alert admins if email queue backs up (>100 messages)

Sentry:
- Alert on critical errors immediately
- Daily summary of error trends
- Automatic grouping of similar errors

---

7. User Interface Requirements

7.1 UI/UX Expectations

Design Principles

1. **Clarity:** Investment data presented clearly with visual hierarchy
2. **Efficiency:** Common tasks (listing creation, discovery) require minimal clicks
3. **Trustworthiness:** Professional design inspires confidence in platform
4. **Responsiveness:** Functional on desktop, tablet, mobile

Visual Design

- **Color Scheme:** Professional palette (primary, secondary, accent colors)
- **Typography:** Clear, readable fonts (minimum 14px body text)
- **Iconography:** Consistent icon set (e.g., Material Icons)
- **Spacing:** Adequate whitespace for readability

7.2 Screen Flows and User Journeys

Journey 1: Investor Discovers and Views Listing

```
Home/Login → Discover Listings (search/filter) → Listing Details
                                                        ↓
                                                View Investment Summary
                                                        ↓
                                                Express Interest (future)
```

Key Screens:

1. **Discover Listings:** Grid/list view with filters (location, price, type)
2. **Listing Detail:** Photos, description, address map, amenities
3. **Investment Summary:** Financial breakdown, ROI calculation, assumptions

---

Journey 2: Sourcing Agent Creates Listing

```
Login → My Listings → Create Listing → Fill Details → Upload Photos
                                                ↓
                                        Add Subunits (multi-let)
                                                ↓
                                        Configure Investment Model
                                                ↓
                                        Preview Listing → Submit for Approval
```

Key Screens:

1. **Create Listing:** Multi-step form (property details, address, pricing)
2. **Subunit Management:** Add/edit subunits, set rental income
3. **Investment Configuration:** Set financing type, bond details, expenses
4. **Preview:** View listing as investors will see it
5. **My Listings Dashboard:** List of all listings by status

Journey 3: Admin Reviews and Approves Listing

```
Login → Manage Listings → Filter by "Submitted" → View Listing Details
                                                        ↓
                                                Review Completeness
                                                        ↓
                                        Approve (publish) OR Reject (with comment)
```

Key Screens:

1. **Manage Listings:** Table view with filters, status indicators
2. **Listing Review:** Full listing details + approval controls
3. **Approval Comment:** Text area for rejection reason or feedback

---

Journey 4: New User Registration and Agreement

```
Registration Page → Fill Form (email, password, name) → Submit
                                                          ↓
                                                Email Verification (optional)
                                                          ↓
                                                Login → Agreement Page
                                                          ↓
                                                Read Agreement → Accept
                                                          ↓
                                                Complete Profile (contact details)
                                                          ↓
                                                Dashboard
```

Key Screens:

1. **Registration:** Form with email, password (with strength indicator), first/last name
2. **Agreement Acceptance:** Full contract text with scroll-to-bottom, accept button
3. **Profile Completion:** Contact details, role selection (if applicable)
4. **Dashboard:** Personalized based on role (investor: discover listings; agent: my listings)

7.3 Input Requirements

Forms

- **Listing Creation Form:**
- Text inputs: Name, description, amenities
- Address autocomplete: Google Places integration
- Money inputs: Buying price, market price (with currency symbol, thousands separator)
- Dropdowns: Listing type, rental approach
- Number inputs: Bedroom count, bathroom count, max occupants
- Checkboxes: Amenities (furnished, pets allowed, etc.)
- File uploads: Photos, documents (drag-and-drop or click to browse)

- **Subunit Form:**

- Similar to listing form but scoped to subunit
- Parent subunit selector (hierarchical dropdown)
- **Investment Configuration Form:**
- Radio buttons: Financing type (Bank Bond / Cash)
- Conditional fields: Bond details (interest rate, term, deposit)
- Repeating sections: Transaction costs, recurring expenses
- **User Registration Form:**
- Email input with validation
- Password input with strength indicator and confirmation
- Text inputs: First name, last name

Validation

- **Client-side validation:** Immediate feedback on invalid input (before form submission)
- **Server-side validation:** Final validation with detailed error messages
- **Required field indicators:** Asterisk or label styling for required fields
- **Format validation:** Email format, phone number format, numeric ranges

---

7.4 Output Requirements

Reports

- **Investment Summary Report:** Exportable PDF showing complete financial breakdown
- **Listing Analytics:** Views, inquiries, time to approval (admin only)
- **Portfolio Summary:** Total investment, properties, projected income (investor)

Dashboards

- **Investor Dashboard:**
- Featured listings (newly published)
- Saved searches and alerts
- Portfolio summary (future)
- **Sourcing Agent Dashboard:**
- My listings by status (Created, In Review, Published, Sold)
- Recent activity (views, inquiries)
- Earnings summary (sourcing fees from sold properties)

- **Admin Dashboard:**
- Pending approvals count with drill-down
- System health metrics (uptime, error rate)
- User registrations and activity trends

Visualizations

- **Investment ROI Chart:** Bar/line chart showing projected returns over time
- **Rental Income Breakdown:** Pie chart showing income by subunit
- **Expense Breakdown:** Pie chart showing transaction costs and recurring expenses
- **Location Map:** Interactive map showing listing location (Google Maps embed)

7.5 Device Support Requirements

Desktop

- **Screen sizes:** 1280px - 2560px width
- **Primary use case:** Listing creation, detailed investment analysis
- **Features:** Full feature set, multi-column layouts, advanced filtering

Tablet

- **Screen sizes:** 768px - 1024px width
- **Primary use case:** Listing discovery and viewing
- **Features:** Responsive layout, touch-optimized controls, simplified navigation

Mobile

- **Screen sizes:** 320px - 767px width
- **Primary use case:** Browse listings, view notifications
- **Features:** Mobile-first navigation (hamburger menu), stacked layouts, essential features only
- **Limitations:** Listing creation on mobile not optimized (recommend desktop for complex forms)

7.6 Accessibility Requirements

- **Keyboard Navigation:** All interactive elements accessible via Tab, Enter, Space, Arrow keys
- **Screen Reader Support:** ARIA labels on all form controls, buttons, and navigation elements
- **Color Contrast:** WCAG AA compliance (4.5:1 for normal text, 3:1 for large text)

- **Focus Indicators:** Visible focus state on all interactive elements
- **Form Labels:** All input fields have associated labels (not just placeholders)
- **Error Announcement:** Screen readers announce form validation errors

---

8. Technical Constraints

8.1 Technology Stack

Backend

- **Framework:** Django 4.2.15
- **Language:** Python 3.9 / 3.12
- **API:** Django REST Framework 3.15.2 + Graphene-Django 3.2.2 (GraphQL)
- **Authentication:** djangorestframework-simplejwt (JWT)
- **Database:** MySQL (production), SQLite (development)
- **ORM:** Django ORM with migrations

Rationale: Django provides robust, batteries-included framework suitable for complex business logic. Dual API support (REST + GraphQL) provides flexibility for different client needs.

Frontend

- **Framework:** Vue.js 2.x
- **Language:** JavaScript (ES6+)
- **Routing:** Vue Router
- **State Management:** Vuex (implied by store directory)
- **API Client:** Apollo Client (GraphQL), Axios (REST)
- **Build Tool:** Webpack (via Vue CLI)

Rationale: Vue.js provides reactive, component-based UI development with gentle learning curve.

Infrastructure

- **Cloud Provider:** AWS (locked to South Africa region: af-south-1)
- **Deployment:**
- Serverless: AWS Lambda + Zappa (Python 3.9, 300s timeout)

- Containerized: Docker + Gunicorn + Nginx
- **Storage:** AWS S3 (af-south-1)
- **Reverse Proxy:** Traefik with Let's Encrypt SSL
- **CDN:** None currently (future enhancement)

Rationale: AWS South Africa region ensures low-latency access for South African users. Lambda provides cost-effective scaling for variable traffic.

8.2 Infrastructure Requirements

AWS Resources

- **Lambda Functions:** Zappa deployment (Python 3.9 runtime)
- **VPC:** Subnets (subnet-0d1dd7887841512ed, subnet-075f2f8925ca62306)
- **Security Groups:** sg-0ad0cad02e74e26b8
- **S3 Buckets:**
- investrand-backend-dev (development)
- investrand-backend-prod (production/UAT)
- **Certificate:** AWS Certificate Manager (arn:aws:acm:us-east-1:629836046130:certificate/a2b81380-67d4-4597-883d-32592a24cb0b)

Note: Infrastructure IDs currently hardcoded in zappa_settings.json. Recommendation: Externalize to environment-specific configuration.

Domains

- **Production:** api.platform.investrand.co.za (backend), platform.investrand.co.za (frontend)
- **UAT:** uat-api-platform.investrand.co.za (backend), uat-platform.investrand.co.za (frontend)
- **Development:** dev-api-platform.investrand.co.za (backend), dev-platform.investrand.co.za (frontend)

Database

- **Engine:** MySQL (production), SQLite (local development)
- **Connection Pooling:** Required for Lambda (due to short-lived execution contexts)
- **Migrations:** Django migrations for schema version control
- **Backup:** Daily automated backups with 30-day retention

8.3 Scalability Expectations

Concurrent Users

- **Current:** 100 concurrent users
- **Year 1:** 500 concurrent users
- **Year 2:** 1,000 concurrent users

Data Volume

- **Listings:** 10,000 by year 2, 100,000 by year 5
- **S3 Storage:** 500 GB by year 2, 5 TB by year 5
- **Database:** 10 GB by year 2, 100 GB by year 5

Performance Targets

- **Page Load:** < 3 seconds
- **API Response:** < 500ms (95th percentile)
- **File Upload:** < 10 seconds for 5MB files

Scaling Strategy:
- **Lambda:** Auto-scales based on traffic
- **Database:** Vertical scaling (upgrade instance size) or read replicas if needed
- **S3:** Inherently scalable
- **Frontend:** CDN (future enhancement) for static asset delivery

---

8.4 Platform Support

Server Environments

- **Operating System:** Linux (AWS Lambda, Docker containers)
- **Python Version:** 3.9 (Lambda), 3.12 (Docker option)
- **Node.js Version:** 14+ (frontend build)

Client Environments

- **Browsers:** Chrome, Firefox, Safari, Edge (latest 2 versions)
- **Operating Systems:** Windows 10+, macOS 10.14+, iOS 13+, Android 9+
- **Minimum Screen Resolution:** 320px width (mobile)

8.5 Multi-Currency Support (Future Enhancement)

Current State:
- All money fields use ZAR (South African Rand)
- Currency hardcoded in Money field defaults and calculations

Future Enhancement:
- Support multi-currency for international expansion
- Currency selection at listing creation
- Exchange rate integration for cross-currency comparisons
- Currency conversion for investor portfolios

Implementation Notes: Requires changes throughout codebase wherever Money fields and calculations assume ZAR.

8.6 Legacy System Considerations

None. InvestRand is a greenfield project with no legacy system integrations.

---

9. Requirements Traceability Matrix

9.1 Business Goals to Functional Requirements

Business Goal	Related Requirements	Traceability
Market Leadership	REQ-PLM-001 to REQ-PLM-013 (Comprehensive listing management)	Robust listing features differentiate platform
Transaction Volume	REQ-INV-001 to REQ-INV-010 (Investment modeling)	Accurate financial tools drive investor confidence → transactions
User Acquisition	REQ-UAM-001 to REQ-UAM-005 (Easy onboarding)	Streamlined registration and onboarding reduces friction
Platform Revenue	REQ-INV-003 (Sourcing fee calculation)	Automated sourcing fee calculation ensures revenue capture
Quality Assurance	REQ-AWE-001 to REQ-AWE-004 (Approval workflows)	Approval system maintains listing quality

---

9.2 Stakeholder Needs to Requirements

Stakeholder	Key Needs	Addressing Requirements
Investors	Accurate ROI calculations	REQ-INV-001 to REQ-INV-010
Investors	Easy property discovery	REQ-PLM-011 (Listing discovery)
Sourcing Agents	Efficient listing creation	REQ-PLM-001 (Create listing)
Sourcing Agents	Fast approval turnaround	REQ-AWE-001 to REQ-AWE-004 (Approval workflows)
Admins	Oversight and control	REQ-ADM-001 to REQ-ADM-004 (Admin functions)
Service Providers	Visibility and lead generation	REQ-SPM-001 to REQ-SPM-003 (Service provider management)

9.3 Requirements to Design Components

Requirement ID	Design Component	Implementation Notes
REQ-PLM-001	property/models.py: Listing model	1,345 lines, core business logic
REQ-INV-001 to REQ-INV-010	investment/models.py: Investment, Financing, Bond models	594 lines, financial calculations
REQ-AWE-001 to REQ-AWE-004	approvals/models.py: Approval, State, StateUpdate models	Generic workflow engine
REQ-UAM-001 to REQ-UAM-005	user_account/models.py, user_profile/models.py	User management
REQ-CAM-001 to REQ-CAM-004	contracts/models.py: Contract, Version, Execution	Agreement management
REQ-NOT-001 to REQ-NOT-006	notification/models.py: Notification, Job, Subscription	Multi-channel notifications
REQ-SPM-001 to REQ-SPM-003	service_providers/models.py: ServiceProvider	Service provider directory

---

9.4 Requirements to Test Cases

Requirement ID	Test Case ID	Test Scenario
REQ-PLM-001	TC-PLM-001	Create listing with all required fields, verify listing created with status "Created"
REQ-INV-003	TC-INV-003	Create investment, verify sourcing fee = MAX(5% * buying_price, R30,000)
REQ-INV-005	TC-INV-005	Create bond with default parameters, verify monthly repayment calculation accuracy
REQ-AWE-001	TC-AWE-001	Submit listing for approval, verify state changes to "Submitted" and StateUpdate created
REQ-UAM-002	TC-UAM-002	Login with valid credentials, verify JWT access and refresh tokens returned
REQ-NFR-PERF-001	TC-PERF-001	Load listing discovery page, measure load time < 3s
REQ-NFR-SEC-001	TC-SEC-001	Attempt login with incorrect password 6 times, verify account locked for 2 hours

---

9.5 Requirements Change History

Requirement ID	Version	Change Date	Change Description	Reason
N/A	1.0	2025-11-19	Initial requirements specification	Reverse engineering from codebase

---

10. Requirements Prioritization

10.1 MoSCoW Prioritization

Must Have (Critical for Launch)

- **User Management:** REQ-UAM-001 to REQ-UAM-005

- **Listing Management:** REQ-PLM-001 to REQ-PLM-007, REQ-PLM-010 to REQ-PLM-013
- **Investment Modeling:** REQ-INV-001 to REQ-INV-010
- **Approval Workflows:** REQ-AWE-001 to REQ-AWE-004
- **Contract Management:** REQ-CAM-001 to REQ-CAM-004
- **Notification System:** REQ-NOT-001, REQ-NOT-005 (Email)
- **Portfolio Management:** REQ-PFM-001 to REQ-PFM-002
- **Admin Functions:** REQ-ADM-001 to REQ-ADM-004
- **Security:** REQ-NFR-SEC-001 to REQ-NFR-SEC-009
- **Performance:** REQ-NFR-PERF-001 to REQ-NFR-PERF-003

Should Have (Important but not Critical)

- **Property Amenities:** REQ-PLM-008
- **Property Modifications:** REQ-PLM-009
- **Service Provider Management:** REQ-SPM-001 to REQ-SPM-003
- **Lease Tracking:** REQ-LSE-001
- **Notification Templates:** REQ-NOT-002 to REQ-NOT-004, REQ-NOT-006
- **Usability:** REQ-NFR-USA-001 to REQ-NFR-USA-004
- **Maintainability:** REQ-NFR-MNT-001 to REQ-NFR-MNT-004
- **Reliability:** REQ-NFR-REL-001 to REQ-NFR-REL-004

Could Have (Nice to Have)

- **Listing Preview:** REQ-PLM-012
- **Approval Closure:** REQ-AWE-004
- **Notification Groups:** REQ-NOT-004
- **Scalability Testing:** REQ-NFR-PERF-005
- **Accessibility:** REQ-NFR-USA-003

Won't Have (Future Enhancements)

- Multi-currency support (noted throughout as future enhancement)
- Native mobile applications
- Real-time chat functionality
- Automated property valuation
- Virus scanning on file uploads (noted as security recommendation)

10.2 Risk-Based Prioritization

Requirement	Risk Level	Priority	Mitigation
REQ-INV-003 (Sourcing fee calculation)	High	Must Have	Automated testing to ensure 100% accuracy
REQ-NFR-SEC-001 to REQ-NFR-SEC-009	High	Must Have	Security audit, penetration testing
REQ-AWE-001 to REQ-AWE-004 (Approval workflows)	Medium	Must Have	Thorough testing of state transitions
REQ-NFR-PERF-001 to REQ-NFR-PERF-003	Medium	Must Have	Load testing, performance monitoring
REQ-SPM-001 to REQ-SPM-003	Low	Should Have	Can launch without service provider directory

11. Assumptions and Constraints Register

11.1 Assumptions

ID	Assumption	Impact if Invalid	Mitigation
ASM-001	Users have reliable internet access	Platform unusable	Optimize for low-bandwidth scenarios, offline mode (future)
ASM-002	Sourcing agents provide accurate property data	Investor trust eroded	Implement admin review process, data validation
ASM-003	Investors understand rental property investment basics	Poor investment decisions, platform liability	Provide educational content, disclaimers
ASM-004	South African market remains stable	Business model viability	Diversify to other markets (multi-currency support)
ASM-005	Users comfortable with web-based platform	Low adoption	Provide comprehensive onboarding, support resources
ASM-006	ZAR sufficient currency for initial launch	Limits international expansion	Plan multi-currency support for future
ASM-007	AWS South Africa region provides sufficient performance	Latency issues for users outside SA	Consider multi-region deployment (future)

11.2 Constraints

ID	Constraint	Type	Workaround
CON-001	Single currency support (ZAR only)	Technical	Document multi-currency as future enhancement
CON-002	AWS infrastructure limited to af-south-1 region	Infrastructure	Accept latency for non-SA users; future multi-region
CON-003	Public S3 bucket (public-read ACL)	Security	Future: Migrate to private buckets with pre-signed URLs
CON-004	Manual listing approval process	Business Process	Future: Implement automated validation checks
CON-005	Minimum sourcing fee R30,000	Business	Limits properties below R600,000 (5% < R30,000)
CON-006	No multi-tenant support (single platform instance)	Architecture	Accept for now; refactor if needed for white-label
CON-007	Django 4.2.15 (not latest version)	Technical	Plan upgrade path to Django 5.x

12. Glossary of Terms

Term	Definition
Account	Business entity representing a user's account container, extends Django User model
Approval	Generic workflow record tracking review and approval status of listings and other entities
Artifact	Document or image file associated with a property listing (photos, reports, leases)
Bond	Mortgage financing for property purchase with interest rate, term, and monthly repayments
Buying Price	Purchase price the investor will pay for the property
Current Rental Income	Actual rental income based on active leases
Expected Rental Income	Projected rental income based on market rates or listing estimates
Financing	Funding method for property purchase (Bank Bond or Cash)
Investment	Financial model container for a property listing, tracks all costs and income
Lister	User who creates and manages property listings (typically Sourcing Agent)
Listing	Property advertisement on the platform with details, pricing, and rental potential
Market Price	Estimated market value of the property
Multi-Let	Rental strategy where property is divided into multiple units rented separately
Recurring Expense	Ongoing monthly or annual costs (e.g., rates, levies, maintenance, bond repayments)
Rental Approach	Strategy for renting property (Multi-Let, Student Accommodation, Single Family)
ROI (Return on Investment)	Annual return percentage calculated as Net Annual Income / Total Investment
Service Provider	Third-party professional offering property-related services (contractors, managers)
Sourcing Agent	User role responsible for finding and listing investment properties
Sourcing Fee	Platform revenue calculated as 5% of buying price (minimum R30,000)
StateUpdate	

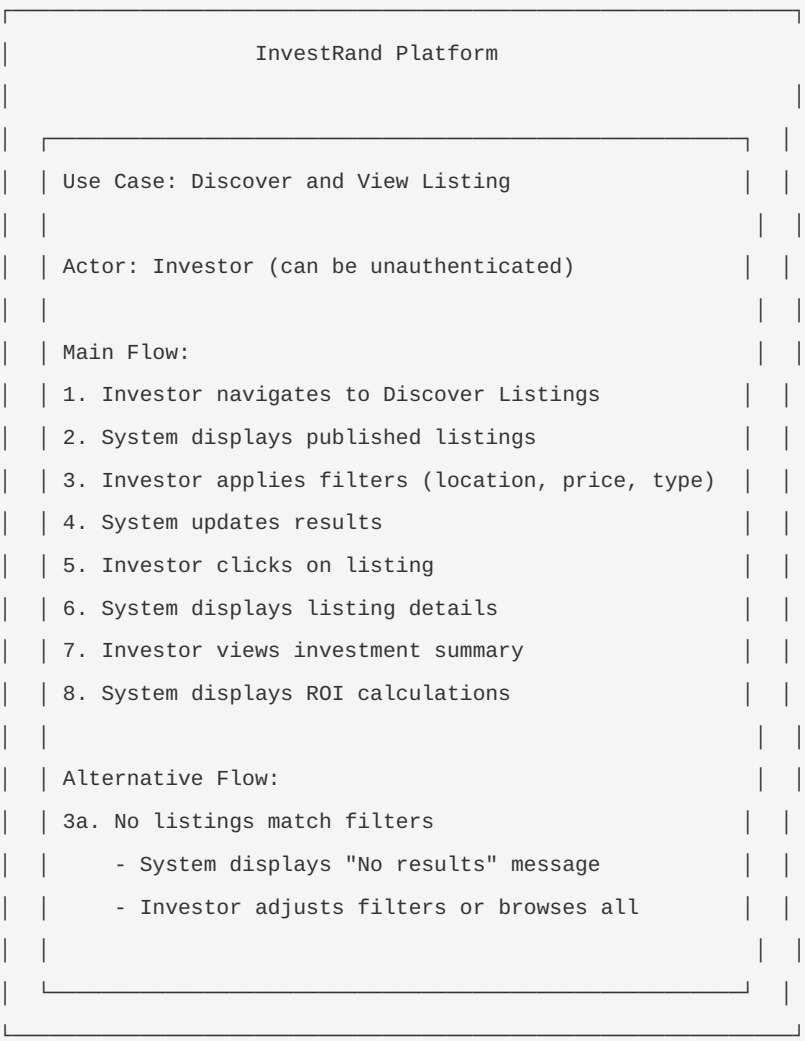
Term	Definition
	Audit record of approval status change with user, timestamp, and optional comment
Subunit	Rentable unit within a property (supports hierarchical structure for multi-let)
Transaction Cost	One-time cost for property acquisition (transfer fees, registration, deposits)
UID	Universally Unique Identifier (UUID) for entities requiring external reference

---

Appendices

Appendix A: Use Case Diagrams

Use Case: Investor Discovers and Views Listing



Use Case: Sourcing Agent Creates Listing

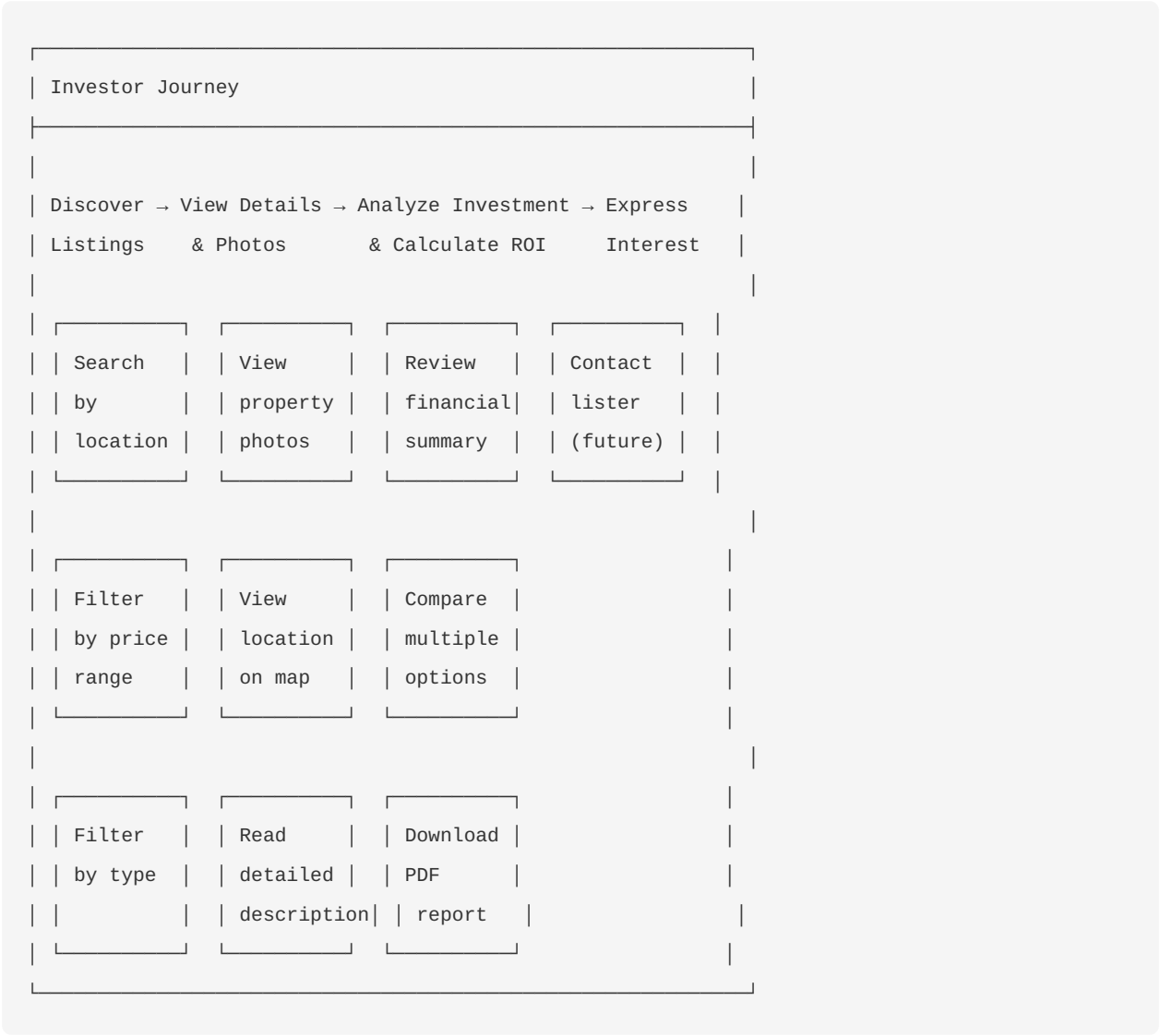
```
┌──────────────────────────────────────────────────────────────┐
│ Use Case: Create Property Listing                            │
│                                                              │
│ Actor: Sourcing Agent (authenticated)                        │
│                                                              │
│ Preconditions:                                               │
│ - User logged in with Sourcing Agent or Admin role           │
│ - User has accepted Proposal to Service agreement            │
│                                                              │
│ Main Flow:                                                   │
│ 1. Agent navigates to "Create Listing"                       │
│ 2. Agent fills property details (name, type, address)        │
│ 3. Agent sets buying price and market price                  │
│ 4. Agent uploads property photos                             │
│ 5. Agent adds subunits (for multi-let properties)            │
│ 6. Agent configures investment model (financing type)        │
│ 7. Agent previews listing                                    │
│ 8. Agent submits for approval                                │
│ 9. System creates listing with status "Created"              │
│ 10. System creates approval with state "Submitted"           │
│ 11. System sends notification to admins                      │
│                                                              │
│ Alternative Flows:                                           │
│ 2a. Required fields missing                                  │
│     - System displays validation errors                      │
│     - Agent corrects and resubmits                           │
│                                                              │
│ 4a. File upload fails                                        │
│     - System displays error message                          │
│     - Agent retries upload                                   │
│                                                              │
│ Postconditions:                                              │
│ - Listing created with unique UID                            │
│ - Approval workflow initiated                                │
│ - Agent can track listing in "My Listings"                   │
└──────────────────────────────────────────────────────────────┘
```

Use Case: Admin Approves Listing

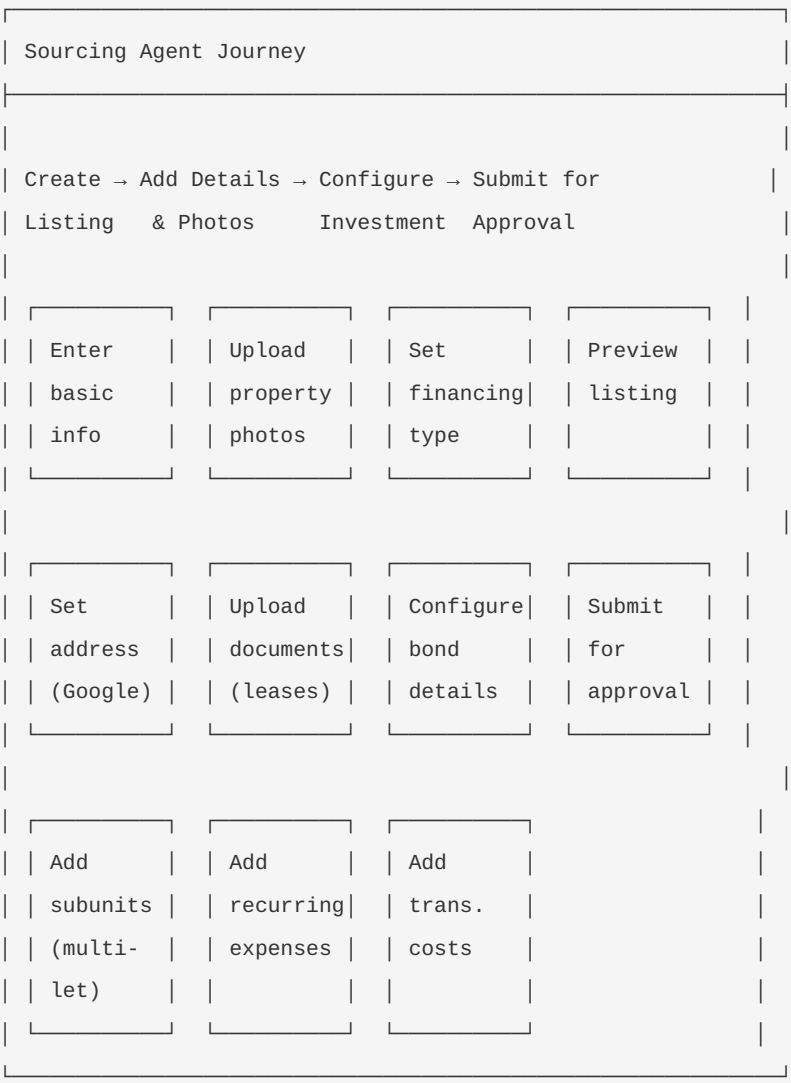
```
┌──────────────────────────────────────────────────────────┐
│ Use Case: Approve Property Listing                       │
│                                                          │
│ Actor: Administrator (authenticated)                     │
│                                                          │
│ Preconditions:                                           │
│ - User logged in with Admin role                         │
│ - Listing exists with approval state "Submitted"         │
│                                                          │
│ Main Flow:                                               │
│ 1. Admin navigates to "Manage Listings"                  │
│ 2. Admin filters by approval state "Submitted"           │
│ 3. System displays pending listings                      │
│ 4. Admin clicks on listing to review                     │
│ 5. System displays full listing details                  │
│ 6. Admin reviews completeness and accuracy               │
│ 7. Admin updates approval state to "Approved"            │
│ 8. System creates StateUpdate record                     │
│ 9. System changes listing status to "Published"          │
│ 10. System sends notification to lister                  │
│                                                          │
│ Alternative Flow: Rejection                              │
│ 7a. Admin determines listing incomplete or inaccurate    │
│     - Admin updates approval state to "Rejected"         │
│     - Admin provides rejection comment (mandatory)       │
│     - System creates StateUpdate with comment            │
│     - System sends notification to lister with feedback  │
│     - Lister can edit and resubmit                       │
│                                                          │
│ Postconditions:                                          │
│ - If approved: Listing visible in public search          │
│ - If rejected: Listing remains hidden, lister notified   │
│ - Approval state change recorded in audit trail          │
└──────────────────────────────────────────────────────────┘
```

Appendix B: User Story Map

Epic: Property Discovery and Investment



Epic: Property Listing Creation



Appendix C: Requirements Change Log

Version	Date	Author	Changes	Reason
1.0	2025-11-19	Senior Requirements Analyst Agent	Initial requirements specification document	Reverse engineering from InvestRand codebase

---

Appendix D: Key File Reference

Component	File Path	Lines	Purpose
Property Listings	/platform-back-end/property/models.py	1,345	Core listing, subunit, rental models
Investment Modeling	/platform-back-end/investment/models.py	594	Investment, financing, bond calculations
Approval Workflows	/platform-back-end/approvals/models.py	248	Generic approval engine
User Accounts	/platform-back-end/user_account/models.py	102	Account, agreement models
User Profiles	/platform-back-end/user_profile/models.py	208	Extended user info, roles
Permissions	/platform-back-end/permission_manager/models.py	185	Role-based access control
Contracts	/platform-back-end/contracts/models.py	244	Contract versioning, execution
Notifications	/platform-back-end/notification/models.py	485	Multi-channel notification system
Service Providers	/platform-back-end/service_providers/models.py	~200 (est)	Service provider directory
Frontend Routes	/platform-front-end/src/routes/routes.js	563	Vue.js routing configuration

---

Document End

This comprehensive requirements specification provides a complete, traceable, and actionable foundation for understanding, maintaining, and enhancing the InvestRand platform. All requirements are derived from analysis of the existing codebase, ensuring accuracy and alignment with current implementation.