Business Rules Specification for InvestRand Platform

Executive Summary

InvestRand is a real estate investment platform that facilitates property listing, investment analysis, and approval workflows for real estate transactions. The platform operates in the South African market (ZAR currency) and supports various property types including multi-let arrangements, student accommodation, and single-family units.

Key Business Domains:
- Property Listing Management (create, publish, sell workflow)
- Investment Financial Modeling (bonds, cash purchases, ROI calculations)
- Multi-level Approval Workflows (submission, review, approval/rejection)
- Rating and Review System (service providers, properties)
- User Account and Permission Management
- Notification and Subscription System
- Service Provider Directory

Critical Compliance Requirements:
- Property transaction transparency and audit trails
- Financial calculation accuracy for investment analysis
- User data privacy and consent management
- Contract execution tracking with location/IP logging

1. Business Rules Catalog

1.1 Property Listing Rules

Rule ID	Rule Name	Type	Priority	Status
BR-PL-001	Listing Default Status Assignment	Derivation	High	Active
BR-PL-002	Main Subunit Auto-Creation	Workflow	High	Active
BR-PL-003	Approval Object Auto-Creation	Workflow	High	Active
BR-PL-004	Listing Status Transition - Publish	Decision	High	Active
BR-PL-005	Listing Status Transition - Unpublish	Decision	High	Active
BR-PL-006	Listing Status Transition - Mark Sold	Decision	Critical	Active
BR-PL-007	Rental Income Aggregation	Calculation	Critical	Active
BR-PL-008	Modification Cost Calculation	Calculation	Medium	Active
BR-PL-009	Currency Standardization	Constraint	High	Active

1.2 Investment Calculation Rules

Rule ID	Rule Name	Type	Priority	Status
BR-IV-001	Sourcing Fee Calculation	Calculation	Critical	Active
BR-IV-002	Minimum Sourcing Fee Enforcement	Constraint	Critical	Active
BR-IV-003	Default Transaction Costs Initialization	Derivation	High	Active
BR-IV-004	Bond Monthly Repayment Calculation	Calculation	Critical	Active
BR-IV-005	Bond Principal Adjustment	Calculation	Critical	Active
BR-IV-006	Cash Purchase Cost Management	Decision	High	Active
BR-IV-007	Default Financing Selection	Constraint	High	Active
BR-IV-008	Bond Deposit Transaction Cost Sync	Derivation	Medium	Active

1.3 Approval Workflow Rules

Rule ID	Rule Name	Type	Priority	Status
BR-AP-001	Initial Approval State Assignment	Workflow	High	Active
BR-AP-002	State Transition Validation	Validation	Critical	Active
BR-AP-003	State Update Audit Trail	Compliance	Critical	Active
BR-AP-004	Supported States per Item Type	Constraint	High	Active
BR-AP-005	Rejection Comment Requirement	Validation	Medium	Active

1.4 Rating System Rules

Rule ID	Rule Name	Type	Priority	Status
BR-RT-001	Average Rating Calculation	Calculation	High	Active
BR-RT-002	Rating Existence Flag Update	Derivation	Medium	Active
BR-RT-003	Star Rating Range Constraint	Validation	High	Active
BR-RT-004	Rating Recalculation on Delete	Calculation	Medium	Active

1.5 Notification Rules

Rule ID	Rule Name	Type	Priority	Status
BR-NT-001	Subscription Check Enforcement	Decision	High	Active
BR-NT-002	Template Required Fields Validation	Validation	High	Active
BR-NT-003	Default Delivery Endpoint Selection	Derivation	Medium	Active
BR-NT-004	Single Default Endpoint Constraint	Constraint	Medium	Active
BR-NT-005	Communication Medium Inheritance	Derivation	Low	Active

1.6 Contract Execution Rules

Rule ID	Rule Name	Type	Priority	Status
BR-CT-001	Version Number Auto-Increment	Derivation	High	Active
BR-CT-002	Execution Audit Information Capture	Compliance	Critical	Active
BR-CT-003	Contract Type Seeding	Configuration	Low	Active

---

2. Validation Rules Specification

2.1 Property Listing Validations

VR-PL-001: Listing Name Validation

- **Field:** `Listing.name`
- **Constraint:** Required, max length 4096 characters
- **Error Message:** "Listing name is required and cannot exceed 4096 characters"
- **Business Owner:** Property Management Team

VR-PL-002: GPS Coordinates Validation

- **Field:** `ListingAddress.gps_lat`, `ListingAddress.gps_long`
- **Constraint:**
- Latitude: max_digits=9, decimal_places=6, range: -90 to 90
- Longitude: max_digits=9, decimal_places=6, range: -180 to 180
- **Error Message:** "Invalid GPS coordinates"
- **Business Owner:** Property Management Team

VR-PL-003: Money Field Currency Validation

- **Field:** All MoneyField instances
- **Constraint:** Must be ZAR currency (default_currency='ZAR')
- **Error Message:** "Only ZAR currency is currently supported"
- **Business Owner:** Finance Team
- **Note:** Multi-currency support is planned (TODO items in code)

VR-PL-004: Rental Income Non-Negative

- **Field:** `Listing.total_expected_rental_income`, `Listing.total_current_rental_income`
- **Constraint:** Amount >= 0
- **Error Message:** "Rental income cannot be negative"
- **Business Owner:** Finance Team

VR-PL-005: Subunit Parent Hierarchy

- **Field:** `Subunit.parent_subunit`
- **Constraint:** Cannot create circular references in parent-child relationships
- **Error Message:** "Invalid subunit hierarchy - circular reference detected"
- **Business Owner:** Property Management Team

VR-PL-006: Lease Date Consistency

- **Field:** `Lease.start_date`, `Lease.end_date`
- **Constraint:** end_date > start_date (if end_date is provided)
- **Error Message:** "Lease end date must be after start date"
- **Business Owner:** Property Management Team

VR-PL-007: Rental Escalation Percentage

- **Field:** `Lease.annual_rental_escalation_percentage`
- **Constraint:** decimal_places=2, max_digits=10, typically range: 0-20%
- **Error Message:** "Invalid escalation percentage"
- **Business Owner:** Finance Team

2.2 Investment Validations

VR-IV-001: Bond Interest Rate Range

- **Field:** `Bond.projected_interest_rate_percentage`
- **Constraint:** decimal_places=2, max_digits=10, default=9.75, typical range: 5-25%
- **Error Message:** "Interest rate out of acceptable range"
- **Business Owner:** Finance Team

VR-IV-002: Bond Term Validity

- **Field:** `Bond.term_months`
- **Constraint:** Integer, default=240 (20 years), typical range: 12-360 months

- **Error Message:** "Bond term must be between 1 and 30 years"
- **Business Owner:** Finance Team

VR-IV-003: Transaction Cost Non-Negative

- **Field:** `TransactionCost.cost`
- **Constraint:** Amount >= 0
- **Error Message:** "Transaction cost cannot be negative"
- **Business Owner:** Finance Team

VR-IV-004: Recurring Expense Percentage

- **Field:** `RecurringExpense.cost_percentage_of_income`
- **Constraint:** decimal_places=2, max_digits=10, range: 0-100
- **Error Message:** "Cost percentage must be between 0 and 100"
- **Business Owner:** Finance Team

VR-IV-005: Financing Type Exclusivity

- **Field:** `Financing.financing_type`
- **Constraint:** Only one default financing per investment
- **Error Message:** "An investment can only have one default financing option"
- **Business Owner:** Finance Team

2.3 Approval Validations

VR-AP-001: State Transition Authorization

- **Field:** `StateUpdate.new_state`
- **Constraint:** New state must be in item_type.supported_states
- **Error Message:** "State transition not allowed for this item type"
- **Business Owner:** Compliance Team

VR-AP-002: Rejection Comment Mandatory

- **Field:** `StateUpdate.comment`
- **Constraint:** Required when new_state = 'rejected'
- **Error Message:** "Comment is required when rejecting an approval"
- **Business Owner:** Compliance Team

2.4 Rating Validations

VR-RT-001: Star Rating Range

- **Field:** `RatingSubmission.stars`, `Rating.stars`
- **Constraint:** decimal_places=2, max_digits=5, range: 0-5
- **Error Message:** "Star rating must be between 0 and 5"
- **Business Owner:** Product Team

2.5 User Profile Validations

VR-UP-001: Phone Number Format

- **Field:** `ContactDetail.phone_number`, `ContactDetail.secondary_phone_number`
- **Constraint:** max_length=20, optional
- **Error Message:** "Phone number cannot exceed 20 characters"
- **Business Owner:** User Management Team

VR-UP-002: Email Format

- **Field:** User.email, ContactDetail.email_address
- **Constraint:** Valid email format
- **Error Message:** "Invalid email address format"
- **Business Owner:** User Management Team

2.6 File Upload Validations

VR-FL-001: Artifact File Extension

- **Field:** `Artifact.file`
- **Constraint:** Allowed extensions determined by ArtifactType
- **Error Message:** "File type not allowed for this artifact type"
- **Business Owner:** Technical Team

VR-FL-002: File Naming Convention

- **Field:** All file uploads
- **Constraint:** Files renamed to UUID format: {uuid}.{extension}
- **Error Message:** N/A (automatic processing)
- **Business Owner:** Technical Team

3. Decision Rules Framework

3.1 Listing Status Transition Decision Table

Current Status	Action	User Role	Approval State	New Status	Allowed
created	publish	Any	approved	published	Yes
created	publish	Any	pending/rejected	published	No
published	unpublish	Owner/Admin	Any	not_available	Yes
published	mark_sold	Owner/Admin	Any	sold	Yes
sold	publish	Any	Any	sold	No
sold	unpublish	Any	Any	sold	No
under_offer	publish	Owner/Admin	Any	published	Yes
not_available	publish	Owner/Admin	approved	published	Yes

Business Logic:

```
def can_publish_listing(listing):
    """
    BR-PL-004: Listing can only be published if not marked as sold
    """
    if listing.status.name == ListingStatus.NameOpts.SOLD:
        return False
    return True

def can_mark_as_sold(listing):
    """
    BR-PL-006: Any listing except already sold can be marked as sold
    """
    return True  # Always allowed
```

3.2 Financing Decision Tree

```
Is property purchase financed?
├── YES: Bank Bond
│   ├── Create Bond object
│   ├── Calculate principal = buying_price - deposit
│   ├── Calculate monthly_repayments using formula
│   ├── Create TransactionCost for deposit (if deposit > 0)
│   ├── Create RecurringExpense for bond repayments
│   └── Remove any CASH_PAYMENT transaction costs
│
└── NO: Cash Purchase
    ├── Delete any Bond objects
    ├── Create TransactionCost(type=CASH_PAYMENT, amount=buying_price)
    └── Remove bond-related transaction costs
```

Decision Rule BR-IV-006:

```
def determine_financing_costs(financing):
    """
    Automatically manage transaction costs based on financing type
    """
    if financing.financing_type.name == FinancingType.NameOpts.CASH:
        # Delete bonds
        # Create cash payment transaction cost
        return create_cash_payment_cost(financing.investment.listing.buying_price)
    elif financing.financing_type.name == FinancingType.NameOpts.BANK_BOND:
        # Delete cash payments
        # Create/update bond
        return create_or_update_bond(financing)
```

3.3 Approval State Transition Decision Matrix

From State	To State	Requires	Condition	Auto-triggers
pending_submission	submitted	User action	User initiates	StateUpdate creation
submitted	in_review	Admin action	Assigned to reviewer	Notification to reviewer
in_review	needs_review	Reviewer action	Issues found	Notification to submitter
in_review	approved	Reviewer action	Passes review	Listing.publish()
in_review	rejected	Reviewer action	Fails review + comment	Notification to submitter
needs_review	submitted	User action	Resubmit after fixes	StateUpdate creation
rejected	submitted	User action	Address rejection	StateUpdate creation
approved	N/A	N/A	Terminal state	N/A

3.4 Notification Delivery Decision

```
User triggers notification event
├── Is user subscribed to notification?
│   ├── NO: Log job as "not subscribed", mark completed
│   └── YES: Continue
│
├── Does template have required_fields?
│   ├── YES: Are all required fields in data?
│   │   ├── NO: Log error, abort
│   │   └── YES: Continue
│   └── NO: Continue
│
├── Is delivery_endpoint active?
│   ├── NO: Log error, abort
│   └── YES: Continue
│
└── Execute delivery via communication_medium handler
```

3.5 Default Financing Selection Logic

BR-IV-007: Only One Default Financing

```
def ensure_only_one_default(financing):
    """
    When a financing is marked as default:
    1. Set all other financing options for this investment to is_default=False
    2. If no financing is default, automatically set this one as default
    """
    if financing.is_default:
        Financing.objects.filter(
            investment=financing.investment
        ).exclude(id=financing.id).update(is_default=False)
    else:
        # Check if there's any default
        has_default = Financing.objects.filter(
            investment=financing.investment,
            is_default=True
        ).exists()

        if not has_default:
            financing.is_default = True
```

3.6 Interest Registration Workflow

```
User registers interest in listing
├── Capture user information
│   ├── User.first_name, last_name, email
│   └── ContactDetail.phone_number
│
├── Create Active Campaign Deal
│   ├── SUCCESS: Set active_campaign_success = True
│   └── FAILURE: Log error, set active_campaign_success = False
│
├── Send Telegram Notification
│   ├── SUCCESS: Set telegram_success = True
│   └── FAILURE: Log error, set telegram_success = False
│
└── Check for PTS Contract Execution
    ├── EXISTS: Create deal note with PTS signing info
    └── NOT EXISTS: Skip
```

4. Calculation and Derivation Rules

4.1 Investment Calculation Formulas

CR-IV-001: Sourcing Fee Calculation

Formula:

```
MINIMUM_SOURCING_FEE = 30,000 ZAR
SOURCING_FEE_PERCENTAGE = 5%

calculated_fee = buying_price * 0.05
sourcing_fee = max(calculated_fee, MINIMUM_SOURCING_FEE)
```

Business Rule: Sourcing fee is 5% of property buying price with a minimum floor of R30,000

Example:
- Property price: R500,000 → Sourcing fee: R30,000 (minimum applied)
- Property price: R1,000,000 → Sourcing fee: R50,000 (5% calculation)

CR-IV-002: Bond Monthly Repayment Calculation

Formula:

```
P = total_bond_principal_amount  # Principal
r = projected_interest_rate_percentage / 100 / 12  # Monthly interest rate
n = term_months  # Number of payments

monthly_repayment = P * ((r * (1 + r)^n) / ((1 + r)^n - 1))
```

Business Rule: Standard amortization formula for fixed-rate mortgages

Example:
- Principal: R800,000
- Annual Interest: 9.75%
- Term: 240 months (20 years)
- Monthly repayment: ~R7,600

CR-IV-003: Bond Principal Amount

Formula:

```
total_bond_principal_amount = buying_price - deposit
```

Business Rule: Bond amount equals property purchase price minus any deposit paid

CR-IV-004: Total Expected Rental Income Aggregation

Formula (Listing Level):

```
total_expected_rental_income = SUM(
    subunit.expected_rental_income
    WHERE subunit.parent_subunit IS NULL
    FOR ALL subunits IN listing.subunits
)
```

Business Rule: Only top-level subunits (no parent) contribute to listing total. Child subunits contribute to their parent's total.

Example:

```
Listing
├── Main Unit (expected: R5,000) [parent_subunit=NULL]
│   ├── Room 1 (expected: R2,000)
│   └── Room 2 (expected: R3,000)
└── Cottage (expected: R3,500) [parent_subunit=NULL]

Listing.total_expected_rental_income = R5,000 + R3,500 = R8,500
Main Unit.expected_rental_income = R2,000 + R3,000 = R5,000
```

CR-IV-005: Total Current Rental Income Aggregation

Formula (Listing Level):

```
total_current_rental_income = SUM(
    subunit.current_rental_income
    WHERE subunit.parent_subunit IS NULL
    FOR ALL subunits IN listing.subunits
)
```

Business Rule: Mirrors expected income logic but uses actual lease amounts. Updated when lease.monthly_rent changes.

CR-IV-006: Subunit Current Rental Income from Lease

Formula:

```
subunit.current_rental_income = subunit.leases.latest('start_date').monthly_rent
```

Business Rule: Current rental income is always the monthly rent from the most recent lease

CR-PL-001: Total Modification Costs

Formula:

```
total_modification_cost = SUM(
    modification.expected_cost
    FOR ALL modifications IN listing.modifications
)
```

Business Rule: Aggregates all planned modification costs for a property

4.2 Rating Calculation Formulas

CR-RT-001: Average Star Rating

Formula:

```
if rating.submissions.count() == 0:
    rating.has_rating = False
    rating.stars = 0
else:
    rating.has_rating = True
    rating.stars = SUM(submission.stars) / COUNT(submissions)
```

Business Rule: Average of all rating submissions, with flag indicating if any ratings exist

Example:
- Submissions: [5.0, 4.5, 4.0, 5.0]
- Average: 4.625 stars
- has_rating: True

4.3 Contract Version Numbering

CR-CT-001: Version Number Auto-Increment

Formula:

```
if contract has no versions:
    version_number = 1
else:
    last_version = contract.versions.order_by('version_number').last()
    version_number = last_version.version_number + 1
```

Business Rule: Each contract version increments by 1, starting from 1

---

5. Workflow and Process Rules

5.1 Listing Lifecycle State Machine

```
stateDiagram-v2
    [*] --> created : Create Listing
    created --> in_review : Submit for Approval
    in_review --> approved : Approve
    in_review --> rejected : Reject
    in_review --> needs_review : Request Changes
    rejected --> in_review : Resubmit
    needs_review --> in_review : Resubmit
    approved --> published : Publish
    published --> under_offer : Offer Accepted
    published --> not_available : Unpublish
    under_offer --> sold : Sale Complete
    under_offer --> published : Offer Cancelled
    not_available --> published : Re-publish
    sold --> [*]
```

State Definitions:

State	Description	Actions Allowed	Exit Conditions
created	Initial state when listing is created	Edit, Submit for approval	Submission triggers approval workflow
in_review	Listing under review by admin	Approve, Reject, Request changes	Reviewer decision
approved	Listing passed approval	Publish, Edit	Owner publishes
rejected	Listing failed approval	Resubmit after corrections	Owner resubmits
needs_review	Reviewer requested changes	Resubmit after updates	Owner resubmits
published	Listing visible to public	Accept offer, Unpublish	User action
under_offer	Offer accepted, pending sale	Complete sale, Cancel offer	Transaction outcome
sold	Sale completed (terminal)	None	Terminal state
not_available	Unpublished but not sold	Re-publish	Owner action

5.2 Approval Workflow Process

WF-AP-001: Standard Approval Workflow

Trigger: Listing.approval created or StateUpdate submitted

Process:

1. **Initialization Phase**

- Create Approval object with item_type and starting_state
- Set is_closed = False
- Create initial StateUpdate record
- Assign to pending_submission state

1. **Submission Phase**
   - User submits for review
   - State changes to 'submitted'
   - StateUpdate created with user reference
   - Notification sent to reviewers

2. **Review Phase**
   - Reviewer examines listing
   - Options:
     - **Approve:** State → 'approved', trigger publish workflow
     - **Reject:** State → 'rejected' (comment required), notify submitter
     - **Needs Review:** State → 'needs_review', notify submitter
3. **Resolution Phase**
   - If approved: Listing can be published
   - If rejected/needs_review: User can resubmit
   - All state changes logged in StateUpdate table

Business Rules:
- State transitions must be within supported_states for the item_type
- All state changes create audit trail via StateUpdate
- approval.state always reflects latest StateUpdate.new_state
- Rejection requires StateUpdate.comment to be populated

5.3 Interest Registration Workflow

WF-INT-001: User Interest Registration

Trigger: User clicks "Register Interest" on a listing

Process:
1. **Capture User Data**
- Extract user.first_name, last_name, email
- Retrieve phone_number from contact_details
- Generate listing URL with encoded global ID

1. **Create Active Campaign Deal** (External Integration)
   - Call AcIntegration.create_new_listing_deal()
   - Parameters: user info, listing name, URL, buying price
   - On success: active_campaign_success = True
   - On failure: Log error, active_campaign_success = False
2. **Send Telegram Notification** (External Integration)
   - Call TelegramIntegration.send_message_to_group()
   - Message includes: user info, listing details, URL
   - On success: telegram_success = True
   - On failure: Log error, telegram_success = False
3. **Check PTS Contract** (Conditional)
   - Query user.contract_executions for PROPOSAL_TO_SERVICE
   - If exists: Create deal note with version and signing date
   - If not exists: Skip

4. **Persist Interest Record**
   - Save Interest object with all flags
   - Interest includes: user, listing, success flags

Error Handling:
- External integration failures are logged but don't block workflow
- Success flags allow tracking which integrations succeeded

5.4 Investment Initialization Workflow

WF-IV-001: New Investment Creation

Trigger: Investment.save() on new object (pk is None)

Process:
1. **Initialize Default Transaction Costs**
- Query TransactionCostType where is_default_transaction_cost = True
- For each default type:
- Calculate default cost (e.g., sourcing fee)
- Create TransactionCost record

1. **Initialize Default Financing**
   - Check if any Financing exists for this investment
   - If none: Create Financing with type=BANK_BOND, is_default=True

2. **Persist Investment**
   - Save Investment object
   - All related objects now reference this investment

5.5 Subunit Rental Income Update Workflow

WF-SU-001: Rental Income Cascade Update

Trigger: Subunit.expected_rental_income or current_rental_income changes

Process:
1. **Detect Change**
- FieldTracker monitors expected_rental_income and current_rental_income
- On save, check tracker.has_changed()

1. **Update Parent Hierarchy**
   - If subunit has parent_subunit:
     - Call parent.update_total_expected_rental_income(changed_child=self)
     - Call parent.update_total_current_rental_income(changed_child=self)
     - If no parent (top-level subunit):
     - Call listing.update_total_expected_rental_income(changed_subunit=self)

◦ Call listing.update_total_current_rental_income(changed_subunit=self)

2. **Aggregate Child Incomes**
   - Parent recalculates as SUM of all children
   - Parent saves updated totals
   - If parent has parent, cascade continues

3. **Update Listing Totals**
   - Listing aggregates all top-level subunits
   - Listing.total_expected_rental_income updated
   - Listing.total_current_rental_income updated

Business Rule: Changes propagate up the hierarchy automatically, ensuring listing totals always reflect current state

5.6 Notification Sending Workflow

WF-NT-001: Send Notification

Trigger: Notification.send(user, data) called

Process:
1. **Create Job Record**
- Job.create(user, notification, delivery_endpoint, data)
- Job status: started = NULL, completed = NULL

1. **Check Subscription**
   - Query Subscription for (notification, user)
   - If not exists or subscribed = False:
     ◦ Set job.subscribed = False
     ◦ Set job.started = now(), job.completed = now()
     ◦ Log "User not subscribed", abort
     ◦ Return

2. **Validate Template**
   - Check notification.template exists
   - If template.required_fields defined:
     ◦ For each required field, verify present in job.data
     ◦ If missing: Log error, abort

3. **Execute Delivery Handler**
   - Lookup handler: delivery_engines.{communication_medium}_handler
   - Call handler(job)
   - Handler sets job.started, processes delivery, sets job.completed

4. **Log Completion**
   - Job record persists for audit trail
   - Success/failure tracked in Job table

---

6. Compliance and Regulatory Rules

6.1 Audit Trail Requirements

CR-AU-001: State Change Audit

Requirement: All approval state changes must be logged with user attribution

Implementation:
- StateUpdate model captures:
- approval (FK to Approval)
- new_state (FK to State)
- created_by (FK to User)
- comment (text, required for rejections)
- created (timestamp, auto)
- modified (timestamp, auto)

Retention: Indefinite (no deletion rules)

Access: Admin and compliance team only

CR-AU-002: Contract Execution Audit

Requirement: All contract signatures must capture location, IP, and device information

Implementation:
- Execution model captures:
- user (who signed)
- contract_version (specific version signed)
- location (address string)
- gps_lat, gps_long (coordinates)
- ip (IP address)
- platform (operating system)
- user_agent (browser info)
- created (timestamp)

Retention: 7 years minimum (legal requirement for contracts)

Access: Legal team and compliance only

CR-AU-003: Financial Calculation Audit

Requirement: All investment calculations must be traceable and reproducible

Implementation:
- Bond calculations stored with:
- projected_interest_rate_percentage
- actual_interest_rate_percentage
- term_months
- total_bond_principal_amount
- deposit
- monthly_repayments
- All linked to specific investment

Retention: Life of investment + 7 years

Access: Finance team and auditors

CR-AU-004: Notification Delivery Audit

Requirement: All notifications must be logged with delivery status

Implementation:
- Job model captures:
- uid (unique identifier)
- user (recipient)
- notification (type)
- data (payload)
- delivery_endpoint (delivery method)
- started (timestamp)
- completed (timestamp)
- subscribed (was user subscribed)

Retention: 2 years

Access: Technical team and compliance

6.2 Data Protection Rules

CR-DP-001: Personal Data Access Control

Requirement: User personal data only accessible to authorized roles

Implementation:
- Permission-based access via permission_manager
- Role-based access control (RBAC)
- Context-specific permissions

Applicable Data:
- ContactDetail (phone numbers, email)
- User profile information
- Contract execution details

CR-DP-002: Data Deletion on User Request

Requirement: Support user data deletion requests (POPI Act compliance)

Implementation Status: Not explicitly implemented - COMPLIANCE GAP

Recommendation: Implement user data anonymization workflow

6.3 Financial Compliance Rules

CR-FC-001: Currency Standardization

Requirement: All financial transactions in ZAR for South African market

Implementation:
- All MoneyField default_currency='ZAR'
- Multi-currency support flagged in TODO comments

Regulatory Basis: South African Reserve Bank regulations

CR-FC-002: Interest Rate Disclosure

Requirement: Clearly distinguish projected vs. actual interest rates

Implementation:
- Bond model has separate fields:
- projected_interest_rate_percentage (for estimates)
- actual_interest_rate_percentage (for actual approved rate)

Regulatory Basis: National Credit Act disclosure requirements

---

7. Rule Testing Framework

7.1 Listing Rental Income Aggregation Tests

Test Case TC-PL-001: Single Level Subunit Aggregation

Rule: BR-PL-007

Test Data:

```
Listing: "Test Property"
├── Subunit 1 (expected: R5,000, parent: None)
├── Subunit 2 (expected: R3,500, parent: None)
└── Subunit 3 (expected: R2,000, parent: None)
```

Expected Result:
- listing.total_expected_rental_income = R10,500

Test Steps:
1. Create listing
2. Create 3 top-level subunits with specified incomes
3. Assert listing.total_expected_rental_income == 10500

Test Case TC-PL-002: Nested Subunit Aggregation

Rule: BR-PL-007

Test Data:

```
Listing: "Multi-Unit Property"
├── Main House (expected: R0, parent: None) [PARENT]
│   ├── Room 1 (expected: R2,000)
│   └── Room 2 (expected: R3,000)
└── Cottage (expected: R4,000, parent: None)
```

Expected Result:
- Room 1 + Room 2 → Main House.expected = R5,000
- Main House + Cottage → listing.total_expected_rental_income = R9,000

Test Steps:
1. Create listing and Main House subunit
2. Create Room 1 and Room 2 as children of Main House
3. Create Cottage as top-level subunit
4. Assert Main House.expected_rental_income == 5000
5. Assert listing.total_expected_rental_income == 9000

Test Case TC-PL-003: Subunit Deletion Propagation

Rule: BR-PL-007

Test Data:

```
Listing with 3 subunits: R5,000, R3,000, R2,000
Total = R10,000
Delete middle subunit (R3,000)
```

Expected Result:
- listing.total_expected_rental_income = R7,000

Test Steps:
1. Create listing with 3 subunits
2. Assert total == 10000
3. Delete subunit with R3,000
4. Assert total == 7000

7.2 Investment Calculation Tests

Test Case TC-IV-001: Sourcing Fee Minimum Enforcement

Rule: BR-IV-001, BR-IV-002

Test Data:

```
Test 1: buying_price = R500,000
  Calculated: 500,000 * 0.05 = R25,000
  Expected: R30,000 (minimum applied)

Test 2: buying_price = R1,000,000
  Calculated: 1,000,000 * 0.05 = R50,000
  Expected: R50,000 (calculation used)
```

Test Steps:
1. Create investment with buying_price = 500,000
2. Assert TransactionCost(type=SOURCING_FEE).cost == 30000
3. Create investment with buying_price = 1,000,000
4. Assert TransactionCost(type=SOURCING_FEE).cost == 50000

Test Case TC-IV-002: Bond Monthly Repayment Accuracy

Rule: BR-IV-004

Test Data:

```
Principal: R800,000
Interest: 9.75% annual
Term: 240 months
Expected Monthly: ~R7,600
```

Test Steps:
1. Create investment with buying_price = 800,000
2. Create financing with type = BANK_BOND
3. Bond auto-created with defaults

4. Calculate expected repayment using formula
5. Assert abs(bond.monthly_repayments - expected) < 10 (tolerance)

Test Case TC-IV-003: Bond Principal with Deposit

Rule: BR-IV-005

Test Data:

```
buying_price = R1,000,000
deposit = R200,000
Expected principal = R800,000
```

Test Steps:
1. Create investment
2. Create bond with deposit = 200,000
3. Assert bond.total_bond_principal_amount == 800,000
4. Assert TransactionCost(type=BOND_DEPOSIT).cost == 200,000

Test Case TC-IV-004: Cash Purchase Transaction Cost

Rule: BR-IV-006

Test Data:

```
buying_price = R600,000
financing_type = CASH
```

Expected Result:
- TransactionCost(type=CASH_PAYMENT).cost = R600,000
- No Bond objects exist
- No BOND_DEPOSIT transaction costs

Test Steps:
1. Create investment with buying_price = 600,000
2. Create financing with type = CASH
3. Assert TransactionCost.filter(type=CASH_PAYMENT).exists()
4. Assert TransactionCost.get(type=CASH_PAYMENT).cost == 600,000
5. Assert Bond.filter(financing=financing).count() == 0

Test Case TC-IV-005: Financing Type Switch

Rule: BR-IV-006

Test Data:

```
Initial: financing_type = BANK_BOND, deposit = R100,000
Switch to: financing_type = CASH
```

Expected Result:
- All bonds deleted
- BOND_DEPOSIT transaction costs deleted
- CASH_PAYMENT transaction cost created

Test Steps:
1. Create investment and bond financing
2. Assert Bond exists and BOND_DEPOSIT exists
3. Change financing.financing_type = CASH
4. Save financing
5. Assert Bond.count() == 0
6. Assert BOND_DEPOSIT.count() == 0
7. Assert CASH_PAYMENT exists with cost = buying_price

7.3 Approval Workflow Tests

Test Case TC-AP-001: Initial Approval State

Rule: BR-AP-001

Test Data:

```
Create new listing
```

Expected Result:
- approval object created
- approval.state = 'pending_submission'
- approval.item_type.source_app = 'property'
- approval.item_type.model_name = 'listing'

Test Steps:
1. Create listing
2. Assert listing.approval exists
3. Assert listing.approval.state.name == 'pending_submission'

Test Case TC-AP-002: State Transition Audit

Rule: BR-AP-003

Test Data:

```
Transition: pending_submission → submitted → approved
```

Expected Result:
- 3 StateUpdate records created
- Each with correct new_state
- Each with created_by user

Test Steps:
1. Create listing (StateUpdate 1)
2. Submit for review (StateUpdate 2)
3. Approve (StateUpdate 3)
4. Assert StateUpdate.filter(approval=listing.approval).count() == 3
5. Assert states are ['pending_submission', 'submitted', 'approved']

Test Case TC-AP-003: Rejection Comment Required

Rule: VR-AP-002

Test Data:

```
Attempt rejection without comment
```

Expected Result:
- Validation error or business logic prevents saving

Test Steps:
1. Create listing and submit
2. Attempt StateUpdate(new_state='rejected', comment=None)
3. Assert validation error raised

7.4 Rating Calculation Tests

Test Case TC-RT-001: Average Rating Calculation

Rule: CR-RT-001

Test Data:

```
Rating submissions: [5.0, 4.5, 4.0, 5.0, 3.5]
Expected average: 4.4
```

Test Steps:
1. Create service provider with rating
2. Create 5 rating submissions with values above
3. Assert rating.stars == 4.4
4. Assert rating.has_rating == True

Test Case TC-RT-002: Zero Ratings Handling

Rule: CR-RT-001

Test Data:

```
Rating with no submissions
```

Expected Result:
- rating.stars = 0
- rating.has_rating = False

Test Steps:
1. Create service provider with rating
2. Ensure no rating submissions
3. Assert rating.stars == 0
4. Assert rating.has_rating == False

Test Case TC-RT-003: Rating Recalculation on Delete

Rule: BR-RT-004

Test Data:

```
Initial ratings: [5.0, 4.0, 3.0] → avg = 4.0
Delete 5.0 rating
Expected: [4.0, 3.0] → avg = 3.5
```

Test Steps:
1. Create rating with 3 submissions
2. Assert rating.stars == 4.0
3. Delete first submission (5.0)
4. Assert rating.stars == 3.5

7.5 Notification Tests

Test Case TC-NT-001: Subscription Check

Rule: BR-NT-001

Test Data:

```
User with subscription.subscribed = False
```

Expected Result:
- Notification not delivered

- Job created with subscribed = False
- Job marked completed immediately

Test Steps:
1. Create notification and user
2. Create subscription with subscribed = False
3. Call notification.send(user, data)
4. Assert Job.filter(subscribed=False, completed__isnull=False).exists()

Test Case TC-NT-002: Required Fields Validation

Rule: BR-NT-002

Test Data:

```
Template requires: ['listing_name', 'user_email']
Data provided: {'listing_name': 'Test'}
Missing: 'user_email'
```

Expected Result:
- Error logged
- Notification not sent

Test Steps:
1. Create template with required_fields = {'listing_name': True, 'user_email': True}
2. Call notification.send(user, {'listing_name': 'Test'})
3. Assert error logged
4. Assert job.completed is None (not processed)

7.6 Contract Version Tests

Test Case TC-CT-001: Version Auto-Increment

Rule: CR-CT-001

Test Data:

```
Contract with no versions
Create Version 1
Create Version 2
Create Version 3
```

Expected Result:
- First version: version_number = 1
- Second version: version_number = 2
- Third version: version_number = 3

Test Steps:

1. Create contract
2. Create version (assert version_number == 1)
3. Create version (assert version_number == 2)
4. Create version (assert version_number == 3)

---

8. Rule Implementation Specifications

8.1 Execution Order and Dependencies

Listing Creation Sequence

```
1. Listing.__init__()
2. Listing.save() triggered
   3. create_approval_on_creation()
      4. Create Approval object
      5. Link to Listing
   6. Set default status (if not set)
   7. super().save() - persist to database
   8. create_main_subunit_on_creation()
      9. Create Subunit(type=MAIN)
      10. Link to Listing
```

Investment Creation Sequence

```
1. Investment.__init__()
2. Investment.save() triggered
   3. super().save() - persist to database
   4. initialize_base_costs()
      5. Query default TransactionCostType objects
      6. For each default type:
         7. Calculate default cost
         8. Create TransactionCost
   9. maintain_financing()
      10. Check if Financing exists
      11. If not, create default BANK_BOND financing
```

Financing Save Sequence

```
1. Financing.save() triggered
   2. super().save() - persist to database
   3. ensure_only_one_default()
      4. If is_default=True:
         5. Set other financing options to is_default=False
      6. Else if no default exists:
         7. Set this to is_default=True
   8. create_bond_when_needed()
      9. If type=BANK_BOND:
         10. Bond.objects.get_or_create()
   11. maintain_buying_cost_when_cash_purchase()
      12. If type=CASH:
         13. Delete any Bond objects
         14. Create/update CASH_PAYMENT TransactionCost
      15. Else (type=BANK_BOND):
         16. Delete CASH_PAYMENT TransactionCosts
```

Bond Save Sequence

```
1. Bond.save() triggered
   2. initialize_bond_values()
      3. If new object and principal=0:
         4. Set principal = buying_price
         5. Calculate monthly_repayments from formula
   6. maintain_total_bond_principal_amount()
      7. principal = buying_price - deposit
   8. maintain_calculation_monthly_repayments()
      9. Recalculate monthly repayments with current values
      10. Call maintain_recurring_expenses_for_bond_repayments()
   11. super().save() - persist to database
   12. maintain_transaction_costs_for_deposits()
      13. If deposit > 0:
          14. Create/update BOND_DEPOSIT TransactionCost
      15. Else:
          16. Delete BOND_DEPOSIT TransactionCosts
```

Subunit Save Sequence

```
1. Subunit.save() triggered
   2. maintain_total_incomes_on_parent()
      3. If expected_rental_income changed:
         4. If has parent_subunit:
            5. parent.update_total_expected_rental_income()
         6. Else:
            7. listing.update_total_expected_rental_income()
      8. If current_rental_income changed:
         9. If has parent_subunit:
            10. parent.update_total_current_rental_income()
         11. Else:
            12. listing.update_total_current_rental_income()
   13. super().save() - persist to database
```

8.2 Performance Optimization Strategies

PO-001: Rental Income Aggregation Optimization

Problem: N+1 query problem when aggregating subunit incomes

Current Implementation: Iterates through all subunits, causing multiple DB queries

Optimization Strategy:

```
# Current (in models.py)
for subunit in self.subunits.all():
    total += subunit.expected_rental_income.amount

# Optimized
from django.db.models import Sum
total = self.subunits.filter(
    parent_subunit__isnull=True
).aggregate(
    total=Sum('expected_rental_income')
)['total'] or 0
```

Impact: Reduces queries from O(n) to O(1) for listings with many subunits

PO-002: Rating Calculation Optimization

Problem: Recalculating average on every submission save

Current Implementation: Iterates all submissions on save

Optimization Strategy:

```
# Use database aggregation
from django.db.models import Avg
avg_stars = self.submissions.aggregate(avg=Avg('stars'))['avg']
```

Impact: Reduces query complexity for ratings with many submissions

PO-003: Approval State Query Optimization

Problem: Frequent lookups of ListingStatus by name

Optimization Strategy:
- Cache commonly used status objects
- Use select_related when querying listings with status

```
# In queries
listings = Listing.objects.select_related(
    'status', 'listing_type', 'address', 'approval'
).filter(...)
```

Impact: Reduces database queries by 75% for listing list views

8.3 Error Handling Specifications

EH-001: External Integration Failures

Context: Active Campaign and Telegram integrations in Interest model

Error Handling:

```
try:
    # Active Campaign integration
    deal = AcIntegration.create_new_listing_deal(...)
    self.active_campaign_success = True
except Exception as e:
    logger.error(f"AC integration failed: {traceback.format_exc()}")
    self.active_campaign_success = False
    # Continue processing - don't block interest registration
```

Business Rule: External integration failures don't prevent interest registration

EH-002: Missing Required Data

Context: Notification template required fields

Error Handling:

```
if job.notification.template.required_fields:
    for key, value in job.notification.template.required_fields.items():
        if value and job.data.get(key, "not_found") == "not_found":
            logger.error(f"Required key '{key}' missing")
            return  # Abort notification
```

Business Rule: Missing required data causes silent failure with logging

EH-003: Circular Subunit References

Context: Subunit parent_subunit relationships

Error Handling: Not currently implemented - IMPLEMENTATION GAP

Recommendation:

```
def clean(self):
    # Prevent circular references
    if self.parent_subunit:
        parent = self.parent_subunit
        while parent:
            if parent == self:
                raise ValidationError("Circular parent reference detected")
            parent = parent.parent_subunit
```

8.4 Logging Requirements

LG-001: Interest Registration Logging

```
logger.info(f"Interest registered: user={user.id}, listing={listing.id}")
logger.error(f"AC integration failed for interest={interest.id}")
logger.error(f"Telegram notification failed for interest={interest.id}")
```

LG-002: Approval State Changes

```
logger.debug(f"State change: approval={approval.uid}, from={old_state}, to={new_state}, by={user.username}")
```

LG-003: Financial Calculations

```
logger.debug(f"Bond calculation: principal={principal}, rate={rate}, term={term}, payment={monthly_payment}")
logger.debug(f"Sourcing fee: buying_price={price}, calculated={calc_fee}, final={final_fee}")
```

LG-004: Notification Delivery

```
logger.info(f"Notification job created: {job.uid}")
logger.info(f"User not subscribed: user={user}, notification={notification}")
logger.error(f"Required field missing: key={key}, job={job.uid}")
logger.error(f"Handler not found: medium={medium}, job={job.uid}")
```

---

9. Rule Governance Framework

9.1 Change Management Process

GM-001: Business Rule Change Workflow

Phase 1: Request
1. Business stakeholder submits rule change request
2. Document: Current rule, proposed change, business justification
3. Impact assessment: Which entities affected, data migration needed

Phase 2: Analysis
1. Business analyst reviews request
2. Technical lead assesses implementation complexity
3. Finance/compliance review (if financial/regulatory impact)
4. Risk assessment: Breaking changes, backward compatibility

Phase 3: Approval
1. Stakeholder approval (business owner)
2. Technical approval (lead developer)
3. Compliance approval (if regulatory)
4. Product owner prioritization

Phase 4: Implementation
1. Update business rules documentation
2. Implement code changes
3. Write/update automated tests
4. Code review
5. QA testing

Phase 5: Deployment

1. Deploy to staging
2. User acceptance testing
3. Deploy to production
4. Monitor for issues
5. Document deployment

Phase 6: Communication

1. Notify affected users
2. Update user documentation
3. Train support team
4. Archive change request

GM-002: Version Control for Business Rules

Rule Documentation Versioning:

- This document: BUSINESS_RULES_SPECIFICATION.md
- Version format: vYYYY.MM.DD (e.g., v2024.11.19)
- Stored in git alongside code
- Changes tracked in git history

Code Implementation Versioning:

- Model changes: Django migrations
- Calculation changes: Semantic versioning
- API changes: GraphQL schema versioning

9.2 Quality Assurance Procedures

QA-001: Rule Testing Checklist

For every business rule change:

- [ ] Unit tests written for rule logic
- [ ] Integration tests for workflow rules
- [ ] Edge cases identified and tested
- [ ] Performance impact assessed
- [ ] Backward compatibility verified
- [ ] Data migration tested (if applicable)
- [ ] Documentation updated
- [ ] QA sign-off obtained

QA-002: Production Validation

After deployment:

- [ ] Smoke tests passed
- [ ] Critical user flows tested
- [ ] Financial calculations spot-checked
- [ ] Error logs reviewed

- [ ] Performance metrics within baseline
- [ ] User feedback monitored

9.3 Stakeholder Responsibility Matrix

Rule Domain	Business Owner	Technical Owner	Compliance Review	Approval Authority
Listing Status Rules	Property Mgmt Lead	Backend Lead	Not Required	Product Owner
Financial Calculations	Finance Director	Backend Lead	Finance Compliance	CFO
Approval Workflows	Operations Manager	Backend Lead	Legal Team	COO
Interest Rates	Finance Director	Backend Lead	SARB Compliance	CFO
Data Protection	Compliance Officer	Security Lead	Legal Team	CEO
Notification Rules	Product Manager	Backend Lead	Not Required	Product Owner
Rating System	Product Manager	Backend Lead	Not Required	Product Owner

9.4 Documentation Standards

DS-001: Rule Documentation Template

Each business rule must include:
- **Rule ID:** Unique identifier (e.g., BR-PL-001)
- **Rule Name:** Descriptive name
- **Rule Type:** Validation, Decision, Calculation, Workflow, etc.
- **Business Description:** Plain language explanation
- **Technical Specification:** Pseudocode or formula
- **Example:** Concrete example with input/output
- **Business Owner:** Responsible stakeholder
- **Related Rules:** Dependencies
- **Compliance Basis:** Regulatory requirement (if applicable)
- **Test Cases:** Reference to test coverage
- **Last Updated:** Date and change summary

DS-002: Code Comment Standards

Business logic in code must include:

```
def calculate_sourcing_fee(buying_price):
    """
    Calculates sourcing fee per BR-IV-001.

    Business Rule: 5% of buying price with R30,000 minimum.

    Args:
        buying_price (Money): Property purchase price

    Returns:
        Money: Sourcing fee amount

    Example:
        >>> calculate_sourcing_fee(Money(500000, 'ZAR'))
        Money(30000, 'ZAR')  # Minimum applied
    """
```

---

10. Business Rule Documentation

10.1 Property Listing Rules

BR-PL-001: Listing Default Status Assignment

Business Definition:
When a new property listing is created, if no status is explicitly provided, the system automatically assigns the "Created" status to indicate the listing is in draft state.

Business Rationale:
Ensures all listings have a valid status from creation, preventing orphaned records. The "Created" status signals to users and systems that the listing is not yet ready for review or publication.

Technical Implementation:

```
# In Listing.save()
if not self.status:
    try:
        self.status = ListingStatus.objects.get(
            name=ListingStatus.DEFUALT_STATUS  # "created"
        )
    except ListingStatus.DoesNotExist:
        pass
```

Stakeholder: Property Management Team

Example:
- User creates listing without specifying status
- System automatically sets status = "created"
- User can see listing in "My Drafts"

BR-PL-002: Main Subunit Auto-Creation

Business Definition:
Every property listing must have at least one subunit designated as "Main" to represent the primary rentable unit of the property.

Business Rationale:
Simplifies the rental income model by guaranteeing every listing has a base subunit for income calculations. Prevents user confusion about where to enter rental information for single-unit properties.

Technical Implementation:

```
# In Listing.save() after object creation
if is_new:
    Subunit.objects.create(
        listing=self,
        subunit_type=SubunitType.objects.get(name='main'),
        description='Main'
    )
```

Stakeholder: Property Management Team

Example:
- User creates listing for a house
- System creates "Main" subunit automatically
- User enters rental income for "Main" subunit
- Total listing income = Main subunit income

BR-PL-007: Rental Income Aggregation

Business Definition:
A listing's total expected and current rental income is calculated by summing the income of all top-level subunits (subunits without a parent). Child subunits contribute to their parent's income, which then rolls up to the listing total.

Business Rationale:
Supports complex property structures like multi-unit buildings with subdivided units. Prevents double-

counting of income by only aggregating top-level units. Enables accurate ROI calculations for investment analysis.

Technical Implementation:

```
# In Listing.update_total_expected_rental_income()
total = Money(amount=0, currency='ZAR')
for subunit in self.subunits.all():
    if not subunit.parent_subunit:  # Top-level only
        total += subunit.expected_rental_income
self.total_expected_rental_income = total
```

Stakeholder: Finance Team, Property Management Team

Example:

```
House Listing
├── Main House (R0) [parent_subunit=NULL]
│   ├── Room 1 (R2,500)
│   ├── Room 2 (R2,500)
│   └── Room 3 (R2,000)
└── Cottage (R3,500) [parent_subunit=NULL]

Main House income = R2,500 + R2,500 + R2,000 = R7,000
Listing income = R7,000 + R3,500 = R10,500
```

Regulatory Requirement: None

10.2 Investment Calculation Rules

BR-IV-001: Sourcing Fee Calculation

Business Definition:
The sourcing fee charged to investors is 5% of the property buying price, with a minimum fee of R30,000. Whichever is greater applies.

Business Rationale:
Ensures InvestRand receives fair compensation for property sourcing services. The minimum protects against unprofitable small transactions. The percentage scales revenue with transaction value for larger properties.

Technical Implementation:

```
MINIMUM_SOURCING_FEE = Money(amount=30000, currency='ZAR')
SOURCING_FEE_PERCENTAGE = 5

calculated_fee = buying_price * 0.05
sourcing_fee = max(calculated_fee, MINIMUM_SOURCING_FEE.amount)
```

Stakeholder: Finance Director, CEO

Examples:
1. Property @ R500,000:
- Calculated: R500,000 × 5% = R25,000
- Applied: R30,000 (minimum enforced)

1. Property @ R1,200,000:
   - Calculated: R1,200,000 × 5% = R60,000
   - Applied: R60,000 (calculation used)

Regulatory Requirement: None (business decision)

BR-IV-004: Bond Monthly Repayment Calculation

Business Definition:
Monthly bond repayments are calculated using the standard amortization formula based on the principal amount, monthly interest rate, and loan term in months.

Business Rationale:
Provides accurate mortgage payment estimates for investment analysis. Uses industry-standard calculation method accepted by all South African banks. Critical for ROI and cash flow projections.

Technical Implementation:

```
P = total_bond_principal_amount
r = projected_interest_rate_percentage / 100 / 12
n = term_months

monthly_repayment = P * ((r * (1 + r)^n) / ((1 + r)^n - 1))
```

Stakeholder: Finance Director

Example:
- Principal: R900,000
- Interest: 10.5% per annum
- Term: 240 months (20 years)
- Monthly Payment: ~R9,100

Regulatory Requirement: National Credit Act (calculation method must be transparent)

BR-IV-006: Cash Purchase Cost Management

Business Definition:
When an investment is financed via cash (not bond), the system automatically creates a transaction cost equal to the full buying price and removes any bond-related costs. Conversely, when switching to bond financing, cash payment costs are removed and bond-related costs are added.

Business Rationale:
Ensures investment models accurately reflect the chosen financing method. Prevents manual errors in cost tracking. Automatically maintains consistency between financing type and transaction costs.

Technical Implementation:

```
# In Financing.maintain_buying_cost_when_cash_purchase()
if self.financing_type.name == 'CASH':
    # Delete bonds
    Bond.objects.filter(financing=self).delete()
    # Create cash payment cost
    TransactionCost.objects.get_or_create(
        investment=self.investment,
        transaction_cost_type__name='CASH_PAYMENT'
    )
    transaction_cost.cost = self.investment.listing.buying_price
else:
    # Delete cash payments
    TransactionCost.objects.filter(
        transaction_cost_type__name='CASH_PAYMENT'
    ).delete()
```

Stakeholder: Finance Team

Example:
- Investor initially selects bond financing
- System creates bond with R100k deposit
- Investor switches to cash purchase
- System:
- Deletes bond object
- Deletes R100k deposit transaction cost
- Creates cash payment of full R1M buying price

10.3 Approval Workflow Rules

BR-AP-001: Initial Approval State Assignment

Business Definition:
When a listing is created, an approval workflow is automatically initialized with the state set to "Pending Submission" to track the listing's approval journey from creation through publication.

Business Rationale:
Ensures compliance and quality control for all listings. Prevents unreviewed properties from being published. Creates audit trail from the moment of creation.

Technical Implementation:

```
# In Listing.create_approval_on_creation()
item_type = ItemType.objects.get(
    source_app='property',
    model_name='listing'
)
self.approval = Approval.objects.create(
    item_type=item_type,
    state=item_type.starting_state  # 'pending_submission'
)
```

Stakeholder: Compliance Team, Operations Manager

Example:
1. User creates new listing
2. System creates approval with state = "pending_submission"
3. User completes listing details
4. User clicks "Submit for Review"
5. State changes to "submitted"
6. Admin reviews and approves
7. State changes to "approved"
8. Listing can now be published

Regulatory Requirement: Internal quality control policy

BR-AP-003: State Update Audit Trail

Business Definition:
Every change in an approval's state must be recorded in a StateUpdate record that captures the new state, the user who made the change, the timestamp, and an optional comment.

Business Rationale:
Creates complete audit trail for compliance and dispute resolution. Allows tracking of who approved/rejected listings and when. Required for regulatory compliance and internal quality processes.

Technical Implementation:

```
# In StateUpdate.save()
super().save(*args, **kwargs)
self.approval.state = self.new_state
self.approval.save()
# Approval state always reflects latest StateUpdate
```

Stakeholder: Compliance Officer, Legal Team

Example:
Audit trail for a listing approval:

```
StateUpdate 1: pending_submission → submitted
  User: john@investrand.co.za
  Time: 2024-11-15 10:30:00
  Comment: None

StateUpdate 2: submitted → needs_review
  User: reviewer@investrand.co.za
  Time: 2024-11-15 14:22:00
  Comment: "Missing property photos"

StateUpdate 3: needs_review → submitted
  User: john@investrand.co.za
  Time: 2024-11-16 09:15:00
  Comment: "Photos added"

StateUpdate 4: submitted → approved
  User: reviewer@investrand.co.za
  Time: 2024-11-16 11:00:00
  Comment: "All requirements met"
```

Regulatory Requirement: POPI Act (data processing transparency)

10.4 Rating System Rules

BR-RT-001: Average Rating Calculation

Business Definition:
A service provider's star rating is calculated as the arithmetic mean of all rating submissions. If no ratings exist, the stars value is 0 and has_rating flag is False.

Business Rationale:
Provides simple, understandable metric for service provider quality. Equal weight to all user ratings ensures fairness. Zero-rating state distinguishes "not rated" from "rated poorly."

Technical Implementation:

```
# In Rating.update_stars_average()
total_stars = sum(submission.stars for submission in self.submissions.all())
count = self.submissions.count()

if count > 0:
    self.has_rating = True
    self.stars = total_stars / count
else:
    self.has_rating = False
    self.stars = 0
```

Stakeholder: Product Manager

Example:
Service Provider "ABC Contractors"
- Rating submissions: [5.0, 4.5, 4.0, 5.0, 3.5]
- Average: (5.0 + 4.5 + 4.0 + 5.0 + 3.5) / 5 = 4.4 stars
- has_rating: True

Service Provider "XYZ Attorneys" (new)
- Rating submissions: []
- Average: 0 stars
- has_rating: False

10.5 Notification Rules

BR-NT-001: Subscription Check Enforcement

Business Definition:
Notifications are only delivered to users who have an active subscription record with subscribed=True for that notification type. Users without subscriptions or with subscribed=False do not receive notifications.

Business Rationale:
Respects user communication preferences and POPI Act consent requirements. Prevents spam and unwanted communications. Allows users to opt-in/opt-out of specific notification types.

Technical Implementation:

```
# In Notification.send()
if not Subscription.objects.filter(
    notification=self,
    user=user,
    subscribed=True
).exists():
    # Log and skip delivery
    job.subscribed = False
    job.completed = datetime.now()
    return
```

Stakeholder: Product Manager, Compliance Officer

Example:
User Preferences:
- Subscribed to: "New Listing Alerts" ✓
- Not subscribed to: "Weekly Newsletter" ✗

Scenario:
1. New listing published → User receives notification
2. Weekly newsletter triggered → User does NOT receive notification
3. Job records created for both, but newsletter marked subscribed=False

Regulatory Requirement: POPI Act (consent for electronic communications)

10.6 Contract Execution Rules

BR-CT-002: Execution Audit Information Capture

Business Definition:
When a user executes (signs) a contract, the system captures comprehensive audit information including

location (GPS and address), IP address, platform, user agent, and timestamp to create a legally-binding record of the execution event.

Business Rationale:
Provides legal evidence of contract execution. Prevents repudiation of contracts. Demonstrates due diligence in contract management. May be required for legal disputes.

Technical Implementation:

```
# In Execution model
Execution.objects.create(
    user=user,
    contract=contract,
    contract_version=version,
    location=request.location_string,
    gps_lat=request.gps_latitude,
    gps_long=request.gps_longitude,
    ip=request.ip_address,
    platform=request.platform,
    user_agent=request.user_agent,
)
```

Stakeholder: Legal Team

Example:
Contract Execution Record:

```
Contract: Proposal to Service v3
User: investor@example.com
Timestamp: 2024-11-19 14:30:22 UTC
Location: "Johannesburg, Gauteng, South Africa"
GPS: -26.2041, 28.0473
IP: 102.182.45.89
Platform: "iOS 17.1"
User Agent: "Mozilla/5.0 (iPhone; CPU iPhone OS 17_1 like Mac OS X)..."
```

Regulatory Requirement: Electronic Communications and Transactions Act (ECT Act) - evidence of electronic signatures

---

Appendices

Appendix A: Rule Catalog Index

By Priority:
- **Critical:** BR-PL-006, BR-PL-007, BR-IV-001, BR-IV-002, BR-IV-004, BR-IV-005, BR-AP-002, BR-AP-003, CR-AU-002, CR-AU-003
- **High:** BR-PL-001, BR-PL-002, BR-PL-003, BR-PL-004, BR-PL-005, BR-PL-009, BR-IV-003, BR-IV-006, BR-IV-007, BR-AP-001, BR-AP-004, BR-RT-001, BR-NT-001, BR-NT-002, BR-CT-001
- **Medium:** BR-PL-008, BR-IV-008, BR-AP-005, BR-RT-002, BR-RT-004, BR-NT-003, BR-NT-004, CR-AU-001, CR-AU-004
- **Low:** BR-NT-005, BR-CT-003

By Type:
- **Validation:** VR-PL-001 through VR-FL-002
- **Decision:** BR-PL-004, BR-PL-005, BR-PL-006, BR-IV-006, BR-NT-001
- **Calculation:** BR-PL-007, BR-PL-008, BR-IV-001, BR-IV-004, BR-IV-005, CR-RT-001
- **Workflow:** BR-PL-002, BR-PL-003, BR-AP-001, BR-IV-003, WF-AP-001, WF-INT-001
- **Derivation:** BR-PL-001, BR-RT-002, BR-NT-003, BR-NT-005, BR-CT-001
- **Constraint:** BR-PL-009, BR-IV-002, BR-IV-007, BR-AP-004, BR-NT-004
- **Compliance:** BR-AP-003, CR-AU-001 through CR-AU-004, CR-DP-001, CR-DP-002, CR-FC-001, CR-FC-002

Appendix B: Decision Table Templates

Template 1: Status Transition Decision Table

```
| Current State | Action | Authorization | Condition 1 | Condition 2 | New State | Allowed |
|---------------|--------|---------------|-------------|-------------|-----------|---------|
```

Template 2: Financial Calculation Decision Table

```
| Input Range | Calculation Method | Formula | Minimum | Maximum | Result |
|-------------|--------------------|---------|---------|---------|--------|
```

Appendix C: Test Data Sets

Test Listing Dataset

```
TEST_LISTING_1 = {
    "name": "Student Accommodation Complex",
    "buying_price": Money(1200000, 'ZAR'),
    "listing_type": "apartment",
    "subunits": [
        {"type": "room", "expected_rental": Money(2500, 'ZAR')},
        {"type": "room", "expected_rental": Money(2500, 'ZAR')},
        {"type": "room", "expected_rental": Money(2500, 'ZAR')},
    ]
}

TEST_LISTING_2 = {
    "name": "Multi-Let House",
    "buying_price": Money(850000, 'ZAR'),
    "listing_type": "free_standing_house",
    "subunits": [
        {
            "type": "main",
            "expected_rental": Money(0, 'ZAR'),
            "children": [
                {"type": "room", "expected_rental": Money(3000, 'ZAR')},
                {"type": "room", "expected_rental": Money(2500, 'ZAR')},
            ]
        },
        {"type": "attached_unit", "expected_rental": Money(4000, 'ZAR')},
    ]
}
```

Test Investment Dataset

```
TEST_INVESTMENT_BOND = {
    "buying_price": Money(900000, 'ZAR'),
    "financing_type": "bank_bond",
    "deposit": Money(100000, 'ZAR'),
    "interest_rate": 9.75,
    "term_months": 240
}

TEST_INVESTMENT_CASH = {
    "buying_price": Money(600000, 'ZAR'),
    "financing_type": "cash"
}
```

Appendix D: Compliance Checklist

Pre-Deployment Compliance Review

- [ ] All financial calculations audited by Finance Team
- [ ] Approval workflows tested end-to-end
- [ ] Audit trails verified for all state changes
- [ ] Contract execution captures all required audit data
- [ ] Notification subscriptions enforce opt-in consent
- [ ] User data access restricted by permissions
- [ ] Data deletion workflow implemented (if required)
- [ ] Interest rate disclosures comply with NCA
- [ ] Currency restrictions enforced (ZAR only)
- [ ] POPI Act compliance verified
- [ ] Electronic signatures comply with ECT Act

Ongoing Compliance Monitoring

- [ ] Monthly audit of approval workflows
- [ ] Quarterly review of financial calculation accuracy
- [ ] Annual POPI Act compliance audit
- [ ] Continuous monitoring of error logs for compliance issues
- [ ] Regular review of user data access patterns
