Real Estate Investment Analysis

Code Documentation

December 2025

Repository:

https://github.com/srikarchowdary03/real_time_real_estate_investment_analysis

Live Application:

https://real-time-real-estate-investment-an-umber.vercel.app/

Contents

Chapter 1

Project Overview

1.1 Project Name

Real-Time Real Estate Investment Analysis

1.2 Description

A web-based application that helps real estate investors evaluate buy-and-hold rental properties. The platform integrates real-time property data with comprehensive financial analysis tools to provide investment metrics, cash flow projections, and deal scoring similar to industry tools like DealCheck.

1.3 Purpose

To provide investors with accurate, real-time investment analysis by combining property listing data with rental estimates and automated financial calculations which eliminates the need for manual spreadsheet analysis.

1.4 Target Users

- Real estate investors evaluating rental properties
- Property analysts comparing multiple investment opportunities
- First-time investors learning to analyze deals

1.5 Key Features

- Property search across US markets with map visualization
- Real-time rental estimates from RentCast API
- Comprehensive investment analysis with 15+ financial metrics
- Multi-family property support with per-unit calculations
- 30-year buy-and-hold projections
- Property saving and portfolio management

- Customizable investor profile with default assumptions
- Investment scoring and deal rating

1.6 URLs

Live Application: https://real-time-real-estate-investment-an-umber.vercel.app/

Repository: https://github.com/srikarchowdary03/real_time_real_estate_investment_analysis

Code Documentation: The complete JSDoc API documentation is available online at: https://srikarchowdary03.github.io/real_time_real_estate_investment_analysis/
This interactive documentation includes:

- All component API references
- Function signatures and parameters
- Code examples
- Type definitions
- Module dependencies

Chapter 2

Technology Stack

2.1 Frontend

Technology	**Version**	**Purpose**
React	18.x	UI component library
Vite	7.x	Build tool and dev server
React Router	6.x	Client-side routing
Tailwind CSS	3.x	Utility-first styling
Lucide React	–	Icon library
Mapbox GL JS	–	Interactive maps

Table 2.1: Frontend Technologies

2.2 Backend Services

Service	**Purpose**
Firebase Authentication	User sign-in with Google
Firebase Firestore	NoSQL database for saved properties and profiles

Table 2.2: Backend Services

2.3 External APIs

API	**Provider**	**Purpose**
Realty-in-US	RapidAPI	Property search and listings
RentCast	RentCast.io	Rental estimates and property details

Table 2.3: External APIs

2.4 Development Tools

Tool	**Purpose**
npm	Package management
Git	Version control
Vercel	Deployment and hosting
VS Code	Recommended IDE

Table 2.4: Development Tools

Chapter 3

Architecture Overview

3.1 Application Architecture

The application follows a layered architecture pattern with clear separation between presentation, business logic, and data layers.

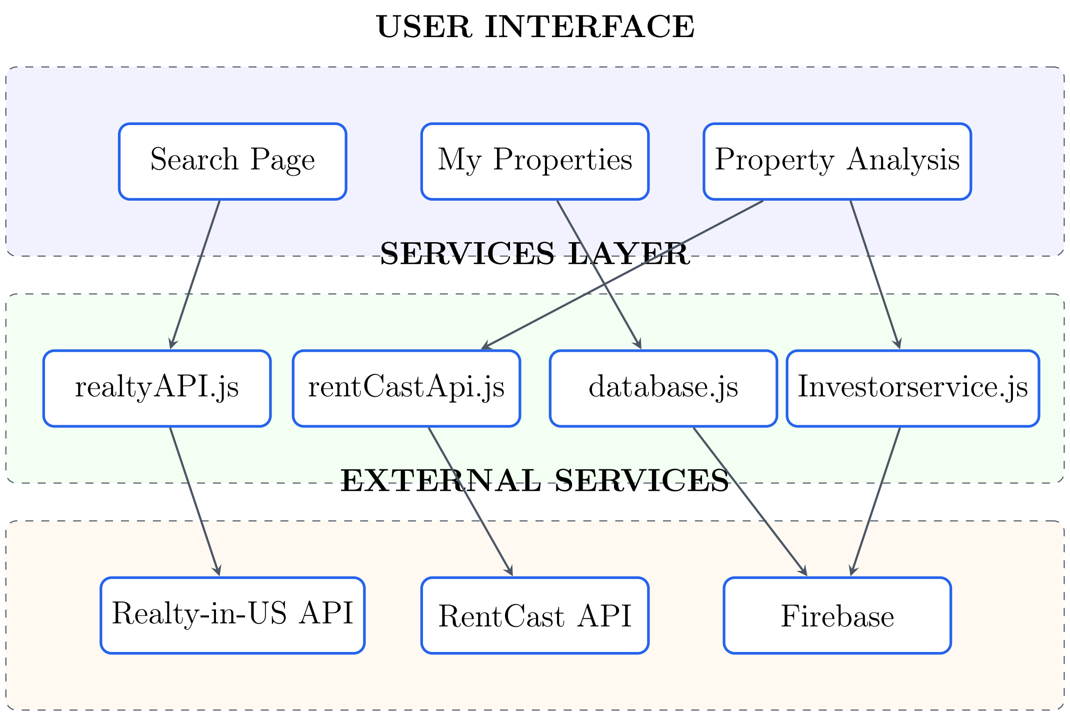

Figure 3.1: Application Architecture Diagram

3.2 Data Flow

3.2.1 Property Search Flow

1. User enters location in PropertySearchBar
2. `realtyAPI.searchProperties()` is called
3. Realty-in-US API returns property listings
4. PropertiesGrid displays results as property cards

3.2.2 Rent Data Flow

1. User Clicks on property card
2. ExpandedPropertyView opens
3. `rentCastApi.getPropertyRentData()` fetches estimate
4. RentCast API returns rent estimate and unit count
5. Property enriched with rent data
6. Quick metrics calculated and displayed

3.2.3 Property Analysis Flow

1. User clicks "Analyze Investment" on a property
2. PropertyAnalysisPage loads and auto-saves property
3. Investor profile defaults are fetched
4. Calculation inputs are initialized
5. `BuyRentHoldCalculator.getCompleteAnalysis()` computes all metrics
6. Results displayed in analysis sections

3.3 State Management

The application uses React's built-in state management:

- **useState:** Local component state for UI and form inputs
- **useEffect:** Side effects for API calls and calculations
- **useContext:** Global authentication state via AuthContext

Chapter 4

Project Structure

4.1 Directory Tree

```
src/
|
|-- components/
|   |-- analysis/                     # Property analysis components
|   |   |-- sections/                 # Analysis section components
|   |   |   |-- CashFlowSection.jsx
|   |   |   |-- FinancialRatiosSection.jsx
|   |   |   |-- FinancingSection.jsx
|   |   |   |-- InvestmentReturnsSection.jsx
|   |   |   |-- PurchaseCriteriaSection.jsx
|   |   |   |-- PurchaseRehabSection.jsx
|   |   |   +-- ValuationSection.jsx
|   |   |
|   |   |-- Buyholdprojections.jsx
|   |   |-- Inputcomponents.jsx
|   |   |-- PropertyAnalysisContent.jsx
|   |   |-- PropertyDescription.jsx
|   |   |-- PropertyPhotos.jsx
|   |   |-- PropertySidebar.jsx
|   |   +-- PurchaseWorksheet.jsx
|   |
|   |-- features/                     # Main feature components
|   |   |-- CalculatorInputs.jsx
|   |   |-- CalculatorResults.jsx
|   |   |-- ExpandedPropertyView.jsx
|   |   |-- InvestmentCalculator.jsx
|   |   |-- PropertiesGrid.jsx
|   |   |-- PropertiesHeader.jsx
|   |   |-- Propertycalculator.jsx
|   |   |-- propertycard.jsx
|   |   |-- PropertyFilters.jsx
|   |   |-- PropertyMap.jsx
|   |   |-- PropertySearchBar.jsx
|   |   +-- ViewModeDropdown.jsx
|   |
|   +-- layout/                       # Layout components
|       |-- Footer.jsx
|       |-- Header.jsx
|       +-- Layout.jsx
|
|-- config/
|   +-- firebase.js                   # Firebase initialization
|
|-- contexts/
```

```
|   +-- AuthContext.jsx              # Authentication provider
|
|-- hooks/
|   |-- useAuth.jsx
|   |-- useSavedProperties.js
|   +-- useToast.js
|
|-- pages/
|   |-- PropertyAnalysisPage.jsx
|   |-- MyProperties.jsx
|   |-- PropertyDetails.jsx
|   +-- InvestorProfile.jsx
|
|-- services/
|   |-- database.js                  # Firestore operations
|   |-- realtyAPI.js                 # Realty-in-US integration
|   |-- rentCastApi.js               # RentCast integration
|   |-- rentCastService.js
|   |-- Investorservice.js
|   +-- Propertyservice.js
|
|-- utils/
|   |-- investmentCalculations.js    # All financial calculations
|   |-- mapHelpers.js
|   +-- boundaryHelpers.js
|
|-- App.jsx                          # Root component with routes
|-- main.jsx                         # Application entry point
+-- index.css                        # Global styles + Tailwind
```

Chapter 5

Configuration & Environment Setup

5.1 Environment Variables

Create a `.env` file in the project root:

```
# Firebase Configuration
VITE_FIREBASE_API_KEY=AIzaSy...
VITE_FIREBASE_AUTH_DOMAIN=your-project.firebaseapp.com
VITE_FIREBASE_PROJECT_ID=your-project-id
VITE_FIREBASE_STORAGE_BUCKET=your-project.appspot.com
VITE_FIREBASE_MESSAGING_SENDER_ID=123456789
VITE_FIREBASE_APP_ID=1:123456789:web:abc123

# API Keys
VITE_RAPIDAPI_KEY=your-rapidapi-key
VITE_RENTCAST_API_KEY=your-rentcast-api-key
```

Important: All Vite environment variables must be prefixed with `VITE_` to be accessible in the frontend.

5.2 Firebase Configuration

firebase.js setup:

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.
     ↪ VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID
};

const app = getApps().length === 0 ? initializeApp(firebaseConfig)
   ↪ : getApp();
export const auth = getAuth(app);
export const db = getFirestore(app);
```

5.3 API Configuration

5.3.1 RapidAPI (Realty-in-US)

1. Sign up at https://rapidapi.com
2. Subscribe to "Realty-in-US" API
3. Copy API key to `VITE_RAPIDAPI_KEY`

5.3.2 RentCast

1. Sign up at https://rentcast.io
2. Generate API key from dashboard
3. Copy to `VITE_RENTCAST_API_KEY`

5.4 Local Development

```
# Install dependencies
npm install

# Start development server
npm run dev

# Build for production
npm run build

# Preview production build
npm run preview
```

Chapter 6

Database Schema

6.1 Firestore Collections

6.1.1 Collection: savedProperties

Document ID Pattern: `{userId}_{propertyId}`

```json
{
  "userId": "firebase-auth-uid",
  "propertyId": "realty-api-property-id",

  "propertyData": {
    "address": "123 Main St",
    "city": "Boston",
    "state": "MA",
    "zipCode": "02101",
    "price": 500000,
    "beds": 3,
    "baths": 2,
    "sqft": 1500
  },

  "thumbnail": "https://...",
  "photos": [
    { "href": "https://..." }
  ],

  "rentEstimate": 2500,
  "rentRangeLow": 2200,
  "rentRangeHigh": 2800,
  "rentCastData": { },

  "annualTaxAmount": 6000,

  "unitCount": 1,
  "isMultiFamily": false,

  "quickScore": "good",
  "estimatedCashFlow": 450,
  "estimatedCapRate": 6.5,
  "investmentBadge": "Good Deal",

  "notes": "",
  "tags": [],

  "createdAt": "Timestamp",
  "updatedAt": "Timestamp"
}
```

Additional fields added by update functions:

- `rentSource` – Source of rent data (e.g., "RentCast")
- `rentDataUpdatedAt` – When rent data was last updated
- `analysis` – Full analysis object from calculator

- `analysisUpdatedAt` – When analysis was last updated

6.1.2 Collection: investorProfiles

Document ID: `{userId}`

```json
{
  "userId": "firebase-auth-uid",

  "financing": {
    "downPaymentPercent": 20,
    "interestRate": 7.0,
    "loanTermYears": 30,
    "closingCostsPercent": 3.0
  },

  "expenses": {
    "vacancyRate": 5,
    "managementRate": 8,
    "maintenanceRate": 5,
    "capExRate": 5,
    "propertyTaxRate": 1.2,
    "insuranceRate": 0.5
  },

  "scoringConfig": {
    "minCapRate": 5,
    "minCashFlow": 200,
    "minCashOnCash": 8,
    "minDCR": 1.2
  },

  "createdAt": "Timestamp",
  "updatedAt": "Timestamp"
}
```

6.2 Firestore Security Rules

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Saved properties - flat structure with userId_propertyId
    match /savedProperties/{docId} {
      allow read: if request.auth != null && resource.data.userId
          ↪ == request.auth.uid;
      allow create: if request.auth != null && request.resource.
          ↪ data.userId == request.auth.uid;
      allow update, delete: if request.auth != null && resource.
          ↪ data.userId == request.auth.uid;
    }

    // Investor profiles
    match /investorProfiles/{userId} {
      allow read, write: if request.auth != null && request.auth.
          ↪ uid == userId;
    }
  }
}
```

Chapter 7

API Integration

7.1 Realty-in-US API (RapidAPI)

7.1.1 Configuration

Base URL: `https://realty-in-us.p.rapidapi.com`

Headers:

```
{
  'X-RapidAPI-Key': import.meta.env.VITE_RAPIDAPI_KEY,
  'X-RapidAPI-Host': 'realty-in-us.p.rapidapi.com',
  'Content-Type': 'application/json'
}
```

7.1.2 Main Endpoints

Endpoint	Method	Purpose
`/properties/v3/list`	POST	Search properties by location
`/properties/v3/detail`	GET	Get single property details
`/properties/v3/get-photos`	GET	Get high-quality photos

Table 7.1: Realty-in-US API Endpoints

7.1.3 Search Properties Example

File: `src/services/realtyAPI.js`

```
// POST /properties/v3/list
const searchProperties = async (city, state) => {
  const response = await fetch(`${BASE_URL}/properties/v3/list`, {
    method: 'POST',
    headers: HEADERS,
    body: JSON.stringify({
      limit: 200,
      offset: 0,
      status: ["for_sale"],
      city: city,
      state_code: state
    })
  });

  return response.json();
};
```

7.1.4 Rate Limiting

Built-in rate limiting prevents hitting API limits:

```
const MIN_REQUEST_INTERVAL = 250; // 250ms between requests

const waitForRateLimit = async () => {
  const timeSinceLastRequest = Date.now() - lastRequestTime;
  if (timeSinceLastRequest < MIN_REQUEST_INTERVAL) {
    await new Promise(resolve =>
      setTimeout(resolve, MIN_REQUEST_INTERVAL -
         ↪ timeSinceLastRequest)
    );
  }
  lastRequestTime = Date.now();
};
```

7.1.5 Image Quality Upgrade

The API returns low-resolution images. Upgrade them for better display:

```
const upgradeImageUrl = (url) => {
  if (!url) return null;

  // Change from -w400_h300_q80 to -w1024_h768_q90
  return url
    .replace(/-w\d+_h\d+/, '-w1024_h768')
    .replace(/_q\d+/, '_q90');
};
```

7.2 RentCast API

7.2.1 Configuration

Base URL: `https://api.rentcast.io/v1`

Headers:

```
{
  'Accept': 'application/json',
  'X-Api-Key': import.meta.env.VITE_RENTCAST_API_KEY
}
```

7.2.2 Endpoints

Endpoint	Method	Purpose
`/avm/rent/long-term`	GET	Get rent estimate (per unit)
`/properties`	GET	Get property details with unit count
`/markets`	GET	Get market statistics

Table 7.2: RentCast API Endpoints

7.2.3 Single Unit Detection (v2.0)

Problem: Single condos (e.g., "Unit 40M") were incorrectly treated as entire buildings, causing massive rent calculation errors.

Solution: Address pattern detection identifies unit numbers to distinguish single units from entire buildings.

```
// Detects: "Unit 40M", "Apt 5B", "#204", etc.
const isSingleUnitInBuilding = (address) => {
  const patterns = [
    /\bunit\s+[a-z0-9]+/i,
    /\bapt\s+[a-z0-9]+/i,
    /\#\s*[a-z0-9]+/
  ];
  return patterns.some(pattern => pattern.test(address));
};
```

7.2.4 Main Function

File: `src/services/rentCastApi.js`

```javascript
export const getCompletePropertyData = async (property) => {
  // Fetch rent estimate and property details
  const [rentData, propertyDetails] = await Promise.all([
    getRentEstimate({ address, city, state, zipCode }),
    getPropertyDetails({ address, city, state, zipCode })
  ]);

  // Detect unit type
  const isSingleUnit = isSingleUnitInBuilding(address);
  const totalUnitsInBuilding = propertyDetails?.unitCount || 1;

  // Calculate units being purchased
  let actualUnitCount = isSingleUnit ? 1 : totalUnitsInBuilding;

  // Calculate total rent
  const perUnitRent = rentData?.rentEstimate || null;
  const totalMonthlyRent = perUnitRent * actualUnitCount;

  return {
    rentEstimate: perUnitRent,
    totalMonthlyRent: totalMonthlyRent,
    unitCount: actualUnitCount,
    isMultiFamily: actualUnitCount > 1 && !isSingleUnit,
    isSingleUnit: isSingleUnit,
    totalUnitsInBuilding: totalUnitsInBuilding
  };
};
```

7.2.5 Response Examples

Single Condo:

```json
{
  "rentEstimate": 2000,
  "totalMonthlyRent": 2000,
  "unitCount": 1,
  "isMultiFamily": false,
  "isSingleUnit": true,
  "totalUnitsInBuilding": 120
}
```

Entire Duplex:

```json
{
  "rentEstimate": 2000,
  "totalMonthlyRent": 4000,
  "unitCount": 2,
  "isMultiFamily": true,
  "isSingleUnit": false,
  "totalUnitsInBuilding": 2
}
```

7.3 API Rate Limits

API	**Free Tier**	**Notes**
Realty-in-US	500/month	Upgrade for more
RentCast	50/month	Cache results

Table 7.3: API Rate Limits

Chapter 8

Core Modules & Business Logic

This chapter documents the `investmentCalculations.js` module, which is the heart of the application. All financial calculations are centralized here.

Location: `src/utils/investmentCalculations.js`

8.1 Default Values (DEFAULTS Object)

```
export const DEFAULTS = {
  // Financing
  ltv: 80,
  firstMtgLTV: 80,
  interestRate: 7.0,
  firstMtgRate: 7.0,
  amortization: 30,
  firstMtgAmortization: 30,
  downPaymentPercent: 20,

  // Closing costs
  purchaseCostsPercent: 3.0,
  closingCostsPercent: 3.0,

  // Expenses (as % of rent)
  vacancyRate: 5.0,
  managementRate: 8.0,
  repairsPercent: 5.0,
  maintenanceRate: 5.0,
  capExRate: 5.0,

  // Property costs (as % of price)
  propertyTaxRate: 1.2,
  insuranceRate: 0.5,

  // Projections
  appreciationRate: 3.0,
  incomeGrowthRate: 2.0,
  expenseGrowthRate: 2.0,
  sellingCosts: 6.0,
  holdingPeriod: 5
};
```

8.2 BuyRentHoldCalculator Class

The main calculator class that performs all investment analysis.

8.2.1 Constructor

```
export class BuyRentHoldCalculator {
  constructor(property, inputs) {
    this.property = property || {};
    this.inputs = inputs || {};
  }
}
```

8.2.2 Calculation Sections

The calculator is organized into sections that mirror the Excel spreadsheet:

1. `getPropertyInfo()` – Property Info
2. `calculatePurchaseInfo()` – Purchase Info → Real Purchase Price
3. `calculateFinancing()` – Financing → Cash Required to Close
4. `calculateIncome()` – Income Annual → Effective Gross Income
5. `calculateOperatingExpenses()` – Operating Expenses Annual
6. `calculateNOI()` – Net Operating Income
7. `calculateCashRequirements()` – Cash Requirements
8. `calculateCashflowSummary()` – Cashflow Summary Annual
9. `calculateQuickAnalysis()` – Quick Analysis (All Ratios)
10. `calculateInvestmentScore()` – Investment Score

8.3 Financial Formulas

8.3.1 Monthly Mortgage Payment (PMT Formula)

The standard amortization formula for calculating monthly principal and interest payment:

$$\boxed{PMT = P \times \frac{r(1+r)^n}{(1+r)^n - 1}} \tag{8.1}$$

Where:

- PMT = Monthly payment
- P = Principal (loan amount)
- r = Monthly interest rate $\left(\frac{\text{Annual Rate}}{12\times100}\right)$
- n = Total number of payments (years × 12)

Implementation:

```javascript
function calculateMonthlyPayment(principal, annualRate, years) {
  if (!principal || principal === 0) return 0;
  if (!annualRate || annualRate === 0) return principal / (years *
     ↪ 12);
  if (!years || years === 0) return 0;

  const monthlyRate = annualRate / 100 / 12;
  const numPayments = years * 12;

  return (principal * monthlyRate * Math.pow(1 + monthlyRate,
     ↪ numPayments)) /
         (Math.pow(1 + monthlyRate, numPayments) - 1);
}
```

8.3.2 Real Purchase Price (RPP)

Total acquisition cost including purchase price, repairs, and closing costs:

$$\boxed{RPP = \text{Offer Price} + \text{Repairs} + \text{Contingency} + \text{Closing Costs}} \tag{8.2}$$

Where Closing Costs can be calculated as:

$$\text{Closing Costs} = \text{Offer Price} \times \frac{\text{Closing Cost \%}}{100} \tag{8.3}$$

Or as itemized:

$$\text{Closing Costs} = \sum(\text{Lender Fee, Broker Fee, Inspections, Appraisals, Legal, etc.}) \tag{8.4}$$

8.3.3 Loan Amount

$$\boxed{\text{Loan Amount} = \text{Offer Price} \times \frac{LTV}{100}} \tag{8.5}$$

Where LTV = Loan-to-Value ratio (typically 80% for 20% down payment)

8.3.4 Cash Required to Close

$$\boxed{\text{Cash Required} = RPP - \text{First Mortgage} - \text{Second Mortgage}} \tag{8.6}$$

8.3.5 Effective Gross Income (EGI)

$$\boxed{EGI = \text{Total Income} - \text{Vacancy Loss}} \tag{8.7}$$

Where:

$$\text{Total Income} = \text{Gross Rents} + \text{Parking} + \text{Storage} + \text{Laundry} + \text{Other} \tag{8.8}$$

$$\text{Vacancy Loss} = \text{Total Income} \times \frac{\text{Vacancy Rate \%}}{100} \tag{8.9}$$

8.3.6 Operating Expenses

Operating expenses are calculated as a combination of fixed costs and percentage-based costs:

$$\boxed{\text{Total Operating Expenses} = \sum \text{All Operating Costs}} \tag{8.10}$$

Key percentage-based expenses (calculated from Gross Rents):

$$\text{Repairs} = \text{Gross Rents} \times \frac{\text{Repairs \%}}{100} \tag{8.11}$$

$$\text{Management} = \text{Gross Rents} \times \frac{\text{Management \%}}{100} \tag{8.12}$$

8.3.7 Net Operating Income (NOI)

The most important metric for property valuation:

$$\boxed{NOI = EGI - \text{Total Operating Expenses}} \tag{8.13}$$

8.3.8 Annual Debt Service

$$\boxed{\text{Annual Debt Service} = (\text{1st Mtg PMT} + \text{2nd Mtg PMT} + \text{IO PMT} + \text{Other}) \times 12} \tag{8.14}$$

8.3.9 Cash Flow

$$\boxed{\text{Annual Cash Flow} = NOI - \text{Annual Debt Service}} \tag{8.15}$$

$$\text{Monthly Cash Flow} = \frac{\text{Annual Cash Flow}}{12} \tag{8.16}$$

$$\text{Cash Flow Per Unit} = \frac{\text{Monthly Cash Flow}}{\text{Number of Units}} \tag{8.17}$$

8.4 Investment Ratios

8.4.1 Capitalization Rate (Cap Rate)

Measures the return on the property independent of financing:

$$\boxed{\text{Cap Rate} = \frac{NOI}{\text{Purchase Price}} \times 100\%} \tag{8.18}$$

Interpretation:

- $\geq 10\%$: Excellent
- $8 - 10\%$: Good
- $6 - 8\%$: Fair
- $4 - 6\%$: Below Average
- $< 4\%$: Poor

8.4.2 Cash-on-Cash Return (CoC ROI)

Measures the return on actual cash invested:

$$\boxed{\text{CoC ROI} = \frac{\text{Annual Cash Flow}}{\text{Total Cash Required}} \times 100\%} \tag{8.19}$$

Interpretation:

- $\geq 12\%$: Excellent
- $8 - 12\%$: Good
- $5 - 8\%$: Fair
- $0 - 5\%$: Below Average
- $< 0\%$: Negative Return

8.4.3 Debt Coverage Ratio (DCR)

Measures the property's ability to cover debt payments:

$$\boxed{DCR = \frac{NOI}{\text{Annual Debt Service}}} \tag{8.20}$$

Interpretation:

- ≥ 1.5: Excellent (50% buffer)
- $1.25 - 1.5$: Good (lender typically requires 1.25)

- $1.1 - 1.25$: Fair
- $1.0 - 1.1$: Marginal (just covering payments)
- < 1.0: Negative cash flow

8.4.4 Gross Rent Multiplier (GRM)

Quick valuation metric:

$$\boxed{GRM = \frac{\text{Purchase Price}}{\text{Annual Gross Rent}}} \tag{8.21}$$

Interpretation:

- Lower GRM = Better value
- Typical range: 8-12 for investment properties
- < 8: Excellent value
- > 15: Potentially overpriced

8.4.5 Expense Ratio

$$\boxed{\text{Expense Ratio} = \frac{\text{Total Operating Expenses}}{\text{Total Income}} \times 100\%} \tag{8.22}$$

8.4.6 Equity ROI (Principal Paydown)

Return from mortgage principal reduction in Year 1:

$$\boxed{\text{Equity ROI} = \frac{\text{Principal Paid Year 1}}{\text{Total Cash Required}} \times 100\%} \tag{8.23}$$

8.4.7 Appreciation ROI

Return from property value appreciation:

$$\boxed{\text{Appreciation ROI} = \frac{\text{FMV} \times \text{Appreciation Rate \%}}{\text{Total Cash Required}} \times 100\%} \tag{8.24}$$

8.4.8 Total ROI

Combined return from all sources:

$$\boxed{\text{Total ROI} = \text{CoC ROI} + \text{Equity ROI} + \text{Appreciation ROI}} \tag{8.25}$$

8.5 Investment Scoring Algorithm

The scoring system rates properties on a 0-100 scale based on weighted metrics:

Metric	**Weight**	**Max Points**
Cash-on-Cash ROI	25%	25
Cap Rate	20%	20
Debt Coverage Ratio	20%	20
Monthly Cash Flow	20%	20
Total ROI	15%	15
Total	**100%**	**100**

Table 8.1: Investment Scoring Weights

8.5.1 Scoring Thresholds

Metric	**Excellent**	**Good**	**Fair**	**Risky**	**Points**
CoC ROI	$\geq$ 12%	$\geq$ 8%	$\geq$ 5%	$\geq$ 0%	25/20/15/10
Cap Rate	$\geq$ 10%	$\geq$ 8%	$\geq$ 6%	$\geq$ 4%	20/17/14/10
DCR	$\geq$ 1.5	$\geq$ 1.25	$\geq$ 1.1	$\geq$ 1.0	20/16/12/8
Cash Flow	$\geq$ \$500	$\geq$ \$300	$\geq$ \$100	$\geq$ \$0	20/16/12/8
Total ROI	$\geq$ 20%	$\geq$ 15%	$\geq$ 10%	$\geq$ 5%	15/12/9/6

Table 8.2: Scoring Thresholds by Metric

8.5.2 Score Badge Mapping

Score Range	**Badge**	**Description**
85-100	Excellent	Outstanding investment - Strong across all metrics
70-84	Good	Good investment - Solid returns expected
50-69	Fair	Fair investment - Average returns
30-49	Risky	Risky investment - Below average metrics
0-29	Avoid	Poor investment - Negative cash flow likely

Table 8.3: Score Badge Mapping

8.6 Multi-Family Detection

The system detects multi-family properties using multiple data sources:

```
export function detectMultiFamily(property, inputs) {
  // Priority 1: RentCast unit count from public records
  if (property?.rentCastData?.features?.unitCount > 1) {
    return {
      isMultiFamily: true,
      units: property.rentCastData.features.unitCount,
      source: 'RentCast'
    };
  }

  // Priority 2: User-specified units
  if (inputs?.numberOfUnits > 1) {
    return {
      isMultiFamily: true,
      units: inputs.numberOfUnits,
      source: 'Input'
    };
  }

  // Priority 3: Property type analysis
  const type = (property?.propertyType || '').toLowerCase();
  if (type.includes('duplex'))
    return { isMultiFamily: true, units: 2, source: 'Type' };
  if (type.includes('triplex'))
    return { isMultiFamily: true, units: 3, source: 'Type' };
  if (type.includes('fourplex'))
    return { isMultiFamily: true, units: 4, source: 'Type' };

  // Default: single family
  return { isMultiFamily: false, units: 1, source: 'Default' };
}
```

8.7 Rent Estimation Fallback

When API data is unavailable, rent is estimated based on price:

```
export function estimateRent(property) {
  const price = property?.price || property?.list_price || 0;
  if (!price) return 0;

  // Price-based multiplier (lower price = higher rent ratio)
  let mult;
  if (price < 150000) mult = 0.009;       // 0.9%
  else if (price < 300000) mult = 0.008; // 0.8%
  else if (price < 500000) mult = 0.007; // 0.7%
  else mult = 0.005;                          // 0.5%

  // Round to nearest $50
  return Math.round((price * mult) / 50) * 50;
}
```

Chapter 9

Key Components

9.1 PropertyAnalysisPage.jsx

Location: `src/pages/PropertyAnalysisPage.jsx`

Purpose: Main analysis page that manages all state and coordinates child components.

9.1.1 Key State Variables

State	Type	Description
property	Object	Full property data
inputs	Object	All calculation inputs
results	Object	Calculation results
activeSection	String	Current sidebar selection
isSaved	Boolean	Save status
investorProfile	Object	User's default settings
manualUnitCount	Number/null	**v2.0:** User override for units

Table 9.1: PropertyAnalysisPage State Variables

9.1.2 Unit Detection Priority (v2.0)

Priority	Source	Description
1	Manual Override	User-specified in worksheet
2	RentCast isSingleUnit	Single unit from address pattern
3	RentCast unitCount	Unit count from public records
4	Property Type	Detection from type string
5	Default	Fallback to 1 unit

Table 9.2: Unit Detection Priority

9.1.3 Multi-Family Detection

```
// Check priority: manual > isSingleUnit > unitCount > type
const multiFamily = useMemo(() => {
  if (property.isSingleUnit) {
    return { isMultiFamily: false, units: 1, isSingleUnit: true };
  }
  if (property.unitCount) {
    return { isMultiFamily: property.unitCount > 1, units: property
      ↪ .unitCount };
  }
  return detectMultiFamily(property); // Fallback
}, [property]);

// Allow manual override
const units = manualUnitCount ?? multiFamily.units;
```

9.1.4 Key Behaviors

1. Auto-saves property when page opens
2. Loads investor profile defaults
3. **v2.0:** Correctly handles single units (no rent multiplication)
4. **v2.0:** Multiplies rent only for entire buildings
5. Allows manual unit count override
6. Updates calculations when inputs change

9.1.5 Input Initialization (v2.0)

```
useEffect(() => {
  // Calculate per-unit rent
  let rentPerUnit;
  if (property.totalMonthlyRent && units > 0) {
    rentPerUnit = property.totalMonthlyRent / units;
  } else if (property.rentEstimate) {
    rentPerUnit = property.rentEstimate;
  } else {
    rentPerUnit = estimateRent(price / units, beds / units, sqft /
      ↪ units);
  }

  // Calculate total annual rent
  const totalAnnualRent = rentPerUnit * units * 12;

  // Set all inputs with calculated rent
  setInputs({
    numberOfUnits: units,
    grossRents: totalAnnualRent,
    // ... other inputs
  });
}, [property, investorProfile, manualUnitCount]); // Key dependency
```

9.2 ExpandedPropertyView.jsx

Location: `src/components/features/ExpandedPropertyView.jsx`

Purpose: Floating overlay that displays property details and fetches RentCast data on hover.

9.2.1 Key Data (v2.0)

```
// Extracted from RentCast API response
const unitCount = rentData?.unitCount || 1;
const isMultiFamily = rentData?.isMultiFamily || false;
const isSingleUnit = rentData?.isSingleUnit || false;
const totalUnitsInBuilding = rentData?.totalUnitsInBuilding ||
    ↪ unitCount;

const perUnitRent = rentData?.rentEstimate || 0;
const totalMonthlyRent = rentData?.totalMonthlyRent || 0;
```

9.2.2 UI Display (v2.0)

Single Unit:

```
{isSingleUnit && (
  <div className="bg-blue-50 rounded-lg p-4">
    <p className="text-2xl font-bold">{formatPrice(perUnitRent)}/mo
        ↪ </p>
    <p className="text-xs mt-2">
      Single Unit: 1 of {totalUnitsInBuilding} units
    </p>
  </div>
)}
```

Multi-Family:

```
{isMultiFamily && (
  <div className="bg-green-50 rounded-lg p-4">
    <p className="text-sm">Per Unit: {formatPrice(perUnitRent)}/mo
        ↪ </p>
    <p className="text-xl font-bold">
      Total: {formatPrice(totalMonthlyRent)}/mo
    </p>
  </div>
)}
```

Badge:

```
{isSingleUnit && <Badge>1/{totalUnitsInBuilding}</Badge>}
{isMultiFamily && <Badge>{unitCount} Units</Badge>}
```

9.2.3 Key Behaviors

1. Fetches RentCast data on mount
2. **v2.0:** Shows different UI for single units vs buildings
3. **v2.0:** Displays per-unit and total rent separately
4. **v2.0:** Badge shows unit context (e.g., 1/120 or 4 Units)
5. Navigates to analysis page with complete data

9.3 Other Key Components

9.3.1 PropertyAnalysisContent.jsx

Location: `src/components/analysis/PropertyAnalysisContent.jsx`
Purpose: Displays all analysis sections and manages calculation execution.
Key Feature: Triggers `BuyRentHoldCalculator` on input changes using `useEffect`.

9.3.2 PurchaseWorksheet.jsx

Location: `src/components/analysis/PurchaseWorksheet.jsx`
Purpose: Editable form for all calculation inputs.
Sections: Purchase Info, Financing, Income, Expenses, Projections.

9.3.3 BuyHoldProjections.jsx

Location: `src/components/analysis/Buyholdprojections.jsx`
Purpose: Displays 5-year projections for property value, equity, and cash flow.

9.4 Testing Checklist

Property Type	unitCount	Expected Rent
Single family home	1	\$2,000 (not multiplied)
Condo "Unit 40M"	1	\$2,000 (not multiplied)
Entire duplex	2	\$4,000 (\$2,000 $\times$ 2)
Entire 4-unit	4	\$8,000 (\$2,000 $\times$ 4)

Table 9.3: Unit Detection Test Scenarios