Product Requirements Document

Advisor Intelligence Assistant - Phase 1

Version 1.0 | Integration-First Architecture

Executive Summary

What We're Building

An intelligence layer that sits between advisors and their fragmented technology stack, synthesizing data from multiple systems (CRM, custodian, portfolio management, document storage, email) into contextual, actionable insights delivered at the moment of need.

What We're NOT Building

- A CRM replacement
- A portfolio management system
- A data warehouse (we'll use theirs if available)
- A standalone application (we integrate into existing workflows)

Core Technical Challenge

Financial advisors use 5-12 different systems that don't talk to each other. Each system is the "source of truth" for different data types. Our job is to create a unified knowledge graph that can answer questions like "What should I discuss with this client tomorrow?" without forcing advisors to manually check each system.

Success Criteria - Phase 1 (90 Days)

- Reduce meeting prep time from 45 minutes to <15 minutes
- 80%+ advisor adoption rate
- 90%+ data accuracy on automated extraction
- Zero security incidents
- Deploy one workflow end-to-end for one client

User Stories & Jobs-to-Be-Done

Primary User: Wealth Advisor

Job Story 1: Meeting Preparation

When I have a client meeting tomorrow, I want to quickly understand what's changed since our last conversation and what I should prioritize discussing, so I don't waste time digging through systems or miss important opportunities.

Acceptance Criteria:

- Single dashboard view with meeting context
- Sub-5 second load time for client brief
- Highlights only actionable items (no noise)
- Links back to source systems for deep-dive

Job Story 2: Client Question Response

When a client calls with a question about their account, I want to instantly see their complete current state across all systems, so I can answer confidently without putting them on hold while I search.

Acceptance Criteria:

- <10 second retrieval of complete client context
- Real-time data from custodian (not cached)
- Clear indication of data freshness
- Ability to drill into any system without leaving interface

Job Story 3: Historical Context Retrieval

When I need to remember what we discussed with a client 6 months ago, I want to find that conversation and related actions in seconds, so I can maintain continuity and avoid asking them to repeat themselves.

Acceptance Criteria:

- Natural language search across all interaction history
- Results ranked by relevance
- Shows both what was discussed AND what actions were taken
- Links to original source documents

Secondary User: Operations/Client Service

Job Story 4: Client Onboarding

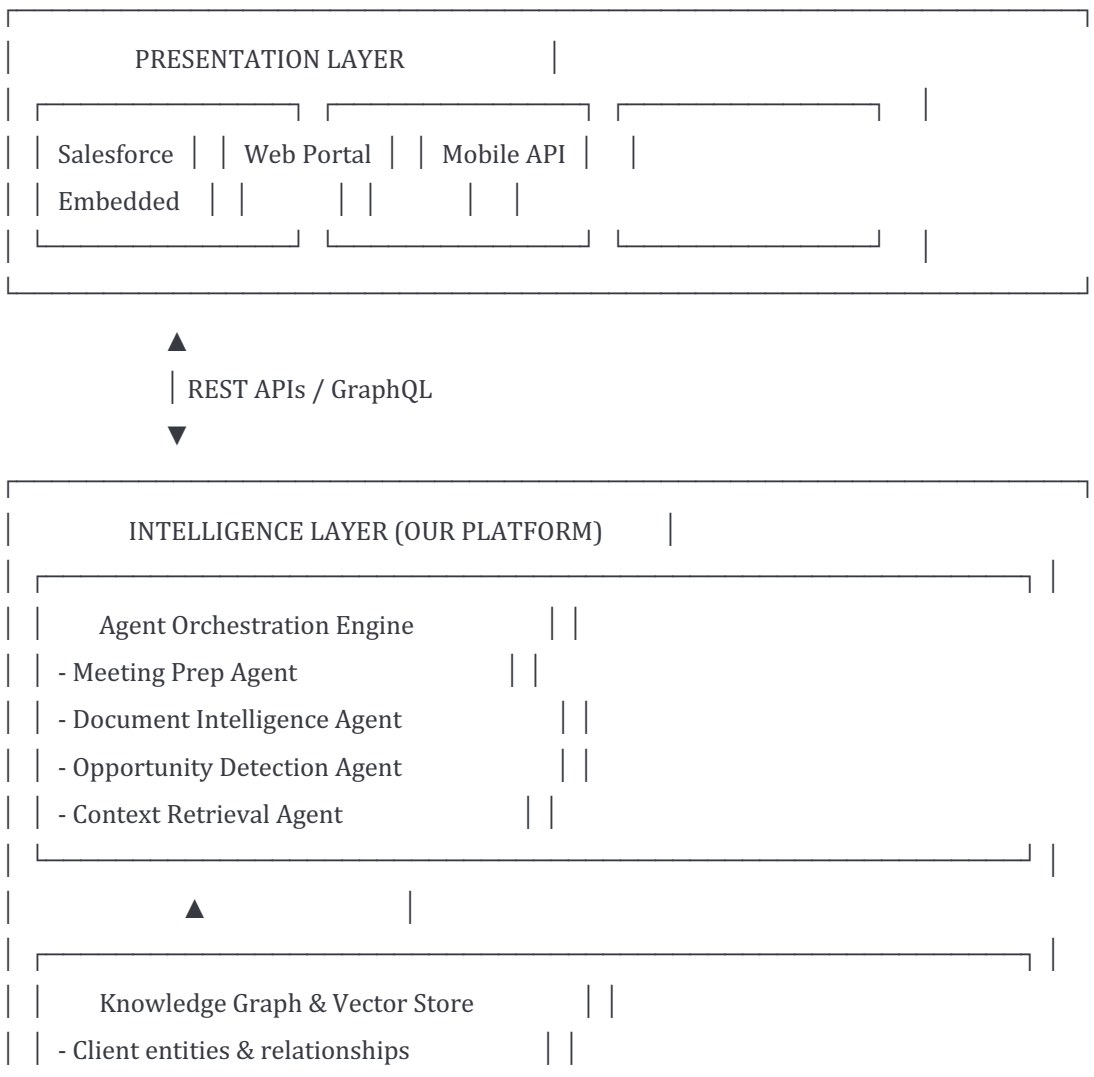
When onboarding a new client, I want the system to automatically extract data from their documents and populate all required forms, so I can focus on reviewing accuracy rather than manual data entry.

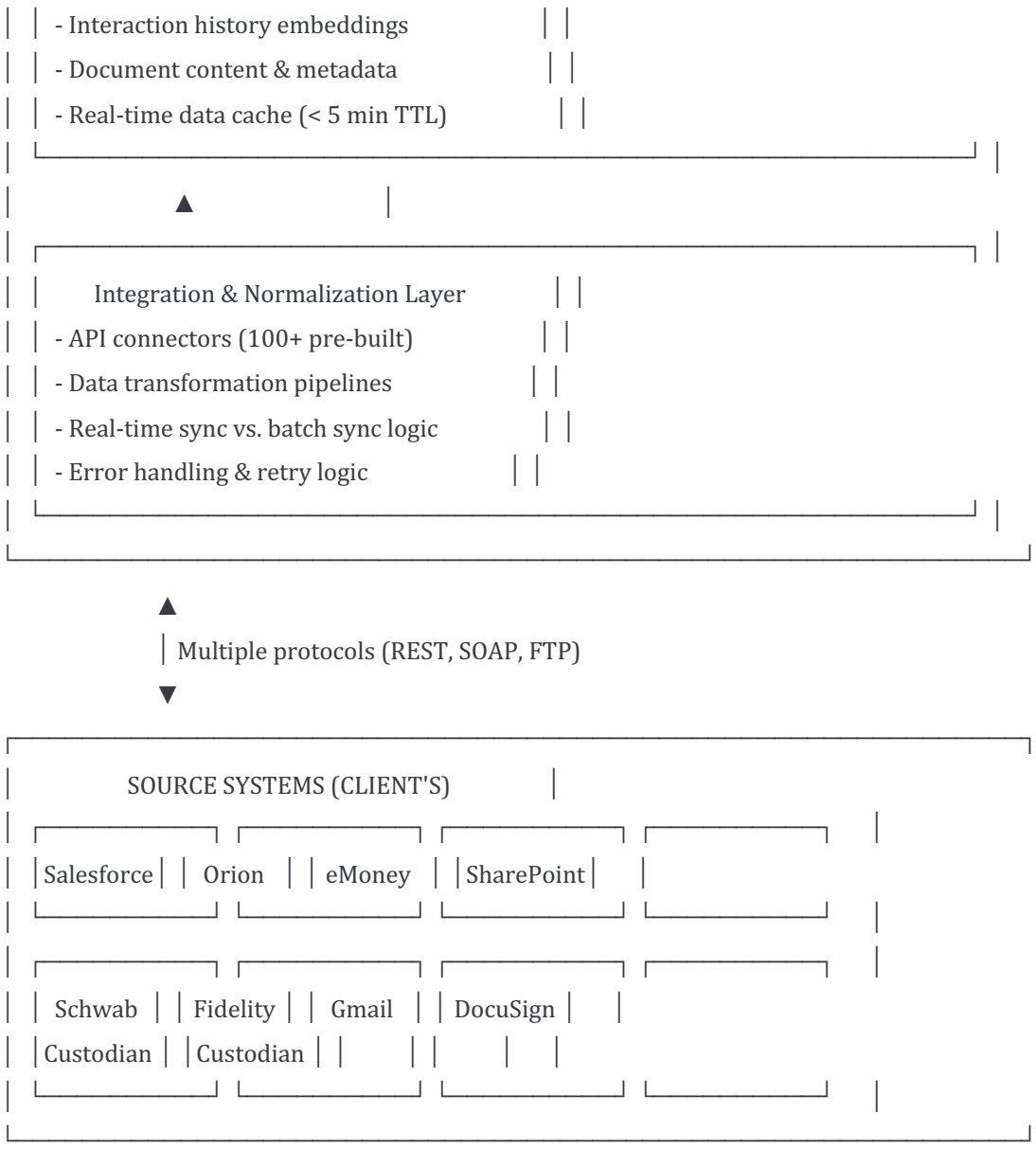
Acceptance Criteria:

- Document classification accuracy >95%
- Data extraction accuracy >90%
- Clear flagging of low-confidence extractions
- Audit trail of all automated vs. manual entries

System Architecture

High-Level Architecture Diagram





Key Architectural Principles

1. **Source Systems Remain Source of Truth**
   - We never store transactional data permanently
   - All writes go back to source systems
   - Our cache is read-only and time-limited
2. **Event-Driven Architecture**
   - Source systems trigger events (webhook preferred, polling fallback)
   - Events update knowledge graph
   - Agents react to knowledge graph changes
3. **Graceful Degradation**
   - If one system is down, others continue working

    - Clear indication to user about data freshness/availability
    - Offline mode with cached data where appropriate
4. **Tenant Isolation**
    - Each client gets dedicated knowledge graph instance
    - No cross-client data contamination
    - Encryption at rest and in transit

Integration Requirements

Phase 1 Required Integrations

For the meeting prep workflow, we need these systems connected:

1. CRM Integration (Salesforce/Redtail/Wealthbox)

Required Data Access:

- Client/Contact records (read)
- Household structure (read)
- Meeting history and notes (read/write)
- Tasks and follow-ups (read/write)
- Custom fields (read - client-specific)

Integration Method:

- **Salesforce:** REST API + Streaming API for real-time updates
- **Redtail:** REST API (polling every 15 min)
- **Wealthbox:** REST API + webhooks

Authentication: OAuth 2.0 with refresh tokens

Data Sync Strategy:

- Initial: Full sync of all client records
- Ongoing: Real-time for updates, daily batch for validation
- Direction: Bidirectional (we read and write back meeting notes)

Rate Limits to Handle:

- Salesforce: 100,000 API calls/24 hours per org
- Redtail: 5,000 calls/hour
- Wealthbox: 10 requests/second

Critical Fields Mapping:

json

```
{
  "client_entity": {
    "id": "string (source system ID)",
    "name": "string",
    "household_id": "string",
    "advisor_id": "string",
    "status": "enum [active, inactive, prospect]",
    "aum": "decimal",
    "last_meeting_date": "datetime",
    "next_meeting_date": "datetime",
    "risk_tolerance": "enum [conservative, moderate, aggressive]",
    "custom_fields": "object (client-specific)"
  }
}
```

2. Portfolio Management System (Orion/eMoney/BlackDiamond)

Required Data Access:

- Account balances (read)
- Holdings and positions (read)
- Performance metrics (read)
- Asset allocation (read)
- Billing information (read)

Integration Method:

- **Orion:** API 2.0 (REST) + Report API
- **eMoney:** Web services API (SOAP)
- **BlackDiamond:** SQL reporting database (read replica)

Authentication:

- Orion: Session-based with API key
- eMoney: WS-Security with certificate
- BlackDiamond: SQL Server authentication

Data Sync Strategy:

- Real-time: Never (too expensive on rate limits)
- Daily batch: End-of-day positions and valuations

- On-demand: When advisor requests specific client brief (<5 min before meeting)

Critical Fields Mapping:

json

```
{
  "portfolio_snapshot": {
    "account_id": "string",
    "account_type": "enum [IRA, Roth, Taxable, Trust]",
    "balance": "decimal",
    "as_of_date": "date",
    "performance_ytd": "decimal",
    "performance_inception": "decimal",
    "asset_allocation": {
      "equity": "decimal",
      "fixed_income": "decimal",
      "alternatives": "decimal",
      "cash": "decimal"
    },
    "top_holdings": [
      {
        "security": "string",
        "value": "decimal",
        "percent_of_portfolio": "decimal"
      }
    ]
  }
}
```

3. Document Storage (SharePoint/Google Drive/Ignite)

Required Data Access:

- Document metadata (read)
- Document content (read)
- Folder structure (read)
- Upload capability (write)

Integration Method:
- **SharePoint:** Microsoft Graph API
- **Google Drive:** Google Drive API v3
- **Ignite:** REST API

Authentication:
- SharePoint: Azure AD OAuth 2.0
- Google Drive: OAuth 2.0 with service account
- Ignite: API key + session token

Data Sync Strategy:
- Initial: Index all documents in client folders
- Ongoing: Real-time via webhooks (if available) or 4-hour polling
- Search: Vector embeddings updated daily

Document Classification Model:
```
Input: Document file (PDF, DOCX, XLSX)
Output: {
  "document_type": "enum [brokerage_statement, trust_document,
                   tax_return, insurance_policy, etc.]",
  "confidence": "decimal (0-1)",
  "extracted_entities": {
    "account_numbers": ["string"],
    "dates": ["datetime"],
    "dollar_amounts": ["decimal"],
    "names": ["string"]
  },
  "client_id": "string (matched to CRM)"
}
```

4. Email Integration (Gmail/Outlook)

Required Data Access:

- Emails sent to/from advisor (read)
- Email metadata (date, subject, participants)
- Attachments (read)

Integration Method:
- **Gmail:** Gmail API
- **Outlook:** Microsoft Graph API

Authentication: OAuth 2.0 with delegated permissions

Data Sync Strategy:
- Initial: Last 2 years of emails
- Ongoing: Real-time via push notifications
- Processing: Extract client references, sentiment, action items

Privacy Considerations:
- NEVER store full email content permanently
- Store only: metadata, client references, action item extractions
- User must explicitly approve email access
- Ability to exclude specific email folders/labels

Email Processing Pipeline:
```
1. Email arrives → Webhook triggers
2. Extract: participants, subject, date
3. NER: Identify if client-related (client name, account number, etc.)
4. If client-related:
   - Extract key entities (dates, dollar amounts, action items)
   - Classify intent (meeting request, document request, question, etc.)
   - Store: metadata + extracted entities (NOT full content)
   - Link to client record in knowledge graph
5. If not client-related: Discard
```

5. Custodian Integration (Schwab/Fidelity/Pershing)

Required Data Access:

- Account balances (read)

- Transaction history (read)
- Pending transfers (read)
- Document delivery status (read)

Integration Method:

- **Schwab Advisor Center:** StreetSmart API (or screen scraping fallback)
- **Fidelity WealthCentral:** WealthScape API
- **Pershing NetX360:** NetXInvestor API

Authentication:

- Varies by custodian (usually certificate-based or SAML)
- May require advisor-level credentials (not client credentials)

Data Sync Strategy:

- Daily batch: End-of-day balances
- Weekly batch: Transaction history
- On-demand: Current positions (when generating client brief)

Critical Constraint:

- Custodian APIs are notoriously unreliable and rate-limited
- MUST have fallback to portfolio management system data
- Clear indication to user if custodian data is stale

Data Model - Knowledge Graph

Core Entities

Client Entity

json

```
{
 "id": "uuid",
 "external_ids": {
  "crm": "string",
  "portfolio_system": "string",
  "custodian": ["string"]
 },
```

```
  "name": "string",
  "type": "enum [individual, joint, trust, entity]",
  "household_id": "uuid",
  "primary_advisor_id": "uuid",
  "status": "enum [active, inactive, prospect]",
  "created_at": "datetime",
  "updated_at": "datetime"
}
```

Interaction Entity

json

```
{
  "id": "uuid",
  "client_id": "uuid",
  "advisor_id": "uuid",
  "type": "enum [meeting, email, call, note]",
  "date": "datetime",
  "summary": "string (AI-generated or manual)",
  "action_items": [
    {
      "description": "string",
      "status": "enum [open, completed, cancelled]",
      "due_date": "date",
      "assigned_to": "uuid"
    }
  ],
  "source_system": "string",
  "source_id": "string (for deep linking)"
}
```

Document Entity

json

```
{
  "id": "uuid",
  "client_id": "uuid",
  "type": "enum (from classification model)",
```

```
  "classification_confidence": "decimal",
  "date": "date",
  "source_system": "string",
  "source_path": "string",
  "extracted_data": "object (varies by document type)",
  "vector_embedding": "array[float] (for semantic search)",
  "processed_at": "datetime"
}
```

Portfolio Snapshot Entity

```
json
{
  "id": "uuid",
  "client_id": "uuid",
  "account_id": "string",
  "as_of_date": "date",
  "balance": "decimal",
  "performance_metrics": {
    "ytd": "decimal",
    "1_year": "decimal",
    "3_year": "decimal",
    "inception": "decimal"
  },
  "asset_allocation": "object",
  "holdings": ["object"],
  "source_system": "string"
}
```

Graph Relationships

```
Client --[MEMBER_OF]--> Household
Client --[ADVISED_BY]--> Advisor
Client --[HAS_INTERACTION]--> Interaction
Client --[OWNS_DOCUMENT]--> Document
Client --[HAS_PORTFOLIO]--> Portfolio Snapshot
```

Interaction --[CREATED_ACTION_ITEM]--> Action Item
Interaction --[REFERENCES_DOCUMENT]--> Document
```

### **Vector Store for Semantic Search**

**What Gets Embedded:**
- Meeting summaries and notes
- Email content (summary only, not full text)
- Document content (for relevant doc types)
- Action items and their context

**Embedding Model:**
- OpenAI `text-embedding-3-large` (3072 dimensions)
- Or client's preferred model if they have data residency requirements

**Vector Database:**
- Pinecone (cloud) or
- Weaviate (self-hosted option for private cloud deployments)

**Search Strategy:**
```
User query: "When did we last discuss the Johnson's estate plan?"

1. Embed query → vector
2. Vector similarity search → top 20 results
3. Rerank using metadata filters:
   - client_id = Johnson household
   - interaction_type includes "meeting"
   - date range = last 2 years
4. LLM synthesis → natural language answer with citations
```

---

# Agent Specifications

## Agent 1: Meeting Preparation Agent
```

Trigger: Advisor navigates to client meeting prep page OR scheduled job runs 24 hours before meeting

Inputs:

- Client ID
- Meeting date/time
- Last meeting date
- Advisor ID

Processing Logic:

python

```python
def generate_meeting_brief(client_id, meeting_date):
    # Step 1: Gather context
    client = fetch_client_entity(client_id)
    last_meeting = fetch_last_interaction(client_id, type="meeting")
    portfolio = fetch_latest_portfolio_snapshot(client_id)
    recent_interactions = fetch_interactions_since(client_id, last_meeting.date)
    open_action_items = fetch_action_items(client_id, status="open")

    # Step 2: Identify changes since last meeting
    portfolio_changes = calculate_portfolio_changes(
        portfolio,
        last_meeting.date
    )

    # Step 3: Detect opportunities
    opportunities = []

    # Example opportunity: Rebalancing needed
    if portfolio_changes.drift_from_target > threshold:
        opportunities.append({
            "type": "rebalancing",
            "priority": "high",
            "description": f"Portfolio has drifted {portfolio_changes.drift}% from target allocation"
        })

    # Example opportunity: Action item overdue
```

```
for item in open_action_items:
    if item.due_date < today:
        opportunities.append({
            "type": "follow_up",
            "priority": "high",
            "description": f"Overdue: {item.description}"
        })

# Step 4: Generate summary using LLM
prompt = f"""
Generate a concise meeting preparation brief for advisor.

Client: {client.name}
Last meeting: {last_meeting.date} - {last_meeting.summary}

Portfolio changes since then:
{portfolio_changes.summary}

Recent interactions:
{format_interactions(recent_interactions)}

Open action items:
{format_action_items(open_action_items)}

Detected opportunities:
{format_opportunities(opportunities)}

Create a brief with:
1. Key discussion topics (3-5 bullets)
2. Questions to ask
3. Recommended actions

Keep it under 300 words. Focus on what's actionable.
"""

llm_response = call_llm(prompt, model="gpt-4o")
```

```
    return {
        "brief": llm_response,
        "portfolio_snapshot": portfolio,
        "opportunities": opportunities,
        "open_action_items": open_action_items,
        "recent_interactions": recent_interactions,
        "generated_at": now()
    }
```

Output:

- HTML/JSON for rendering in UI
- Deep links to source systems for each data point
- Confidence scores for AI-generated content

Error Handling:

- If portfolio system unavailable: Use last cached data + warning banner
- If CRM unavailable: Halt and show error (can't proceed without client context)
- If LLM fails: Return structured data without natural language summary

Performance SLA:

- < 5 seconds for cached/recent data
- < 15 seconds for full refresh
- Timeout at 30 seconds with partial results

Agent 2: Document Intelligence Agent

Trigger: Document uploaded to designated folder OR new document detected in sync

Inputs:

- Document file (PDF, DOCX, XLSX, etc.)
- Source system metadata
- Optional: Client ID hint

Processing Pipeline:

python

```
def process_document(file_path, source_metadata):
```

```
# Step 1: Document classification
doc_type, confidence = classify_document(file_path)

if confidence < 0.7:
    # Flag for human review
    return {
        "status": "needs_review",
        "suggested_type": doc_type,
        "confidence": confidence
    }

# Step 2: Extract structured data based on type
if doc_type == "brokerage_statement":
    extracted_data = extract_brokerage_statement(file_path)
elif doc_type == "trust_document":
    extracted_data = extract_trust_document(file_path)
elif doc_type == "tax_return":
    extracted_data = extract_tax_return(file_path)
else:
    extracted_data = extract_generic(file_path)

# Step 3: Entity resolution - which client does this belong to?
client_id = resolve_client(
    extracted_data.names,
    extracted_data.account_numbers,
    source_metadata.folder_path
)

# Step 4: Validation rules
validation_errors = validate_document(doc_type, extracted_data)

# Step 5: Store in knowledge graph
document_entity = create_document_entity(
    client_id=client_id,
    doc_type=doc_type,
    extracted_data=extracted_data,
```

```
    source_path=source_metadata.path,
    validation_errors=validation_errors
  )

  # Step 6: Generate vector embedding for search
  text_content = extract_text_for_embedding(file_path)
  embedding = generate_embedding(text_content)
  store_vector(document_entity.id, embedding)

  return {
    "status": "processed",
    "document_id": document_entity.id,
    "client_id": client_id,
    "extracted_data": extracted_data,
    "validation_errors": validation_errors
  }
```

Document Type Handlers:

Brokerage Statement Extractor:

python

```
def extract_brokerage_statement(file_path):
  # Use vision model for PDF
  images = convert_pdf_to_images(file_path)

  # Our pre-trained model (98% accuracy on financial docs)
  result = vision_model.extract(images, schema={
    "statement_date": "date",
    "account_number": "string",
    "account_type": "enum",
    "beginning_balance": "decimal",
    "ending_balance": "decimal",
    "holdings": [
      {
        "security_name": "string",
        "symbol": "string",
        "quantity": "decimal",
```

```
            "price": "decimal",
            "value": "decimal"
        }
    ],
    "transactions": [
        {
            "date": "date",
            "type": "enum [buy, sell, dividend, interest, fee]",
            "description": "string",
            "amount": "decimal"
        }
    ]
})

return result
```

Trust Document Extractor:

python

```
def extract_trust_document(file_path):
    # Trusts are complex - use LLM with structured output
    text = extract_text(file_path)

    result = call_llm(text, schema={
        "trust_name": "string",
        "trust_type": "enum [revocable, irrevocable]",
        "trustee": ["string"],
        "beneficiaries": [
            {
                "name": "string",
                "relationship": "string",
                "share": "string"
            }
        ],
        "creation_date": "date",
        "key_provisions": ["string"]
    })
```

```
    return result
```

Validation Rules:

python

```
def validate_document(doc_type, extracted_data):
    errors = []

    if doc_type == "brokerage_statement":
        # Check if statement is recent
        if extracted_data.statement_date < (today - 90 days):
            errors.append({
                "severity": "warning",
                "field": "statement_date",
                "message": "Statement is more than 90 days old"
            })

        # Check if holdings balance matches ending balance
        holdings_total = sum([h.value for h in extracted_data.holdings])
        if abs(holdings_total - extracted_data.ending_balance) > 1.00:
            errors.append({
                "severity": "error",
                "field": "balance_mismatch",
                "message": f"Holdings total ${holdings_total} doesn't match ending balance
${extracted_data.ending_balance}"
            })

    return errors
```

Agent 3: Opportunity Detection Agent

Trigger: Scheduled job (runs nightly) OR on-demand when generating client brief

Inputs:

- Client ID
- Optional: Specific opportunity types to check

Opportunity Types:

1. Rebalancing Opportunity

python

```
def detect_rebalancing_opportunity(client_id):
    portfolio = fetch_latest_portfolio(client_id)
    target_allocation = fetch_target_allocation(client_id)

    drift = calculate_drift(portfolio.allocation, target_allocation)

    if drift > client.rebalancing_threshold:
        return {
            "type": "rebalancing",
            "priority": "high" if drift > 10 else "medium",
            "description": f"Portfolio has drifted {drift}% from target",
            "recommended_action": "Schedule rebalancing review",
            "data": {
                "current_allocation": portfolio.allocation,
                "target_allocation": target_allocation,
                "drift_by_asset_class": calculate_drift_by_class()
            }
        }
```

2. Tax-Loss Harvesting Opportunity

python

```
def detect_tax_loss_harvesting(client_id):
    taxable_accounts = fetch_accounts(client_id, type="taxable")

    opportunities = []
    for account in taxable_accounts:
        holdings = fetch_holdings(account.id)

        for holding in holdings:
            if holding.unrealized_loss > threshold:
                # Check if wash sale rule would apply
                recent_purchases = check_wash_sale_risk(
```

```
            holding.security_id,
            lookback_days=30
        )

        if not recent_purchases:
            opportunities.append({
                "security": holding.security_name,
                "loss_amount": holding.unrealized_loss,
                "account": account.name
            })

    if opportunities:
        return {
            "type": "tax_loss_harvesting",
            "priority": "high",
            "description": f"Found {len(opportunities)} tax-loss harvesting opportunities totaling ${sum([o.loss_amount for o in opportunities])}",
            "data": opportunities
        }
```

3. Lending Opportunity (If client is UPTIQ user)

python

```
def detect_lending_opportunity(client_id):
    # Check if client has liquid portfolio
    portfolio = fetch_latest_portfolio(client_id)
    liquid_value = calculate_liquid_assets(portfolio)

    # Check recent interactions for mentions of large purchases
    recent_interactions = fetch_interactions(client_id, days=90)
    large_purchase_signals = scan_for_keywords(
        recent_interactions,
        keywords=["house", "buy", "purchase", "vacation home", "downpayment"]
    )

    if liquid_value > 500000 and large_purchase_signals:
        # Calculate potential loan amount
        potential_loan = liquid_value * 0.7  # 70% LTV typical
```

```
    return {
        "type": "lending",
        "priority": "medium",
        "description": f"Client may be interested in securities-backed loan (up to ${potential_loan})",
        "recommended_action": "Discuss lending options in next meeting"
    }
```

4. Insurance Gap

python

```
def detect_insurance_gap(client_id):
    # Get client life events and assets
    client = fetch_client(client_id)
    dependents = fetch_household_members(client.household_id, role="dependent")
    total_assets = fetch_total_assets(client_id)

    # Check if we have insurance policy data
    insurance_docs = fetch_documents(
        client_id,
        type="insurance_policy",
        status="active"
    )

    if not insurance_docs and (dependents or total_assets > 1000000):
        return {
            "type": "insurance_gap",
            "priority": "high",
            "description": "No life insurance policies on file for client with dependents",
            "recommended_action": "Schedule insurance needs analysis"
        }

    # Calculate coverage gap if we have policies
    if insurance_docs:
        total_coverage = sum([doc.extracted_data.death_benefit for doc in insurance_docs])
        recommended_coverage = calculate_recommended_coverage(client)
```

```
    if total_coverage < recommended_coverage * 0.8:
        return {
            "type": "insurance_gap",
            "priority": "medium",
            "description": f"Current coverage ${total_coverage} is below recommended
${recommended_coverage}",
            "data": {
                "current": total_coverage,
                "recommended": recommended_coverage,
                "gap": recommended_coverage - total_coverage
            }
        }
```

Opportunity Scoring Algorithm:

python

```
def score_opportunities(opportunities, client_context):
    """
    Prioritize opportunities based on:
    - Financial impact
    - Time sensitivity
    - Client preferences
    - Regulatory requirements
    """

    for opp in opportunities:
        score = 0

        # Financial impact (0-40 points)
        if opp.estimated_value > 100000:
            score += 40
        elif opp.estimated_value > 50000:
            score += 30
        elif opp.estimated_value > 10000:
            score += 20

        # Time sensitivity (0-30 points)
        if opp.deadline and opp.deadline < (today + 30 days):
```

```
            score += 30
        elif opp.deadline and opp.deadline < (today + 90 days):
            score += 20

        # Client engagement level (0-20 points)
        if client_context.last_interaction_days < 30:
            score += 20
        elif client_context.last_interaction_days < 90:
            score += 10

        # Regulatory/compliance (0-10 points)
        if opp.type in ["tax_loss_harvesting", "required_distribution"]:
            score += 10

        opp.priority_score = score

    return sorted(opportunities, key=lambda x: x.priority_score, reverse=True)
```

API Specifications

External API (For Client Consumption)

Base URL: `https://api.uptiq.com/v1`

Authentication: Bearer token (JWT) with scoped permissions

Endpoint: Generate Meeting Brief

```
POST /clients/{client_id}/meeting-brief
```

Request:

```
json
{
```

```
  "meeting_date": "2025-11-15T10:00:00Z",
  "force_refresh": false  // Optional: bypass cache
}
```

Response:

json


```
{
  "client_id": "uuid",
  "generated_at": "2025-11-14T15:30:00Z",
  "data_freshness": {
    "portfolio": "2025-11-13T23:59:59Z",
    "crm": "2025-11-14T15:29:00Z",
    "documents": "2025-11-14T10:00:00Z"
  },
  "summary": {
    "last_meeting": {
      "date": "2025-08-15",
      "summary": "Discussed Q2 performance and college planning",
      "action_items_completed": 3,
      "action_items_pending": 1
    },
    "key_topics": [
      "Review portfolio rebalancing opportunity (10% drift from target)",
      "Follow up on estate planning documents (pending)",
      "Discuss tax-loss harvesting opportunity ($15K potential savings)"
    ],
    "questions_to_ask": [
      "Have you completed the trust amendment we discussed?",
      "Any changes to your income or tax situation?",
      "Are you still planning to purchase vacation home next year?"
    ]
  },
  "portfolio": {
    "total_value": 2450000,
    "change_since_last_meeting": {
      "absolute": 125000,
```

```
    "percent": 5.4
   },
   "asset_allocation": {
    "equity": 0.62,
    "fixed_income": 0.25,
    "alternatives": 0.08,
    "cash": 0.05
   }
  },
  "opportunities": [
   {
    "type": "rebalancing",
    "priority": "high",
    "priority_score": 85,
    "description": "Portfolio has drifted 10.2% from target allocation",
    "estimated_value": null,
    "recommended_action": "Schedule rebalancing review"
   },
   {
    "type": "tax_loss_harvesting",
    "priority": "medium",
    "priority_score": 72,
    "description": "3 securities with unrealized losses totaling $15,200",
    "estimated_value": 15200,
    "recommended_action": "Discuss tax-loss harvesting before year-end"
   }
  ],
  "pending_action_items": [
   {
    "description": "Send updated trust documents to estate attorney",
    "due_date": "2025-09-30",
    "days_overdue": 45,
    "assigned_to": "advisor"
   }
  ],
  "recent_interactions": [
```

```
    {
      "date": "2025-10-20",
      "type": "email",
      "summary": "Client inquired about mortgage refinancing options",
      "action_required": false
    }
  ],
  "deep_links": {
    "crm_record": "https://yourcrm.com/clients/12345",
    "portfolio": "https://orion.com/accounts/...",
    "documents": "https://sharepoint.com/clients/..."
  }
}
```

Error Responses:
- `404` - Client not found
- `503` - One or more source systems unavailable (includes partial data if possible)
- `429` - Rate limit exceeded

Endpoint: Search Client History
```
GET /clients/{client_id}/search
```

Query Parameters:
- `q` (required): Search query (natural language)
- `types` (optional): Filter by interaction types (comma-separated)
- `start_date` (optional): ISO 8601 date
- `end_date` (optional): ISO 8601 date
- `limit` (optional): Default 10, max 50

Example:
```
```

GET /clients/abc-123/search?q=estate%20planning&start_date=2024-01-01&limit=5

Response:

json


```
{
  "query": "estate planning",
  "results_count": 5,
  "results": [
    {
      "id": "interaction-789",
      "type": "meeting",
      "date": "2024-08-15",
      "relevance_score": 0.92,
      "summary": "Discussed updating trust to add grandchildren as beneficiaries",
      "snippet": "...client expressed desire to include grandchildren in estate plan. Recommended trust amendment to add per stirpes distribution...",
      "source_system": "salesforce",
      "deep_link": "https://salesforce.com/..."
    },
    {
      "id": "doc-456",
      "type": "document",
      "date": "2024-06-10",
      "relevance_score": 0.87,
      "document_type": "trust_document",
      "summary": "Johnson Family Revocable Trust - Original document",
      "source_system": "sharepoint",
      "deep_link": "https://sharepoint.com/..."
    }
  ]
}
```

Endpoint: Process Document

```
```

POST /documents/process
```

**Request:** (multipart/form-data)
```
file: <binary>
client_id: "uuid" (optional - for hint)
source_system: "sharepoint" | "google_drive" | "manual_upload"

Response:

json
{
  "document_id": "uuid",
  "status": "processed" | "needs_review",
  "classification": {
    "type": "brokerage_statement",
    "confidence": 0.95
  },
  "client_id": "uuid",
  "extracted_data": {
    "statement_date": "2025-10-31",
    "account_number": "****6789",
    "ending_balance": 450000,
    "holdings_count": 23
  },
  "validation_errors": [
    {
      "severity": "warning",
      "field": "statement_date",
      "message": "Statement is 14 days old"
    }
  ],
  "deep_link": "https://yourapp.com/documents/uuid"
}
```
```

Internal APIs (Between Our Services)

Knowledge Graph Query API

```
POST /internal/kg/query
```

Purpose: Flexible graph queries for agents

Request:

```json
{
  "query": "MATCH (c:Client {id: $client_id})-[:HAS_INTERACTION]->(i:Interaction) WHERE i.date > $start_date RETURN i ORDER BY i.date DESC",
  "params": {
    "client_id": "uuid",
    "start_date": "2025-01-01"
  }
}
```

Security & Compliance Requirements

Data Encryption

At Rest:
- Database: AES-256 encryption
- Object storage: Server-side encryption with customer-managed keys
- Backups: Encrypted with separate key rotation schedule

In Transit:
- All APIs: TLS 1.3 minimum
- Internal service communication: mTLS (mutual TLS)
- Webhooks: HMAC signature verification

Authentication & Authorization

OAuth 2.0 Flows:
- Authorization Code Flow (for user-facing integrations)
- Client Credentials Flow (for service-to-service)
- Refresh token rotation mandatory

RBAC (Role-Based Access Control):
```
Roles:
- Advisor: Can view all client data they're assigned to
- Operations: Can view/edit client data, cannot delete
- Compliance: Read-only access to all data + audit logs
- Admin: Full access + user management

Permissions:
- clients.read
- clients.write
- documents.read
- documents.write
- opportunities.read
- admin.users
- admin.audit_logs
```

API Rate Limiting:
```
- Standard tier: 1,000 requests/hour per advisor
- Burst allowance: 100 requests/minute
- Document upload: 50/hour
- Search queries: 200/hour
```

Audit Logging

What Gets Logged:

- Every API call (endpoint, parameters, user, timestamp, IP)

- Every data access (which client record, which fields, by whom)
- Every AI inference (input, output, model used, confidence scores)
- Every document processed (classification, extraction results, validation)
- Authentication events (login, logout, token refresh, failures)

Meeting Brief Generation Prompt Template:

```

You are an AI assistant helping a financial advisor prepare for a client meeting.

CLIENT CONTEXT:

Name: {client_name}

Household Type: {household_type}

Total AUM: {aum}

Risk Profile: {risk_profile}

LAST MEETING ({last_meeting_date}):

Summary: {last_meeting_summary}

Action Items Discussed: {action_items}
```