# Project Specification: Investment Insights iOS App

## 1. Project Overview

**Objective:** Develop an iOS application for beginner investors in Japan that consolidates and visualizes multi-dimensional data—fundamentals, technical indicators, sentiment, macroeconomics, ESG—and explains why, when, and to what extent prices may move. The app must support Japanese (default), English, and Chinese languages, with Japanese-focused UX by default.

**Platform:** iOS (SwiftUI + Combine + MVVM) backed by Firebase BaaS

**Core Features:**

* Market Forecast
* Tracked Securities
* Economic Calendar
* Learning Center
* Settings & Notifications

**Release Target:** Launch Beta via TestFlight and official App Store release within **3 months**.

## 2. Requirements Definition

### 2.1 User Stories

1. As a novice investor, I want clear definitions of financial terms inline so I can learn while browsing market data.
2. As a user, I can favorite a security and see its Open/Mid/Close prices and multi-horizon forecast scores with concise rationale daily.
3. As a user, I can view overall market, sector, and product-category forecast scores for short-term (1 week), mid-term (1 month), and long-term (6+ months).
4. As a user, I can see upcoming economic indicators and corporate events for Japan, the US, and major economies on a calendar with impact explanations.
5. As a learner, I can access basic investment concepts, market structure guides, a glossary, and mini-lessons directly inside the app.
6. As a user, I can configure push notifications for event releases and forecast alerts.

### 2.2 Functional Requirements

1. **Market Forecast Tab**
   * Generate forecast scores (0–100) for indices, FX pairs, GICS sectors, and industry categories.
   * Swipe gesture to switch time horizon: Short (1W), Mid (1M), Long (6M+).
   * Data source annotations and rationale tags per card.
2. **Tracked Securities Tab**
   * Display only user-favorited tickers with Open, Mid, and Close prices (no realtime streaming).
   * Show short/mid/long forecast icons (▲/→/▼) with composite score.
   * Provide one-sentence summary of key drivers.
   * Tooltip buttons for term definitions.
3. **Economic Calendar Tab**
   * Monthly calendar view annotated with scheduled macro indicators and corporate events.
   * List view toggles: Today, This Week, This Month.
   * Tap event to show forecast vs actual, historical context, and brief impact analysis.
4. **Learning Center Tab**
   * Sections: Product Guide, Market Guide, Glossary, Tutorials.
   * Content stored in Firestore, pre-populated via AI-summarized translations from Investopedia, major securities firms, central banks, IMF, OECD.
   * List and detail screens with inline tooltips and diagrams.
5. **Settings Tab**
   * Push notification toggles (economic events, forecast signals).
   * Account login/logout using Firebase Auth (email/SNS).
   * Display and record agreement to terms of use and disclaimers.

### 2.3 Non-Functional Requirements

* **Performance:** App launch under 3 seconds on iPhone 11 and above.
* **Offline Support:** Cache latest data locally to maintain viewing when offline.
* **Security:** Firebase Security Rules enforce read/write access; protect API keys via Secret Manager.
* **Scalability:** Firestore auto-scales with increasing data size and users.
* **Internationalization:** Default Japanese UI, with in-app switch to English or Chinese translation.

## 3. User Interface Design & Screen Flows

### 3.1 Tab Bar (Bottom)

| Icon | Label |
| --- | --- |
| 🔮 | Forecast |
| ⭐️ | My Picks |
| 📅 | Calendar |
| 📚 | Learn |
| ⚙️ | Settings |

### 3.2 Forecast Screen

* **Time Horizon Picker:** Segmented control: 1W | 1M | 6M+
* **Card Collections:**
  1. **Global Indices:** Nikkei 225, TOPIX, DJIA, S&P 500, Nasdaq Composite, FTSE 100, DAX, CAC 40, Hang Seng, Shanghai Composite, KOSPI, ASX 200
  2. **Major FX Pairs:** USD/JPY, EUR/JPY, GBP/JPY, AUD/JPY, USD/EUR, USD/GBP, USD/CAD, USD/CHF
  3. **GICS Sectors:** Technology, Financials, Healthcare, Consumer Discretionary, Consumer Staples, Energy, Utilities, Industrials, Materials, Communication Services, Real Estate
  4. **Industry Categories:** Semiconductors, Automobiles, Retail, Pharmaceuticals, Airlines, Renewable Energy, Oil & Gas, Banking & Finance, E‑commerce, Real Estate
* **Data Sources:** Alpha Vantage, Yahoo Finance, Quandl, Financial Modeling Prep
* **Card Contents:** Score (0–100), trend icon (▲/→/▼), driver tags (e.g., interest rates, economic indicators, sentiment)
* **Interactions:** Tap card to drill into related securities list or calendar context.

### 3.3 My Picks Screen

* **Search & Filter Bar:** Filter by sector or forecast direction.
* **Tickers List:** Each row shows:
  + Ticker symbol and name
  + Open, Mid, Close prices
  + Forecast icons for 1W/1M/6M
  + One-line explanation: e.g., “High trading volume indicates buying pressure.”
  + Favorite star toggle
* **Navigation:** Tap row → Security Detail screen.

### 3.4 Security Detail Screen

* **Header:** Ticker name, symbol, composite forecast score.
* **Price Chart:** Three-point line chart for open, mid, close using Swift Charts.
* **Horizon Tabs:** 1W | 1M | 6M+ to switch forecast view.
* **Driver Breakdown Table:** Five factors with quantitative metrics and qualitative commentary: | Factor | Score | Quantitative Metric | Qualitative Commentary | | ———– | —– | ——————————————– | ——————————————————————- | | Fundamental | 68 | Revenue YoY +8%, EPS YoY +12%, P/E 10x | Balance of valuation and earnings growth supports a bullish stance. | | Technical | 82 | 50-day MA breakout, RSI 60, Volume +20% | Uptrend confirmed; buying momentum strengthening. | | Sentiment | 75 | News positive rate 70%, Social mentions +30% | Sentiment improving with more positive coverage. | | Macro | 55 | BOJ rate hold, USD/JPY ±0.2% | Stable interest rate environment; currency risk low. | | ESG | 48 | E:65 / S:45 / G:40 | Governance lagging, watch for long-term risks. |
* **Auto-generated Summary:** > “Fundamental indicators are strong with stable growth. Technicals show a breakout above the 50-day moving average without overheating. Sentiment is positive. However, lower governance scores indicate potential long-term risks.”
* **Tooltips:** “?” icons next to each term open a modal with definitions in the selected language.

### 3.5 Calendar Screen

* **Monthly Calendar View:** Dots under dates with events.
* **Event List View:** Toggle between Today, This Week, This Month.
* **Event Detail Popup:** Shows actual vs forecast vs prior, brief impact note.

### 3.6 Learning Screen

* **Sections:**
  1. **Product Guide:** How to read indices and FX charts, liquidity basics.
  2. **Market Guide:** Structure and interaction of equity, bond, FX, commodity markets.
  3. **Glossary:** 50+ key terms (P/E, P/B, ROE, RSI, CPI, NFP, Volatility).
  4. **Tutorials:** 3–5 slide mini-courses on chart reading, financial statements, economic indicators, risk management.
* **Content Population:** Firestore collection guides/{section}/{document} prefilled via AI-generated, human-reviewed summaries from Investopedia, major securities firms, central bank sites, IMF/OECD.
* **Presentation:** Tapping a list item opens a detailed modal with inline diagrams and tooltips.

### 3.7 Settings Screen

* **Notifications:** Toggles for economic events and forecast signals.
* **Account:** Firebase Auth login/logout, language selection (JP/EN/CN).
* **Legal:** Terms of Use, Privacy Policy, Disclaimer acknowledgement.

## 4. Data & Analysis Engine

### 4.1 Firestore Data Model

/market\_predictions/{horizon}/{category}:  
 score: number  
 factors: {  
 fundamental: number,  
 technical: number,  
 sentiment: number,  
 macro: number,  
 esg: number  
 }  
 tags: [string]  
  
/stocks/{ticker}:  
 prices: [ { date: string, open: number, high: number, low: number, close: number } ]  
 forecasts: {  
 1W: { score, breakdown, comments },  
 1M: { … },  
 6M: { … }  
 }  
  
/calendar/{yyyy-MM-dd}:  
 events: [ { type, name, actual, forecast, previous, analysis } ]  
  
/guides/{section}/{docId}:  
 title: string  
 content: string  
 lastUpdated: timestamp

### 4.2 Forecast Calculation Logic

# Baseline weights sum to 1.0  
weights = {  
 'fundamental': 0.5,  
 'technical': 0.2,  
 'sentiment': 0.2,  
 'macro': 0.05,  
 'esg': 0.05  
}  
  
# Detect major external events and adjust weights dynamically  
if detect\_major\_event():  
 weights['macro'] += 0.1  
 weights['sentiment'] += 0.1  
 # Normalize  
 total = sum(weights.values())  
 weights = {k: v/total for k,v in weights.items()}  
  
# Machine learning optimization  
ml\_weights = train\_ml\_weights(historical\_factor\_data)  
weights.update(ml\_weights)  
  
# Compute score for each horizon  
factors = fetch\_factors\_for\_ticker(ticker, horizon)  
score = sum(factors[k] \* weights[k] for k in weights)

* **detect\_major\_event():** Uses NLP on news volume and social spikes to flag geopolitical risk.
* **train\_ml\_weights():** Applies Random Forest feature importance on historical factor vs return data.

### 4.3 Correlation & Machine Learning

1. Store factor time series in TimescaleDB.
2. Compute correlation matrices with Pandas .corr().
3. Train Random Forest regressors to obtain feature importance.
4. Update Firestore /weights/{horizon} documents nightly.

## 5. Technical Stack

| Layer | Technology |
| --- | --- |
| Frontend (iOS) | SwiftUI, Combine, Swift Charts, Localization APIs |
| Backend (BaaS) | Firebase Auth, Firestore, Cloud Functions (Node.js/Python), Scheduler |
| Analytics & ML | Python, Pandas, scikit-learn running in Firebase Cloud Functions |
| Notifications | Firebase Cloud Messaging |
| CI/CD | GitHub Actions, Fastlane, TestFlight, App Store Connect |
| Design | Figma |

## 6. Setup & Deployment Guide

### A. Firebase & Backend Setup

1. **Create Firebase Project**
   * In Firebase Console, create a new project named “InvestmentInsights”.
   * Enable **Authentication**, **Firestore**, **Cloud Functions**, **Cloud Scheduler**, and **Cloud Messaging**.
   * Add your iOS app bundle ID and download GoogleService-Info.plist. Add this file to Resources/ in Xcode and exclude it from version control; configure Secret Manager for any API secrets.
2. **Configure Firestore Rules**
   * Write security rules to allow read access for public data collections (/market\_predictions, /calendar, /guides), and restrict write access to authenticated Cloud Functions.
3. **Initialize Cloud Functions & Emulator**
   * Install Firebase CLI: npm install -g firebase-tools.
   * In project root: firebase login && firebase init functions firestore emulators.
   * Implement data-fetching functions in functions/ (Node.js or Python), referencing external APIs (Alpha Vantage, News API, etc.).
   * Configure **Cloud Scheduler** in functions/index.js to trigger these functions daily:
   * exports.scheduledDataFetch = functions.pubsub.schedule('every 24 hours').onRun(async (context) => {  
      // fetch and write to Firestore  
     });
4. **Deploy Cloud Functions**
   * Test locally with firebase emulators:start --only functions,firestore.
   * Deploy to production: firebase deploy --only functions,firestore.

### B. ML & Forecast Engine

1. **Implement Weight-Training Cloud Function (Python)**
   * **Runtime**: Python 3.10 (Cloud Functions 2nd gen)
   * **File structure**: create a folder functions/python\_train\_weights/:
   * functions/  
     ├── python\_train\_weights/  
     │ ├── main.py  
     │ └── requirements.txt  
     └── ...
   * **requirements.txt**:
   * google-cloud-firestore  
     pandas  
     scikit-learn  
     numpy  
     functions-framework
   * **main.py**:
   * import os  
     import numpy as np  
     import pandas as pd  
     from google.cloud import firestore  
     from sklearn.ensemble import RandomForestRegressor  
     from functions\_framework import cloud\_event  
       
     @cloud\_event  
     def train\_forecast\_weights(cloud\_event):  
      # Initialize Firestore client  
      db = firestore.Client()  
       
      # Load historical factor data  
      # Expected collection: 'historical\_factors/{horizon}/{ticker}' with documents containing factor fields  
      factor\_records = []  
      target\_records = []  
      horizons = ['1W', '1M', '6M']  
      factor\_names = ['fundamental', 'technical', 'sentiment', 'macro', 'esg']  
       
      for horizon in horizons:  
      docs = db.collection('historical\_factors').document(horizon).collections()  
      for coll in docs:  
      for doc in coll.stream():  
      data = doc.to\_dict()  
      # Assume data contains factors and actual return for this horizon  
      factor\_records.append([data.get(f, 0) for f in factor\_names])  
      target\_records.append(data.get('actual\_return', 0))  
       
      # Prepare DataFrame  
      X = pd.DataFrame(factor\_records, columns=factor\_names)  
      y = np.array(target\_records)  
       
      if len(X) < 10:  
      print("Not enough data to train model.")  
      return  
       
      # Train Random Forest for each horizon separately  
      for idx, horizon in enumerate(horizons):  
      # Filter records by horizon if necessary; here using all data for simplicity  
      model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
      model.fit(X, y)  
      importances = model.feature\_importances\_  
       
      # Normalize importances to sum to 1  
      total = importances.sum()  
      weights = {name: float(imp / total) for name, imp in zip(factor\_names, importances)}  
       
      # Write weights back to Firestore  
      db.collection('weights').document(horizon).set(weights)  
      print(f"Updated weights for {horizon}: {weights}")
   * **Configuration**: In firebase.json, add Python function entrypoint:
   * {  
      "functions": [  
      {  
      "source": "functions/python\_train\_weights",  
      "runtime": "python310",  
      "entryPoint": "train\_forecast\_weights"  
      }  
      ]  
     }
   * **Scheduling**: Use Cloud Scheduler to trigger the function via Pub/Sub topic:
   * gcloud scheduler jobs create pubsub train-weights-job \  
      --schedule="0 2 \* \* \*" \  
      --topic=projects/YOUR\_PROJECT/topics/cloud-functions-publish \  
      --message-body='{"name":"train\_forecast\_weights"}'
2. **Baseline Weight Logic**
   * Deploy a second Cloud Function calculateForecast (Node.js/Python) that:
     + Reads raw factor values and weights from Firestore
     + Computes forecast scores for each horizon
     + Writes results to /market\_predictions and /stocks/{ticker}/forecasts
3. **No Docker Needed**
   * By using Python runtime in Cloud Functions 2nd gen, you avoid managing containers.

### C. iOS Project Setup iOS Project Setup

1. **Initialize Xcode Workspace**
   * Open Xcode; **File > New > Workspace…** name it InvestmentInsights.
   * Inside, create two projects: one for the main App target and one for a Swift Package to hold shared Models and Utilities.
2. **Create App Target**
   * **File > New > Project… > App (iOS)**. Name it InvestmentInsightsApp, select **SwiftUI** interface and **SwiftUI App** life cycle.
3. **Organize Folder Structure**
   * Create groups in Xcode: App/, Screens/, ViewModels/, Models/, Services/, Utilities/, Resources/.
   * Add Assets.xcassets and Info.plist to Resources/.
4. **Dependency Management via SPM**
   * In Xcode: **File > Swift Packages > Add Package Dependency…**
     + https://github.com/firebase/firebase-ios-sdk.git (select Auth, Firestore, Messaging)
     + https://github.com/davecom/SwiftCharts.git (for charts)
     + https://github.com/realm/SwiftLint.git (for linting)
5. **Environment Configuration**
   * Create AppEnvironment.swift to manage configuration:
   * struct AppEnvironment: ObservableObject {  
      let firestore = Firestore.firestore()  
      let messaging = Messaging.messaging()  
      // other shared services  
     }
   * Load GoogleService-Info.plist automatically via FirebaseApp.configure() in InvestmentInsightsApp.swift.
6. **Initialize Firebase in App Entry**

* @main  
  struct InvestmentInsightsApp: App {  
   @UIApplicationDelegateAdaptor(AppDelegate.self) var delegate  
   var body: some Scene {  
   WindowGroup {  
   ContentView()  
   .environmentObject(AppEnvironment())  
   }  
   }  
  }  
    
  class AppDelegate: NSObject, UIApplicationDelegate {  
   func application(\_ application: UIApplication,  
   didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool {  
   FirebaseApp.configure()  
   return true  
   }  
  }

1. **Localization Setup**
   * Enable Base Internationalization: Project > Info > Localizations: add ja, en, zh-Hans.
   * Create Localizable.strings files for each language, use NSLocalizedString in code.
2. **Core Data Caching (Optional)**
   * If extra offline caching is desired, add a lightweight Core Data model with entities mirroring Firestore collections.
   * Use PersistenceController to manage the Core Data stack.
3. **Linting & Formatting**
   * Integrate SwiftLint: add run script in Build Phases: /usr/local/bin/swiftlint.
   * Configure .swiftlint.yml with your preferred rules.
4. **Project Configuration**
   * Add GoogleService-Info.plist to .gitignore.
   * Ensure API keys and secrets live in Secret Manager, not in code.

### D. CI/CD Pipeline

1. **GitHub Actions**
   * Create .github/workflows/ci.yml to build and test the app on push:
   * jobs:  
      build:  
      runs-on: macos-latest  
      steps:  
      - uses: actions/checkout@v2  
      - name: Set up Ruby for Fastlane  
      uses: ruby/setup-ruby@v1  
      - name: Install dependencies  
      run: bundle install  
      - name: Build and test  
      run: fastlane test
2. **Automate TestFlight Deployment**
   * Add fastlane/Appfile and fastlane/Fastfile to manage certificates and TestFlight submission.
   * Configure a GitHub Action release.yml to trigger on release tags: build, archive, and upload to TestFlight.

### E. App Store Release

1. **Internal Testing**
   * Invite team members via TestFlight; gather crash reports and feedback.
2. **External Testing**
   * Configure TestFlight external testers; ensure compliance with Apple guidelines.
3. **App Store Submission**
   * Prepare metadata, screenshots (JP/EN/CN).
   * Complete App Store Connect listing and submit for review.

## 7. AI Agent Collaboration Tips

* **UI Code Generation:** Provide Figma JSON or screenshots to the agent with a prompt like “Generate SwiftUI layouts for these designs.”
* **Content Summarization:** Direct the agent to “Summarize Investopedia’s article on RSI in Japanese (~150 words) with tooltips for technical terms.”
* **Integration Snippets:** Ask the agent to “Produce Swift code to fetch Firestore collection /stocks and map to ViewModels.”

## 8. Disclaimer

**Disclaimer:** This app provides informational content only and is not an investment advisory service. Forecasts and analyses are based on historical data and do not guarantee future performance. Investment decisions are solely the user’s responsibility.

## 9. Implementation Roadmap: Step-by-Step Plan

Follow these steps in sequential order to build and ship the MVP within 2–3 months. Core data pipelines and Forecast UI come first; Authentication and polish are last.

1. **Initialize Firebase Project**
   * Create the Firebase project and enable Auth, Firestore, Functions, Scheduler, and Messaging.
   * Secure API keys and external secrets via Secret Manager.
2. **Set Up Local Development Environment**
   * Clone the repository, open Xcode and create the SwiftUI workspace.
   * Install Firebase SDK and other dependencies via Swift Package Manager.
   * Initialize Firebase Functions and Firestore emulator with firebase init.
3. **Configure Firestore Security Rules**
   * Define rules to allow read-only access for public collections (/market\_predictions, /calendar, /guides) and restrict write access to Cloud Functions.
4. **Implement Weight-Training Cloud Function**
   * Create trainForecastWeights in functions/ to compute and update weights in Firestore.
   * Schedule via Cloud Scheduler to run daily.
5. **Implement Data-Fetching Cloud Functions**
   * Write Functions (Node.js/Python) to fetch raw factors (market prices, indices, FX, sentiment, calendar events) from external APIs.
   * Schedule these functions to run daily via Cloud Scheduler.
6. **Verify Data Pipeline End-to-End**
   * Confirm raw factors are stored in Firestore or TimescaleDB.
   * Ensure Cloud Run weights and Functions compute forecast scores and write /market\_predictions and /stocks/{ticker}/forecasts correctly.
7. **Design & Implement Forecast Tab UI**
   * Develop SwiftUI views for the time-horizon picker and forecast cards.
   * Bind to Firestore data, display scores, trend icons, and driver tags.
8. **Validate Forecast UI & Caching**
   * Test Firestore reads and implement offline caching with local persistence.
   * Simulate data updates to verify UI refresh.
9. **Implement My Picks Tab UI**
   * Build list view showing favorites with Open/Mid/Close, forecast icons, and summary text.
   * Integrate favorite toggles and data binding to /stocks/{ticker}/forecasts.
10. **Implement Calendar Tab UI**
    * Create monthly calendar and list toggle views in SwiftUI.
    * Link to /calendar data and display event detail popups.
11. **Implement Learning Center UI**
    * Render category list and detail modals for guides from Firestore.
    * Add inline tooltips and diagrams for key concepts.
12. **Implement Settings & Notifications UI**
    * Add toggles for push notifications via Firebase Cloud Messaging.
    * Display legal links and disclaimer acknowledgment.
13. **Add Firebase Auth Integration**
    * Implement email and social login/logout flows.
    * Secure write operations with authenticated user checks in Firestore Rules.
14. **Localization & UI Polish**
    * Localize all strings for Japanese (default), English, and Simplified Chinese.
    * Refine animations, loading states, and accessibility labels.
15. **Testing, CI/CD & App Store Release**
    * Write unit tests for Functions and forecast logic; add UI tests for core flows.
    * Configure GitHub Actions and Fastlane for TestFlight distribution.
    * Gather TestFlight feedback, fix critical issues, and submit final build to the App Store.