Below is an example of a conversational flow between a financial advisor and an AI copilot that goes far beyond a static, traditional application interface. In this scenario, the copilot not only pulls together key data points in real time but also proactively offers strategic insights, compares alternatives, and handles scheduling—all through natural dialogue that empowers the advisor to focus on high-level client strategy.

Advisor: "Good morning, Copilot. What's the latest update on my key clients' portfolios?"

Copilot: "Good morning! Here's a quick snapshot:

- **Client A:** Portfolio up 6% YTD, with strong performance in technology.
- **Client B:** Down 2% this week due to volatility in energy stocks.
- **Client C:** Stable, with a modest 1% gain. Would you like to dive into any specific client's details?"

Advisor: "Let's review Client B. Why are they underperforming, and what can be done to mitigate this risk?"

Copilot: "Client B's portfolio has significant exposure to energy, which has been volatile due to recent geopolitical tensions. I've run a risk analysis that shows if you reduce the energy allocation by 10% and shift those funds into defensive sectors like healthcare and consumer staples, overall portfolio volatility could drop by 1.5% while potentially boosting risk-adjusted returns by around 0.8%. Would you like a detailed comparative report on this adjustment?"

Advisor: "Yes, please generate that report and update me on any market news that might affect this strategy."

Copilot: "Report is being generated now. Additionally, I've identified that both Reuters and Bloomberg have reported a positive outlook on the healthcare sector—great timing for a shift. I'll notify you once the report is ready."

After a brief pause...

Copilot: "The comparative report is ready and has been added to your secure dashboard. It includes historical performance, potential gains, and risk adjustments. I also suggest proactively informing Client B about this strategy shift. Would you like me to draft an email summarizing these insights?"

Advisor: "Yes, draft that email. Also, schedule a follow-up meeting with Client B to discuss these changes in detail."

Copilot: "Drafting the email now... Done! I've sent it to Client B's registered email address. I've also checked your calendar and scheduled a meeting for next Tuesday at 3 PM to review the adjustments with them. Is there anything else you'd like to explore?"

Advisor: "Can you also provide a quick view of any regulatory updates or recent SEC filings that might influence our upcoming strategy session?"

Copilot: "Certainly. There's a recent 10-Q filing from XYZ Inc. that shows important risk disclosures, and the SEC updated its guidelines on energy investments last week. I've summarized these updates and added them to your briefing package. You can review them on your dashboard."

Advisor: "Excellent work, Copilot. This integrated conversation saves me hours of manual research and lets me prepare high-value, tailored advice for my clients."

Copilot: "I'm glad to help! I'm here to ensure you always have the most relevant, up-to-date insights at your fingertips. Let me know anytime you need further analysis or scheduling adjustments."

Why This Conversational Flow Is More Valuable

- **Proactivity:** Instead of passively displaying data, the copilot analyzes trends, simulates alternative strategies, and offers tailored recommendations.
- **Interactivity:** The advisor can ask follow-up questions and explore scenario analyses in real time, rather than navigating through multiple static screens.
- **Efficiency:** Routine tasks like report generation, email drafting, and meeting scheduling are handled within the conversation—saving significant time.
- **Personalization:** Insights are customized to each client's portfolio and risk profile, with context-sensitive alerts (e.g., regulatory updates and market news) integrated into the dialogue.
- **Streamlined Decision-Making:** The copilot aggregates data from multiple sources (market data, regulatory filings, CRM, etc.) into a cohesive briefing, empowering the advisor to make well-informed, strategic decisions.

This conversational approach turns what used to be a series of disconnected tasks into a seamless, interactive advisory session—delivering higher value than traditional application interfaces.

Below is a set of underlying APIs that could power the advanced conversational flow between an advisor and the copilot. Each API focuses on a specific data source or functionality required during the conversation. They work together to pull real-time data, perform analyses, generate reports, and handle scheduling and communications. Here's how they could be defined:

---

1. Client Profile API

- **Endpoint:** `GET /api/clients/{clientId}`
- **Purpose:** Retrieve detailed client information—including name, risk profile, historical interactions, and meeting history—from the CRM system.
- **Example Response:**

  ```json
  {
    "clientId": "123",
    "name": "Jane Doe",
    "riskProfile": "Moderate",
    "lastMeeting": "2024-01-15",
    "preferences": {
      "investment": "stocks",
      "communication": "email"
    }
  }
  ```

---

2. Portfolio API

- **Endpoint:** `GET /api/clients/{clientId}/portfolio`
- **Purpose:** Fetch current portfolio performance, asset allocation, returns, and other key performance indicators.
- **Example Response:**

```
{
  "clientId": "123",
  "performanceYTD": "5.8%",
  "assetAllocation": {
    "equities": 60,
    "fixedIncome": 25,
    "alternatives": 15
  },
  "sectorBreakdown": {
    "technology": "30%",
    "healthcare": "20%",
    "energy": "15%"
  }
}
```

---

3. Risk Analysis / Scenario API

- **Endpoint:** `POST /api/clients/{clientId}/scenario-analysis`
- **Purpose:** Accept scenario parameters (e.g., a 1% increase in interest rates) and return the simulated impact on the portfolio (e.g., potential loss/gain, changes in volatility).
- **Request Example:**

```
{
  "scenario": "interest_rate_rise",
  "rateIncrease": 1.0
}
```

- **Example Response:**

```
{
  "clientId": "123",
  "expectedImpact": {
    "fixedIncome": "-1.2%",
    "overallReturn": "-0.8%"
  },
  "recommendation": "Consider shifting 10% from energy to defensive
sectors."
}
```

---

4. Market Data API

- **Endpoint:** `GET /api/market/quotes`
- **Purpose:** Provide up-to-date market quotes, index levels, and economic indicators.
- **Example Response:**

```
{
  "AAPL": 175.23,
  "GOOG": 2801.50,
  "TSLA": 720.15,
  "indices": {
    "S&P500": 4300,
    "DowJones": 35000
  }
}
```

---

5. Financial News API

- **Endpoint:** `GET /api/news/financial`
- **Purpose:** Retrieve the latest headlines and news articles from sources such as Reuters or Bloomberg that are relevant to the investor's portfolio.
- **Example Response:**

```
[
  {
    "headline": "Healthcare stocks surge amid new policy reforms",
    "source": "Reuters",
    "timestamp": "2024-02-01T08:00:00Z"
  },
  {
    "headline": "Energy sector faces volatility due to geopolitical tensions",
    "source": "Bloomberg",
    "timestamp": "2024-02-01T09:30:00Z"
  }
]
```

---

6. Reporting / Comparative Analysis API

- **Endpoint:** `POST /api/reports/generate`

- **Purpose:** Generate a detailed comparative analysis report based on input parameters (such as comparing alternative portfolio adjustment strategies).
- **Request Example:**

```
{
  "clientId": "123",
  "analysisType": "comparative",
  "options": [
    {"strategy": "shift_to_defensive", "percentage": 5},
    {"strategy": "diversify_with_commodities", "percentage": 3}
  ]
}
```

- **Example Response:**

```
{
  "clientId": "123",
  "report": "Detailed analysis showing that increasing exposure to defensive stocks could lower portfolio volatility by 1.5% and boost risk-adjusted returns by 0.8%. Full details available in the attached PDF."
}
```

---

7. Email Communication API

- **Endpoint:** `POST /api/email/draft`
- **Purpose:** Draft and send emails based on generated insights. This API accepts email content, subject, recipient address, and other metadata.
- **Request Example:**

```
{
  "recipient": "clientB@example.com",
  "subject": "Portfolio Strategy Update",
  "body": "Dear Client, please find attached the comparative analysis report for the proposed portfolio adjustment..."
}
```

- **Example Response:**

```
{ "status": "success", "message": "Email sent successfully." }
```

---

8. Calendar / Scheduling API

- **Endpoint:** `POST /api/calendar/schedule`
- **Purpose:** Schedule meetings by adding events to the advisor’s calendar. This API handles details like meeting time, duration, participants, and location (or virtual meeting link).
- **Request Example:**

```
{
  "advisorId": "adv456",
  "clientId": "123",
  "title": "Follow-up Meeting with Client B",
  "startTime": "2024-02-20T15:00:00Z",
  "duration": 60
}
```

- **Example Response:**

```
{ "status": "success", "eventId": "evt789", "message": "Meeting scheduled
successfully." }
```

---

9. Regulatory and Compliance API

- **Endpoint:** `GET /api/regulatory/updates`
- **Purpose:** Retrieve the latest regulatory updates, SEC filings, or compliance changes that may affect investment strategies.
- **Example Response:**

```
[
  {
    "document": "10-Q",
    "company": "XYZ Inc",
    "filingDate": "2024-01-22",
    "details": "Updated risk disclosures regarding energy investments."
  },
  {
    "document": "SEC Guidelines",
    "issueDate": "2024-01-18",
    "details": "New guidelines on sustainable investing."
  }
]
```

How These APIs Power the Conversation

In the conversational flow, the advisor interacts with the copilot, which under the hood makes calls to these APIs:

- When the advisor asks for a portfolio update, the copilot calls the **Client Profile API** and **Portfolio API**.
- For risk and scenario analysis, the **Risk Analysis API** is invoked.
- To discuss market conditions, the **Market Data API** and **Financial News API** are used.
- When the advisor requests a detailed report, the **Reporting API** generates a comparative analysis.
- To handle follow-up communications, the **Email API** drafts and sends an email.
- The **Calendar API** schedules meetings, and the **Regulatory API** provides compliance updates.

This orchestration of multiple API calls via a natural language interface transforms static data into a dynamic, interactive conversation—delivering much higher value than a traditional application interface.

These API definitions form the backbone of the copilot’s functionality, enabling it to aggregate, analyze, and present data in real time, all while engaging in an interactive dialogue with the advisor.

Below is an example solution that shows how you could simulate these APIs using Spring Boot. In this example, we define several model classes (POJOs), a controller that maps the endpoints (each returning stubbed JSON responses), and an OpenAPI YAML file that documents the API.

---

1. Model Classes

Below are some representative model classes.

Client.java

```java
package com.example.copilot.model;

import java.time.LocalDate;
import java.util.Map;

public class Client {
    private String clientId;
    private String name;
    private LocalDate lastMeeting;
    private Map<String, String> preferences; // e.g., {"risk": "moderate", "investment": "stocks"}

    // Getters and setters

    public String getClientId() {
        return clientId;
    }
    public void setClientId(String clientId) {
        this.clientId = clientId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public LocalDate getLastMeeting() {
        return lastMeeting;
    }
    public void setLastMeeting(LocalDate lastMeeting) {
        this.lastMeeting = lastMeeting;
    }
    public Map<String, String> getPreferences() {
        return preferences;
    }
    public void setPreferences(Map<String, String> preferences) {
        this.preferences = preferences;
    }
}
```

FinancialAccount.java

```
package com.example.copilot.model;

import java.math.BigDecimal;

public class FinancialAccount {
    private String account;
    private BigDecimal balance;

    // Getters and setters
    public String getAccount() {
        return account;
    }
    public void setAccount(String account) {
        this.account = account;
    }
    public BigDecimal getBalance() {
        return balance;
    }
    public void setBalance(BigDecimal balance) {
        this.balance = balance;
    }
}
```

MarketQuote.java

```
package com.example.copilot.model;

import java.math.BigDecimal;

public class MarketQuote {
    private String ticker;
    private BigDecimal price;
    // Getters and setters
    public String getTicker() {
        return ticker;
    }
    public void setTicker(String ticker) {
        this.ticker = ticker;
    }
    public BigDecimal getPrice() {
        return price;
    }
    public void setPrice(BigDecimal price) {
        this.price = price;
    }
}
```

FinancialNews.java

```
package com.example.copilot.model;

import java.time.Instant;

public class FinancialNews {
    private String headline;
    private String source;
    private Instant timestamp;

    // Getters and setters
    public String getHeadline() {
        return headline;
    }
    public void setHeadline(String headline) {
        this.headline = headline;
    }
    public String getSource() {
        return source;
    }
    public void setSource(String source) {
        this.source = source;
    }
    public Instant getTimestamp() {
        return timestamp;
    }
    public void setTimestamp(Instant timestamp) {
        this.timestamp = timestamp;
    }
}
```

PortfolioAnalysis.java

```java
package com.example.copilot.model;

public class PortfolioAnalysis {
    private String clientId;
    private String performance;  // e.g., "8% YTD"
    private String risk;         // e.g., "Low to Moderate"

    // Getters and setters
    public String getClientId() {
        return clientId;
    }
    public void setClientId(String clientId) {
        this.clientId = clientId;
    }
    public String getPerformance() {
        return performance;
    }
    public void setPerformance(String performance) {
        this.performance = performance;
    }
    public String getRisk() {
        return risk;
    }
    public void setRisk(String risk) {
        this.risk = risk;
    }
}
```

CalendarEvent.java

```java
package com.example.copilot.model;

import java.time.ZonedDateTime;

public class CalendarEvent {
    private String event;
    private ZonedDateTime time;

    // Getters and setters
    public String getEvent() {
        return event;
    }
    public void setEvent(String event) {
        this.event = event;
    }
    public ZonedDateTime getTime() {
        return time;
    }
    public void setTime(ZonedDateTime time) {
        this.time = time;
    }
}
```

SecFiling.java

```
package com.example.copilot.model;

import java.time.LocalDate;

public class SecFiling {
    private String company;
    private LocalDate filingDate;
    private String document;

    // Getters and setters
    public String getCompany() {
        return company;
    }
    public void setCompany(String company) {
        this.company = company;
    }
    public LocalDate getFilingDate() {
        return filingDate;
    }
    public void setFilingDate(LocalDate filingDate) {
        this.filingDate = filingDate;
    }
    public String getDocument() {
        return document;
    }
    public void setDocument(String document) {
        this.document = document;
    }
}
```

ReportingDashboard.java

```
package com.example.copilot.model;

import java.util.Map;

public class ReportingDashboard {
    private String summary;
    private Map<String, String> keyMetrics; // e.g., {"ROI": "10%", "Net Worth": "$500K"}

    // Getters and setters
    public String getSummary() {
        return summary;
    }
    public void setSummary(String summary) {
        this.summary = summary;
    }
    public Map<String, String> getKeyMetrics() {
        return keyMetrics;
    }
    public void setKeyMetrics(Map<String, String> keyMetrics) {
        this.keyMetrics = keyMetrics;
    }
}
```

ScenarioAnalysisRequest.java

```
package com.example.copilot.model;

public class ScenarioAnalysisRequest {
    private String scenario; // e.g., "interest_rate_rise"
    private double rateIncrease; // e.g., 1.0 for 1%

    // Getters and setters
    public String getScenario() {
        return scenario;
    }
    public void setScenario(String scenario) {
        this.scenario = scenario;
    }
    public double getRateIncrease() {
        return rateIncrease;
    }
    public void setRateIncrease(double rateIncrease) {
        this.rateIncrease = rateIncrease;
    }
}
```

ScenarioAnalysisResponse.java

```
package com.example.copilot.model;

import java.util.Map;

public class ScenarioAnalysisResponse {
    private String clientId;
    private Map<String, String> expectedImpact; // e.g., {"fixedIncome": "-1.2%", "overallReturn": "-0.8%"}
    private String recommendation;

    // Getters and setters
    public String getClientId() {
        return clientId;
    }
    public void setClientId(String clientId) {
        this.clientId = clientId;
    }
    public Map<String, String> getExpectedImpact() {
        return expectedImpact;
    }
    public void setExpectedImpact(Map<String, String> expectedImpact) {
        this.expectedImpact = expectedImpact;
    }
    public String getRecommendation() {
        return recommendation;
    }
    public void setRecommendation(String recommendation) {
        this.recommendation = recommendation;
    }
}
```

ReportingRequest.java

```
package com.example.copilot.model;

import java.util.List;
import java.util.Map;

public class ReportingRequest {
    private String clientId;
    private String analysisType; // e.g., "comparative"
    private List<Map<String, Object>> options; // e.g., list of strategies with
parameters

    // Getters and setters
    public String getClientId() {
        return clientId;
    }
    public void setClientId(String clientId) {
        this.clientId = clientId;
    }
    public String getAnalysisType() {
        return analysisType;
    }
    public void setAnalysisType(String analysisType) {
        this.analysisType = analysisType;
    }
    public List<Map<String, Object>> getOptions() {
        return options;
    }
    public void setOptions(List<Map<String, Object>> options) {
        this.options = options;
    }
}
```

ReportingResponse.java

```java
package com.example.copilot.model;

public class ReportingResponse {
    private String clientId;
    private String report;

    // Getters and setters
    public String getClientId() {
        return clientId;
    }
    public void setClientId(String clientId) {
        this.clientId = clientId;
    }
    public String getReport() {
        return report;
    }
    public void setReport(String report) {
        this.report = report;
    }
}
```

EmailRequest.java

```java
package com.example.copilot.model;

public class EmailRequest {
    private String recipient;
    private String subject;
    private String body;

    // Getters and setters
    public String getRecipient() {
        return recipient;
    }
    public void setRecipient(String recipient) {
        this.recipient = recipient;
    }
    public String getSubject() {
        return subject;
    }
    public void setSubject(String subject) {
        this.subject = subject;
    }
    public String getBody() {
        return body;
    }
    public void setBody(String body) {
        this.body = body;
    }
}
```

EmailResponse.java

```java
package com.example.copilot.model;

public class EmailResponse {
    private String status;
    private String message;

    // Getters and setters
    public String getStatus() {
        return status;
    }
    public void setStatus(String status) {
        this.status = status;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

CalendarScheduleRequest.java

```java
package com.example.copilot.model;

import java.time.ZonedDateTime;

public class CalendarScheduleRequest {
    private String advisorId;
    private String clientId;
    private String title;
    private ZonedDateTime startTime;
    private int duration; // in minutes

    // Getters and setters
    public String getAdvisorId() {
        return advisorId;
    }
    public void setAdvisorId(String advisorId) {
        this.advisorId = advisorId;
    }
    public String getClientId() {
        return clientId;
    }
    public void setClientId(String clientId) {
        this.clientId = clientId;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public ZonedDateTime getStartTime() {
        return startTime;
    }
    public void setStartTime(ZonedDateTime startTime) {
        this.startTime = startTime;
    }
    public int getDuration() {
        return duration;
    }
    public void setDuration(int duration) {
        this.duration = duration;
    }
}
```

CalendarScheduleResponse.java

```
package com.example.copilot.model;

public class CalendarScheduleResponse {
    private String status;
    private String eventId;
    private String message;

    // Getters and setters
    public String getStatus() {
        return status;
    }
    public void setStatus(String status) {
        this.status = status;
    }
    public String getEventId() {
        return eventId;
    }
    public void setEventId(String eventId) {
        this.eventId = eventId;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

---

2. Controller Class

Below is a simplified Spring Boot controller that simulates the API endpoints.

CopilotController.java

```
package com.example.copilot.controller;

import com.example.copilot.model.*;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.math.BigDecimal;
import java.time.Instant;
import java.time.LocalDate;
import java.time.ZonedDateTime;
import java.util.*;

@RestController
@RequestMapping("/api")
public class CopilotController {

    // --- Client Profile API ---
    @GetMapping("/clients/{clientId}")
    public ResponseEntity<Client> getClientProfile(@PathVariable String clientId)
{
        Client client = new Client();
        client.setClientId(clientId);
        client.setName("Jane Doe");
        client.setLastMeeting(LocalDate.of(2024, 1, 15));
        client.setPreferences(Map.of("risk", "moderate", "investment", "stocks"));
        return ResponseEntity.ok(client);
    }

    // --- Financial Accounts API ---
    @GetMapping("/clients/{clientId}/accounts")
    public ResponseEntity<List<FinancialAccount>>
getFinancialAccounts(@PathVariable String clientId) {
        List<FinancialAccount> accounts = new ArrayList<>();
        FinancialAccount checking = new FinancialAccount();
        checking.setAccount("Checking");
        checking.setBalance(new BigDecimal("1500.00"));
        FinancialAccount savings = new FinancialAccount();
        savings.setAccount("Savings");
        savings.setBalance(new BigDecimal("5000.00"));
        accounts.add(checking);
        accounts.add(savings);
        return ResponseEntity.ok(accounts);
    }

    // --- Portfolio Analysis API ---
    @GetMapping("/clients/{clientId}/portfolio")
    public ResponseEntity<PortfolioAnalysis> getPortfolioAnalysis(@PathVariable String clientId) {
        PortfolioAnalysis analysis = new PortfolioAnalysis();
        analysis.setClientId(clientId);
        analysis.setPerformance("5.8% YTD");
        analysis.setRisk("Moderate");
        return ResponseEntity.ok(analysis);
    }

    // --- Scenario Analysis API ---
    @PostMapping("/clients/{clientId}/scenario-analysis")
    public ResponseEntity<ScenarioAnalysisResponse> performScenarioAnalysis(@PathVariable String clientId,
                                                                             @RequestBody ScenarioAnalysisRequest request) {
        ScenarioAnalysisResponse response = new ScenarioAnalysisResponse();
        response.setClientId(clientId);
        response.setExpectedImpact(Map.of("fixedIncome", "-1.2%", "overallReturn", "-0.8%"));
        response.setRecommendation("Consider shifting 10% from energy to defensive sectors.");
        return ResponseEntity.ok(response);
    }

    // --- Market Data API ---
    @GetMapping("/market/quotes")
    public ResponseEntity<Map<String, Object>> getMarketQuotes() {
        Map<String, Object> quotes = new HashMap<>();
        quotes.put("AAPL", new BigDecimal("175.23"));
        quotes.put("GOOG", new BigDecimal("2801.50"));
        quotes.put("TSLA", new BigDecimal("720.15"));
        quotes.put("indices", Map.of("S&P500", 4300, "DowJones", 35000));
        return ResponseEntity.ok(quotes);
    }

    // --- Financial News API ---
    @GetMapping("/news/financial")
    public ResponseEntity<List<FinancialNews>> getFinancialNews() {
        List<FinancialNews> newsList = new ArrayList<>();
        FinancialNews news1 = new FinancialNews();
        news1.setHeadline("Healthcare stocks surge amid new policy reforms");
        news1.setSource("Reuters");
        news1.setTimestamp(Instant.now());
        FinancialNews news2 = new FinancialNews();
        news2.setHeadline("Energy sector faces volatility due to geopolitical tensions");
        news2.setSource("Bloomberg");
        news2.setTimestamp(Instant.now());
        newsList.add(news1);
        newsList.add(news2);
        return ResponseEntity.ok(newsList);
    }

    // --- Reporting API ---
    @PostMapping("/reports/generate")
    public ResponseEntity<ReportingResponse> generateReport(@RequestBody
ReportingRequest request) {
        ReportingResponse response = new ReportingResponse();
        response.setClientId(request.getClientId());
        response.setReport("Detailed comparative analysis report generated. [See
attached details]");
        return ResponseEntity.ok(response);
    }

    // --- Email API ---
    @PostMapping("/email/draft")
    public ResponseEntity<EmailResponse> draftEmail(@RequestBody EmailRequest
request) {
        EmailResponse response = new EmailResponse();
        response.setStatus("success");
        response.setMessage("Email sent successfully to " +
request.getRecipient());
        return ResponseEntity.ok(response);
    }

    // --- Calendar / Scheduling API ---
    @PostMapping("/calendar/schedule")
    public ResponseEntity<CalendarScheduleResponse> scheduleMeeting(@RequestBody
CalendarScheduleRequest request) {
        CalendarScheduleResponse response = new CalendarScheduleResponse();
        response.setStatus("success");
        response.setEventId(UUID.randomUUID().toString());
        response.setMessage("Meeting scheduled successfully for " +
request.getTitle());
        return ResponseEntity.ok(response);
    }

    // --- Regulatory and Compliance API ---
    @GetMapping("/regulatory/updates")
    public ResponseEntity<List<SecFiling>> getRegulatoryUpdates() {
        List<SecFiling> filings = new ArrayList<>();
        SecFiling filing1 = new SecFiling();
        filing1.setCompany("XYZ Inc");
        filing1.setFilingDate(LocalDate.of(2024, 1, 22));
        filing1.setDocument("10-Q");
        SecFiling filing2 = new SecFiling();
        filing2.setCompany("ABC Corp");
        filing2.setFilingDate(LocalDate.of(2024, 1, 20));
        filing2.setDocument("10-K");
        filings.add(filing1);
        filings.add(filing2);
        return ResponseEntity.ok(filings);
    }
    // --- Reporting Dashboard API ---
    @GetMapping("/reporting/dashboard")
    public ResponseEntity<ReportingDashboard> getDashboard() {
        ReportingDashboard dashboard = new ReportingDashboard();
        dashboard.setSummary("All systems nominal");
        dashboard.setKeyMetrics(Map.of("ROI", "10%", "Net Worth", "$500K"));
        return ResponseEntity.ok(dashboard);
    }
}
```

---

3. OpenAPI Specification (YAML)

Below is an example OpenAPI 3.0 YAML specification for these endpoints. Save this as, for example, `openapi.yaml`.

```yaml
openapi: 3.0.3
info:
  title: Copilot API
  version: "1.0.0"
  description: API endpoints for an AI copilot assisting financial advisors.
servers:
  - url: http://localhost:8080
paths:
  /api/clients/{clientId}:
    get:
      summary: Get client profile
      parameters:
        - in: path
          name: clientId
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Client profile
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Client'
  /api/clients/{clientId}/accounts:
    get:
      summary: Get financial accounts for a client
      parameters:
        - in: path
          name: clientId
          required: true
          schema:
            type: string
      responses:
        '200':
          description: List of financial accounts
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/FinancialAccount'
  /api/clients/{clientId}/portfolio:
    get:
      summary: Get portfolio analysis for a client
      parameters:
        - in: path
          name: clientId
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Portfolio analysis
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/PortfolioAnalysis'
  /api/clients/{clientId}/scenario-analysis:
    post:
      summary: Perform scenario analysis on a client's portfolio
      parameters:
        - in: path
          name: clientId
          required: true
          schema:
            type: string
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ScenarioAnalysisRequest'
      responses:
        '200':
          description: Scenario analysis result
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ScenarioAnalysisResponse'
  /api/market/quotes:
    get:
      summary: Get market quotes and indices
      responses:
        '200':
          description: Market data
          content:
            application/json:
                schema:
                  type: object
  /api/news/financial:
    get:
      summary: Get financial news headlines
      responses:
        '200':
          description: List of financial news
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/FinancialNews'
  /api/reports/generate:
    post:
      summary: Generate a comparative analysis report
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ReportingRequest'
      responses:
        '200':
          description: Report generated
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ReportingResponse'
  /api/email/draft:
    post:
      summary: Draft and send an email
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/EmailRequest'
      responses:
        '200':
          description: Email response
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/EmailResponse'
  /api/calendar/schedule:
    post:
      summary: Schedule a meeting
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CalendarScheduleRequest'
      responses:
        '200':
          description: Meeting scheduled
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CalendarScheduleResponse'
  /api/regulatory/updates:
    get:
      summary: Get regulatory and compliance updates
      responses:
        '200':
          description: List of SEC filings and regulatory updates
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/SecFiling'
  /api/reporting/dashboard:
    get:
      summary: Get the reporting dashboard summary
      responses:
        '200':
          description: Dashboard summary
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ReportingDashboard'
components:
  schemas:
    Client:
      type: object
      properties:
        clientId:
          type: string
        name:
          type: string
        lastMeeting:
          type: string
          format: date
        preferences:
          type: object
          additionalProperties:
            type: string
    FinancialAccount:
      type: object
      properties:
        account:
          type: string
        balance:
          type: number
          format: double
    PortfolioAnalysis:
      type: object
      properties:
        clientId:
          type: string
        performance:
          type: string
        risk:
          type: string
    ScenarioAnalysisRequest:
      type: object
      properties:
        scenario:
          type: string
        rateIncrease:
          type: number
          format: double
    ScenarioAnalysisResponse:
      type: object
      properties:
        clientId:
          type: string
        expectedImpact:
          type: object
          additionalProperties:
            type: string
        recommendation:
          type: string
    MarketQuote:
      type: object
      properties:
        ticker:
          type: string
        price:
          type: number
          format: double
    FinancialNews:
      type: object
      properties:
        headline:
          type: string
        source:
          type: string
        timestamp:
          type: string
          format: date-time
    ReportingRequest:
      type: object
      properties:
        clientId:
          type: string
        analysisType:
          type: string
        options:
          type: array
          items:
            type: object
    ReportingResponse:
      type: object
      properties:
        clientId:
          type: string
        report:
          type: string
    EmailRequest:
      type: object
      properties:
        recipient:
          type: string
        subject:
          type: string
        body:
          type: string
    EmailResponse:
      type: object
      properties:
        status:
          type: string
        message:
          type: string
    CalendarScheduleRequest:
      type: object
      properties:
        advisorId:
          type: string
        clientId:
          type: string
        title:
          type: string
        startTime:
          type: string
          format: date-time
        duration:
          type: integer
    CalendarScheduleResponse:
      type: object
      properties:
        status:
          type: string
        eventId:
          type: string
        message:
          type: string
    SecFiling:
      type: object
      properties:
        company:
          type: string
        filingDate:
          type: string
          format: date
        document:
          type: string
    ReportingDashboard:
      type: object
      properties:
        summary:
          type: string
        keyMetrics:
          type: object
          additionalProperties:
            type: string
```

Summary

- **Model Classes:** Define the data models (Client, FinancialAccount, etc.) used by the APIs.
- **Controller:** The `CopilotController` class provides endpoints to simulate the functionality for client data, portfolio analysis, scenario analysis, market data, news, report generation, email, calendar scheduling, regulatory updates, and dashboard reporting.
- **OpenAPI Spec:** The provided YAML document describes these endpoints, their request/response bodies, and the underlying data models in OpenAPI 3.0 format.

This setup can be imported into a Spring Boot project to simulate the full API stack powering a killer conversational copilot for financial advisors.