**Technical Specification: SchemaBot**  
*(Natural Language to API Execution Engine)*

**1. System Overview**

SchemaBot is a **modular monolith** that enables users to interact with REST/GraphQL APIs via natural language. It decouples API configuration management from execution logic, leveraging an **event-driven architecture** with four core components:

1. **WebUI** (Frontend)
2. **Settings.Api** (Configuration Backend)
3. **Conversational Agent** (AI-Driven Query Generator)
4. **API Executor**

**2. Component Specifications**

**Component 1: WebUI (Blazor Frontend)**

* **Tech Stack**:
  + .NET 9, Blazor Server with Radzen Components, Bootstrap 5.
* **Features**:
  + **Configuration Management**:
    - Integrates with **Settings.Api** for CRUD operations (no direct DB access).
    - Schema upload via drag-and-drop (swagger.json/GraphQL).
  + **Chat Interface**:
    - Natural language input (e.g., "Show active users").
    - Response rendering: Tables (Radzen DataGrid), raw JSON/GraphQL viewer.
  + **Event Integration**:
    - Publishes UserQueryEvent (user input + ApiConfigId).
    - Subscribes to APIResultEvent for real-time updates.

**Component 2: Settings.Api (Configuration Backend)**

* **Tech Stack**:
  + .NET 9 REST API (Minimal APIs), EF Core, PostgreSQL.
* **Features**:
  + **CRUD Operations**:
    - Store/retrieve API schemas, context prompts, and auth configurations.
    - Schema validation (e.g., Swagger 3.0 compliance).
  + **Security**:
    - AES-256 encryption for API keys/tokens.
    - Role-based access control (RBAC) for configuration edits.
  + **Endpoints**:
    - GET /api/configurations/{id} → Returns schema + context prompts.
    - POST /api/context-prompts → Attach usage hints to endpoints.

**Component 3: Conversational Agent (Semantic Kernel Service)**

* **Tech Stack**:
  + .NET 9 Background Service, Semantic Kernel SDK, Ollama API Client.
* **Workflow**:
  1. Subscribe to UserQueryEvent.
  2. Fetch schema/context from **Settings.Api** using ApiConfigId.
  3. Build **augmented prompt**:

Copy

User Query: "Get total pets"

Schema: { endpoints: [{ path: "/pets/count", method: "GET" }] }

Context: "Always use /pets/count for pet totals."

Output Format: JSON with endpoint, method, parameters.

* 1. Generate API command via Ollama (DeepSeek model).
  2. Publish validated APICommandEvent.

**Component 4: API Executor**

* **Tech Stack**:
  + .NET 9 Background Service, HttpClient, GraphQL.Client.
* **Functionality**:
  + Execute REST/GraphQL requests from APICommandEvent.
  + Handle auth (OAuth2, API keys), retries, and HTTPS security.
  + Return standardized APIResultEvent:

json

Copy

{

"data": {},

"statusCode": 200,

"correlationId": "abc123",

"error": null

}

**3. Architecture**

**Event Broker & Communication**

* **Events**:

| **Event Name** | **Payload** |
| --- | --- |
| UserQueryEvent | { CorrelationId, UserId, QueryText, ApiConfigId } |
| APICommandEvent | { CorrelationId, Endpoint, Method, Parameters, Body } |
| APIResultEvent | { CorrelationId, Data, StatusCode, Error } |
| SchemaUpdatedEvent | { ApiConfigId, Timestamp } (Triggers agent cache refresh) |

* **Concurrency**:
  + Thread-safe event processing with System.Threading.Channels.
  + Parallel execution of API calls in **API Executor**.

**Solution Structure**

Copy

SchemaBot/

├── SchemaBot.WebUI/ # Blazor frontend

├── SchemaBot.Settings.Api/ # Configuration REST API

├── SchemaBot.Agent/ # Conversational Agent (Semantic Kernel)

├── SchemaBot.Executor/ # API Execution Engine

├── SchemaBot.Core/ # Shared models, events, interfaces

└── SchemaBot.Infrastructure/ # Event Broker, DB integrations

**Key Interfaces**

1. **IApiConfigurationService** (Settings.Api):

csharp

Copy

Task<ApiConfig> GetConfigurationAsync(Guid apiConfigId);

Task UpdateSchemaAsync(Guid id, string schemaJson);

1. **IEventBroker** (Infrastructure):

csharp

Copy

void Publish<TEvent>(TEvent @event);

IObservable<TEvent> Subscribe<TEvent>();

**4. Data Flow**

mermaid

Copy

sequenceDiagram

participant User

participant WebUI

participant Settings.Api

participant Agent

participant Executor

participant ExternalAPI

User->>WebUI: "List users with orders"

WebUI->>Settings.Api: GET /api/configurations/123

Settings.Api-->>WebUI: Schema + Context

WebUI->>Agent: Publish UserQueryEvent

Agent->>Ollama: Send augmented prompt

Ollama-->>Agent: { endpoint: "/users", method: "GET" }

Agent->>Executor: Publish APICommandEvent

Executor->>ExternalAPI: GET /users?hasOrders=true

ExternalAPI-->>Executor: Response

Executor->>WebUI: Publish APIResultEvent

WebUI->>User: Render table

**5. Non-Functional Requirements**

* **Performance**:
  + Ollama response time < 3s (local GPU/CPU optimization).
  + Schema loading from **Settings.Api** < 500ms (caching with Redis optional).
* **Security**:
  + All credentials encrypted in transit (HTTPS) and at rest (AES-256).
  + Certificate pinning for external API calls.
* **Error Handling**:
  + Retry failed API calls (exponential backoff).
  + Model response validation with JSON Schema.

**6. Next Steps**

1. Implement **Settings.Api** schema validation middleware.
2. Add caching layer in **Conversational Agent** for frequently used schemas.
3. Develop RBAC policies in **Settings.Api** (e.g., Admin/Editor roles).

SchemaBot bridges natural language and API execution while maintaining enterprise-grade security and scalability. Ready to proceed with implementation? 🔍🚀