Ministry Regulation Q&A System: Project Plan

Introduction

The **Ministry Regulation Q&A System** is a cutting-edge platform designed to provide accurate, accessible, and ministry-worthy answers to questions about regulatory policies, specifically targeting Arabic-speaking users. By employing **Retrieval-Augmented Generation (RAG)** technology, the system processes official 2024 ministry regulation documents to deliver concise, contextually grounded responses in Arabic. This project serves students, educators, and the general public, simplifying access to complex regulatory information while maintaining high standards of quality, usability, and scalability.

Project Objectives

- Deliver a fully functional prototype by **April 16, 2025**, utilizing 2024 ministry regulation documents as the primary knowledge base.
- Develop a robust, scalable, and secure backend using Python technologies that adhere to Software Engineering (SE) best practices and SOLID principles.
- Create an intuitive, Arabic right-to-left (RTL) compatible frontend with a seamless user experience.
- Ensure responses are accurate, relevant, and derived from official sources, meeting ministry standards.
- Foster a collaborative team environment where all five members contribute equally to a cohesive deliverable.

Team Members

- **Ainouche Abderahmane**
- **Moulahcene Riadh**
- **Zamoum Abdelhakim**
- **Guerroudj Abdennour**
- **Mestouri Oussama**

Technology Stack

The technology stack is carefully selected to balance performance, scalability, Arabic language support, and flexibility for experimentation. Versions reflect the latest stable releases as of March 21, 2025,

sourced from official repositories (e.g., PyPI, npm). Each technology's purpose and rationale are explained thoroughly to justify its inclusion.

Backend Technologies

Technology/Package	Version	Purpose	Rationale	Documentation Link
FastAPI	0.115.0	High-performance, asynchronous API framework	Chosen for its speed (async support), automatic OpenAPI generation, and ease of scaling with Python's ecosystem. Ideal for real-time Q&A responses and adheres to SOLID's Dependency Inversion by supporting dependency injection.	FastAPI Docs
LangChain	0.3.1	Framework for building RAG pipelines	Provides a modular, extensible framework for RAG, supporting multiple LLMs and vector stores. Its abstraction layers align with SOLID's Open/Closed principle, allowing easy LLM swapping and scalability through chaining components.	LangChain Docs
LangGraph	0.2.23	Advanced workflows for complex query handling	Extends LangChain with graph-based workflows, enabling multi-step reasoning (e.g., follow-up questions). Chosen for flexibility and scalability, supporting SOLID's Single Responsibility by isolating workflow logic.	LangGraph Docs
LangChain-Groq	0.2.0	Integration with Groq's LLMs	Facilitates experimentation with Groq LLMs (e.g., Llama 3 70B). Its modular design allows seamless swapping with other LLM providers (e.g., Google), supporting SOLID's Open/Closed principle.	LangChain-Groq Docs
Sentence Transformers	3.1.1	Multilingual embeddings for Arabic text	Offers robust multilingual embeddings, critical for Arabic document retrieval in RAG. Chosen for its proven performance with Arabic (e.g., `paraphrase-multilingual-MiniLM-L12-v2`), ensuring accurate similarity search and scalability in vector storage.	Hugging Face Docs

Technology/Package	Version	Purpose	Rationale	Documentation Link
Supabase Python Client	2.7.4	Database and vector storage management	Provides a Python interface to Supabase, enabling seamless integration with PostgreSQL and pgvector. Chosen for its simplicity, real-time capabilities, and scalability, aligning with SOLID's Dependency Inversion via abstracted DB access.	Supabase Python Docs
Pydantic	2.9.2	Data validation and settings management	Ensures type safety and configuration management (e.g., API payloads, env vars). Chosen for its robust validation, supporting SOLID's Single Responsibility by isolating data modeling from business logic, and scalability through structured data handling.	Pydantic Docs
python-dotenv	1.0.1	Environment variable management	Manages sensitive configuration (e.g., API keys) securely outside code. Chosen for its simplicity and alignment with SE best practices, ensuring flexibility and scalability by decoupling config from implementation.	python-dotenv Docs
Uvicorn	0.30.6	ASGI server for running FastAPI	Lightweight, high-performance server for FastAPI. Chosen for its async capabilities, ensuring scalability under high load, and compatibility with Python's ecosystem, supporting SOLID's Dependency Inversion via configurable deployment.	Uvicorn Docs

Frontend Technologies

Technology/Package	Version	Purpose	Rationale	Documentation Link

Technology/Package	Version	Purpose	Rationale	Documentation Link
Next.js	14.2.14	React framework for building the UI	Provides server-side rendering, static site generation, and a robust ecosystem for UI development. Chosen for its scalability, ease of deployment (e.g., Vercel), and support for RTL styling, critical for Arabic usability.	Next.js Docs
npx create-agent-chat-app	-	Template for chat application UI	A LangChain-provided template for rapid chat UI prototyping. Chosen to accelerate frontend development while ensuring compatibility with LangChain's backend, supporting scalability and team collaboration on UI customization.	LangChain Chat App
Supabase JS Client	2.45.4	Authentication and chat history management	Enables frontend integration with Supabase for user auth and real-time data. Chosen for its simplicity, real-time updates, and scalability, aligning with SOLID's Dependency Inversion by abstracting auth logic.	Supabase JS Docs

Database Technologies

Technology/Package	Version	Purpose	Rationale	Documentation Link
Supabase	-	PostgreSQL database with pgvector extension	Open-source, scalable alternative to Firebase with PostgreSQL backend. Chosen for its vector search (pgvector), real-time capabilities, and built-in auth, ensuring scalability and SOLID's Single Responsibility by isolating data storage.	Supabase Docs

Technology/Package	Version	Purpose	Rationale	Documentation Link
pgvector	0.7.4	Vector similarity search for RAG	Extends PostgreSQL with vector operations for efficient similarity search in RAG. Chosen for its performance, integration with Supabase, and scalability, supporting SOLID's Open/Closed principle by allowing vector store enhancements.	pgvector GitHub

Large Language Model (LLM) Options

Model Options	Provider	Purpose	Rationale	Documentation Link
Groq LLMs (e.g., Llama 3 70B)	Groq	Fast, multilingual LLMs for experimentation	Groq offers high-speed inference, ideal for real-time Q&A. Llama 3 70B is a starting point due to its multilingual capabilities, but we'll experiment with others for Arabic performance. Chosen for speed and flexibility.	Groq Docs
Google Gemini 2.0	Google	Alternative LLM for experimentation	Google's latest model (assumed available by 2025) may offer superior Arabic support. Chosen for its potential robustness and integration with Google's ecosystem, ensuring flexibility in LLM swapping.	Google AI Docs
Others (TBD)	Various	Additional LLMs for testing	We'll explore other providers (e.g., Hugging Face models) to compare performance. Flexibility is key, aligning with SOLID's Open/Closed principle for seamless LLM substitution.	TBD based on experimentation

LLM Flexibility

- **Why Not Definitive**: We will experiment with multiple LLMs (Groq's Llama 3 70B, Google Gemini 2.0, etc.) to determine the best fit for Arabic Q&A. The backend will use an abstraction layer (e.g., LangChain's LLM interface) to swap LLMs seamlessly by updating configuration (e.g., API keys, model names) without altering core logic, adhering to SOLID's Open/Closed and Dependency Inversion principles.

Project Architecture

The system comprises three interconnected components designed for scalability and modularity:

Workflow

1. **User Interaction**: Users input Arabic queries via the Next.js frontend chat interface.
2. **API Communication**: Frontend sends HTTP requests to FastAPI endpoints.
3. **RAG Processing**: LangChain retrieves relevant document chunks from Supabase, and the selected LLM generates answers.
4. **Response Delivery**: Answers are returned to the frontend, displayed with Arabic RTL support.
5. **Data Persistence**: Queries, responses, and user data are stored in Supabase for history and auditing.

Backend Components

- **Document Ingestion**:
  - Processes ministry documents, splits them into chunks, and generates embeddings for vector search.
  - Stores data in Supabase with pgvector for efficient retrieval.
- **RAG Pipeline**:
  - Retrieves document chunks based on query embeddings and generates responses using a configurable LLM.
  - Supports multi-step workflows via LangGraph for complex queries.
- **API**:
  - Exposes endpoints (`/qa` for queries, `/history` for chat logs) with Supabase JWT authentication for security.

Frontend Components

- **Chat Interface**:
  - Built with Next.js and customized from the "npx create-agent-chat-app" template.
  - Ensures real-time interaction and Arabic RTL compatibility.
- **Authentication**:
  - Integrates Supabase JS client for user login, registration, and session management.

Database Schema

Table	Columns	Purpose
`users`	`id` (UUID), `email` (string), `created_at` (timestamp)	Manages user accounts
`chat_sessions`	`id` (UUID), `user_id` (UUID), `timestamp` (timestamp)	Tracks user chat sessions

Table	Columns	Purpose
`messages`	`id` (UUID), `session_id` (UUID), `query` (text), `response` (text), `timestamp` (timestamp)	Stores chat history
`documents`	`id` (UUID), `content` (text), `metadata` (JSON)	Holds document chunks
`embeddings`	`id` (UUID), `vector` (vector), `document_id` (UUID)	Stores vector embeddings

Scalability Features

- **Backend**: Async FastAPI and modular LangChain components handle high concurrency.
- **Database**: Supabase's PostgreSQL with pgvector scales with user load and vector data.
- **Frontend**: Next.js supports server-side rendering and static generation for performance.

Development Plan

Each phase includes specific tasks and deliverables.

Phase 1: Database Setup

- **Tasks**: Set up Supabase project, enable pgvector, create and test database schema.
- **Deliverables**: Fully operational Supabase instance with all tables configured.

Phase 2: Document Ingestion

- **Tasks**: Collect 2024 ministry documents, preprocess them, implement ingestion pipeline, store embeddings in Supabase.
- **Deliverables**: Ingested documents ready for RAG processing.

Phase 3: RAG Pipeline

- **Tasks**: Develop RAG system, integrate initial LLM (e.g., Groq's Llama 3 70B), test with Arabic queries, experiment with alternative LLMs (e.g., Gemini 2.0).
- **Deliverables**: Working RAG pipeline with flexible LLM configuration.

Phase 4: Backend API

- **Tasks**: Build FastAPI server, implement endpoints, secure with JWT, add logging and error handling.
- **Deliverables**: Secure, functional API integrated with RAG.

Phase 5: Frontend Development

- **Tasks**: Set up Next.js, customize chat template, integrate with backend and Supabase, style for Arabic RTL.
- **Deliverables**: Fully functional chat UI.

Phase 6: Testing and Deployment

- **Tasks**: Write comprehensive tests, deploy backend and frontend to cloud platforms, optimize performance.
- **Deliverables**: Deployed, tested prototype.

Phase 7: Final Review and Buffer

- **Tasks**: Conduct final testing, refine based on feedback, prepare documentation and submission.
- **Deliverables**: Polished prototype ready for April 16, 2025.

Team Responsibilities

Work is divided equally among five members, with specific primary roles and shared frontend contributions. Each role aligns with SOLID principles for modularity and scalability.

Ainouche Abderahmane: RAG Pipeline Lead

- **Primary Responsibilities**:
  - Design and implement the RAG pipeline using LangChain and LangGraph.
  - Integrate initial LLM (Groq's Llama 3 70B) and create an abstraction layer for swapping LLMs (e.g., Google Gemini 2.0).
  - Test Arabic query performance and refine retrieval/generation logic.
  - Experiment with multiple LLMs, documenting performance metrics (e.g., speed, accuracy).
- **Effort**: 2 weeks development, 1 week testing and experimentation.
- **Frontend Contribution**: Collaborate on integrating RAG responses into the Next.js UI, ensuring smooth data flow.

Moulahcene Riadh: Database and Ingestion Lead

- **Primary Responsibilities**:
  - Set up the Supabase project, enable pgvector, and configure all database tables.
  - Collect and preprocess 2024 ministry documents (e.g., PDFs, text files).
  - Implement the document ingestion pipeline, ensuring embeddings are generated and stored efficiently.
  - Test database performance under load to ensure scalability.
- **Effort**: 1 week database setup, 2 weeks ingestion development.
- **Frontend Contribution**: Assist with Supabase JS client integration for chat history and authentication.

Zamoum Abdelhakim: Backend API Lead

- **Primary Responsibilities**:
  - Develop the FastAPI server, implementing `/qa` and `/history` endpoints.
  - Secure endpoints with Supabase JWT authentication, ensuring user data protection.
  - Add comprehensive logging and error handling for debugging and scalability.
  - Test API performance with simulated user traffic.
- **Effort**: 2 weeks API development, 1 week testing and refinement.
- **Frontend Contribution**: Ensure API endpoints align with frontend requirements, assisting with API call integration.

Mestouri Oussama: Testing and Deployment Lead

- **Primary Responsibilities**:
  - Develop a testing strategy, writing unit and integration tests for backend and frontend.
  - Deploy the backend to a cloud platform (e.g., Railway) and frontend to Vercel.
  - Monitor and optimize deployed system performance (e.g., response time, scalability).
  - Document deployment process for team reference.
- **Effort**: 1 week testing setup, 2 weeks deployment and optimization.
- **Frontend Contribution**: Test UI functionality, responsiveness, and RTL rendering.

Guerroudj Abdennour: Frontend Development Lead

- **Primary Responsibilities**:
  - Set up the Next.js project using "npx create-agent-chat-app" as a base.
  - Customize the chat interface, integrating with FastAPI and Supabase JS client.
  - Lead RTL styling and Arabic usability enhancements, ensuring a ministry-worthy UI.
  - Coordinate team frontend efforts for consistency.
- **Effort**: 1 week setup, 2 weeks integration and styling.
- **Frontend Contribution**: Oversee UI development, ensuring all team inputs are cohesive.

Shared Frontend Responsibilities

- **All Members**:
  - Contribute to Next.js frontend development, including chat components, authentication flows, and RTL styling.
  - Participate in weekly UI sync meetings to integrate backend components and test usability.
  - Review and refine the UI for a seamless Arabic user experience.

Development Instructions

These instructions are explicit, removing ambiguity, and tailored for each member on Windows.

Ainouche Abderahmane: RAG Pipeline Lead

- **Steps**:
  1. Ensure LangChain tools are installed: Open Command Prompt in `backend/`, run `pip list` to verify `langchain`, `langchain-groq`, `langchain-community`.
  2. Obtain API keys for Groq (from groq.com) and Google Gemini (from ai.google)—store in `backend/.env` (e.g., `GROQ_API_KEY=xxx`, `GOOGLE_API_KEY=yyy`).
  3. Develop the RAG pipeline in `backend/src/core/rag.py`, creating an LLM abstraction layer (e.g., config file or class) to switch between Groq, Google, and other LLMs by changing keys/model names.
  4. Test with Arabic queries: Run pipeline manually with sample questions (e.g., "ما هي اللوائح الجديدة لعام 2024؟") and log performance metrics (speed, accuracy).
  5. Experiment with at least three LLMs (e.g., Llama 3 70B, Gemini 2.0, one other), documenting results in a shared doc.
  6. Collaborate on UI: In `frontend/pages/`, integrate RAG responses into the chat component, testing real-time display.
- **Tools**: Python, VS Code, Command Prompt, Groq/Google APIs.

Moulahcene Riadh: Database and Ingestion Lead

- **Steps**:
  1. Create a Supabase project: Visit supabase.com, sign up, create a new project, note the URL and anon key, store in `backend/.env` (e.g., `SUPABASE_URL=xxx`, `SUPABASE_KEY=yyy`).
  2. Enable pgvector: In Supabase dashboard, go to SQL Editor, run `CREATE EXTENSION pgvector;`, verify with `SELECT * FROM pg_extension;`.

3. Set up tables: Use Supabase dashboard's Table Editor to create `users`, `chat_sessions`, `messages`, `documents`, `embeddings` with specified columns, or run SQL scripts provided in team docs.
4. Collect 2024 ministry documents (e.g., PDFs from ministry sources), convert to text if needed (manual or via tools like Adobe Acrobat), save in `backend/data/`.
5. Develop ingestion pipeline in `backend/src/core/ingestion.py`, testing with a sample document to ensure chunks and embeddings are stored in Supabase.
6. Test database scalability: Insert 100+ sample records, query with `SELECT * FROM embeddings LIMIT 10;` in Supabase SQL Editor to verify.
7. Assist with UI: In `frontend/lib/supabase.js`, configure Supabase JS client with your project's URL and key, test chat history retrieval.

- **Tools**: Supabase dashboard, Python, VS Code, Command Prompt.

Zamoum Abdelhakim: Backend API Lead

- **Steps**:
  1. Verify FastAPI setup: In `backend/`, run `pip list` to confirm `fastapi`, `uvicorn` are installed.
  2. Develop API in `backend/src/routers/api.py` and `backend/src/main.py`, implementing `/qa` (POST) and `/history` (GET) endpoints.
  3. Secure with Supabase JWT: Use Supabase auth token from frontend, validate in API using Supabase Python client's auth methods.
  4. Add logging: Configure Python's `logging` module to log requests/errors to `backend/logs/api.log`.
  5. Test locally: Run `uvicorn src.main:app --reload` in `backend/`, use a tool like Postman to send sample requests (e.g., POST `http://localhost:8000/qa` with `{"query": "ما هي القوانين؟"}`).
  6. Collaborate on UI: In `frontend/api/`, ensure API calls match endpoint specs, test with sample queries.
- **Tools**: FastAPI, Postman, VS Code, Command Prompt.

Mestouri Oussama: Testing and Deployment Lead

- **Steps**:
  1. Install testing tools: In `backend/`, run `pip install pytest` to confirm installation.
  2. Develop tests: Create `backend/tests/test_rag.py`, `test_api.py`, etc., covering RAG, API, and ingestion (e.g., mock Supabase responses, test LLM outputs).
  3. Set up Railway: Sign up at railway.app, link Git repo, configure `backend/` deployment with `Procfile` (e.g., `web: uvicorn src.main:app --host 0.0.0.0 --port`

`$PORT`).

4. Set up Vercel: Sign up at vercel.com, link `frontend/` repo, run `vercel --prod` in `frontend/` to deploy.
5. Test deployed system: Use Postman or browser to hit deployed endpoints (e.g., `<railway-url>/qa`), verify UI at `<vercel-url>`.
6. Optimize: Monitor Railway logs, adjust Supabase indexing if slow, document findings.
7. Test UI: In `frontend/`, verify chat functionality and RTL rendering post-deployment.

- **Tools**: Pytest, Railway CLI, Vercel CLI, VS Code, Command Prompt.

Guerroudj Abdennour: Frontend Development Lead

- **Steps**:
  1. Verify Node.js: Run `node --version` to confirm 14+.
  2. Set up frontend: In `frontend/`, run `npx create-agent-chat-app`, select default options, then `npm install` to set up.
  3. Configure Supabase: In `frontend/lib/supabase.js`, add Supabase URL and key from Moulahcene's setup, test auth with `npm run dev`.
  4. Integrate API: In `frontend/api/`, create functions to call Zamoum's endpoints (e.g., `fetch('/qa', { method: 'POST', body: JSON.stringify({ query }) })`).
  5. Style RTL: In `frontend/styles/global.css`, add `body { direction: rtl; font-family: Amiri; }`, test with Arabic text.
  6. Lead UI sync: Schedule weekly meetings, assign UI tasks (e.g., chat input to Ainouche, history to Moulahcene), review progress.
  7. Test locally: Run `npm run dev` in `frontend/`, visit `http://localhost:3000` to verify.
- **Tools**: Next.js, npm, VS Code, Command Prompt.

Resources and Documentation

Official Documentation

- **FastAPI**: Comprehensive guide for API development.
- **LangChain**: RAG pipeline and LLM integration details.
- **LangGraph**: Workflow design and examples.
- **Supabase**: Database setup, auth, and real-time features.
- **Next.js**: Frontend development and deployment.
- **Groq**: LLM access and configuration.

Course Resources (Assumed from Abdelhakim Cheriet's Course)

- **AI and NLP**: Concepts of RAG, embeddings, and LLMs for Arabic processing.
- **Web Development**: RESTful API design and React fundamentals.
- **Databases**: PostgreSQL basics and vector storage techniques.

Online Resources

- **Hugging Face Sentence Transformers**: Latest embedding models for Arabic.
- **pgvector GitHub**: Vector search optimization tips.
- **Arabic NLP Research (ArabicaQA)**: Dataset and benchmarks for Arabic Q&A testing.

Scalability and SOLID Adherence

- **Scalability**:
  - Backend uses async FastAPI and Supabase's real-time features to handle high user loads.
  - Database leverages pgvector indexing for efficient vector searches.
  - Frontend employs Next.js's server-side rendering for performance.
- **SOLID Principles**:
  - **Single Responsibility**: Each component (ingestion, RAG, API) has one clear purpose.
  - **Open/Closed**: LLM abstraction allows extension without modifying core logic.
  - **Liskov Substitution**: Future LLM or DB clients can replace current ones seamlessly.
  - **Interface Segregation**: API and DB clients expose only necessary methods.
  - **Dependency Inversion**: Dependencies (e.g., Supabase client, LLM config) are injected, not hardcoded.

Challenges and Solutions

- **Challenge**: Variable Arabic LLM performance.
  - **Solution**: Experiment with multiple LLMs, refine embeddings, and document results.
- **Challenge**: High query latency.
  - **Solution**: Optimize pgvector indexing, cache frequent queries, and scale backend with cloud resources.
- **Challenge**: Team coordination.
  - **Solution**: Weekly syncs, shared docs, and clear task ownership.

Conclusion

This detailed plan ensures a scalable, ministry-worthy Q&A system by April 16, 2025. With explicit roles, flexible LLM experimentation, and adherence to SOLID principles, Ainouche, Moulahcene, Zamoum, Guerroudj, and Mestouri will deliver a high-quality prototype under Abdelhakim Cheriet's guidance.

Team Commitment: We pledge to collaborate, innovate, and exceed expectations.