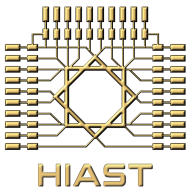
الجمهورية العربية السورية
المعهد العالي للعلوم التطبيقية والتكنولوجيا
قسم المعلوميات



بناء موقع ويب للمعهد العالي للعلوم التطبيقية والتكنولوجيا مدعوم بتشات بوت ذكي

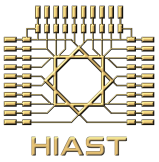
تقرير مشروع سنة رابعة
اختصاص هندسة البرمجيات والذكاء صنعي

إعداد
علاء زريقي

إشراف
م. شادي بلول

2025/08/13

Syrian Arab Republic
Higher Institute for Applied Sciences and Technology
Department of Information Systems



Design and Implementation of BOT for a web page

by

Alaa Zureiki

Fourth Year Project Report
Specialization in Software Engineering and Artificial Intelligence

Under supervision of

Eng. Shadi Ballol

13/8/2025

كلمة شكر

أتوجه بالشكر الجزيل للمشرف القائم على المشروع المهندس شادي بلول على دعمه المستمر وتوجيهاته القيمة طوال فترة تنفيذ هذا المشروع، لقد كان لحرصه دور كبير في تحقيق هذا الإنجاز، حيث لم يدخر جهدًا في تقديم النصائح والإرشادات التي ساهمت في تحسين جودة العمل .

كما أشكر المخبريين العاملين في قسم النظم المعلوماتية على مساعداتهم القيّمة وتعاونهم الدائم، وتقديمهم المساعدة.

إهداء

إلى من كانوا لي السند والدعم في كل خطوة خطوتها، إلى من غمروني بحبهم وحنانهم، إلى أبي وأمي...

إلى أخي وأختي، شركاء الطفولة وأصدقاء العمر...

إلى رفاقي الأعزاء، الذين كانوا لي العون والسند في كل لحظة، والذين شاركوني الأفراح والأحزان...

فلكم مني أسمى آيات الشكر والتقدير

الملخَّص

بعد التطور الكبير في النماذج اللغوية الكبيرة (LLM) وتقنية الاسترجاع المعزز بالتوليد (RAG) برزت فرصة مهمة في مجال تطوير مواقع الويب وتعزيز قدراتها. تُعد هذه التقنيات أدوات قوية لتحويل مواقع الويب إلى منصات تفاعلية ذكية، قادرة على تقديم خدمة عملاء استثنائية والإجابة الدقيقة والسريعة على استفسارات المستخدمين. وانطلاقاً من هذه الأهمية، يهدف هذا المشروع إلى بناء نسخة محدثة لموقع الويب الخاص بالمعهد العالي للعلوم التطبيقية والتكنولوجيا وتضمين تشات بوت (Chat Bot) ذكي يعتمد على هذه التقنيات. سيساهم هذا في توفير تجربة مستخدم محسنة وفعالة لزوار الموقع من خلال تقديم معلومات دقيقة وسريعة بناءً على محتوى الموقع الرسمي

الكلمات المفتاحية: النماذج اللغوية الكبيرة، التوليد المعزز بالاسترجاع، الذكاء الاصطناعي، تجربة المستخدم، المواقع الذكية، التعليم العالي، معالجة اللغة الطبيعية

Abstract

The significant advancements in Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) technologies have created new opportunities for enhancing web development. These powerful tools enable the transformation of traditional websites into intelligent interactive platforms capable of delivering exceptional user experiences through accurate and instant responses to user inquiries.

This project aims to develop an upgraded version of the Higher Institute of Applied Sciences and Technology website, incorporating an intelligent chatbot powered by these cutting-edge technologies. The implemented solution provides visitors with improved and efficient user experiences by delivering precise, real-time information extracted from the official website content.

The system demonstrates how artificial intelligence can revolutionize academic websites by combining advanced natural language processing with institutional knowledge bases, setting a benchmark for smart educational platforms.

Keywords: *Large Language Models, Retrieval-Augmented Generation, Artificial Intelligence, User Experience, Smart Websites, Higher Education, Natural Language Processing*

المصطلحات

المصطلح بالعربية	المصطلح بالإنكليزية
الذكاء الاصطناعي	Artificial Intelligence (AI)
النماذج اللغوية الكبيرة	Large Language Model (LLM)
توليد مَعزَّز بالاسترجاع	Retrieval-Augmented Generation (RAG)
قاعدة بيانات المتجهات	Vector Database
تضمين	Embedding
تقطيع	Chunking
حجم القطعة النصية	Chunk Size
تقطيع دلالي	Semantic Chunking
البينة المعمارية النظيفة	Clean Architecture
نمط الوسيط	Mediator design pattern
فصل مسؤولية الأوامر والاستعلامات	Command Query Responsibility Segregation (CQRS)
فصل الأوامر عن الاستعلامات	Command Query Separation (CQS)
جارف الويب	Web Scraper
عنكبوت الويب	Web Spider
واجهة برمجة التطبيقات	Application Programming Interface (API)

المحتويات

قائمة الأشكال

قائمة الجداول

الفصل

1

مقدمة عن المشروع

1.1 مقدمة

في ظل التطور المتسارع الذي تشهده النماذج اللغوية الكبيرة (LLM) والتقنيات المتقدمة في مجال الذكاء الاصطناعي، برزت الحاجة إلى تحديث وتطوير الواجهات الرقمية للمؤسسات الأكاديمية لتلبية توقعات المستخدمين الحديثة. لم يعد الموقع الإلكتروني مجرد منصة لعرض المعلومات بشكل ثابت، بل أصبح أداة تفاعلية قوية لتقديم خدمة استثنائية والإجابة الدقيقة والسريعة على استفسارات الزوار.

يأتي هذا المشروع استجابة لهذه الحاجة، حيث يتناول بناء نسخة محدثة ومتكاملة لموقع الويب الخاص بالمعهد العالي للعلوم التطبيقية والتكنولوجيا. لا يقتصر المشروع على تحديث الواجهات لتكون عصرية وجذابة فحسب، بل يهدف بشكل أساسي إلى تعزيز قدرات الموقع من خلال دمج تقنية التوليد المعزز بالاسترجاع ،(RAG) وتضمين "تشات بوت" ذكي يعتمد على هذه التقنية ليعمل كمصدر معرفة دقيق ومحدث باستمرار، مستمداً معلوماته مباشرة من محتوى الموقع الرسمي. سيساهم هذا التكامل في توفير تجربة مستخدم محسنة وفعالة، وتحويل الموقع من منصة معلوماتية إلى مساعد رقمي ذكي.

2.1 هدف المشروع

تمحور الهدف الرئيسي لهذا المشروع حول تصميم وتطوير مكونين برمجيين أساسيين ومستقلين، ومن ثم دمجهما لإنشاء منصة ويب متكاملة وذكية للمعهد العالي للعلوم التطبيقية والتكنولوجيا. تمثلت الأهداف المنفصلة في التالي:

1. تطوير خدمة محادثة ذكية (تشات بوت) كمنتج مستقل: الهدف الأول كان بناء **خدمة قائمة بذاتها وقابلة لإعادة الاستخدام** تعتمد على تقنيات الذكاء الاصطناعي المتقدمة (RAG و LLM). تم تصميم هذه الخدمة لتكون **قابلة للتكامل مع أي موقع ويب** حيث تقوم بشكل آلي ببناء قاعدة معرفة خاصة بها من محتواه، ومن ثم تقديم إجابات دقيقة وموثوقة بناءً على هذا السياق.

2. بناء منصة ويب متكاملة للمعهد: الهدف الثاني كان تطوير موقع ويب حديث للمعهد بواجهات مستخدم جذابة، مدعومًا **بنظام إدارة محتوى**. يمكّن هذا النظام المسؤولين من إدارة جميع بيانات وصفحات الموقع .

يكمن الهدف النهائي للمشروع في **الدمج المتناغم** بين هذين النظامين، حيث يتم تضمين خدمة المحادثة الذكية ضمن منصة الويب الخاصة بالمعهد، مما يؤدي إلى تعزيز الحضور الرقمي للمعهد، تحسين تجربة المستخدمين، وتوفير أداة فعالة, متوفرة دوما ومبتكرة للتواصل وتقديم المعلومات .

3.1 متطلبات المشروع

تم تحديد متطلبات المشروع بناءً على الأدوار الرئيسية للمستخدمين (الزائر والمدير)، بالإضافة إلى الخصائص التقنية للنظام ككل. تنقسم هذه المتطلبات إلى وظيفية وغير وظيفية.

1.3.1 المتطلبات الوظيفية الخاصة بالمستخدم

يجب أن يوفر النظام للمستخدم العام (الزائر) القدرة على التفاعل مع الموقع والمساعد الذكي بسهولة وفعالية. تشمل المتطلبات الوظيفية للمستخدم ما يلي:

- **تصفح محتوى الموقع:** يجب أن يتمكن المستخدم من تصفح جميع الصفحات والمحتويات العامة للموقع، مثل (الأخبار، الأقسام الأكاديمية، البرامج الدراسية، شروط القبول، الكتب في المكتبة الرقمية، وغيرها).
- **التفاعل مع المساعد الذكي (التشات بوت):**
  - يجب أن يتمكن المستخدم من طرح أسئلة باللغة الطبيعية (العربية أو الإنجليزية) على التشات بوت.
  - يجب أن يقوم النظام بفهم سؤال المستخدم وتقديم إجابة دقيقة وموجزة بناءً على المحتوى الرسمي المسترجع من قاعدة المعرفة.
  - يجب أن يدعم التشات بوت المحادثة المستمرة (تذكر السياق) للإجابة على الأسئلة المتابعة.
- **البحث والتصفية:** يجب أن يتمكن المستخدم من البحث عن محتوى معين وتصفيته ضمن أقسام الموقع المختلفة (مثل تصفية الكتب حسب المؤلف أو سنة النشر).
- **دعم اللغات:** يجب أن يتمكن المستخدم من التنقل بين اللغتين العربية والإنجليزية في جميع واجهات الموقع.

2.3.1 المتطلبات الوظيفية الخاصة بالمدير

يجب أن يوفر النظام للمدير (Admin) صلاحيات كاملة لإدارة محتوى الموقع والنظام من خلال لوحة تحكم آمنة. تشمل المتطلبات الوظيفية للمدير ما يلي:

- **المصادقة والوصول الآمن:**
  - يجب أن يتمكن المدير من تسجيل الدخول إلى لوحة التحكم باستخدام بيانات اعتماد آمنة (بريد إلكتروني وكلمة مرور).
  - يجب أن تكون جميع وظائف الإدارة محمية ولا يمكن الوصول إليها إلا بعد تسجيل الدخول بنجاح.
- **إدارة المحتوى :** يجب أن يتمكن المدير من القيام بعمليات الإنشاء، القراءة، التحديث، والحذف على جميع كيانات النظام، بما في ذلك:
  - إدارة الصفحات الثابتة .
  - إدارة المنشورات و تصنيفاتها .
  - إدارة البرامج الأكاديمية و التخصصات.
  - إدارة الكتب و الوسائط في المكتبة الرقمية.
  - إدارة الأسئلة الشائعة و تصنيفاتها.
- **إدارة المساعد الذكي:**
  - يجب أن يتمكن المدير من إعادة تدريب قاعدة المعرفة الخاصة بالبوت عند تحديث محتوى الموقع لضمان دقة الإجابات.
  - يجب أن يتمكن المدير من تغيير استراتيجية الاسترجاع الخاصة بالبوت.

3.3.1 المتطلبات غير الوظيفية

هي الخصائص ومعايير الجودة التي يجب أن يلتزم بها النظام لضمان أدائه وموثوقيته. تشمل هذه المتطلبات ما يلي:

- **الأداء:** يجب أن يكون الموقع سريع الاستجابة، مع زمن تحميل للصفحات لا يتجاوز ثلاث ثوان في الظروف العادية. كما يجب أن يقدم التشات بوت إجاباته في غضون زمن لا يتجاوز الستة ثوان.
- **الأمان :(Security)** يجب حماية لوحة التحكم من الوصول غير المصرح به.
- **قابلية التوسع:** يجب تصميم النظام بطريقة تسمح له بالتعامل مع زيادة عدد المستخدمين وحجم المحتوى في المستقبل دون تدهور كبير في الأداء.

- **قابلية الصيانة:** يجب أن يكون الكود البرمجي منظماً ومبنياً على معمارية نظيفة، مما يسهل عملية فهمه وتعديله وإضافة ميزات جديدة في المستقبل.
- **التوافقية:** يجب أن تعمل واجهات الموقع بشكل سليم على مختلف المتصفحات الحديثة وعلى مختلف أحجام الشاشات.
- **التوافرية:** يجب تصميم النظام ليكون متاحاً للمستخدمين على مدار الساعة
- **قيود التقنيات المستخدمة:** يجب استخدام إطار العمل NET. للتطبيق الخلفي . تم فرض هذا القيد لضمان توافق المشروع مع المكدس التقني (Technology Stack) المعتمد في المعهد العالي، مما يسهل على مهندسي المعهد استكمال المشروع وتطويره ودمجه مع الأنظمة الأخرى في المستقبل.

الفصل

2

الدراسة النظرية

1.2 مبادئ التصميم للكائنات الموجهة (SOLID Principles)

تعتبر مبادئ SOLID مجموعة من خمسة مبادئ أساسية في البرمجة الكائنية تهدف إلى جعل تصميم البرمجيات أكثر قابلية للفهم، المرونة، والصيانة. تمثل هذه المبادئ حجر الزاوية في كتابة كود نظيف ومنظم، وهي الأساس الذي تُبنى عليه العديد من الأنماط المعمارية المتقدمة، بما في ذلك البنية المعمارية النظيفة التي تم اعتمادها في هذا المشروع.

S- مبدأ المسؤولية الواحدة (Single Responsibility Principle): ينص هذا المبدأ على أن كل كلاس أو وحدة برمجية يجب أن يكون لها مسؤولية واحدة فقط.

O- مبدأ الفتح/الإغلاق (Open/Closed Principle): ينص هذا المبدأ على أن الوحدات البرمجية يجب أن تكون مفتوحة للتوسع و لكن مغلقة للتعديل. تم تحقيق ذلك في بنية المساعد الذكي، حيث يمكن إضافة "استراتيجيات" جديدة لجلب المعلومات دون الحاجة إلى تعديل الكود الأساسي الذي يدير الاستراتيجيات.

L- مبدأ استبدال ليسكوف (Liskov Substitution Principle): ينص هذا المبدأ على أن الكائنات من الصفوف المشتق يجب أن تكون قادرة على استبدال كائنات الصفوف الأساسي دون التأثير على صحة البرنامج.

I- مبدأ فصل الواجهات (Interface Segregation Principle): ينص هذا المبدأ على أنه من الأفضل أن يكون لديك العديد من الواجهات الصغيرة والمحددة بدلاً من واجهة واحدة كبيرة وعامة.

D- مبدأ عكس التبعية(Dependency Inversion Principle): ينص هذا المبدأ على أن الوحدات عالية المستوى لا يجب أن تعتمد على الوحدات منخفضة المستوى، بل يجب أن يعتمد كلاهما على تجريدات. هذا هو المبدأ الأكثر أهمية في البنية النظيفة.

2.2 البنية المعمارية النظيفة (Clean Architecture)

1.2.2 مفهوم البنية المعمارية النظيفة

هي فلسفة لتصميم البرمجيات ونمط تصميمي معماري، يهدف إلى إنشاء نظام برمجي قابل للصيانة ومستقل عن تفاصيل التنفيذ. يعزز هذا النهج استقلالية المكونات المختلفة للنّظام، مثل واجهات المستخدم ومنطق الأعمال وتخزين البيانات وغيرها، مما يسمح لهذه المكّونات بالتطور بشكل مستقل دون التأثير على النظام بأكمله، كما توفّر هذه المنهجيّة قاعدة رماز (codebase) قابلة للاختبار، مما يسهل إدارة وتوسيع النظام البرنامجي [2].
ينتج عن استخدام البنية النظيفة نظام يحمل الخواص التالية [2]:

مستقل عن أطر العمل (Independence of Frameworks): لا يرتبط منطق العمل الأساسي والمكونات الأساسية للنظام بأي إطار خارجي أو مكتبة أو تقنية معينة، الأمر الذي يسمح باستبدال أو تحديث هذه التبعيات الخارجية دون التأثير على الوظائف الأساسيّة للنظام.

استقلالية واجهات المستخدم (Independence of User Interface): تكون واجهة المستخدم منفصلة عن بقية النظام، مما يسمح بإجراء تغييرات على واجهة المستخدم دون التأثير على منطق الأعمال الأساسي، بالإضافة إلى إمكانية وجود واجهات مستخدم متعددة (الويب، الجوال، ..إلخ) تستخدم نفس منطق الأعمال الأساسي.

استقلالية تخزين البيانات (Independence of Data Storage): إمكانية تغيير آلية تخزين المعطيات دون التأثير على بقية أجزاء النظام.

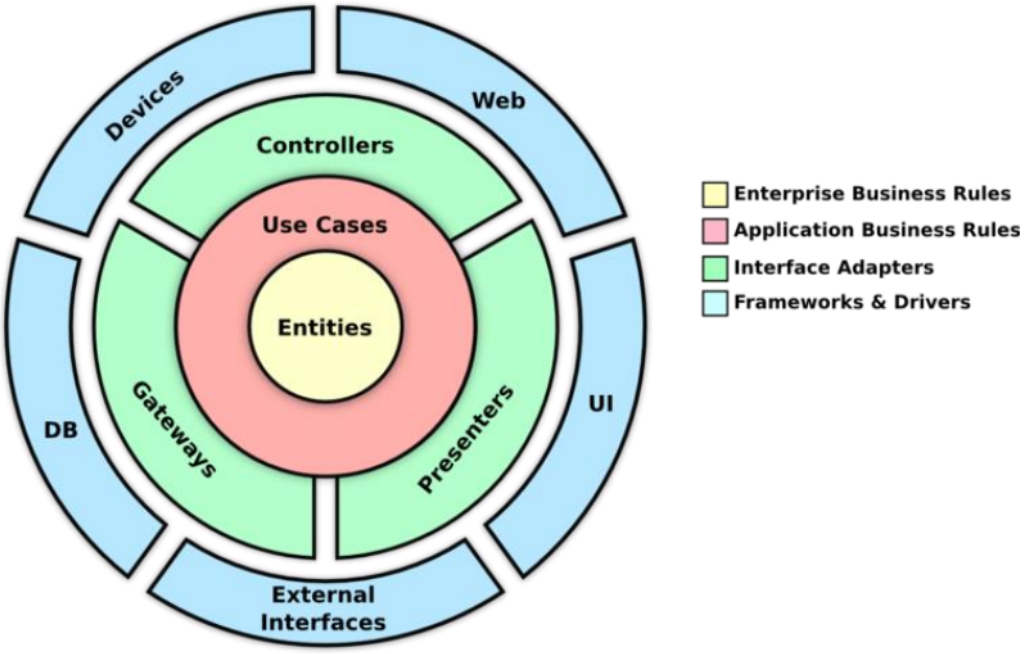
نظام قابل للاختبار :(Testable) إمكانية اختبار قواعد العمل بدون واجهة المستخدم أو قاعدة البيانات أو خادم الويب أو أي عنصر خارجي آخر.

2.2.2 قاعدة التبعية

تعد قاعدة التبعية مبدأً أساسياً يوجّه كيفية تنظيم المكونات والوحدات داخل النظام البرمجي وتفاعلها مع بعضها البعض في البنية النظيفة. تنص قاعدة التبعية على أن التبعيات يجب أن تشير دائماً إلى الداخل نحو جوهر النظام، مما يعني أن المكونات ذات المستوى الأعلى والأكثر تجريداً وثباتاً يجب ألا تعتمد على المكونات ذات المستوى الأدنى، بدلاً من ذلك، يجب أن تتدفق التبعيات من الطبقات الخارجية نحو الطبقات الداخلية للبنية [2].

3.2.2 مكونات البنية النظيفة

تتكون البنية النظيفة بشكل رئيسي من الطبقات التالية [2]:



الشكل 1.2: مكونات البنية المعمارية النظيفة.

- **Entities:** تمثّل الكيانات عناصر الأعمال الأساسية وقواعد التطبيق، حيث تقوم بتغليف البيانات الأساسية والسلوك الذي يحدد الغرض من التطبيق. تكون هذه الكيانات مستقلة عن العوامل والأطر الخارجية، وتركز فقط على تمثيل مجال الأعمال.
- **Use Cases:** تحدد حالات الاستخدام منطق عمل التطبيق وتنظم التفاعلات بين الكيانات، حيث تلخّص العمليات المحددة ومهام سير العمل التي يدعمها التطبيق، وتكون حالات الاستخدام مستقلة عن الواجهات الخارجية والتقنيات.
- **Interface Adapters**: محولات الواجهة تسد الفجوة بين حالات الاستخدام والعالم الخارجي، حيث تعمل على تحقيق التواصل وتحويل البيانات بين نواة التطبيق وأطر العمل الخارجية، مما يضمن بقاء النواة معزولة عن تفاصيل التنفيذ.
- **Frameworks and Drivers:** الأطر والمحركات هي الطبقة الخارجية للبنية، التي توفر الأدوات والبنية التحتية المطلوبة لتشغيل التطبيق، مثل أطر عمل الويب ومكتبات واجهة المستخدم وقواعد البيانات وغيرها. تتفاعل هذه الأدوات مع التطبيق من خلال محولات الواجهة.

3.2 أنظمة إدارة المحتوى (Content Management Systems - CMS)

نظام إدارة المحتوى هو تطبيق برمجي يتيح للمستخدمين إنشاء المحتوى الرقمي وتعديله وإدارته دون الحاجة إلى معرفة تقنية متخصصة. على مر السنين، تطورت بنيات هذه الأنظمة بشكل كبير لتلبية متطلبات مختلفة. نستعرض هنا ثلاثة نماذج رئيسية.

1.0.3.2 نظام إدارة المحتوى التقليدي (Traditional CMS)

يُعرف أيضاً بالنظام المتجانس (Monolithic). في هذا النموذج، تكون طبقة إدارة المحتوى (الجزء الخلفي - Backend) وطبقة عرض المحتوى (الجزء الأمامي - Frontend) مترابطتين بشدة في تطبيق واحد. يقوم النظام بإدارة المحتوى في قاعدة بيانات، وعندما يطلب المستخدم صفحة، يقوم الخادم بتجميعها من قوالب محددة مسبقاً وتقديمها كصفحة HTML. من أشهر الأمثلة على ذلك WordPress و Drupal. يتميز هذا النموذج بسهولة الإعداد للمستخدمين غير التقنيين، لكنه يفتقر إلى المرونة في عرض المحتوى على قنوات مختلفة غير الويب.

2.0.3.2 مولدات المواقع الثابتة (Static Site Generators - SSG)

يمثل هذا النهج طريقة مختلفة تماماً. بدلاً من بناء الصفحات عند كل طلب، يقوم مولد المواقع الثابتة ببناء جميع صفحات الموقع كملفات HTML ثابتة مرة واحدة أثناء عملية النشر (Build Time). يتم أخذ المحتوى عادةً من ملفات نصية بسيطة مثل Markdown. من أشهر الأمثلة على ذلك Jekyll و Hugo. يتميز هذا النهج بالأداء الفائق والأمان العالي لأن لا توجد قاعدة بيانات أو معالجة من جانب الخادم للتعامل معها عند خدمة المستخدم، ولكنه أقل ديناميكية ويتطلب إعادة بناء الموقع بالكامل لنشر أي تغيير في المحتوى.

3.0.3.2 نظام إدارة المحتوى بدون رأس (Headless CMS)

هو النمط المعماري الذي تم اعتماده في هذا المشروع. يقوم هذا النموذج بفصل طبقة إدارة المحتوى ("الجسم") عن طبقة عرض المحتوى ("الرأس") بشكل كامل. يركز نظام Headless CMS فقط على تخزين المحتوى وتنظيمه وتوفيره عبر واجهات برمجة التطبيقات (API)، عادةً بتنسيق JSON.

هذا الفصل المعماري يوفر مرونة هائلة، حيث يمكن للمطورين بناء أي عدد من "الرؤوس" (مثل موقع ويب بتقنية React [13]، تطبيق جوال أصلي، أو حتى تغذية نظام ذكاء اصطناعي) التي تستهلك المحتوى من نفس النظام الخلفي. يجمع هذا النهج بين ديناميكية إدارة المحتوى في النموذج التقليدي ومرونة فصل الواجهات التي تتيح أداءً عالياً وأماناً معززاً.

الجدول 1.2: مقارنة بين بنيات أنظمة إدارة المحتوى المختلفة

المعيار	CMS Traditional	Generator Site Static	**CMS Headless**
البنية	متجانسة (مترابطة)	ثابتة (مبنية مسبقاً)	**مفصولة (Decoupled)**
المرونة	منخفضة (مقيدة بالقوالب)	منخفضة (للمحتوى الديناميكي)	**عالية جداً (أي واجهة أمامية)**
الأداء	متوسط (يعتمد على الخادم)	عالي جداً (ملفات ثابتة)	**عالي (يعتمد على الواجهة الأمامية)**
الأمان	متوسط (نقاط هجوم متعددة)	عالي جداً (سطح هجوم صغير)	**عالي (فصل الواجهات يقلل المخاطر)**
قابلية التوسع	متوسطة (تتطلب توسيع التطبيق كاملاً)	عالية (سهولة توزيع الملفات الثابتة)	**عالية جداً (توسيع الواجهة والخلفية بشكل مستقل)**
سهولة إدارة المحتوى	عالية جداً (لغير التقنيين)	منخفضة (تتطلب معرفة تقنية)	**عالية (من خلال واجهات مخصصة أو لوحة تحكم)**

4.2 فصل الأوامر والاستعلامات (CQS)

مبدأ فصل الأوامر والاستعلامات (CQS) ينص على أن كل تابع (Method) في الكائن يجب أن يكون إما **أمراً (Command)** أو **استعلاماً (Query)**، ولكن ليس كليهما في نفس الوقت.

- **الأمر (Command):** هو تابع يقوم بتغيير حالة الكائن أو النظام (أي له آثار جانبية)، ولكنه لا يعيد أي قيمة. على سبيل المثال، تابع يقوم بحفظ بيانات في قاعدة البيانات.

- **الاستعلام (Query):** هو تابع يقوم بإعادة بيانات من النظام، ولكنه لا يغير حالة النظام بأي شكل من الأشكال (أي ليس له آثار جانبية). على سبيل المثال، تابع يقوم بقراءة بيانات من قاعدة البيانات وعرضها.

يهدف هذا المبدأ إلى تعزيز وضوح النظام وقابلية سلوكه للتنبؤ. يتحقق ذلك عبر الفصل بين التوابع التي تعدّل الحالة (الأوامر) وتلك التي تسترجع البيانات فقط (الاستعلامات). هذا الفصل يحد من الآثار الجانبية غير المتوقعة ويسهّل اكتشاف الأخطاء وتصحيحها، مما يعزز موثوقية النظام.

يعتبر مبدأ CQS هو الأساس النظري الذي تطور عنه النمط المعماري الأكثر تقدماً وهو فصل مسؤولية الأوامر والاستعلامات (CQRS) [4].

5.2 أنماط التصميم (Design Patterns)

أنماط التصميم هي حلول عامة ومجربة وقابلة لإعادة الاستخدام للمشاكل الشائعة التي تواجه المطورين ضمن سياق معين في تصميم البرمجيات. لا تمثل هذه الأنماط كوداً برمجياً يمكن نسخه مباشرة، بل هي وصف أو قالب لكيفية حل مشكلة معينة يمكن استخدامه في مواقف مختلفة، مما يساعد على بناء أنظمة أكثر مرونة وقابلية للصيانة [4].

1.5.2 أنماط الوصول للبيانات: Repository و Unit of Work

1.1.5.2 نمط المستودع (Repository Pattern)

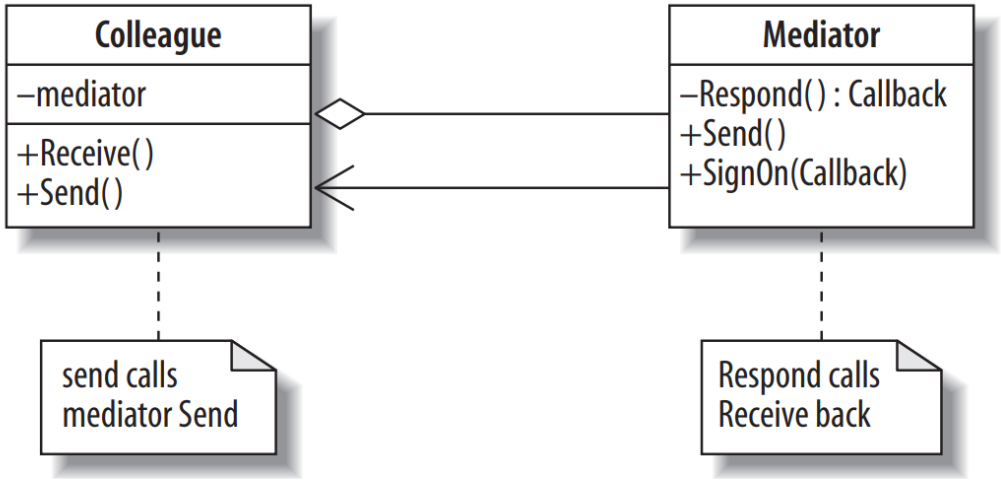
نمط المستودع هو نمط تصميمي يعمل كوسيط بين طبقة منطق العمل وطبقة الوصول للبيانات. الهدف الرئيسي منه هو تجريد منطق الوصول إلى البيانات وإخفاء تفاصيل كيفية جلبها وتخزينها، سواء كانت من قاعدة بيانات، أو خدمة ويب، أو ملف. يقدم هذا النمط واجهة بسيطة تشبه المجموعات للتعامل مع كائنات المجال، مما يفصل منطق العمل عن تقنيات التخزين.

2.1.5.2 نمط وحدة العمل (Unit of Work Pattern)

يعمل هذا النمط عادةً جنباً إلى جنب مع نمط المستودع، ويهدف إلى إدارة المعاملات (Transactions) في قاعدة البيانات. يقوم النمط بتجميع قائمة من العمليات (مثل الإضافة، التحديث، الحذف) التي يجب أن تنفذ كوحدة ذرية واحدة. يضمن هذا النمط أنه إما أن تنجح جميع العمليات معاً أو تفشل جميعها، مما يحافظ على تناسق وسلامة البيانات.

2.5.2 Mediator Pattern

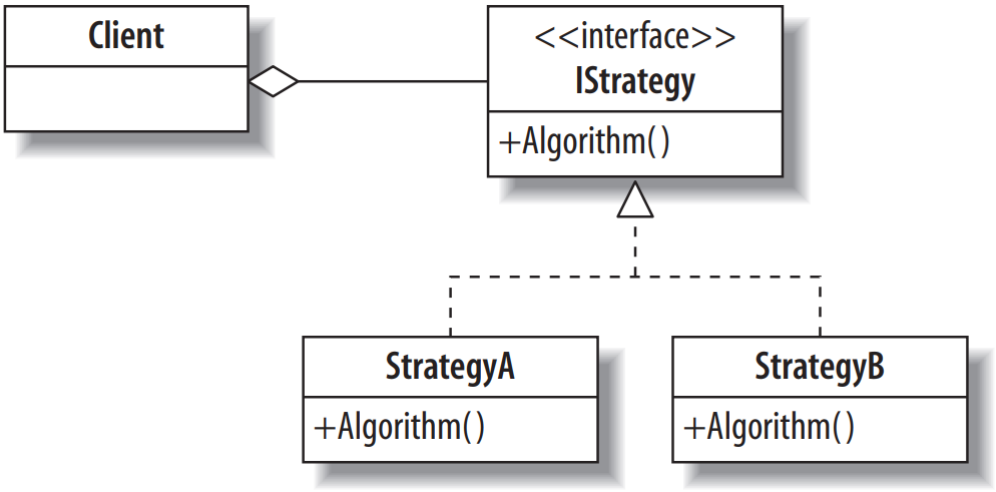
نمط الوسيط هو نمط تصميم سلوكي يهدف إلى تقليل التبعيات المباشرة بين مجموعة من الكائنات [4]. يتم ذلك عن طريق تغليف تفاعلاتها المعقدة ضمن كائن وسيط واحد (Mediator)). بدلاً من أن تتواصل الكائنات مع بعضها البعض مباشرة، مما يؤدي إلى شبكة معقدة من العلاقات (High Coupling)، فإنها ترسل طلباتها إلى الوسيط، الذي يقوم بدوره بتوجيه الطلب إلى المستلم المناسب. هذا يسهل صيانة الكود ويجعل النظام أكثر مرونة.



الشكل 2.2: مخطط الصفوف (Class Diagram) لبنية نمط الوسيط.

Strategy Pattern 3.5.2

هو نمط تصميم سلوكي يسمح بتعريف عائلة من الخوارزميات، وتغليف كل واحدة منها في كائن مستقل، وجعلها قابلة للتبديل [4]. يتيح هذا النمط للعميل اختيار الخوارزمية التي يريد استخدامها في وقت التشغيل دون أن يكون لديه معرفة مسبقة بتفاصيل تنفيذها. يستخدم هذا النمط لزيادة مرونة النظام وقابليته للتوسع عند وجود طرق متعددة لتنفيذ مهمة معينة.



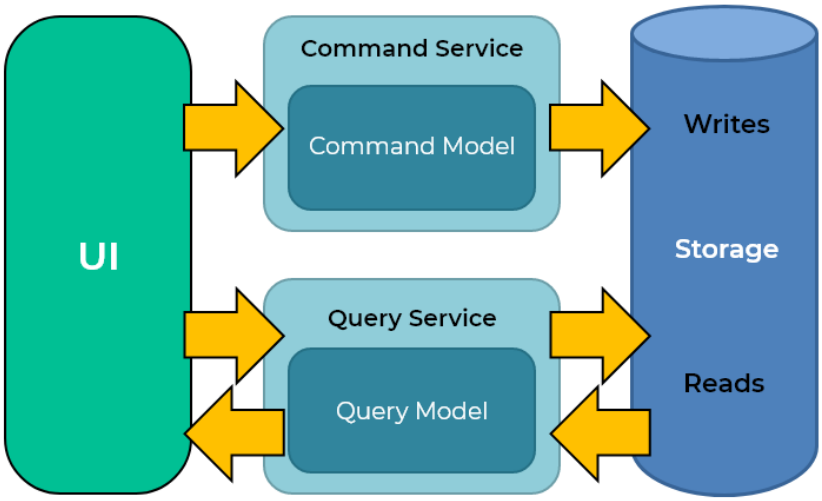
الشكل 3.2: مخطط الصفوف (Class Diagram) لبنية نمط الاستراتيجية.

فصل مسؤولية الأوامر والاستعلامات (CQRS) 4.5.2

هو نمط معماري يأخذ مبدأ **فصل الأوامر والاستعلامات**، الذي تمت مناقشته سابقاً، إلى مستوى النظام بأكمله. يفصل هذا النمط بين نماذج عمليات قراءة البيانات (Queries) ونماذج عمليات تعديل البيانات (Commands). الهدف من ذلك هو إنشاء مسارين منفصلين ومستقلين تماماً داخل التطبيق:

- **مسار الأوامر (Command Stack):** مخصص لمعالجة الطلبات التي تغير حالة النظام.
- **مسار الاستعلامات (Query Stack):** مخصص لمعالجة الطلبات التي تقرأ البيانات فقط.

هذا الفصل يسمح بتحسين أداء كل مسار بشكل مستقل، وتقليل تعقيد النماذج، وزيادة قابلية النظام للتوسع.



الشكل 4.2: مخطط يوضح تدفق البيانات في نمط CQRS.

5.5.2 نمط الاستجابة الموحدة (API Result Pattern)

هو نمط يستخدم لتوحيد هيكلية الاستجابات التي تعيدها واجهات برمجة التطبيقات. بدلاً من أن تعيد كل واجهة بياناتها بتنسيق مختلف، يتم تغليف جميع الاستجابات، سواء كانت ناجحة أو فاشلة، ضمن كائن موحد له بنية ثابتة. عادةً ما تحتوي هذه البنية على حقول مثل حالة النجاح، والبيانات الفعلية، ورسائل الخطأ. هذا النمط يسهل بشكل كبير على تطبيقات العميل (مثل تطبيقات الويب أو الجوال) التعامل مع الاستجابات ومعالجتها بطريقة متسقة.

6.2 تجريف الويب (Web Scraping)

تجريف الويب هو عملية آلية لاستخراج كميات كبيرة من البيانات من مواقع الويب. بدلاً من أن يقوم المستخدم بنسخ ولصق المحتوى يدوياً، يتم استخدام برامج أو "بوتات" (Bots) لزيارة صفحات الويب، تحليل بنيتها (عادةً ما تكون بتنسيق HTML)، واستخلاص المعلومات المطلوبة بشكل منظم، ثم حفظها في تنسيق قابل للاستخدام مثل ملفات JSON أو قواعد البيانات.
في سياق هذا المشروع، يعتبر تجريف الويب خطوة حاسمة وأساسية. لكي يتمكن المساعد الذكي من الإجابة على الأسئلة بدقة، يجب أن تكون لديه قاعدة معرفة شاملة ومحدثة. لذلك، تم بناء آلية لتجريف الويب تقوم بتغذية نظام RAG بالبيانات اللازمة.
تتضمن عملية تجريف الويب بشكل عام الخطوات الأساسية التالية:

1. **إرسال طلب HTTP:** يقوم البرنامج بإرسال طلب إلى خادم الويب للحصول على محتوى صفحة معينة، تماماً كما يفعل المتصفح.

2. **تحليل HTML (Parsing):** بعد استلام محتوى الصفحة، يتم استخدام مكتبات متخصصة (مثل BeautifulSoup في بايثون [15]) لتحليل شجرة HTML وتسهيل التنقل بين عناصرها المختلفة.

3. **استخراج البيانات (Extraction):** يتم تحديد العناصر التي تحتوي على البيانات المطلوبة (مثل الفقرات النصية، العناوين، الجداول) باستخدام محددات CSS أو XPath، ثم يتم استخلاص المحتوى النصي منها.

4. **التنظيف والتنسيق (Cleaning and Formatting):** غالباً ما تحتاج البيانات المستخرجة إلى عملية تنظيف لإزالة العناصر غير المرغوب فيها مثل إعلانات، قوائم التنقل، أو وسوم HTML الزائدة، ثم يتم تنسيقها وحفظها.

في هذا المشروع، تم تصميم خوارزمية تجريف عودي لا تكتفي بصفحة واحدة، بل تبدأ من رابط معين وتتبع جميع الروابط الداخلية لجمع المحتوى من كافة صفحات الموقع ذات الصلة، وسيتم استعراض تفاصيل تصميم هذه الخوارزمية في الفصول اللاحقة.

7.2 منهجيات الذكاء الاصطناعي لتطبيقات المحادثة

يشهد مجال الذكاء الاصطناعي تطوراً هائلاً، خاصة في مجال معالجة اللغات الطبيعية. في هذا القسم، نستعرض المفاهيم الأساسية التي تشكل العمود الفقري للمساعد الذكي في هذا المشروع.

1.7.2 نماذج اللغة الكبيرة (Large Language Models - LLM)

نماذج اللغة الكبيرة هي نماذج ذكاء اصطناعي متقدمة تم تدريبها على كميات هائلة من البيانات النصية، مما أكسبها قدرة فائقة على فهم اللغة البشرية وتوليد نصوص متماسكة وذات معنى في مختلف السياقات. تعمل هذه النماذج كشبكات عصبونية عميقة، عادة ما تكون مبنية على معمارية المحولات (Transformer Architecture)، مما يمكنها من التعامل مع العلاقات المعقدة بين الكلمات.

على الرغم من قدراتها الهائلة، إلا أن نماذج LLM تواجه تحديين أساسيين:

- **المعرفة المحدودة:** تقتصر معرفتها على البيانات التي تم تدريبها عليها، والتي تصبح قديمة مع مرور الوقت.
- **الهلوسة (Hallucination):** قد تقوم النماذج أحياناً بتوليد معلومات غير صحيحة أو غير موجودة في بيانات تدريبها بثقة تامة.

للتغلب على هذه التحديات، تم تطوير تقنيات متقدمة مثل التوليد المعزز بالاسترجاع.

2.7.2 التوليد المعزز بالاسترجاع (Retrieval-Augmented Generation - RAG)

لمعالجة تحديات نماذج اللغة الكبيرة، تم تقديم تقنية التوليد المعزز بالاسترجاع [3].

1.2.7.2 ما هو التوليد المعزز بالاسترجاع

هي تقنية تهدف إلى تعزيز قدرات نماذج اللغة الكبيرة من خلال ربطها بمصدر معرفة خارجي ومحدث. بدلاً من الاعتماد فقط على المعرفة الداخلية للنموذج، يقوم نظام RAG بتنفيذ عملية من ثلاث مراحل أساسية، والتي يشير إليها الاختصار نفسه:

R - الاسترجاع (Retrieval): هي عملية البحث في قاعدة المعرفة الخارجية (مثل المستندات أو محتوى موقع الويب) والعثور على الأجزاء الأكثر صلة بسؤال المستخدم.

A - التعزيز (Augmentation): هي عملية أخذ المعلومات التي تم استرجاعها و "تعزيز" أو "إثراء" سؤال المستخدم الأصلي بها، عن طريق دمجها معه لتكوين سياق متكامل.

G - التوليد (Generation): هي المرحلة النهائية التي يتم فيها تقديم السياق المعزز إلى نموذج اللغة الكبير (LLM) ليقوم بتوليد إجابة طبيعية ودقيقة بناءً على هذا السياق.

بهذه الطريقة، يحول RAG مهمة الإجابة على الأسئلة من "تذكر" المعلومة من ذاكرة النموذج الداخلية إلى مهمة "قراءة وفهم" نص مقدم، مثلما يفعل الإنسان عند الإجابة على سؤال من كتاب مفتوح. هذا يجعل الإجابات أكثر دقة وموثوقية، ويقلل بشكل كبير من مشكلة الهلوسة.

2.2.7.2 المكونات الأساسية لـ RAG: التقسيم، التضمين، والاسترجاع

لتجهيز وتشغيل نظام RAG، تمر العملية بعدة خطوات أساسية:

- **التقسيم (Chunking):** نظراً لوجود حدود على حجم النص الذي يمكن معالجته، يتم تقسيم المستندات الطويلة إلى أجزاء أصغر ومتماسكة تسمى "القطع" (Chunks). هذه الخطوة ضرورية لتسهيل عملية التضمين والاسترجاع الدقيق. يعتمد اختيار استراتيجية التقسيم على طبيعة البيانات، وهناك عدة طرق شائعة لتحقيق ذلك:
  - **التقسيم بحسب حجم ثابت (Fixed-size Chunking):** هي أبسط طريقة، حيث يتم تقسيم النص إلى أجزاء ذات حجم ثابت مع إمكانية وجود تداخل (Overlap) بين القطع المتجاورة لضمان عدم فقدان السياق.
  - **التقسيم العودي المعتمد على المحتوى (Content-Aware Recursive Chunking):** طريقة أكثر ذكاءً، تحاول تقسيم النص بناءً على بنيته الدلالية (مثل فواصل الفقرات والجمل).
  - **التقسيم المعتمد على نوع المستند (Document-specific Chunking):** يتم تصميم الاستراتيجية لتناسب نوع المستند، مثل التقسيم بناءً على العناوين الرئيسية في ملفات Markdown.

- **التضمين (Embedding):** هي عملية تحويل كل قطعة نصية إلى تمثيل رقمي على شكل متجه (Vector) باستخدام نموذج تضمين متخصص. هذا المتجه يلتقط المعنى الدلالي للنص.

نموذج التضمين (Google Embedding Model): لتحويل القطع النصية إلى متجهات رقمية، تم الاعتماد على نماذج التضمين التي تقدمها جوجل [14]، وتحديدًا نموذج 'models/embedding-001'. تُعد هذه النماذج جزءًا من عائلة نماذج Gemini، وهي مصممة لتحقيق أداء عالٍ في فهم المعنى الدلالي للنصوص. تم اختيارها لهذا المشروع لعدة أسباب رئيسية: أولاً، لدعمها القوي للتعدد اللغوي، مما يجعلها فعالة جدًا في التعامل مع المحتوى باللغتين العربية والإنجليزية. ثانيًا، لقدرتها على توليد متجهات عالية الجودة تعكس الفروق الدقيقة في المعنى، وهو أمر حاسم لضمان دقة عملية البحث عن التشابه في نظام RAG.

- **التخزين والفهرسة (Storage and Indexing):** يتم تخزين المتجهات الناتجة في قاعدة بيانات متخصصة تسمى قاعدة البيانات المتجهة (Vector Database)، والتي تقوم بفهرستها للسماح بالبحث السريع عن التشابه.

- **الاسترجاع (Retrieval):** عندما يطرح المستخدم سؤالاً، يتم تحويل سؤاله أيضاً إلى متجه. بعد ذلك، يتم البحث في قاعدة البيانات المتجهة عن المتجهات الأكثر تشابهاً معه. يتم قياس هذا "التشابه الدلالي" رياضياً باستخدام دوال قياسية في مجال التنقيب عن البيانات واسترجاع المعلومات [16]:

تشابه جيب التمام (Cosine Similarity): يقيس جيب التمام للزاوية بين متجهين، وهو فعال بشكل خاص في تحديد تشابه النصوص بغض النظر عن طولها.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1.2}$$

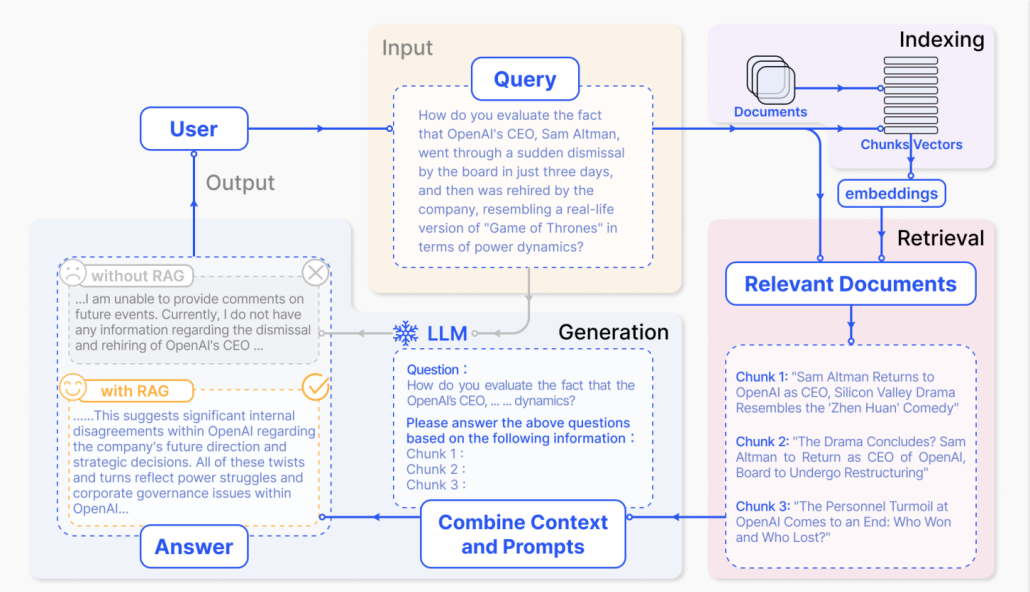
المسافة الإقليدية (Euclidean Distance): تقيس المسافة الهندسية المستقيمة بين نقطتين في الفضاء المتجهي. كلما كانت القيمة أصغر، كان التشابه أكبر.

$$d(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \tag{2.2}$$

الضرب النقطي (Dot Product): يقيس التشابه مع الأخذ في الاعتبار حجم (magnitude) المتجهات، مما يجعله حساسًا لكل من الاتجاه والقوة.

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i B_i \tag{3.2}$$

القطع النصية التي تحقق أعلى درجة تشابه هي التي يتم استرجاعها لتشكيل السياق الذي سيتم إرساله إلى النموذج اللغوي لتوليد الإجابة النهائية.



الشكل 2.5: مثال يوضح كيفية عمل ال RAG و الفرق بينها و بين ال LLM

الفصل

3

الدراسة المرجعية

في هذا الفصل، نقوم باستعراض الأبحاث والدراسات الحديثة التي شكلت الأساس النظري والعملي لتصميم المساعد الذكي في هذا المشروع. يركز هذا الاستعراض على فهم تطور تقنيات التوليد المعزز بالاسترجاع (RAG) وتحديد المنهجيات الأكثر تقدماً وفعالية.

1.3 استعراض شامل لمنهجيات RAG وتطورها

بناءً على ورقة المسح الشامل التي قدمها Gao et al. (2024) [3], يمكن تقسيم تطور تقنيات RAG إلى ثلاث مراحل رئيسية. لم يقتصر هذا المشروع على تطبيق المرحلة الأولى فقط، بل استلهم من مفاهيم المراحل المتقدمة.

RAG التقليدي (Naive RAG): يمثل هذا النهج البنية الأساسية لـ RAG التي تم شرحها في الفصل النظري (فهرسة، استرجاع، توليد). على الرغم من فعاليته، إلا أنه يواجه تحديات مثل استرجاع مستندات غير دقيقة عندما تكون أسئلة المستخدم معقدة أو غامضة، مما يؤدي إلى إجابات ضعيفة.

RAG المتقدم (Advanced RAG): لمعالجة قصور النهج التقليدي، يقدم الباحثون تحسينات على مراحل مختلفة من خط الأنابيب. يركز هذا النهج على تقنيات ما قبل الاسترجاع (Pre-retrieval) وما بعد الاسترجاع (Post-retrieval). من أبرز هذه التقنيات التي تم استلهامها في مشروعنا:

- **تحسين الاستعلام (Query Optimization):** بدلاً من استخدام سؤال المستخدم كما هو، يتم إعادة صياغته أو توسيعه لتوليد استعلامات أكثر دقة وشمولية، مما يحسن من فرصة العثور على المستندات الصحيحة.
- **إعادة الترتيب (Re-ranking):** بعد استرجاع مجموعة أولية من المستندات، يتم استخدام نموذج أكثر تعقيداً لإعادة ترتيبها ووضع المستندات الأكثر صلة في المقدمة، مما يضمن أن السياق المقدم للنموذج اللغوي هو الأعلى جودة.

RAG النمطي (Modular RAG): يمثل هذا النهج رؤية مستقبلية أكثر مرونة، حيث يتم التعامل مع مكونات RAG كوحدات نمطية (Modules) يمكن استبدالها أو ترتيبها. من أبرز الأمثلة على هذه الوحدات هي **وحدة الدمج (Fusion Module)**,

التي تهدف إلى دمج نتائج البحث من مصادر متعددة، وهو المبدأ الذي تقوم عليه تقنية RAG-Fusion.

2.3 دراسة معمقة لتقنية RAG-Fusion

كأحد التطبيقات المتقدمة والمبتكرة لمفاهيم Advanced RAG، تم دراسة تقنية RAG-Fusion التي قدمها Rakesh et al. (2024) [1] بشكل معمق، حيث تمثل هذه التقنية إحدى الاستراتيجيات الأساسية المطبقة في هذا المشروع.

تهدف RAG-Fusion إلى التغلب على قيود البحث المعتمد على متجه واحد، وذلك من خلال توليد وجهات نظر متعددة لسؤال المستخدم ودمج نتائج البحث الخاصة بها. تتم العملية عبر الخطوات التالية:

1. **توليد استعلامات متعددة:** يتم أولاً تمرير سؤال المستخدم الأصلي إلى نموذج لغوي كبير (LLM) ويُطلب منه توليد عدة صيغ مختلفة من السؤال، كل صيغة تمثل منظوراً أو زاوية مختلفة.

2. **البحث المتوازي:** يتم إجراء عملية بحث متجهي مستقلة لكل استعلام من الاستعلامات التي تم توليدها، مما ينتج عنه عدة قوائم من المستندات المسترجعة.

3. **إعادة الترتيب والدمج (Reranking and Fusion):** تهدف هذه الخطوة إلى دمج قوائم النتائج المتعددة في قائمة واحدة نهائية ومحسنة، وهناك أساليب مختلفة لتحقيق ذلك.

   أحد الأساليب هو **تصويت الأغلبية (Majority Vote)**، الذي يعطي الأولوية **لتكرار** ظهور المستند. في هذه الطريقة، يتم ترتيب المستندات التي تظهر في عدد أكبر من القوائم في مرتبة أعلى، مما يعكس الإجماع حولها.

   أسلوب آخر، وهو المتبع في الأبحاث المرجعية، هو **Reciprocal Rank Fusion (RRF)**. هذه الخوارزمية تعطي الأولوية لكل من **التكرار والترتيب**. فهي تمنح وزناً أعلى للمستندات التي لا تظهر بشكل متكرر فحسب، بل تحتل أيضاً مراتب متقدمة في قوائمها. يتم حساب نقاط كل مستند في كل قائمة باستخدام المعادلة التالية [1]:

$$\text{rrf\_score} = \frac{1}{k + \text{rank}} \tag{1.3}$$

حيث أن:

- **rank:** هو ترتيب المستند الحالي في قائمة النتائج (1 للمركز الأول، 2 للثاني، وهكذا).
- **k:** هو ثابت "تنعيم" (عادة ما تكون قيمته 60) يستخدم للتقليل من التأثير الطاغي للمراتب الأولى.

بعد حساب النقاط، يتم تجميعها لكل مستند فريد عبر جميع القوائم، مما يضمن أن المستندات التي اعتبرتها كل عملية بحث فردية أكثر أهمية تحظى بأولوية أكبر في القائمة النهائية المدمجة.

4. **التوليد النهائي:** يتم أخذ أفضل المستندات من القائمة المدمجة النهائية وتقديمها كسياق للنموذج اللغوي لتوليد الإجابة النهائية.

أثبتت هذه التقنية قدرتها على تحسين جودة الاسترجاع بشكل ملحوظ، خاصة مع الأسئلة المعقدة التي تتطلب دمج معلومات من مصادر متعددة.

3.3 خلاصة وتبرير النهج المختار للمشروع

بناءً على الدراسات السابقة، يتضح أن الاعتماد على RAG التقليدي قد لا يكون كافياً لتقديم تجربة مستخدم مثالية. لذلك، تم اتخاذ قرار تصميمي استراتيجي في هذا المشروع يتجاوز التنفيذ الأساسي، حيث تم بناء نظام مرن يطبق مفاهيم Advanced RAG بشكل عملي.
لتلبية متطلبات المستخدمين المختلفة، من الأسئلة البسيطة والمباشرة إلى الأسئلة المعقدة ومتعددة الأوجه، تم تصميم وتنفيذ عدة استراتيجيات مختلفة للإجابة، بما في ذلك:

- استراتيجية RAG افتراضية للأسئلة البسيطة.
- استراتيجية تعتمد على تحويل الاستعلام (Query Transformation).
- استراتيجية RAG-Fusion الكاملة للأسئلة المعقدة.

ولتحقيق هذه المرونة، تم استخدام **نمط التصميم الاستراتيجي (Strategy Pattern)**، الذي تم شرحه في الفصل النظري. يسمح هذا النمط للنظام بالتبديل بين هذه الاستراتيجيات بسهولة، ويوفر للمدير القدرة على اختيار الاستراتيجية الأنسب عبر لوحة التحكم. هذا النهج لا يضمن فقط دقة الإجابات، بل يجعل النظام قابلاً للتوسع مستقبلاً لإضافة استراتيجيات جديدة دون المساس ببنيته الأساسية.

الفصل

4

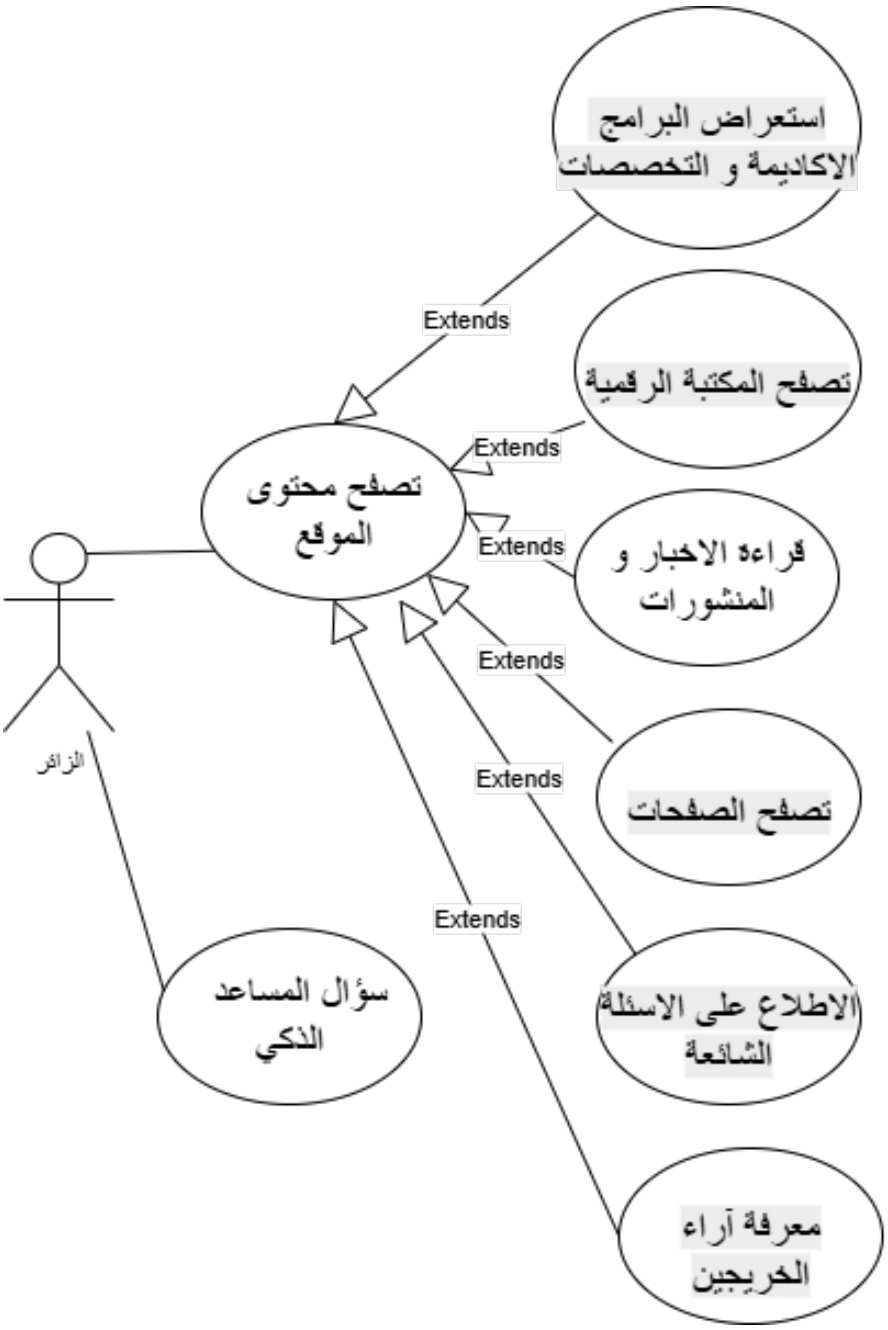
الدراسة التحليلية

في هذا الفصل، نقوم بتحليل متطلبات النظام بشكل منهجي لتحديد الوظائف التي يجب أن يقدمها والجهات الفاعلة التي تتفاعل معه. يتم استخدام مخططات حالات الاستخدام لتوفير تمثيل مرئي وواضح لهذه الوظائف والعلاقات بينها.

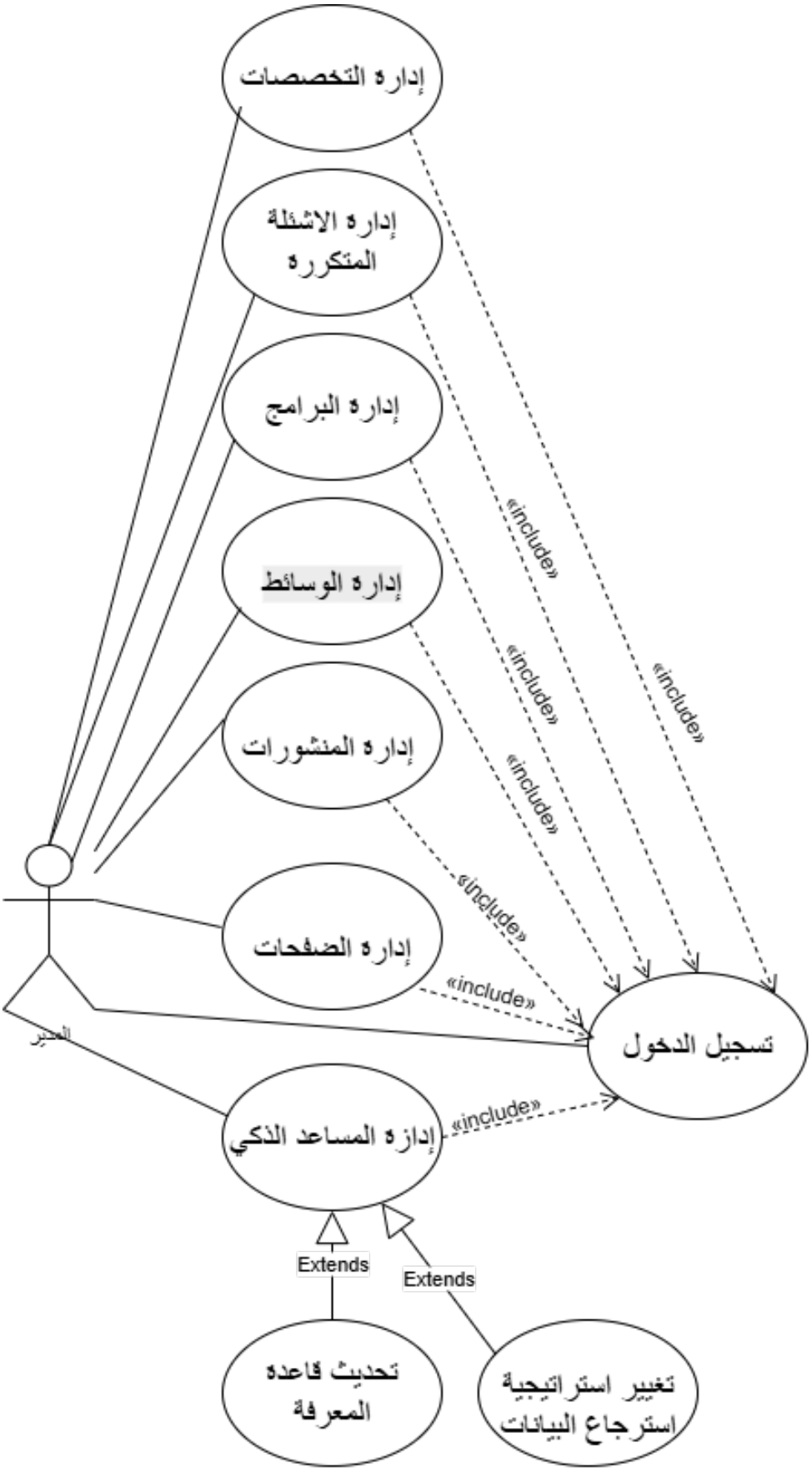
1.4 مخططات حالات الاستخدام (Use Case Diagrams)

تم تحديد فاعلين رئيسيين للنظام: **الزائر** (المستخدم العام للموقع) **والمدير** (المسؤول عن إدارة المحتوى). توضح المخططات التالية حالات الاستخدام لكل فاعل.

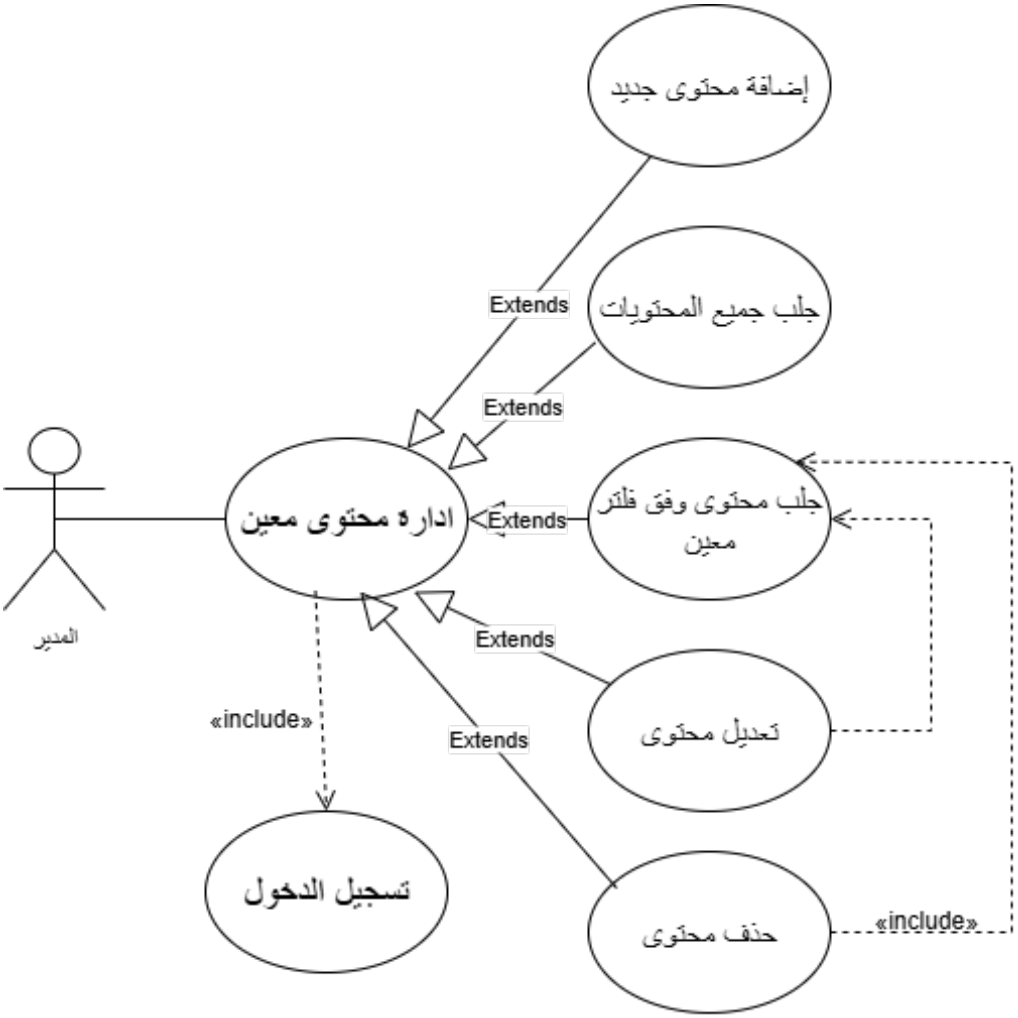
يوضح المخطط القادم ما المقصود ب"إدارة" لم يتم اضافته في المخطط السابق لكي لا يتعقد المخطط.



الشكل 1.4: مخطط حالات الاستخدام الخاصة بالزائر.



الشكل 2.4: مخطط حالات الاستخدام الرئيسية الخاصة بالمدير.



الشكل 3.4: مخطط حالات المعلقة بأدارة محتوى ما في الموقع من قبل المدير.

2.4 السرد النصي لحالات الاستخدام (Use Case Scenarios)

في هذا القسم، يتم تفصيل حالات الاستخدام الرئيسية التي تم تحديدها في المخططات السابقة. يتم استخدام جداول موحدة لوصف كل حالة استخدام وسيناريو النجاح الأساسي لها، مما يوضح التفاعل خطوة بخطوة بين الفاعل والنظام.

1.2.4 حالات استخدام الزائر

1.1.2.4 حالة استخدام: استعراض الكتب في المكتبة الرقمية

جدول حالة الاستخدام: استعراض الكتب	
الوصف	**يقوم الزائر بتصفح قائمة الكتب المتاحة في المكتبة الرقمية.**
الفاعلون	الزائر
الشروط المسبقة	لا يوجد شرط مسبق.
الشروط اللاحقة	يتم عرض قائمة الكتب للزائر.

السيناريو الأساسي للنجاح:

الزائر	النظام
1. يختار قسم "المكتبة الرقمية".	
	2. يعرض قائمة الكتب.

المسارات البديلة: لا يوجد.

مسارات الأخطاء: في حال كان الغرض الذي يبحث عنه الزائر غير موجود، يعرض النظام رسالة تفيد بذلك.

ملاحظة: تتبع بقية حالات تصفح الموقع نفس السيناريو.

2.1.2.4 حالة استخدام: التفاعل مع المساعد الذكي

جدول حالة الاستخدام: التفاعل مع المساعد الذكي	
الوصف	**يقوم الزائر بطرح سؤال على التشات بوت.**
الفاعلون	الزائر
الشروط المسبقة	واجهة التشات بوت متاحة.
الشروط اللاحقة	يتلقى الزائر إجابة.

السيناريو الأساسي للنجاح:

الزائر	النظام
1. يكتب سؤالاً ويرسله.	
	2. يتم استقبال السؤال. 3. تحليل السؤال و توليد الإجابة. 4. عرض الإجابة للزائر.

المسارات البديلة: لا يوجد.

مسارات الأخطاء: في حال لم يتمكن النظام من توليد إجابة واضحة بسبب عدم توفر معلومات كافية، يعرض رسالة اعتذار للمستخدم انه لا يملك المعلومات الكافية للاجابة على سؤال المستخدم.

2.2.4 حالات استخدام المدير

1.2.2.4 حالة استخدام: إنشاء منشور جديد

جدول حالة الاستخدام: إنشاء منشور جديد	
الوصف	**يقوم المدير بإضافة خبر أو منشور جديد إلى الموقع.**
الفاعلون	المدير
الشروط المسبقة	المدير قام بتسجيل الدخول بنجاح.
الشروط اللاحقة	يتم حفظ المنشور الجديد.

السيناريو الأساسي للنجاح:

المدير	النظام
1. يختار "إضافة منشور جديد".	
	2. يعرض نموذج الإدخال.
3. يملأ الحقول المطلوبة ويضغط "حفظ".	
	4. يتحقق من صحة البيانات ويحفظها في قاعدة البيانات. 5. يعرض رسالة نجاح.

المسارات البديلة: لا يوجد.

مسارات الأخطاء: في حال قام المدير بإرسال النموذج مع ترك حقول مطلوبة فارغة (مثل العنوان)، يقوم النظام بعرض رسالة خطأ بجانب الحقل المطلوب ولا يتم حفظ المنشور.

ملاحظة: تتبع عمليات إنشاء محتوى جديد لبقية مكونات الموقع نفس بنية السيناريو.

2.2.2.4 حالة استخدام: تحديث منشور موجود

جدول حالة الاستخدام: تحديث منشور	
الوصف	**يقوم المدير بتعديل بيانات منشور موجود مسبقًا.**
الفاعلون	المدير
الشروط المسبقة	المدير قام بتسجيل الدخول، والمنشور المراد تعديله موجود في النظام.
الشروط اللاحقة	يتم تحديث بيانات المنشور.

السيناريو الأساسي للنجاح:

المدير	النظام
1. يتصفح قائمة المنشورات ويضغط على زر "تعديل" بجانب المنشور المطلوب.	
	2. يعرض نموذج التعديل مع ملء الحقول بالبيانات الحالية للمنشور.
3. يقوم بتغيير البيانات المطلوبة في النموذج ويضغط "حفظ التعديلات".	
	4. يتحقق من صحة البيانات الجديدة ويقوم بتحديثها . 5. يعرض رسالة نجاح: "تم تحديث المنشور بنجاح".

المسارات البديلة: لا يوجد.

مسارات الأخطاء:

- في حال ترك حقول مطلوبة فارغة، يعرض النظام رسالة خطأ ولا يتم حفظ التعديلات.
- في حال كان المنشور قد تم حذفه من قبل مدير آخر، يعرض النظام رسالة تفيد بأن المنشور لم يعد موجودًا.

3.2.2.4 حالة استخدام: حذف منشور

جدول حالة الاستخدام: حذف منشور	
الوصف	**يقوم المدير بحذف منشور بشكل نهائي من الموقع.**
الفاعلون	المدير
الشروط المسبقة	المدير قام بتسجيل الدخول، والمنشور المراد حذفه موجود في النظام.
الشروط اللاحقة	يتم حذف المنشور.

السيناريو الأساسي للنجاح:

المدير	النظام
1. يتصفح قائمة المنشورات ويضغط على زر "حذف" بجانب المنشور المطلوب.	
	2. يعرض نافذة تأكيد تسأل "هل أنت متأكد من رغبتك في الحذف؟".
3. يضغط على زر "تأكيد الحذف".	
	4. يقوم بحذف المنشور. 5. يعرض رسالة نجاح: "تم حذف المنشور بنجاح".

المسارات البديلة: يقوم المدير بالضغط على زر "إلغاء" في نافذة التأكيد، فيتم إغلاق النافذة ولا يتم حذف المنشور.

مسارات الأخطاء: في حال كان المنشور قد تم حذفه بالفعل، يعرض النظام رسالة خطأ تفيد بأن المنشور لم يعد موجودًا.

4.2.2.4 حالة استخدام: إعادة تدريب قاعدة المعرفة

السيناريو الأساسي للنجاح:

المدير	النظام
1. يضغط زر "بدء التجريف" لجمع البيانات من الموقع.	
	2. يقوم النظام بتشغيل خوارزمية التجريف ويجمع المحتوى من كافة صفحات الموقع. 3. يعرض رسالة نجاح عملية جمع المعلومات.
3. يضغط زر "إعادة تدريب قاعدة المعرفة".	
	4. يقرأ النظام البيانات المجمعة، ويقوم بتقسيمها وتضمينها لبناء قاعدة المعرفة الجديدة. 5. يعرض رسالة نجاح: "تم تحديث قاعدة المعرفة بنجاح".

المسارات البديلة: لا يوجد.

مسارات الأخطاء:

- **خطأ في التجريف:** في حال فشلت عملية التجريف (مثلاً، الموقع المستهدف غير متاح)، يعرض النظام رسالة خطأ للمدير تفيد بفشل عملية جمع البيانات.
- **خطأ في إعادة التدريب:** في حال فشلت عملية إعادة التدريب (مثلاً، ملف البيانات فارغ أو تالف)، يعرض النظام رسالة خطأ تفيد بفشل بناء قاعدة المعرفة.

5.2.2.4 حالة استخدام: تغيير استراتيجية الاسترجاع للمساعد الذكي

جدول حالة الاستخدام: تغيير استراتيجية الاسترجاع	
الوصف	**يقوم المدير بتغيير خوارزمية الإجابة التي يستخدمها المساعد الذكي.**
الفاعلون	المدير
الشروط المسبقة	المدير قام بتسجيل الدخول بنجاح.
الشروط اللاحقة	يتم تغيير استراتيجية الاسترجاع النشطة في النظام.

السيناريو الأساسي للنجاح:

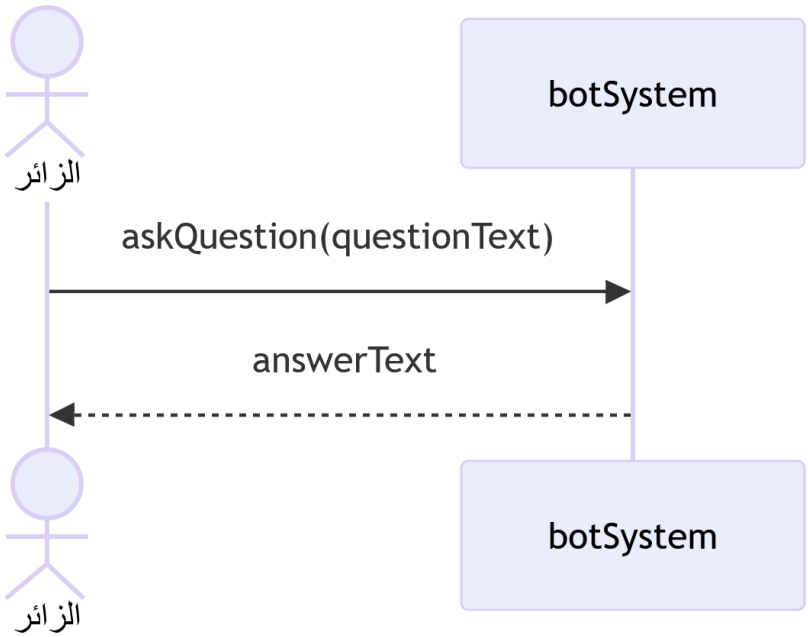
المدير	النظام
1. ينتقل إلى قسم إدارة المساعد الذكي.	
	2. يعرض الاستراتيجية النشطة حاليًا.
3. يختار استراتيجية جديدة من القائمة و يضغط "حفظ".	
	4. يتم تحديث الاستراتيجية النشطة. 5. يعرض رسالة نجاح: "تم تغيير الاستراتيجية بنجاح".

المسارات البديلة: لا يوجد.

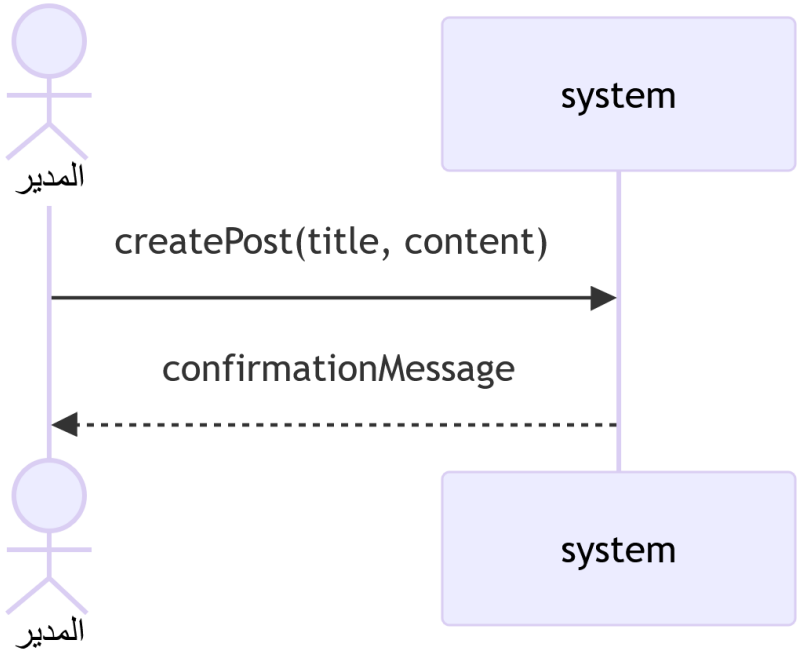
مسارات الأخطاء: في حال فشل النظام في الاتصال بالخدمة الخلفية لتحديث الاستراتيجية، يعرض رسالة خطأ للمدير تفيد بفشل العملية، وتبقى الاستراتيجية القديمة هي النشطة.

3.4 مخططات تسلسل النظام (System Sequence Diagrams - SSD)

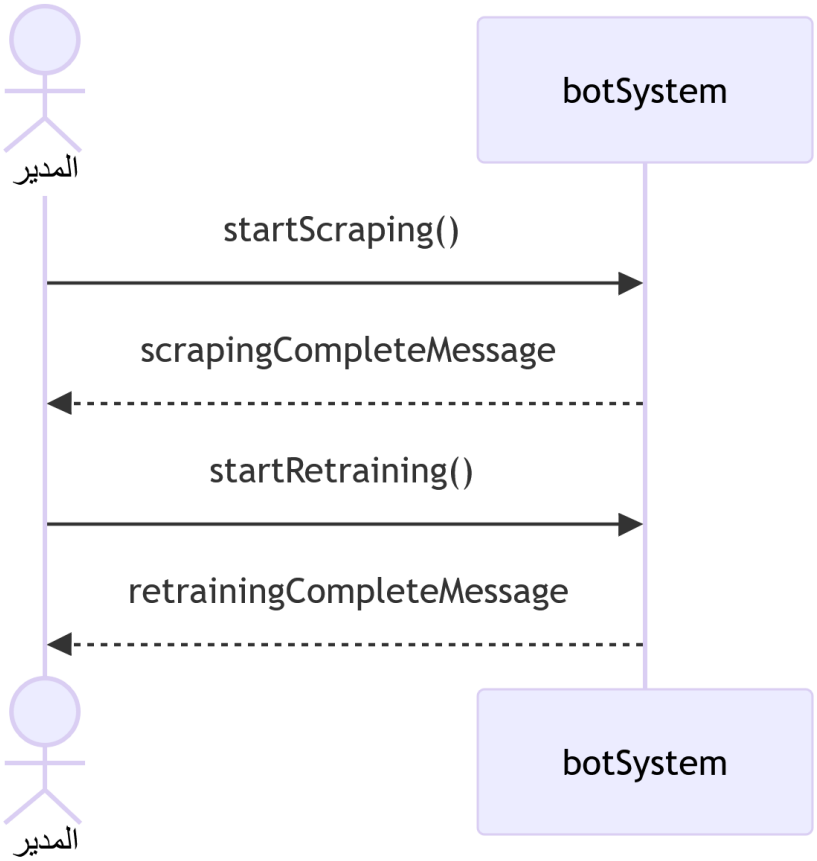
مخطط تسلسل النظام (SSD) هو أداة تحليلية توضح التفاعلات بين الفاعلين الخارجيين والنظام ككل، الذي يُعامل كصندوق أسود. يركز المخطط على تحديد العمليات التي يطلقها الفاعل والعمليات التي يجب على النظام توفيرها للرد على هذه الأحداث. توضح المخططات التالية السيناريوهات الرئيسية للتفاعل مع النظام.



الشكل 4.4: مخطط تسلسل النظام (SSD) لسيناريو "التفاعل مع المساعد الذكي".



الشكل 5.4: مخطط تسلسل النظام (SSD) لسيناريو "إنشاء منشور جديد".

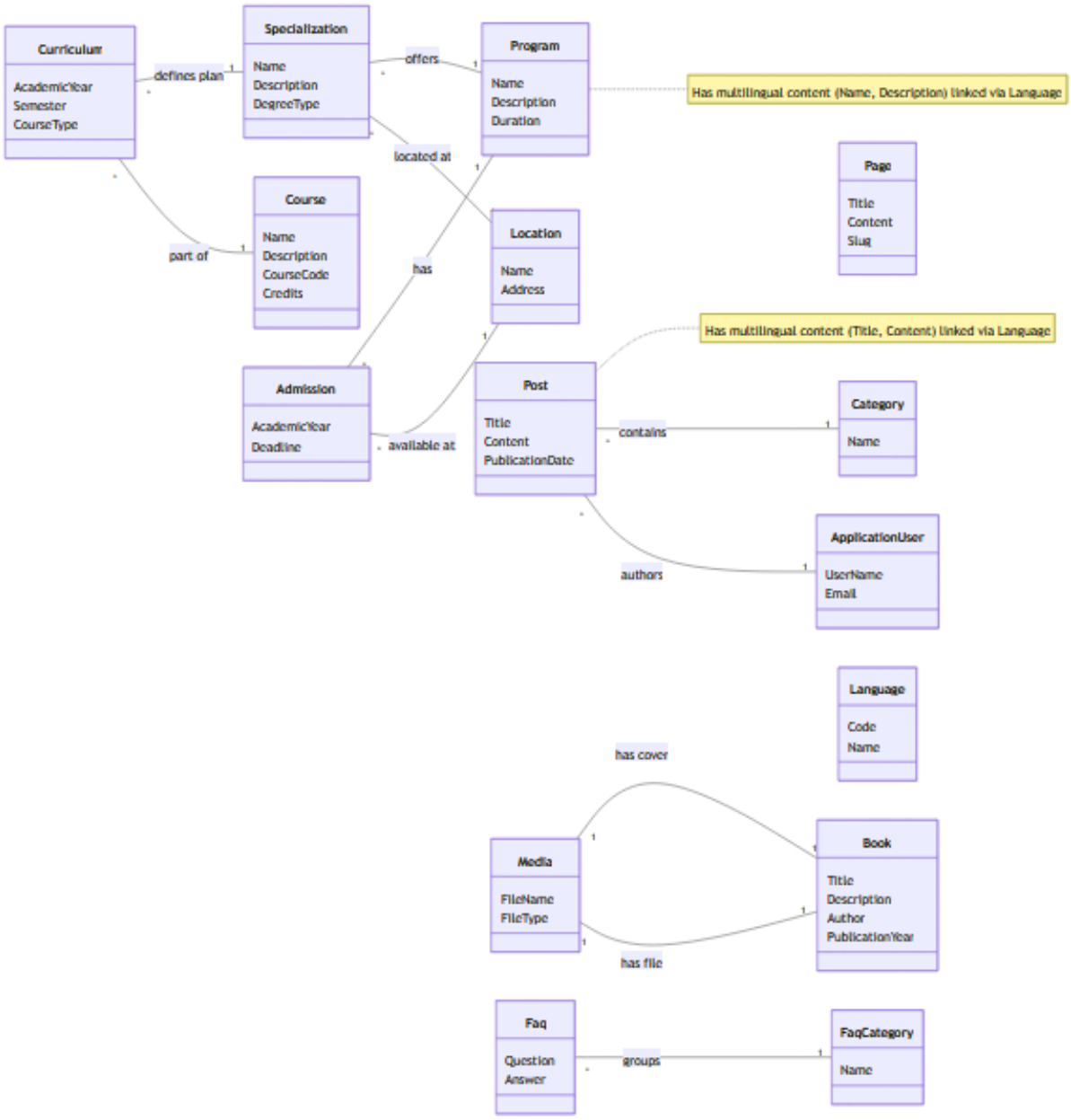


الشكل 6.4: مخطط تسلسل النظام (SSD) لسيناريو "تحديث قاعدة المعرفة" على مرحلتين.

4.4 مخطط نموذج المجال (Domain Model)

بعد تحليل المتطلبات وحالات الاستخدام، تم استخلاص المفاهيم والكيانات الأساسية التي تشكل جوهر النظام. يمثل مخطط نموذج المجال (Domain Model) في الشكل 7.4 تمثيلاً مرئياً ومجرداً لهذه الكيانات، ويوضح السمات الرئيسية لكل منها والعلاقات المنطقية التي تربطها ببعضها البعض.

يركز هذا المخطط على "ماذا" يفعله النظام من منظور الأعمال، متجاهلاً التفاصيل التقنية للتنفيذ أو البنية الفيزيائية لقاعدة البيانات. إنه يوضح المفاهيم الأساسية مثل `Program`, `Specialization`, `Post`، و `Book` وكيفية ارتباطها ببعضها من خلال علاقات مثل "يحتوي على" أو "يتبع لـ". يعتبر هذا المخطط حجر الأساس الذي تم بناء تصميم الكود وقاعدة البيانات عليه في المراحل اللاحقة.



الشكل 7.4: مخطط نموذج المجال (Domain Model) للكيانات الأساسية في النظام.

الفصل

5

الأدوات والتقنيات

يعتمد نجاح أي مشروع برمجي بشكل كبير على اختيار المجموعة المناسبة من الأدوات وأطر العمل (Technology Stack). في هذا الفصل، نستعرض التقنيات الرئيسية التي تم اختيارها لبناء كل مكون من مكونات النظام الثلاثة، مع تبرير أسباب هذا الاختيار بناءً على متطلبات المشروع.

1.5 تقنيات التطبيق الخلفي الأساسي (Backend-Web)

تم بناء التطبيق الخلفي الأساسي، المسؤول عن إدارة منطق العمل والبيانات، باستخدام منظومة Microsoft .NET المعروفة بقوتها واستقرارها في بناء التطبيقات على مستوى المؤسسات.

1.1.5 إطار العمل ومنصة التشغيل: .NET 8.0

تم اختيار .NET 8 [6] كمنصة أساسية للمشروع لعدة أسباب، أبرزها الأداء العالي الذي يوفره، ودعمه القوي للبرمجة غير المتزامنة (Asynchronous Programming) وهو أمر حاسم لتطبيقات الويب عالية الأداء. كما يوفر بيئة تطوير متكاملة وغنية تدعمها Microsoft، ونظاماً قوياً لإدارة الحزم (NuGet)، بالإضافة إلى كونه الخيار المعتمد ضمن المكدس التقني (Technology Stack) للمعهد العالي.

2.1.5 واجهة برمجة التطبيقات ASP.NET Core Web API

لبناء واجهات API قوية ومنظمة، تم استخدام ASP.NET Core [7]، وهو جزء من منظومة .NET. تم اختياره لقدرته على بناء خدمات ويب سريعة، وآمنة، وقابلة للتطوير. يتميز بدعمه المدمج لآلية حقن التبعيات (Dependency Injection)، مما يسهل تطبيق البنية النظيفة، بالإضافة إلى تكامله السلس مع بقية مكونات .NET.

3.1.5 Entity Framework Core

هو إطار عمل للربط بين الكائنات وقواعد البيانات (Object-Relational Mapper - ORM) خاص بـ .NET. [8]. تم اختياره لأنه يبسط بشكل كبير عمليات التعامل مع قاعدة البيانات، حيث يسمح للمطورين بالتعامل مع البيانات باستخدام كائنات C# بدلاً من كتابة استعلامات SQL بشكل مباشر. هذا يسرع من عملية التطوير ويقلل من الأخطاء، كما يسهل تطبيق نمط المستودع (Repository Pattern).

4.1.5 نظام إدارة قواعد البيانات Microsoft SQL Server

لتخزين البيانات العلائقية المنظمة الخاصة بالموقع (مثل بيانات المستخدمين، البرامج الأكاديمية، والمنشورات)، تم اختيار نظام إدارة قواعد البيانات Microsoft SQL Server. تم اتخاذ هذا القرار بناءً على توافقه الممتاز وتكامله العميق مع منظومة .NET. وإطار عمل Entity Framework Core، مما يضمن أداءً عالياً وموثوقية في إدارة البيانات. كما أنه يعتبر الخيار القياسي في العديد من بيئات المؤسسات.

5.1.5 تطبيق نمط الوسيط MediatR

هي مكتبة شائعة لتطبيق نمط الوسيط (Mediator Pattern) في تطبيقات .NET.. تم اختيارها لتكون الأداة التنفيذية لنمط CQRS. تسهل MediatR عملية فصل الطلبات عن معالجاتها، مما ينتج عنه كود أكثر نظافة وتنظيماً، حيث يتم حصر كل منطق عمل في معالج (Handler) خاص به، وهو ما يتوافق تماماً مع أهداف البنية النظيفة.

6.1.5 المصادقة والتفويض ASP.NET Core Identity & JWT

لتأمين النظام، تم الاعتماد على ASP.NET Core Identity لإدارة المستخدمين والأدوار وكلمات المرور. تم اختيار هذه المنصة لأنها توفر حلاً متكاملاً وآمناً وجاهزاً للاستخدام. ولتأمين واجهات API، تم استخدام معيار JSON Web Tokens (JWT)، الذي يسمح بإنشاء "توكن" (Token) مشفر يتم تبادله بين الواجهة الأمامية والخلفية لإثبات هوية المستخدم وصلاحياته مع كل طلب.

2.5 التقنيات المستخدمة لبناء قسم الذكاء الاصطناعي

تم بناء خدمة الذكاء الاصطناعي باستخدام منظومة بايثون (Python)، والتي تعتبر اللغة الرائدة في مجال علم البيانات والذكاء الاصطناعي بفضل مكتباتها الغنية والمتخصصة.

1.2.5 إطار العمل FastAPI (Python)

تم اختيار FastAPI [9] لبناء واجهات API الخاصة بخدمة الذكاء الاصطناعي بفضل أدائه العالي جداً، والذي يجعله من أسرع أطر عمل الويب في Python. كما يتميز بسهولة الاستخدام ودعمه المدمج للبرمجة غير المتزامنة، وقدرته على توليد توثيق تفاعلي لواجهات API بشكل تلقائي، مما يسرع من عملية التطوير والاختبار.

2.2.5 مكتبات الذكاء الاصطناعي LangChain, Google Generative AI

LangChain [10] هي مكتبة متخصصة تعمل كإطار عمل لتنسيق وبناء التطبيقات التي تعتمد على نماذج اللغة الكبيرة. تم اختيارها لأنها تبسط بشكل كبير عملية بناء خطوط أنابيب RAG المعقدة. تم استخدام مكتبة Google Generative AI [14] للتفاعل المباشر مع نماذج جوجل (مثل Gemini) لتوليد التضمينات والإجابات.

3.2.5 قاعدة البيانات المتجهة ChromaDB

هي قاعدة بيانات متجهة مفتوحة المصدر تم تصميمها خصيصاً لتخزين وإدارة التضمينات (Embeddings) [11]. تم اختيار ChromaDB لسهولة إعدادها وتشغيلها محلياً، وتكاملها الممتاز مع مكتبة LangChain، مما يجعلها خياراً مثالياً للمشاريع التي تحتاج إلى بناء سريع لنظام RAG.

4.2.5 مكتبات تجريف الويب BeautifulSoup & Requests

لتغذية قاعدة المعرفة، تم استخدام مكتبة Requests لإرسال طلبات HTTP وجلب محتوى صفحات الويب، ومكتبة BeautifulSoup [15] لتحليل محتوى HTML المستلم واستخلاص النصوص النظيفة منه. تم اختيار هاتين المكتبتين لأنهما تعتبران المعيار الصناعي في Python لعمليات تجريف الويب، وتتميزان ببساطتهما وقوتهما.

3.5 تقنيات الواجهات الأمامية (Frontend-Web)

1.3.5 إطار العمل والمكتبة: Next.js & React

تم اختيار React [13] كمكتبة أساسية لبناء واجهات المستخدم بفضل بنيتها القائمة على المكونات. تم استخدام إطار العمل Next.js [12] فوق React للاستفادة من ميزاته المتقدمة مثل التوجيه المعتمد على نظام الملفات وقدرات التصيير من جانب الخادم، مما يحسن من أداء الموقع وتجربة المستخدم.

2.3.5 إدارة الحالة الدولية (i18n): React Context

لدعم اللغتين العربية والإنجليزية، تم استخدام React Context، وهي آلية مدمجة في React لمشاركة الحالات (State) بين المكونات دون الحاجة إلى تمرير الخصائص (Props) عبر مستويات متعددة. تم اختيار هذه الطريقة لبساطتها وفعاليتها في إدارة الحالة العامة للغة المختارة وتوفيرها لجميع المكونات التي تحتاجها لعرض النصوص المترجمة.

الفصل

6

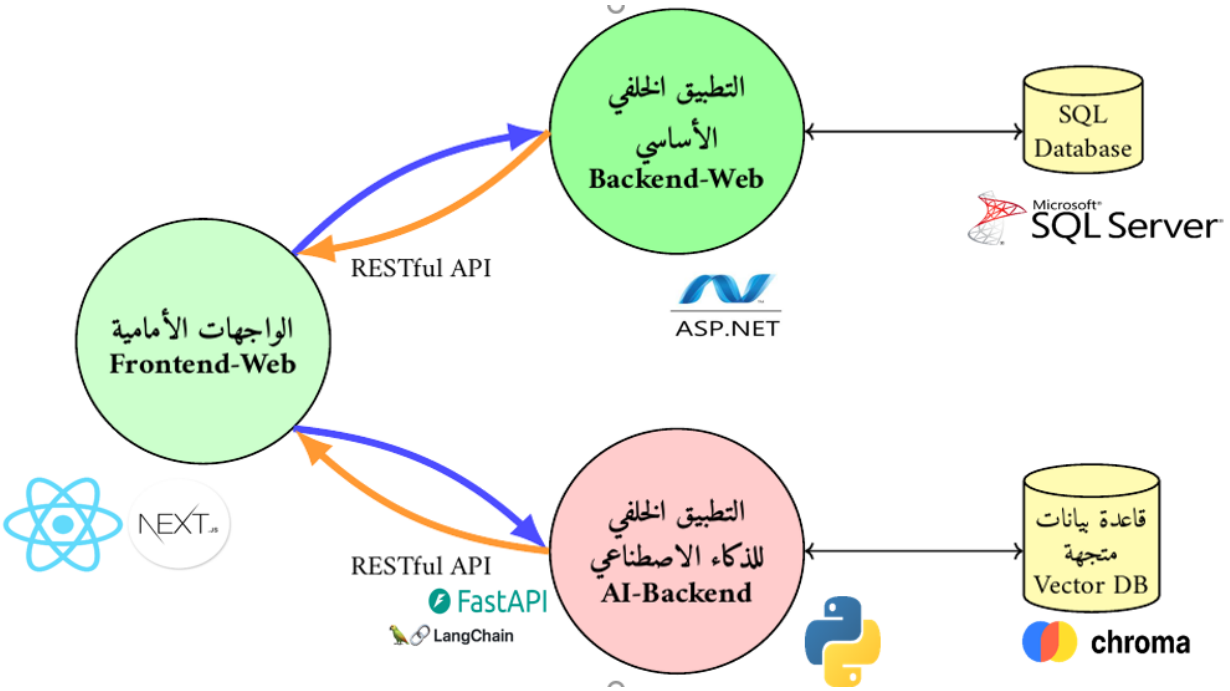
بنية و تصميم النظام

بعد استعراض الأسس النظرية وتحديد الأدوات التقنية، يتناول هذا الفصل كيف تم تصميم النظام. يتم فيه شرح كيفية تصميم كل مكون من مكونات المشروع وتوضيح العلاقات بينها لتحقيق المتطلبات التي تم تحديدها سابقاً.

1.6 البنية المعمارية العامة

تم تصميم النظام بشكل معياري على **(Modular)**, حيث تم تقسيمه إلى ثلاثة أقسام رئيسية و مستقلة تماما لكل منها مسؤولياته و تقنياته الخاصة. تتكون البنية العامة من الخدمات والمكونات التالية، كما هو موضح في الشكل 1.6:

- **الواجهات الأمامية (Frontend-Web):** هو تطبيق العميل الذي يعمل في متصفح المستخدم. تم بناؤه باستخدام React و Next.js، ويعمل كمنسق مركزي يقوم بتوجيه الطلبات إلى الخدمات الخلفية المتخصصة.
- **خدمة إدارة المحتوى (Backend-Web):** هي خدمة مصغرة مسؤولة عن منطق العمل التقليدي. تم بناؤها باستخدام ASP.NET Core مع قاعدة بيانات SQL Server، وتتولى إدارة المحتوى المنظم للموقع والمصادقة.
- **خدمة المحادثة الذكية (AI-Backend):** هي خدمة مصغرة مستقلة تماماً تم تصميمها **كمنتج منفصل وقابل للتوصيل (Pluggable)**. تم بناؤها باستخدام إطار العمل FastAPI بلغة Python مع مكتبة LangChain، وهي قادرة على العمل مع أي موقع ويب عبر تزويدها برابطه فقط، حيث تقوم ببناء قاعدة معرفتها الخاصة والإجابة على الأسئة منها.



الشكل 1.6: البنية المعمارية للنظام، مع توضيح التقنيات المستخدمة في كل خدمة مستقلة.

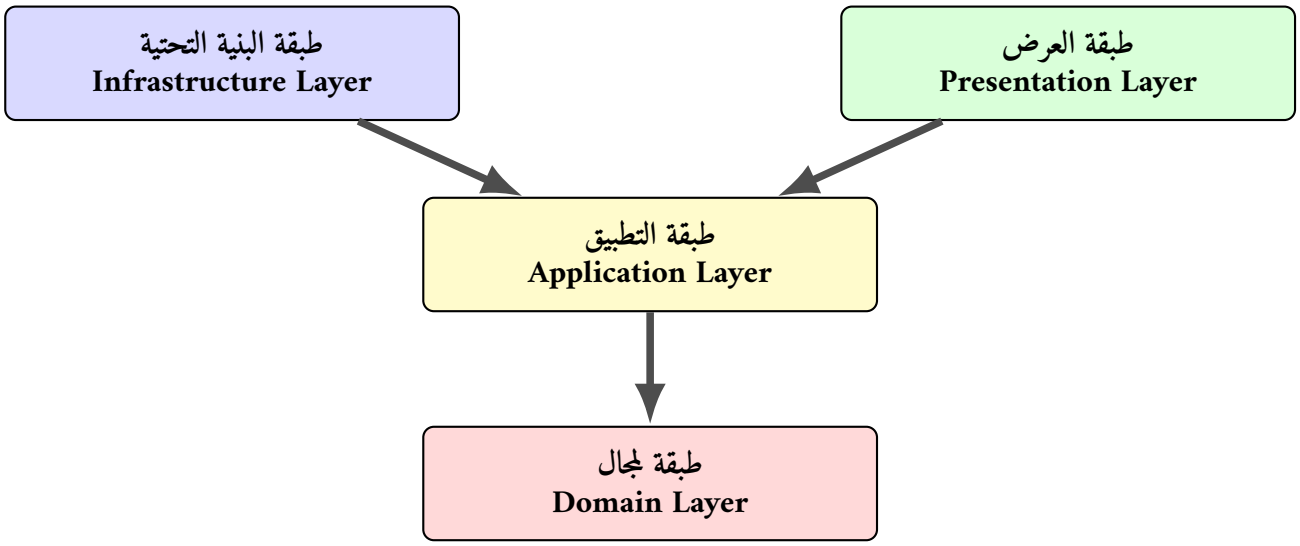
2.6 تصميم التطبيق الخلفي الأساسي (Backend-Web)

تم تصميم هذا القسم ليعمل كنظام إدارة محتوى بدون رأس (Headless CMS) قوي ومرن. يعتمد تصميمه بالكامل على البنية النظيفة لتوفير واجهات API آمنة ومنظمة لإدارة جميع كيانات المحتوى في النظام. نستعرض هنا التصميم الداخلي لكل طبقة من طبقاته التي تحقق هذا الهدف.

1.2.6 طبقات البنية النظيفة

لتحقيق فصل حقيقي للمسؤوليات وفرض قاعدة التبعية بشكل صارم، تم تقسيم الكود البرمجي للتطبيق الخلفي 'Backend-Web' إلى أربعة مشاريع منفصلة في Visual Studio، حيث يمثل كل مشروع طبقة من طبقات البنية النظيفة. يوضح الشكل 2.6 هذه الهيكلية العمودية وتدفق التبعيات بينها.

نستعرض فيما يلي محتويات ومسؤوليات كل طبقة من هذه الطبقات كما تم تنفيذها في المشروع:



الشكل 2.6: تصميم طبقات البنية النظيفة في Backend-Web وتدفق التبعيات.

1.1.2.6 طبقة المجال (Domain)

هي قلب النظام والطبقة الأكثر مركزية، وتحتوي فقط على الكيانات وقواعد العمل الأساسية. مكوناتها الرئيسية هي:

- **الكيانات (Entities):** مثل Book, Course, والتي تحتوي على الخصائص والقواعد الأساسية للعمل.
- **الكيانات الأساسية (Base Classes):** مثل BaseEntity لضمان وجود معرف فريد لكل كيان.

2.1.2.6 طبقة التطبيق (Application)

تعمل هذه الطبقة كمنسق لمنطق العمل (Use Cases) وتعتمد فقط على طبقة المجال. هي التي تحدد "ماذا" يمكن للنظام أن يفعله، دون الاهتمام بـ "كيف" يتم ذلك. مكوناتها الأساسية هي:

- **الواجهات (العقود):** تحتوي على جميع الواجهات (Interfaces) التي تمثل العقود التي يجب أن تلتزم بها الطبقات الخارجية، مثل IAppRepository, IAppUnitOfWork, و IBookService.
- **نماذج نقل البيانات (DTOs):** كائنات بسيطة مثل BookDto تُستخدم لنقل البيانات من وإلى طبقة العرض، مما يمنع تسريب تفاصيل كيانات المجال إلى الخارج.
- **الأوامر والاستعلامات (Commands/Queries):** تمثل كل حالة استخدام كطلب منفصل يتم إرساله عبر الوسيط.
- **المعالجات (Handlers):** تحتوي على التنفيذ الفعلي لمنطق كل حالة استخدام، مما يحقق مبدأ المسؤولية الواحدة.

3.1.2.6 طبقة البنية التحتية (Infrastructure)

هذه الطبقة مسؤولة عن "كيفية" تنفيذ العقود التي تم تعريفها في طبقة التطبيق. هي الطبقة التي تتعامل مع كل التفاصيل التقنية والعالم الخارجي. من أبرز مسؤولياتها:

- **الوصول إلى قاعدة البيانات:** تحتوي على سياق Entity Framework (ApplicationDbContext) والتنفيذ الفعلي لواجهات IAppRepository و IAppUnitOfWork. هي المكان الوحيد الذي يحتوي على استعلامات قاعدة البيانات.
- **تنفيذ الخدمات:** تحتوي على التنفيذ الفعلي للخدمات مثل AuthenticationService التي تتعامل مع JWT، و FileService التي تتعامل مع نظام الملفات.

4.1.2.6 طبقة العرض وتدفق الطلبات (API)

تم تصميم طبقة العرض لتكون بوابة الدخول الآمنة والمنظمة إلى النظام. يتم فيها تنسيق تدفق الطلبات من العالم الخارجي إلى منطق العمل الداخلي.

تدفق الطلب النموذجي: تم تصميم تدفق الطلب ليتبع نمطاً منظماً يضمن فصل المسؤوليات، كما يلي:

1. يستقبل **المتحكم (Controller)** طلباً خارجياً عبر HTTP.
2. يقوم المتحكم باستدعاء التابع المناسب في **الخدمة (Service)** الموافقة (مثل 'IBookService').
3. تقوم **الخدمة** بتنسيق حالة الاستخدام. هي التي تنشئ كائن الأمر (Command) أو الاستعلام (Query) المناسب.
4. ترسل الخدمة هذا الكائن إلى **الوسيط (Mediator)**.
5. يقوم الوسيط بتحديد **المعالج (Handler)** المختص في طبقة التطبيق وتنفيذ منطق العمل الفعلي.

هذا التصميم يجعل المتحكمات بسيطة جداً ومسؤوليتها الوحيدة هي التعامل مع HTTP، بينما تعمل الخدمات كواجهة نظيفة لتنسيق حالات الاستخدام المعقدة.

المكونات الرئيسية الأخرى:

- **هيكلية واجهات API:** تم تنظيم نقاط النهاية (Endpoints) باستخدام المناطق (Areas) لفصل وظائف المدير (/api/Admin) عن وظائف الزائر (/api/User).
- **معالجة الأخطاء الشاملة (Global Exception Handler):** تم استخدام برمجية وسيطة (Middleware) لاعتراض أي خطأ غير معالج في النظام، وتسجيله، وإعادة استجابة HTTP موحدة، مما يمنع تسريب تفاصيل الأخطاء الحساسة.

- **نمط الاستجابة الموحدة:** يتم تطبيق هذا النمط عبر الفئة 'ApiResponse' لضمان اتساق جميع الاستجابات الصادرة من النظام.

3.6 تصميم المجيب الالي (ChatBot)

1.3.6 خوارزمية التجريف العودي (Recursive Web Scraping Algorithm)

تم تصميم آلية جمع البيانات لتكون ديناميكية وقادرة على تغطية الموقع بالكامل. تعتمد الخوارزمية على مبدأ التجريف العودي، حيث تبدأ من رابط أساسي، وتقوم باستخلاص المحتوى النصي منه، ثم تبحث عن جميع الروابط الداخلية في الصفحة وتضيف الجديدة منها إلى قائمة مهام للمعالجة لاحقاً. يتم الاحتفاظ بسجل للروابط التي تمت زيارتها لتجنب التكرار والحلقات اللانهائية.
تتبع الخوارزمية منهجية منظمة يمكن تلخيصها في ثلاث مراحل رئيسية: مرحلة الاكتشاف (Crawling)، مرحلة الاستخلاص (Scraping)، ومرحلة التخزين (Storage). فيما يلي تفصيل لخطوات تنفيذ الخوارزمية:

1.1.3.6 خطوات عمل الخوارزمية

1. **التهيئة والبدء (Initialization):**

- تبدأ العملية من رابط أساسي واحد (Start URL) يتم تزويدها به.
- يتم تهيئة ثلاث قوائم بيانات أساسية لإدارة عملية التجريف:

- **قائمة الروابط قيد الانتظار (Queue of URLs to Visit):** تحتوي في البداية على الرابط الأساسي فقط، وتُستخدم لتخزين الروابط التي سيتم زيارتها في المستوى الحالي من البحث.
- **القائمة الرئيسية للروابط المكتشفة (Master List of Found URLs):** قائمة فريدة تحتفظ بجميع الروابط التي تم العثور عليها حتى الآن لمنع التكرار.
- **سجل الروابط المعالَجة (Record of Processed URLs):** قائمة تحفظ الروابط التي تم تجريفها بالفعل لتجنب إعادة معالجتها والدخول في حلقات لانهائية.

2. **مرحلة الاكتشاف العودي للروابط (Recursive Crawling):**

- تقوم الخوارزمية بالمرور على كل رابط موجود في "قائمة الروابط قيد الانتظار".
- لكل رابط، يتم تحميل محتوى الصفحة الخاصة به والبحث ضمنها عن جميع الروابط الأخرى (الوسوم '<a>').
- يتم التحقق من كل رابط جديد:

- هل ينتمي إلى نفس نطاق الموقع (Same Domain)؟

- هل هو رابط لصفحة HTML وليس ملفاً (مثل pdf. أو zip.)؟
- هل هو رابط جديد لم يتم اكتشافه من قبل (غير موجود في "القائمة الرئيسية")؟

• إذا استوفى الرابط الشروط، يتم إضافته إلى "القائمة الرئيسية" وإلى "قائمة الروابط قيد الانتظار" ليتم معالجته في الجولة التالية.

• تستمر هذه العملية بشكل عودي حتى لا يتبقى روابط جديدة لاكتشافها.

3. **مرحلة استخلاص المحتوى النصي (Content Scraping):**

• بعد اكتمال مرحلة الاكتشاف والحصول على القائمة النهائية لجميع الروابط الفريدة في الموقع، تبدأ مرحلة الاستخلاص.

• تمر الخوارزمية على كل رابط في "القائمة الرئيسية".

• لكل صفحة، يتم تحميل محتوى الـ HTML وتنظيفه عبر إزالة الأجزاء غير المرغوبة التي لا تحتوي على محتوى نصي مفيد، مثل:

- وسوم الأكواد البرمجية (<script>).
- وسوم التنسيق والأنماط (<style>).

• بعد عملية التنظيف، يتم استخلاص النص المتبقي فقط وتوحيد المسافات والفراغات فيه للحصول على نص نظيف ومتجانس.

4. **تجميع البيانات وتخزينها (Data Aggregation and Storage):**

• يتم تجميع كل محتوى نصي نظيف مع الرابط الأصلي الذي تم استخلاصه منه.

• تُحفظ جميع النتائج في ملف JSON واحد منظم يحتوي على قسمين:

- **بيانات وصفية (Metadata):** تشمل الرابط الذي بدأت منه العملية، وتاريخ التجريف، وعدد الروابط التي تم العثور عليها بنجاح.
- **المحتوى المُجرّف (Scraped Content):** قائمة من الكائنات، حيث يمثل كل كائن صفحة واحدة ويحتوي على رابطها ومحتواها النصي النظيف.

بهذه الطريقة، تضمن الخوارزمية الحصول على قاعدة بيانات نصية شاملة ومنظمة من الموقع المستهدف، جاهزة للاستخدام في تدريب وتغذية نموذج روبوت المحادثة.

2.3.6 وحدة الـ Embedder

تم تصميم وحدة الـ 'Embedder' لتكون المحرك المركزي لعملية الفهرسة في نظام RAG. هذه الوحدة مسؤولة عن تنسيق جميع الخطوات اللازمة لتحويل المحتوى النصي إلى قاعدة معرفة متجهة. تشمل مسؤولياتها تحميل البيانات المجرفة، وتقسيمها إلى قطع

(Chunks) باستخدام استراتيجيات مختلفة، واستدعاء نموذج التضمين لتحويلها إلى متجهات، وأخيراً تخزين هذه المتجهات في قاعدة البيانات المتجهة ChromaDB.

اختيار نموذج التضمين (Embedding Model): تم استخدام نموذج **Google Embedding** لحساب التضمين الخاص بالنصوص. تم اختيار هذا النموذج لعدة أسباب استراتيجية وتقنية:

- **التوافقية والأداء:** يتميز النموذج بالتوافق الكامل مع منظومة Google Generative AI، مما يضمن أداءً مستقراً وموثوقاً ضمن بيئة المشروع.
- **الدعم للتعدد اللغوي:** يدعم النموذج مجموعة واسعة من اللغات، بما في ذلك اللغتين العربية والإنجليزية، وهو أمر حاسم لتلبية متطلبات المشروع التي تقتضي دعم اللغتين في المحادثة والمحتوى.
- **الجودة والدقة:** تم تدريب النموذج على كميات هائلة من البيانات، مما يجعله قادراً على توليد تمثيلات متجهة عالية الجودة تلتقط المعاني الدلالية للنصوص بدقة، وهو ما ينعكس إيجاباً على دقة عملية الاسترجاع في نظام RAG.

1.2.3.6 خوارزمية بناء قاعدة البيانات المتجهة

تتبع وحدة الـ Embedder سلسلة من الخطوات المنطقية لإعادة بناء قاعدة البيانات المعرفية من جديد، وهو ما يُعرف بعملية إعادة التدريب (Retraining). يمكن تلخيص هذه الخوارزمية كالتالي:

1. **تحميل البيانات الأولية (Load Data):**
   - تقوم الخوارزمية بقراءة البيانات المجرفة من ملف JSON الناتج عن مرحلة التجريف.
   - يتم استخلاص المحتوى النصي لكل صفحة مع الاحتفاظ برابطها الأصلي كبيانات وصفية (Metadata).
2. **تقسيم البيانات إلى قطع (Chunk Data):**
   - يتم تقسيم المستندات النصية الطويلة إلى قطع أصغر حجماً.
   - تُستخدم استراتيجية التقسيم العودي (RecursiveCharacterTextSplitter) التي تقسم النص بناءً على فواصل محددة (مثل الفقرات، الأسطر، والمسافات) للحفاظ على البنية الدلالية للنص قدر الإمكان و بحجم لا يزيد عن 1200 لحجم القطعة الواحدة.
   - تم تحديد تداخل (Overlap) قدره 200 محرف بين القطع المتتالية لضمان عدم فقدان السياق عند حدود القطع.
3. **إنشاء التضمينات وحفظها (Create and Save Embeddings):**
   - تمر الخوارزمية على كل قطعة نصية ناتجة عن عملية التقسيم.

- لكل قطعة، يتم استدعاء نموذج التضمين (Google Embedding Model) الذي تم ذكره في الدراسة النظرية سابقا لتحويل النص إلى متجه رقمي عالي الأبعاد يمثل معناه الدلالي.
- يتم تخزين كل قطعة نصية مع المتجه الرقمي المقابل لها في قاعدة البيانات المتجهة ChromaDB. هذه العملية تنشئ فهرساً يمكن البحث فيه لاحقاً باستخدام التشابه الدلالي.

بانتهاء هذه الخطوات، تكون قاعدة البيانات المعرفية قد تم بناؤها بالكامل وجاهزة لعمليات الاسترجاع التي تخدم روبوت المحادثة.

3.3.6 تصميم استراتيجيات RAG

لتوفير المرونة والقدرة على التعامل مع أنواع مختلفة من الأسئلة، تم تصميم وتنفيذ ثلاثة استراتيجيات بناءً على نمط التصميم الاستراتيجي (Strategy Pattern).

1.3.3.6 الاستراتيجية الافتراضية (Default RAG Pipeline)

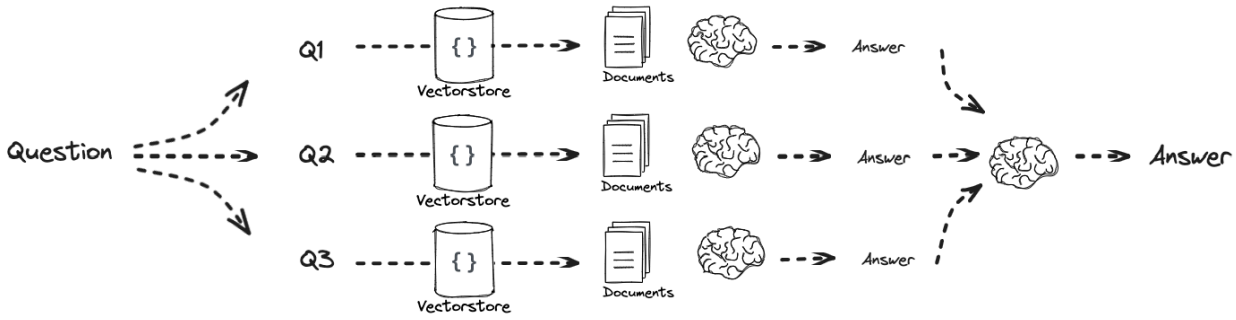
تمثل هذه الاستراتيجية التنفيذ الأساسي والمباشر لخط أنابيب RAG. هي مصممة للتعامل مع الأسئلة البسيطة والواضحة. تتم العملية كالتالي:

1. يتم أخذ سؤال المستخدم كما هو.
2. يتم تحويل السؤال إلى متجه (Embedding).
3. يتم استرجاع أفضل k من القطع النصية المتشابهة من قاعدة البيانات المتجهة.
4. يتم إرسال السؤال مع القطع المسترجعة كسياق إلى النموذج اللغوي لتوليد الإجابة.

هذه الاستراتيجية تتميز بالسرعة والكفاءة للأسئلة المباشرة.

2.3.3.6 استراتيجية تحويل الاستعلام (Query Transformation)

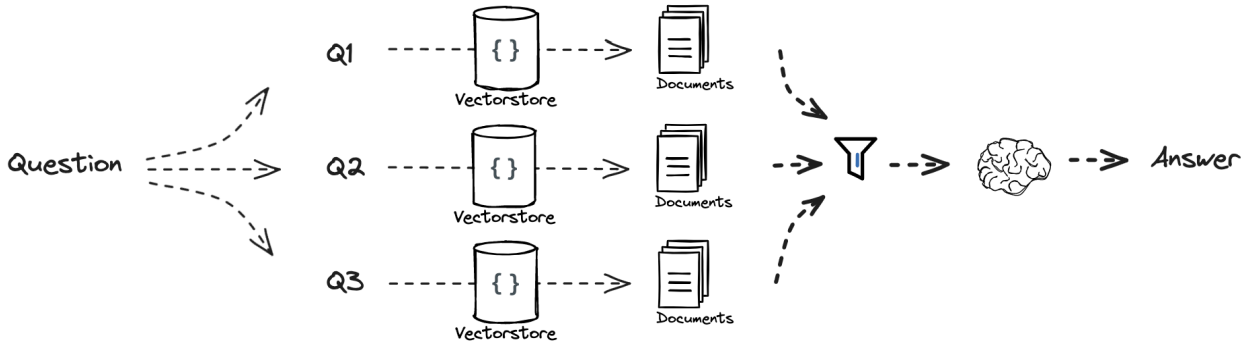
تهدف هذه الاستراتيجية إلى التعامل مع الأسئلة المعقدة عن طريق تفكيكها. كما هو موضح في الشكل 3.6، يتم أولاً استخدام نموذج لغوي لتوليد عدة أسئلة فرعية من السؤال الأصلي. يتم بعد ذلك الحصول على إجابة لكل سؤال فرعي بشكل مستقل، وأخيراً يتم دمج هذه الإجابات المتعددة لتكوين إجابة نهائية وشاملة.



الشكل 3.6: مخطط يوضح آلية عمل استراتيجية تحويل الاستعلام.

3.3.3.6 استراتيجية الدمج (RAG-Fusion)

تمثل هذه الاستراتيجية نهجاً مختلفاً لتحسين دقة الاسترجاع. كما هو موضح في الشكل 4.6، يتم أيضاً توليد استعلامات متعددة، ولكن بدلاً من الإجابة عليها بشكل منفصل، يتم استرجاع المستندات لكل استعلام. بعد ذلك، يتم استخدام خوارزمية Reciprocal Rank Fusion التي تحدثنا عنها في الدراسة المرجعية لدمج قوائم المستندات وإعادة ترتيبها، بحيث تتصدر المستندات الأكثر أهمية وتكراراً القائمة النهائية. أخيراً، يتم استخدام هذه القائمة المدمجة من المستندات لتوليد إجابة واحدة عالية الجودة.



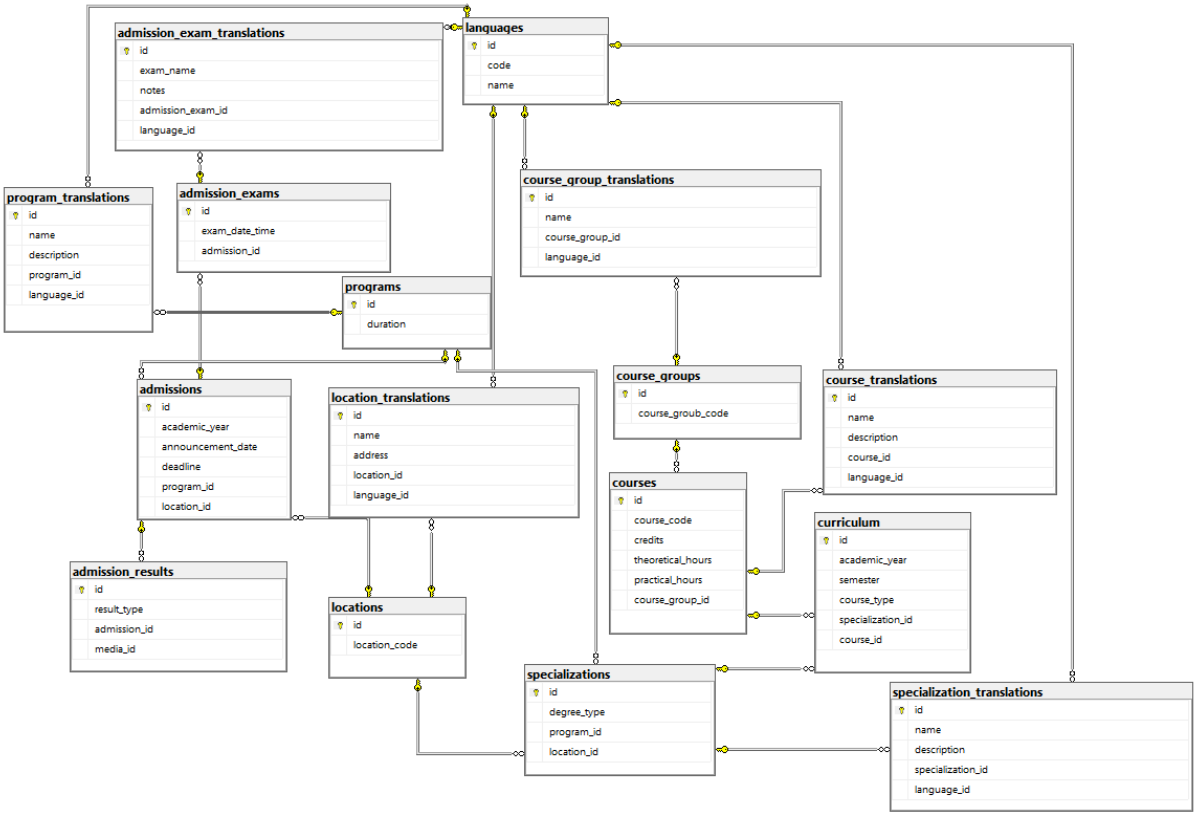
الشكل 4.6: مخطط يوضح آلية عمل خط أنابيب RAG-Fusion.

4.6 تصميم قواعد البيانات

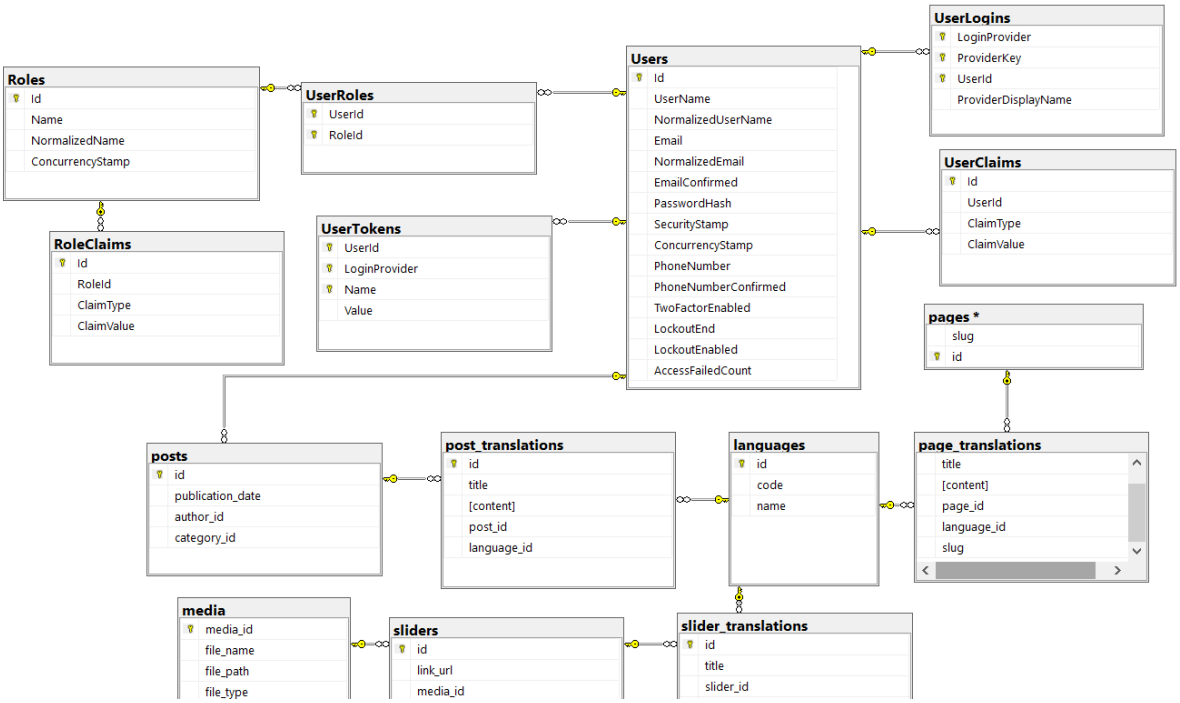
يعتمد النظام على نموذجين مختلفين لتخزين البيانات لتلبية المتطلبات المتباينة لكل قسم خلفي. تم تصميم قاعدة بيانات علائقية لتخزين البيانات المنظمة، وقاعدة بيانات متجهة لتخزين المعرفة غير المنظمة.

1.4.6 تصميم قاعدة البيانات العلائقية (SQL Database)

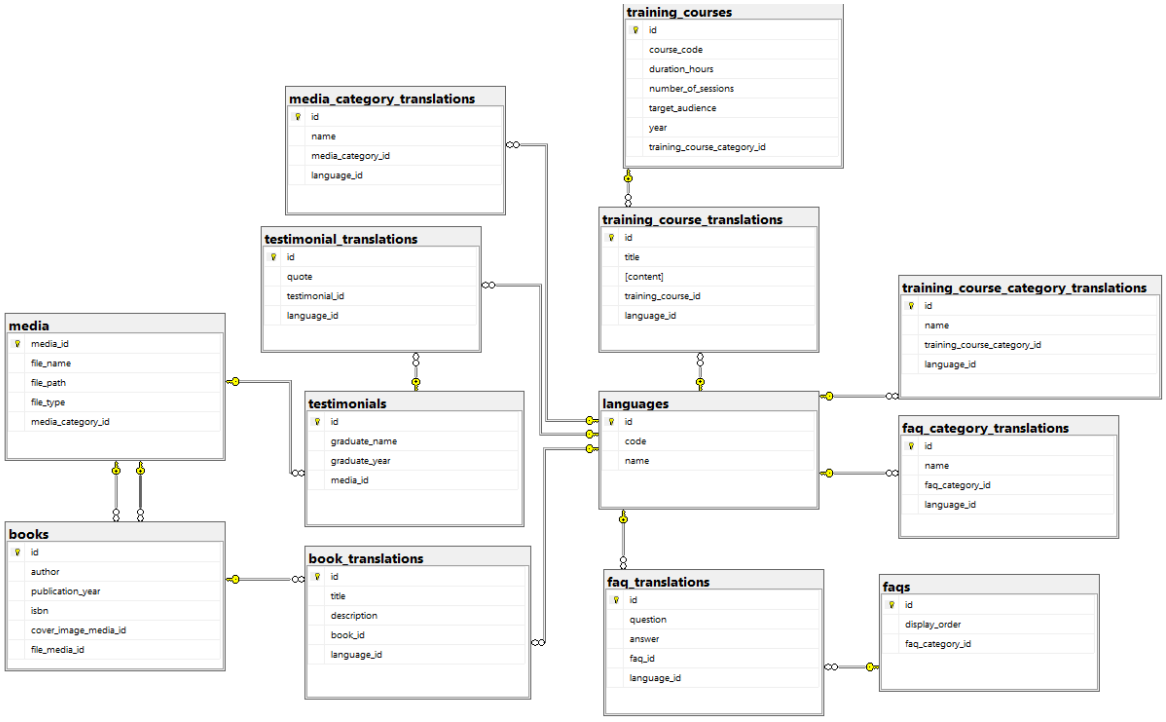
بناءً على مخطط الصفوف المفاهيمي الذي تم تحديده في مرحلة التحليل، تم تصميم المخطط الفيزيائي لقاعدة البيانات العلائقية. تم اتباع نهج **Code-First** باستخدام Entity Framework Core.



الشكل 5.6: مخطط علاقات الكيانات (ERD) الخاص بإدارة البرامج الأكاديمية والتخصصات والقبول.



الشكل 6.6: مخطط علاقات الكيانات (ERD) الخاص بإدارة المستخدمين والصلاحيات، بالإضافة إلى المحتوى العام للموقع مثل المنشورات والصفحات.



الشكل 7.6: مخطط علاقات الكيانات (ERD) الخاص بالبيانات الإضافية مثل الكتب في المكتبة، الأسئلة الشائعة، والدورات التدريبية.

2.4.6 تصميم قاعدة البيانات المتجهة (Vector Database)

تم تصميم قاعدة البيانات المتجهة لتكون بمثابة "ذاكرة" المساعد الذكي. بدلاً من تخزين البيانات في جداول تقليدية، تقوم بتخزين تمثيلات رقمية لمعنى النصوص (التضمينات - Embeddings).

تعتمد البنية على مفهوم "المجموعات" (Collections). كل سجل داخل المجموعة يتكون من المتجه (Embedding)، المحتوى النصي الفعلي (Content)، والبيانات الوصفية (Metadata) مثل رابط الصفحة المصدر.

الفصل 7

تنجيز النظام

1.7 عرض مقتطفات من كود التنفيذ للتطبيق الخلفي (Backend-Web)

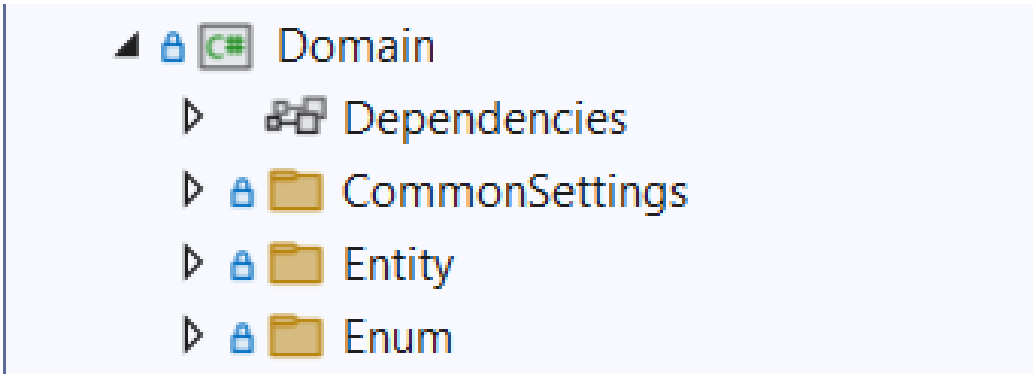
في هذا الفصل، نستعرض كيفية ترجمة البنية المعمارية النظيفة إلى كود برمجي فعلي في تطبيق الواجهة الخلفية Backend-Web. تم تقسيم الكود إلى أربعة مشاريع منطقية، يمثل كل مشروع منها طبقة مستقلة في البنية، مما يضمن فصل المسؤوليات بشكل صارم. نوضح في الأقسام التالية محتوى ومسؤولية كل طبقة، مع الإشارة إلى المخططات التي توضح هيكلية كل مشروع.

1.1.7 طبقة المجال (Domain Layer)

تمثل هذه الطبقة جوهر النظام وتضم الكيانات الأساسية والقواعد الخاصة بها، وهي الطبقة الأكثر استقلالية ولا تعتمد على أي مشروع آخر. يحتوي مشروع ForthYear.Domain على ثلاث مجلدات رئيسية:

- **Entities:** يحتوي على الكيانات الأساسية للمشروع مثل Post.cs و Category.cs، بالإضافة إلى كيانات الترجمة مثل PostTranslation.cs.
- **Enums:** يضم جميع أنواع التعدادات (Enumerations) المستخدمة في النظام.
- **CommonSettings:** يضم ملف إعدادات عامة (DefaultSetting) يحتوي على قيم ثابتة، مثل أسماء الأدوار الافتراضية للمستخدمين وبيانات الاعتماد الخاصة بالمدير الافتراضي.

يوضح الشكل 1.7 هيكلية هذه الطبقة.



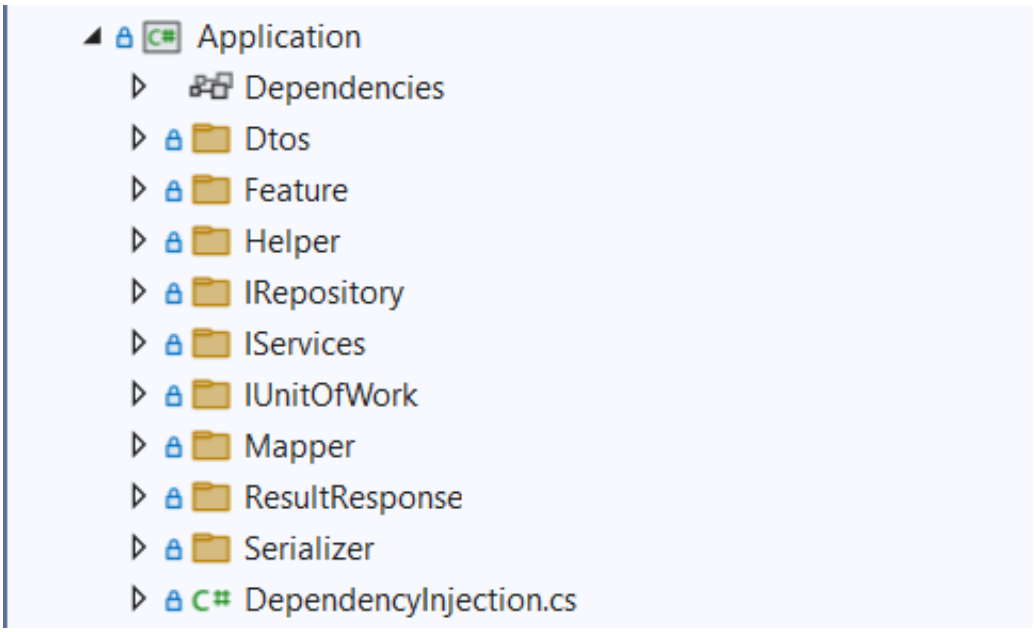
الشكل 1.7: مكونات طبقة المجال (Domain Layer).

2.1.7 طبقة التطبيق (Application Layer)

هذه الطبقة هي العصب المركزي للتطبيق، حيث يتم فيها تنفيذ منطق العمل الحقيقي. لا تحتوي على أي تفاصيل تقنية خارجية، بل تعتمد فقط على طبقة المجال. ينقسم مشروع ForthYear.Application إلى:

- **DTOs:** كائنات نقل البيانات التي تُستخدم للتواصل مع طبقة العرض.
- **IRepository و IUnitOfWork:** واجهات المستودعات ووحدة العمل التي تحدد العقود الخاصة بالوصول للبيانات.
- **Mappers:** ملفات AutoMapper التي تقوم بتحويل الكيانات إلى DTOs والعكس.
- **Features:** هنا يتم تطبيق نمط CQRS، حيث يحتوي كل مجلد على مجلدات فرعية لكل كيان (entity) مثل Post، وكل منها يحتوي على مجلدات للأوامر (Commands) والاستعلامات (Queries) الخاصة به.
- **IService:** واجهات الخدمات التي يتم تنفيذها في الطبقات الخارجية.
- **ResultResponse:** يحتوي على الفئة ApiResponse التي تنفذ نمط Result Pattern لتوحيد استجابات API.
- **DependencyInjection:** ملف يقوم بتكوين AutoMapper و MediatR.

يوضح الشكل 2.7 هيكلية هذه الطبقة.



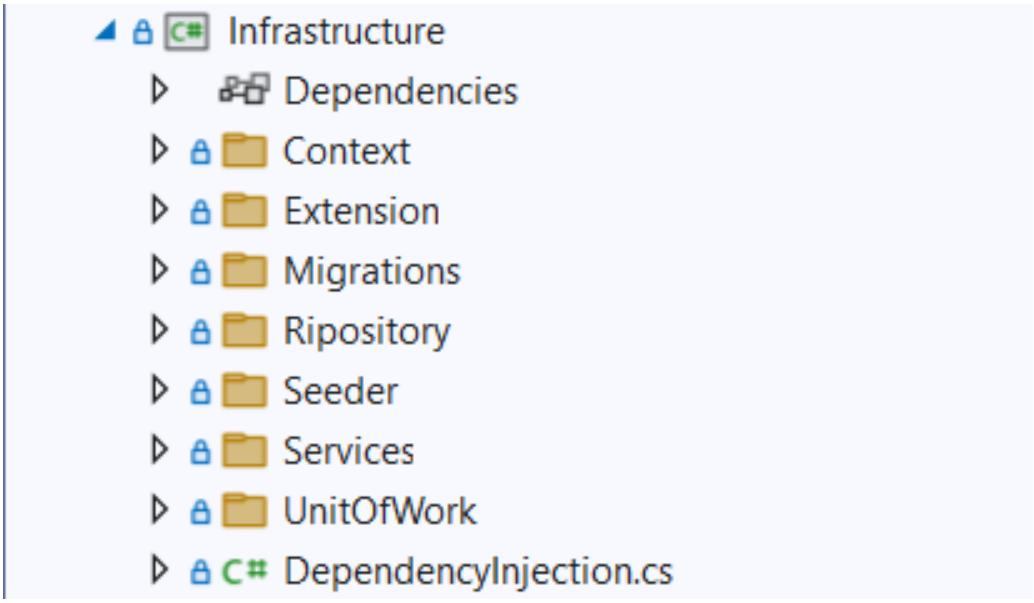
الشكل 2.7: مكونات طبقة التطبيق (Application Layer).

3.1.7 طبقة البنية التحتية (Infrastructure Layer)

هذه الطبقة هي التي تربط تطبيقك بالعالم الخارجي. هي المسؤولة عن التفاصيل التقنية مثل الوصول إلى قاعدة البيانات ونظام الملفات. يحتوي مشروع ForthYear.Infrastructure على:

- **Context:** يضم ملف ApplicationDbContext الذي يمثل سياق قاعدة البيانات.
- **Migrations:** ملفات الترحيل التي تُستخدم لتحديث هيكل قاعدة البيانات.
- **Repository و UnitOfWork:** التنفيذ الفعلي لواجهات المستودعات ووحدة العمل.
- **Service:** التنفيذ الفعلي لخدمات خارجية مثل خدمة المصادقة.
- **Seeder:** ملفات لملء قاعدة البيانات ببيانات أولية عند تشغيل التطبيق لأول مرة.
- **DependencyInjection:** ملف يتم فيه إعداد حقن التبعيات للخدمات وتحديد خيارات الاتصال بقاعدة البيانات.

يوضح الشكل 3.7 هيكلية هذه الطبقة.



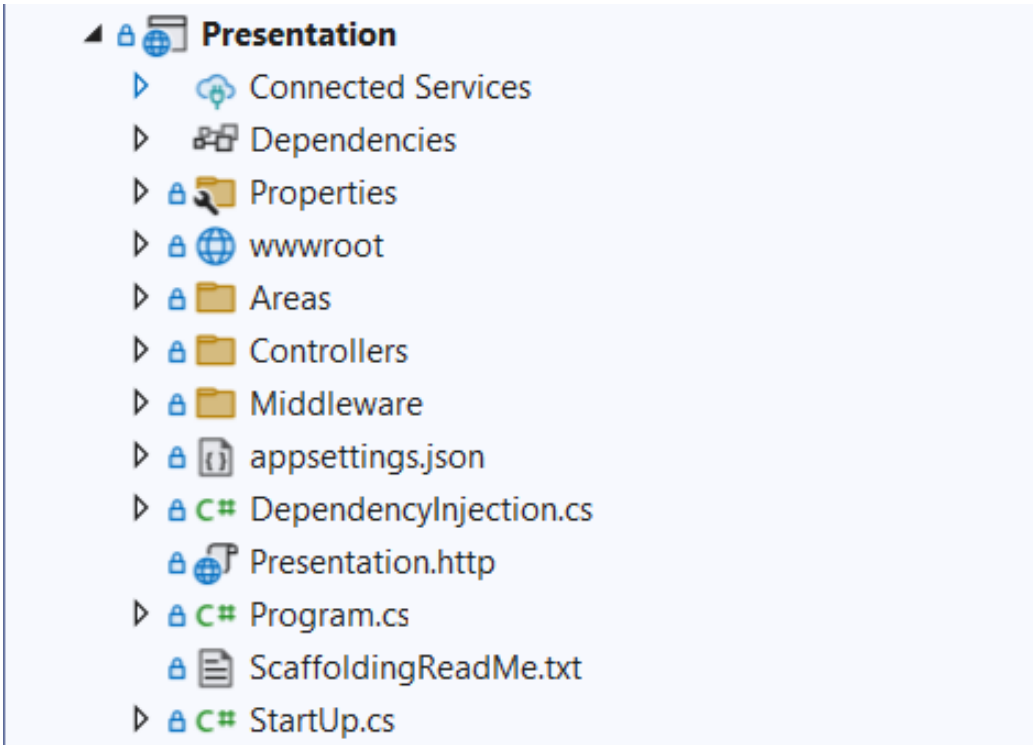
الشكل 3.7: مكونات طبقة البنية التحتية (Infrastructure Layer).

4.1.7 طبقة العرض (Presentation Layer)

طبقة العرض هي نقطة الدخول الرئيسية للتطبيق، وهي مسؤولة عن استقبال طلبات HTTP وتوجيهها إلى طبقة التطبيق. لا تحتوي هذه الطبقة على أي منطق عمل، بل تركز على تنسيق الاتصال. يحتوي مشروع ForthYear.API على:

- **Middleware:** يضم فئة GlobalExceptionHandler التي تتعامل مع الأخطاء غير المعالجة.
- **Program.cs** و **Startup.cs:** ملفات إعدادات وتشغيل التطبيق.
- **Areas:** يضم مجلدات لوحدات التحكم (Controllers) الخاصة بالمدير (Admin) والمستخدم (User).
- **DependencyInjection:** ملف يتم فيه تحديد إعدادات JWT والمصادقة والتفويض.

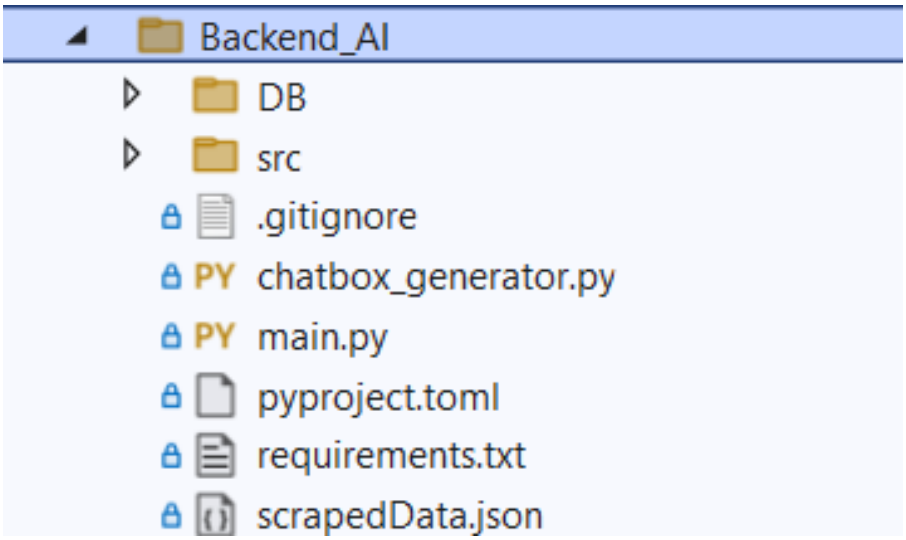
يوضح الشكل 4.7 هيكلية هذه الطبقة.



الشكل 4.7: مكونات طبقة العرض (Presentation Layer).

2.7 عرض مقتطفات من كود التنفيذ للتطبيق الخلفي للذكاء الاصطناعي (AI-Backend)

تم بناء التطبيق الخلفي للذكاء الاصطناعي باستخدام إطار العمل FastAPI بلغة Python، وهو مصمم ليكون خدمة مصغرة مستقلة تمامًا. يوضح الشكل 5.7 الهيكل العام لمجلدات المشروع.



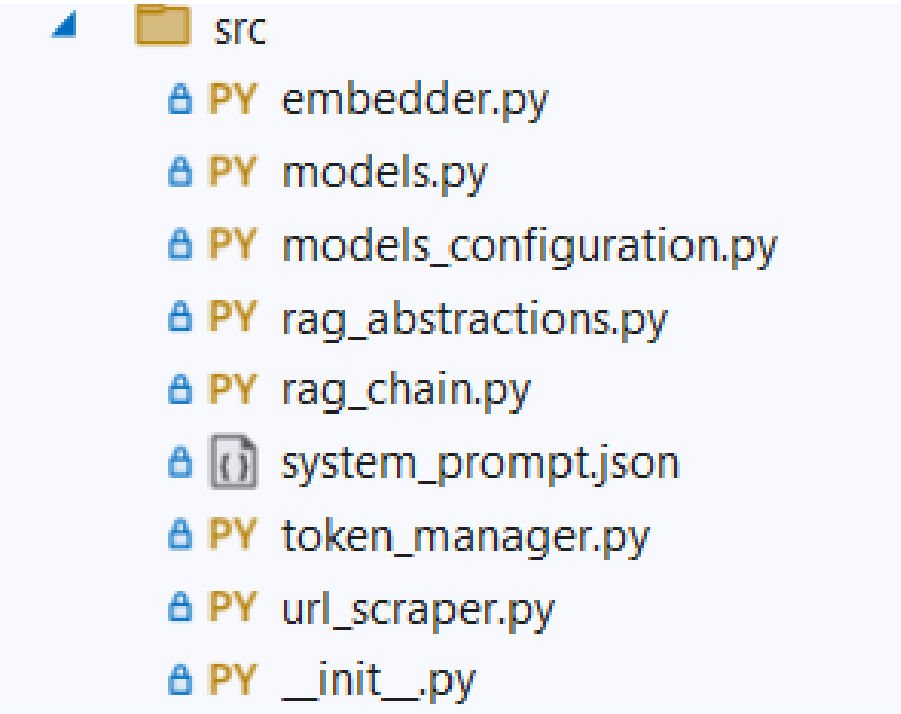
الشكل 5.7: الهيكل العام لمجلدات مشروع التطبيق الخلفي للذكاء الاصطناعي.

تتوزع المسؤوليات الرئيسية في هذا الهيكل كالتالي:

- **main.py:** هو نقطة الدخول الرئيسية للتطبيق، حيث يتم تعريف خادم FastAPI وجميع نقاط النهاية (APIs) التي يستقبل من خلالها الطلبات.
- **chatbox_generator.py:** مكون فريد مسؤول عن توليد كود الواجهة الأمامية (Chatbox) بشكل ديناميكي، والذي يتم حقنه في موقع الويب الخاص بالعميل.
- **مجلد DB:** هو المجلد المخصص لتخزين قواعد البيانات المتجهة (Vector Database) التي يتم إنشاؤها بواسطة ChromaDB.

يحتوي مجلد **src** على المنطق البرمجي الأساسي للتطبيق، والذي تم فصله إلى وحدات متخصصة كما هو موضح في الشكل 6.7. فيما يلي شرح موجز لمسؤولية كل ملف داخل مجلد **src**:

- **url_scraper.py:** يحتوي على الكود الخاص بجارف الويب العودي، المسؤول عن جمع المحتوى النصي من صفحات الويب المستهدفة.
- **embedder.py:** يدير جميع عمليات قاعدة البيانات المتجهة، بما في ذلك تحميل البيانات، تقسيمها إلى قطع، وتضمينها (Embedding).

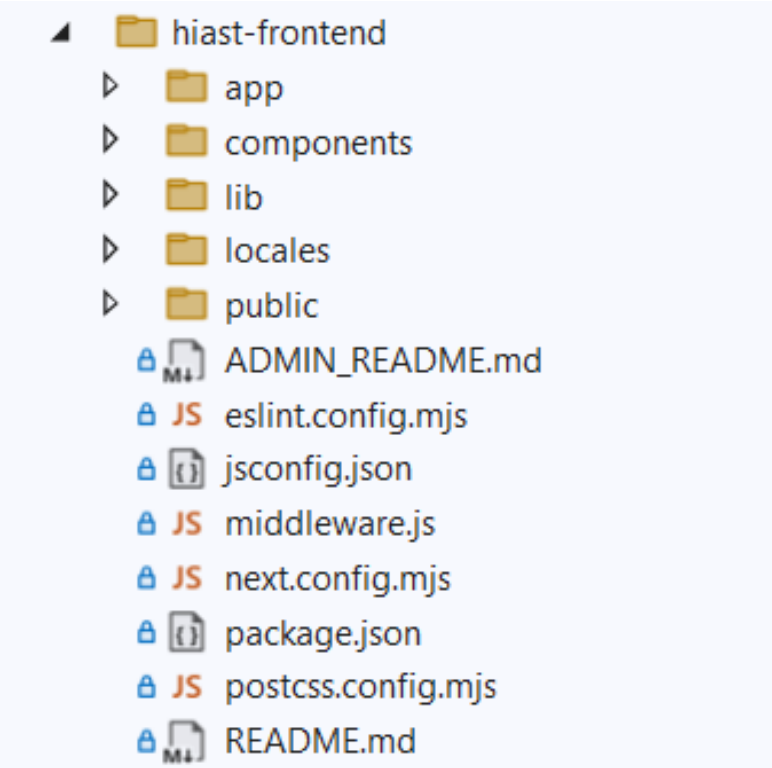


الشكل 6.7: مكونات مجلد src الذي يحتوي على المنطق الأساسي للتطبيق.

- **rag_chain.py:** يضم الوظائف الأساسية لبناء سلسلة RAG باستخدام مكتبة LangChain، وهو مسؤول عن عملية الاسترجاع الأولية.
- **rag_abstractions.py:** يُطبّق نمط التصميم الاستراتيجي (Strategy Pattern) عبر تعريف وتنفيذ استراتيجيات RAG المختلفة (الافتراضية، تحويل الاستعلام، والدمج).
- **token_manager.py:** أداة مساعدة لإدارة طول سياق المحادثة عبر حساب عدد التوكنز وحذف الرسائل القديمة عند تجاوز الحد المسموح.
- **models_configuration.py:** يقوم بعزل إعدادات النماذج اللغوية ونماذج التضمين في مكان واحد لتسهيل تعديلها.
- **models.py:** يُعرّف هياكل البيانات لطلبات واستجابات الـ API باستخدام Pydantic لضمان صحة البيانات.
- **system_prompt.json:** ملف إعدادات خارجي يخزن التعليمات والنصوص التوجيهية للنموذج اللغوي، مما يفصلها عن الكود البرمجي.

3.7 عرض مقتطفات من كود التنفيذ للواجهات الأمامية (Frontend-Web)

تم بناء تطبيق الواجهة الأمامية باستخدام إطار العمل Next.js ومكتبة React. تم اتباع هيكلية تنظيمية قياسية تعتمد على فصل المسؤوليات لضمان سهولة الصيانة والتطوير المستقبلي. يوضح الشكل 7.7 الهيكل العام لمجلدات المشروع.

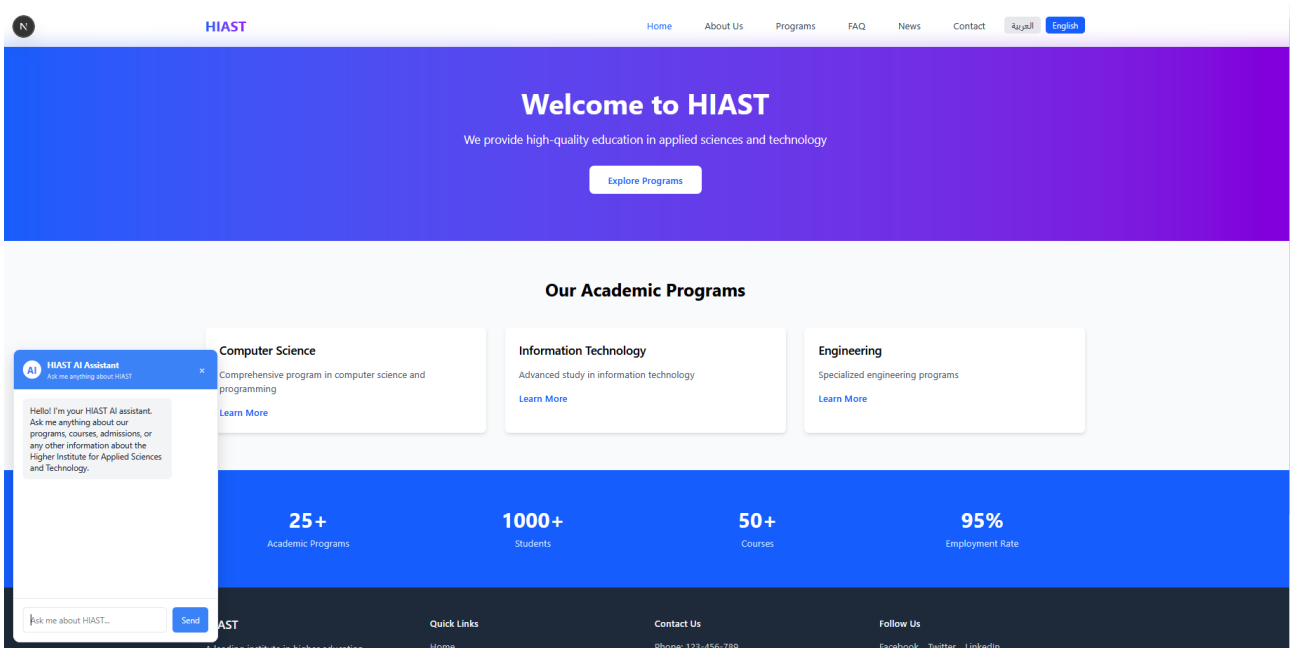


الشكل 7.7: الهيكل العام لمجلدات مشروع الواجهة الأمامية.

فيما يلي شرح لمسؤولية المجلدات الرئيسية في المشروع:

- **مجلد app:** يحتوي هذا المجلد على جميع صفحات التطبيق وهيكلية التوجيه (Routing)، وذلك بناءً على نظام التوجيه المعتمد على نظام الملفات الذي يوفره إطار العمل Next.js.
- **مجلد components:** يضم هذا المجلد جميع مكونات React القابلة لإعادة الاستخدام، مثل الأزرار، بطاقات العرض، ونماذج الإدخال، والتي يتم تجميعها لبناء الصفحات الكاملة.
- **مجلد lib:** يحتوي على طبقة الاتصال مع الواجهات البرمجية (API) الخلفية. يتم هنا تعريف جميع الدوال المسؤولة عن جلب البيانات من خدمة إدارة المحتوى وخدمة الذكاء الاصطناعي.
- **مجلد locales:** يضم هذا المجلد ملفات الترجمة (عادة بصيغة JSON) الخاصة بدعم تعدد اللغات في التطبيق، حيث يحتوي على جميع النصوص باللغتين العربية والإنجليزية.

بعد اكتمال مراحل التصميم والتنفيذ، تم الوصول إلى الواجهة النهائية للمشروع التي تدمج بين منصة الويب التفاعلية والمساعد الذكي. يوضح الشكل 8.7 الواجهة الرئيسية لموقع المعهد، حيث يظهر التصميم الحديث والمتجاوب، مع نافذة المساعد الذكي المدمجة في الزاوية السفلية اليسرى، جاهزة للتفاعل مع المستخدم.



الشكل 8.7: الواجهة الرئيسية لموقع المعهد العالي (HIAST) مع ظهور نافذة المساعد الذكي (Chatbot) في وضع الاستخدام.

الفصل

8

الاختبار والنتائج

1.8 اختبار اعتماديات البنية النظيفة

أحد أهم أهداف البنية المعمارية النظيفة هو فرض **قاعدة التبعية** الصارمة، والتي تنص على أن التبعيات يجب أن تشير دائمًا إلى الداخل (من الطبقات الخارجية إلى الطبقات الداخلية). لضمان عدم انتهاك هذه القاعدة بشكل غير مقصود أثناء عملية التطوير، تم إنشاء مجموعة من الاختبارات الآلية المخصصة للتحقق من صحة هذه الاعتماديات.

تم إنشاء مشروع اختبار منفصل باسم 'ArchitectureTests' باستخدام إطار العمل xUnit. تقوم هذه الاختبارات بفحص كل طبقة من طبقات المشروع للتحقق من أنها لا تعتمد على طبقات غير مسموح بها. على سبيل المثال، يتم التحقق من أن طبقة المجال (Domain) لا تحتوي على أي تبعيات لمشاريع أخرى، وأن طبقة التطبيق (Application) تعتمد فقط على طبقة المجال. يوضح الشكل 1.8 نتيجة تنفيذ هذه الاختبارات.

```
PS C:\Users\ASUS\source\repos\HIAST_RAG\ForthYear\Backend_Web\Presentation> dotnet test
Restore complete (1.1s)
  Domain succeeded (0.2s) → C:\Users\ASUS\source\repos\HIAST_RAG\ForthYear\Backend_Web\Domain\bin\Debug\net8.0\Domain.dll
  Application succeeded (0.4s) → C:\Users\ASUS\source\repos\HIAST_RAG\ForthYear\Backend_Web\Application\bin\Debug\net8.0\Application.dll
  Infrastructure succeeded (0.4s) → C:\Users\ASUS\source\repos\HIAST_RAG\ForthYear\Backend_Web\Infrastructure\bin\Debug\net8.0\Infrastructure.dll
  Presentation succeeded (0.7s) → bin\Debug\net8.0\Presentation.dll
  ArchitectureTests succeeded (0.5s) → C:\Users\ASUS\source\repos\HIAST_RAG\ForthYear\Backend_Web\ArchitectureTests\bin\Debug\net8.0\ArchitectureTests.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.5.3.1+6b60a9e56a (64-bit .NET 8.0.13)
[xUnit.net 00:00:00.06]   Discovering: ArchitectureTests
[xUnit.net 00:00:00.09]   Discovered:  ArchitectureTests
[xUnit.net 00:00:00.09]   Starting:    ArchitectureTests
[xUnit.net 00:00:00.40]   Finished:    ArchitectureTests
  ArchitectureTests test succeeded (1.0s)

Test summary: total: 7, failed: 0, succeeded: 7, skipped: 0, duration: 1.0s
Build succeeded in 4.7s

Workload updates are available. Run `dotnet workload list` for more information.
PS C:\Users\ASUS\source\repos\HIAST_RAG\ForthYear\Backend_Web\Presentation>
```

الشكل 1.8: نتيجة تنفيذ اختبارات التحقق من اعتماديات البنية النظيفة بنجاح.

كما هو واضح في ملخص الاختبار (Test summary)، تم تنفيذ 7 اختبارات بنجاح كامل (succeeded: 7, failed: 0). هذه النتيجة تؤكد برمجيًا أن بنية التطبيق الخلفي تلتزم بقاعدة التبعية بشكل صحيح، مما يضمن بقاء النظام قابلاً للصيانة والتطوير دون مشاكل مستقبلية ناتجة عن التبعيات الخاطئة بين الطبقات.

الخاتمة والآفاق المستقبلية

الخاتمة

في ختام هذا المشروع، تم بنجاح تصميم وتنفيذ منصة ويب متكاملة وحديثة للمعهد العالي للعلوم التطبيقية والتكنولوجيا، معززة بمساعد ذكي يعتمد على أحدث تقنيات الذكاء الاصطناعي. تم تحقيق ذلك من خلال فصل النظام إلى مكونين رئيسيين: خدمة إدارة المحتوى، وخدمة المحادثة الذكية.

تم بناء خدمة إدارة المحتوى باستخدام تقنيات .NET. مع تطبيق صارم لمبادئ **البنية النظيفة** ونمط CQRS، مما نتج عنه نظام خلفي قوي وآمن لإدارة بيانات الموقع. أما خدمة المحادثة الذكية، فقد تم تصميمها كـ**منتج مستقل وقابل للتوصيل** باستخدام Python و FastAPI، مع تطبيق تقنية RAG ودعم استراتيجيات متعددة متقدمة مثل Query Transformation و RAG-Fusion، مما يضمن تقديم إجابات دقيقة وموثوقة.

لقد نجح المشروع في تحقيق أهدافه المتمثلة في تحديث الحضور الرقمي للمعهد وتقديم أداة تفاعلية مبتكرة، مما يمثل خطوة مهمة نحو تحسين تجربة المستخدم وتسهيل الوصول إلى المعلومات.

الآفاق المستقبلية

على الرغم من أن المشروع قد حقق أهدافه الأساسية، إلا أن هناك العديد من الآفاق المستقبلية التي يمكن البناء عليها لتوسيع قدرات النظام وتحسين أدائه:

- **توسيع التكامل مع أنظمة المعهد:** يمكن في المستقبل ربط منصة الويب والخدمات الخلفية مع الأنظمة الأخرى في المعهد

العالي، مثل بوابة الطلاب الإلكترونية، مما يسمح بتقديم خدمات متكاملة وتجربة موحدة للمستخدمين.

- **تعميم خدمة المساعد الذكي:** بما أن خدمة المحادثة الذكية قد صُممت لتكون منتجًا مستقلاً وقابلاً للتوصيل، يمكن بسهولة تطبيقها على مواقع الويب الأخرى الخاصة بالمعهد (مثل مواقع الأقسام أو المخابر او المنصة الطلابية) لتوفير مساعد ذكي متخصص لكل منها.

- **تحسين أداء استرجاع البيانات (RAPTOR):** يمكن تحسين سرعة ودقة استجابة البوت عبر تطوير آلية البحث في قاعدة البيانات المتجهة. إحدى الطرق المتقدمة لذلك هي تقنية **RAPTOR (Recursive Abstractive Processing for Tree-Organized Retrieval)**، حيث تعمل هذه التقنية عبر عنقدة المتجهات، حيث يتم إنشاء ممثل لكل عنقود. عند وصول استعلام جديد، تتم مقارنته أولاً مع ممثلي العناقيد، وبعد تحديد العنقود الأكثر صلة، يتم إجراء البحث الدقيق داخل هذا العنقود فقط، مما يقلل بشكل كبير من مساحة البحث ويزيد من سرعة الاستجابة.

- **تطوير المجيب الآلي إلى مساعد وظيفي (LLM Agent) :** يمكن تطوير البوت من مجرد مجيب آلي للمعلومات إلى "وكيل ذكي" (LLM Agent) قادر على تنفيذ مهام وظيفية نيابة عن المستخدم. بدلاً من أن يخبر الطالب بكيفية التسجيل في دورة، يمكنه أن يقوم بعملية التسجيل له مباشرة عبر التكامل مع API المنصة الطلابية. هذا يتطلب تزويد النموذج اللغوي بالقدرة على استخدام "أدوات" (APIs) مختلفة، وهو ما يمثل الجيل القادم من المساعدين الأذكياء.

المراجع العلميَّة

[1] Zackary Rackauckas, *RAG-Fusion*: *a New Take on Retrieval-Augmented Generation*, International Journal on Natural Language Computing (IJNLC), 2024, vol 13, number 1.

[2] Robert C. Martin, 2017, *Clean Architecture*: *A Craftsman's Guide to Software Structure and Design*, Pearson Professional.

[3] Yunfan Gao et al., 2024, *Retrieval-Augmented Generation for Large Language Models*: *A Survey*, available at https://github.com/Tongji-KGLLM/RAG-Survey

[4] Judith Bishop, 2008, *C* 3.0 *Design Patterns*, O'Reilly Media.

[5] Parishkar Sarthi et al., 2024, *RAPTOR*: *Recursive Abstractive Processing for Tree-Organized Retrieval*, arXiv preprint arXiv:2401.18059.

[6] Microsoft, *.NET Documentation*, available at https://learn.microsoft.com/en-us/dotnet/

[7] Microsoft, *ASP.NET Core Documentation*, available at https://learn.microsoft.com/en-us/aspnet/core/

[8] Microsoft, *Entity Framework Core Documentation*, available at https://learn.microsoft.com/en-us/ef/core/

[9] Tiangolo, *FastAPI Documentation*, available at https://fastapi.tiangolo.com/

[10] LangChain AI, *LangChain Documentation*, available at https://python.langchain.com/docs/get_started/introduction

[11] Chroma, *ChromaDB Documentation*, available at https://docs.trychroma.com/

[12] Vercel, *Next.js Documentation*, available at https://nextjs.org/docs

[13] React Team, *React Documentation*, available at https://react.dev/

[14] Google, *Google AI Gemini API Documentation*, available at https://ai.google.dev/gemini-api/docs

[15] Leonard Richardson, *Beautiful Soup Documentation*, available at https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[16] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, 2018, *Introduction to Data Mining*, 2nd Edition, Pearson.