ثمانيَة

تقديم التكليف

محمد موساوي
مهندس تطوير أنظمة خلفية

دليل التصفح

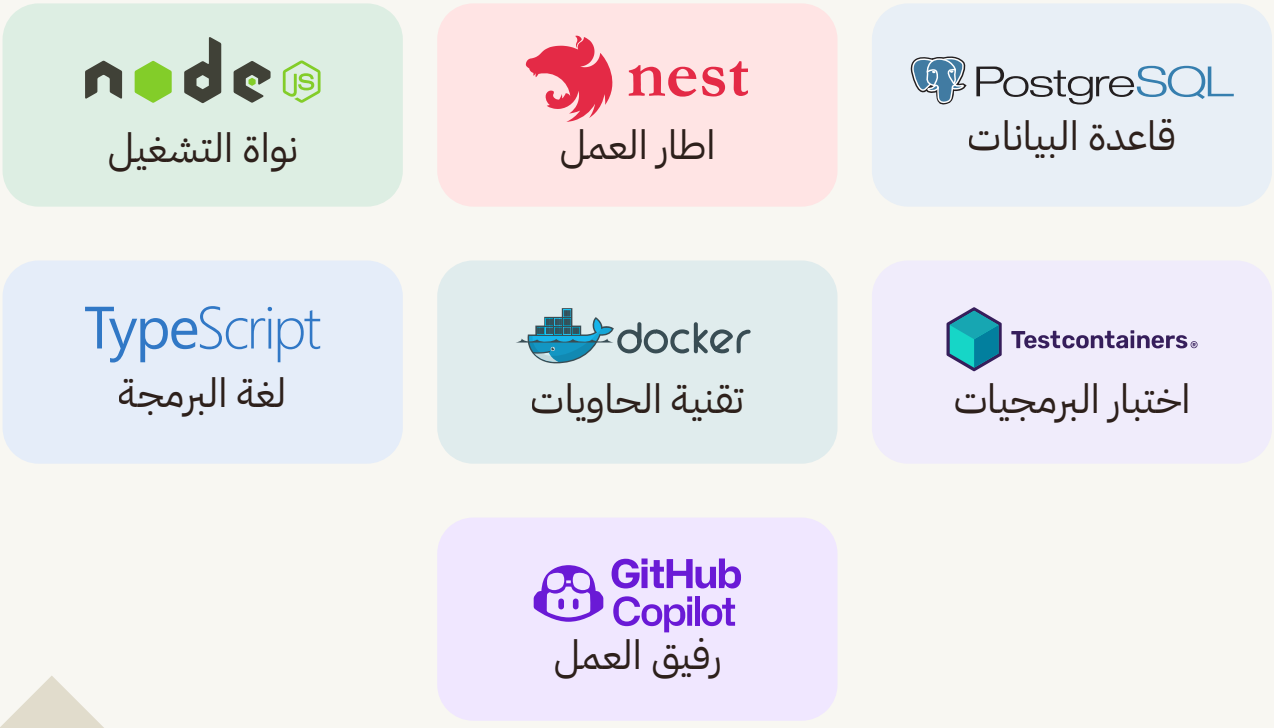
1 - نظرة عامة على المشروع

قمت ببناء المشروع باستخدام NestJS. رغم أنها تجربتي الأولى مع هذا الإطار، إلا أن خبرتي العميقة في بيئة Node.js (وتحديداً Express.js) وتمكني من TypeScript (من خلال React وAngular) **جعلت منحنى التعلم سلساً للغاية.**

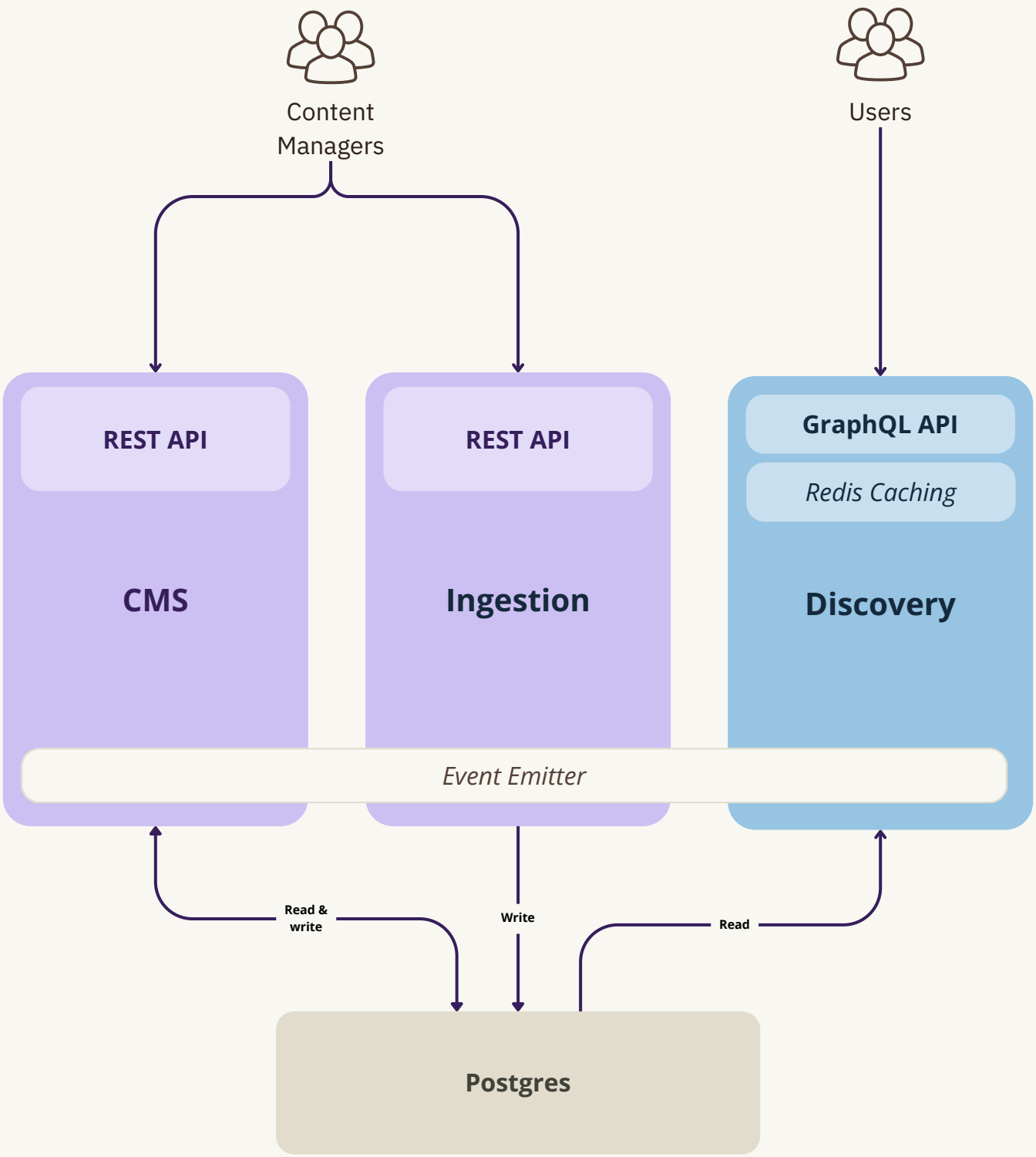
لقد فضلت مبدأ "البساطة على التعقيد". كانت فكرتي الأولية هي بناء نظام خدمات مصغرة (Microservices) يعتمد على NATS و Elasticsearch. **لكن بعد دراسة متأنية للمتطلبات، قررت اعتماد نمط Modular Monolith.**

يتكون النظام من ثلاث وحدات (Modules) تتشارك مخطط البيانات (Data Schema)، ولكن تم تصميمها بشكل مفصول منطقياً لضمان تحقيق مبدأ فصل الاهتمامات (Separation of Concerns).

1.1 - التقنيات المستخدمة:



2 - نظرة متعمقة في البنية المعمارية



تستجيب البنية المعمارية لمشروعي بشكل مباشر للمتطلب الجوهري لتكليف، من خلال ثلاث وحدات مستقلة :

- **وحدة إدارة المحتوى (CMS Module):** تخدم المحررين الداخليين عبر واجهة REST API، مما يتيح إجراء عمليات إدارة البيانات الكاملة (CRUD) على البرامج والمحتوى.
- **وحدة الاستكشاف (Discovery Module):** تخدم المستخدمين العموميين عبر واجهة GraphQL، وقد صُممت خصيصاً لدعم استعلامات القراءة المرنة وعمليات البحث.
- **وحدة الجلب/التغذية (Ingestion Module):** تستعرض قابلية التوسع المستقبلية لعمليات الاستيراد، وهي مهيأة لإضافة محولات (Adapters) لمنصات مثل YouTube، وRSS، وSpotify. و هذا يحقق متطلب "الاستيراد من مصادر متعددة مستقبلاً".

؟

لماذا GraphQL؟

إخترت GraphQL لبناء Discovery API نظراً لتوافقه المثالي مع أنماط استهلاك البيانات لدى المستخدمين النهائيين، **خصوصا في حالة المحتوى الغني لثمانية (بودكاست، وثائقيات، نشرات بريدية...)**

احتياجات المستخدمين عند تصفح المحتوى تتسم بالتنوع والديناميكية: فقد تتطلب واجهة معينة الاكتفاء بالعناوين والصور المصغرة فقط، بينما تتطلب واجهة أخرى البيانات الوصفية الكاملة ... تقنية GraphQL تمنح العميل المرونة لطلب ما يحتاجه بدقة.

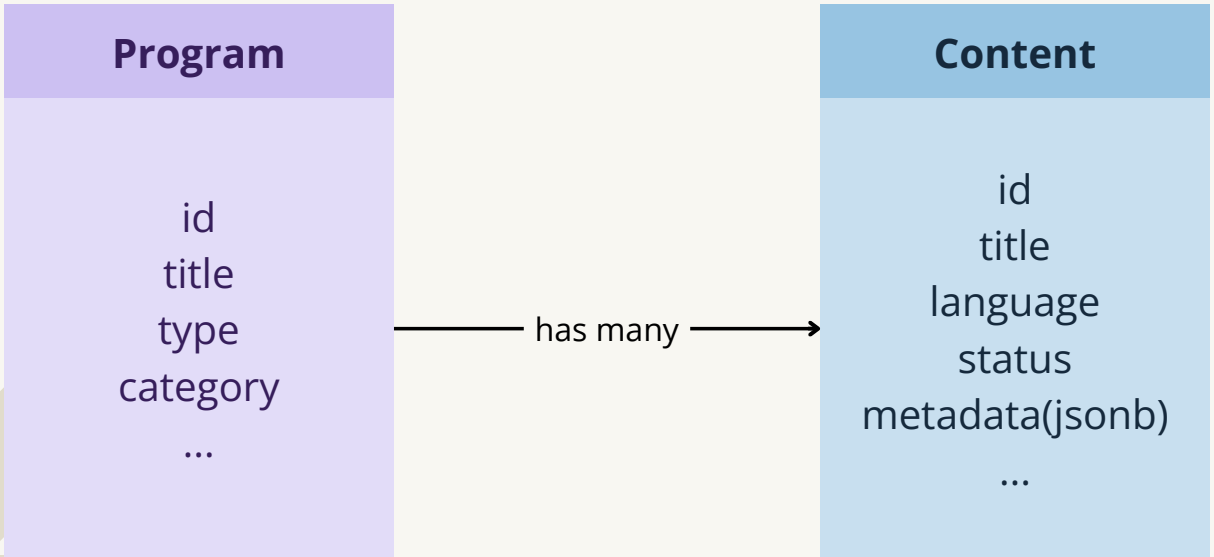
3 - القرارات التقنية الجوهرية

خلال العمل على المشروع، توجب علي اتخاذ عدة قرارات مفصلية نظراً لضيق الوقت ووجود بعض الغموض في المتطلبات.
هدفي الرئيسي كان بناء أساس متين لنظام قابل للتوسع، بحيث يمكن اعتباره "التكرار الأول" (First Iteration) ضمن مشروع يتبع منهجية أجايل (Agile). وفي الوقت ذاته، حرصت على تجنب "المبالغة في الهندسة" أو الغوص في تفاصيل تقنية معقدة تقع خارج النطاق الحالي للمتطلبات.

1.3 - لماذا Modular Monolith بدلاً من Microservices؟

هذا السؤال طُرح في نموذج التقديم للوظيفة، وأظن أن الهدف منه هو معرفة كيفية اتخاذ القرارات الصحيحة التي تسهّل العمل في بداية المشاريع وتُسرّع من الوقت اللازم لطرح المنتج في السوق.
يوفر نمط Modular Monolith الفوائد المعمارية للخدمات المصغرة مثل الحدود الواضحة (Clear Boundaries) بدون الترف التقني الزائد، و الحلول لمشاكل لا نعاني منها أصلاً في اطار هذا العمل المصغر.

2.3 - كيف تم تحديد تم تصميم الجداول والعلاقات في قاعدة البيانات؟



يرتكز النظام في جوهره على **جدولين أساسيين**: Program و Content، وهما يعكسان بدقة التراتبية الطبيعية للإنتاج الإعلامي.

يمثل جدول Program السلاسل البرامجية (مثل "فنجان" أو "ثمانية أسئلة")، بينما يمثل جدول Content الحلقات الفردية أو الأعمال المستقلة.

هذا التجريد ليس خياراً عشوائياً، بل هو محاكاة هندسية لآلية تنظيم ونشر المحتوى في "ثمانية" (الاعتماد على السلاسل)، مما يجعل نموذج البيانات (Data Model) بديهياً وسهل الفهم لكل من المحررين والمطورين.

يدعم هذا النظام المحتوى المستقل ايضا مثل الأفلام الوثائقية المنفردة أو الفيديوهات الترويجية، دون الاضطرار لحشرها تحت تصنيف وهمي.

أخيراً، يستحق عمود metadata (JSONB) وقفة خاصة. بدلاً من إنشاء جداول منفصلة لكل نوع محتوى (مثل podcast_metadata، documentary_metadata)، نقوم بتخزين البيانات الخاصة بالنوع داخل هيكل JSON مرن. فمثلاً، قد تحتوي حلقة البودكاست على رقم الحلقة و مدتها، بينما قد تحتوي نشرة بريدية مستقبلية على الزمن اللازم للقرائة.

هذا النهج يقايض التحقق الصارم من المخطط (Schema Enforcement) بمرونة عالية في وقت التشغيل. تتولى واجهات TypeScript التحقق من صحة الهيكل داخل كود التطبيق، بينما تظل قاعدة البيانات مستقلة عن تفاصيل المخطط (Schema-agnostic).

3.3 - ماذا كنت سأغير لو توفر لي المزيد من الوقت؟

نظراً لنطاق المهمة المحدد، قمت بتبسيطات متعمدة للتركيز على إظهار "التفكير المعماري" بدلاً من الغرق في تعقيدات البنية التحتية.

هذه خارطة الطريق لتطوير هذا النظام لو كان مشروعاً إنتاجياً بجدول زمني ممتد:

1. **إدارة الصلاحيات (Authentication & Authorization):** التنفيذ الحالي يفتقر لإدارة المستخدمين، وهي فجوة واضحة لأي نظام CMS حقيقي. مع وقت أطول، كنت سأضيف مصادقة تعتمد على JWT مع تحكم في الوصول مبني على الأدوار
2. **استراتيجيات استيراد فعلية:** يوضح المستورد الوهمي (Mock) لليوتيوب نمط الاستراتيجية (Strategy Pattern)، لكن الإنتاج الفعلي يحتاج لتكامل حقيقي مع الـ APIs. كنت سأنفذ استدعاءات فعلية لـ YouTube Data API v3، وتحليل RSS.
3. **محرك بحث Elasticsearch:** بحث PostgreSQL النصي يعمل جيداً في النطاق الحالي، لكن المحتوى العربي الإنتاجي سيستفيد بشكل كبير من ميزات Elasticsearch المتفوقة: التسامح مع الأخطاء المطبعية، المطابقة الضبابية (Fuzzy Matching)...
4. **المراقبة الشاملة (Comprehensive Observability):** يفتقر التنفيذ الحالي للمراقبة الدقيقة. كنت سأضيف OpenTelemetry للتتبع الموزع، ومقاييس Prometheus لمراقبة معدلات نجاح الكاش (Hit Rates) وزمن الاستجابة...
5. **الحماية وتحديد معدل الاستخدام (Rate Limiting & Security):** واجهة Discovery حالياً غير محمية. بيئة الإنتاج تتطلب تحديد معدل الطلبات (لكل IP ومستخدم)، والتحقق من صحة المدخلات وتعقيمها، وضبط CORS، وتحليل تعقيد استعلامات GraphQL لمنع الاستعلامات المتداخلة المكلفة من إرهاق قاعدة البيانات

4 - تصفح المشروع على Github

المشروع جاهز للتشغيل بالكامل وموثق بدقة. يوفر **ملف README.md** تعليمات البدء السريع: الأمر make up يقوم بتشغيل PostgreSQL و Redis عبر Docker Compose، والأمر make dev يطلق تطبيق NestJS، بينما يقوم make test بتنفيذ حزمة الاختبارات الكاملة.

أما **ملف design.md**، فيحتوي على وثيقة التصميم الكاملة، متضمنة مخططات البنية المعمارية (Architecture Diagrams)، ونماذج البيانات، والأسس المنطقية لكل قرار تقني.

كما يتضمن **مجلد postman** مجموعة كاملة لاختبار كل من CMS REST API و Discovery GraphQL API، يمكنك استيرادها لاستكشاف جميع نقاط الاتصال (Endpoints) بطلبات مجهزة مسبقاً.

قاعدة الكود (Codebase) منظمة تماماً كما هو موضح في قسم "نظرة متعمقة في البنية المعمارية": المجلدات cms، و discovery، و ingestion هي وحدات مستقلة ذاتياً (Self-contained) يمكنك استكشافها بشكل منفصل.

للحصول على أفضل تجربة مراجعة، أوصي بالخطوات التالية:

- نفذ الأمر هذا لبدء تشغيل النظام:

```
make up && make dev
```

- قم باستيراد مجموعة Postman وأنشئ "برنامجاً" (Program) يحتوي على بعض المحتوى.

- افتح http://localhost:3000/graphql لتجربة الاستعلام عبر Discovery API.

- راجع اختبارات E2E في مجلد test لترى كيف تتكامل الوحدات مع بعضها.

شكرا

ثمانية