# تقرير مشروع AuraLens: تحليل وتوثيق النظام

## الفصل الأول: مقدمة المشروع

### 1.1 ملخص المشروع

"AuraLens" هو نظام متطور تم تصميمه ليكون بمثابة مساعد بصري يعمل بالذكاء الاصطناعي، وهو موجه خصيصًا للمكفوفين وضعاف البصر. الهدف الأساسي للمشروع هو تمكين المستخدمين من فهم واستكشاف بيئتهم المحيطة بشكل أفضل من خلال واجهة تطبيق بسيطة ومباشرة على الهاتف المحمول. يعتمد النظام على بنية "العميل والخادم" (Client-Server) حيث يقوم تطبيق الهاتف بالتقاط الصور والفيديو، بينما تتولى الخوادم الخلفية القوية عمليات التحليل المعقدة باستخدام نماذج الذكاء الاصطناعي المتقدمة.

### 1.2 أهداف المشروع

* **تمكين الاستقلالية:** تزويد المستخدمين المكفوفين بأداة قوية تساعدهم على التنقل والتفاعل مع العالم من حولهم بثقة أكبر وتقليل الاعتماد على الآخرين.
* **تحليل البيئة بشكل شامل:** تقديم مجموعة متكاملة من الميزات التي تغطي الاحتياجات المختلفة، وتشمل:
  + **الإجابة على الأسئلة المرئية (VQA):** تمكين المستخدم من التقاط صورة وطرح أسئلة باللغة الطبيعية حولها للحصول على وصف وتحليل فوري.
  + **التعرف الضوئي على الحروف (OCR):** استخلاص النصوص من اللافتات، قوائم الطعام، المستندات، وغيرها، وقراءتها بصوت عالٍ.
  + **تحليل الفيديو المباشر:** تسجيل مقاطع فيديو للمحيط وطرح أسئلة حول محتواها لفهم الأحداث والتفاعلات التي جرت.
* **السلامة أولاً:** إعطاء الأولوية القصوى لتحديد وتنبيه المستخدم من المخاطر المحتملة في بيئته، مثل العوائق أو الأسطح الساخنة، وتقديم هذه التحذيرات قبل أي معلومات أخرى.
* **سهولة الاستخدام:** تصميم واجهة مستخدم بسيطة وبديهية يمكن تشغيلها بالكامل عبر قارئات الشاشة القياسية، دون الحاجة إلى تدريب مسبق.
* **قابلية التخصيص:** السماح للمستخدم باختيار نموذج الذكاء الاصطناعي ووضع التحليل (موجز أو مفصل) الذي يناسب تفضيلاته، مع حفظ هذه الإعدادات للاستخدام المستقبلي.
* **الأداء والموثوقية:** ضمان استجابة سريعة للطلبات (في غضون 5-20 ثانية في الظروف العادية) وتقديم نتائج دقيقة وموثوقة، خاصة فيما يتعلق بالسلامة.

## الفصل الثاني: متطلبات النظام

### 2.1 المتطلبات الوظيفية

#### 2.1.1 النظام الأساسي وتفاعل المستخدم

* **1.1.1:** يجب أن يكون النظام قابلاً للتصفح والتشغيل بشكل كامل باستخدام برامج قارئ الشاشة القياسية (Standard Screen Readers).
* **1.1.2:** يجب أن تحتوي جميع العناصر التفاعلية والمعلوماتية على الشاشة على تسميات نصية متوافقة لضمان إمكانية الوصول عبر قارئ الشاشة.
* **1.1.3:** يجب تصميم واجهة مستخدم التطبيق ومسارات العمل ببساطة تامة لتكون بديهية للمستخدم لأول مرة دون الحاجة إلى برنامج تعليمي مخصص.

#### 2.1.2 التحكم في السرد والوصف

* **1.2.1:** يجب أن يكون المستخدم قادرًا على الاختيار بين وضعي وصف:
  + **الوضع الموجز:** يوفر ملخصًا مقتضبًا من جملة واحدة للمشهد أو النص.
  + **الوضع المفصل:** يوفر تحليلاً أكثر شمولاً ووصفًا.

#### 2.1.3 الإجابة على الأسئلة المرئية (Visual Question Answering - VQA)

* **1.3.1:** يجب أن يكون المستخدم قادرًا على التقاط صورة وطرح سؤال باللغة الطبيعية عنها.
* **1.3.2:** يجب على النظام إعطاء الأولوية لتحديد مخاطر السلامة المحتملة والإبلاغ عنها في جميع استجابات الإجابة على الأسئلة المرئية. يجب تقديم تحذيرات السلامة أولاً.
* **1.3.3:** إذا كان نموذج الذكاء الاصطناعي غير متأكد من تحليله، فيجب عليه تقديم أفضل تخمين لديه دون ذكر عدم اليقين صراحةً.
* **1.3.4:** لن يحتفظ النظام بسياق الاستعلامات السابقة. يتم التعامل مع كل سؤال كطلب جديد ومستقل.

#### 2.1.4 التعرف الضوئي على الحروف (Optical Character Recognition - OCR)

* **1.4.1:** يجب أن يكون المستخدم قادرًا على التقاط صورة لنص.
* **1.4.2:** يجب على النظام استخلاص المحتوى النصي وقراءته بصوت عالٍ من قوائم المطاعم المطبوعة، والبريد، وملصقات المنتجات، ولافتات الشوارع، والكتابة اليدوية.

#### 2.1.5 تسجيل وتحليل الفيديو

* **1.5.1:** يجب أن يكون المستخدم قادرًا على بدء وإيقاف تسجيل فيديو لمحيطه.
* **1.5.2:** بعد التسجيل، يجب أن يكون المستخدم قادرًا على طرح أسئلة باللغة الطبيعية حول محتوى الفيديو.

#### 2.1.6 معالجة الأخطاء والاتصال

* **1.6.1:** إذا فشل التطبيق في تلقي استجابة من الخادم، فيجب عليه إبلاغ المستخدم بالفشل.
* **1.6.2:** في حالة عدم توفر اتصال بالإنترنت عند محاولة إجراء ما، يجب على النظام إبلاغ المستخدم فورًا برسالة خطأ.

#### 2.1.7 إدارة النماذج

* **1.7.1:** يجب أن يكون المستخدم قادرًا على تحديد نموذج الذكاء الاصطناعي المفضل لديه من قائمة متوفرة.
* **1.7.2:** سيتم تحديث قائمة نماذج الذكاء الاصطناعي المتاحة مع إصدارات التطبيق الجديدة.

### 2.2 المتطلبات غير الوظيفية

* **2.1.1 الأداء:** بالنسبة لطلبات الإجابة على الأسئلة المرئية والتعرف الضوئي على الحروف، يجب أن يهدف النظام إلى تقديم استجابة في غضون 5-20 ثانية في ظل ظروف الشبكة العادية.
* **2.2.1 الدقة والموثوقية:** الأولوية القصوى للدقة هي اكتشاف مخاطر السلامة والإبلاغ عنها. لا يجوز إغفال أي أخطار محتملة.
* **2.2.2:** بالنسبة للأوصاف غير المتعلقة بالسلامة، يُقبل مستوى "جيد بما فيه الكفاية" من الدقة الذي يوفر وصفًا متماسكًا ومفيدًا.
* **2.3.1 سهولة الاستخدام وإمكانية الوصول:** يجب تحسين جميع قرارات التصميم لتناسب تجربة المستخدم غير البصرية، مع إعطاء الأولوية للبساطة ووضوح المعلومات الصوتية.
* **2.4.1 التخصيص:** يجب على التطبيق حفظ آخر نموذج ذكاء اصطناعي اختاره المستخدم واستخدامه تلقائيًا للطلبات اللاحقة حتى يتم تغييره.
* **2.4.2:** يجب على التطبيق حفظ وضع الوصف المفضل للمستخدم (موجز أو مفصل) وتطبيقه على جميع الطلبات المستقبلية حتى يتم تغييره.
* **2.5.1 البيانات والسجل:** سيتم تخزين جميع الصور ومقاطع الفيديو التي يلتقطها المستخدم على خوادم النظام.
* **2.5.2:** لن يوفر التطبيق للمستخدم إمكانية الوصول إلى سجل استعلاماته.
* **2.6.1 الأمان:** سيعمل التطبيق كنظام مجهول الهوية. لن يُطلب من المستخدمين إنشاء حساب.
* **2.7.1 سلوك النظام:** لن يدعم التطبيق المعالجة في الخلفية. إذا انتقل المستخدم بعيدًا عن التطبيق أو أغلق الجهاز، فسيتم إنهاء أي عملية جارية.
* **2.7.2:** سيظل التطبيق صامتًا أثناء معالجة الطلب من قبل الخادم.
* **2.7.3:** لن يقدم التطبيق أي تحذيرات تتعلق باستخدام البطارية أو البيانات.

## الفصل الثالث: البنية الهندسية للنظام

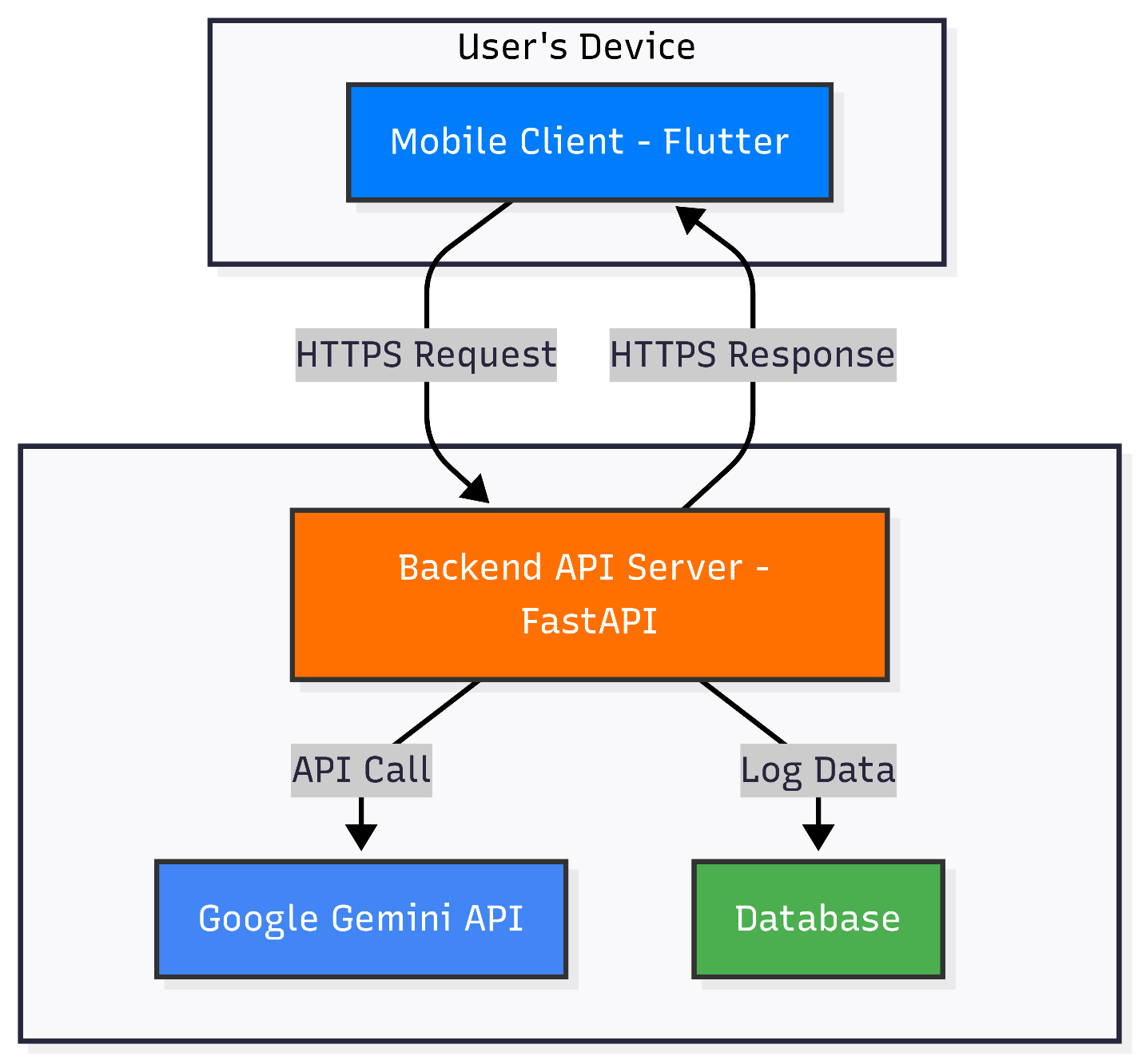
### 3.1 نظرة عامة تنفيذية والبنية الهندسية للنظام

نظام AuraLens، وهو مساعد بصري عالي الأداء يعمل بالذكاء الاصطناعي للمكفوفين وضعاف البصر (BLV). تم تصميم النظام مع التركيز على قابلية التوسع (Scalability)، وقابلية الصيانة (Maintainability) لدعم مهمته الأساسية المتمثلة في تقديم تحليل بيئي سريع ودقيق وآمن.

تعتمد البنية الهندسية على نموذج العميل والخادم (Client-Server Model) منفصل المكونات. يتولى تطبيق محمول خفيف الوزن ومتعدد المنصات (Lightweight, Cross-Platform Mobile Client) مسؤولية تفاعل المستخدم والتقاط البيانات، بينما يتولى خادم خلفي قوي وعديم الحالة (Stateless Backend Server) جميع عمليات المعالجة الحاسوبية المكثفة للذكاء الاصطناعي. يضمن هذا الفصل تجربة مستخدم سريعة الاستجابة ويسمح بالتطوير المستقل لمكونات العميل (Client) والخادم (Server).

**التقنيات الأساسية:**

* **الواجهة الأمامية (Client):** Flutter Framework (Dart)
* **الواجهة الخلفية (Server):** FastAPI Framework (Python)
* **خدمة الرؤية بالذكاء الاصطناعي:** Google Gemini 2.5 Family (API-Driven)
* **تخزين البيانات:** NoSQL Database

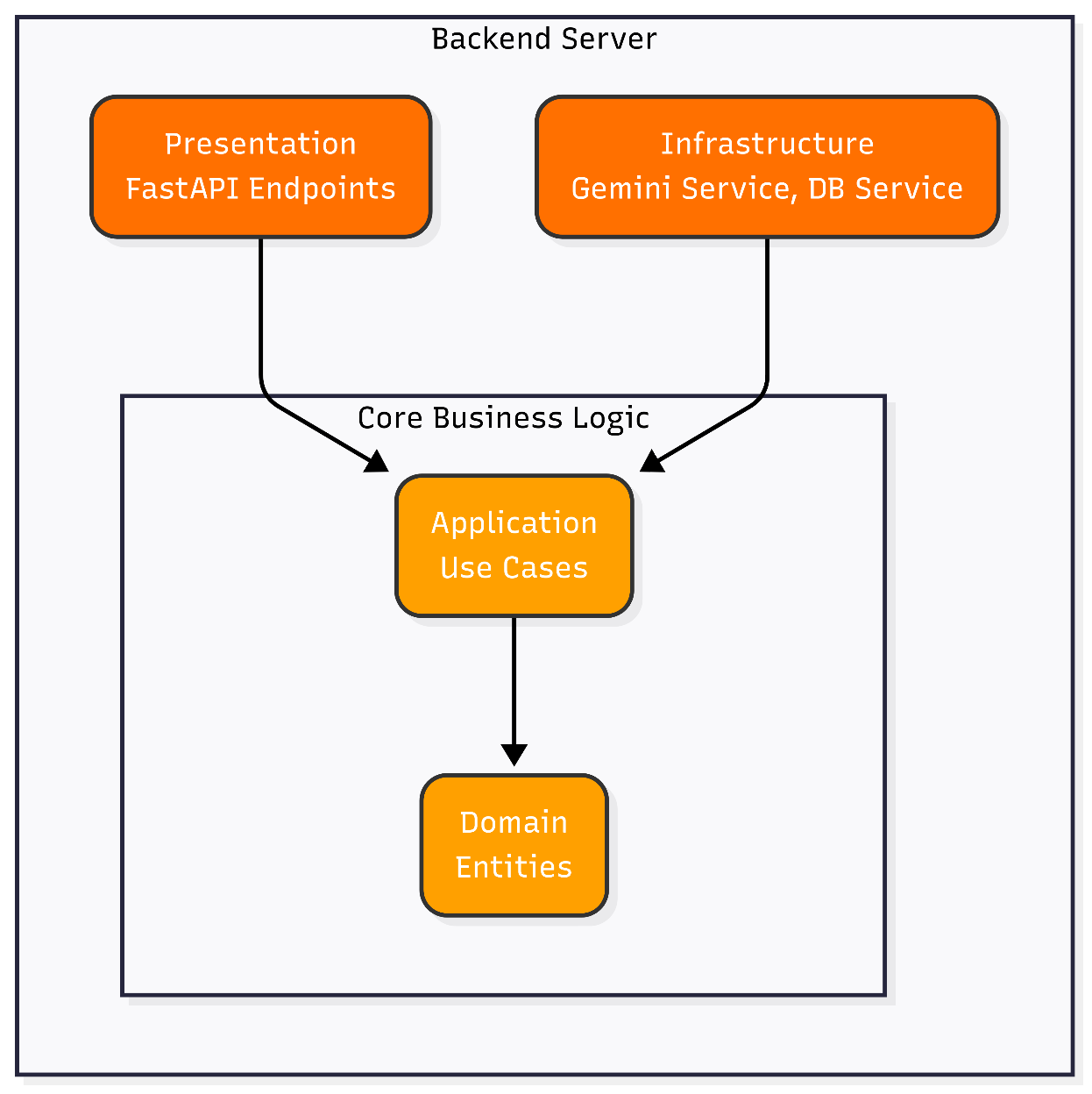


يقدم هذا المخطط نظرة شاملة على المكونات الرئيسية وتفاعلاتها، مما يؤسس للسياق العام للنظام.

### 3.2 البنية الهندسية للواجهة الخلفية (Backend)

تم تصميم الواجهة الخلفية باتباع مبادئ العمارة النظيفة (Clean Architecture). هذا النهج متعدد الطبقات يعزل منطق العمل (Business Logic) عن الاهتمامات الخارجية مثل قواعد البيانات وأطر العمل، مما يعزز قابلية الاختبار ويبسط الصيانة. يتم تطبيق قاعدة التبعية: تعتمد الطبقات الخارجية على الطبقات الداخلية، لكن الطبقات الداخلية لا تدرك أي شيء عن الطبقات الخارجية. يتم تحقيق ذلك من خلال استخدام نمط التصميم حقن التبعية (Dependency Injection)، الذي تتم إدارته بواسطة إطار عمل FastAPI.

**الطبقات المعمارية (Architectural Layers):**

****

يوضح المخطط أعلاه الطبقات الأربع المتميزة للواجهة الخلفية والاتجاه للتبعيات، والتي تتدفق إلى الداخل نحو

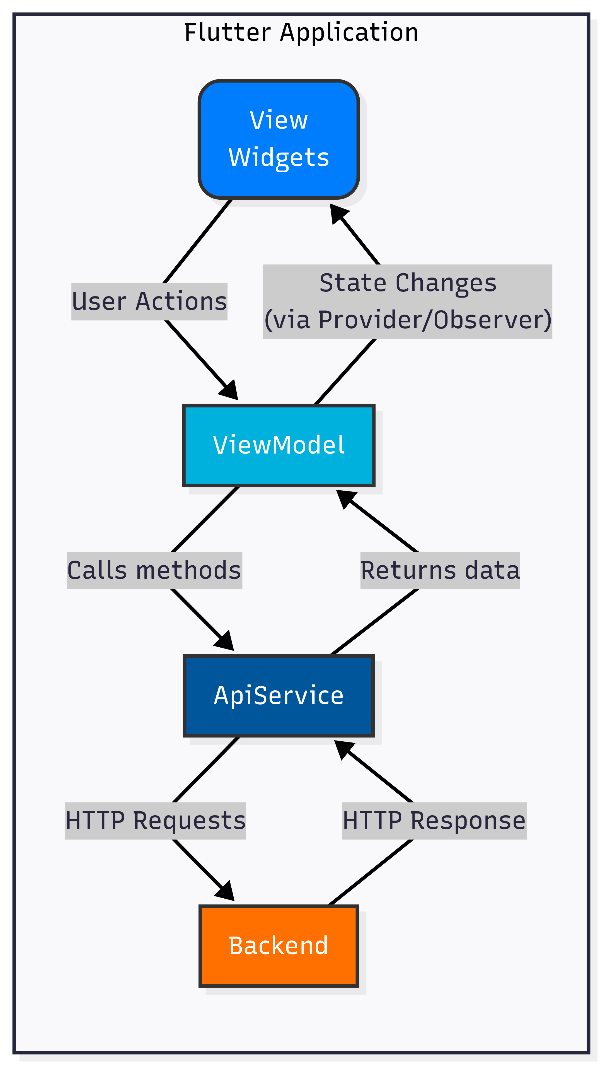
كيانات ال *Domain*

* **طبقة المجال (Domain Layer):** هي جوهر التطبيق، وتحتوي على منطق العمل للكيان وهياكل البيانات (Entities). وهي عبارة عن نماذج بسيطة مستقلة عن جميع الطبقات الأخرى.
* **طبقة التطبيق (Application Layer):** تحتوي على منطق العمل الخاص بالتطبيق (Use Cases). يقوم كل "use case" بتنسيق تدفق البيانات بين طبقتي المجال والبنية التحتية. وتعتمد على واجهات مجردة (Interfaces) للخدمات التي تحتاجها، وليس على تطبيقات ملموسة.
* **طبقة العرض (Presentation Layer):** هي نقطة الدخول إلى النظام، وهي عبارة عن واجهة برمجة تطبيقات RESTful مبنية باستخدام FastAPI. تتولى معالجة عمليات إرسال واستقبال طلبات HTTP، والتحقق من صحتها، وتستخدم نظام حقن التبعية (Dependency Injection) في FastAPI لتزويد طبقة التطبيق بالخدمات التي تحتاجها في وقت التشغيل.
* **طبقة البنية التحتية (Infrastructure Layer):** هي الطبقة الخارجية، وتحتوي على التطبيقات الملموسة للخدمات الخارجية مثل GeminiVisionService و DatabaseService. هنا توجد جميع عمليات الإدخال/الإخراج والتكامل مع الأطراف الثالثة (Third Party).

### 3.3 البنية الهندسية للواجهة الأمامية (Frontend)

العميل (Client) هو تطبيق متعدد المنصات مبني باستخدام Flutter. تم تصميم بنيته حول نمط التصميم (Model-View-ViewModel - MVVM)، الذي يوفر فصلاً واضحًا بين واجهة المستخدم (UI)، وإدارة الحالة (State Management)، ومنطق العمل (Business Logic).

تتم معالجة إدارة الحالة بشكل تعريفي باستخدام حزمة **Provider**، والتي تعمل كمحدد موقع الخدمة (Service Locator) للوصول إلى خدمات مثل ApiService وكتطبيق لنمط المراقب (Observer Pattern) لتحديثات واجهة المستخدم التفاعلية.



يُظهر المخطط أعلاه العلاقة بين ال *View* وال *ViewModel* وال *Model* ، ويسلط الضوء على تدفق

البيانات أحادي الاتجاه الذي يجعل واجهة المستخدم تفاعلية.

* **العرض (View):** تتألف من شجرة الويدجت (Widget Tree) التي تحدد واجهة المستخدم. وهي عديمة الحالة وتفاعلية (Stateless and Reactive)، وتفوض جميع إجراءات المستخدم إلى ViewModel، ويُعاد بناؤها تلقائيًا عند تغير حالة ViewModel.
* **نموذج العرض (ViewModel):** الوسيط الذي يحتفظ بحالة واجهة المستخدم (isLoading, resultData) ومنطق العمل. يستخدم ChangeNotifier لإعلام طرق العرض (Views) المستمعة بتحديثات الحالة.
* **النموذج (Model):** فئات Dart بسيطة تمثل هياكل البيانات للتطبيق، وتقوم بشكل أساسي بتحليل استجابات JSON من واجهة برمجة تطبيقات الواجهة الخلفية إلى كائنات محددة النوع (Strongly-Typed Objects).

## الفصل الرابع: تحليل تدفقات العمل الرئيسية

### 4.1 تهيئة المستخدم لأول مرة وإنشاء المعرّف (ID)

تم تصميم تدفق العمل هذا لمنح المستخدم معرّفًا فريدًا ومجهول الهوية عند أول اتصال ناجح له بالخادم. هذا المعرّف ضروري لتسجيل وتتبع الطلبات في الواجهة الخلفية.

1. **طلب عنوان IP (الواجهة الأمامية):**
   * عند تحميل HomePage، يتم التحقق مما إذا كان قد تم تعيين عنوان IP للخادم لهذه الجلسة.
   * إذا لم يتم العثور على عنوان IP، يظهر مربع حوار يطلب من المستخدم إدخاله. تتم إدارة ذلك بواسطة SettingsService.
2. **بدء تهيئة المستخدم (الواجهة الأمامية):**
   * بمجرد أن يحفظ المستخدم عنوان IP، يحاول التطبيق على الفور تهيئة المستخدم عن طريق استدعاء \_apiService.initializeUser().
   * داخل api\_service.dart، تتحقق دالة initializeUser أولاً مما إذا كان معرّف المستخدم موجودًا بالفعل باستخدام UserService.
   * إذا لم يتم العثور على معرّف، فإنه يرسل طلب POST إلى نقطة النهاية الخلفية: http://<server\_ip>:8000/api/v1/users/init.
3. **إنشاء وإرجاع المعرّف (الواجهة الخلفية):**
   * تستقبل الواجهة الخلفية الطلب عند نقطة النهاية /users/init.
   * تقوم الواجهة الخلفية بإنشاء معرّف فريد جديد.
   * ثم ترسل استجابة JSON تحتوي على المعرّف الجديد، على سبيل المثال: {"user\_id": "some-unique-identifier"}.
4. **حفظ معرّف المستخدم (الواجهة الأمامية):**
   * تستقبل ApiService استجابة 201 Created. وتقوم بتحليل JSON باستخدام نموذج UserInitResponse لاستخراج المعرّف.
   * يتم بعد ذلك حفظ هذا المعرّف الجديد بشكل آمن في التخزين المحلي للجهاز باستخدام UserService.
   * أخيرًا، يتم تحديث HomeViewModel، ويتم عرض معرّف المستخدم الجديد في شريط التطبيق HomePage.

### 4.2 جلب قائمة النماذج المتاحة

يعمل تدفق العمل هذا مباشرة بعد تهيئة المستخدم، مما يضمن أن التطبيق يعرف نماذج الذكاء الاصطناعي التي يمكنه تقديمها للميزات المختلفة.

1. **بدء جلب النماذج (الواجهة الأمامية):**
   * فور نجاح تهيئة المستخدم، تستدعي HomePage الدالة context.read<ModelsViewModel>().fetchModels().
   * يتم التعامل مع هذا الاستدعاء بواسطة ModelsViewModel، الذي يضبط حالته على isLoading = true ثم يستدعي دالة \_apiService.getModels().
2. **طلب قائمة النماذج (الواجهة الأمامية):**
   * تقوم دالة getModels في api\_service.dart بإنشاء وإرسال طلب GET إلى نقطة النهاية الخلفية: http://<server\_ip>:8000/api/v1/models/.
3. **تحميل وإرجاع النماذج (الواجهة الخلفية):**
   * تستقبل نقطة النهاية /models/ في الواجهة الخلفية الطلب.
   * تقوم بقراءة ملف التكوين configs/models.yaml من قرص الخادم.
   * تقوم بتحليل هذا الملف للعثور على جميع النماذج التي تم تمييزها كـ selectable: true.
   * يعيد الخادم بعد ذلك كائن JSON حيث تكون المفاتيح هي أسماء الميزات والقيم هي قوائم بأسماء النماذج.
4. **تخزين وتوزيع النماذج (الواجهة الأمامية):**
   * تستقبل ApiService استجابة 200 OK وتقوم بفك تشفير نص JSON إلى ModelMap.
   * يتم تخزين خريطة النماذج هذه في ModelsViewModel.
   * الآن، يمكن للشاشات الأخرى مثل VQAScreen و OCRScreen الوصول إلى ModelsViewModel المركزي لملء قوائمها المنسدلة "AI Model".

### 4.3 إجراء طلب الإجابة على الأسئلة المرئية (VQA)

هذه ميزة أساسية في التطبيق، تتضمن تحميل الصور، وسؤال باللغة الطبيعية، وإجابة مفصلة تم إنشاؤها بواسطة الذكاء الاصطناعي.

1. **التقاط الصورة والإدخال (الواجهة الأمامية):**
   * في VQAScreen، يلتقط المستخدم صورة ويكتب سؤالاً. كما يختار نموذج الذكاء الاصطناعي ووضع التحليل المطلوب ("موجز" أو "شامل").
   * عند الضغط على زر "الحصول على إجابة"، يتم استدعاء دالة \_getAnswer.
2. **بدء جلب VQA (الواجهة الأمامية):**
   * تستدعي \_getAnswer الدالة context.read<VqaViewModel>().fetchVqaResult() مع مسار الصورة ونص السؤال.
   * يقوم VqaViewModel بضبط حالته على التحميل، ويمسح أي نتائج قديمة، ويستدعي \_apiService.getVqaResult(). بشكل حاسم، يمرر ليس فقط الصورة والسؤال، ولكن أيضًا \_selectedModel و \_selectedMode من حالته.
3. **إنشاء وإرسال الطلب (الواجهة الأمامية):**
   * في api\_service.dart، تنشئ دالة getVqaResult طلب POST من نوع multipart/form-data.
   * ترفق الصورة والسؤال واسم النموذج ووضع التحليل في نص الطلب.
   * بشكل حاسم، تضيف معرّف المستخدم إلى رؤوس الطلب كـ X-User-ID.
   * يتم إرسال الطلب إلى نقطة النهاية الخلفية: /api/v1/vqa/.
4. **معالجة الطلب (الواجهة الخلفية):**
   * تستقبل الواجهة الخلفية الطلب عند نقطة النهاية /vqa/.
   * **طبقة العرض (Presentation):** تتحقق من صحة البيانات الواردة.
   * **طبقة التطبيق (Application):** تستدعي VQAUseCase. تأخذ حالة الاستخدام هذه mode ("brief" أو "thorough") وتستخدمها لتحديد موجه النظام المناسب من configs/prompts.yaml.
   * **طبقة البنية التحتية (Infrastructure):** تستدعي حالة الاستخدام GeminiVisionService، وتمرر الصورة والموجه الذي تم إنشاؤه بالكامل. تقوم هذه الخدمة بإجراء استدعاء API الفعلي إلى Google Gemini API. كما تستدعي DatabaseService لتسجيل الطلب بأكمله والنتيجة النهائية.
5. **استلام وعرض النتيجة (الواجهة الأمامية):**
   * تستقبل ApiService استجابة JSON من الواجهة الخلفية وتقوم بتحليلها إلى كائن VqaResult.
   * يتم تمرير هذه النتيجة مرة أخرى إلى VqaViewModel، الذي يقوم بتحديث حالته بالإجابة ويضبط isLoading على false.
   * نظرًا لأن VQAScreen "تراقب" ViewModel، فإنه يعيد بناء نفسه تلقائيًا. يعرض \_buildResultDisplay الآن الإجابة من الذكاء الاصطناعي.

### 4.4 إجراء طلب التعرف الضوئي على الحروف (OCR)

تم تبسيط تدفق العمل هذا لاستخراج النص بسرعة من صورة. إنه مشابه لـ VQA ولكنه أبسط لأنه لا يتطلب سؤالاً.

1. **التقاط الصورة (الواجهة الأمامية):**
   * في OCRScreen، ينقر المستخدم على زر "مسح النص". هذا الإجراء الفردي يفتح الكاميرا على الفور.
   * بمجرد التقاط صورة، يتم استدعاء دالة \_pickImageAndAnalyze.
2. **بدء جلب OCR (الواجهة الأمامية):**
   * تستدعي الدالة context.read<OcrViewModel>().fetchOcrResult() مع مسار الصورة الجديد.
   * يستدعي OcrViewModel (الذي قام بالفعل بتحميل آخر نموذج OCR استخدمه المستخدم عبر PreferenceService) \_apiService.getOcrResult()، ويمرر مسار الصورة والنموذج المحدد.
3. **إنشاء وإرسال الطلب (الواجهة الأمامية):**
   * ينشئ api\_service.dart طلب POST من نوع multipart/form-data، ويرفق الصورة و model\_option.
   * يتم إرسال الطلب إلى نقطة النهاية الخلفية: /api/v1/ocr/.
4. **معالجة الطلب (الواجهة الخلفية):**
   * تستقبل نقطة النهاية /ocr/ في الواجهة الخلفية الطلب.
   * تستدعي OCRUseCase التي تستخدم موجهًا محددًا من configs/prompts.yaml يسمى text\_extraction، والذي يوجه نموذج Gemini إلى "استخراج كل النص ذي الصلة من الصورة."
   * ترسل GeminiVisionService الصورة والموجه إلى Gemini API.
   * تعيد الواجهة الخلفية النص المستخرج في استجابة JSON.
5. **استلام وعرض النتيجة (الواجهة الأمامية):**
   * تقوم ApiService بتحليل استجابة JSON إلى كائن OcrResult.
   * يتم تمرير هذه النتيجة إلى OcrViewModel.
   * يتم تحديث OCRScreen، ويعرض \_buildResultDisplay النص المستخرج.

### 4.5 الجلسة المباشرة، التسجيل، والأسئلة والأجوبة

هذا هو تدفق العمل الأكثر ديناميكية، وهو مصمم للسياق المستمر حول بيئة متغيرة.

1. **بدء جلسة (الواجهة الأمامية والخلفية):**
   * عندما ينتقل المستخدم إلى SessionScreen، فإنه يستدعي على الفور \_viewModel.initializeSession().
   * يستدعي SessionViewModel \_apiService.startSession()، الذي يقوم بإجراء طلب POST إلى نقطة النهاية الخلفية /session/start.
   * تقوم الواجهة الخلفية بإنشاء session\_id فريد جديد، وتخزينه، وإعادته إلى التطبيق. يحفظ التطبيق هذا session\_id في حالة SessionViewModel.
2. **بدء التسجيل (الواجهة الأمامية):**
   * يضغط المستخدم على زر "بدء التسجيل". يقوم SessionViewModel بضبط \_isRecording = true ويبدأ Timer.periodic.
   * كل 5 ثوانٍ، يعمل هذا المؤقت ويستدعي دالة لالتقاط الوسائط (\_captureFrame أو \_captureVideo).
3. **معالجة إطارات/مقاطع الوسائط (الواجهة الأمامية والخلفية):**
   * تستدعي دالة الالتقاط دالة processCapturedFrame أو processCapturedClip في ViewModel.
   * يستدعي ViewModel بعد ذلك دالة ApiService المقابلة، ويرسل الصورة/الفيديو و session\_id إلى نقاط النهاية الخلفية.
   * **الواجهة الخلفية:** لكل قطعة من الوسائط، تقوم الواجهة الخلفية بتشغيل نموذج video\_scene\_extractor لوصف الحدث. ثم تستخدم نموذج video\_scene\_aggregator مع موجه narrative\_aggregator لدمج هذا الحدث الجديد مع السرد الحالي للجلسة، وإنشاء قصة متماسكة ومحدثة.
4. **طرح سؤال (الواجهة الأمامية):**
   * أثناء التسجيل (أو بعد إيقافه)، يكتب المستخدم سؤالاً ويرسله. هذا يستدعي \_viewModel.askQuestion().
   * يستدعي SessionViewModel \_apiService.askQuestion()، ويمرر session\_id والسؤال والنموذج والوضع المحددين.
   * يتم إرسال الطلب كـ POST إلى نقطة النهاية /session/query.
5. **الإجابة بناءً على السياق (الواجهة الخلفية):**
   * تسترد نقطة النهاية /session/query في الواجهة الخلفية السرد المجمع لـ session\_id المحدد.
   * تستخدم موجه contextual\_qa من configs/prompts.yaml، والذي يدرج سرد الجلسة وسؤال المستخدم في قالب لـ Gemini API. يخبر الموجه الذكاء الاصطناعي صراحةً "استخدم السياق المقدم فقط للإجابة."
   * ترسل الواجهة الخلفية إجابة الذكاء الاصطناعي مرة أخرى إلى التطبيق.
6. **عرض الإجابة (الواجهة الأمامية):**
   * يستقبل SessionViewModel الإجابة ويقوم بتحديث حقل الحالة \_currentAnswer.
   * يعيد SessionScreen بناء نفسه، ويعرض \_buildResultDisplay الإجابة السياقية للمستخدم.