التقرير الاستراتيجي الشامل للتحول المعماري: خارطة طريق إلى معايير الجودة Delivery Ways الانتقال بمشروع العالمية (Uber-Quality)

1. الملخص التنفيذي والتوجه الاستراتيجي

يهدف هذا التقرير البحثي الموسع إلى صياغة خارطة طريق موحدة، احترافية، وشديدة التفصيل لتحويل مشروع "Delivery Ways" يعتمد على المحاكاة، إلى منصة إنتاجية متكاملة تضاهي (MVP) من حالته الراهنة كنموذج أولي (DW_workspace المشار إليه بـ) (Forensic Technical) يستند هذا التحليل إلى دراسة جنائية تقنية .Uber في جودتها وموثوقيتها التطبيقات العالمية الرائدة مثل للوثائق المقدمة، وتحليل معمق للكود المصدري، مع دمج أفضل الممارسات الهندسية المستخلصة من مئات الأوراق البحثية (Analysis والمصادر التقنية المرفقة.[1]

يعتمد على أنماط معمارية (High-end Skeleton) "يُظهر الوضع الحالي للمشروع مفارقة جوهرية؛ فهو يمتلك "هيكلاً عظمياً فاخراً إلا أنه يعاني من ،(Monorepo) واستراتيجية المستودع الأحادي (Hexagonal Architecture) متقدمة مثل البنية السداسية إن الاعتماد المفرط على .(Data Integrity) وسلامة البيانات (Backend Logic) "ضمور عضلي حاد" في المنطق الخلفي يخلق شعوراً زائفاً بالأمان، حيث تبدو (In-Memory Persistence) والتخزين في الذاكرة (Stubs/Mocks) التطبيقات الوهمية الواجهات مكتملة بينما العمليات الفعلية غائبة أو محاكية.[1]

لا يعني مجرد تحسين الواجهات، بل يتطلب هندسة أنظمة معقدة تتعامل مع التزامن "Uber إن الوصول إلى "مستوى "بمنهجية "الصفر ثقة (Cybersecurity) والأمن السيبراني ،(Geospatial Data) والبيانات الجيومكانية ،(Concurrency) يرفض هذا التقرير أي حلول مؤقتة أو شاشات "ديمو"، ويضع خطة تنفيذية صارمة لاستبدال كل مكون وهمي بخدمة .(Zero Trust) حقيقية، قوية، وقابلة للتوسع.

---

2. التحليل الجنائي المعماري: تشريح الوضع الراهن

لفهم عمق الفجوة بين الواقع والمأمول، يجب أولاً تفكيك البنية الحالية للمشروع، وتحليل القرارات الهندسية التي اتخذت، وتأثيراتها المباشرة وغير المباشرة على قابلية التوسع والصيانة.

2.1 استراتيجية المستودع الأحادي (Monorepo) وتعقيدات Melos

يعتمد المشروع نهج المستودع الأحادي المدار بواسطة أداة **Melos**، وهو خيار استراتيجي سليم نظرياً للمشاريع الكبيرة التي تتطلب مشاركة الكود بين تطبيقات متعددة (تطبيق السائق، تطبيق العميل، لوحة التحكم). يقسم المشروع الكود إلى ثلاث طبقات رئيسية داخل مجلد /packages:

1. **Core Packages:** تحتوي على منطق العمل المشترك والكيانات الأساسية.
2. **Shims (Interfaces):** تعرف العقود والواجهات البرمجية، وتعمل كطبقة عزل.
3. **Implementations (Impl):** تحتوي على الكود الفعلي الذي ينفذ هذه الواجهات.[1]

لكي يفهم المطور .(Shim Hell) "ومع ذلك، كشف التحليل الجنائي أن هذا التنفيذ يعاني من تعقيد مفرط يُعرف بـ "جحيم المحولات الجديد ميزة بسيطة مثل "تسجيل الدخول"، فإنه مضطر للتنقل بين ثلاث إلى أربع حزم مختلفة (auth_shims, auth_core,

بشكل كبير ويقلل من سرعة التطوير (Cognitive Load) مما يرفع العبء المعرفي، (auth_supabase_impl
(Development Velocity).[1]

لربط الحزم pubspec_overrides.yaml الأخطر من ذلك هو أزمة إدارة التبعيات. يعتمد المشروع بشكل مكثف على ملف هذا النمط، وإن كان مريحاً أثناء التطوير المحلي، يمثل "قنبلة موقوتة" في خط أنابيب التكامل المستمر .(path:../) بمسارات محلية لا يمكنها حل هذه المسارات النسبية بنفس طريقة الجهاز (Codemagic أو GitHub Actions البيئات السحابية (مثل .(CI/CD) هذا الخرق يعني أن المشروع، في .(DevOps Parity) "المحلي، مما يؤدي إلى فشل البناء وكسر مبدأ "تكافؤ بيئة التطوير والتشغيل حالته الحالية، غير قابل للنشر الآلي، وهو شرط أساسي لأي نظام إنتاجي.[1]

2.2 وهم الوظائف: فخ الـ Stub Implementations

Stripe لعزل منطق العمل عن التفاصيل التقنية. نظرياً، هذا يسمح باستبدال (Adapter Pattern) يستخدم المشروع نمط المحولات دون تعديل الكود الأساسي. لكن عملياً، تم استغلال هذا النمط لإخفاء ديون تقنية ضخمة عبر استخدام حزم وهمية مثل PayPal بـ payments_stub_impl و pricing_stub_impl.[1]

هذه المكونات ليست مجرد عناصر نائبة؛ بل هي تزييف للواقع الوظيفي للنظام. عندما يطلب المستخدم رحلة، يقوم بتوليد سعر عشوائي محلياً دون أي اعتبار للمسافة الفعلية، حركة المرور، أو خوارزميات التسعير الديناميكي. pricing_stub_impl هذا يخلق فجوة خطيرة بين ما يراه المطور (نظام يعمل) وبين الحقيقة (نظام أجوف). الانتقال إلى الإنتاج يتطلب "استئصالاً جراحياً" لهذه يعطي نتائج "خضراء" كاذبة لا (E2E Tests) الوحدات واستبدالها بتكاملات حقيقية، حيث أن الاعتماد عليها في اختبارات التكامل تعكس متانة النظام في العالم الحقيقي.[1]

2.3 هشاشة البيانات: التخزين في الذاكرة

عبر ملفات (RAM) أظهر التحليل أن بيانات حيوية، مثل حالة الرحلة الحالية وسجل الطلبات، يتم تخزينها في ذاكرة الوصول العشوائي هذا يعني أن إغلاق التطبيق أو تعطل نظام التشغيل يؤدي إلى فقدان فوري وكامل .[1] in_memory_ride_repository.dart مثل للبيانات. في سياق تطبيق توصيل، هذا السيناريو كارثي؛ حيث قد يجد الراكب نفسه في منتصف الطريق دون أي سجل للرحلة على هاتفه، بينما يستمر السائق في القيادة دون وجهة مؤكدة.

لحفظ ملفات (Document Store) تشير الأدلة إلى أنه يتم التعامل معها كمخزن مستندات بسيط، Supabase حتى عند استخدام والقيود (Foreign Keys) يستفيد من العلاقات (Relational Engine) بدلاً من استخدامها كمحرك قواعد بيانات علائقي، JSON يتجاهل تماماً متطلبات (Frontend-first mindset) هذا النهج المرتكز على الواجهة .(Data Integrity) لضمان سلامة البيانات الأنظمة المالية واللوجستية التي لا تحتمل الخطأ أو فقدان البيانات.[1]

---

3. ركائز الهندسة الخلفية: بناء القلب النابض للنظام

بالكامل، (Backend Infrastructure) يجب إعادة بناء البنية التحتية الخلفية، "Uber للانتقال من هذا الوضع الهش إلى "مستوى مع التركيز على أربع ركائز أساسية: الهندسة الجيومكانية، التحكم في التزامن، إدارة الحالة، والأمن.

3.1 الركيزة الأولى: الهندسة الجيومكانية المتقدمة (PostGIS vs H3)

تطبيقات التوصيل ليست مجرد خرائط؛ بل هي أنظمة لوجستية معقدة تعتمد على دقة البيانات المكانية. يعتمد المشروع حالياً على حسابات البسيطة، والتي تفترض أن الأرض مسطحة أو كروية تماماً وأن Haversine أو صيغة (Euclidean Distance) المسافة الإقليدية الحركة تتم في خطوط مستقيمة.[1] هذا "الوهم الإقليدي" يؤدي إلى تقديرات كارثية للأسعار وأوقات الوصول، حيث يتجاهل شبكات الطرق،

.الاتجاهات، والعوائق الطبيعية

3.1.1 PostGIS: المحرك المكاني العلائقي

مما يحولها إلى قاعدة بيانات مكانية قوية. ،PostgreSQL وهو الامتداد القياسي لقواعد بيانات ،**PostGIS** الحل الجذري يبدأ باعتماد GiST المنفذة عبر) **R-Trees** وفهارس مكانية متقدمة مثل (Geometry Types) أنواع بيانات هندسية PostGIS يوفر indices).[2]

مما يجعل استعلامات مثل "جد أقرب ،(Bounding Boxes) بتجميع الكائنات القريبة في مستطيلات إحاطة R-Tree تسمح فهارس الذي يقتل الأداء.[4] (Full Table Scan) سائقين في دائرة نصف قطرها 5 كم" تتم بسرعة فائقة، بدلاً من المسح الكامل للجداول ،(pgRouting ضروري لتخزين مسارات الرحلات بدقة، وحساب المسافات الفعلية عبر شبكات الطرق (باستخدام PostGIS استخدام (Geofencing).[5] وتحديد المناطق الجغرافية بدقة

3.1.2 H3: نظام الفهرسة السداسي الهرمي

العالم H3 للتحليل والتحسين على نطاق واسع.[6] يقسم **H3** نظام Uber في التخزين والاستعلام الدقيق، طورت PostGIS بينما يتفوق لماذا السداسيات؟ لأن المسافة بين مركز أي سداسي ومراكز جيرانه الستة متساوية تماماً، .(Hexagons) إلى شبكة من الخلايا السداسية والمثلثات.[7] (S2/Geohash) وهي خاصية فريدة تفتقدها المربعات

وهي ضرورية لإنشاء خرائط ،(Gradients) وحساب التدرجات (Smoothing) مثالياً لعمليات التمهيد H3 هذه الخاصية تجعل يسمح بتجميع البيانات H3 كما أن [8].(Surge Pricing) وتحديد مناطق التسعير الديناميكي (Heatmaps) حرارية للطلب بدلاً من العمليات الهندسية المعقدة.[9] (bit integers-بسرعة هائلة باستخدام عمليات الأعداد الصحيحة (64 (Aggregation)

كطبقة الحقيقة لتخزين المواقع الدقيقة، **PostGIS** نستخدم .(Hybrid Approach) **القرار الاستراتيجي:** سنعتمد نهجاً هجيناً لتسهيل H3 كطبقة تحليلية وتسعيرية، حيث يتم تحويل إحداثيات السائقين والطلبات إلى فهارس **H3** المسارات، وحدود المناطق. ونستخدم وحساب الأسعار بناءً على العرض والطلب في كل "خلية".[10] (Matching) عمليات المطابقة السريعة

3.2 الركيزة الثانية: التحكم في التزامن (Concurrency & Locking)

تخيل أن طلباً واحداً يُبث لسائقين (أ) و (ب). .(Race Condition) "أكبر كابوس تشغيلي في تطبيقات التوصيل هو "حالة التسابق يضغط كلاهما على "قبول" في نفس اللحظة. بدون آلية قفل، سيقوم الخادم بمعالجة الطلبين بالتوازي، ويسند الرحلة لكليهما. النتيجة: سائقان يتجهان لنفس الراكب، وخصم مزدوج من البطاقة، وفقدان للثقة.[12]

3.2.1 استراتيجية الأقفال الموزعة (Redis Distributed Locks - Redlock)

طريقة لتنفيذ أقفال موزعة عبر عدة عقد **Redlock** هو مخزن بيانات في الذاكرة معروف بسرعته العالية. توفر خوارزمية Redis Redis.[13]

- **المميزات:** سرعة فائقة ،(Microseconds latency) إنتاجية عالية، مثالية للأنظمة الموزعة.
- **العيوب:** تعقيد التنفيذ (ضبط الوقت، الرموز المميزة)، واحتمالية نادرة لحدوث "Split-brain" في حالات انقطاع الشبكة، مما قد يؤدي لفقدان القفل أو تكراره.[13]

3.2.2 استراتيجية القفل المتشائم في قاعدة البيانات (PostgreSQL Pessimistic Locking)

يمكن للمعاملة قفل صف معين ،SELECT... FOR UPDATE باستخدام تعليمة .(ACID) ضمانات المعاملات PostgreSQL توفر في قاعدة البيانات، مما يمنع أي معاملة أخرى من تعديله حتى تنتهي الأولى.[15]

- **المميزات: اتساق قوي (Strong Consistency).** يضمن استحالة تعديل سجل الرحلة من قبل أكثر من عملية واحدة في نفس الوقت. البساطة في التنفيذ لأنها تعتمد على مدير الأقفال الداخلي لقاعدة البيانات.[16]
- **العيوب:** زمن استجابة أعلى قليلاً مقارنة بـ Redis، واستهلاك اتصالات قاعدة البيانات.

التوصية النهائية: في عملية **إسناد الرحلة** (Ride Assignment)، تعتبر صحة البيانات (Correctness) أهم من السرعة القصوى. خطأ واحد في الإسناد يكلف أكثر بكثير من تأخير بضعة أجزاء من الثانية. لذلك، سنعتمد **PostgreSQL Pessimistic Locking** في Redis لضمان عدم حدوث إسناد مزدوج مطلقاً. يمكن استخدام (Atomic Transaction) داخل معاملة ذرية (Rate Limiting) سيناريوهات أقل أهمية مثل تحديد معدل الطلبات.[14]

3.3 الركيزة الثالثة: إدارة الحالة والهندسة النظيفة (Clean Architecture)

لضمان قابلية صيانة الكود مع نمو المشروع، يجب الالتزام الصارم بمبادئ الهندسة النظيفة وفصل المسؤوليات.

3.3.1 طبقات الهندسة النظيفة

يجب إعادة هيكلة الكود لفصل الطبقات بوضوح تام [17]:

1. **طبقة العرض (Presentation Layer):** تشمل الـ Widgets ومزودات الحالة (Riverpod Providers). مسؤوليتها فقط رسم الواجهة وإرسال الأحداث. لا يجب أن تحتوي على أي منطق عمل.
2. **طبقة المجال (Domain Layer):** تحتوي على الكيانات (Entities) وحالات الاستخدام (Use Cases). هذا هو "عقل" التطبيق الذي يحتوي على قواعد العمل (مثلاً: "لا يمكن للعميل طلب رحلة إذا كان رصيده سالباً"). يجب أن تكون هذه الطبقة نقية وبدون أي تبعيات خارجية (مثل Flutter أو HTTP clients).
3. **طبقة البيانات (Data Layer):** تشمل المستودعات (Repositories) ومصادر البيانات (Data Sources). هي المسؤولة عن جلب البيانات من الـ API أو قاعدة البيانات المحلية، وتحويلها إلى كيانات المجال.

3.3.2 نقاء Riverpod

كشف التدقيق عن "تلوث" في استخدام Riverpod، حيث يتم خلطه مع ChangeNotifier[1]. هذا يخلق مشكلة "تعدد مصادر الحقيقة" (Two Sources of Truth).

- **الإجراء التصحيحي: Pure Riverpod.** فرض استخدام Riverpod في) Notifier أو StateNotifier يجب أن تكون هي المدير الوحيد للحالة. هذا يضمن قابلية الاختبار (Testability) ويسهل تتبع تدفق البيانات.[19] يجب منع تمرير BuildContext إلى طبقات منطق العمل تماماً.[20]

3.4 الركيزة الرابعة: الوضع الأمني وتحصين الواجهات (Security Posture)

تتعامل تطبيقات التوصيل مع بيانات حساسة للغاية (مواقع، مدفوعات، معلومات شخصية). يجب أن تبنى المنظومة الأمنية على مبدأ "عدم الثقة مطلقاً" (Zero Trust).[1]

3.4.1 ثغرة BOLA (Broken Object Level Authorization)

تُعد هذه الثغرة الأمنية رقم 1 في عالم الـ APIs.[21] تحدث عندما يقوم المهاجم بتغيير معرف المورد في رابط الـ API (مثلاً تغيير GET /orders/123 إلى GET /orders/124) للوصول إلى بيانات مستخدم آخر.

- **الحل الجذري:** لا يكفي التحقق من أن المستخدم "مسجل دخول". يجب أن تقوم الطبقة الوسيطة (Middleware) بالتحقق من **الملكية (Ownership)**: "هل المستخدم Y يملك الحق في الوصول للطلب X في كل طلب. يجب طرح السؤال: "هل المستخدم Y قبل إرجاع أي بايت من البيانات؟"[21]

3.4.2 ثغرة الإسناد الشامل (Mass Assignment)

تسمح أطر العمل الحديثة أحياناً بربط مدخلات العميل مباشرة بكائنات قاعدة البيانات. قد يستغل المهاجم ذلك لإرسال حقول غير متوقعة في طلب تحديث الملف الشخصي.[23] {"wallet_balance": 1000000} أو {"is_admin": true} مثل

- **الحل الجذري:** استخدام كائنات نقل البيانات (DTOs) لجميع المدخلات. يجب تحديد الحقول المسموح بتعديلها صراحة (Allow-listing). يمنع JSON الخام إلى دوال تحديث قاعدة البيانات.[23] منعاً باتاً تمرير كائنات

3.4.3 الامتثال والخصوصية (GDPR/DSR)

على الرغم من وجود شاشات لحذف الحساب، إلا أنها غير مرتبطة بوظائف حقيقية. الامتثال لقوانين الخصوصية يتطلب آليات فعلية لإخفاء البيانات للمستخدم مع الاحتفاظ بها للأغراض القانونية لفترة محددة) و **Soft Delete**: للحذف. يجب تنفيذ نوعين من الحذف مسح فيزيائي كامل بعد انتهاء الفترة القانونية).[1] **Hard Delete**)

---

4. خارطة الطريق الموحدة للتنفيذ (The Unified Execution Roadmap)

بناءً على التحليل الجنائي والركائز الهندسية، نقدم خارطة الطريق الموحدة. هذه الخطة تسلسلية وإلزامية؛ لا يمكن القفز فوق المراحل لأن "كل مرحلة تبني الأساس لما بعدها. الهدف هو تحويل المشروع من "نموذج تجريبي" إلى "منصة إنتاجية".

المرحلة 1: التثبيت والتنظيف (Stabilization & Cleanup)

المدة المقدرة: 3 أسابيع
الهدف: ضمان قابلية البناء (Buildability) والتنظيف "النزيف وتنظيف الديون التقنية المعمارية لإيقاف".

1. **إصلاح نظام التبعيات:**
   - إزالة الاعتماد الكلي على pubspec_overrides.yaml.
   - تحويل الحزم الداخلية لاستخدام مراجع Git (Git References) مع تثبيت الـ commit hash أو لضمان الثبات، نشرها في مستودع خاص.
   - ضمان نجاح أمر melos bootstrap في بيئة Docker نظيفة، لتحقيق **DevOps Parity**.[1]
2. **توحيد نظام التصميم:**
   - دمج مكتبات Bui و Bux المتناثرة في حزمة Design System موحدة.
   - تطبيق قواعد B-STYLE الصارمة وإزالة كافة استثناءات التحليل (Linter Exclusions).
3. **تصحيح المسارات:**
   - استبدال جميع الاستيرادات النسبية الخاطئة (import '../ui/ui.dart') باستيرادات حزم مطلقة (package:dw_ui/...) لمنع تداخل الطبقات.

المرحلة 2: تصليب البنية الخلفية (Backend Hardening)

المدة المقدرة: 5 أسابيع
الهدف: استبدال البيانات الوهمية بمنطق خادم حقيقي وآمن.

1. **ترحيل قاعدة البيانات:**
   - الانتقال من التخزين في الذاكرة/JSON إلى **PostgreSQL**.
   - تصميم مخطط بيانات (Schema) 3) مطبع (NF يدعم العلاقات المعقدة (Users, Drivers, Rides,

Transactions).
    - تفعيل امتداد **PostGIS** لدعم البيانات المكانية.
2. **تحديث نظام المصادقة:**
    - إزالة auth_stub_impl نهائياً.
    - تفعيل تكامل كامل مع **Supabase Auth** أو **Auth0**، بما في ذلك إدارة الجلسات، تجديد الرموز (Token Refresh)، والتحقق الثنائي (2FA).
    - تخزين الرموز (Tokens) في تخزين آمن مشفر (FlutterSecureStorage) بدلاً من التخزين البسيط.
3. **بوابة API والطبقة الوسيطة:**
    - بناء طبقة Middleware (سواء Cloud Functions أو خادم مخصص) لاعتراض جميع الطلبات.
    - تطبيق **Input Validation** (باستخدام DTOs) و **Rate Limiting** في هذه الطبقة لحماية النظام من الهجمات.[25]

المرحلة 3: التميز الجيومكاني والتشغيلي (Geospatial & Ops Excellence)

المدة المقدرة: 4 أسابيع
الهدف: تنفيذ منطق لوجستي ذكي يضاهي جودة Uber.

1. **ترقية المنطق الجيومكاني:**
    - استبدال حسابات المسافة الإقليدية بتكامل مع **Routing APIs** (Google Routes / Mapbox / OSRM) لحساب المسافة الفعلية ووقت القيادة.
    - استخدام استعلامات **PostGIS** المكانية للبحث عن السائقين في النطاق الجغرافي (Radius Search).
    - تجميع بيانات الطلب والعرض لأغراض التسعير الديناميكي والتحليلات.[6] **H3** خطوة متقدمة) دمج)
2. **التحكم في التزامن (Dispatcher):**
    - تطوير نقطة نهاية "قبول الرحلة" باستخدام **PostgreSQL Row-Level Locking** (SELECT FOR UPDATE) لضمان الذرية (Atomicity).
    - ضمان أن عملية "تخصيص السائق -> تحديث الحالة -> إشعار المستخدم" تتم ككتلة واحدة غير قابلة للتجزئة.
3. **إعادة هيكلة إدارة الحالة:**
    - إعادة كتابة تدفق RideRequest في تطبيق Flutter باستخدام **Pure Riverpod**.
    - ضمان تفاعل الواجهة مع تدفقات البيانات (Streams) القادمة من الخلفية (WebSocket/Supabase Realtime) بدلاً من التحديثات المحلية المتفائلة التي قد تتعارض مع الواقع.

المرحلة 4: الجاهزية المالية والقانونية (Production Readiness)

المدة المقدرة: 3 أسابيع
الهدف: تفعيل المدفوعات والامتثال للقوانين.

1. **البنية التحتية للمدفوعات:**
    - استبدال payments_stub_impl بـ SDKs حقيقية لـ Stripe/PayPal.
    - تنفيذ **Webhooks** آمنة على الخادم لتأكيد الدفع (عدم الثقة مطلقاً في رد تطبيق العميل بنجاح الدفع).
2. **المراقبة والرصد (Observability):**
    - استبدال عبارات print بنظام تسجيل منظم (Structured Logging).
    - ربط التطبيق بمنصات تتبع الأخطاء (Sentry/Crashlytics).
    - إعداد تنبيهات للفشل الحرج (مثل فشل حلقة توزيع الطلبات).
3. **الامتثال القانوني:**
    - تنفيذ منطق الحذف الناعم (Soft Delete) والحذف الصلب (Hard Delete) للامتثال لـ GDPR.
    - ضمان تشفير البيانات الشخصية (PII) في قاعدة البيانات (Encryption at Rest).

5. مواصفات التنفيذ التفصيلية: التذكرة الأولى

بناءً على التحليل، لا يمكن البدء بأي ميزة جديدة قبل إصلاح العمود الفقري للمشروع. التذكرة الأولى تركز على إزالة العوائق الهيكلية التي تمنع النشر والإنتاج.

التذكرة التنفيذية الأولى (The First Execution Ticket)

Ticket ID: DW-CORE-001
Title: Infrastructure Remediation: Remove Local Path Dependencies & Decouple Auth Stubs
Type: Epic / Architectural Refactor
Priority: Critical (P0) - Blocker for CI/CD Pipeline
Effort: 5 Story Points (Approx. 3 Days)
Assignee: Senior Backend/DevOps Engineer
السياق (Context):
يعتمد مشروع DW_workspace حالياً على تجاوزات مسارات محلية هشة (pubspec_overrides.yaml) وتطبيقات وهمية (auth_stub_impl)، مما يجعل عملية البناء والنشر السحابي مستحيلة. هذا الوضع يخلق سيناريو "يعمل على جهازي فقط" (Works on my machine)، وهو ما يتعارض جذرياً مع معايير الإنتاج الهندسية.1
الأهداف (Objectives):

1. **إزالة التبعيات المحلية:** التخلص التام من الاعتماد على المسارات المحلية (path:../) لتمكين البناء السحابي.
2. **تصليب المصادقة:** نقل وحدة المصادقة من "وضع المحاكاة" (Stub Mode) إلى "تطبيق حقيقي" (Real Implementation) قادر على حفظ جلسات المستخدمين بشكل آمن.
3. **نظافة الكود:** إصلاح أخطاء المحلل (Linter) الحرجة التي يتم تجاهلها حالياً لفرض بوابات الجودة.

نطاق العمل (Scope of Work):

أ. إعادة هيكلة إدارة التبعيات (Dependency Management Refactor):

- **تدقيق (Audit):** مسح جميع ملفات pubspec.yaml في طبقات core, shims, و impl. تحديد كل حالة استخدام لـ path:../.
- **إعادة هيكلة (Refactor):** استبدال هذه المسارات المحلية.
  - أو المسارات النسبية فقط ضمن هيكلية Melos استخدام النسخ المدار بواسطة، (dw_ui بالنسبة للحزم الداخلية (مثل overrides) بدون *Melos* المونوريبو المدعومة من.
  - بالنسبة للحزم الخارجية، استخدام **Git dependencies** مباشرة مع تحديد commit hash لضمان عدم تغيير الكود (Immutability).
- **تنظيف (Cleanup):** حذف ملف pubspec_overrides.yaml نهائياً من المستودع. إضافته إلى gitignore. لمنع رفعه مستقبلاً.
- **التحقق (Verification):** تشغيل melos bootstrap و flutter pub get في حاوية Docker نظيفة تماماً. يجب أن ينجح البناء دون أي تدخل يدوي.

ب. تصليب المصادقة (Auth Vertical Hardening):

- **إيقاف وترقية (Deprecation/Promotion):** وسم auth_stub_impl بـ @deprecated. إزالته من إعدادات البناء الافتراضية في main.dart. تعيين auth_supabase_impl (أو المزود الحقيقي المختار) كتطبيق افتراضي لنكهات

dev و prod.

- **أمن الإعدادات (Configuration Security):** إزالة أي بيانات اعتماد (Credentials) مكتوبة في الكود (Hardcoded). إعادة هيكلة الكود لقراءة عناوين ومفاتيح Supabase من متغيرات البيئة وقت البناء (--dart-define). أو ملفات .env غير مرفوعة على Git).
- **فحص الاستمرارية (Persistence Check):** تدقيق كود auth_supabase_impl للتأكد من استخدامه لـ FlutterSecureStorage لتخزين رمز JWT. التأكد من بقاء الرمز فعالاً بعد إعادة تشغيل التطبيق.

ج. نظافة الكود والتحليل (Code Hygiene & Linting):

- **إصلاح الاستيرادات:** تشغيل بحث واستبدال شامل لإصلاح أخطاء "Target of URI doesn't exist". استبدال الاستيرادات النسبية التي تعبر حدود الحزم (مثل import '../ui/ui.dart') باستيرادات حزم مطلقة (مثل import 'package:dw_ui/ui.dart').
- **تحليل صارم:** تفعيل قواعد Linter صارمة لحزمة auth. حل انتهاكات avoid-banned-imports عن طريق فرض أن auth_impl تستورد وحدات الميزات من auth_shims فقط، ولا تستورد أبداً من auth_impl مباشرة.

معايير القبول (Acceptance Criteria):

- [ ] **نجاح CI:** خط أنابيب GitHub Actions/Codemagic يستطيع سحب الكود وبناء ملف APK للأندرويد (flutter build apk --release) بدون أخطاء.
- [ ] **مستودع نظيف:** ملف pubspec_overrides.yaml غير موجود في المستودع البعيد.
- [ ] **مصادقة حقيقية:** يمكن للمستخدم التسجيل وتسجيل الدخول عبر التطبيق، ويظهر السجل في لوحة تحكم Supabase (إثبات الاتصال بالخلفية).
- [ ] **استمرارية الجلسة:** قتل التطبيق وإعادة تشغيله يحافظ على حالة "مسجل الدخول" للمستخدم دون الحاجة لإعادة إدخال البيانات.
- [ ] **سجلات الإنتاج:** سجلات بناء النسخة النهائية (Release Logs) تخلو تماماً من أي إشارة لتهيئة مزودات "Stub" أو "Mock".

---

6. الخاتمة

عند مفترق طرق حاسم. يمتلك المشروع أساساً معمارياً متطوراً ينم عن بعد نظر في التصميم "Delivery Ways" يقف مشروع المعياري والقابلية للتوسع، إلا أن هذا الهيكل يفتقر إلى "الجهاز العصبي والعضلي" الحقيقي—المنطق الخلفي الفعلي، تكامل البيانات، وبروتوكولات الأمان—اللازم للعمل في العالم الحقيقي.

DW-CORE-001، من خلال تنفيذ خارطة الطريق الاستراتيجية الموحدة هذه، وتحديداً البدء بعملية التثبيت الموضحة في التذكرة هذا التحول ليس مجرد كتابة كود؛ بل هو تبني فلسفة Uber. سينتقل المشروع من كونه نموذجاً أولياً متطوراً شكلياً إلى منصة قوية بجودة **الصحة التقنية قبل الراحة** و **هندسة الأنظمة قبل النمذجة السريعة**. الطريق شاق وصارم، ولكنه المسار الوحيد لإنتاج منتج قابل للتوسع والمنافسة في سوق يتطلب الكمال التقني.

Works cited

1. خارطة طريق لتطوير تطبيق توصيل.pdf
2. Choosing and Setting Up a Cloud PostGIS Solution for Felt: Navigating the Modern GIS Stack, accessed December 6, 2025, https://felt.com/blog/setting-up-postgis

3. PostGIS for GIS Data: Solve Spatial Challenges, Boost Performance, accessed December 6, 2025, https://www.cybrosys.com/postgres/gis-data-handling-with-postgis/
4. The Modern Geospatial Stack: From PostGIS to GeoAI | by Oleg - Medium, accessed December 6, 2025, https://medium.com/@KilgortTrout/the-modern-geospatial-stack-from-postgis-to-geoai-fcea95b311ca
5. PostgreSQL Best Practices: Supporting 25.1 Billion Parcels Per Day - Alibaba Cloud, accessed December 6, 2025, https://www.alibabacloud.com/blog/postgresql-best-practices-supporting-25-1-billion-parcels-per-day_597039
6. H3: Uber's Hexagonal Hierarchical Spatial Index | Uber Blog, accessed December 6, 2025, https://www.uber.com/blog/h3/
7. Geospatial Indexing Explained: A Comparison of Geohash, S2, and H3 | Call me Ben, accessed December 6, 2025, https://benfeifke.com/posts/geospatial-indexing-explained/
8. Visualizing City Cores with H3, Uber's Open Source Geospatial Indexing System | Uber Blog, accessed December 6, 2025, https://www.uber.com/blog/visualizing-city-cores-with-h3/
9. Real-time aggregation and joins of large geospatial data in HeavyDB using Uber H3 : r/gis, accessed December 6, 2025, https://www.reddit.com/r/gis/comments/1kbi261/realtime_aggregation_and_joins_of_large/
10. Amazon RDS for PostgreSQL now supports h3-pg for geospatial indexing - AWS, accessed December 6, 2025, https://aws.amazon.com/about-aws/whats-new/2023/09/amazon-rds-postgresql-h3-pg-geospatial-indexing/
11. Using Uber H3 Indexing Library in Postgres for Geospatial Data Analytics - Json Singh, accessed December 6, 2025, https://jsonsingh.com/blog/uber-h3/
12. How can I prevent race conditions using Redis? - Stack Overflow, accessed December 6, 2025, https://stackoverflow.com/questions/28513940/how-can-i-prevent-race-conditions-using-redis
13. Distributed Locks with Redis | Docs, accessed December 6, 2025, https://redis.io/docs/latest/develop/clients/patterns/distributed-locks/
14. Redis Locking vs Postgres | by Ankur Kothari - Medium, accessed December 6, 2025, https://medium.com/@ankurkothari/redis-locking-vs-postgres-0e156536dfaa
15. Transactional Locking to Prevent Race Conditions - Database Tip - SQL for Devs, accessed December 6, 2025, https://sqlfordevs.com/transaction-locking-prevent-race-condition
16. Redis Locks vs PostgreSQL Advisory Locks: A Comprehensive Comparison, accessed December 6, 2025, https://master-spring-ter.medium.com/redis-locks-vs-postgresql-advisory-locks-a-comprehensive-comparison-8a127aebc630

17. Flutter Clean Architecture: Build Scalable Apps Step-by-Step - Djamware, accessed December 6, 2025, https://www.djamware.com/post/68fd9dee1157e31c6604ab8f/flutter-clean-architecture-build-scalable-apps-stepbystep
18. Building Scalable Flutter Apps with Clean Architecture | by Survildhaduk - Medium, accessed December 6, 2025, https://medium.com/@survildhaduk/building-scalable-flutter-apps-with-clean-architecture-9395f0537d5b
19. Efficient State Management with Riverpod Flutter in Flutter Apps - Mindbowser, accessed December 6, 2025, https://www.mindbowser.com/state-management-riverpod-flutter/
20. Flutter State Management with Riverpod: A Complete Guide - Djamware, accessed December 6, 2025, https://www.djamware.com/post/68ddea278ec2f45f716d5a13/flutter-state-management-with-riverpod-a-complete-guide
21. API Security Risk - Top 10 Threats & How to Prevent Them - AppSentinels, accessed December 6, 2025, https://appsentinels.ai/blog/top-10-api-security-risks/
22. OWASP API Security Top 10 Risks - Wiz, accessed December 6, 2025, https://www.wiz.io/academy/owasp-api-security
23. Mass Assignment: When Your API Accepts Too Much Trust | by InstaTunnel - Medium, accessed December 6, 2025, https://medium.com/@instatunnel/mass-assignment-when-your-api-accepts-too-much-trust-2bd5d675e843
24. API Security Playbook: 10 Best Practices Developers Should Follow - Fyld, accessed December 6, 2025, https://www.fyld.pt/blog/api-security-10-practices-developers/
25. How to Secure APIs – 10 Best Practices to Follow | A10 Networks, accessed December 6, 2025, https://www.a10networks.com/blog/how-to-secure-apis-10-best-practices-to-follow/