ChatGPT

نظرة عامة على الوضع الحالي والخطة المقترحة

بناءً على مراجعة مشروع **Delivery Ways – Mobility Super App** الحالي، الأساس الهندسي نظيف ومستقر: كل الاختبارات الـ158 تمر بنجاح، ولا توجد أخطاء تحليلية أو استيرادات محظورة في `lib/` [1] [2] . ومع ذلك، تجربة المستخدم والمحتوى الوظيفي لا تزال نموذجية (Prototype). للمضي قدمًا نحو نسخة إنتاجية من مستوى Uber/Bolt، نحتاج إلى تنفيذ التعديلات التالية:

- **حالة المشروع الحالية:** تتضمن شاشة Home أولية مع بطاقات Ride/Parcels/Food ونشاط سابق تجريبي. مسار Ride تفاعلي جزئي (اختيار موقع Pickup/Destination مع أزرار، بدون خرائط حقيقية أو تتبع سائق). شاشات Parcels/Food حالياً عبارة عن نوافذ "قادم قريبًا" أو "demo" (مثال: *Coming Soon* في `lib/screens/_placeholders.dart` [3]). التدفقات الخاصة بالتسجيل والتوثيق موجودة بنيةً (codes وملفات دراسات) لكنها غير مفعّلة حقيقيًا. نظام التصميم (`DWTheme`, `DWColors`, `DwSpacing`, وغيرها) معرف عبر **design_system_shims** وطبقات Foundation/Components/B-UI، لكنه غير مفعّل على كامل التطبيق (على سبيل المثال، ملف `main.dart` يستخدم `ThemeData.fromSeed` بدلاً من تطبيق DWTheme الموحّدة [4]).

- **الأهداف المرجوة:** تحويل التطبيق إلى جودة إنتاجية كاملة، بما في ذلك:

- تفعيل واعتماد كامل **النظام التصميمي** الموحد (الألوان، الطباعة، المسافات، المكونات المشتركة مثل AppButton وAppCard) في جميع الشاشات [5] .
- مسار Ride مكتمل: اختيار موقع بدء ونهاية متصل بخدمات الموقع والخرائط (`mobility_shims` و`maps_shims`)، حساب أجرة تقديرية، وحالة الرحلة الكاملة (يبحث عن سائق → تم تعيين سائق → السائق في الطريق → الرحلة جارية → مكتملة) مع تتبع على الخريطة. (علماً أن الحزم الداعمة مثل `mobility_shims`, `mobility_adapter_geolocator`, `mobility_uplink_impl` موجودة وجاهزة للاستخدام [6] .)
- مسارات Parcels/Food معالجة: إزالة كل رسائل "Coming Soon" [3] ، واختيار إما بناء نموذج MVP بسيط (مثل إنشاء وشحن الطرود، أو تصفح طلبات الطعام بالبيانات الوهمية) أو إخفاء هذه التابات خلف Feature Flags إن لم تتوفر المواصفات بعد.
- تفعيل تدفقات **Onboarding/Auth/DSR**: عرض شاشات التعريف والتسجيل الفعلية (مثلاً 3–4 شاشات تعريفية + صفحة المصادقة باستخدام OTP) وفق الدراسات المرفقة. كذلك تفعيل شاشات الـData Subject Rights (تصدير/محو البيانات) باستخدام `accounts_shims` و`dsr_ux_adapter` المدمجين (ملفات `lib/screens/settings/dsr_*` جاهزة [7]).
- **التلميع وتجربة المستخدم**: إضافة حالات فارغة (Empty States) لكل صفحة (مثل Home، الطلبات، الدفع، التتبع)، وحالات خطأ موحّدة، وحركات انتقال سلسة (Hero, Animations) وتعليقات تفاعلية على الأزرار. الهدف هو تحقيق واجهة بسيطة وواضحة كمنتج تشغيلي، لا مجرد نموذج تجريبي.

مهام التحسين الرئيسية (بناءً على الترتيب المقترح)

1. **النظام التصميمي وواجهة التطبيق الشاملة (App Shell):**
2. دراسة تصميم الواجهة الحالية ومقارنتها مع مكتبة المكونات (DWTheme وDWTypography وDwColors)، مرفقة في `design_system_shims`) ودراسات UX. يجب تطبيق `DWTheme` بشكل كامل عبر جميع الشاشات بدلاً من `ThemeData.fromSeed` الحالي [4] .
3. توحيد الـScaffold العام للتطبيق (AppShell) وشريط التنقل السفلي (BottomNav) وشاشات التطبيق (AppBars) والأزرار والبطاقات والتنبيهات، وفق نظام 8pt grid ومقاييس الطباعة (تأكد من تفعيل المكونات القياسية مثل `AppCard.standard`, `AppButton` المستوردة عبر **B-UI** [5]).

4. تحقيق أساس بصري متناسق: استخدام الطبقات في `design_system_foundation` (DwColors, DwSpacing, DwTypography) وترجمة القيم من الـDWTheme، وضمان التباين والأبعاد الصحيحة (WCAG basics).

5. **مسار Ride (الانتقال والملاحة):**

6. تصميم واجهة المستخدم لمسار Ride كاملة: تشمل شاشة اختيار الموقع (Pickup/Destination) باستخدام مصادر الموقع في `mobility_shims` (خدمات الموقع، إذن الموقع) وربطها بخدمات الخرائط في `maps_shims`.
7. حساب الأجرة التقديرية عبر بيانات mock أو بواسطة اتفاقية مع الـ`mobility_uplink_impl` (مع تحديد وضوحه، حتى لو افتراضية).
8. بناء الحالات المختلفة لحالة الرحلة في الواجهة: البحث عن سائق → سائق محدد → السائق قادم → رحلة جارية → مكتملة. يمكن استغلال مزودات `mobility_shims` (مثل `BackgroundTrackingController`) لتغيير واجهة المستخدم تلقائياً حسب حالة الرحلة.

9. تمكين **التتبع الحي** على الخريطة: عبر ربط `MapViewBuilder` (كما في `MapsScreen`) بتحريك الكاميرا أو وضع العلامات على الخريطة أثناء حالة الرحلة الجارية. كلها تتم بالاعتماد على حزم الشيمات (شيفرات `mobility_adapter_*` و`maps_adapter_*`) بدون أي استيراد مباشر للـSDK.

10. **مسارات Foodg Parcels:**

11. إزالة جميع النصوص والعناصر الوهمية من نسخ العميل (راجع `lib/screens/_placeholders.dart` حيث تظهر لافتة "Coming soon" [3]).
12. **Parcels:** تصميم MVP بسيط يسمح بإنشاء طلب طرد جديد (اختيار تفاصيل الشحنة)، وعرض قائمة الطرود السابقة، وعرض تفاصيل كل طرد (باستخدام بيانات اختبارية أو mock API).
13. **Food:** بناء واجهة تصفح قائمة مطاعم أو قائمة وجبات (يمكن استخدام بيانات وهمية) مع شاشة تفاصيل طلب بسيطة. إن تعذر الإنجاز بسرعة، يمكن إخفاء تبويب Food مؤقتاً عبر Feature Flag أو التعليق على الوصول إليه.

14. في كل الأحوال، يجب التخلص من أي Placeholder ثابت وضمان دخول المستخدم إلى محتوى وظيفي حقيقي أو لا يرى القوائم أصلاً.

15. **تدفقات Onboarding/Auth وواجهات DSR:**

16. **Onboarding:** تفعيل مسار الشاشات التعريفية (على الأقل 3–4 شاشات) استنادًا إلى محتوى الدراسات المرفقة. بعد انتهاء المعاينة، الانتقال إلى شاشة المصادقة.
17. **Auth/OTP:** تطبيق واجهة مستخدم محترفة لتسجيل الدخول/التسجيل مع طلب رقم الهاتف أو البريد وإرسال رمز OTP (حتى لو تم استخدام Stub للـbackend). التأكد من تغذية الحالة وإدارة الأخطاء (مثل رمز منتهي الصلاحية) بشكل ملائم.
18. **DSR (حقوق البيانات):** تفعيل شاشات تصدير البيانات ومحوها (`/settings/dsr_export`, `/settings/dsr_erasure`) التي تعتمد على `accounts_shims` و`dsr_ux_adapter`. الأكواد موجودة في `lib/screens/settings/dsr_*` وهي جاهزة للاستخدام [7]، طالما تم تفعيل المشروع (استدعاء `NoOpDsrFactory` في main يتم استبداله بتنفيذ حقيقي أو stub بإشراف فريق accounts).

19. تصميم حالات فارغة أو رسائل مناسبة إذا كان المستخدم لم ينشئ حساباً بعد أو ليس لديه بيانات للتصدير/المحو. هذا يرفع مستوى الاحترافية ويجنب الرسائل العامة المبهمة.

20. **التلميع العام وتجربة المستخدم:**

21. **حالات فارغة (Empty States):** لكل صفحة (مثل Home عند عدم وجود رحلات سابقة، صفحة الطلبات الفارغة، صفحة المدفوعات الفارغة)، إنشاء عرض مخصص باستخدام مكونات `EmptyState` أو `UiEmptyState` من B-UI، لضمان تجربة متماسكة عند عدم وجود بيانات.
22. **حالات الخطأ:** تعريف صياغة موحدة لشاشات الخطأ (مثل فقدان الاتصال، خطأ من الخادم، رفض إذن الموقع) باستغلال مكونات `ErrorView` أو `UiErrorState` في B-UI [8] ، وتضمين أزرار إعادة المحاولة.
23. **حركات وانتقالات سلسة:** إضافة تأثيرات مصغّرة مثل انتقالات Hero للصور أو العناصر المشتركة بين الشاشات، ورسوم متحركة للزر عند الضغط (كبداية زر يتحول إلى مؤشر تحميل)، وتسلسلات ظهور قائمة منسدلة (`UiStaggeredList` مثلاً) لعرض عناصر القائمة، بما يدعم انطباع التطبيق المنتج والمنسق.
24. **ملاحظات تفاعلية:** ضمان أن الأزرار (AppButton) تظهر حالة "تحميل" عند إجراء عملية طويلة (مثلاً إرسال نموذج Ride)، وأنه ثمة ردود فعل بصرية (شبكة، ظل، أو تغيير لون) عند التفاعل. يمكن استخدام مكونات مثل `UiLoadingButton` أو `UiSpinner` من B-UI [9] للمساعدة في ذلك.

القيود والمعايير الهندسية

- **عدم المساس بالاختبارات والبنية الحالية:** يجب ألا تؤدي التعديلات إلى فشل أي من الـ158 اختبارًا الموجودة حالياً. إذا تطلب الأمر تحديث الاختبارات (مثلاً عند تغيير سلوك واجهة المستخدم)، فيجب الحفاظ على التغطية الجزئية والحرص على إضافة اختبارات جديدة خاصة بالمسارات الجديدة (Ride، Parcels, Food) بما لا يقل عن مستوى التغطية الحالي.
- **عدم كسر بنية الـshims والملفات:** مثلاً، لا يجب استيراد أي مكتبة خارجية (Stripe، Geolocator، Google Maps SDK، إلخ) مباشرة داخل مجلد `app/lib`؛ بل تُستدعى هذه الخدمات من خلال حزم الـshims والمعالجات الموجودة تحت `packages/`. المستودع معدّ مسبقاً لاستدعاء جميع الخدمات عبر طبقة تجريد (كما يؤكد تقرير الحالة: "Shims Architecture: All SDK access through shims" [2]).
- **القابلية للتجميع والتشغيل:** بعد الانتهاء من التعديلات، يجب أن يظل المشروع قابلًا للتشغيل كمجلد واحد (`DW_workspace`) بعد تنفيذ

```
flutter pub get
flutter test
flutter build apk --debug
flutter build ios --no-codesign
```

بنجاح دون أخطاء. هذا يتطلب المحافظة على تنظيم الـpubspec والتبعيات داخل الـworkspace والأدلة الوظيفية (الحزم/packages).
- **الالتزام بمعايير الجودة:** المحافظة على مستويات عالية من الأداء (تجنب عمليات حظر واجهة المستخدم أثناء التحميل)، ومراعاة أفضل الممارسات الأمنية (كما هو معمول به مثل TLS Pinning الموجود بالفعل)، وضمان وصول ذوي الاحتياجات الخاصة (WCAG مثل استخدام نصوص بديلة للأيقونات والكثير).

تعريف النجاح

نسخة الإنتاج تُعرف بأنها جاهزة عندما نحقق الشروط التالية:

- **نجاح كامل للاختبارات الحالية:** يجب أن يظل حجم الاختبارات وعددها (الـ158 اختبارًا) مثبتًا، دون فشل، بعد إجراء التعديلات (كما هو الحالي مع تحليل صفري للأخطاء [1]).
- **التجميع الناجح:** يتمكن المطور من تشغيل تطبيق Android وiOS بفلتر (debug) بدون أخطاء (`flutter build apk --debug` و`flutter build ios --no-codesign`).
- **غياب أي نصوص "Coming Soon" أو تجريبية:** جميع شاشات التطبيق تعكس وظائف حقيقية؛ ولا تظهر عبارات ثابتة مثل "coming soon" أو "demo". (على سبيل المثال، تم العثور على عبارة "Coming soon" في الكود المصدري [3] ويجب استبدالها بمحتوى حقيقي أو إخفاؤها).

- **اكتمال مسار Ride بسلاسة:** يستطيع المستخدم عمل رحلة كاملة بدءًا من إدخال العنوان وحتى انتهاء الرحلة بتتبع واضح على الخريطة وحساب أجرة، دون وجود عوائق في الواجهة (أي حالات الانتقال واضحة وتنقل بين الحالات بسلاسة).
- **الواجهات تعمل كمنتج حقيقي:** شاشات Home، Parcels، Food، Orders، Payments، Profile، Settings تظهر مرتبة ومتكاملة كما في تطبيق حقيقي، مع تصميم متناسق وأداء سلس (وفق هوية Delivery Ways)، وليس كمجرد نموذج أولي.
- **إجماع التصميم:** الواجهة موحدة من حيث الألوان والخطوط والفراغات والمكونات، بوضوح وبساطة تعادل تطبيقات رائدة (Uber/Bolt) [5] [1] .

عند استيفاء هذه النقاط، يصبح التطبيق جاهزًا للتسليم للعميل كنسخة إنتاجية. كافة هذه الملاحظات والخطوات تستند إلى بنية المشروع الحالية (حزم الشيمات، الدراسات المرفقة، والوثائق الفنية) دون إعادة اختراع الهندسة من الصفر، مع الحفاظ على نظافة الكود واستقرار البنية القائمة [10] [2] .

---

[1] [2] [6] [7] [10] PROJECT_STATUS_v3.2.1.md
https://github.com/abdulrhmanasami/DW_workspace/blob/7e29d9226adc4d75d27f8ff013c7a2d790dc21a1/docs/reports/PROJECT_STATUS_v3.2.1.md

[3] _placeholders.dart
https://github.com/abdulrhmanasami/DW_workspace/blob/7e29d9226adc4d75d27f8ff013c7a2d790dc21a1/lib/screens/_placeholders.dart

[4] main.dart
https://github.com/abdulrhmanasami/DW_workspace/blob/7e29d9226adc4d75d27f8ff013c7a2d790dc21a1/lib/main.dart

[5] [8] [9] ui.dart
https://github.com/abdulrhmanasami/DW_workspace/blob/7e29d9226adc4d75d27f8ff013c7a2d790dc21a1/B-ui/lib/ui/ui.dart