ChatGPT

تحليل المشروع العميق وتقرير عدم التوافق

أولًا: الفحص الشامل ومقارنة الكود مع دراسة المشروع

بعد مراجعة شاملة لكافة ملفات الكود في مستودع **shoobydo** ومقارنتها بدراسة المشروع والوثائق الفنية المتوفرة، تبيّن وجود عدد من نقاط عدم التطابق ومواطن الخلل بين ما هو مخطط له نظريًا وما تم تنفيذه عمليًا في الكود. فيما يلي أبرز النتائج:

- **اختلاف في الهيكلية والمعمارية:** المخطط التقني للمشروع (دراسة المشروع) يفترض اعتماد معمارية **ثلاثية الطبقات مع Microservices** لكل مكون [1] ، في حين أن التطبيق الحالي تم بناؤه كمنظومة متراصة (monolith) تجمع الواجهة الأمامية والخلفية في مستودع واحد. على سبيل المثال، الوثائق تقترح استخدام إطار **Flask** للطبقة الخلفية [2] ضمن هذه المعمارية، لكن الكود الفعلي يستخدم إطار **FastAPI** بديلًا (كما يظهر في تهيئة التطبيق الخلفي) [3] . هذا تغيير في تقنية التنفيذ قد يؤثر على طريقة بناء الخدمات المصغرة والتوسّعية المستقبلية.

- **تقنيات الواجهة الأمامية (Frontend) غير مطبّقة وفق الدراسة:** تشير **دراسة المشروع** إلى استخدام أحدث تقنيات React مع **TypeScript** بالإضافة إلى **Tailwind CSS** ومكتبة مكونات UI مثل **Shadcn/UI** وخواص تفاعلية من **Framer Motion** [4] لضمان واجهة غنية ومتناسقة. إلا أن الكود الحالي تخلّى عن Tailwind ومكتبة Shadcn، حيث تم **إزالة Tailwind CSS وتبسيط تصميم الواجهات عبر CSS عام بسيط** [5] . الواجهة الأمامية (Next.js 14 React) تستخدم بعض أصناف CSS محدودة (مثل `.btn` , `.table` , وغيرها) معرّفة بطريقة يدوية في ملف CSS عام بدلًا من إطار تصميم متكامل، مما يعني عدم الالتزام بالكامل بإرشادات **الهوية البصرية** المحددة (كالألوان والخطوط الموحدة المذكورة في الدراسة [6]). هذا يظهر تفاوتًا بين التصميم المخطط (الذي كان يفترض وجود نظام تصميم موحد وأصناف جاهزة) وبين التنفيذ الحالي.

- **نقص في تطبيق ميزات الأمن والصلاحيات:** ركّزت الدراسة التقنية على جوانب أمنية قوية مثل **تشفير البيانات الحساسة، نظام مصادقة Multi-Factor Authentication (MFA)، جلسات JWT، ونظام أدوار وصلاحيات** [7] . لكن الكود الحالي **لا يتضمن أي نظام توثيق أو صلاحيات** – جميع واجهات API مفتوحة دون حماية. لا توجد جداول أو نماذج للمستخدمين (Users) في قاعدة البيانات، ولا أي تحقق هوية على مستوى المسارات الخلفية. هذا يُعتبر خللًا جوهريًا مقارنةً بالمتطلبات الأمنية للمشروع. فعلى سبيل المثال، الدراسة ذكرت وجوب وجود جدول **Users** في قاعدة البيانات مع نظام IAM متقدم [8] [9] ، في حين أن التطبيق لا يحتوي إلا على جدول **الموردين** فقط [10] ، ولا أثر لأي نموذج مستخدم أو توكنات JWT في الكود.

- **قصور في نموذج البيانات مقابل الدراسة:** بحسب وثائق المشروع، هيكل قاعدة البيانات المخطط **يشمل جداول متعددة** (مثل: المستخدمين، الطلبات، المنتجات، العملاء، المعاملات، التحليلات) وعلاقات بينها [8] لضمان تطبيع البيانات وتغطية نطاق العمل الكامل. التنفيذ الحالي يقتصر فقط على **جدول "الموردين"** لتخزين بيانات المورد وملف الإكسل المرتبط به [10] . **لا يوجد تنفيذ لجدول الطلبات أو المنتجات أو العملاء** رغم أن الدراسة تفترض وجودها. هذا يعني أن كثيرًا من وظائف النظام المخططة (إدارة الطلبات، تتبع المنتجات، تحليل المبيعات، ...إلخ) غير موجودة بعد في الكود، مما يشكّل فجوة كبيرة بين الدراسة والتنفيذ.

- **فجوات في نطاق الـ API المقدم:** هيكلية الـ API المستهدفة وفق الدراسة تفترض مسارات RESTful منظمة تحت مسار موحد (مثال: `/api/dropship/...`) تشمل إدارة الطلبات والمنتجات والموردين والأسواق والتحليلات [11] . أما التطبيق الحالي فيوفّر **مجموعة محدودة من الـ API** تقتصر على صحة

النظام والتقارير والموردين فقط [12] . فعلى سبيل المثال، لا يوجد في الكود الحالي أي مسار لإدارة **الطلبات** `/orders` أو **المنتجات** `/products` أو **الأسواق** `/markets` كما جاء في الدراسة [11] ؛ ويقتصر نطاق الـ API الفعلي على: مسارات التقارير (`/reports/summary` , `/reports/kpis` , `/reports/costs`)، ومسارات الموردين (`/suppliers` CRUD ورفع الملفات) [12] ، ومسارات فحص الاتصال بقاعدة البيانات والكاش (`/db/ping` , `/cache/ping`) [13] [14] . هذا القصور يعني أن **أجزاء كبيرة من وظائف المنصة غير منفذة بعد** (مثل التكامل مع منصات التجارة، إدارة الكتالوج، معالجة الطلبات).

- **التكاملات الخارجية غير منفذة:** تضمنت خطة المشروع التكامل مع منصات تجارة إلكترونية خارجية مثل **Shopify, WooCommerce, Amazon, eBay** لتحديث المنتجات والطلبات بشكل آني [15] ، بالإضافة إلى بوابات الدفع (Stripe, PayPal, ... إلخ) وخدمات الشحن الأوروبية [16] . حاليًا لا يوجد في الكود أي تنفيذ لهذه التكاملات أو حتى واجهات ربط (APIs أو خدمات) تمهيدية لها. هذا غياب بارز مقارنة بالمخطط، إذ يفترض على الأقل وجود وحدات أو إعدادات جاهزة للتكامل (مثلاً مفاتيح API أو أصناف عملاء Clients لهذه المنصات) لكن لا شيء من ذلك متوفر بعد.

- **ملاحظات أخرى على مستوى الكود والبنية:** هناك بعض الأمور الإضافية المكتشفة خلال الفحص:

- **التعامل مع البيانات وملفات Excel:** يعتمد التطبيق حاليًا على قراءة ملفات Excel مباشرةً عند كل طلب تقرير (مثلاً `/reports/kpis` يقوم بفحص كل ملفات الـ Excel وحساب الصفوف [17] [18]). هذا التصميم يفي بالغرض في النموذج الأولي، لكنه **غير فعال** مع زيادة حجم البيانات، وكان من المتوقع حسب أفضل الممارسات نقل هذه البيانات إلى قاعدة البيانات أو التخزين المؤقت (الكاش) لضمان سرعة الاستجابة. بالفعل ذُكر في الدراسة اعتماد استيراتيجيات تخزين مؤقت متعددة المستويات لضمان الأداء [19] ، وهو ما لم يُنفّذ بعد (باستثناء توفر Redis غير المستخدم فعليًا حتى الآن).
- **وظيفة إعادة الفهرسة (Reindex) وتنظيف البيانات:** وظيفة إعادة الفهرسة للموردين تقوم بمسح ملفات Excel وإضافة أي ملف جديد لقائمة الموردين أو تحديث القائم [20] [21] ، لكنها **لا تحذف** من قاعدة البيانات أي مدخل لمورّد لم يعد ملفه موجودًا. هذا يعني احتمال تراكم بيانات يتيمة إذا حُذف ملف Excel يدويًا من المجلد.
- **التعامل مع الأخطاء والصلاحية:** الكود الحالي يتجاهل الأخطاء بصمت في عدة مواضع. على سبيل المثال، عند قراءة أوراق Excel في عملية الفهرسة، يتم تمرير أي خطأ دون تسجيله [22] [23] ، مما قد يصعّب تتبع مشاكل البيانات. أيضًا، القيود على مستوى قاعدة البيانات (مثل فريدية مسار الملف لكل مورد) غير معالجَة برسالة واضحة للمستخدم في الواجهة الأمامية – فإذا حاول المستخدم إضافة مورد بنفس ملف المسار الموجود سينتج خطأ من قاعدة البيانات بدون توضيح وافي في الواجهة (حاليًا يتم عرض رسالة الخطأ الخام فقط) [24] [25] .
- **اختبارات واستقرار النظام:** توجد مجموعة اختبارات وحدات أساسية تغطي الصحة العامة ورفع الملفات وعمليات CRUD للموردين [26] [27] . **إلا أن الاختبارات تتجنب بعض الحالات** (مثلاً تخطي اختبارات قاعدة البيانات إن لم تكن متوفرة) ولا توجد اختبارات شاملة لباقي الوحدات (لاختبار واجهات الطلبات أو المنتجات لأنها غير موجودة أصلًا). هذا يشير إلى الحاجة لتوسيع التغطية الاختبارية تماشيًا مع زيادة ميزات النظام.

خلاصة هذا الفحص: التطبيق في **مرحلة تأسيسية مبكرة** ، وقد حقق الأساسيات في إدارة الموردين وواجهة تقارير أولية، لكنه **غير متوافق بعد مع جزء كبير من دراسة المشروع ومتطلباتها** . هناك فروقات حرجة في الهيكلية والتقنيات المستخدمة، ونقص في الكثير من الخصائص الرئيسية المخطط لها.

ثانيًا: تقرير عدم التوافق (الفروقات الحرجة والتوصيات)

فيما يلي قائمة بالفروقات الحرجة بين مخرجات الكود الحالية ومتطلبات الدراسة، مع تحديد مواضع الخلل والتوصية الفورية لكل منها:

1. **عدم وجود طبقات وخدمات مطابقة للمعمارية المخططة:** الدراسة تقترح بنية Microservices مقسمة (مثلاً خدمات مستقلة للطلبات والمنتجات والموردين) [1] ، بينما الكود جمع كل شيء في خدمة واحدة.

2. موضع الخلل: هيكل المشروع/ تصميم التطبيق ككل.

3. **التوصية:** إعادة النظر في تقسيم المنظومة منطقيًا. يمكن بدء فصل المنطق بحسب النطاقات (Modules) داخل المشروع كخطوة أولى – مثال: فصل مكوّن إدارة الطلبات عن مكوّن إدارة الموردين – مع قابلية فصلها إلى خدمات مستقلة مستقبلًا عند ازدياد الحجم. ضمان أن يكون لكل نطاق طبقة واجهة **API** وطبقة بيانات خاصة به، وفق مبدأ **الفصل المنطقي** لتهيئة الطريق لـ microservices مستقبلاً.

4. **اختلاف تقنية الإطار الخلفي عن المخطط:** تم استخدام FastAPI في حين أن الدراسة الأصلية تبنّت Flask [2].

5. موضع الخلل: اختيار التقنية في الطبقة الخلفية (apps/backend).

6. **التوصية:** لا يعدّ ذلك خللًا وظيفيًا طالما يوفر FastAPI نفس واجهات الـREST المطلوبة. ولكن ينبغي **تعديل التوثيق والمعمارية لتتبنى FastAPI** رسميًا أو توفير مبررات التحول عن Flask. كما يُنصح بالتأكد من أن استخدام FastAPI ما زال متوافقًا مع باقي أجزاء الخطة (مثلاً التكامل مع ORM والتوسعية). إن لزم الأمر، توحيد القرار بالمضي مع FastAPI مع مراعاة تنفيذ جميع ميزات الأمان والتوسعة التي كانت ستُطبق في Flask.

7. **عدم اعتماد إطار تصميم واجهة (CSS Framework) مطابق للهوية البصرية:** الكود الحالي أزال Tailwind CSS [5] وخسر بذلك نظام التصميم الموحد المخطط له، مما يصعّب الحفاظ على الاتساق في الواجهة.

8. موضع الخلل: الواجهة الأمامية (apps/frontend) – التنسيق والتصميم البصري.

9. **التوصية:** اعتماد **نظام تصميم Frontend موحّد** . الأفضل العودة لتطبيق ما جاء في الدراسة باستخدام Tailwind CSS ومكتبة مكونات جاهزة (مثل Shadcn/UI) لضمان الاتساق وسرعة التطوير [4] . في حال تبيّن أن إزالته كان لأسباب وجيهة، فيجب على الأقل إنشاء **ملف CSS شامل للهوية** يعرّف المتغيرات والأصناف الرئيسية (ألوان العلامة التجارية والخطوط... إلخ [6]) واستخدامها بشكل موحّد عبر المكونات. الهدف هو الوصول لتصميم احترافي ومتسق يعكس الهوية التجارية على كل الصفحات (كما في صفحة Brand Identity الحالية).

10. **نقص طبقة الأمن والتوثيق:** لا يوجد أي نظام تسجيل دخول أو صلاحيات مستخدمين حاليًا، رغم تأكيد الدراسة على ذلك [7] . كل واجهات الـAPI مفتوحة مما يعرّض النظام لمخاطر.

11. موضع الخلل: غياب وحدة المستخدمين/auth في كامل التطبيق (Frontend وBackend).

12. **التوصية:** تنفيذ **نظام توثيق Authentication متكامل** بشكل فوري. ويتضمن ذلك:

    - إنشاء نموذج **مستخدم (User)** في قاعدة البيانات [8] يشمل الحقول الأساسية (مع تشفير كلمات المرور وفق خوارزمية قوية مثل bcrypt).
    - بناء واجهات تسجيل دخول JWT (مثلاً `/auth/login` تُرجع token) وتسجيل خروج وتجديد جلسة.
    - تطبيق **نظام صلاحيات Roles** (مثلاً مشرف، مدير، قارئ بيانات) وتقييد الوصول إلى واجهات الإدارة الحساسة (كإضافة أو حذف الموردين) بناءً على الدور.
    - إضافة طبقة Middleware في FastAPI للتحقق من JWT وصلاحية كل طلب.
    - اعتماد **سياسة أمان شاملة** كما في الدراسة: مثلاً إلزامية استخدام HTTPS، تضمين حماية من هجمات CSRF/XSS/SQL عبر المكتبات أو الممارسات المناسبة، وتفعيل **معدل طلبات (Rate Limiting)** لحماية الـAPI [28] .

13. **عدم تنفيذ الكثير من الوحدات الوظيفية الأساسية:** يشمل ذلك إدارة **الطلبات، المنتجات، العملاء، التقارير التحليلية التفصيلية** وغيرها، والتي ذكرتها الدراسة كأجزاء أساسية في المنصة.

14. موضع الخلل: غياب وحدات كاملة – لا كود ولا واجهات ولا جداول تخص هذه المجالات.

15. **التوصية:** الشروع في **بناء الوحدات الناقصة** ضمن خارطة طريق منظمة. على سبيل المثال، على التوالي:

    - **وحدة إدارة المنتجات:** إنشاء جدول **Products** في قاعدة البيانات مع الحقول اللازمة (اسم المنتج، التصنيف، المورد المرتبط، السعر، المخزون...إلخ) [8] ، وواجهات API (`/products`) لإضافة وتعديل وحذف المنتجات، بالإضافة إلى واجهة عرض في الـFrontend (صفحة "الكتالوج" بدل البيانات الثابتة الحالية [29] [30]).
    - **وحدة إدارة الطلبات:** تصميم جدول **Orders** يشمل تفاصيل الطلب (المنتجات المطلوبة، العميل، الحالة، التوقيت...إلخ) [31] ، وواجهات API (`/orders`) لمعالجة إنشاء وتحديث حالات الطلب، مع واجهة أمامية (صفحة إدارة الطلبات في لوحة التحكم) لعرض الطلبات وتحديث حالتها [11] .
    - **وحدة إدارة العملاء:** إضافة جدول **Customers** لحفظ بيانات العملاء وسجل مشترياتهم [8] ، وربطه بجداول الطلبات.
    - **وحدة التحليلات الموسّعة:** حاليًا يتم حساب بعض المؤشرات (عدد الملفات والصفوف) بشكل مباشر، لكن يُنصح بتنفيذ **نظام تحليلات أكثر تقدمًا**. قد يتضمن ذلك إنشاء جدول **Analytics** أو مستودع بيانات منفصل (Data Warehouse) يجمع بيانات الأداء بشكل مجمع [32] ، وتوفير واجهات `/analytics` مخصصة كما في التخطيط [11] لعرض **مؤشرات الأداء الرئيسية (KPIs)** واتجاهات المبيعات. يمكن أيضاً استخدام مكتبة رسوميات أوسع نطاقًا (مثل لوحات بيانات تفاعلية باستخدام Recharts وهي بالفعل مستخدمة جزئيًا الآن [33] [34]).

16. **الفجوة في التكاملات مع الأنظمة الخارجية:** عدم وجود أي جزء من الكود خاص بالتكامل مع منصات التجارة الإلكترونية أو بوابات الدفع رغم أهمية ذلك في الدراسة [15] [16] .

17. موضع الخلل: غياب موديولات التكامل Integration modules ضمن المشروع.

18. **التوصية:** التخطيط المسبق لتنفيذ **طبقة تكامل خارجية** . يمكن بدء ذلك بإنشاء هيكل Modules أو حزم مستقلة مخصصة لكل تكامل مهم:

    - مثلاً **وحدة تكامل Shopify** تتضمن عميل API يتعامل مع Shopify (جلب المنتجات، تحديث المخزون...).
    - **وحدة WooCommerce** وأخرى لـ**Amazon/eBay**... وهكذا.
    - نفس الشيء ينطبق على **بوابات الدفع** (Stripe، PayPal...) بحيث يكون هناك وحدة للتعامل مع مدفوعات Stripe (إنشاء شحنات، تحصيل المدفوعات) وأخرى لـ PayPal...إلخ.

    في البداية، قد تُنشأ هذه الوحدات بشكل مبسط (مجرد دوال تتعامل مع طلبات HTTP خارجية) مع إعداد **مفاتيح التكوين API Keys** في ملفات البيئة الآمنة. ثم يتم توسيعها لاحقًا. الهدف هو وضع أساس **قابل للتطوير** يتيح إضافة التكاملات تدريجيًا دون إعادة هيكلة كبيرة مستقبلًا.

19. **تحسين الأداء والتخزين المؤقت:** حاليًا بعض العمليات الثقيلة (كقراءة كل ملفات Excel لكل طلب تقرير) قد تؤدي إلى بطء مع نمو البيانات. الدراسة تطرقت إلى استراتيجيات **Caching** متعددة المستويات [19] وتحسينات قاعدة البيانات (فهارس، تقسيم جداول...إلخ) [35] لضمان الأداء العالي.

20. موضع الخلل: أسلوب القراءة المباشرة للملفات في `/reports/kpis` و `/reports/costs` وعدم استخدام الكاش بعد.

21. **التوصية:** تطبيق **آليات تحسين الأداء** قبل وصول النظام للإنتاج:

    - تفعيل استخدام **Redis** الموجود لتحقيق تخزين مؤقت للبيانات المتكررة. مثلاً: نتائج إحصائية الـKPIs يمكن تخزينها مؤقتًا وتحديثها بشكل مجدول بدل حسابها في كل مرة.

- نقل المنطق التحليلي الثقيل إلى **مهام خلفية غير متزامنة** أو خدمات منفصلة. على سبيل المثال، عند رفع ملف Excel جديد أو حدوث تغيير كبير، تُحدّث الإحصاءات في الخلفية وتُخزّن النتائج. يمكن استخدام مكتبة مهام مثل **Celery أو RQ** متصلة بـ Redis لتوزيع المهام الثقيلة.
- تحسين استعلامات قاعدة البيانات بإضافة **فهارس مناسبة** عندما تُضاف جداول جديدة (مثل فهرس على حقل تاريخ الطلب في جدول Orders لتسريع تحليل المبيعات حسب التاريخ). وضمان استخدام معاملات قاعدة البيانات بكفاءة (مثلاً استخدام `SELECT ... COUNT(*)` بدلاً من جلب كل الصفوف للعدّ حيثما أمكن).
- مراقبة أداء عملية رفع ملفات Excel وتحويلها – ربما اعتماد مكتبة **Pandas** ملائم حاليًا، لكن إذا زاد حجم الملفات، يفضّل التفكير في طرق معالجة بيانات أكثر كفاءة (مثل المعالجة على دفعات أو استخدام مكتبات أخف لقراءة بعض البيانات).

22. **تدعيم جودة الاختبار والضبط المستمر:** على الرغم من وجود بنية CI حالية لإجراء الاختبارات وتدقيق الشيفرة، هناك علامات على التغاضي عن بعض المشاكل (مثال: تم إعداد خطوةLint لكنها تتجاوز الأخطاء ولا تفشل البناء [36]). كذلك يتم تخطي بعض الاختبارات المتعلقة بقاعدة البيانات في ظروف معينة [37] [27] .

23. موضع الخلل: إعدادات خط التجميع (CI) وجودة الاختبارات.

24. **التوصية:** اعتماد **منهجية اختبار صارمة** وتحسين خط CI/CD:

- إصلاح جميع أخطاء الـLint الحالية في مشروع frontend ثم **تفعيل فشل البناء عند أي خطأ Lint** لضمان التزام الجميع بالمعايير البرمجية.
- توسيع تغطية الاختبارات لتشمل السيناريوهات الجديدة: بعد إضافة الوحدات المقترحة (منتجات، طلبات...إلخ) يجب كتابة اختبارات وحدة وتكامل لها. أيضًا يُنصح بإضافة اختبارات للواجهة الأمامية (مثلاً باستخدام React Testing Library أو Cypress للاختبارات التكاملية) لضمان عمل تدفقات المستخدم بشكل صحيح.
- تحسين إعدادات CI لتشغيل **خدمات قاعدة بيانات وهمية** (مثلاً Postgres و Redis في بيئة الاختبار عبر Docker) حتى لا يتم تخطي أي اختبار يعتمد على قاعدة البيانات. هذا يضمن اكتشاف أي مشكلة تكامل بين الخدمة وقاعدة البيانات مبكرًا.
- إعداد **بيئة Stage** أو اختبار شبيهة بالإنتاج يمكن النشر إليها تلقائيًا بعد نجاح الاختبارات، وذلك لاختبار التكاملات الخارجية يدويًا إن أمكن قبل النشر للإنتاج.

25. **التوثيق والتوافق مع المتطلبات التنظيمية:** الوثائق القانونية (سياسات الخصوصية والشروط) جاهزة ضمن المشروع [38] ولكن يجب التأكد من عكسها فعليًا في الواجهة الأمامية (مثلاً صفحة روابط للخصوصية والشروط). كما أن الامتثال للأنظمة (مثل GDPR) يتطلب توفير خصائص في النظام (مثل إمكانية حذف بيانات المستخدم، وخاصية الموافقة على جمع البيانات).

26. موضع الخلل: تكامل المتطلبات القانونية والتنظيمية في التطبيق العملي.

27. **التوصية:** إضافة ما يلزم لجعل التطبيق **ملتزمًا باللوائح الأوروبية** كما هو مخطط:

- تضمين صفحات لإظهار سياسات الخصوصية والشروط للمستخدمين، وتأكيد حصول الموافقة منهم (مثلاً نافذة إشعار بسياسة الكوكيز عند الدخول الأول).
- تنفيذ ميزات مثل **حق النسيان** للمستخدمين (إنشاء إجراء لحذف بيانات المستخدم عند الطلب) لضمان الالتزام بـ GDPR [39] .
- تعيين وتوثيق مسؤول حماية بيانات افتراضي (DPO) وإجراءات للإبلاغ عن أي خرق أمني، كما جاءت به الدراسة، ضمن دليل سياسات داخلي.
- التأكد أن جميع البيانات الحساسة (مثلاً بريد المستخدم أو عناوينه عند إضافتهم مستقبلًا) مشفرة في قاعدة البيانات [40] ، واستعمال بروتوكول TLS في الاتصالات (في التطوير المحلي ممكن الاستثناء، لكن التحضير للنشر الإنتاجي يجب أن يتضمن SSL).

28. **خارطة الطريق المستقبلية (مقترح تحسين مستمر):** على الرغم من أن النقاط أعلاه تغطي الإصلاحات المطلوبة للوصول إلى مستوى متوافق مع الدراسة الحالية، يجب أيضًا **تحديد مراحل تطوير مستقبلية** كما أوصت الدراسة لتحقيق رؤية المنصة المتكاملة.

- التوصية: تقسيم خطة الإصلاح والتطوير إلى **مراحل**:
- **مرحلة أولى (قصيرة المدى):** التركيز على سد الفجوات الحرجة – بناء الوحدات الناقصة (منتجات، طلبات، مستخدمين) وتطبيق الأمن وتحسين الهيكلية الأساسية. الهدف جعل النظام يعمل بكل الوظائف الأساسية المتوقعة (CRUD كامل لكل الكيانات الرئيسية، واجهات وتقارير أساسية) خلال 2-3 أشهر [41] .
- **مرحلة ثانية (متوسطة المدى):** التكامل مع الخدمات الخارجية الرئيسية (على الأقل منصة تجارة إلكترونية واحدة وبوابة دفع واحدة)، وتحسينات الأداء والاستقرار (التخزين المؤقت، الاختبارات، إلخ)، إضافة ميزات متقدمة كتحسين الأسعار التلقائي أو مزايا الذكاء الاصطناعي الأولية إن أمكن [42] . هذه تمتد خلال 4-6 أشهر التالية.
- **مرحلة ثالثة (طويلة المدى):** توسيع نطاق المنصة بابتكار ميزات تنافسية مثل **تحليلات تنبؤية بالذكاء الاصطناعي** (مثلاً التنبؤ بالطلب، Chatbot خدمة عملاء ذكي) [42] ، **تكامل أجهزة إنترنت الأشياء** (لتتبع المخزون) [43] ، ودعم أسواق إضافية أو لغات أخرى. هذه المرحلة تجعل المنصة "تحفة هندسية" رائدة، وقد تمتد على 6 أشهر أو أكثر، مع تقييم مستمر للعائد على الاستثمار المحقق [44] [45] .

ثالثًا: خطة الإصلاح والبناء (خارطة طريق تنفيذية)

استنادًا إلى التحليل أعلاه، نعرض فيما يلي **خطة منهجية لإصلاح وتطوير المشروع** ، على شكل سلسلة **طلبات / مهام جاهزة للتنفيذ** بالترتيب والأولوية. تم صياغة هذه المهام بطريقة احترافية مثل تذاكر التطوير في الشركات الكبرى، بحيث يمكن اعتمادها مباشرةً كخارطة طريق لإعادة المشروع إلى المسار الصحيح المتوافق مع الدراسة:

1. **إعادة هيكلة المشروع إلى طبقات واضحة:** قم بتنظيم الكود ضمن طبقات (Layered Architecture) أو وحدات (Modules) بناءً على نطاق العمل. على سبيل المثال، إنشئ مجلدات مستقلة لـ **الموردين**، **المنتجات**، **الطلبات** ضمن المشروع الخلفي مع فصل نماذج البيانات Data Models، وواجهات API، ووظائف الخدمة لكل منها. تأكد أن كل وحدة يمكن فصلها مستقبلًا في خدمة مستقلة (Microservice) دون تغييرات جذرية – هذا يتضمن تحديد واجهات تواصل واضحة بين الوحدات (مثلاً قد يتواصل طلب مع مورد لجلب بيانات المورد عبر طبقة خدمة بدلاً من استعلام مباشر). هدف هذه الخطوة تحسين قابلية الصيانة والتوسعة [1] .

2. **تأسيس نموذج البيانات الكامل في قاعدة البيانات:** باستخدام **SQLAlchemy وأدوات Alembic للهجرات**، أنشئ جميع الجداول الناقصة وفق تخطيط البيانات في الدراسة [8] :

3. أضف جدول **Users** (مع حقول: معرف، اسم مستخدم/بريد إلكتروني، كلمة مرور مشفرة، دور role، ...).
4. أضف جدول **Products** (مع الحقول الأساسية المذكورة أعلاه وربط المنتج بمورّد عبر مفتاح أجنبي Supplier_id).
5. أضف جدول **Orders** (يرتبط بجدول Users كعميل وبجدول Products عبر جدول وسيط أو حقل قائمة منتجات الطلب OrderItems). احفظ في جدول Order معلومات الحالة (جديد، معالج، مشحون...إلخ) وتاريخ الإنشاء والتحديث.
6. أضف أي جداول أخرى داعمة مطلوبة (مثل **Customers** إن كان العملاء كيان منفصل عن المستخدمين، **OrderItems** لتفاصيل منتجات كل طلب، **Transactions** للمدفوعات... بناءً على ما ورد في الدراسة).

7. بعد تعريف النماذج، أنشئ **هجرات Alembic** جديدة لضمان تحديث قاعدة البيانات بسهولة. تأكد من إضافة **علاقات (Foreign Keys)** وفهارس ملائمة على الحقول المهمة (مثل فهرس على email في Users، وعلى order_date في Orders) [46] .

8. **تطبيق نظام المصادقة والصلاحيات:** طوّر وحدة **Authentication** متكاملة:

9. أضف **واجهة تسجيل** (`/auth/register`) لإنشاء مستخدم جديد (إن كان التطبيق موجّهًا لعامة المستخدمين)، مع مراعاة التحقق من صحة البيانات (Validations) وتشفير كلمة المرور قبل الحفظ.
10. أضف **واجهة تسجيل الدخول** (`/auth/login`) تقوم بالتحقق من بيانات المستخدم وإصدار **JSON Web Token (JWT)** يحمل هوية المستخدم وصلاحياته. استخدم مفتاح سري (SECRET_KEY) مخزّن في إعدادات البيئة لتوقيع التوكن. حدد عمرًا للتوكن (مثلاً 1 ساعة) وآلية للتجديد (refresh token) إن لزم.
11. فعّل **Middleware للتحقق من JWT** على جميع مسارات الـAPI المحمية. مثلًا: يجب أن يكون تقديم التوكن في ترويسة Authorization شرطًا للوصول إلى مسارات إدارة الموردين والمنتجات والطلبات (باستثناء ربما مسارات عامة كالتقارير العامة أو تسجيل المستخدم).
12. نفّذ **نظام Roles/Permissions**: حدد أدوارًا (مثل "مشرف" يمتلك كل الصلاحيات، "مدير منتجات"، "مدير طلبات" بصلاحيات جزئية، "عميل" بصلاحيات محدودة لرؤية طلباته فقط، ...). طبق ذلك بفحص الدور ضمن كل Endpoint حساس. مثلًا: فقط المستخدم ذو الدور Admin يمكنه حذف مورد أو منتج. يمكن تخزين الدور ضمن حقل في جدول Users واستخدامه بعد فك شفرة JWT.

13. أضف **اختبارات وحدة وتكامل** لضمان أن المسارات محمية فعليًا (جرب طلب موارد دون JWT وتوقع 401 Unauthorized، وجرب بJWT بصلاحيات غير كافية وتوقع 403 Forbidden).

14. **توسيع واجهات الـAPI لتشمل المنتجات والطلبات:** بالاستفادة من الوحدات الجديدة ونماذج البيانات:

15. أنشئ مسارات **إدارة المنتجات** ضمن FastAPI (`POST /products` , `GET /products` , `PUT /products/{id}` , `DELETE /products/{id}`) مع ربطها بالمنطق وقاعدة البيانات. تأكد من تطبيق نفس النمط المتبع في الموردين (استخدام Pydantic Schemas للمدخلات والمخرجات لضمان التحقق من البيانات). هذه الواجهات تتيح للمشرف إضافة منتجات جديدة وتعديلها وحذفها.
16. أنشئ مسارات **إدارة الطلبات** (`POST /orders` , `GET /orders` , `GET /orders/{id}` , `PUT /orders/{id}` وربما `DELETE /orders/{id}` إذا كانت السياسة تسمح بإلغائها). هذه الواجهات ستمكّن النظام من استقبال طلبات جديدة (عن طريق واجهة أمامية أو تكامل API مع المتاجر) وتحديث حالاتها. تأكد من أن إنشاء طلب جديد يربط بالعميل المناسب (المصادق) ويُنشئ قيودًا صحيحة مع المنتجات المطلوبة.
17. أضف مسار `/orders/{id}/status` (اختياري) لتحديث حالة الطلب بشكل مباشر (مثلاً تغيير الحالة إلى "مُرسل" أو "مكتمل").

18. نفّذ **قواعد العمل** اللازمة ضمن هذه الواجهات: مثلاً عند إنشاء طلب تحقق من المخزون المتوفر لكل منتج وخصم الكمية المطلوبة، وعند حذف منتج تأكد من التعامل مع علاقته بالطلبات (ربما منع حذفه إن كان هناك طلبات مرتبطة، أو وضع علامة تعطيل بدلاً من الحذف الفعلي). هذه التفاصيل مستمدة من المعرفة بأن النظام يجب أن يحافظ على **تكامل البيانات** والعمليات التجارية الصحيحة.

19. **تحسين الواجهة الأمامية وإضافة الصفحات الناقصة:** بناءً على التحديثات أعلاه، قم بتطوير صفحات تكميلية في تطبيق Next.js:

20. أنشئ صفحة **إدارة المنتجات** (`/products`) ديناميكية بدل الصفحة الثابتة الحالية [29] [30] . اجعلها تعرض قائمة المنتجات من قاعدة البيانات عبر طلب `/products` GET، مع إمكانية إضافة منتج جديد (نموذج في الواجهة يستدعي POST /products) وتعديل الموجود (مثلاً نافذة أو حقل تعديل داخل الجدول) وحذف منتج. هذه الصفحة شبيهة بفكرة صفحة الموردين الحالية لكن لمجال المنتجات.
21. أنشئ صفحة **إدارة الطلبات** (`/orders`). تحتوي جدولًا يعرض أحدث الطلبات (رقم الطلب، اسم العميل أو بريده، المبلغ الإجمالي، الحالة الحالية، التاريخ...). أضف إمكانية تصفية الطلبات حسب الحالة أو التاريخ. عبر الواجهة يمكن تمكين **تحديث حالة الطلب** (مثلاً قائمة منسدلة لتغيير الحالة، ترتبط بوظيفة تستدعي PUT /orders/{id} لتحديث الحالة).
22. أضف صفحة **تسجيل الدخول** (`/login`) ضمن التطبيق الأمامي. هذه الصفحة تسمح للمستخدم بإدخال بياناته واستلام JWT (يمكن تخزين التوكن في Local Storage أو كوكي حسب القرار الأمني). بعد تسجيل

الدخول الناجح، يوجّه المستخدم للوحة التحكم Dashboard أو الصفحة المناسبة، وتصبح الطلبات اللاحقة موقّعة بالتوكن تلقائيًا (مثلاً باستخدام **خاصية fetch API** مهيأة عالميًا لإضافة التوكن في الترويسات).

23. حدّث شريط التصفّح العلوي في الواجهة (`layout.tsx`) لإظهار روابط الصفحات الجديدة (مثل "Orders" و "Products") إن لم تكن موجودة [47] وإضافة عنصر **"تسجيل خروج"** إن كان المستخدم مسجلاً دخوله (زر يقوم بمسح التوكن وإعادة توجيه المستخدم لصفحة login).

24. **تحسين التصميم والتجربة:** طبّق نظام التصميم الموحد المتفق عليه في الخطوة (3). راجع جميع الصفحات الحالية (Dashboard, Suppliers, Costs, Analytics, Brand...إلخ) وتأكد من استخدام الألوان والخطوط الصحيحة [48] [6] ، وتوحيد نمط الأزرار والجداول عبر المكوّنات. مثلاً، عرف مكوّن Button عام بمظهر موحد (`<Button className="btn">`) واستخدمه بدلاً من الأزرار العادية. أضف **تحسينات تفاعلية** بسيطة لجعل التجربة أكثر سلاسة (مثل مؤشر تحميل أثناء جلب البيانات، رسائل تأكيد عند الحذف، حوارات تأكيد modal عند تنفيذ عمليات خطرة كالحذف). ويمكن إدخال **Framer Motion** تدريجيًا لإضفاء حيوية على العناصر (مثل تحريك القوائم المنسدلة أو البطاقات عند ظهورها).

25. **تنفيذ التكامل مع منصة تجارة إلكترونية (نموذج أولي):** كخطوة لإثبات صحة التكاملات الخارجية، ابدأ بتكامل واحدة من المنصات المذكورة (مثلاً **Shopify** كونها مشهورة):

26. أنشئ **موديول تكامل Shopify** في المشروع الخلفي يتضمن عميل API للتواصل مع Shopify (استخدم مكتبة رسمية إن وجدت أو HTTP requests مع REST API لShopify). وفّر في هذا الموديول وظائف مثل `sync_products_from_shopify()` لجلب منتجات المتجر و`push_order_to_shopify(order)` لإرسال طلبات من منصتك إلى Shopify في حال رغبتم بذلك.
27. أضف **إعدادات ضبط** في ملف بيئة (.env) لرموز API اللازمة (مثل SHOPIFY_API_KEY, SHOPIFY_PASSWORD, SHOPIFY_STORE_URL...) بحيث يمكن تدويرها بسهولة.
28. جرّب استدعاء إحدى وظائف التكامل عبر واجهة إدارية (مثلاً يمكنك إضافة زر في لوحة التحكم "مزامنة المنتجات" يستدعي مسارًا خلفيًا جديدًا `/integrations/shopify/sync` ليقوم بجلب المنتجات وتخزينها في قاعدة بياناتك). اجعل نتيجة العملية تظهر للمستخدم (عدد المنتجات المضافة أو المحدثة...).

29. هذه المهمة ستركز على جانب واحد كتجربة ، وسيتم تعميم نفس النمط على باقي المنصات لاحقًا. الهدف هنا هو ضمان أن هيكلية التطبيق **قابلة لإضافة التكاملات بسهولة** بدون تعديل جوهري – أي عن طريق إضافة وحدات جديدة ومسارات تكامل دون كسر شيء قائم.

30. **تعزيز مستوى الأمان والامتثال:** بعد إضافة نظام التوثيق والتوسع في الميزات، تأكد من تدعيم الجوانب الأمنية كما يلي:

31. قم بتفعيل **تشفير الاتصال SSL/TLS** في بيئة التشغيل الفعلية (Production). يمكن استخدام وكيل عكسي مثل Nginx مع شهادة Let's Encrypt عند النشر. هذا ضروري خصوصًا إذا سيتم تمرير بيانات حساسة كبيانات الدفع مستقبلاً [49] .
32. نفّذ عمليات **التدقيق الأمني** المذكورة في الدراسة: مثل تسجيل كل العمليات الحساسة (إضافة/حذف مورد أو منتج، تحديث حالة طلب) في **سجلAudit Log🗒** مع طابع زمني والمستخدم القائم بالعملية، للمراجعة الأمنية عند الحاجة [28] [50] .
33. أضف آليات **النسخ الاحتياطي الدوري** لقاعدة البيانات (يمكن كتابة سكربت بسيط لأخذ نسخة احتياطية يومية من Postgres وحفظها في وحدة تخزين آمنة، أو استخدام حلول سحابية لذلك) [40] . اختبر استراتيجية الاستعادة من النسخ الاحتياطية لتضمن فعالية خطة التعافي من الكوارث [51] .

34. تأكد من أن **سياسات الخصوصية** المعلنة مطبقة: مثلاً عدم الاحتفاظ بسجلات حساسة لفترات أطول من اللازم، وإمكانية تصدير بيانات المستخدم له عند الطلب (حق قابلية نقل البيانات في GDPR). هذه قد تكون ميزات تحتاج دعم برمجي مستقبلاً (مثل صفحة لحساب المستخدم تعرض بياناته مع زر "تحميل بياناتي").

35. **تحسين الأداء والتوسع الأفقي:** مع بدء استخدام النظام بشكل أكبر، قم بإدخال تحسينات الأداء التالية كما تقتضي أفضل الممارسات:

36. فعّل نظام **الكاش في طبقة التطبيق**: استخدم Redis لتخزين نتائج الاستعلامات الشائعة (مثلاً نتائج `/reports/summary` و`/reports/kpis` يمكن تحديثها كل ساعة بدل حسابها عند كل طلب). استخدم مفاتيح cache واضحة (مثل `kpis_summary_cache`) وقم بتفريغها أو تحديثها عند حدوث تغيير كبير (كرفع ملف Excel جديد عبر `/upload`).
37. نفّذ **التقسيم Pagination** على الواجهات التي قد تزداد بياناتها مع الوقت. مثال: في واجهة عرض الطلبات `/orders`، قم بعرض 50 طلبًا في الصفحة مثلاً مع إمكانية التنقل بين الصفحات عبر query params في API (`/orders?page=2&page_size=50`). هذا سيمنع استرجاع آلاف السجلات دفعة واحدة مما يحسّن الأداء ويخفف الحمل على الواجهة.
38. راقب أداء **استعلامات قاعدة البيانات** باستخدام أدوات مراقبة (يمكن تفعيل تسجيل الاستعلامات البطيئة في Postgres أو استخدام مكتبة Profiling في SQLAlchemy). إذا لوحظت أي استعلامات بطيئة، حسّنها بإضافة الفهارس أو بإعادة صياغة الاستعلام (مثل استخدام join بدلاً من استعلامات فرعية متعددة).

39. خطط من الآن لكيفية **التوسّع الأفقي** للخادم الخلفي: كون التطبيق يستخدم FastAPI، يمكن نشر نسخ متعددة منه وراء موازن تحميل Load Balancer إذا زاد عدد المستخدمين. تأكد أن التطبيق **عديم الحالة stateless** قدر الإمكان (باستثناء الجلسات المخزنة في JWT/cookies) لتسهيل التوسع. نفس الأمر للواجهة الأمامية (توسيع نطاق Next.js مثلاً باستخدام Vercel أو Node cluster).

40. **اختبارات شاملة وضمان الجودة قبل الإصدار:** بعدما تصبح الميزات الأساسية مكتملة، ركّز على **جولة اختبارات شاملة (QA)**:

41. قم بإعداد **بيئة اختبار متكاملة** (Integration Testing Environment) حيث يتم نشر نسخة من الخدمات (مثلاً باستخدام Docker Compose تشمل التطبيق الخلفي، قاعدة بيانات، Redis، وربما خدمة تكامل وهمية) ومن ثم إجراء سيناريوهات اختبار قريبة من الواقع. على سبيل المثال: سيناريو تسجيل مستخدم جديد -> إضافة مورد -> إضافة منتج -> إنشاء طلب جديد -> تغيير حالة الطلب -> التحقق من تحديث المخزون. هذا يساعد على اكتشاف أي خلل في تتابع الخطوات عبر الوحدات المختلفة.
42. أشرك فريقًا (أو مختبِر مستقل) لإجراء **اختبار قبولي** (UAT) على الواجهة الأمامية: تأكد أن جميع الصفحات تعمل بشكل صحيح على مختلف المتصفحات والأجهزة، وأن تجربة المستخدم سلسة (بما في ذلك الرسائل وحالات الخطأ). أي ملاحظات يتم توثيقها وتصحيحها ضمن هذه المرحلة.

43. راجع **توثيق المشروع** (ملفات README والأدلة التشغيلية) وقم بتحديثها لتعكس الوضع الجديد. ينبغي أن تذكر التقنيات المستخدمة (تأكيد استخدام FastAPI مثلًا بدل Flask) [52]، وطريقة تشغيل البيئة (قد تتغير مع إضافة خدمات كRedis)، وطريقة استخدام الـAPI (تحديث قسم endpoints لإضافة ما تم استحداثه). كذلك راجع **دراسة المشروع المبدئية** وعدّل ما يلزمها (أو أضف ملحقًا) ليوضح أي تغييرات مبررة تمت على التصميم الأصلي أثناء التنفيذ. الهدف هو أن يكون هناك توافق تام بين ما هو موثق وما هو مطبق فعليًا.

44. **التحضير للإطلاق ونشر الإصدار الأول (MVP):** عند إكمال ما سبق، سيكون المشروع قد انتقل من مرحلة التأسيس إلى جاهزية إطلاق تجريبية. قم بالتالي لضمان إطلاق ناجح:

    - **نشر التطبيق في بيئة حقيقية**: أعد ضبط إعدادات Docker/docker-compose لتعمل في وضع الإنتاج (تفعيل وضع الإنتاج لـNext.js، وضبط عدد العمال workers في Gunicorn/ UVicorn لتشغيل FastAPI، الخ). انشر قاعدة البيانات على خادم موثوق أو خدمة سحابية (مثل RDS)، ونفس الشيء لRedis. اختبر سيناريوهات التحميل (مثلاً باستخدام أداة JMeter أو Locust لعمل ضغط اصطناعي على الـAPI) وتأكد من قدرة النظام على تحمل عدد مستخدمين معقول في المرحلة الأولى.
    - **مراقبة التشغيل**: فعّل خدمات مراقبة (Monitoring) وتنبيهات: استخدم أدوات مثل **Prometheus/ Grafana** أو خدمات سحابية لمراقبة أداء الخوادم (CPU, Memory) وأداء التطبيق (زمن الاستجابة للواجهات، معدل الأخطاء). أضف **نقاط قياس (Metrics)** داخل الكود لجمع معلومات هامة (عدد الطلبات المكتملة يوميًا، معدل التحويل...إلخ) [53] [54]. هذه المؤشرات تساعد في اتخاذ قرارات تحسين مستمرة كما أكدت الدراسة.
    - **دعم ما بعد الإطلاق**: جهّز خطة **لدعم المستخدمين والتعامل مع المشكلات الطارئة**. عين مسؤولًا لمراقبة الأخطاء (يمكن استخدام Sentry أو مشابه لجمع الأخطاء runtime من التطبيق

الأمامي والخلفي). تأكد من وجود **خطة لاسترجاع النظام** إن حدثت مشكلة كبيرة (Restore Plan) كما سبق وأعددت النسخ الاحتياطية [51] .

- **تجميع الملاحظات ووضع خطة تحسين تالية**: بعد الإطلاق التجريبي (Pilot) في سوق أو اثنين كما ورد (مثلاً ألمانيا وفرنسا كبداية [55])، اجمع ملاحظات المستخدمين الحقيقيين وأداء النظام الفعلي. بناءً على ذلك، رتب أولويات المرحلة التالية من التطوير. قد تتضمن تحسين بعض الميزات الحالية أو إضافة أخرى (مثل دعم مزايا تسويقية إضافية، أو تحسين خوارزميات الأسعار) [56] . احرص على إبقاء التطوير **حلقة مستمرة من التحسين** وفق منهجية Agile – خطط لإصدارات دورية (مثلاً كل شهر) تتضمن تحسينات أو ميزات جديدة، لضمان بقاء المنصة مواكبة للأهداف الإستراتيجية المذكورة في الدراسة الشاملة [57] .

ختامًا: باتباع هذه الخطة التفصيلية وتنفيذ المهام بالترتيب المذكور، سيتم إصلاح هيكلية المشروع ومعالجة نقاط عدم التوافق مع **دراسة المشروع الأصلية** ، وصولًا إلى بناء منصة متكاملة بمستوى **تحفة هندسية برمجية**. هذه الخارطة تعتمد مبدأ الخطوات الاحترافية المنهجية، بحيث يمكن استخدامها كمرجع لتنفيذ التحسينات والوصول بالمشروع إلى المعاييرمن最高 في الصناعة، محققين رؤية منصة **EuroDropship Pro** كما تصورتها الدراسة [58] [45] . كل مهمة مذكورة أعلاه تستند إلى أفضل ما توصلت إليه علوم هندسة البرمجيات حاليًا، وتقتدي بنهج الشركات التقنية الكبرى في بناء الأنظمة الموزعة الآمنة والقابلة للتوسع – مما يضمن أن يكون المشروع بعد الإصلاح على المسار الصحيح وبجاهزية عالية للنجاح في البيئات الحقيقية.

المصادر: تم استخراج المعلومات ومقارنة المتطلبات من وثائق المشروع (`docs/...`) ومن كود المستودع الحالي للتأكد من كل نقطة (مشار إليها بأرقام الأسطر أعلاه). يُرجى مراجعة هذه المصادر [2] [52] للتفاصيل الدقيقة لكل مقارنة مذكورة.

---

[1] [2] [4] [7] [8] [9] [11] [15] [16] [19] [28] [31] [32] [35] [39] [40] [41] [42] [43] [44] [45] [46] [49] [50] [51] [53] [54] [56]

[57] [58] tech_platform_architecture.md

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/docs/research Report/Sources/tech_platform_architecture.md_01

[3] [13] [14] [17] [18] main.py

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/backend/app main.py

[5] CHANGELOG.md

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/CHANGELOG.md

[6] [48] brand_identity.md

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/docs brand_identity.md

[10] models.py

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/backend/app models.py

[12] [52] README.md

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/README.md

[20] [21] [22] [23] suppliers.py

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/backend/app routers/suppliers.py

[24] [25] page.tsx

https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/frontend/app suppliers/page.tsx

test_api.py 37 27 26

/https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/backend tests/test_api.py

page.tsx 30 29

/https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/frontend/app products/page.tsx

page.tsx 34 33

/https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/frontend/app analytics/page.tsx

ci.yml 36

/https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/.github/workflows ci.yml

final_report_outline.md 38

/https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/docs final_report_outline.md

layout.tsx 47

/https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/apps/frontend/app layout.tsx

final_report.md 55

/https://github.com/abdulrhmanasami/shoobydo/blob/2603fcf07a543b84d074f857ab84a71e8bdc45f1/docs final_report.md