ChatGPT

مهام مشروع Shoobydo

فريق البرمجة الخلفية (Backend)

وفقًا للخطة الهندسية الموثّقة في التقرير التنفيذي [1] وهيكلية المستودع الحالية [2] ، تتركز المرحلة الأولى (الشهر 1–2) على إعادة الهيكلة وإنشاء الأساسيات. تشمل مهام الفريق:

- **إعادة هيكلة النظام إلى خدمات مصغرة (Microservices)** لكل وحدة أساسية (Users, Customers, Products, Orders). **تعتمد على:** تحديد التصميم المعماري النهائي لتقسيم الوظائف. (ملاحظة للمُنفّذ: يجب تنظيم المستودع الحالي وفقًا للخدمات المصغّرة والفصل الواضح للمسؤوليات) [1] .
- **إنشاء قاعدة بيانات PostgreSQL والجداول الأساسية** (Users, Customers, Products, Orders, Inventory) حسب النموذج المطلوب [1] . **تعتمد على:** تحديد قواعد البيانات ونموذج البيانات (الهيكلية) لكل خدمة.
- **تحديد نموذج بيانات المستخدم وحقول الصلاحيات** (roles) في جدول `users` : إضافة قيم الأدوار (`admin` , `manager` , `viewer`) في نوع بيانات `userrole` بقاعدة البيانات [3] . **تعتمد على:** وجود جدول `users` العامودي في قاعدة البيانات.
- **تطوير واجهات برمجة تطبيقات (API) أساسية CRUD للوحدات** التالية حسب التصميم المذكور في README [4] [5] :
- **وحدة العملاء (Customers):** مسارات `/customers/` (GET, POST)، `/customers/{id}` (GET, PUT, DELETE) [4] .
- **وحدة المنتجات (Products):** مسارات `/products/` (GET, POST)، `/products/{id}` (GET, PUT, DELETE) [5] .
- **وحدة الطلبات (Orders):** مسارات `/orders/` (GET, POST)، `/orders/{id}` (GET, PUT, DELETE) [6] .
- **وحدة الموردين (Suppliers):** تحسين مسارات CRUD الحالية (POST, PUT, DELETE لـ `/suppliers`). **تعتمد على:** إنشاء الجداول المناسبة في قاعدة البيانات. (ملاحظة: يجب الالتزام بالمسارات الموثّقة في README واختبارها بدورة حياة CRUD كاملة) [4] [5] .
- **تنفيذ تسجيل الدخول وإدارة الجلسات:** إعداد مسار `/auth/login` (وإن لزم `/auth/register`) للتحقق من بيانات اعتماد المستخدم وإصدار JWT [2] . **تعتمد على:** وجود جدول Users بأعمدة `email` و `hashed_password` (انظر [14]).
- **تفعيل نظام الترخيص (RBAC):** حماية المسارات الحساسة باستخدام الأدوار (على سبيل المثال، فقط `admin` و `manager` يمكنهما إنشاء/تعديل المستخدمين والطلبات) [4] [7] . **تعتمد على:** اكتمال إضافة أدوار المستخدم في قاعدة البيانات.

المرحلة 2 (الشهر 3–4)

تتركز هذه المرحلة على الأمن والأداء [8] :

- **تنفيذ نظام المصادقة JWT وإضافة المصادقة متعددة العوامل (2FA)** [8] . إنشاء مسارات دعم تغيير كلمة المرور وإعادة تعيينها إن وجد. **تعتمد على:** إعداد مفاتيح JWT وحماية التخزين الآمن لكلمات المرور.
- **تفعيل خدمة Redis للتخزين المؤقت (Cache)** وربطها بالـ Backend [8] . **تعتمد على:** تشغيل خادم Redis في البيئة (Docker) وضبط إعدادات الاتصال في التطبيق.
- **تحسين استعلامات قاعدة البيانات:** إضافة فهارس (Indexes) للجداول الأساسية حسب الحاجة [8] . **تعتمد على:** مراجعة الاستعلامات الشائعة والتأكد من تطبيق الفهارس في الترحيلات (Alembic).

- **استكمال واجهات API للوحدات المتبقية:** إن لم تنفذ بعد، بناء CRUD لـ **وحدة المستخدمين (Users)** (إنشاء/عرض/تعديل/حذف المستخدم) وربطها بمسارات المصادقة. **تعتمد على:** وجود جدول Users وجداول الصلاحيات.
- **وحدة إدارة المخزون (Inventory):** إنشاء جداول `stock_on_hand`, `stock_reserved`, `stock_movements` وإضافة مسارات API لضبط المخزون (`GET /inventory/products/{id}`) وحجزه/إطلاقه (`POST /adjust`, `/reserve`, `/release` وغيرها) [9]. **تعتمد على:** وجود جدول `products` الربط. (ملاحظة: على سبيل المثال، مسار `GET /inventory/products/{id}` للحصول على حالة المخزون `stock` [9])
- **توثيق نماذج البيانات والمسارات:** تحديث وثائق API (مثل OpenAPI/Swagger) لتعكس التغييرات الجديدة، وكتابة وصف للنماذج (Schemas) المستخدمة. **تعتمد على:** اكتمال تطوير الموديلات (Pydantic) في المشروع.
- **إنشاء اختبارات وحدة (Unit Tests)** لواجهات API الجديدة (Auth, Customers, Products, Orders) باستخدام إطار العمل (pytest) [10]. **تعتمد على:** إضافة نقطة دخول للاختبارات في CI.

المرحلة 3 (الشهر 5–6)

تتركز على **التكامل والخدمات الجديدة** [11]:

- **دمج بوابة الدفع Stripe:** إعداد مفاتيح API واختبار العمل مع الوضع التجريبي (Test). إنشاء مسار استجابة Webhook لمعالجة الدفعات (مثلاً تحويل حالة الطلب إلى `paid`) [11]. **تعتمد على:** إمكانية إنشاء سجل Order جديد ودمج حالة الدفع فيه.
- **دمج منصة التجارة الإلكترونية (Shopify/WooCommerce):** بناء خدمة استيراد/مزامنة للمنتجات والطلبات. على سبيل المثال، جلب المنتجات من المتجر الخارجي وإضافتها لجداول `products` في النظام، وتحديث المخزون عند وجود طلبات جديدة [11]. **تعتمد على:** واجهة Shopify APIs وإمكانية جدولة أو استقبال Webhooks للطلبات الجديدة.
- **تطوير وحدة التحليلات (Analytics/KPI):** بناء خدمة لحساب مؤشرات الأداء (مثل مجموع المبيعات اليومي/الشهري، عدد الطلبات المكتملة، العملاء الجدد) وعرضها من خلال نقاط نهاية API (مثلاً `/reports/kpis`) [11]. **تعتمد على:** وجود بيانات كافية في جداول Orders وCustomers، وربطها بمسارات API جديدة لعرض الإحصائيات.
- **تكامل Redis في استعلامات التحليل:** استخدام Redis لتخزين نتائج التحليلات المتكررة (الكاش) لتسريع استجابة الـ API. **تعتمد على:** تشغيل Redis مسبقًا في المرحلة السابقة.
- **إعادة تصميم الواجهة الأمامية (UI):** بالرغم من كونه مسؤولية فريق الواجهة الأمامية، يجب التنسيق مع الفريق المتخصص لإعادة تصميم واجهات المستخدم باستخدام Shadcn/UI وTailwind CSS [11]. **تعتمد على:** إكمال تصميم الهوية البصرية وتوفير التصميمات النهائية (Mockups). (ملاحظة: استخدام Shadcn/UI يستدعي تثبيت مكتبات React المطلوبة في فرونت إند.)

المرحلة 4 (الشهر 7–8)

مرحلة الإطلاق التجريبي والتثبيت [12]:

- **إطلاق بيئة تجريبية (Beta) للتطبيق:** نشر الخدمات في بيئة منفصلة وتفعيل ملاحظات المراقبة (Logging/Metrics) [12]. **تعتمد على:** الانتهاء من مهام CI/CD ونشر الحاويات بنجاح.
- **مراقبة الأداء والمعالجة:** جمع بيانات الأداء (استخدام CPU، استجابة DB) وإنشاء لوحات بيانات (مثلاً Grafana) لمراقبة النظام. **تعتمد على:** إعداد أدوات المراقبة كـPrometheus/ELK إن توفر.
- **إصلاح الأخطاء وتحسين الواجهة:** استكمال معالجة أخطاء الاستثناء (Bugs) التي تظهر في النسخة التجريبية، وتحسين تجربة المستخدم بناءً على التعليقات [12]. **تعتمد على:** جمع ملاحظات من الاختبار التجريبي (Beta feedback).

فريق البرمجة الأمامية (Frontend)

المرحلة 1 (الشهر 1–2)

- **بناء الصفحات الأساسية للوحة التحكم (Dashboard):** إنشاء صفحات `/`, `/dashboard`, `/analytics`, `/costs`, `suppliers` وربطها بالـ API الحالي [13]. **تعتمد على:** توفر نقاط النهاية البسيطة للبيانات (مثلاً قائمة الموردين والتكاليف).
- **إعداد هيكل واجهة المستخدم:** تثبيت Next.js وتهيئة البيئة بناءً على سياسة التشغيل السريع (`tools/start_frontend.sh`)، وضمان توافق الإصدارات مع `.env` [14]. **تعتمد على:** اكتمال إعداد خلفية العمل (Node/NPM, اتصال بالـ API).
- **تطبيق أسلوب الـ CSS الحالي:** العمل على تبسيط التنسيق بدون Tailwind (حسب الملاحظات في README) [15]. **تعتمد على:** تصميم أولي للهوية المرئية.

المرحلة 2 (الشهر 3–4)

- **تطوير واجهة تسجيل الدخول والتسجيل:** بناء صفحات الـLogin والـRegister وربطهما بخدمة المصادقة في الـBackend (JWT) [2]. **تعتمد على:** إطلاق مسارات Auth في الواجهة الخلفية. (ملاحظة: يجب التحقق من تخزين الـJWT في الكوكيز أو localStorage بأمان).
- **ضبط نموذج إدارة المستخدم:** إضافة واجهات لتعديل بيانات المستخدم وتغيير الصلاحيات (حسب دور الحساب). **تعتمد على:** إكمال مسارات Authg Users في الـBackend.
- **اختبار تفاعل الواجهة مع المصادقة:** التأكد من أن الواجهة تستخدم الـJWT المصدّق بشكل صحيح لحماية الصفحات (مثلاً، إعادة التوجيه إلى Login إذا لم يكن المستخدم مسجل دخول). **تعتمد على:** تجربة تسجيل الدخول والتأكد من تخزين الـToken.

المرحلة 3 (الشهر 5–6)

- **إعادة تصميم الواجهة باستخدام Tailwind CSSg Shadcn/UI:** إعادة بناء واجهات المستخدم لتصبح أكثر تفاعلية وجاذبية [11]. **تعتمد على:** تسليم مكونات التصميم من مصمم UX/UI والهوية البصرية الجديدة. (ملاحظة: يجب تحديث إعدادات Tailwind ومراجعة أنماط التصميم لتحقيق تجانس بصري).
- **تنفيذ صفحة التحليلات (Analytics Dashboard):** عرض مؤشرات الأداء (KPI) البيانية التي يوفرها الـBackend (مثلاً رسوم بيانية للمبيعات) [11]. **تعتمد على:** استدعاء الـAPI الخاصة بالتحليلات وتحويل البيانات إلى رسوم بيانية.
- **دمج Stripe Checkout:** إضافة واجهة دفع (Checkout form) والتأكد من عملها مع خدمة Stripe في الـBackend. **تعتمد على:** استكمال الربط مع Stripe في المرحلة السابقة.
- **تحديث صفحات العملاء والمنتجات:** الاستفادة من مسارات API الجديدة لـCustomers وProductsg لعرض بيانات العملاء والمنتجات بشكل ديناميكي. **تعتمد على:** توفر بيانات صحيحة من الـAPI.

المرحلة 4 (الشهر 7–8)

- **تحسين تجربة المستخدم (UX):** بناءً على ملاحظات مرحلة البيتا، تعديل الواجهة لتبسيط الإجراءات (مثلاً تحسين التنقل والعرض). **تعتمد على:** نتائج اختبارات المستخدمين الأولية.
- **إصلاح مشاكل العرض (UI Bugs):** معالجة أي أخطاء في عرض المكونات عبر المتصفحات الشائعة وضمان استجابتها (Responsive) للهواتف. **تعتمد على:** نتائج الاختبارات والتقارير المكتشفة في البيتا.

فريق DevOps / البنية التحتية

المرحلة 1 (الشهر 1–2)

- **إعداد بيئة التطوير عبر Docker Compose:** تهيئة الحاويات (PostgreSQL, Redis, Backend, Frontend) وضبط المتغيرات البيئية (env) للتطوير [14] . **تعتمد على:** وثائق التشغيل السريع (`tools/dev_up.sh`) وملفات البيئة المحددة.
- **إنشاء مراحل CI/CD أولية:** بناء خط (Pipeline) يجري على GitHub Actions (أو أداة مماثلة) لبناء الحاويات وتشغيل الاختبارات التلقائية (Unit Tests) [10] . **تعتمد على:** إضافة سكربتات التشغيل (`run_tests.sh`) وملفات تكوين CI.
- **ضبط نسخ قواعد البيانات:** ضمان إنشاء قواعد البيانات تلقائيًا عند بدء الحاويات (باستخدام Alembic أو سكربت تلقائي). **تعتمد على:** تهيئة Alembic ومزامنات الهجرات (Migrations).

المرحلة 2 (الشهر 3–4)

- **إدماج Redis في البيئة:** تشغيل حاوية Redis وضبط الاتصال في إعدادات التطبيق. **تعتمد على:** إعدادات Docker Compose المحدثة.
- **تحسين تكوين قاعدة البيانات:** تنفيذ ترحيلات (Migrations) لإضافة الفهارس وتحسين قيود الجداول [8] . **تعتمد على:** إكمال هياكل الجداول الجديدة في الـBackend.
- **تأمين بيانات البيئة:** إدارة مفاتيح JWT وStripe وأسرار قاعدة البيانات باستخدام نظام إدارة الأسرار (Secrets) في المنصة أو ملفات `.env` مؤمنة. **تعتمد على:** استخراج المتغيرات الهامة من التطوير اليدوي وتركيزها.
- **مراقبة الاختبارات والأخطاء:** إعداد نظام تسجيل الأخطاء (Logging) ورصدها عبر أدوات (مثلاً ELK أو Sentry) أثناء التطوير. **تعتمد على:** دمج مكتبة تسجيل مثل Logging أو إطار عمل مراقبة.

المرحلة 3 (الشهر 5–6)

- **توسيع CI/CD للنشر البيئي:** بناء وإطلاق الحاويات في بيئة Staging (بيئة تجريبية حقيقية) وإجراء اختبارات Smoke آلية بعد كل نشر. **تعتمد على:** بنية الـDocker Compose المتكاملة وخطوط الأوامر المكتوبة.
- **مراقبة الإنتاجية (Monitoring):** نشر أدوات مراقبة الأداء والموارد (مثل Prometheus/Grafana أو CloudWatch) لمتابعة تطبيقات Backend وFront-end. **تعتمد على:** تخصيص قواعد التتبع وإعداد قواعد التنبيه (Alerts) في البداية.
- **تكامل التقارير اللوجستية:** التأكد من أن خدمات Logging تم ربطها بجودة التقارير (Logs/metrics) في وقت التشغيل التجريبي [12] . **تعتمد على:** توفر مسارات Logging endpoints في التطبيق.

المرحلة 4 (الشهر 7–8)

- **إطلاق النسخة التجريبية (Beta Deployment):** نشر الحاويات في بيئة خارجية (Production Beta) مع التأكد من صلاحية النطاق (Domain) والشهادات الأمنية (SSL). **تعتمد على:** اكتمال بنية البيئة وضبط الشبكات.
- **التصحيح النهائي (Hotfixes):** تطبيق أي تصحيحات عاجلة للبيئة الإنتاجية خلال التجارب الأولية. **تعتمد على:** تقارير الخطأ من الاختبار التجريبي.
- **توثيق خطوط CI/CD والكود:** تحديث ملفات الوثائق (`README` , Wiki) لشرح عملية النشر التلقائي وكيفية تشغيل السكربتات. **تعتمد على:** الانتهاء من إعداد الـCI/CD وسيناريوهات الاختبار.

فريق التكامل (Integrations)

المرحلة 3 (الشهر 5–6)

- **تنفيذ ربط الدفع بـStripe:** كما في مهمة DevOps/Backend، إعداد الـAPI Key وتلقي Webhooks على `/payments/stripe`. **تعتمد على:** وجود نُقطة نهاية في الـBackend للتعامل مع استجابة الدفع.
- **تنفيذ ربط منصة Shopify (أو WooCommerce):** بناء خدمة تقوم باستيراد المنتجات والطلبات من المنصة الخارجية دورياً (أو عبر Webhooks) وتحديث النظام الداخلي. **تعتمد على:** معرفة واجهات برمجة التطبيقات (APIs) الخاصة بالمنصة الخارجية وخدمات قائمة Stock/Sales. (ملاحظة: يجب تنسيق جداول `products`، `orders`، `inventory` مع البيانات الجديدة.)
- **اختبار ربط الأنظمة الخارجية:** إجراء اختبارات شاملة (End-to-End) للتأكد من سير عمليات الدفع والاستيراد بشكل صحيح (مثلاً إنشاء طلب تجريبي واختبار خصم الرصيد). **تعتمد على:** اكتمال الإعدادات في Backend وتوافر بيئات اختبار Stripe/Shopify.

المرحلة 4 (الشهر 7–8)

- **ضبط وتوثيق آليات التكامل:** ضمان وجود سجلات تدقيق (audit logs) لعمليات التكامل، وتوثيق كيفية إعداد المفاتيح (API Keys) في بيئة الإنتاج. **تعتمد على:** إتمام ربط Stripe/Shopify وتحديد المتطلبات الأمنية.
- **تحسين استقرار التكامل:** معالجة أي أعطال تظهر أثناء التعامل مع المنصات الخارجية (مثل إعادة المحاولة عند فشل Webhook). **تعتمد على:** تقييم أداء الوصلات وحالة الشبكة أثناء الاختبار التجريبي.

فريق الاختبار (Testing)

المرحلة 1 (الشهر 1–2)

- **إعداد إطار اختبار موحد:** استخدام pytest (أو إطار مشابه) لكتابة اختبارات الوحدات (Unit Tests) للـBackend [10]. **تعتمد على:** تهيئة بيئة الاختبار (تثبيت قواعد بيانات وهمية أو استخدام SQLite للاختبارات).
- **اختبارات CRUD الأساسية:** كتابة اختبارات وحدات للعمليات الأساسية (إنشاء/قراءة/تحديث/حذف) على وحدات `suppliers` الموجودة ووحدات `customers` و`products` و`orders` عند إتاحتها. **تعتمد على:** المسارات المنفذة حديثًا في الـBackend.
- **مراجعة التغطية (Coverage):** إعداد تقرير تغطية اختبار تلقائي للتأكد من تغطية الأكواد المهمة. **تعتمد على:** تشغيل اختبارات الوحدة ضمن الـCI وتقرير التغطية الناتج.

المرحلة 2 (الشهر 3–4)

- **اختبارات النظام (Integration Tests) لمصادقة المستخدم:** التأكد من عمل نظام المصادقة بشكل كامل (تسجيل دخول/خروج، حماية مسارات API حسب الدور) باستخدام اختبارات متكاملة. **تعتمد على:** تفعيل JWT وRBAC في الـBackend.
- **اختبارات الأداء (Performance Tests):** قياس زمن استجابة API قبل وبعد إضافة Redis/فهارس قاعدة البيانات لتقييم التحسينات. **تعتمد على:** توفر بيئة اختبار قابلة لإجراء اختبارات تحميل محدودة.
- **اختبارات الجلسة (Smoke Tests):** إعداد سيناريوهات اختبار سريعة (Smoke) للتأكد من أن الخدمات الرئيسية تعمل بعد كل تحديث. **تعتمد على:** نشر بنية بيئات الاختبار (Docker) في الـCI.

المرحلة 3 (الشهر 5–6)

- **اختبارات نهاية إلى نهاية (E2E) لعمليات الدفع:** محاكاة عملية شراء كاملة عبر Stripe والتأكد من تحديث النظام الداخلي بالحالة الصحيحة (مثلاً تغيير حالة الطلب إلى مدفوع). **تعتمد على:** ربط Endpoints مع Stripe في الـBackend والواجهة الأمامية.

- **اختبارات التكامل مع Shopify:** التأكد من استيراد المنتجات والطلبات بشكل صحيح من المتجر الخارجي. **تعتمد على:** وجود بيانات اختبار في متجر Shopify وتهيئة مفاتيح الوصول.
- **اختبارات واجهة المستخدم (UI Tests):** باستخدام أداة مثل Cypress أو Selenium، كتابة سيناريوهات للتأكد من أن الواجهة الأمامية تعمل بشكل سليم (التنقل بين الصفحات، إجراء العمليات). **تعتمد على:** إكمال تطوير واجهة المستخدم الأولى.

المرحلة 4 (الشهر 7–8)

- **اختبار الإصدار التجريبي (Beta Testing):** إجراء اختبارات شاملة (ضمن فريق داخلي) على النسخة الكاملة من النظام قبل الإطلاق النهائي. **تعتمد على:** اكتمال جميع الوظائف المطلوبة في المراحل السابقة.
- **اختبارات استقرار (Regression Tests):** التأكد من أن التعديلات الأخيرة لم تكسر وظائف سابقة. **تعتمد على:** جميع حالات الاختبار المكتوبة سابقًا والمشغلة في الـCI.
- **توثيق نتائج الاختبارات:** تجميع نتائج الاختبارات وكتابة تقارير الأخطاء وتقديمها للفرق المعنية لإصلاحها. **تعتمد على:** اكتمال إجراء الاختبارات وتوثيقها.

ملاحظة عامة: يجب توثيق جميع التغيرات (في README أو Wiki المشروع)، وضمان نشر ملفات التهيئة (مثل Docker Compose وCI config) وتنظيمها بشكل واضح. كما ينبغي تحسين الأداء باستمرار (مثل استخدام الكاش Redis وفهارس قاعدة البيانات) وتضمين هذه المهام ضمن خطة العمل في كل مرحلة [8] [14] . كافة المهام أعلاه محددة وقابلة للتنفيذ مع ذكر التبعيات اللازمة قبل الشروع بها.

المصادر: التقرير التنفيذي لمشروع Shoobydo [1] [8] [11] [12] وملفات المشروع (README وملفات المسارات) [2] [4] [13] [16] .

---

[1] [8] [11] [12] تقرير تنفيذي حول مشروع Shoobydo.pdf
file://file-RzRRcFfqtPT8nEc8mzpUFm

[2] main.py
https://github.com/abdulrhmanasami/shoobydo/blob/d1758febb20a3b5432b309aaa6179430e3db8bf3/app/main.py

[3] a1b2c3d4e5f6_create_users_table.py
/https://github.com/abdulrhmanasami/shoobydo/blob/d1758febb20a3b5432b309aaa6179430e3db8bf3/apps/backend
alembic/versions/a1b2c3d4e5f6_create_users_table.py

[4] [5] [6] [7] [9] [10] [13] [14] [15] [16] README.md
https://github.com/abdulrhmanasami/shoobydo/blob/d1758febb20a3b5432b309aaa6179430e3db8bf3/README.md