HERBVERSE HACK – MARKETMITRA & STOCKLY

Project Documentation

Team: **Harshit Dhundale**

Date: May 10, 2025

1. Overview

MarketMitra (customer app) and Stockly (vendor app) form a cross-platform marketplace ecosystem for natural products. Both apps were originally built as responsive React web applications and then packaged into true Android APKs via a React Native–Expo WebView wrapper. A single Node.js/Express backend with MongoDB and Redis powers all functionality and data.

2. App Architecture

The project is structured as a unified platform with two distinct mobile applications – MarketMitra (customer-facing) and Stockly (vendor-facing) – backed by a single shared server. Both front-end apps were originally built as responsive React web applications and then intentionally packaged into Android mobile apps using a React Native wrapper strategy. Specifically, each mobile app is an Expo-managed React Native application containing a WebView that loads the respective React web app. This approach allows 100% reuse of the web code for mobile, ensuring the Android APKs deliver the exact same features and interface as the web versions. It was a deliberate technical choice to meet hackathon guidelines (providing installable APKs) while maximizing development speed and consistency across platforms.

On the back-end, a Node.js Express server powers all application logic through a set of RESTful API endpoints. The mobile (and web) front-ends communicate with this backend over HTTPS, exchanging JSON data for all operations (such as fetching product lists, submitting orders, etc.). Data is persisted in a MongoDB database, which stores collections for users, products, orders, and other entities. Redis is incorporated as a high-speed in-memory store to cache frequently accessed data and manage transient session information – for example, temporary OTP codes during login verification or caching product catalog data to improve performance. The tech stack also includes Razorpay integration for payments: during checkout, the front-end calls a secure payment API on the backend which in turn uses Razorpay's Node.js SDK to initiate and verify transactions. This ensures sensitive payment processing is done server-side, with the front-end handling just the necessary user interactions. Overall, the architecture cleanly separates concerns: the React/React Native front-ends handle presentation and user experience, while the Express/Node backend handles data management, business logic, and third-party integrations. This high-level design results in two apps that function seamlessly with a single source of truth on the server, simplifying maintenance and delivering a consistent experience to both customers and vendors.

3. Key Features

Both MarketMitra (Customer App) and Stockly (Vendor App) are packed with features that cover the end-to-end needs of a two-sided marketplace platform. The major features include:

OTP-based Authentication: Both apps use secure one-time password (OTP) verification for user login and signup. Users enter their mobile number and receive an OTP via SMS, which the app verifies through the backend. This streamlines onboarding for customers and vendors by removing the need for remembering passwords while ensuring account security.

Product Browsing & Search (Customer App): MarketMitra allows customers to easily browse products offered by various vendors. Products are organized into categories with options to search by name or filter by category. Each product has a detailed page showing images, descriptions, pricing, and availability. The interface is designed for quick discovery, letting users scroll through listings and tap to view details within the app's smooth WebView-powered UI.

Shopping Cart and Order Management (Customer App): Customers can add products to a virtual shopping cart as they browse. The cart view lets them review selected items, adjust quantities, and see the total price. When ready, users proceed to place an order. Upon checkout, the app collects a delivery address (or uses a saved address) and then initiates payment. After an order is placed, MarketMitra provides an order confirmation and allows the customer to track the order status. Users can also view their order history, with details of past orders and their statuses, all fetched securely from the backend.

Payment Integration with Razorpay: A seamless payment experience is integrated into the customer app via Razorpay. When a customer checks out, the app (through the WebView) invokes the Razorpay payment gateway – supporting credit/debit cards, UPI, netbanking, etc. – to complete the transaction. The integration is handled through the backend for security: the server creates orders and verification using Razorpay's APIs, while the customer sees a smooth payment pop-up and confirmation. This ensures that transactions are secure and fast, with the status (success/failure) communicated back to the app and reflected in the order details.

Real-Time Order Status & Delivery Tracking (Customer App): After payment, customers can track their order progress in real time. The order status (e.g., Confirmed, Packed, Dispatched, Out for Delivery, Delivered) is updated by the vendor via the Stockly app and reflected to the customer in MarketMitra. The customer app periodically fetches status updates from the backend (or listens for updates) so that users always see the latest information. While not using live map tracking, the app provides clear step-by-step updates on the delivery progress. Push notifications or in-app alerts can also notify the user of status changes (if implemented).

Vendor Inventory Management (Vendor App): Stockly is equipped with robust inventory control features for sellers. Vendors can create new product listings by entering product details, uploading images, setting prices, and available stock quantities. They can edit existing product info or deactivate items that are out of stock. The inventory list is presented in a dashboard format, showing all products at a glance with their current stock status. Any changes a vendor makes (like updating stock count or price) are immediately saved to MongoDB via the backend and would be reflected on the customer app product listings. This real-time inventory management ensures that customers only see available products and that vendors have full control over their catalog.

Order Management for Vendors (Vendor App): The Stockly app allows vendors to handle incoming orders efficiently. When a customer places an order, it appears in the vendor's order list with details such as items purchased, quantities, customer name, and delivery address. Vendors can view each order, then update its status as they process it – for example, marking an order as “Accepted,” then “Dispatched,” and finally “Delivered” once fulfilled. These actions not only help the vendor organize their workflow but also automatically send updates to the customer’s app via the backend. The order management feature ensures vendors can keep track of all open orders and maintain communication with customers through status updates.

Analytics Dashboard (Vendor App): To help vendors make informed decisions, Stockly provides a basic analytics section. Here, sellers can see key metrics like total sales, number of orders over time (daily/weekly/monthly), revenue earned, and possibly best-selling products. This data is presented in a concise dashboard format (charts or summary cards) within the app. By analyzing these trends, small business owners get insights into their performance directly from the app. The analytics feature underscores the platform’s aim to not just facilitate sales but also empower vendors with information to grow their business.

Community Forums & Support: The platform includes a forum feature to foster community engagement. Accessible from the apps, the forum allows users to post questions, share knowledge, and discuss topics relevant to the marketplace. For example, vendors might exchange tips on improving sales or managing inventory, and customers might discuss product recommendations or seek support. The forum is implemented as a discussion board style interface within the app’s WebView, backed by the same Node/Express backend (with MongoDB storing posts and comments). This creates an interactive community space beyond transactions. In addition, the presence of forums and possibly an FAQ or support section helps users get help and stay engaged with the application beyond just buying or selling.

(In summary, both apps cover the full spectrum of features needed for a marketplace: from onboarding with OTP, product discovery and purchase, secure payments, delivery updates, to vendor-side listing management, order fulfillment, and even analytics and community interaction.)

4. Code / Project Structure

The codebase is organized in a mono-repo style, containing three main packages: one for the backend and one each for the two frontends. This means all the code resides in a single repository with clearly separated folders, for example: a /backend directory for the server, /marketmitra-frontend for the customer web app, and /stockly-frontend for the vendor web app. Each of these is structured logically and follows best practices for maintainability:

Backend (Node.js/Express): The backend folder contains an Express application divided into layers. There are separate sub-folders for routes, controllers, models, and services. For instance, under routes you might find files like auth.js, product.js, order.js, etc., each defining the API endpoints for that domain (e.g., /api/auth/login, /api/products/:id). Controllers encapsulate the business logic for each request – for example, an order controller will handle creating a new order, updating order status, etc., interacting with database models as needed. Models are defined (likely using Mongoose for MongoDB) for each major entity: User, Product, Order, etc., enforcing a schema and providing an interface to query and update the MongoDB collections. Services or utilities are used for auxiliary tasks like sending OTP SMS, processing payments via Razorpay, or caching with Redis. By following a modular MVC (Model-View-Controller) pattern, the backend code remains clean and easy to navigate. Each piece of functionality (authentication, payments, forum posts, etc.) is implemented in its own module, which improves readability and facilitates team collaboration. Configuration for the server (such as database connection strings, API keys for Razorpay, SMS gateway credentials for OTP, etc.) is managed with environment variables and organized in a config file, keeping sensitive information secure and separate from logic. Overall, the backend adheres to best practices such as error handling middleware for Express, input validation (to ensure, for example, an order request has all required fields), and using Redis not only for caching but also to store session tokens or OTP verification data with expiration for security.

Front-end (React Web Apps for Customer & Vendor): Each front-end application (MarketMitra and Stockly) has its own dedicated project within the monorepo, but they share a similar structure since both are React-based. The code is written in JavaScript (or TypeScript if used) and leverages common libraries like React Router (for navigating between pages/screens), Redux (for state management), and Axios/Fetch for API calls to the backend. In each front-end project, the source (/src) directory is organized into folders like components, pages (or screens), redux (store), services, and assets. Reusable UI components (buttons, product cards, navigation headers, etc.) are placed in the components folder to keep the code DRY (Don't Repeat Yourself), since many components (like product listing cards or forms) are used in multiple places. The pages (or screens) folder contains higher-level components that correspond to entire screens or views (for example, a HomePage, ProductDetailsPage, CartPage for MarketMitra, or an InventoryPage, OrdersPage for Stockly). These pages tie together various components and connect with the Redux store or backend services as needed.

State management in both apps is handled with Redux for predictable state updates and global availability of key data (this was set up to maintain clean separation of logic and UI). For instance, there are Redux slices or reducers for authentication (holding the current user/vendor info and login state), products (storing product lists fetched from the server), cart (for the customer app, tracking items the user added), and orders (tracking order status and history). Actions are defined to fetch data from the backend and update the state; e.g., an action to load all products dispatches an API call to the backend and then on success, saves the product list in the Redux store. This approach ensures that different components can easily access shared data without prop-drilling, and it centralizes updates for consistency.

The services or api folder in each front-end contains helper modules for making HTTP requests to the backend. Rather than calling fetch or axios in components directly, these service functions (for example, api.login(phone, otp), api.fetchProducts(), or api.updateOrderStatus(orderId, status)) encapsulate the request details (URLs, request bodies, headers, etc.). This not only makes the components simpler and focused on UI, but also makes it easy to update API endpoints in one place if they change.

Both front-ends follow best practices in React development: they are written as functional components with hooks (e.g., using useState, useEffect for lifecycle), they avoid unnecessary re-renders by structuring state appropriately, and they ensure UI and logic are separated. The code is formatted and documented for readability. Additionally, because the two apps reside in one repo, they can share certain constants or utility functions (for example, perhaps some form validation logic or a date formatting helper) by having a shared utilities file or just by duplicating minimal code where necessary. This monorepo setup made it easier to keep the front-end projects consistent (for instance, using the same version of React and libraries, and similar linting rules), and also to coordinate changes between the front-ends and backend during development.

Finally, when it came to building the mobile apps, each React project was bundled with an Expo React Native wrapper. Essentially, the web build of each app (which is a collection of HTML/CSS/JS) is loaded into a WebView component in a simple React Native app container. The repository might include an /android directory or Expo configuration for building the APKs. This doesn't affect the structure of the web code itself but is part of the deployment packaging. All in all, the project structure is cleanly separated yet centrally managed, making the hackathon submission easier to develop, test, and deploy.

5. Mobile Flows (Key Screens & Usage)

Both MarketMitra and Stockly provide a comprehensive set of screens and user flows, all of which are delivered through the mobile apps with identical functionality to their web counterparts. Thanks to the WebView-based React Native approach, the navigation and UI in the mobile apps mirror the web experience exactly, ensuring users on mobile can do everything they could on the web. Below is an overview of the key screens and flows in each app and how users interact with them on mobile:

Onboarding and Login/Registration (Both Apps): When a user opens the app for the first time, they are greeted with a simple onboarding or welcome screen. New users (whether customers or vendors) are prompted to register by providing their phone number. Upon entering their number, the app triggers the OTP authentication flow – a 6-digit one-time passcode is sent via SMS. The user then enters this code into the app to verify their account. For returning users, the process is similar: they input their phone number and verify via OTP (essentially passwordless login). Once verified, the backend creates or retrieves the user's profile (for customers) or vendor account details (for sellers) and the user proceeds into the main app. This flow is straightforward on mobile; the screens for phone input and OTP entry are part of the React web app's UI rendered in the WebView, so they look and feel native. Error messages (like incorrect OTP) and loading indicators are in place to guide the user. After successful login, the user's session is established (e.g., a token saved) so they remain logged in on the device.

Customer App - Browsing Products and Searching: In MarketMitra, the first main screen after login is typically a home or catalog screen showing featured products or categories. The mobile app displays this in an infinite scroll or grid layout optimized for small screens. Customers can navigate through categories or use a search bar to find specific products. When they tap on a product listing, the app navigates (via React Router, within the WebView) to the Product Details screen. This screen shows a larger product image, detailed description, price, seller info, and an option to add to cart. All this happens seamlessly within the mobile app container, so to the user it feels like a native e-commerce app. They can easily go back to the list or search results as needed. The consistency of the design (carried over from the responsive web UI) means the app retains a clean and intuitive interface on mobile.

Customer App - Cart and Checkout Flow: When a customer decides to purchase items, they tap "Add to Cart" on product pages. This updates a cart icon or section in the app (using Redux state under the hood). The user can view their Shopping Cart screen at any time, which lists all selected items, quantities, and the subtotal. On mobile, this is presented in a scrollable list with options to change quantity or remove items. From the cart, proceeding to Checkout prompts the user to enter or confirm their delivery address (there may be an address form or selection if multiple addresses are supported) and then to initiate payment. When the user confirms the order, the app opens the Razorpay payment interface (which might appear as a popup or a new screen overlay within the WebView). The user can then enter their payment details securely. After payment is completed (or canceled), the app returns to an Order Confirmation screen. If successful, a thank-you message and order summary are shown, and the order is now recorded in the system. Throughout this flow, the mobile app ensures all necessary validations (like ensuring cart isn't empty, address is provided, payment result is received) are handled, providing a smooth transaction experience. Any errors (payment failure, network issues) are caught and displayed to the user with guidance to retry if needed.

Customer App - Order Tracking: Once an order is placed, the user can navigate to an Orders or My Orders section in MarketMitra. This section shows a list of all orders the customer has made, with the latest one typically at the top. Each order entry displays key info like the order number, date, total amount, and current status (e.g., "Pending Confirmation", "Dispatched", etc.). The user can tap an order to view the Order Details screen, which itemizes the products in that order and shows the timeline of status updates. For example, an order timeline might show: Ordered (time stamp), Accepted by Seller (time stamp), Out for Delivery (time), Delivered (time). As the vendor updates the order in Stockly, these statuses change accordingly. On mobile, the app might use visual indicators like checkmarks or progress bars to illustrate the stage of the order. This way, customers are never in the dark about their purchase. The use of the WebView doesn't limit this functionality – it refreshes or dynamically updates the content as the status changes (the app could pull the latest status each time the user opens the order, or use web socket/polling via the backend if implemented). This flow greatly enhances trust and transparency, as a customer can open the app at any time to see exactly where their delivery stands.

Vendor App - Dashboard Overview: After a vendor log into Stockly, they are taken to a Dashboard or home screen tailored for the seller's needs. This screen provides a quick overview of their business at a glance. For example, it might show the number of new orders awaiting action, a summary of total sales today, low-stock alerts for any products, and maybe shortcuts to common tasks (like "Add Product" or "View Orders"). On the mobile app, this dashboard is presented in a clear, card-based layout optimized for small screens. The vendor can tap on any section (e.g., tap the orders card to go to the Orders list, or a stock alert to go to the inventory page) to dive deeper. The dashboard essentially helps the vendor quickly assess what needs attention each day. Because it's part of the web code, it can easily be updated with new widgets or information, and it remains consistent between the web portal and the mobile app.

Vendor App - Inventory Management Screens: One of the core flows in Stockly is managing the product catalog. The Inventory screen lists all products the vendor has added, each with details like name, price, and current stock count. On mobile, this list might be scrollable with each product in a collapsible panel or separate row, possibly color-coded if stock is low or out. The vendor can tap a product to edit it, which opens an Edit Product form where fields like name, description, price, and stock quantity can be updated, and new photos can be uploaded. There is also an Add New Product flow, reachable via an "Add Product" button. This brings up a blank form where the vendor enters all required info and submits it. The app (through the WebView form) will call the backend to save the product in the database. Validations (like required fields, numeric fields for price and stock) are handled to ensure data integrity. If the platform supports product categories or tags, those can be selected here as well. After adding or editing, the vendor is taken back to the inventory list where the new or updated item appears. All these screens are designed to be mobile-friendly, with large input fields and buttons, since they were built as responsive web pages. The flow allows a vendor to maintain their storefront directly from their phone, anytime.

Vendor App - Order Processing Flow: Equally important for the vendor is the Orders section in Stockly. This shows a list of customer orders that have been placed with that vendor. Each order entry shows an order ID, the date, and its current status (e.g., "New" for newly placed orders, "Dispatched", etc.). A vendor can tap an order to open the Order Detail screen. Here, they see the full contents of the order (items and quantities), the customer's name and delivery address, and any notes. The vendor then has controls to update the order status. For instance, initially an order might need to be "Accepted" or confirmed. Once the vendor is ready to ship, they can mark it as "Dispatched". Finally, after delivery (perhaps confirmed by their delivery person or the customer), they mark it "Delivered". These actions are usually buttons or dropdowns on the order detail screen. Tapping them will trigger an update via the backend (which also updates the customer's view of the order). There might be a confirmation prompt to avoid accidental changes. On mobile, this workflow is very convenient – a vendor can handle orders on the go, simply by using their phone to check new orders and update statuses as they prepare and send out items. The UI provides feedback for each action (like "Order marked as Dispatched" toast message) so the vendor knows the update was successful. This end-to-end order fulfillment flow within the app means vendors don't need any other system to manage their online orders.

Vendor App - Analytics & Forum Access: In Stockly, after dealing with inventory and orders, a vendor might visit the Analytics section to review their business metrics. On the mobile app, this might be a separate tab or accessible from the side menu. The Analytics screen can include charts or lists showing sales over time, comparisons of weekly or monthly performance, and product-specific stats (like top-selling items). For example, a bar chart could show revenue each day for the past week, or a pie chart could show percentage contribution of each category to sales. Even though it's a hackathon project, this feature demonstrates foresight by giving vendors actionable insights right on their phone. Additionally, both apps include access to the Community Forums (if implemented as a common area). From the mobile app menu, a user (customer or vendor) can navigate to the forum page. This loads a discussion board where users can scroll through topics, read posts, and contribute by writing their own posts or replies. On a phone screen, the forum is formatted for readability – threads might be listed vertically, and tapping one shows the conversation. Users can type and submit responses easily. This kind of engagement flow keeps users within the app for community support and interaction. Finally, the mobile apps likely also have common utility screens such as Profile/Account Settings (where a user or vendor can update their name, picture, or other info, and log out), and maybe a Support/Help section linking to help articles or contact info for assistance. Each of these flows is delivered through the same web code via WebView, meaning the experience is consistent and fully functional.

In all of the above flows, it's important to note that the mobile apps are not missing any functionality compared to the web versions – they are in fact the exact same application interfaces, optimized for touch via responsive design and packaged in a mobile container. This ensures that a customer using MarketMitra or a vendor using Stockly on Android gets the full feature set: from account creation to purchasing or order fulfillment and beyond, every feature is accessible and works smoothly. The use of React Native with WebView was an intentional architectural decision to achieve this parity and speed of development. By doing so, the team delivered two robust, fully-featured mobile applications within the hackathon timeframe, adhering to all submission guidelines and providing a polished user experience on both web and mobile platforms.

6. Summary

A. Key Features

MarketMitra (Customer)

1. Secure Onboarding: Phone-OTP login/signup; password reset.
2. Product Discovery: Browse and filter by category, price, availability; search bar.
3. Shopping Cart & Checkout: Add to cart, enter address, pay via Razorpay, receive invoice by email.
4. Order Tracking: View order history; real-time status updates (‘‘Processing’’,“Confirmed,” “Dispatched,” “Delivered”); retry payment within 30 minutes for failed orders.
5. Community Forum: Post questions and replies in a discussion board.
6. Profile Management: View/edit personal details; logout.

Stockly (Vendor/Admin)

1. Secure Onboarding: Same OTP flow as customer app.
2. Dashboard Analytics: Total products, orders, revenue summary, low-stock alerts, popular items.
3. Inventory Management: CRUD operations on products; sort by type, price, stock; dead-stock handling.
4. Order Management: View incoming orders; update status through fulfillment stages; bulk delete.
5. Community Forum & Profile: Engage with other vendors; manage account settings.

B. Code / Project Structure

```
/root
  /backend
    /controllers    ← Business logic per entity (auth, product, order…)
    /models         ← Mongoose schemas: User, Product, Order, ForumPost…
    /routes         ← Express routers: /api/auth, /api/products, /api/orders…
    /services       ← Payments (Razorpay), email (Nodemailer), OTP (Redis)…
    /middleware     ← JWT auth, error handlers, rate limiting, input sanitizing
    server.js       ← Entry point
```

C. Mobile Flows (Screens)

1. Onboarding & Auth: Welcome → Phone input → OTP verification → Home/dashboard.

2. Product Browsing: Category list → Product listing → Product details → Add to cart.

3. Cart & Checkout: View cart → Edit quantities → Enter address → Razorpay payment → Confirmation & invoice email.

4. Order Tracking: Orders list → Order details → Status timeline updates.

5. Vendor Dashboard: Summary cards → Quick actions (Add Product, View Orders).

6. Inventory: List products → Add/Edit/Delete product form → Save to backend.

7. Order Processing: New orders list → Detail view → Update status → Notify customer.

8. Analytics & Forum: Charts and metrics → Community discussion board.

9. Profile: View/edit profile info → Logout.

D. Screenshots & Demo Links

MarketMitra Demo: https://marketmitra.vercel.app

Stockly Demo: https://stockly-mu.vercel.app

Apk Demos: https://github.com/Harshit-Dhundale/Hackathon/tree/main/apk

Source Code: https://github.com/Harshit-Dhundale/Hackathon

Video Demo: https://youtu.be/4yCc4pIWgqY

Screenshots: https://github.com/Harshit-Dhundale/Hackathon/tree/main/screenshots

---**End of Documentation**--