

Product Overview: UniThrift

Many students often overspend on different products like clothes, supplies, gadgets, etc. while simultaneously wasting unused products. UniThrift offers the solution for this in the form of an efficient marketplace for students to buy and sell products, new or second hand, within their school community, making transactions easier and safer. The technology stack used supports quick MVP development, establishing a strong foundation for future scale and enhancements.

Frontend: HTML, CSS, JavaScript

Role: Determines what the user can see while handling product browsing, search, cart management, and overall user interaction (UI and UX). It interfaces directly with REST API to seamlessly update dynamic content.

Justification: It is widely used when it comes to web applications as it provides a foundation for early development while being relatively lightweight, creating a more efficient work environment. As the project grows into a more complete product, the foundation set by HTML, CSS, and JS can easily be integrated with more advanced/complex tools.

Backend Framework: Node.js with Express.js

Role: Manages business logic, API routing, data processing, communication with the database.

Justification: Node.js and Express.js as a framework enables JavaScript to be used more consistently and effectively across the entire product, making the development process much easier for us. Since we created a marketplace application, frequent browsing and communication between buyers and sellers is something this backend framework can excel at due to the non-blocking, event-driven architecture for handling multiple user requests.

API Communication: REST API with JSON

Role: Communication bridge between frontend UI actions and backend logic/data. The REST API sends and receives JSON data.

Justification: REST provides a predictable communication pattern and supports clear separation between interface and logic. This basically means that it reduces integration complexity and allows us to work in parallel with each other. Furthermore, JSON provides lightweight data transfers that are easy to debug. Lastly, REST APIs are easily integrated to other platforms.

Middleware: CORS, express.json()

Role: Ensures smooth request handling. Express.json() parses JSON data sent through requests like login and listing creation while CORS allows the frontend test server to communicate with the backend securely.

Justification: Helps maintain smooth and secure communication between the frontend and the backend. Express.json() ensures that any user data sent to the server is properly understood, while CORS protects the system from unauthorized access. These applications improve reliability and can easily support future upgrades.

Server: Python3 HTTP server

Role: Hosts frontend locally without backend dependency allowing easier UI development before the server was complete.

Justification: No additional configurations or installations were required. It supports early interface development even if backend features are still incomplete, enforcing a realistic testing environment with correct file handling and simulated API calls.

Database / Mock Storage: Prisma with SQLite

Role: Stores essential marketplace data such as users, listings, and item information. Prisma acts as the database manager while SQLite acts as the storage during development.

Justification: Prisma allows us to manage data efficiently while keeping the setup simple for development. SQLite is lightweight and requires no complex server installation, ideal for an MVP.

Data Flow Description

When a user performs an action like viewing listings or managing their cart, the frontend sends a request to the backend through the REST API. This request is handled by Node.js and the Express.js server, where middleware like express.json() and CORS process the data and ensure secure communication. After that, the backend retrieves or updates information stored in the SQLite database through Prisma. Once the operation is complete, the backend sends the right data back to the frontend in JSON format. The frontend then updates what is shown on the screen, allowing users to continue navigating the marketplace smoothly and efficiently.

Core Transaction Flow

The core transaction flow of UniThrift captures the full lifecycle of a buyer-seller interaction within the marketplace, from initial product discovery to order fulfillment. It is designed to ensure smooth, secure, and traceable transactions entirely within the university community.

- 1. Listing Discovery and Offer Creation:** The process begins when a buyer browses available listings on the frontend interface. Using the REST API, the frontend requests product data from the backend, which retrieves the relevant listing information from the SQLite database via Prisma. Once the buyer finds an item of interest, they can send an offer for that specific product. This action triggers a POST request to the backend API containing the offer details (e.g., buyer/product ID, and proposed price). The backend validates the data through Express.js middleware (express.json() for parsing and CORS for security) before recording the offer in the database.
- 2. Seller Response and Transaction Initiation:** The seller, upon receiving a notification of the offer, can view it through their own user interface. When the seller accepts an offer, the backend updates the listing status and generates a corresponding order record through Prisma. This marks the formal start of a transaction between the buyer and the seller. The updated order data is then returned to the frontend in JSON format to update both parties' dashboards in real time.
- 3. Payment Processing:** Once an order is created, the buyer is redirected to a payment gateway interface integrated into the frontend. This step uses REST API calls to pass transaction details securely to the backend, where payment verification logic (simulated for the MVP stage) confirms whether the payment was successful. Upon successful confirmation, the backend updates the order's payment status in the SQLite database through Prisma and sends the confirmation back to the frontend for user display.
- 4. Fulfillment and Completion:** Following payment confirmation, the order is marked as ready for fulfillment. The buyer and seller then coordinate a meetup or pickup within their campus.. After the exchange, the seller marks the order as completed via the frontend, which triggers another backend API call to update the order's status to fulfilled. The backend stores this final state in the database and returns a JSON confirmation, completing the transaction flow.
- 5. Feedback and System Update (Post-Transaction):** After fulfillment, both parties may optionally leave feedback or ratings (if implemented later). The database updates help improve trust and reputation within the UniThrift ecosystem. This final loop ensures data integrity and allows the marketplace to provide reliable user experiences for future transactions.