# WEB APP PRICE COMPARATOR – PRICE COMPARE APP

# Planning

## Define core features

Key features are:

* Users can add items with price and store information
* View list of items with price comparisons across stores
* Option to edit/delete prices
* Search or filter by item name or store

## Choose tech stack

* Frontend: React
* Backend: Express.js
* Database: MongoDB

# Coding

## Day 1 (07/27/25) – Frontend with React

### Learn to install Node.js

* **Go to** [**Node.js — Download Node.js®**](https://nodejs.org/en/download) **to install .msi file**
* On terminal: **node -v**

**npm -v** (Node Package Manager)

### Create a React app

* On terminal: **cd C:\Users\tranl\OneDrive\price-compare-app**

**npx create-react-app client**

(Việc này sẽ **tạo folder with name: client/ đầy đủ file React** bên trong như package.json, src/, public/...)

**cd client**

**npm start**

* You can now view client in the browser:

Local: <http://localhost:3000>

On Your Network: <http://192.168.0.109:3000>

* Open file “App.js” inside client/src/ folder with any text editor (VS Code or VS)

**Write the new app like below:**

import React from 'react';

function App() {

  return (

    <div style={{padding: 20}}>

      <h1>🛒 Price Compare App</h1>

      <button onClick={() => alert("Hello!")}> Click me </button>

    </div>

  );

}

export default App;

### What I’ve learned

* React là một cách viết HTML kiểu mới, dùng JavaScript lồng vào.
* <button onClick={...}> là mình nói "khi click, thì làm gì"
* Bé chưa cần hiểu sâu, cứ làm và xem chuyện gì xảy ra!

## Day 2 (07/28/25) – Backend w/ Express.js + Node.js

* Receiving item data from the React frontend and storing it temporarily in memory for now.
* Will build a basic backend where:
* React sends item name, store, and price
* Express receives the item and saves it (temporarily in memory)
* React can later fetch and show all saved items

### Quick knowledge

* **Node.js** lets us run JavaScript outside the browser — on the server.

**Express.js** is a framework built on top of Node.js that makes it easier to build **APIs** (which React can talk to).

### Create an Express server

* In terminal: **cd C:\Users\tranl\OneDrive\price-compare-app\server**
* Initialize Node.js project:

**npm init -y**

* This creates a package.json file so we can install backend libraries
* Install Express and CORS:

**npm install express cors**

* express = for making the backend/API
* cors = allows React (running on another port) to talk to the backend
* In price-compare-app/server/, create a file called server.js:

const express = require('express');

const cors = require('cors');

const app = express();

const PORT = 5000;

app.use(cors());

app.use(express.json());

let items = []; // TEMPORARY in-memory data

// Handle POST (save new item)

app.post('/api/items', (req, res) => {

const { name, store, price } = req.body;

const newItem = { name, store, price: parseFloat(price) };

items.push(newItem);

res.status(201).json(newItem);

});

// Handle GET (show all items)

app.get('/api/items', (req, res) => {

res.json(items);

});

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

* In terminal, run:

**node server.js**

### What I’ve learned

* Added 2 routes:
* POST /api/items → save an item
* GET /api/items → return all saved items

## Day 3 (07/28/25) - connect Frontend ↔ Backend

* Goal: Make React talk to backend using ***API requests*** via a library called ***axios***.

I’ve built both:

* ✅ React app (frontend)
* ✅ Express server (backend)

Today’s goal is to **connect the two**, so that:

* When you submit an item from the React form → it gets sent to the Express server
* Then the server saves it (in memory for now)
* You can also **fetch all items** and display them in the frontend

## Questions

Please explain simple and clear to understand.

**For Day 1 and 2**:

### Reason for installing Node.js at the very beginning?

👉 Because **Node.js is the engine** that lets us run JavaScript code **outside of the browser**.

Think of it like this:

* Normally, JavaScript runs **inside Chrome or a website**
* But to build things like **servers**, **databases**, or **backend logic**, we need to run JavaScript **on your computer** (not inside the browser)

📦 Also, Node.js gives us a tool called **npm** (Node Package Manager), which lets us install other tools like React, Express, etc.

✅ So: **Node.js is required** to build and run both your frontend (React) and backend (Express).

### Why does “npx create-react-app client” create so many files like package.json, src/, public/...? Why do I need a whole folder like that?

👉 npx create-react-app client means:

“Hey computer, use a tool to create a ready-to-use React app inside the folder called client.”

Inside client/:

* package.json: tells the app which tools and libraries it uses (like React)
* src/: where your React **code and components** live (your logic)
* public/: where things like your base index.html live (your static content)

✅ React is a **complex tool** under the hood, so this command sets up everything you need **automatically**, inside its own folder so it doesn’t mess with other files.

🧠 Why in a separate folder?  
Because your project has **two separate apps**:

* Frontend (React) → lives in client/
* Backend (Express) → lives in server/

### Meaning of “a ready-to-use React app”?

**🧩 What does “a ready-to-use React app” mean?**

When I say **“a ready-to-use React app”**, I mean:

A full React project that’s already **set up for you to start coding immediately**, without needing to configure anything from scratch.

**⚙️ Without create-react-app, you'd have to:**

* Manually install and set up **React**
* Configure **Webpack** (a tool that bundles files)
* Configure **Babel** (a tool that lets you use modern JavaScript)
* Set up **folders**, **scripts**, and **development server**
* Handle a bunch of settings for compiling, testing, etc.

😩 That’s a lot — especially for beginners.

**✅ But with npx create-react-app, you get all this automatically:**

* A src/ folder where you can start writing components
* A working development server that hot-refreshes
* package.json that includes React and helpful scripts like:
  + npm start → runs the dev server
  + npm run build → prepares your app for deployment
* public/ folder with a basic index.html already linked

So… you can just jump straight into writing code like:

function App() {

return <h1>Hello world</h1>;

}

No setup headaches. Just open App.js and start building 🛠️

**📦 Think of it like:**

* You're baking a cake 🍰
* **Without create-react-app**: You have to go buy flour, sugar, baking soda, measure everything…
* **With create-react-app**: You get a pre-made cake mix. Just add water and bake!

That’s a “ready-to-use React app” 😄

### What is the “server”?

In “**Node.js** lets us run JavaScript outside the browser — on the server.”

💡 A **server** is just a **computer** (or program) that listens for requests and gives back answers.

For example:

* When React wants to **save an item**, it sends a request like:

POST /api/items { name: "Milk", store: "Target", price: 2.99 }

* The **server (Express)** listens and says:

“Got it! I’ll save this item.”

✅ So your server.js is the place that:

* Listens on port 5000
* Waits for requests
* Handles saving/getting items

### **What is a framework built on top of something? What are APIs? Why do we need API (Is it for saving the items)?**

In “Express.js is a framework built on top of Node.js that makes it easier to build APIs (which React can talk to).”

**👉 What is a framework?**

A **framework** is like a **ready-made structure** that helps you build something faster.

* **Express.js** is a framework built **on top of Node.js**.
* That means Express uses Node under the hood, but gives you **shortcuts** to do common things like:
  + Create routes
  + Handle requests
  + Send back data

**👉 What is an API?**

**API = Application Programming Interface**

Think of it as a **menu** that React uses to **talk to the backend**.

* React: “I want to add a new item”
* Express API: “Sure, send it to me at /api/items”

✅ So yes — API is how we **save items**, **get items**, etc.  
It’s like a translator between **frontend (React)** and **backend (Express)**.

### What is server.js?

server.js is your **main backend file**. It:

* Starts the Express server
* Handles incoming requests (like add/get items)
* Runs when you type node server.js

🧠 You can think of server.js as the **"brain"** of your server.

### Which part I have learned is Express.js, and which part is Node.js?

Let’s break it down with your actual code!

**👉 This part is from Node.js:**

const express = require('express'); // ← uses Node to run

const app = express();

app.listen(5000);

Node lets us run this code on your computer.

**👉 This part is Express.js:**

app.post('/api/items', (req, res) => { ... });

app.get('/api/items', (req, res) => { ... });

Express gives us easy tools like app.get and app.post to create routes (APIs).

💡 So: **Node = engine**, **Express = framework that runs on the engine**

**For Day 3**:

### Concepts Summary

| **Concept** | **In Simple Words** |
| --- | --- |
| **Node.js** | Runs JavaScript outside browser (backend engine) |
| **npm** | Tool for installing libraries like React/Express |
| **Express.js** | Framework on Node.js for building backend easily |
| **create-react-app** | Quickly sets up everything for React frontend |
| **API** | Way for React to talk to the server (e.g., save items) |
| **server.js** | File that starts your backend and handles requests |

# Deploying