ETL-Express – eCommerce Website Report

Link to Repo - **.wpress backup is provided within a zip in the turnitin submition**

---

1. Why I Picked the Stack I Did

When I started this assignment the brief was clear: build a small e-commerce site that talks to a custom React dashboard. I wanted something that could live inside Docker, feel modern, but still let me use WooCommerce because that is what the course covers. After trying a few block themes I landed on **SaasLauncher**. It ships with a clean hero section, nice gradients and—most important—full block-editor support. I made a child theme straight away; everyone tells you to never touch a parent theme file, and after breaking things twice I totally agree.

Docker was non-negotiable for me because my laptop is already crowded with older projects. The Compose stack I wrote spins up MariaDB 10.11, WordPress 6.8 on PHP 8.3-FPM, Redis for object cache, Mailhog for e-mail catch-all and Nginx as the web front. Everything talks over an internal Docker bridge so the host stays clean. That decision saved me later when I needed to wipe everything and start fresh—one `docker compose down -v` and I had a blank slate.

---

2. Plugin Line-up

WordPress can do a lot out of the box, but turning it into a store still means plugins. After some trial and error my list settled at the table below. I kept the table because it is the quickest way to see what each add-on does.

Purpose	Plugin
Storefront	WooCommerce
Payments	WooCommerce PayPal Payments (Sandbox) and real · WooPayments
Security	Wordfence · Wordfence Login Security
Marketing / Pixel	Google for WooCommerce · Kliken Ads + Pixel for Meta
SEO	Yoast SEO
Speed / Cache	W3 Total Cache
Email	WP Mail SMTP (routes to Mailhog during development)
Product Filters	WOOF – WooCommerce Product Filters
Imports & Backups	All-in-One WP Migration · Duplicator · Advanced Import
Dev Tools	Query Monitor
Custom CSS/JS	Simple Custom CSS & JS

Mailhog is reachable at http://localhost:8025. Every order e-mail drops in there so I can read it without sending anything to real inboxes.

---

3. What the Site Can Do Right Now

I began by creating the fifteen demo products.Each product now shows a hero image, a short excerpt and the longer description tab. I styled the buttons to match the purple gradient of SaasLauncher, which looks way less "default WooCommerce grey."

A shopper lands on the home page, scrolls the catalogue, and then adds items to the cart. The cart and checkout pages use the SaasLauncher full-width template so there is no ugly sidebar. On checkout they see two sandbox gateways: **PayPal** The PayPal is easiest because it guides you to a PayPal page and then back. Once payment is accepted WooCommerce fires the `processing` → `completed` e-mails. They both appear inside Mailhog, complete with order details.

On the back end I enabled the WooCommerce analytics dashboard, the Orders screen, and reports.

Performance is decent. Autoptimize aggregates and minifies JS and CSS, lazy-loads images and strips query strings. W3 Total Cache stores static HTML and object cache in Redis. Not perfect, but good for a local stack.

Yoast SEO took the most patience. I went through every product and page, wrote meta titles, added a focus keyphrase, and tried to get the famous "green smiley."

---

4. The Hard Parts and How I Got Past Them

4.1 WordPress Inside Docker: Permission Nightmares, Lost Files and a Busy UI

The first two hours were spent chasing **permission errors**. WordPress runs as `www-data`, but my host files belonged to me. The Theme Editor refused to save, uploads failed silently, and the Customiser showed a blank sidebar. After reading three Stack Overflow threads I finally ran:

```
 sudo chown -R $USER:www-data wp-content/themes/saaslauncher
sudo chmod -R 775 wp-content/themes/saaslauncher
```

Lesson learned: inside Docker, PHP needs write access, but I still need to edit with VS Code, so group write (`775`) and the right owner pair solves it.

Then the site went white. The error log said it could not find `inc/core/init.php`. Block themes look like pure HTML, but they still include PHP helpers. My local copy was incomplete because Git ignored the `inc` directory. I opened an interactive shell inside the container, tar-zipped the real theme folder, `docker cp`-ed it out, and extracted it on the host. Suddenly the site lived again.

The WordPress admin UI itself took the longest to get used to. Screen Options hide half the columns, and some settings are buried under nested menus ,now I can find things quickly, but that first pass was slow and frustrating.

4.2 Marketing Plugins That Wouldn't Play Nice

I wanted to connect the store to Google Shopping and Facebook so my products could appear in ads automatically. That means setting up **Google for WooCommerce** and **Kliken Ads + Pixel for Meta**.

Google for WooCommerce walks you through Merchant Center setup and then tries to call back to your site with a verification token. Because I'm running on **localhost:8080** behind Docker, Google could not reach the callback URL. I spun up an `ngrok` tunnel, but Merchant Center refused the self-signed certificate and flagged the domain as unsafe. I spent an afternoon tweaking `ngrok` settings before I accepted the limitation: without a public HTTPS domain the wizard will never finish.

Kliken Ads had a similar roadblock. The plugin demands a live URL so Meta can verify the pixel. It kept looping on the "Connect your store" step, insisting my domain was unreachable. Two hours disappeared into troubleshooting before I parked both plugins. They will be revisited once I deploy to a staging VPS with Let's Encrypt.

4.3 Building the React Mock ETL App

The dashboard lives in `etl-express-ui/` and runs on port **3000**. I used **Create React App** because it is familiar, though I know Vite would build faster. My stack: React 19, MUI 6, Recharts, and **@xyflow/react**—the maintained fork of React Flow—plus Lodash for debouncing.

Shared state. Everything lives inside a React Context called `PipelineContext`: each pipeline has `nodes`, `edges`, `runCount`, `load`, and `lastRun`. Wrapping the app means every page can read and write that shared store.

Dashboard page. Shows three KPI cards, a management table, and two graphs. You can add a pipeline via a modal dialog, edit rows, or hit the play icon to simulate a run.

Pipeline Builder. The left sidebar lists pipelines and a **Node Toolbox**. Drag a node type onto the canvas, connect edges, right-click for a toolbar (edit, delete, config). Grid snaps keep the layout neat, the mini-map helps when the flow gets huge, and everything respects dark-mode colours.

Simulated runs. Clicking play sorts nodes by Y-coordinate, then runs them in sequence. Borders go orange (running) then green (completed). The edge animates dashed while data "moves." After each node, the dashboard increments `runCount` and rolls a random `load` percentage.

Package headaches. `react-flow-renderer` is abandoned and broke on Node 21. Upgrading to **@xyflow/react** fixed it but forced import changes and a CSS tweak. CodeMirror 6 wants ES-modules, clashing with CRA's CommonJS loader. For now I patched a local `package.json` with `{ "type": "module" }`—ugly but effective

4.4 Daily Workflow

Mounting the theme as a volume means I can edit `functions.php`, hit refresh and see changes—no docker rebuilds. Mailhog mirrors every WooCommerce e-mail, so I tweak templates without spamming real accounts. **Query Monitor** stays active in the admin bar and flags slow queries or hooks. If a page drags I open the console and know exactly which function is guilty.

5. What I Learned Along the Way

1. **Docker is forgiving once you grasp ownership.** Most "WordPress can't upload" errors trace back to wrong UID/GID on files.
2. **Block themes still rely on PHP.** Even though they look like pure HTML and JSON, removing one helper file will white-screen the site.
3. **The WordPress admin and UI is dense.** Very dense
4. **React + WordPress = fine.** Keep ports separate during dev, add CORS headers only when hosting behind the same domain.
5. **Marketing plugins need public HTTPS.** Localhost + self-signed certs break their onboarding.
6. **CSS micro-patches save sanity.** One extra rule fixed the Wordfence 2-FA layout.
7. **Backups are king.** All-in-One WP Migration restored my site twice when I broke `functions.php`.

6. Who I'm Building This For & Why It Looks Like This

I'm building **ETL-Express** for small and medium-sized businesses, start-ups and solo freelancers who need to move data but don't have a full-time data engineer. I'm also thinking about younger, tech-savvy people in their twenties and thirties—people like me—who expect dark-mode apps and clean, modern interfaces.

I chose the purple-on-black style because it feels familiar to anyone who spends their day in tools like VS Code or Discord. The bright pink-to-purple buttons pop on a laptop, phone or classroom projector, so it's always clear where to click next. Large text and wide cards make the key numbers easy to read during quick stand-ups.

The drag-and-drop pipeline builder means you can put a data flow together without writing a lot of code.

In short, I designed ETL-Express to be friendly, modern and stress-free, so everyday people can move their data around without getting lost.

7. Closing Thoughts

This project taught me more than I expected. I thought WordPress would be the easy part and React would be the challenge; it turned out Docker permissions and plugin quirks consumed most of my time. Still, I now have a working shop, a data-pipeline mock app, and a clear path to production. The next milestone is getting a public domain so marketing integrations can finally light up.

Appendix – Developer Manual / README

Below is the same README I keep in the repo so anyone can spin the project up in minutes.

ETL-Express: Full-Stack WordPress + React Starter

Overview

ETL-Express is a Dockerized dev environment that combines:

- WordPress + WooCommerce stack (MariaDB, Redis, Mailhog, Nginx)
- React dashboard in `etl-express-ui/`
- Wordfence 2-FA, SaasLauncher theme (child-themed), and full local editing

Prerequisites

- Docker & Docker Compose
- Node 21.7.1 + Yarn 1.22 (or npm)

Quick Start

```
 # clone and enter the project
git clone <repo-url>
cd etl_express_starter

# start containers (WordPress - 8080, Mailhog - 8025)
docker compose up -d
```

1. **Restore backup** – open `http://localhost:8080/wp-admin`, install *All-in-One WP Migration*, import the `.wpress` file.
2. **Run React** – in a second terminal:

```
 cd etl-express-ui
yarn install
yarn start   # opens http://localhost:3000
```

Folder Map

Path	What's inside
`etl-express-ui/`	React dashboard source
`docker-compose.yml`	WordPress + DB + Nginx stack
`nginx.conf`	Passes PHP to the `wordpress:9000` container
`wp-content/themes/saaslauncher/`	SaasLauncher theme mounted as live volume

Security Notes

- Wordfence supplies 2-FA and a firewall; dark-mode CSS lives in `style.css`.
- Theme is mounted with:

  ```
  - ./wp-content/themes/saaslauncher:/var/www/html/wp-content/themes/saaslauncher
  ```

so edits are instant, no container rebuild.

Common Fixes

- **Blank theme or 2-FA box invisible?** Verify `inc/` exists and run:

  ```
   sudo chown -R $USER:www-data wp-content/themes/saaslauncher
  sudo chmod -R 775 wp-content/themes/saaslauncher
  ```

- **Need e-mails?** Open Mailhog at `http://localhost:8025`.

That’s it—run the containers, import the backup, start the React dev server, and you have the full stack running locally.