DAILY ACCOMPLISHMENT REPORT

Field	Detail
Project	AA2000 Security Website (E-commerce)
Date	February 23, 2026
Prepared by	Clarence A. Portugal
Position	Intern

Summary

Accomplishments for the day focused first on converting the project to React (framework/language migration); then implementing the database using MySQL on XAMPP and converting from hardcoded data to database-driven; building the admin panel with product management, database-backed login, dashboard with charts and stat cards, mobile-responsive layout, product filters, and realtime sync; and adding checkout orders to the database with an admin Orders page to view them.

Accomplishments

1. Conversion to React (Framework / Language)

Area	Action
Framework migration	Converted the existing website/project to React (from previous language or framework, e.g. plain HTML/JS or other).
Project setup	Set up React project (e.g. Vite + TypeScript); structured components, pages, and routing for the AA2000 e-commerce site.
UI & pages	Rebuilt or migrated pages (Home, Products, Blogs, etc.) and UI components in React; maintained site structure and content.
Base for later work	This React codebase became the base for adding database, admin panel, and all subsequent features.

2. Database Implementation (MySQL on XAMPP)

Area	Action
MySQL / XAMPP	Set up MySQL via XAMPP: start MySQL in XAMPP Control Panel; created database (e.g. `aa2000`) for the project.
Schema	Created `scripts/schema.sql`: `products` table (id, name, category, price, description, full_description, image, specs, inclusions, installation_price, created_at) and `admin_users` table (id, username, password_hash, created_at). Run in MySQL (e.g. `mysql -u root -p aa2000 < scripts/schema.sql`) to create tables.
Connection & .env	Configured `.env` with MYSQL_HOST, MYSQL_USER, MYSQL_PASSWORD, MYSQL_DATABASE, and PORT so the server connects to the local MySQL (XAMPP) instance.
Server (Express + mysql2)	Built `server/index.js`: Express app with `mysql2/promise` `createPool()` using .env; all product and auth data now read/write from MySQL instead of in-memory or hardcoded.
Convert code to use DB	Removed hardcoded products from the codebase; frontend ProductsContext fetches from GET /api/products; admin add/edit/delete call POST/PUT/DELETE /api/products which persist to MySQL; localStorage used only as fallback when API/server is not running.

3. Database & Backend (API & Auth)

Item	Description
Products API	GET /api/products (list), GET /api/products/:id (one), POST (create), PUT /api/products/:id (update), DELETE /api/products/:id; all use MySQL via pool.query.
Seed script	Added `npm run seed-admin` (runs `scripts/seed-admin.js`) to create default admin user (username: admin, password: admin) with bcrypt hash in `admin_users`; run once after schema.
Login API	Added POST /api/auth/login; validates username/password against `admin_users` table (bcrypt.compare); returns success or 401.

Docs	SETUP-LOCAL.md (or project docs) for steps: XAMPP MySQL on, create DB, run schema, run seed-admin, set .env, run server.

4. Admin Panel & Product Management

Area	Action
Admin panel	Implemented admin section with login; credentials validated against MySQL `admin_users` table (bcrypt).
Products CRUD	Admin can add, edit, and delete products; data stored in MySQL via API or localStorage when API offline.
Hardcoded removal	Removed all hardcoded products; products list now loaded from API/database only.
Products page filter	Added category filter on admin Products page: All, CCTV, Fire Alarms, Burglar Alarms; table and count update by selected filter.

5. Admin Dashboard

Area	Action
Stat cards	Four cards: Total Products (with “In X categories”), Categories (with avg products each + tags), Catalog value (total + avg per product), Data source (MySQL/Local + Connected/Offline and “X loaded”). Each card shows two metrics.
Charts	Products by category (bar chart, distinct color per category); Category distribution (donut with legend); Top by price (top 5 products list). Wrapped in ErrorBoundary so dashboard still works if charts fail.
Responsive	Stat cards: 2 per row on mobile, 4 on desktop; charts and top-by-price responsive; loading skeletons for stats and charts.
Refresh	Added Refresh button in header; refetchSilent used for background polling (no loading flash).

6. Admin Layout & Mobile

Item	Description
Sidebar (desktop)	Sidebar fixed; only main content area scrolls (h-screen overflow-auto on main, lg:ml-60 for content).
Mobile drawer	Burger menu in header (mobile-only); sidebar slides in as overlay; tap toggles open/close; X to close.
Nav	Dashboard and Products in sidebar; View site and Log out at bottom; active link highlighted (cyan).

7. Login & UX

Item	Description
Login form	Username + password fields; submit calls /api/auth/login; redirect to /admin/dashboard on success.
Show/hide password	Eye icon next to password field; toggles type between password and text.
Scrollbars	Hid all scrollbars site-wide in index.css (scrollbar-width: none; -ms-overflow-style: none; ::-webkit-scrollbar { display: none }); scroll behavior unchanged.

8. Realtime Updates

Item	Description
Polling	Admin layout runs refetchSilent() every 15 seconds so products data stays in sync with API/database.
After mutate	After add, edit, or delete product, refetchSilent() runs so list and dashboard reflect server state immediately.
refetchSilent	Added in ProductsContext; fetches from API and updates state without setting loading (no spinner).

9. Code Quality & Build

Type	Change

Context	ProductsContext: refetchSilent, safe defaults for products; AdminDashboard: safe data for charts (percent ?? 0, etc.).
Routes	Admin routes: /admin (login), /admin/dashboard, /admin/products; AdminLayout wraps dashboard and products.
Build	npm run build passes; recharts + react-is for charts.

10. Checkout & Orders (Database + Admin)

Area	Action
Schema	Added `orders` table (id, full_name, email, phone, address, city, zip_code, subtotal, discount_amount, discount_code, total, status, created_at) and `order_items` table (id, order_id, product_id, product_name, price, quantity). Card/CVV/expiry not stored (security).
Orders API	POST /api/orders: accepts customer info + items + totals; inserts into orders and order_items in a transaction. GET /api/orders: list all orders for admin. GET /api/orders/:id: single order with line items.
Checkout → DB	Checkout page now calls submitOrder() with form data and cart items; order is saved to MySQL. On success: confirmation, clear cart, redirect. On failure: error message and submit button re-enabled.
Admin Orders	New page Admin → Orders (/admin/orders): table of orders (ID, date, customer, email, total, status). Expand row to see phone, address, discount, and list of items (product name, qty, line total).

Files Modified

Lahat ng nasa listahan ay bahagi ng conversion to React at ng kasunod na database/admin work — hindi lang iilang file ang na-edit.

Category	Files
Config / Build	package.json, vite.config.ts, tsconfig.json, index.html, postcss.config.js, tailwind.config, .env
Entry & App	src/main.tsx, src/App.tsx, src/index.css, src/vite-env.d.ts

Layout & Components	Layout.tsx, Navbar.tsx, Footer.tsx, PageHead.tsx, PageTransition.tsx, PageSkeleton.tsx, ErrorBoundary.tsx, components/index.ts
Pages (Public)	Home.tsx, Products.tsx, ProductDetails.tsx, Blogs.tsx, About.tsx, Contact.tsx, Cart.tsx, Checkout.tsx, Redeem.tsx, pages/index.ts
Pages (Admin)	AdminLogin.tsx, AdminLayout.tsx, AdminDashboard.tsx, AdminProducts.tsx, AdminOrders.tsx
Contexts	ProductsContext.tsx, CartContext.tsx
Lib & Utils	lib/api.ts, lib/validations.ts, lib/animations.ts
Types & Constants	types/index.ts, constants/index.ts, constants/blogs.ts
Server	server/index.js
Scripts	schema.sql, seed-admin.js
Docs	SETUP-LOCAL.md (or project docs)

Notes

- **Sidebar:** Fixed on desktop; content only scrolls. On mobile, drawer with burger; nav and footer in sidebar. Nav includes Dashboard, Products, Orders.
- **Products:** Data from MySQL when API running; localStorage fallback when offline. Realtime: 15s polling + refetch after add/edit/delete.
- **Charts:** Bar chart (per-category colors), donut, top-by-price list; empty states when no products.
- **Orders:** Checkout submissions are saved to database (orders + order_items). Admin → Orders lists all checkouts; expand row to see items and shipping info.