# What you have (quick map)

Based on the public repo layout, you’re running a monorepo with PNPM workspaces:

/  
├─ apps/ # likely frontend + backend apps  
├─ utils/ # shared packages/helpers  
├─ scripts/ # CLIs/maintenance  
├─ database/ # SQL/migrations/seeds  
├─ .github/workflows/ # CI  
├─ package.json # root scripts  
├─ pnpm-workspace.yaml # workspace config  
├─ tsconfig.json # TS base (needs tightening)  
└─ docs/, repo-report/ # documentation

This plan turns the “mess” into a clean, type-safe monorepo you can ship from cPanel (frontend static) and Render (backend Node/Express) without breaking your current DB and SKU invariants.

# Migration goals

1. **npm→pnpm made reliable**: deterministic lockfile, workspace isolation, strict engine versions.
2. **JS/JSX→TS/TSX**: incremental, with allowJs for Phase 1, then strict: true in Phase 2.
3. **Consistent ESM/CJS**: one module system per package; align build outputs.
4. **Vite build for frontend** (or keep CRA temporarily), deploy /dist to cPanel.
5. **Express backend** on Render with Zod validation, typed Postgres access, and correct CORS.
6. **AAA a11y & perf**: React Query, Suspense, skeletons, virtual lists.
7. **CI**: Type-check, lint, test, build for every package.

# 0) Prereqs

* **Node**: 20.11.x (LTS) locked via .nvmrc + engines.
* **PNPM**: 9.x (lock via packageManager field).
* **TypeScript**: 5.6+

corepack enable  
corepack prepare pnpm@9.11.0 --activate  
node -v # v20.11.x  
pnpm -v # 9.11.0

Add .nvmrc with v20.11.1 and ask Render to use it; cPanel only receives built assets.

# 1) Workspace wiring

## pnpm-workspace.yaml

packages:  
 - "apps/\*"  
 - "utils/\*"  
 - "scripts/\*"  
 - "database"

## Root package.json (authoritative scripts)

{  
 "name": "mana-meeples-singles-market",  
 "private": true,  
 "packageManager": "pnpm@9.11.0",  
 "engines": { "node": ">=20.11.0" },  
 "scripts": {  
 "dev": "pnpm -r --parallel run dev",  
 "build": "pnpm -r run build",  
 "typecheck": "pnpm -r run typecheck",  
 "lint": "pnpm -r run lint",  
 "test": "pnpm -r run test",  
 "format": "pnpm -r run format",  
 "clean": "pnpm -r run clean && rimraf node\_modules .turbo",  
 "ci": "pnpm typecheck && pnpm lint && pnpm test && pnpm build"  
 },  
 "devDependencies": {  
 "typescript": "^5.6.3",  
 "@types/node": "^20.12.12",  
 "eslint": "^9.12.0",  
 "eslint-config-prettier": "^9.1.0",  
 "eslint-plugin-import": "^2.29.1",  
 "eslint-plugin-react": "^7.37.1",  
 "eslint-plugin-react-hooks": "^4.6.2",  
 "prettier": "^3.3.3",  
 "rimraf": "^6.0.1",  
 "turbo": "^2.1.3"  
 }  
}

You can swap Turbo for plain PNPM if you prefer—scripts above work either way.

# 2) TypeScript foundation

## tsconfig.base.json (at repo root)

{  
 "compilerOptions": {  
 "target": "ES2022",  
 "lib": ["ES2022", "DOM", "DOM.Iterable"],  
 "module": "ESNext",  
 "moduleResolution": "Bundler",  
 "baseUrl": ".",  
 "paths": {  
 "@utils/\*": ["utils/\*/src/\*"],  
 "@shared/\*": ["utils/shared/src/\*"]  
 },  
 "allowJs": true,   
 "checkJs": false,  
 "jsx": "react-jsx",  
 "noEmit": true,  
 "forceConsistentCasingInFileNames": true,  
 "resolveJsonModule": true,  
 "isolatedModules": true,  
 "skipLibCheck": true,  
 "strict": false  
 },  
 "include": ["apps", "utils", "scripts", "database", "types.d.ts"],  
 "exclude": ["\*\*/dist", "\*\*/.next", "\*\*/build"]  
}

**Phase 2**: flip strict: true, add exactOptionalPropertyTypes: true, and remove allowJs after conversion.

## Per-package tsconfig

* **Frontend** apps/web/tsconfig.json extends base; sets types: ["vite/client"].
* **Backend** apps/api/tsconfig.json extends base; types: ["node"].

{  
 "extends": "../../tsconfig.base.json",  
 "compilerOptions": { "outDir": "dist" },  
 "include": ["src"]  
}

# 3) Linting & formatting

.eslintrc.cjs at root:

module.exports = {  
 root: true,  
 extends: [  
 "eslint:recommended",  
 "plugin:react/recommended",  
 "plugin:react-hooks/recommended",  
 "plugin:import/recommended",  
 "plugin:import/typescript",  
 "prettier"  
 ],  
 parser: "@typescript-eslint/parser",  
 plugins: ["@typescript-eslint"],  
 settings: { react: { version: "detect" } },  
 env: { browser: true, node: true, es2022: true },  
 rules: {  
 "react/prop-types": "off",  
 "import/no-unresolved": "off" // handled by TS path mapping  
 },  
 ignorePatterns: ["\*\*/dist", "\*\*/\*.d.ts"]  
};

prettier.config.cjs:

module.exports = { semi: true, singleQuote: false, printWidth: 100 };

# 4) Frontend (apps/web) — Vite + React + AAA

**package.json**

{  
 "name": "web",  
 "private": true,  
 "version": "0.1.0",  
 "type": "module",  
 "scripts": {  
 "dev": "vite",  
 "build": "vite build",  
 "preview": "vite preview",  
 "typecheck": "tsc -p tsconfig.json --noEmit",  
 "lint": "eslint src --ext .ts,.tsx",  
 "test": "vitest run",  
 "format": "prettier --write \"\*\*/\*.{ts,tsx,css,md}\""  
 },  
 "dependencies": {  
 "react": "^18.3.1",  
 "react-dom": "^18.3.1",  
 "@tanstack/react-query": "^5.56.2",  
 "zod": "^3.23.8"  
 },  
 "devDependencies": {  
 "vite": "^5.4.8",  
 "@vitejs/plugin-react": "^4.3.1",  
 "vitest": "^2.1.3",  
 "@testing-library/react": "^16.0.1",  
 "@testing-library/user-event": "^14.5.2",  
 "@types/react": "^18.3.12",  
 "@types/react-dom": "^18.3.1"  
 }  
}

**vite.config.ts**

import { defineConfig } from "vite";  
import react from "@vitejs/plugin-react";  
  
export default defineConfig({  
 plugins: [react()],  
 build: { sourcemap: true, outDir: "dist" },  
 server: { port: 5173, strictPort: true }  
});

**AAA baseline**

* Global landmarks (<header>, <nav aria-label>/skip links, <main id="main">, <footer>)
* Focus ring visible at 3:1 contrast, not removed.
* Keyboard: all interactive elements reachable in DOM order; roving tabindex for composite widgets.
* Announce route/section changes: aria-live="polite" region.
* Respect prefers-reduced-motion; skeletons not spinners.

**Example: accessible VariationSelector.tsx**

import { useId, useEffect, useRef } from "react";  
  
type Option = { id: string; label: string; disabled?: boolean };  
export function VariationSelector({  
 label = "Choose variation",  
 options,  
 value,  
 onChange  
}: {  
 label?: string;  
 options: Option[];  
 value: string | null;  
 onChange: (id: string) => void;  
}) {  
 const groupId = useId();  
 const liveRef = useRef<HTMLDivElement>(null);  
  
 useEffect(() => {  
 if (!value || !liveRef.current) return;  
 liveRef.current.textContent = `Selected ${options.find(o => o.id === value)?.label ?? ""}`;  
 }, [value, options]);  
  
 return (  
 <fieldset aria-labelledby={`${groupId}-label`}>  
 <legend id={`${groupId}-label`}>{label}</legend>  
 <div role="radiogroup" aria-labelledby={`${groupId}-label`}>  
 {options.map((o) => (  
 <label key={o.id} style={{ display: "block" }}>  
 <input  
 type="radio"  
 name={groupId}  
 value={o.id}  
 checked={value === o.id}  
 onChange={() => onChange(o.id)}  
 disabled={o.disabled}  
 />  
 {o.label}  
 </label>  
 ))}  
 </div>  
 <div aria-live="polite" aria-atomic="true" ref={liveRef} style={{ position: "absolute", left: -9999 }} />  
 </fieldset>  
 );  
}

**Deploy to cPanel**

* pnpm -F web build → upload /apps/web/dist to your cPanel docroot (or via CI → FTP).

# 5) Backend (apps/api) — Express + Zod + Postgres

**package.json**

{  
 "name": "api",  
 "private": true,  
 "version": "0.1.0",  
 "type": "module",  
 "scripts": {  
 "dev": "node --watch --enable-source-maps ./src/index.ts",  
 "build": "tsc -p tsconfig.json",  
 "start": "node ./dist/index.js",  
 "typecheck": "tsc -p tsconfig.json --noEmit",  
 "lint": "eslint src --ext .ts",  
 "test": "vitest run"  
 },  
 "dependencies": {  
 "express": "^4.21.1",  
 "zod": "^3.23.8",  
 "pg": "^8.12.0",  
 "cors": "^2.8.5",  
 "helmet": "^7.1.0"  
 },  
 "devDependencies": {  
 "@types/express": "^4.17.21",  
 "@types/cors": "^2.8.17"  
 }  
}

**src/index.ts**

import express from "express";  
import cors from "cors";  
import helmet from "helmet";  
import { cardsRouter } from "./routes/cards";  
  
const app = express();  
app.use(helmet());  
app.use(express.json({ limit: "1mb" }));  
  
app.use(cors({  
 origin: [process.env.FRONTEND\_ORIGIN ?? "\*"],  
 methods: ["GET", "POST", "PATCH"],  
 allowedHeaders: ["Content-Type", "Authorization"],  
 credentials: true  
}));  
  
app.get("/health", (\_req, res) => res.json({ ok: true }));  
app.use("/cards", cardsRouter);  
  
const port = Number(process.env.PORT || 3000);  
app.listen(port, () => console.log(`API listening on :${port}`));

**src/db.ts** (typed pool)

import { Pool } from "pg";  
export const pool = new Pool({ connectionString: process.env.DATABASE\_URL });  
export async function query<T = unknown>(text: string, params: unknown[] = []) {  
 const res = await pool.query<T>(text, params);  
 return res.rows;  
}

**src/routes/cards.ts** (uses your schema constraints)

import { Router } from "express";  
import { z } from "zod";  
import { query } from "../db";  
  
const router = Router();  
  
const listParams = z.object({  
 q: z.string().trim().max(64).optional(),  
 set\_id: z.string().trim().optional(),  
 finish: z.enum(["NONFOIL","FOIL","ETCHED"]).optional(),  
 treatment: z.string().trim().optional(),  
 limit: z.coerce.number().int().min(1).max(50).default(24),  
 offset: z.coerce.number().int().min(0).default(0)  
});  
  
router.get("/", async (req, res) => {  
 const p = listParams.parse(req.query);  
 const values: any[] = [];  
 const where: string[] = [];  
  
 if (p.q) { values.push(p.q); where.push(`search\_tsv @@ plainto\_tsquery('simple', $${values.length})`); }  
 if (p.set\_id) { values.push(p.set\_id); where.push(`set\_id = $${values.length}`); }  
 if (p.finish) { values.push(p.finish); where.push(`finish = $${values.length}`); }  
 if (p.treatment) { values.push(p.treatment.toUpperCase()); where.push(`treatment = $${values.length}`); }  
  
 const sql = `  
 SELECT id, name, set\_id, card\_number, finish, treatment, border\_color, frame\_effect, sku  
 FROM cards  
 ${where.length ? `WHERE ${where.join(" AND ")}` : ""}  
 ORDER BY set\_id, card\_number  
 LIMIT $${values.push(p.limit)} OFFSET $${values.push(p.offset)};  
 `;  
 const rows = await query(sql, values);  
 res.json({ items: rows, nextOffset: p.offset + p.limit });  
});  
  
export { router as cardsRouter };

Uses existing indexes: search\_tsv (GIN), and btree on finish, treatment—avoids N+1.

**Render**: set DATABASE\_URL, FRONTEND\_ORIGIN env vars; use start script.

# 6) JS→TS conversion strategy

1. Phase 1: **Rename** files to .ts/.tsx; add minimal types for props; keep any where needed.
2. **Enable JSDoc types** in tricky modules as an intermediate step.
3. Drop allowJs; turn on strict and exactOptionalPropertyTypes.
4. Add **React Query** types for API hooks.

**Codemods**

pnpm dlx ts-migrate-full .  
# or lightweight  
pnpm dlx jscodeshift -t node\_modules/react-codemod/transforms/class.js apps/web/src

# 7) Shared types + API client

utils/shared/src/types.ts

export type Finish = "NONFOIL" | "FOIL" | "ETCHED";  
export interface CardListItem {  
 id: string; name: string; set\_id: string; card\_number: string;  
 finish: Finish; treatment: string | null; border\_color: string | null;  
 frame\_effect: string | null; sku: string;  
}

utils/shared/src/http.ts

export async function api<T>(input: RequestInfo, init?: RequestInit): Promise<T> {  
 const res = await fetch(input, { headers: { "Content-Type": "application/json" }, ...init });  
 if (!res.ok) throw new Error(`${res.status}: ${await res.text()}`);  
 return (await res.json()) as T;  
}

# 8) A11y-first frontend slices (ready to paste)

## CardGrid (virtualized)

import { memo } from "react";  
import { useVirtualizer } from "@tanstack/react-virtual";  
  
export const CardGrid = memo(function CardGrid({ items }: { items: { id: string; name: string }[] }) {  
 const parentHeight = 700; const rowHeight = 140;  
 const rowVirtualizer = useVirtualizer({ count: items.length, getScrollElement: () => document.getElementById("grid-scroll"), estimateSize: () => rowHeight });  
 return (  
 <div id="grid-scroll" role="grid" aria-rowcount={items.length} style={{ height: parentHeight, overflow: "auto", position: "relative" }}>  
 <div style={{ height: rowVirtualizer.getTotalSize(), width: "100%", position: "relative" }}>  
 {rowVirtualizer.getVirtualItems().map(vi => (  
 <div role="row" aria-rowindex={vi.index+1} key={vi.key} style={{ position: "absolute", top: vi.start, left: 0, width: "100%", height: rowHeight }}>  
 {/\* your card cell \*/}  
 </div>  
 ))}  
 </div>  
 </div>  
 );  
});

## Filters (debounced, SR-friendly)

import { useId } from "react";  
export function SearchBox({ value, onChange }: { value: string; onChange: (v: string)=>void }) {  
 const id = useId();  
 return (  
 <div>  
 <label htmlFor={id}>Search cards</label>  
 <input id={id} type="search" value={value} onChange={e=>onChange(e.target.value)}  
 inputMode="search" aria-describedby={`${id}-hint`} />  
 <div id={`${id}-hint`} className="sr-only">Type a name or number. Results update as you type.</div>  
 </div>  
 );  
}

# 9) CI (GitHub Actions)

.github/workflows/ci.yml

name: CI  
on: [push, pull\_request]  
jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v4  
 - uses: actions/setup-node@v4  
 with: { node-version: 20, cache: 'pnpm' }  
 - run: corepack enable && corepack prepare pnpm@9.11.0 --activate  
 - run: pnpm i --frozen-lockfile  
 - run: pnpm ci

# 10) Render & cPanel deployment notes

**Render (api)**

* Build: pnpm -F api build
* Start: pnpm -F api start
* Env: DATABASE\_URL, FRONTEND\_ORIGIN (your cPanel URL)

**cPanel (web)**

* Build locally or via CI: pnpm -F web build
* Upload apps/web/dist → public\_html/ (or subdir)
* Ensure API base URL is env-driven: VITE\_API\_BASE=https://your-render-service.onrender.com

# 11) Common gotchas fixed by this plan

* **Mixed ESM/CJS** → each package sets "type": "module" (or omit) consistently and builds to ESM.
* **CORS** → explicit origin allowlist; test with test-cors-fix.sh.
* **Path aliases failing in TS** → moduleResolution: Bundler + Vite handles aliases.
* **Default imports from CJS** → enable esModuleInterop if needed, or switch to named imports.
* **Out-of-sync lockfiles** → single pnpm-lock.yaml at root; CI uses --frozen-lockfile.

# 12) Next steps checklist

# 13) Optional DB migrations (only if needed)

If you want faster facet lookups per set/game without touching your tables:

-- Concurrent refresh support  
CREATE EXTENSION IF NOT EXISTS pg\_cron;  
-- schedule hourly refresh if materialized views exist  
-- SELECT cron.schedule('refresh\_mv\_set\_variation\_filters', '0 \* \* \* \*', $$REFRESH MATERIALIZED VIEW CONCURRENTLY mv\_set\_variation\_filters;$$);

Keep SKUs unique and stable. Keep cards (set\_id, card\_number, finish) unique as-is.

# 14) Dev ergonomics

* VS Code: Recommend extensions—ESLint, Prettier, TypeScript TSServer plugin.
* Path IntelliSense: TS paths match real folders; avoid deep relative imports.
* Absolute imports from @shared/\* across apps via PNPM linking.

## That’s it

Use this as your single source of truth while cleaning up. Ping me with any specific files that are noisy—I’ll rewrite them in TS with tests and a11y baked in.