Vercel app explanation

what is vercel app?

Vercel is a cloud platform for deploying front-end web apps—especially Next.js—without managing servers. When people say "a Vercel app," they usually mean a web application that's deployed as a **Vercel Project** and served on Vercel's global edge network.

What you get

- **Git → Auto deploys:** Push to GitHub/GitLab/Bitbucket → Vercel builds and deploys. Preview URLs for every PR, one Production URL on merge.
- **Serverless & Edge Functions:** API routes and SSR run on demand (Node/Edge Runtime), scaled automatically.
- **Global CDN:** Static assets cached at the edge; image/font optimization.
- **Env vars & secrets, cron jobs, and analytics.**
- **Storage options:** Vercel KV (Redis), Postgres, Blob storage (optional add-ons).
- **Custom domains & HTTPS** out of the box.

How it works (quick)

1. `vercel` (CLI) or connect your repo in the dashboard.
2. Vercel detects your framework (e.g., Next.js) and runs the build.
3. Static files go to the CDN; dynamic routes become Serverless/Edge Functions.
4. You get a preview URL; merge to deploy to Production.
5. Add your custom domain; set `ENV` in Project Settings.

Tiny example (Next.js)

```bash
npm i -g vercel
vercel login
vercel   # creates a Preview deployment
vercel --prod  # promotes to Production
```

Optional `vercel.json`:

```json
```

```
{
  "buildCommand": "next build",
  "outputDirectory": ".next",
  "rewrites": [{ "source": "/api/(.*)", "destination": "/api/$1" }]
}
```

Gotchas

- Cold starts for Serverless (use Edge Functions for lower latency).
- Choose the right region for data/latency.
- Mind your plan limits (bandwidth, invocations, storage).

If you tell me what you're deploying (Next.js, static site, FastAPI behind a proxy, etc.), I can give you exact steps and config.

"Using AI tools of your choice, build a responsive single-page app that displays cryptocurrency prices from a free API (e.g., CoinGecko).
Requirements:
Homepage with a list of coins, prices, and % change.
Use Vanry/USDT Pair as the first crypto currency to show.
Clicking a coin shows a detail view with a chart.
Add a search + filter feature.
UI must look professional (use Tailwind, Material UI, or styled components).
Bonus: Dark/light mode toggle.
Document where AI helped you (prompts, code snippets) and what you adjusted manually."

I want to embark on the project above the finally deploy it in vercel. Could you please do an expansive research on the fastest yet most impressive approach to display it and make sure the features appear professional and the functionality is well thought after. I want to finish the project in an evening or so. I have strong familiarity with python so I would appreciate if you build the core logic using it (maybe fastAPI, or anything else but just make sure it is flawless). And for the front-end/display prioritize aesthetics, reactivity and simplicity of stack (as It is not my strongest suit).

Please provide a mindful expansive technical plan with code if possible, I will take it from there!

awesome project. here's a punchy, evening-friendly plan that gets you a slick SPA, a tiny Python core, and a one-click Vercel deploy. i'll give you the "why," the architecture, the exact steps, and ready-to-paste code.

---

why this approach

- **Front-end:** Next.js (App Router) + Tailwind + shadcn/ui + TanStack Query + Recharts. This combo ships fast, looks pro, and deploys flawlessly to Vercel.
- **Core logic in Python:** A **FastAPI** app exposed as a **Vercel Python Function** at `/api/*`. Vercel's Python runtime (beta) accepts an ASGI app directly—perfect for FastAPI—and runs on all plans.
- **Data:** CoinGecko's **free Demo API key** (30 req/min, 10k/month) is enough for an SPA with light polling; key is required since keyless public API was deprecated.

- **Caching:** We return `Cache-Control` directives so Vercel's edge cache absorbs most traffic (saves your rate limit and makes the app feel instant).
- **Spec fit:** Homepage list (price + %), **Vanry/USDT pinned first**, detail modal with chart, search/filter, dark/light toggle, and a small "AI notes" doc.

---

architecture (monorepo in one Vercel project)

```python
crypto-spa/
├── app/                    # Next.js (App Router) UI
│   ├── page.tsx            # coin list + search + filters + modal chart
│   ├── globals.css
│   └── components/
│       ├── ThemeToggle.tsx
│       ├── CoinRow.tsx
│       ├── CoinChart.tsx
│       └── SearchAndFilters.tsx
```

```
├── api/                # Vercel Functions (Python) – FastAPI
│   └── index.py          # exposes FastAPI "app"
├── requirements.txt      # FastAPI/httpx/cachetools
├── package.json
├── tailwind.config.ts
├── postcss.config.js
├── vercel.json         # ensure python runtime & headers
├── .env.local          # COINGECKO_API_KEY
└── docs/AI_NOTES.md      # where AI helped
```

How requests flow

UI → `/api/markets` (list, 24h change, optional sparkline)
UI → `/api/coins/{id}/chart?range=7d` (detail chart)
UI → `/api/tickers/vanar-chain?target=USDT` (for VANRY/USDT pin)

The FastAPI layer calls CoinGecko, normalizes data, sets caching headers, and returns clean JSON. (CoinGecko endpoints: `coins/markets`, `coins/{id}/market_chart`, `coins/{id}`/`tickers`.)

---

step-by-step in your evening

1. **Scaffold UI**

   ```bash
   npx create-next-app@latest crypto-spa --ts --eslint --app --tailwind
   cd crypto-spa
   npm i @tanstack/react-query recharts class-variance-authority tailwind-merge lucide-react next-themes
   npx shadcn-ui@latest init -d
   npx shadcn-ui@latest add button input select card table dialog toggle switch
   ```

2. **Python core**
   Create `api/index.py` and `requirements.txt` (below). Add **COINGECKO_API_KEY** in Vercel project settings later. (CoinGecko free Demo key docs/rates linked above.)
3. **UI components**
   Add list + search/filter + detail modal with chart + theme toggle.
4. **Vercel deploy**

```bash
npm i -g vercel
vercel
```

Vercel auto-detects Next.js and your Python function. (Python runtime accepts WSGI/ASGI via `app` var.)

5. **Polish**
   - tune s-maxage (Edge cache) to ~30s for lists, 60–120s for charts (great UX and API limits).
   - ensure **VANRY/USDT** stays pinned top using the tickers endpoint (we'll choose the highest-volume USDT market, e.g., Binance).

---

code — backend (FastAPI on Vercel)

`requirements.txt`

```ini
fastapi==0.111.0
httpx==0.27.0
cachetools==5.3.3
```

`api/index.py`

```python
from fastapi import FastAPI, HTTPException, Query
from fastapi.responses import JSONResponse
from typing import Optional
import os, time, math
import httpx
from cachetools import TTLCache

app = FastAPI(title="Crypto SPA API", version="1.0")

API_BASE = "https://pro-api.coingecko.com/api/v3"
API_KEY = os.environ.get("COINGECKO_API_KEY", "")
HEADERS = {"x-cg-demo-api-key": API_KEY} if API_KEY else {}

# Caches: keep tiny TTLs; Vercel Edge cache will do most heavy lifting.
```

```python
cache_markets = TTLCache(maxsize=8, ttl=25)   # list cache ~25s
cache_chart   = TTLCache(maxsize=64, ttl=50)  # chart cache ~50s
cache_tickers = TTLCache(maxsize=16, ttl=45)  # tickers cache ~45s

def cdn_cache_headers(seconds: int):
    # Vercel honors s-maxage (edge); browsers see max-age=0 to avoid stale UI.
    # Docs: https://vercel.com/docs/headers/cache-control-headers
    return {
        "Cache-Control": "public, max-age=0, must-revalidate",
        "Vercel-CDN-Cache-Control": f"max-age={seconds}",
    }

async def _get(client: httpx.AsyncClient, url: str, params: dict):
    r = await client.get(url, params=params, headers=HEADERS, timeout=10)
    if r.status_code >= 400:
        raise HTTPException(r.status_code, r.text)
    return r.json()

@app.get("/api/markets")
async def markets(
    vs: str = Query("usd"),
    per_page: int = Query(50, ge=1, le=250),
    page: int = Query(1, ge=1),
    search: Optional[str] = None,
    sparkline: bool = Query(True)
):
    key = (vs, per_page, page, search, sparkline)
    if key in cache_markets:
        data = cache_markets[key]
        return JSONResponse(data, headers=cdn_cache_headers(30))

    params = {
        "vs_currency": vs,
        "order": "market_cap_desc",
        "per_page": per_page,
        "page": page,
        "sparkline": "true" if sparkline else "false",
        "price_change_percentage": "1h,24h,7d"
    }
    # optional name/symbol filter via CoinGecko's markets lookup
    if search:
        params["names"] = search
    async with httpx.AsyncClient() as client:
        raw = await _get(client, f"{API_BASE}/coins/markets", params)

    # only return fields the SPA needs (keeps payload small)
    data = [
        {
            "id": c["id"],
            "symbol": c["symbol"].upper(),
            "name": c["name"],
            "image": c.get("image"),
            "price": c.get("current_price"),
            "change24h": c.get("price_change_percentage_24h"),
            "mcap": c.get("market_cap"),
            "sparkline": c.get("sparkline_in_7d", {}).get("price", []),
            "rank": c.get("market_cap_rank"),
        } for c in raw
    ]
    cache_markets[key] = data
    return JSONResponse(data, headers=cdn_cache_headers(30))

@app.get("/api/coins/{coin_id}/chart")
async def coin_chart(coin_id: str, range: str = Query("7d")):
    # map range -> days/interval
    mapping = {
        "24h": ("1", "minutely"),
        "7d": ("7", "hourly"),
        "30d": ("30", "daily"),
        "90d": ("90", "daily")
    }
    days, interval = mapping.get(range, ("7", "hourly"))
    key = (coin_id, days, interval)
    if key in cache_chart:
        return JSONResponse(cache_chart[key], headers=cdn_cache_headers(60))

    params = {"vs_currency": "usd", "days": days, "interval": interval}
    async with httpx.AsyncClient() as client:
        raw = await _get(client, f"{API_BASE}/coins/{coin_id}/market_chart", params)

    # coingecko returns [["ts","price"], ...] arrays under "prices"
    out = [{"t": p[0], "price": p[1]} for p in raw.get("prices", [])]
    cache_chart[key] = out
    return JSONResponse(out, headers=cdn_cache_headers(60))

@app.get("/api/tickers/{coin_id}")
async def coin_tickers(coin_id: str, target: str = Query("USDT")):
    key = (coin_id, target.upper())
    if key in cache_tickers:
        return JSONResponse(cache_tickers[key], headers=cdn_cache_headers(45))
    async with httpx.AsyncClient() as client:
        raw = await _get(client, f"{API_BASE}/coins/{coin_id}", {"tickers": "true"})
    tickers = raw.get("tickers", [])
    # pick best USDT pair by highest volume (and trust_score if present)
    usdt = [t for t in tickers if str(t.get("target")).upper() == target.upper()]
    usdt.sort(key=lambda t: (t.get("trust_score") == "green", t.get("volume", 0)), reverse=True)
    best = usdt[0] if usdt else None
    payload = {
        "exchange": best.get("market", {}).get("name") if best else None,
        "pair": f'{best.get("base")}/{best.get("target")}' if best else None,
        "last": best.get("last") if best else None
    } if best else {}
    cache_tickers[key] = payload
    return JSONResponse(payload, headers=cdn_cache_headers(45))

# Health
@app.get("/api/health")
async def health():
    return {"ok": True, "ts": int(time.time())}
```

`vercel.json`

```json
{
  "functions": {
    "api/**/*.py": { "runtime": "python3.12" }
  },
  "headers": [
    {
      "source": "/api/(.*)",
      "headers": [
        { "key": "Access-Control-Allow-Origin", "value": "*" }
      ]
    }
  ]
}
```

> Notes
> • Vercel’s Python runtime accepts an ASGI `app` variable (FastAPI) and is available on all plans (beta).
> • We use `Vercel-CDN-Cache-Control` / `Cache-Control` to steer edge + browser caching.
> • CoinGecko endpoints & chart semantics are documented here.

---

code — front-end (Next.js + Tailwind + shadcn)

`app/globals.css` (already created by the Next/Tailwind template)

`app/layout.tsx`

```tsx
import "./globals.css";
import { ThemeProvider } from "next-themes";

export const metadata = { title: "Crypto Prices", description: "SPA with FastAPI backend" };

export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html suppressHydrationWarning lang="en">
      <body>
        <ThemeProvider attribute="class" defaultTheme="system" enableSystem>
          <div className="min-h-screen bg-background text-foreground">{children}</div>
        </ThemeProvider>
      </body>
    </html>
  );
}
```

`app/components/ThemeToggle.tsx`

```tsx
```

```
"use client";
import { useTheme } from "next-themes";
import { Moon, Sun } from "lucide-react";
import { Button } from "@/components/ui/button";

export default function ThemeToggle() {
  const { theme, setTheme } = useTheme();
  const isDark = theme === "dark";
  return (
    <Button variant="outline" size="icon" onClick={() => setTheme(isDark ? "light" : "dark")}>
      {isDark ? <Sun className="h-4 w-4" /> : <Moon className="h-4 w-4" />}
    </Button>
  );
}
```

`app/components/CoinChart.tsx`

```tsx
"use client";
import { useQuery } from "@tanstack/react-query";
import { LineChart, Line, ResponsiveContainer, Tooltip, XAxis, YAxis } from "recharts";

export default function CoinChart({ coinId, range }: { coinId: string; range:
"24h"|"7d"|"30d"|"90d"; }) {
  const { data } = useQuery({
    queryKey: ["chart", coinId, range],
    queryFn: async () => {
      const r = await fetch(`/api/coins/${coinId}/chart?range=${range}`);
      return r.json();
    },
    refetchInterval: 30000
  });

  return (
    <div className="h-64 w-full">
      <ResponsiveContainer>
        <LineChart data={data || []}>
          <XAxis dataKey="t" tickFormatter={(t) => new Date(t).toLocaleDateString()} hide />
          <YAxis domain={["auto", "auto"]} hide />
          <Tooltip labelFormatter={(t) => new Date(+t).toLocaleString()} />
          <Line type="monotone" dataKey="price" strokeWidth={2} dot={false} />
      </LineChart>
    </ResponsiveContainer>
  </div>
 );
}
```

`app/components/CoinRow.tsx`

```tsx
import Image from "next/image";
import { Card } from "@/components/ui/card";

export default function CoinRow({ c, onClick }: { c: any; onClick: () => void }) {
 const pct = c.change24h ?? 0;
 return (
  <Card onClick={onClick} className="flex items-center justify-between p-3 hover:bg-
accent cursor-pointer">
   <div className="flex items-center gap-3">
    <Image src={c.image} alt={c.name} width={24} height={24} />
    <div className="flex flex-col">
     <span className="font-medium">{c.name} <span className="text-muted-
foreground">({c.symbol})</span></span>
     <span className="text-xs text-muted-foreground">Rank #{c.rank}</span>
    </div>
   </div>
   <div className="text-right">
    <div className="font-semibold">${c.price?.toLocaleString()}</div>
    <div className={`${pct >= 0 ? "text-green-600" : "text-red-600"} text-sm`}>
     {pct?.toFixed(2)}%
    </div>
   </div>
  </Card>
 );
}
```

`app/components/SearchAndFilters.tsx`

```tsx
"use client";
import { Input } from "@/components/ui/input";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from
```

```
"@/components/ui/select";

export default function SearchAndFilters({ query, setQuery, sort, setSort }:
  { query: string; setQuery: (v:string)=>void; sort: string; setSort: (v:string)=>void; }) {
  return (
    <div className="flex flex-col md:flex-row gap-3">
      <Input placeholder="Search by name" value={query} onChange=
{e=>setQuery(e.target.value)} />
      <Select value={sort} onValueChange={setSort}>
        <SelectTrigger className="w-[220px]"><SelectValue placeholder="Sort" />
</SelectTrigger>
        <SelectContent>
          <SelectItem value="mcap">Market Cap</SelectItem>
          <SelectItem value="gainers">Top Gainers</SelectItem>
          <SelectItem value="losers">Top Losers</SelectItem>
        </SelectContent>
      </Select>
    </div>
  )
}
```

`app/page.tsx`

```tsx
"use client";
import { useEffect, useMemo, useState } from "react";
import { QueryClient, QueryClientProvider, useQuery } from "@tanstack/react-query";
import CoinRow from "./components/CoinRow";
import CoinChart from "./components/CoinChart";
import SearchAndFilters from "./components/SearchAndFilters";
import ThemeToggle from "./components/ThemeToggle";
import { Dialog, DialogContent, DialogHeader, DialogTitle } from "@/components/ui/dialog";
import { Card } from "@/components/ui/card";
const qc = new QueryClient();

function HomeImpl() {
  const [query, setQuery] = useState("");
  const [sort, setSort] = useState("mcap");
  const [selected, setSelected] = useState<any | null>(null);
  const [range, setRange] = useState<"24h"|"7d"|"30d"|"90d">("7d");

  const { data: markets } = useQuery({
    queryKey: ["markets", query],
    queryFn: async () => {
      const url = new URL("/api/markets", window.location.origin);
      url.searchParams.set("per_page", "100");
      if (query) url.searchParams.set("search", query);
      const r = await fetch(url);
      return r.json();
    },
    refetchInterval: 30000
  });

  const { data: vanry } = useQuery({
    queryKey: ["vanry-usdt"],
    queryFn: async () => (await fetch("/api/tickers/vanar-chain?target=USDT")).json(),
    refetchInterval: 60000
  });

  const list = useMemo(() => {
    if (!markets) return [];
    let rows = markets.slice();
    if (sort === "gainers") rows.sort((a:any,b:any)=> (b.change24h??0) - (a.change24h??0));
    else if (sort === "losers") rows.sort((a:any,b:any)=> (a.change24h??0) - (b.change24h??0));
    return rows;
  }, [markets, sort]);

  return (
    <div className="mx-auto max-w-5xl p-4 space-y-4">
      <header className="flex items-center justify-between">
        <h1 className="text-2xl font-semibold">Crypto Prices</h1>
        <ThemeToggle />
      </header>

      {/* pinned VANRY/USDT */}
      <Card className="p-4">
        <div className="text-sm text-muted-foreground">Pinned Pair</div>
        <div className="flex items-end justify-between">
          <div className="text-lg font-medium">VANRY/USDT {vanry?.exchange ? `@
${vanry.exchange}` : ""}</div>
          <div className="text-2xl font-bold">{vanry?.last ? `${vanry.last.toFixed(6)} USDT` : "—"}
</div>
        </div>
      </Card>
      <SearchAndFilters query={query} setQuery={setQuery} sort={sort} setSort={setSort} />

      <div className="grid gap-2">
        {list?.map((c:any) => (
          <CoinRow key={c.id} c={c} onClick={()=>setSelected(c)} />
        ))}
      </div>

      <Dialog open={!!selected} onOpenChange={(o)=>!o && setSelected(null)}>
        <DialogContent className="max-w-3xl">
          <DialogHeader>
            <DialogTitle>{selected?.name} ({selected?.symbol})</DialogTitle>
          </DialogHeader>
          <div className="flex gap-2 mb-2">
            {(["24h","7d","30d","90d"] as const).map(r => (
              <button key={r} onClick={()=>setRange(r)}
                className={`px-3 py-1 rounded border ${range===r? "bg-primary text-primary-
foreground":"hover:bg-accent"}`}>
                {r}
              </button>
            ))}
          </div>
          {selected && <CoinChart coinId={selected.id} range={range} />}
        </DialogContent>
      </Dialog>
    </div>
  );
}

export default function Home() {
  return <QueryClientProvider client={qc}><HomeImpl/></QueryClientProvider>;
}
```

config & env

`.env.local`

```ini
```

```
COINGECKO_API_KEY=your_demo_api_key_here
```

`tailwind.config.ts` (ensure shadcn preset added per its setup; the create-next-app template already wires Tailwind)

---

deployment

1. Push to GitHub, then **New Project** on Vercel and connect.
2. Add env var `COINGECKO_API_KEY` (Project Settings → Environment Variables).
3. Deploy.
    - You'll get `/api/*` FastAPI endpoints and a production URL.
    - Python runtime (beta) allows ASGI `app` —no `uvicorn` needed.

If Python gives trouble in your workspace (rare, but there are forum threads): make the UI a separate Vercel project, and deploy the same FastAPI to Render/Fly.io with CORS enabled; point the UI to that base URL. I'd still try the Python function first since Vercel officially documents it.

UX details that make it feel "pro"

- **Pinned VANRY/USDT** first, with exchange name and live last price (highest-volume USDT ticker, often Binance/Gate).
- **List polish:** subtle hover, rank, tiny 7-day sparkline (already available in the markets endpoint via `sparkline=true`), green/red % with accessible contrast.

- **Detail modal** with 24h/7d/30d/90d, chart tooltip showing price and timestamp.
- **Dark/light toggle** with next-themes.
- **Loading & empty states:** skeletons (shadcn "Skeleton" component) and friendly "no results".

---

rate limits & refresh

- Demo plan: 30 req/min, 10k/mo. With edge caching (`Vercel-CDN-Cache-Control`) and 30s/60s TTLs, you'll sit well below limits even with refetch intervals.

- CoinGecko's market endpoints are edge-cached on their side too; we're okay polling every 30s for the list and every 30–60s for the chart.

---

documenting AI help (drop this in `docs/AI_NOTES.md`)

```md
# AI Collaboration Log

## Prompts & Outcomes
- "Design a one-evening plan for a Vercel SPA with a FastAPI core and CoinGecko data."
  - Outcome: Stack choice (Next.js + shadcn + TanStack Query + Recharts), API surface (/api/markets, /api/coins/:id/chart, /api/tickers/:coin), caching strategy, and code scaffolding.
- "Write FastAPI endpoints with TTL cache and Vercel CDN caching headers."
  - Outcome: `api/index.py` with httpx + cachetools; normalized JSON shapes; `Vercel-CDN-Cache-Control` headers.
- "Build React components for list/search/modal chart."
  - Outcome: `CoinRow`, `SearchAndFilters`, `CoinChart`, `ThemeToggle`, `page.tsx` wiring.

## Manual Adjustments
- Pinned **VANRY/USDT** logic: filtered tickers by `target=USDT` and sorted by `volume` + `trust_score`.
- Tightened payload fields for faster loads.
- Adjusted cache TTLs after testing (30s list, 60s chart).
- UI: spacing, accessible colors, dialog sizing, range buttons.
```

---

small test checklist

- [ ] Homepage shows VANRY/USDT pinned with last price.
- [ ] Search returns "vanry" and "bitcoin" properly.
- [ ] Clicking a row opens modal with chart; range buttons swap series.
- [ ] Dark/light toggle persists.
- [ ] Network tab shows `Vercel-CDN-Cache-Control` and ~30s reuse.

---

references

- **Vercel Python runtime (ASGI app) + availability on all plans (beta):** Vercel docs.

- **Vercel cache headers & edge caching behavior:** Vercel docs.
- **CoinGecko API & Demo plan rate limits:** pricing/FAQ; and public post noting keyless deprecation.
- **Endpoints you'll call:** `coins/markets`, `coins/{id}/market_chart`, coin page/tickers.

---

if you want, i can split this into a ready-to-clone repo structure or add a mini sparkline in the list—just say the word.

Sources