# Frontend Developer- Technical task: Streaming Dashboard Clone

**Objective:** Build a simplified streaming service dashboard (like Netflix/Hulu) using **Next.js 14 App Router** and **TypeScript**, fetching data from a **public external API**. Candidates are expected to use **AI coding tools** to accelerate development, demonstrating effective prompt engineering and code integration.

**Time Limit:** 1 Day (6-8 hours estimated work).

---

## Phase 1: Setup and External API Integration

1. **Project Initialization:** Initialize a new Next.js project using **TypeScript** and **Tailwind CSS**. Initialize a Git repository for version control.
2. **Select Public API:** Choose a simple, free public API for movie/content data (e.g., TMDB). Obtain the necessary **API Key** (if required) and note the endpoints needed for listing content and fetching details by ID.
3. **Environment Variables:** Securely store your API Key in a local `.env.local` file. This key must be accessed only on the server side.
4. **TypeScript Typing:** Write the necessary **TypeScript interfaces** to define the expected structure of the external API response (e.g., for `Movie` objects with properties like `id`, `title`, and `posterUrl`) to ensure type safety.

---

## Phase 2: Homepage (`/`) Development and Server Components

5. **Main Layout and Header:** Design the main application layout in the root `app/page.tsx`. Create a fixed **Header/Navigation component** (can be a Client Component for interactivity) with a logo and navigation links.
6. **Server-Side Data Fetching:** In `app/page.tsx` (a **Server Component**), use the native **`fetch` API** to call your **external public API**. Use the secured **environment variable** for the API key. **Do not use `useState` or `useEffect` for this primary fetch.**
7. **Hero Banner:** Render the first fetched item as a large **Hero Banner** at the top. This component **must use the Next.js `<Image/>` component** with `fill` and `priority` props for optimization and layout stability.
8. **Movie Row Component:** Develop a reusable `<MovieRow movies={...} categoryTitle={...}/>` **Client Component**. This component will handle displaying a list of posters and enabling horizontal scrolling.

9. **Dashboard Rendering:** In `app/page.tsx`, filter the fetched movie data (e.g., by category or popularity) and pass the filtered results to at least three instances of your `<MovieRow/>` component.
10. **Link and Image Optimization:** Ensure every movie item/poster within the row uses the **Next.js `<Link/>` component** to enable navigation, and the **Next.js `<Image/>` component** for optimized poster loading.

---

## Phase 3: Dynamic Routing and Vercel Deployment

11. **Implement Dynamic Routing:** Create the necessary file structure for a dynamic route: **`app/movie/[id]/page.tsx`**.
12. **Detail Page Data Fetching:** In `app/movie/[id]/page.tsx` (a **Server Component**), retrieve the unique movie ID using the **`params`** prop. Use this ID to make a second, specific API call (to the external public API) to fetch the detailed information for that single movie.
13. **Render Detail Page:** Display the detailed movie information (title, description, large poster using `<Image/>`) on a responsive layout.
14. **Final Deployment:** Deploy the completed, working application to **Vercel**. During the Vercel setup, you must correctly configure the **API key** as a **Vercel Environment Variable** to ensure the production build works.
15. **Submission Documentation:** Create a file (e.g., `AI_Report.md`) detailing: a) The **AI tools** used (e.g., ChatGPT, Copilot). b) **Specific parts of the code** (e.g., Tailwind styling, complex utility functions, configuration setup) that were heavily reliant on AI-generated prompts. c) The **Vercel live URL** and the **GitHub repository link**.