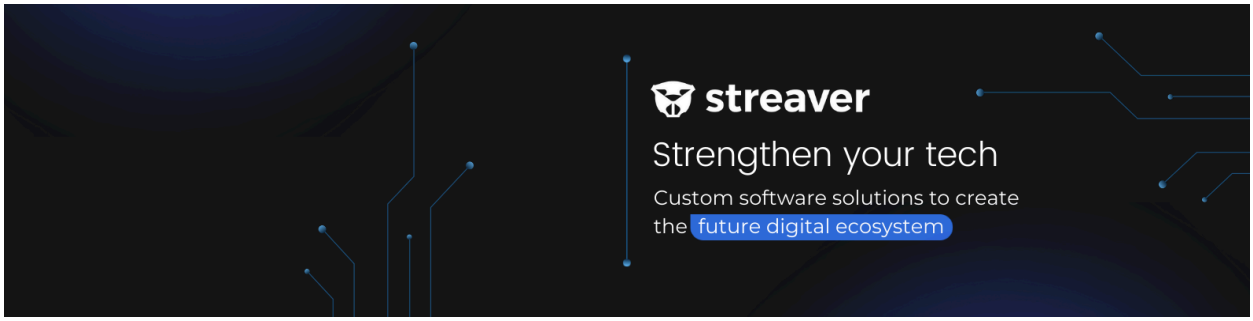


App details

We would like to build a web app using TypeScript (TS) for the frontend and the backend. Because of that, we have decided that a technology like NextJS seems like a perfect match for our needs. It provides a lot of functionality for developers and is a “production-ready” framework.

The application we are trying to build allows users to list posts from an API and allows the users to filter the posts by the **userId** of whoever wrote the post. The users of the application you are building usually travel a lot and are in places with bad or unstable internet connections and because of that we want to add some features that allow users to have a better experience.

Step 1 - Setup

Create an application using NextJS and upload it to a **public** repository in GitHub. If you prefer a private repository you need to request the usernames of the evaluation team. Ensure a file named **assumptions.md** is placed at the project's root to detail all presumed factors during development.

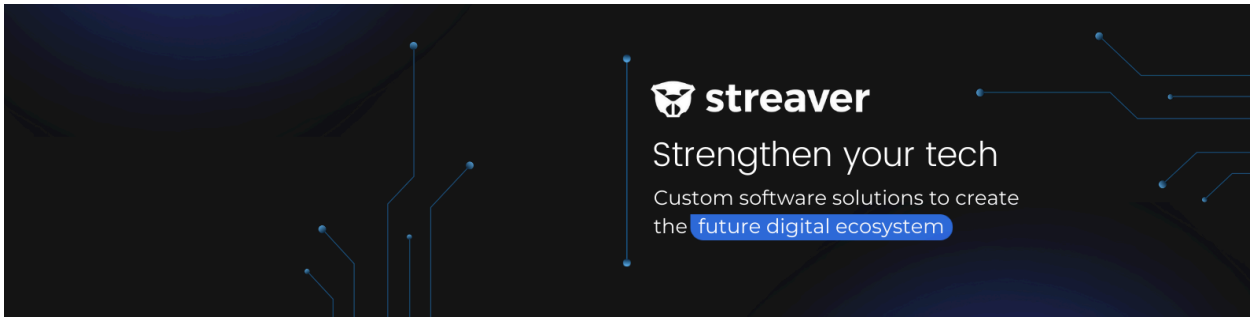
Step 2 - Deployment

Configure an account on Vercel to host and deploy the application in an automated fashion each time someone pushes changes to the main branch of the repository. Optionally it should also be configured to deploy each pull request targeting the main branch.

Step 3 - Listing

Create a /posts page that lists all the posts returned by the endpoint https://jsonplaceholder.typicode.com/posts (via an HTTP GET request). For the UI, use a list of “card” elements. You can see some examples at https://tailwindcss.com/ (but you don’t need to use TailwindCSS if you don’t want to)

We would also like to use the library SWR https://swr.vercel.app/ and set it up so that the requests are automatically re-executed after we regain connection to the internet after losing it or after an error happens.



Step 4 - Search

Each post has a *userId* attribute which can be filtered by. We would like to have an HTML input in the application that is used to filter the posts using the endpoint https://jsonplaceholder.typicode.com/posts?userId=1 (replace 1 by the input value).

For the search, we also want to use a debouncing mechanism to avoid hitting the API with each keypress; this will help prevent unnecessary requests. An example of such a mechanism would be https://github.com/vercel/swr/issues/110.

Step 5 - Slow connection notifications (optional)

The library SWR (React) provides a mechanism to handle slow connections. Investigate that option and use it to show some kind of message to the user that the requests are taking longer than expected. Once the listing or search requests finish, remove the notification.