hviana / **faster_react** Public

Tiny Full-Stack React framework. Avoid Overengineering. Automatic routes, reload and compo
combining SSR and CSR. 100% Deno, no Node dependencies. Fully compatible with Deno Dep

MIT license
96 stars 6 forks Branches Tags Activity

Star

Code Issues Pull requests Discussions Actions Projects Secu

识别到网页内正在下载...
来自当前网页的文件
https://kkgithub.com/hvia...
该文件可转存至百度网盘
转存地址：我的网盘/来自：浏览器扩展
不占用电脑空间 一键转存，秒速下载
暂不转存 转存至网盘
30天内不再提醒

main 1 Branch 1 Tag Go to fil

hviana Update deno.json ✓ ed6e13b · 3 weeks ago

.github/workflows	Update deno.yml	3 months ago
.vscode	send	3 months ago
app	update to react 19 and fix translations	last month
.gitignore	send	3 months ago
LICENSE	send	3 months ago
README.md	update to react 19 and fix translations	last month
deno.json	Update deno.json	3 weeks ago
main.ts	temp	last month
options.json	improve template example	2 months ago

50M

development stage stable tests pass

🚀 faster_react

Important

Please give a star! ⭐

🌟 Introduction

`faster_react` is a tiny Full-Stack React framework. He avoids Overengineering. This framework **uses its own RSC engine, combining SSR and CSR**, and automatically generates routes for React components. To utilize this, you must use the routes helper provided by the framework (React Router). The framework's configuration file is located at `options.json`.

🎯 What Does `faster_react` Do for You?

Focus solely on development! This framework handles:

- 🗺️ **Automatic route generation** for React components.
- 🔄 **Automatic inclusion** of new React components when `framework => "dev": true`.
- 📦 **Automatic frontend bundling** when `framework => "dev": true`.

- **Automatic browser reload** when `framework => "dev": true`.
- **Automatic frontend minification** when `framework => "dev": false`.
- **Automatic backend reload** when changes are detected and `framework => "dev": true`.
- **Automatic detection** of Deno Deploy environment. Test in other serverless environments .

> **Note:** The project includes a simple application example demonstrating each functionality. The
> optional. You can use whatever CSS framework you want.

About Faster

This framework uses a middleware library called Faster. Faster is an optimized middleware server v ,
built on top of native HTTP APIs with no dependencies. It includes a collection of useful middlewa

- Log file
- Serve static
- CORS
- Session
- Rate limit
- Token
- Body parsers
- Redirect
- Proxy
- Handle upload

Fully compatible with Deno Deploy and other enviroments. Examples of all resources are available in the README. Faster's ideology is simple: all you need is an optimized middleware manager; all other functionality is middleware.

Contents

Benchmarks

`faster_react` has only **0.9%** of the code quantity of Deno Fresh.

Benchmark Command:

```
# Deno Fresh
git clone https://kkgithub.com/denoland/fresh.git
cd fresh
```

```
git ls-files | xargs wc -l
# Output: 104132 (version 1.7.3)

# faster_react
git clone https://kkgithub.com/hviana/faster_react.git
cd faster_react
git ls-files | xargs wc -l
# Output: 1037 (version 20.1)
```

识别到网页内正在下载...

来自当前网页的文件
https://kkgithub.com/hvia...

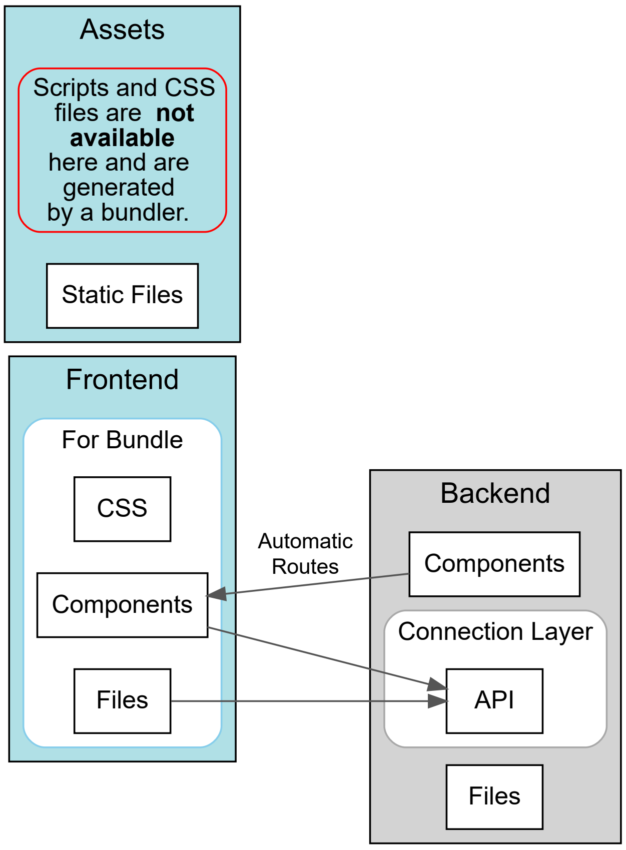
该文件可转存至百度网盘
转存地址：我的网盘/来自：浏览器扩展
不占用电脑空间 一键转存，秒速下载

30天内不再提醒

🏗️ Architecture

This framework utilizes **Headless Architecture** [1] to build the application, combined with the **Mic** routes in the backend.

> **Headless Architecture** provides complete freedom to the developer, reducing the learning curve. Despite this freedom, there is an **explicit separation between backend and frontend**, which aids in development.
>
> The **Middleware Design Pattern** offers a practical and straightforward method for defining API routes.



📂 App Structure

All application folders are inside the `app` folder.

📦 Get Deno Kv and Deno Kv Fs

On the backend, if a **Deno KV** instance is available, access instances via `Server.kv` and `Server.kvFs`:

```
import { Server } from "faster";
```

See **Deno KV** settings in `options.json`.

> **Deno KV File System (`Server.kvFs`)**: Compatible with Deno Deploy. Saves files in 64KB chunks. Organize files into directories, control the KB/s rate for saving and reading files, impose rate limits, set user space limits, and limit concurrent operations—useful for controlling uploads/downloads. Utilizes the Web Streams API.

More details: deno_kv_fs

Backend API

Imports: Import your backend libraries here.
Organization: Files can be organized into subdirectories.
File Extension: Use `.ts` files.
Structure: Flexible file and folder structure that doesn't influence anything.
Routing: Define routes using any pattern you prefer.
Exports: Must have a `default export` with a function (can be asynchronous).
Function Input: Receives an instance of `Server` from `faster`.
Usage: Perform backend manipulations here (e.g., fetching data from a database), including a
Routes: Define your custom API routes. For help, see: faster

Backend Components

Optionality: A backend component is optional for a frontend component.

Imports: Import your backend libraries here.

Organization: Organize files into subdirectories.

File Extension: Use `.ts` files.

Correspondence: Each file should have the same folder structure and name as the corresponding frontend component but with a `.ts` extension.

Example:
Frontend: `frontend/components/checkout/cart.tsx`
Backend: `backend/components/checkout/cart.ts`

Exports: Must have a `default export` with an object of type `BackendComponent`:

```
import { type BackendComponent } from "@helpers/backend/types.ts";
```

Usage: Intercept a frontend component request:

Before Processing (`before?: RouteFn[]`): List of middleware functions (see: faster). Use to check headers (`ctx.req.headers`) or search params (`ctx.url.searchParams`), like tokens, impose rate limits etc.

> **Note:** To cancel page processing, do not call `await next()` at the end of a middleware function.

README MIT license

After Processing (`after?: (props: JSONObject) => void | Promise<void>`): Function receives the `props` that will be passed to the component. Add backend data to these `props`, such as data from a database. Can be asynchronous.

> **Note:** Only use props data in JSON-like representation, or hydration will fail.

Backend Files

Imports: Import your backend libraries here.
Organization: Organize files into subdirectories.
File Extension: Use `.ts` files.
Usage: Free to make exports or calls (including asynchronous).
Purpose: Group common functions/objects for `backend/api`, `backend/components`, and other `backend/files`, such as user validations.

Frontend Components

Imports: Use only frontend libraries.
Organization: Organize files into subdirectories.
File Extension: Use `.tsx` files.
Rendering: Rendered on the server and hydrated on the client.

- **Routes Generated:** Two routes per file (e.g., `frontend/components/checkout/cart.tsx`):
  - **Page Route:** For rendering as a page, e.g., `/pages/checkout/cart`.
  - **Component Route:** For rendering as a component, e.g., `/components/checkout/cart`.
- **Initial Route (`/`):** Points to `frontend/components/index.tsx`.
- **Exports:** Must have a `default export` with the React Function/Component.
- **Props Passed to Component:**
  - Form-submitted data (or JSON POST).
  - URL search parameters (e.g., `/pages/myPage?a=1&b=2` results in `{a:1, b:2}`).
  - Manipulations from `backend/components`.

🎨 Frontend CSS

Application CSS style files.

- **Multiple Files:** Automatically compiled.
- **Organization:** Organize files into subdirectories.

📜 Frontend Files

- **Imports:** Use only frontend libraries.
- **Organization:** Organize files into subdirectories.
- **File Extensions:** Use `.ts` and `.js` files.
- **Usage:** Free to make exports or calls (including asynchronous).
- **Difference from Components:** Scripts are not automatically delivered to the client. They need to be imported by the `frontend/components`.
- **Purpose:** Group common functions/objects for React Functions/Components, like form field validations. Can have `frontend/files` common to other `frontend/files`.

🌎 Frontend Translations

- **File Extensions:** Use `.json` files.
- **Correspondence:** Each file should have the same folder structure and name as the corresponding frontend component but with a `.json` extension.
  - **Example:**
    - Frontend: `frontend/components/checkout/cart.tsx`
    - Backend: `frontend/translations/checkout/cart.json`
- **Usage:**

In `frontend/components/index.tsx`:

```
import {
  detectedLang,
  useTranslation,
} from "@helpers/frontend/translations.ts";
const Home = () => {
  const t = useTranslation();
  //Any .init parameter of i18next (minus ns) is valid in useTranslation.
  //Ex: useTranslation({ lng: ["es"], fallbackLng: "en" }) etc.
  //On the client side, the language is automatically detected (if you don't specify).
  //On the server, the language is "en" (if you don't specify).
  //The "en" is also the default fallbackLng.
  return (
    <div className="app-name">
      {t("index.appName", { endExample: "!" })}
    </div>
  );
};
export default Home;
```

In `frontend/translations/en/index.json`:

```
{
  "appName": "My SaaS App {{endExample}}"
}
```

The framework translation is just a wrapper over i18next. See the i18next documentation if you ha

Static

Files served statically. Routes are generated based on the folder and file structure.

Example: `localhost:8080/static/favicon.ico` matches `static/favicon.ico`.

React Router

Since the framework has its own routing system, a third-party routing library is unnecessary. Use the framework helper:

> **Note:** Direct form submissions for page routes path also work.

```
import { getJSON, route } from "@helpers/frontend/route.ts";
```

Interface Parameters:

```
interface Route {
  headers?: Record<string, string>; // When routing to a page, headers are encoded in the URL. Intercept them in ctx.url.search
  content?:
    | Record<any, any>
    | (() => Record<any, any> | Promise<Record<any, any>>);
  path: string;
  startLoad?: () => void | Promise<void>;
  endLoad?: () => void | Promise<void>;
  onError?: (e: Error) => void | Promise<void>;
  disableSSR?: boolean; //For component routes. Disables SSR; defaults to false.
  elSelector?: string; // Required for component routes.
  method?: string; // Only for API routes. Optional; defaults to GET or POST.
}
```

Examples

Navigating to a Page with Search Params:

```
// URL search params passed as properties to the page. Props receive `{a:1}`
<button onClick={route({ path: "/pages/test?a=1" })}>
  Go to Test Page
</button>;
```

Passing Additional Parameters:

```
// Props receive `{a:1, example:"exampleStr"}`
<button
  onClick={route({
    path: "/pages/test?a=1",
    content: { example: "exampleStr" },
  })}
>
  Go to Test Page with Extra Data
</button>;
```

Using Asynchronous Content:

```
// Props receive `{a:1, ...JSONResponse}`
<button
  onClick={route({
    path: "/pages/test?a=1",
    content: async () => {
      return await getJSON({
        path: "/example/json",
        content: {
          test: "testData",
        },
      });
    },
  })}
>
  Go to Test Page with Async Data
</button>;
```

Programmatic Routing:

```
(async () => {
  if (user.loggedIn) {
    await route({
      path: "/pages/dash",
      content: { userId: user.id, token: token },
    })();
  } else {
    await route({ path: "/pages/users/login" })();
  }
})();
```

Loading a Component:

```
<button
  onClick={route({
    path: "/components/parts/counter",
    elSelector: "#counter",
  })}
>
  Load Counter Component
</button>;
```

Making an API Call:

```
<button
  onClick={async () => {
    const res = await getJSON({
      path: "/example/json",
      content: {
        test: "testData",
      },
    });
    console.log(res);
    alert(JSON.stringify(res));
  }}
>
  Fetch JSON Data
</button>;
```

In the case of page routes, you can use this example to pass the URL parameters for the headers in the backend (if you really need it):

```
const signupBackendComponent: BackendComponent = {
  before: [
    async (ctx: Context, next: NextFunc) => {
      ctx.req = new Request(ctx.req, {
        headers: {
          ...Object.fromEntries(ctx.req.headers as any),
          "Authorization": `Bearer token ${ctx.url.searchParams.get("token")}`,
        },
      });
      await next();
    },
  ],
};
export default signupBackendComponent;
```

Forms submit for page routes work. For components, you can use the following:

```
<form
  method="POST"
  action=""
  encType="multipart/form-data"
  onSubmit={async (event) => {
      event.preventDefault();
      const data: any = new FormData(event.target as any);
      const formObject = Object.fromEntries(data.entries());
      await route({
        startLoad: () => setLoading(true), //useState
        endLoad: () => setLoading(false),
        path: "/components/register",
        elSelector: "#dash-content",
        content: formObject,
      })();
  }}
>
```

📦 Packages Included

Several packages are included to assist in developing React applications. Here are some examples of imports you can use without additional configuration:

```
import {/* your imports */} from "react";
import {/* your imports */} from "react/";
import {/* your imports */} from "i18next";
import {/* your imports */} from "react-dom";
import {/* your imports */} from "react-dom/server";
import {/* your imports */} from "react-dom/client";
import {/* your imports */} from "react/jsx-runtime";
s;
import {/* your imports */} from "@helpers/frontend/route.ts";
import {/* your imports */} from "@helpers/frontend/translations.ts";
import {/* your imports */} from "@helpers/backend/types.ts";
import {/* your imports */} from "faster";
import {/* your imports */} from "deno_kv_fs";
import {/* your imports */} from "jose"; //manage tokens
import { options, server } from "@core"; // Useful for accessing the server instance.
```

🛠️ Creating a Project

You can simply download this repository. Alternatively, use the command (requires `git` installed and configured):

```
deno run -A -r "https://deno.land/x/faster_react_core/new.ts" myProjectFolder
```

Customize and configure the server in `options.json`.

🚀 Running a Project

Execute the command:

Development:

```
deno task serve
```

Production:

```
deno serve main.ts #Add your permissions, port, certificate etc. see: https://docs.deno.com/runtime/reference/cli/serve
```

🌐 Deploy

Install Deployctl:

```
deno install -A --global jsr:@deno/deployctl
```

Deploy Your Project:

```
deployctl deploy
```

> **Note:** For production, set `framework => "dev": false` in `options.json`.

识别到网页内正在下载...

来自当前网页的文件
https://kkgithub.com/hvia...

该文件可转存至百度网盘
转存地址：我的网盘/来自：浏览器扩展
不占用电脑空间 一键转存，秒速下载

30天内不再提醒

📖 References

[1] Dragana Markovic, Milic Scekic, Alessio Bucaioni, and Antonio Cicchetti. 2022. *Could Jamstack Be the Future of Web Applications Architecture? An Empirical Study.* In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing* (SAC '22). Association for Computing Machinery, New York, NY, USA, 1872–1881. DOI: 10.1145/3477314.3506991

[2] Brown, Ethan. *Web Development with Node and Express: Leveraging the JavaScript Stack*. O'Reilly Media, 2019. URL: http://www.oreilly.com/catalog/9781492053484

🧑‍💻 About

Releases 1

v20.7 Latest
3 weeks ago

Packages

No packages published

Languages

● **TypeScript** 99.7% ● **CSS** 0.3%