Start

What is Quality API?

Introduction to Quality API

Quality API is a powerful library for building type-safe, extensible Next.js APIs. It seamlessly integrates with popular TypeScript libraries like Zod and Next Auth, preserving full type safety across your entire application.

Set up once to establish a robust foundation, then build anything on top of it.

Small and lightweight

Quality API has just one dependency: Next.js.

Its sole purpose is to simplify the complexities of writing Next.js endpoints, letting developers focus more on their project - not the codebase.

Example of endpoint

```
import QualityApi from "@quality-api/core";

export const POST =
  QualityApi.initBuilder()
    .handle(request => {
      return Response.json({ yourRequestBody: request.body });
    });
```

Quick start

Installation

To install Quality API, simply run the following command in your Next.js project's root folder:

npm:

```
npm install @quality-api/core
```

Yarn:

```
yarn add @quality-api/core
```

pnpm:

```
pnpm install @quality-api/core
```

... And that's it! You're now all good to go.

Your first endpoint and middleware

When building endpoints with Quality API, you're using the *builder pattern*, which is common in languages like Rust.

Each middleware is defined in chronological order, meaning the middleware defined first, is executed first when calling the endpoint, naturally.

Here is an example of an endpoint fetching a list of users with filters via URL search parameters, using Zod.

```
import QualityApi from "@quality-api/core";
import Db from "@/db";
import z from "zod".

const searchParamsSchema = z.object({
```

```
  searchText: z.string().optional().default(null)
});

const validateSearchParams = QualityApi.createMiddleware(async r => {
  const parseResult = await searchParamsSchema.safeParseAsync(r.body);

  if (parseResult.success) {
    return r.transformBody(parseResult.data);
  }
  else {
    return Response.json({ error: "Invalid search params" }, { status: 400 });
  }
});

export const GET =
  QualityApi.initBuilder()
    .mw(validateSearchParams)
    .handle(request => {

      const users = await Db.query(`
        SELECT *
        FROM users
        WHERE $1 IS NULL OR $1 IN name
      `, [request.url.searchParams.searchText])

      return Response.json(users);
    });
```

In this example, the request's search params are validated before the request is handled.

When using `request.url.searchParams`, it's correctly type-annotated to the Zod schema.

Usually, it's recommended separating middleware and schemas into separate files, but this is all up to you - naturally, it does not affect the usability.