a4arpon /
deno-back-end-starterkit

Code Issues Pull requests Actions Security Insights

This repository has been archived by the owner on Nov 3, 2024. It is now read-only.

Ready to go Deno Starter Kit for back-end web server development.

deno-hono-pono.deno.dev/

10 stars **1** fork **1** watching Branches Activity
Tags

Public archive repository

60M

2 Branches **0** Tags Go to file t Go to file Code

a4arpon Merge pull request #7 from a4arpon/dev-wayne

a8f76d9 · 6 months ago

.vscode	v1 router modified	6 months ago
.zed	routes added	6 months ago
src	stable api simplified	6 months ago
.env.example	env loader modified and ...	6 months ago
.gitignore	Git Init	7 months ago
Dockerfile	routes added	6 months ago
README.md	Hash modifier added	6 months ago
deno.json	Hash modifier added	6 months ago
deno.lock	readme added	6 months ago

Deno Back-End Starter Kit

This repository is a ready-to-use starter kit for Deno, offering a well-organized and structured API setup similar to what you might expect from a Node.js environment. It includes configurations for essential packages such as Hono, Mongoose, and jsonwebtoken.

Important Note

This starter kit is not intended for newcomers. It is recommended for developers who have advanced experience with Node.js and TypeScript, and who understand the Deno architecture. If you are new to these technologies, it is advisable to first familiarize yourself with the basics before using this starter kit.

Author

Name: Mr. Wayne

Website: a4arpon.me

GitHub: a4arpon

LinkedIn: a4arpon

Facebook: a4arpon

Packages Used

Deno

Hono

Mongoose

jsonwebtoken

Why Use This Starter Kit?

This starter kit solves the problem of having an unstructured and disorganized API setup in Deno. Unlike Node.js, there isn't a wealth of well-organized starter templates for Deno. This kit provides:

A structured folder organization

Pre-configured middlewares for logging, CORS, and security

A ready-to-use database connection setup with Mongoose

Standardized response and error handling mechanisms

Flexibility to add plugins and change database drivers as needed

What's Next?

I'm willing to include these things in the Starter-Kit in future

- Deno KV
- Deno Corn
- Mail System
- Authorization Guards

Folder Structure

```
.
├── deno.json            # Deno configuration file
├── deno.lock            # Lock file for dependencies
├── README.md            # Project documentation
└── src
    ├── config           # Configuration files
    │   └── helmet.config.ts
    ├── main.ts          # Main application entry point
    ├── models           # Database models
    │   └── user.model.ts
    ├── routes           # API route definitions
    │   ├── router.ts
    │   └── stable.routes.ts
    ├── middlewares      # Application guards and middlewares
    ├── services         # Business logic and services
    │   └── users.services.ts
    ├── types            # Type definitions
    │   └── shared.types.ts
    └── utils            # Utility functions
        ├── async.ts
        └── response.ts
```

.env File Sample

```
MONGODB_URL=""
```

Key Components

Main Application Setup (`main.ts`)

- **Environment Loader**: Loads environment variables.
- **Middlewares**: Sets up logging, CORS, and security headers.

Database Connection: Connects to MongoDB using Mongoose.

Router Setup: Defines base routes and integrates with the router.

Router (`router.ts`)

Stable Routes: Handles stable API routes.

Beta Routes: Placeholder for beta routes, currently returns a not active message.

Stable Routes (`stable.routes.ts`)

Defines the routes for user-related operations.

User Services (`user.services.ts`)

Contains the business logic for user operations, such as creating a user and retrieving users.

Utility Functions

Async Handler (`async.ts`): Safely handles async operations to catch errors.

```
export function safeAsync(func) {
  return async (context) => {
    try {
      return await func(context)
    } catch (error) {
      return exception(
        context,
        HTTPStatus.InternalServerError,
        "Internal Server Error",
      )
    }
  }
}
```

Explanation: This function wraps asynchronous operations to handle errors gracefully. It catches any errors thrown within the wrapped function and returns a standardized error response.

Response (`response.ts`): Standardizes JSON responses.

```
export function response(context, message, data, extra, success) {
  return context.json({
    success: success || true,
    message,
    ...extra,
```

```
    data,
  })
}
```

README

Exception (`response.ts`): Handles exceptions and sends error responses.

```
export function exception(context, status, error) {
  if (status && error instanceof Error) {
    context.status(status)
    return context.json({
      success: false,
      status: status,
      message: error.message,
    })
  }
  context.status(500)
  return context.json({
    success: false,
    status: 500,
    message: "Internal Server Error",
  })
}
```

Explanation: This function handles exceptions and sends a structured error response. It sets the HTTP status code and returns an error message.

Customization

Adding Plugins

You can easily add plugins to this starter kit from the [Hono website](). Follow the instructions on the website to integrate additional functionalities as per your requirements.

Changing Database Driver

The starter kit uses Mongoose for MongoDB. However, you can replace it with any other database driver by updating the database connection logic in `main.ts` and adjusting the models and services accordingly.

Getting Started

Clone the repository:

```
git clone https://github.com/a4arpon/deno-starter-kit.git
```

Install Deno: Follow the instructions on the [official Deno website](#).

Run the application:

```
deno task start
```

Contribution

Deployments 21

- **Production** 6 months ago
- **Preview** 6 months ago

+ 19 deployments

Languages

- **TypeScript** 100.0%