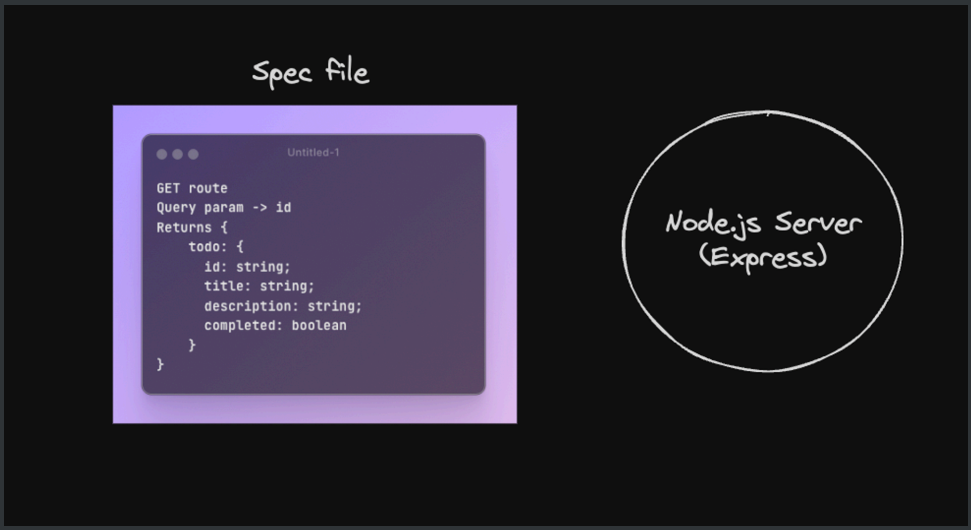
Why OpenAPI Spec

When you create backend, it's very hard for other people to know the exact `shape` of your routes

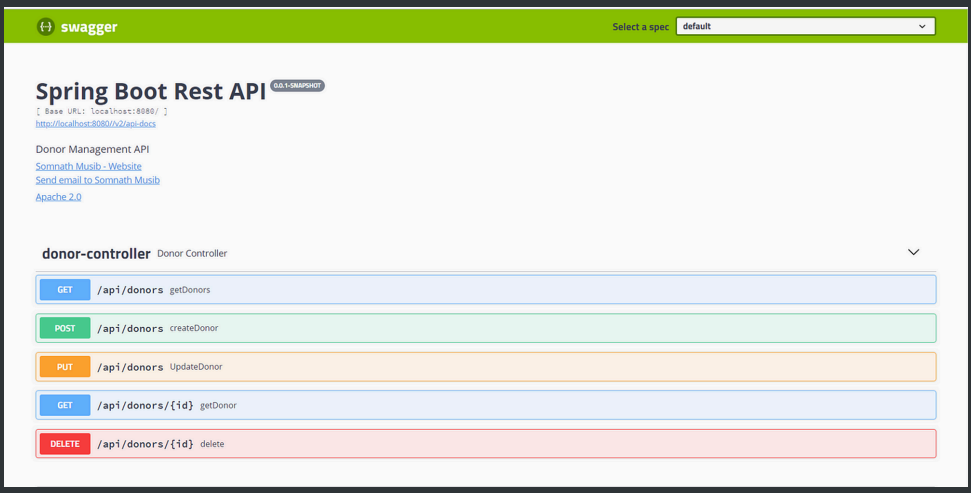
Wouldn't it be nice if you could describe, in a single file the `shape` of your routes?

For example - https://sum-server.100xdevs.com/todo?id=1



If you have this single long file that lists all your routes, you could

1. Auto generate documentation pages (Ref https://binance-docs.github.io/apidocs/spot/en/#query-current-order-count-usage-trade)
2. Auto generate `clients` in various languages (Java, JS, Go...)
3. Let the world look at your API routes shape without actually opening your code
4. Let AIs know how to `hit` your APIs in a single file, without sharing your code with the AI



What is the OpenAPI Spec

The OpenAPI Specification (OAS) is a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of a service without access to source code, additional documentation, or network traffic inspection. When properly defined via OpenAPI, a consumer can understand and interact with the remote service with minimal implementation logic.

Developed initially by Swagger, and later donated to the OpenAPI Initiative under the Linux Foundation, the OpenAPI Specification has become a widely adopted industry standard for defining and using APIs.

Good reference file - https://github.com/knadh/listmonk/blob/1bf7e362bf6bee23e5e2e15f8c7cf12e23860df6/docs/swagger/collections.yaml

Parts of the spec file

For a simple server

server.js

```
import express from 'express';

const app = express();
const port = 3000;

app.use(express.json());

let users = [
    { id: 1, name: 'John Doe' },
    { id: 2, name: 'Jane Doe' }
];

app.get('/users', (req, res) => {
    const { name } = req.query;

    if (name) {
        const filteredUsers = users.filter(user => user.name.toLowerCase().includes(
        res.json(filteredUsers);
    } else {
        res.json(users);
    }
});

app.listen(port, () => {
    console.log(`Server running on http://localhost:${port}`);
});
```

OpenAPI Spec

```
openapi: 3.0.0
info:
  title: User API
  description: API to manage users
  version: "1.0.0"
servers:
  - url: http://localhost:3000
paths:
  /users:
    get:
      summary: Get a list of users
      description: Retrieves a list of users, optionally filtered by name.
      parameters:
        - in: query
          name: name
          schema:
            type: string
          required: false
          description: Name filter for user lookup.
      responses:
        '200':
          description: A list of users
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/User'
components:
  schemas:
```

```yaml
    User:
      type: object
      properties:
        id:
          type: integer
          format: int64
          description: The unique identifier of the user.
        name:
          type: string
          description: The name of the user.
      required:
        - id
        - name
```

Try visiting

```
http://localhost:3000/users?name=John Doe,a,http://localhost:3000/users?name=John%20[
```

Copy

How to create a spec

1. Write it by hand (bad, but still happens)
2. Auto generate it from your code
    1. Easy in languages that have deep types like Rust
    2. Slightly harder in languages like Go/Rust
    3. Node.js has some libraries/codebases that let you do it
        1. With express - https://www.npmjs.com/package/express-openapi (highly verbose)
        2. Without express - https://github.com/lukeautry/tsoa (Cohort 1 video)
    4. Hono has a native implementation with zod - https://hono.dev/snippets/zod-openapi

We'll be going through `d`, but we've covered `c.ii` in Cohort 1

Hono + Zod + OpenAPI

Ref https://hono.dev/snippets/zod-openapi

Copy

```ts
import { z } from '@hono/zod-openapi'
import { createRoute } from '@hono/zod-openapi'
import { OpenAPIHono } from '@hono/zod-openapi'

const ParamsSchema = z.object({
  id: z
    .string()
    .min(3)
    .openapi({
      param: {
        name: 'id',
        in: 'path',
      },
      example: '1212121',
    }),
})

const UserSchema = z
  .object({
    id: z.string().openapi({
      example: '123',
    }),
    name: z.string().openapi({
      example: 'John Doe',
    }),
    age: z.number().openapi({
      example: 42,
    }),
  })
  .openapi('User')
```

```ts
const route = createRoute({
  method: 'get',
  path: '/users/{id}',
  request: {
    params: ParamsSchema,
  },
  responses: {
    200: {
      content: {
        'application/json': {
          schema: UserSchema,
        },
      },
      description: 'Retrieve the user',
    },
  },
})

const app = new OpenAPIHono()

app.openapi(route, (c) => {
  const { id } = c.req.valid('param')
  return c.json({
    id,
    age: 20,
    name: 'Ultra-man',
  })
})

// The OpenAPI documentation will be available at /doc
app.doc('/doc', {
  openapi: '3.0.0',
  info: {
    version: '1.0.0',
    title: 'My API',
  },
})

export default app
```

Try running the app locally and visiting

http://localhost:8787/users/123123

http://localhost:8787/doc

Create a swagger page

Given the OpenAPI Spec, you can create a swagger page for your app

https://hono.dev/snippets/swagger-ui

```ts
app.get('/ui', swaggerUI({ url: '/doc' }))
```

Copy

Try visiting http://localhost:8787/ui

Auto generated clients

Given you have a yaml/json file that describes the `shape` of your routes, lets try generating a `ts` client that we can use in a `Node.js` / `React` app to talk to the backend

Ref https://www.npmjs.com/package/openapi-typescript-codegen

1. Store the OpenAPI Spec in a file (spec.json)

```
{
  "openapi": "3.0.0",
  "info": {
    "version": "1.0.0",
```

Copy

```
    "title": "My API"
  },
  "components": {
    "schemas": {
      "User": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string",
            "example": "123"
          },
          "name": {
            "type": "string",
            "example": "John Doe"
          },
          "age": {
            "type": "number",
            "example": 42
          }
        },
        "required": [
          "id",
          "name",
          "age"
        ]
      }
    },
    "parameters": {

    }
  },
  "paths": {
    "/users/{id}": {
      "get": {
        "parameters": [
          {
            "schema": {
              "type": "string",
              "minLength": 3,
              "example": "1212121"
            },
            "required": true,
            "name": "id",
            "in": "path"
          }
        ],
        "responses": {
          "200": {
            "description": "Retrieve the user",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/User"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

2. Generate the client

```
npx openapi-typescript-codegen --input ./spec.json --output ./generated
```

Copy

1. Explore the client

```
cd generated
cat index.ts
```

Copy

1. Use it in a different project