Nuxt Fundamentals

Server sided routes

Peter Kassenaar –
info@kassenaar.com

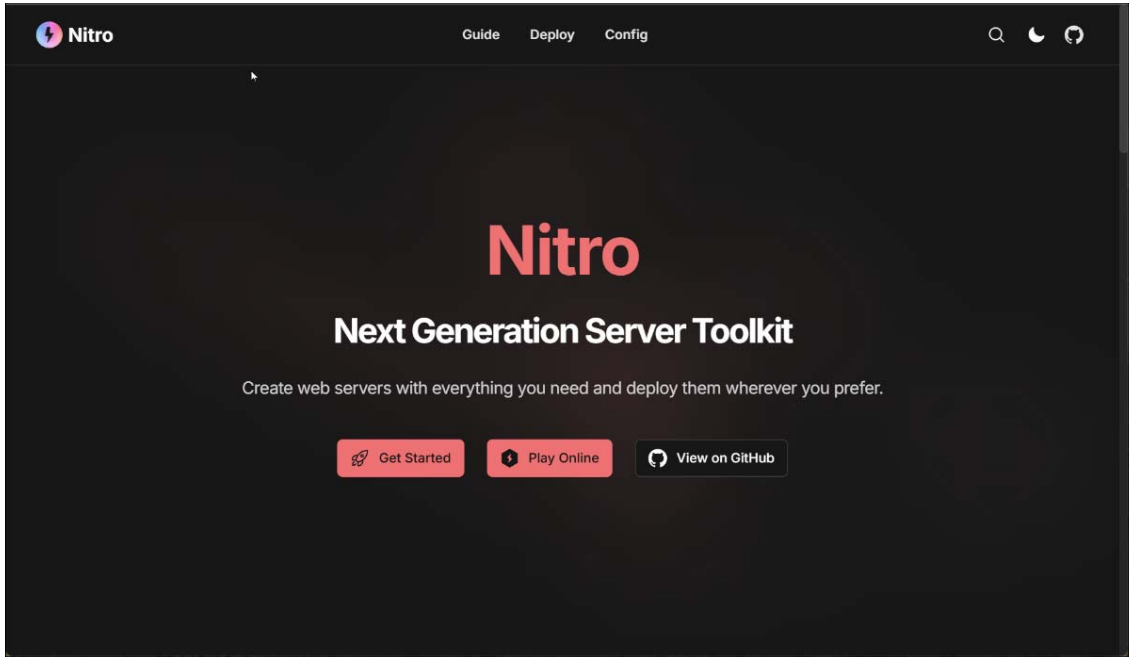
Server Sided routes

What are Server Routes and why should you use them?

Can’t we just communicate via `fetch()` / `useFetch()`?

What are Server Routes?

- Nuxt uses Nitro as its (development) server
  - You can compare this with Node + Express, Nginx, etc.
  - Which means you can define endpoints to be called from the frontend (e.g. your Nuxt application)
  - https://nitro.build/
- There is a special folder ./server in your application to host server routes
- (sub)folders and files inside this folder become endpoints that you can call from Vue


Nitro
Guide
Deploy
Config
Nitro
Next Generation Server Toolkit
Create web servers with everything you need and deploy them wherever you prefer.
Get Started
Play Online
View on GitHub

Why would you use Server Routes?

- After all, we can already use `fetch()` and `useFetch()` in our components, right?
- YES. But:
  - All URLs + params are exposed to the outside world, this way.
  - We might not want to expose private API keys
  - We might want to communicate with a server not supporting CORS
- Solution: use the built-in Nitro server
  - Do your calls in a server route, using `$fetch()` to create a server-to-server call
  - These are NOT exposed to the frontend
  - Actually, we're using the built-in server as some kind of proxy server

Creating a server route

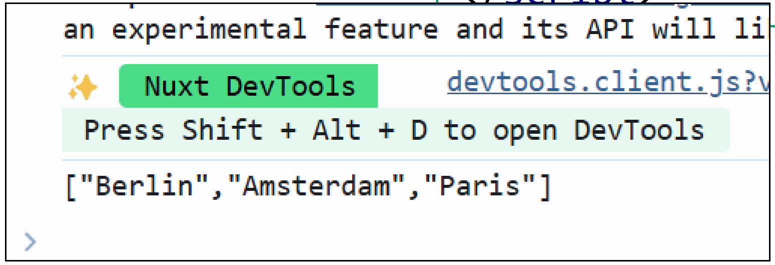
- Create a folder ./server/api
  - Naming your folder /api is convention, not mandatory
- For instance:
  - A file like ./server/api/cities.ts, becomes
  - fetch('/api/cities') in the Vue component
  - All JavaScript code inside cities.js will NOT be readable from the browser

Server route

```
// cities.ts
export default defineEventHandler(() => {
    return [
        'Berlin', 'Amsterdam', 'Paris'
    ]
})
```

Component

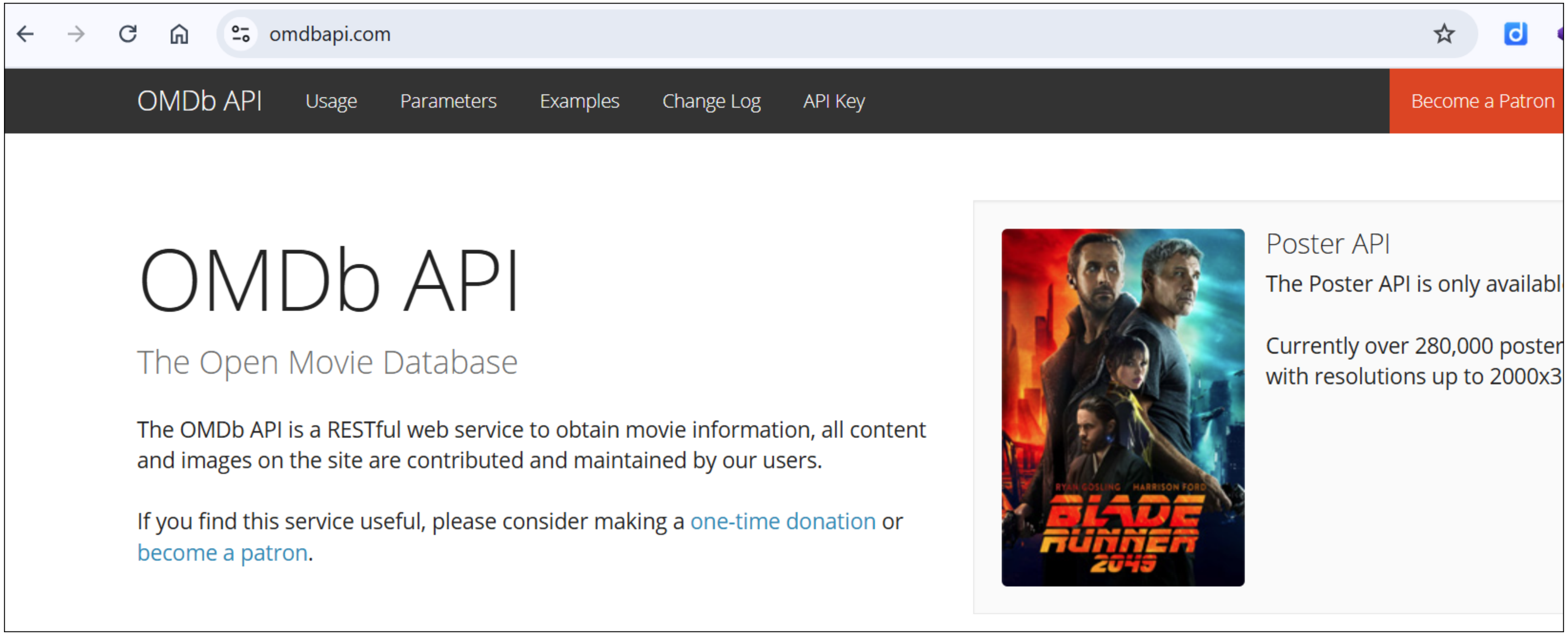
```
<script setup lang="ts">
  const { data: cities, pending, error } =
              await useFetch('/api/cities');
  console.log(JSON.stringify(cities.value))
</script>
```



```
an experimental feature and its API will li
✨ Nuxt DevTools          devtools.client.js?v
 Press Shift + Alt + D to open DevTools
["Berlin","Amsterdam","Paris"]
```

Browser DevTools

Example – using a private key


omdbapi.com

OMDb API

The Open Movie Database

The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

If you find this service useful, please consider making a one-time donation or become a patron.

Poster API

The Poster API is only availabl

Currently over 280,000 poster
with resolutions up to 2000x3

https://www.omdbapi.com/

Fetching movie information from OMDb API

- Situation:
  - I signed up for a private key
  - I don’t want to expose this key to the outer world
- Solution: create a server route like `./api/movies`
- Inside `defineEventHandler(() => { …} )`:
  - We want to be able to search for a movie name,
  - handle *query params* using the event parameter
- First: log results to the console
- Later: update the UI to search and show the movies.

1. Fetching details from the API

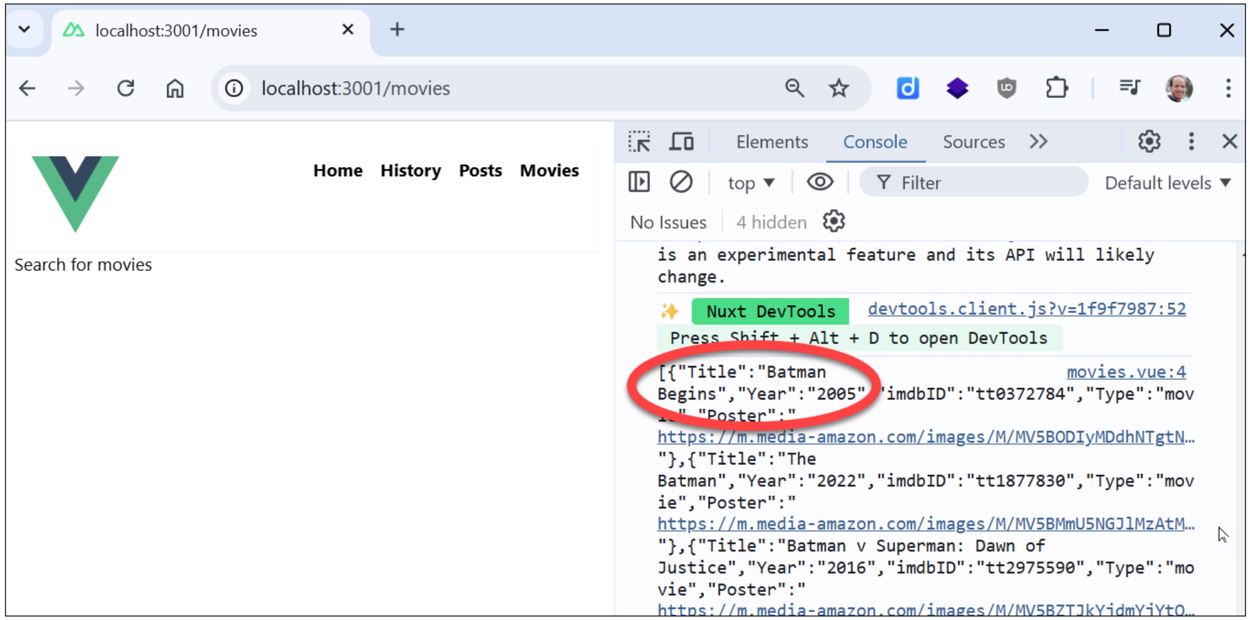
```
// movies.ts
export default defineEventHandler(async (event) => {
    const {name} = getQuery(event)
    const apiKey = 'f1f56c8e'; // my private key. Don't expose this to outside world!
    const url = `https://www.omdbapi.com/?apikey=${apiKey}&s=${name}`
    const result = await $fetch(url)
    return result['Search']
})
```

1. Get the movie name from the url, using `getQuery(event)`
2. Get the `apiKey` from the API, after signing up
3. Mix the name and apiKey in the final URL, *retrieved from API documentation*!
4. Use `$fetch(url)` in backend/server routes!
5. Return the results to the frontend

2. Calling the server route from a component

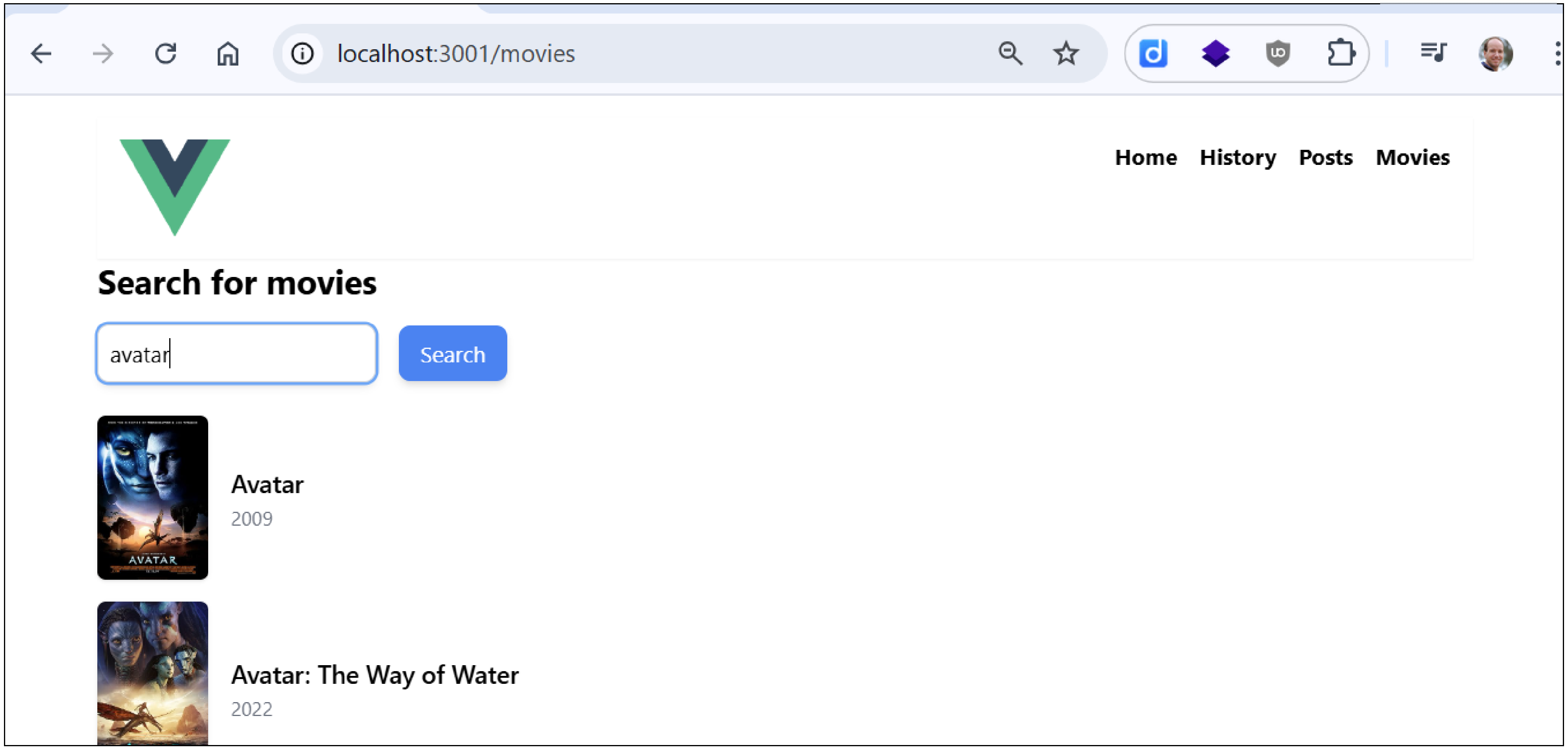
For instance, using the hardcoded name batman

```
<!--Movies.vue-->
<script setup lang="ts">
const {data : movies} = await useFetch('/api/movies?name=batman')
console.log(JSON.stringify(movies.value));
</script>
```



```
is an experimental feature and its API will likely
change.
Nuxt DevTools  devtools.client.js?v=1f9f7987:52
Press Shift + Alt + D to open DevTools
[{"Title":"Batman                            movies.vue:4
Begins","Year":"2005","imdbID":"tt0372784","Type":"mov
ie","Poster":"
https://m.media-amazon.com/images/M/MV5BODIyMDdhNTgtN…
"},{"Title":"The
Batman","Year":"2022","imdbID":"tt1877830","Type":"mov
ie","Poster":"
https://m.media-amazon.com/images/M/MV5BMmU5NGJlMzAtM…
"},{"Title":"Batman v Superman: Dawn of
Justice","Year":"2016","imdbID":"tt2975590","Type":"mo
vie","Poster":"
https://m.media-amazon.com/images/M/MV5BZTJkYjdmYjYtO…
```

3. Finalizing the UI to search for movies


localhost:3001/movies

Home History Posts Movies

Search for movies

avatar

Search

Avatar
2009

Avatar: The Way of Water
2022

Using `v-model`, and some other `ref()`’s and HTML-tags here to format movie data

Updating HTML

- Just adding a bunch of Tailwind CSS-classes
  - See `movies.vue` for details

```
<template>
<h2 class="text-2xl font-bold mb-4">Search for movies</h2>
  <div class="flex gap-4 items-center mb-6">
    <input
        type="text"
        v-model="movieName"
        @keyup.enter="searchMovies"
        placeholder="Enter movie name"
        class="p-2 border border-gray-300 …"
    >
    <button
        @click="searchMovies"
        class="bg-blue-500 text-white px-4 …"
    >
…
</template>
```

Logic

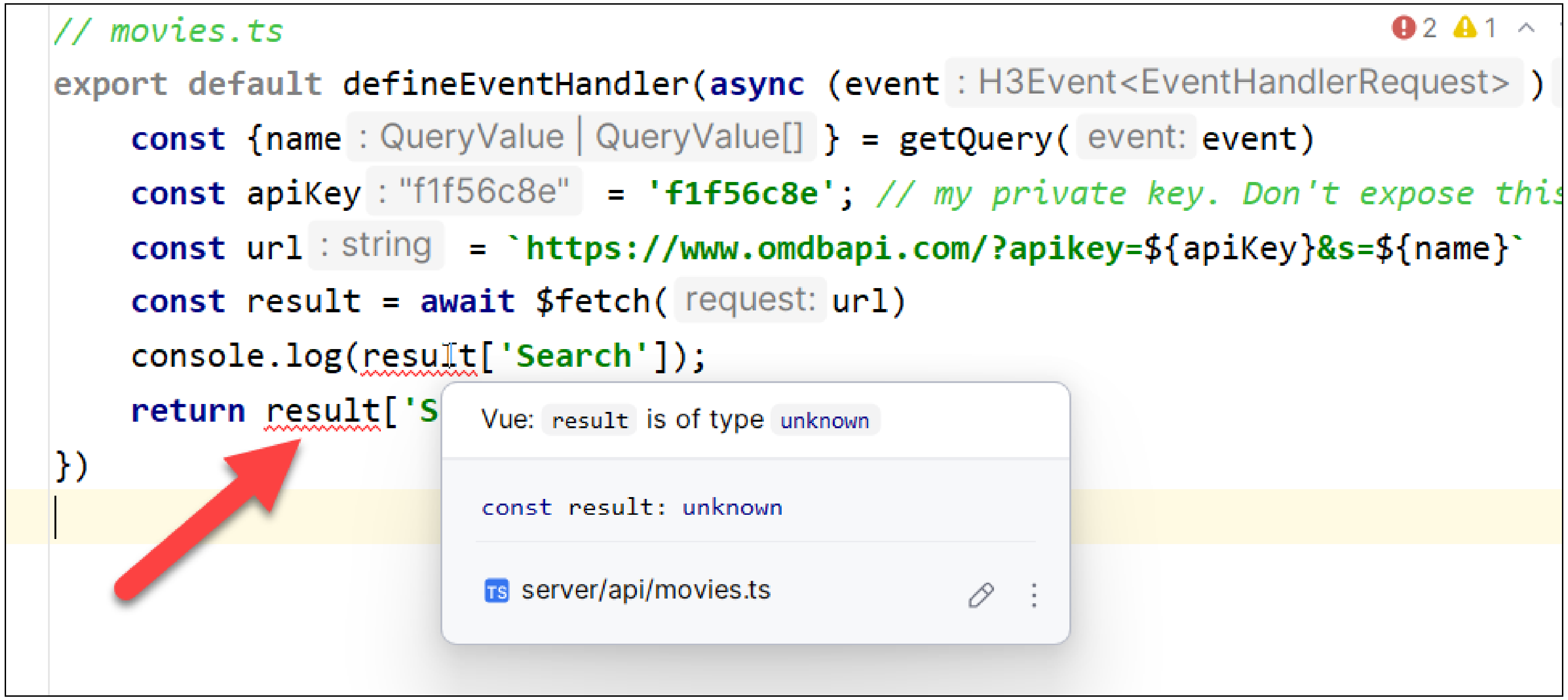
- Update the TypeScript by adding variables
  - movieName → coming from the textbox
  - movies → an array with resulting movies
  - searchMovies() → the function that calls the server route

```
<script setup lang="ts">
// variables
const movieName = ref('')
const movies = ref<any>([])

// function to search for movies. See also ./api/movies.ts
const searchMovies = async () => {
  try {
    movies.value = await $fetch(`/api/movies`, {
      params: {name: movieName.value}
    });
    movieName.value = ''; // reset movie name
    console.log(JSON.stringify(movies.value)) // debugging - may be removed!
  } catch (error) {
    console.error('An error occurred while fetching movies:', error)
  }
}
</script>
```

4. Satisfying the IDE by using interfaces

- Because we use TypeScript, the server route is complaining about unknown types



```
// movies.ts
export default defineEventHandler(async (event : H3Event<EventHandlerRequest> )
    const {name : QueryValue | QueryValue[] } = getQuery( event: event)
    const apiKey : "f1f56c8e"  = 'f1f56c8e'; // my private key. Don't expose this
    const url : string  = `https://www.omdbapi.com/?apikey=${apiKey}&s=${name}`
    const result = await $fetch( request: url)
    console.log(result['Search']);
    return result['S
})
```

Vue: `result` is of type `unknown`

```
const result: unknown
```

server/api/movies.ts

Solution: create interface for result

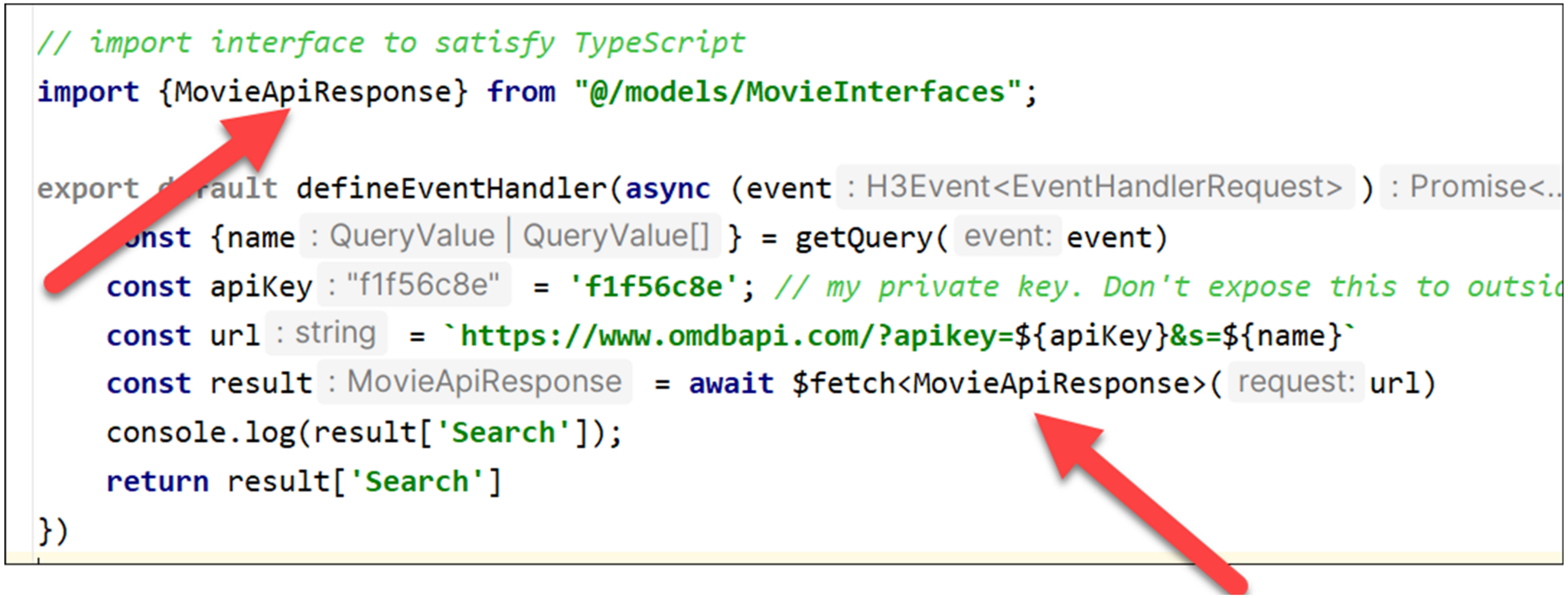
- NOT necessary when using plain JavaScript
- Always think of the response of YOUR API!
- Convention: store TypeScript interfaces in a `./models` directory

```typescript
// Type results according to your API
export interface Movie {
    Title: string;
    Year: string;
    imdbID: string;
    Type: string;
    Poster: string;
}

export interface MovieApiResponse {
    Search: Movie[];
    totalResults: string;
    Response: string;
}
```

Use the interfaces in ./api/movies.ts

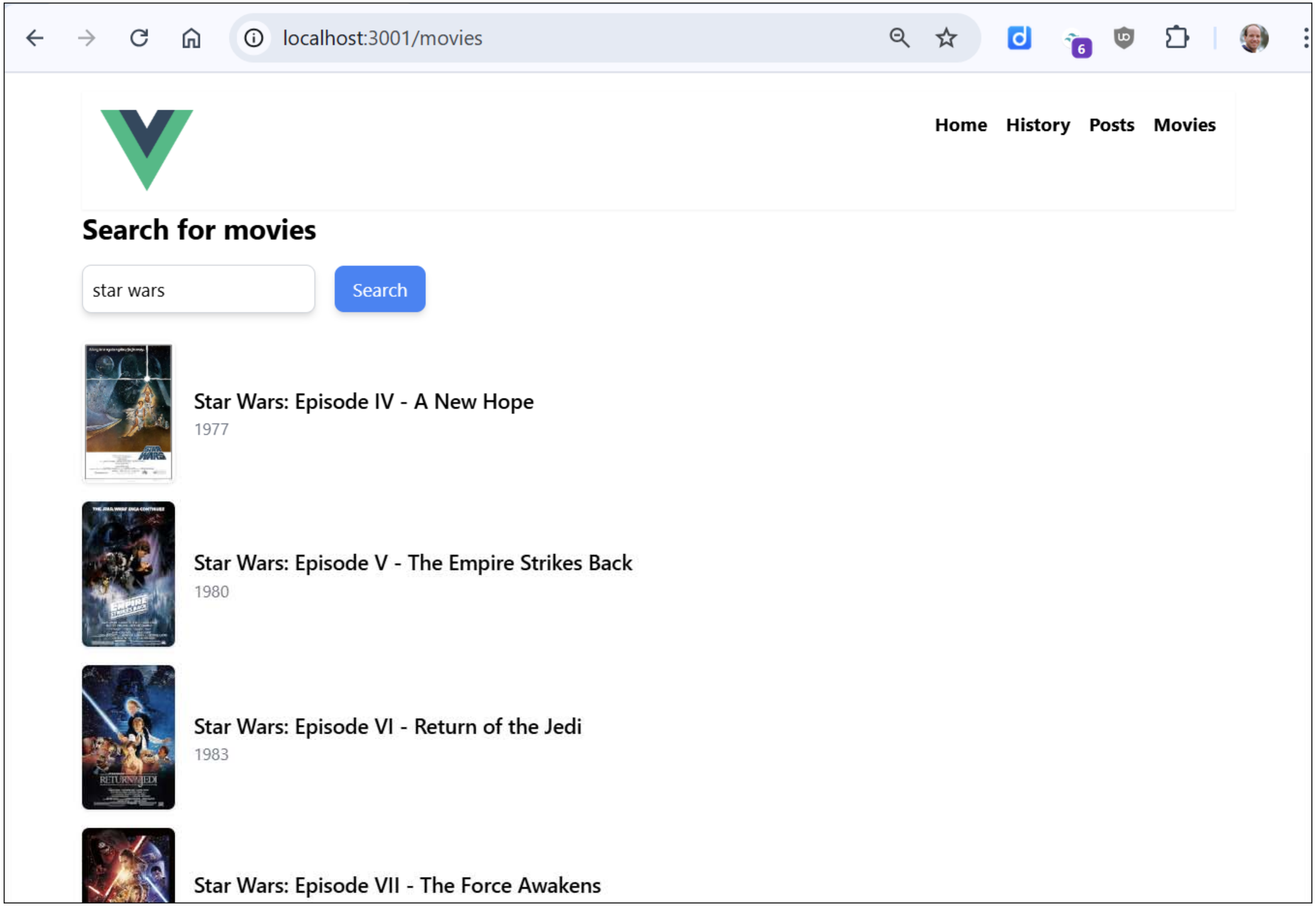
- Import the interface and use it in your server route
- Use it as a Generic Type: $fetch<T>(…)
- You can also use it in your frontend components



```
// import interface to satisfy TypeScript
import {MovieApiResponse} from "@/models/MovieInterfaces";

export default defineEventHandler(async (event : H3Event<EventHandlerRequest> ) : Promise<..
    const {name : QueryValue | QueryValue[] } = getQuery( event: event)
    const apiKey : "f1f56c8e"  = 'f1f56c8e'; // my private key. Don't expose this to outsid
    const url : string  = `https://www.omdbapi.com/?apikey=${apiKey}&s=${name}`
    const result : MovieApiResponse  = await $fetch<MovieApiResponse>( request: url)
    console.log(result['Search']);
    return result['Search']
})
```

Result – for instance


localhost:3001/movies

Home History Posts Movies

Search for movies

star wars

Search

Star Wars: Episode IV - A New Hope
1977

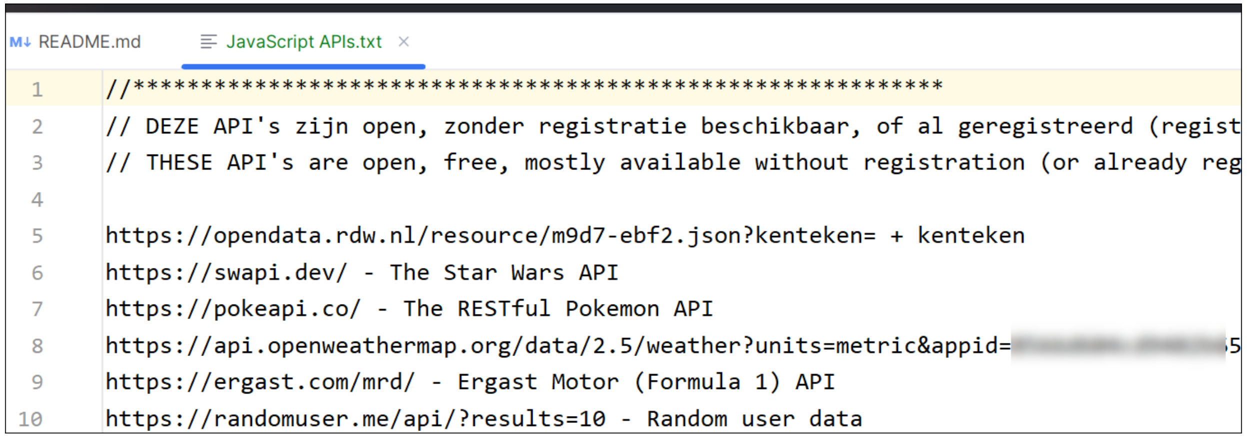
Star Wars: Episode V - The Empire Strikes Back
1980

Star Wars: Episode VI - Return of the Jedi
1983

Star Wars: Episode VII - The Force Awakens

Workshop #1

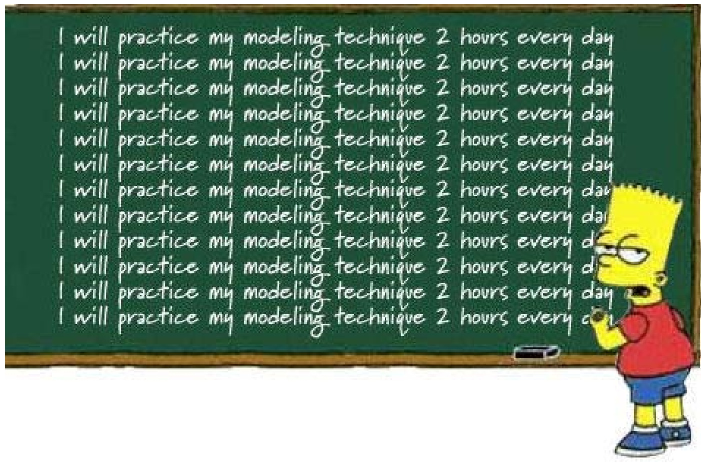
- Create a small application using server routes, using one of the API's available in `./JavaScript APIs.txt`.
- Note: NOT all API's require signing up for a private key. In that case, just use the provided address from a server route
  - Create server route(s)
  - Create a component
  - Create UI to search/display stuff
- Example: `../examples/180-server-routes`



README.md   JavaScript APIs.txt

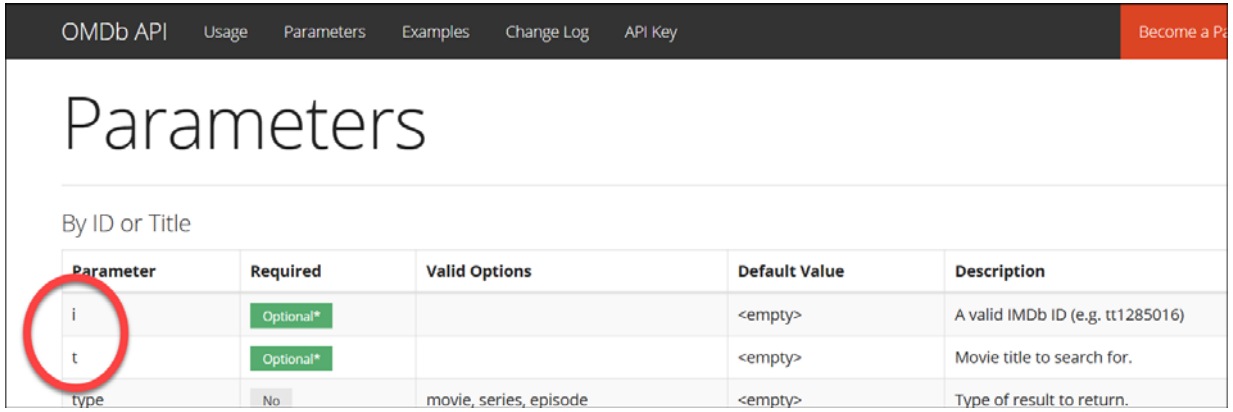
```
//****************************************************************
// DEZE API's zijn open, zonder registratie beschikbaar, of al geregistreerd (regist
// THESE API's are open, free, mostly available without registration (or already reg

https://opendata.rdw.nl/resource/m9d7-ebf2.json?kenteken= + kenteken
https://swapi.dev/ - The Star Wars API
https://pokeapi.co/ - The RESTful Pokemon API
https://api.openweathermap.org/data/2.5/weather?units=metric&appid=[redacted]5
https://ergast.com/mrd/ - Ergast Motor (Formula 1) API
https://randomuser.me/api/?results=10 - Random user data
```


I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day

Workshop #2 - optional

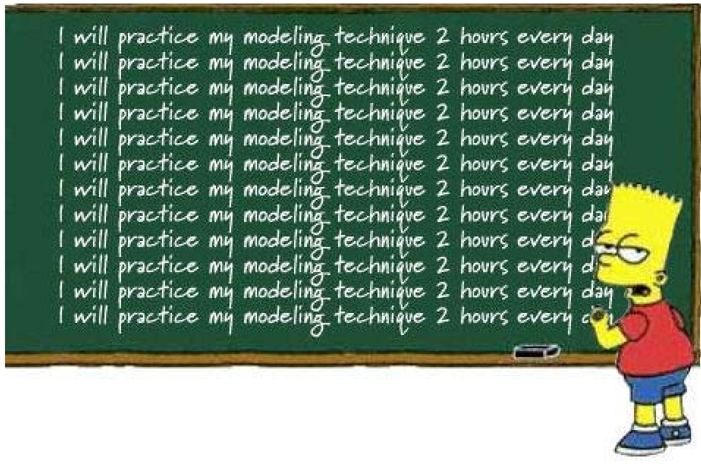
- Start from `../examples/180-server-routes`
- Create a Movie Detail page.
- Requirements:
  - Clicking on a movie opens a detail page
  - Make a subsequent server route request, using the `imdbID` property as a key
  - Create TypeScript interfaces for the movie details
  - Tip: use the `i=`… or `t=`… parameter from the documentation at omdbapi.com.



Parameters

By ID or Title

Parameter	Required	Valid Options	Default Value	Description
i	Optional*		<empty>	A valid IMDb ID (e.g. tt1285016)
t	Optional*		<empty>	Movie title to search for.
type	No	movie, series, episode	<empty>	Type of result to return.



Checkpoint

- You know what server routes in a Nuxt application are
- You know how to create server routes
- You can name some scenarios when use server routes
- You can create TypeScript interfaces for the results
- You are able to do server route calls from your frontend