Building Nuxt.js Applications 🔥

Wifi: beachcodevue
Password: vueconf2022

Breaks:
10:30 - 15 mins
12:30 ish 45mins
2:30 - 30 mins
4:30 + finish

Intro

Who are you? Who am I? 😅

What is Nuxt?

What we will cover!

— Create Nuxt app
— Dynamic Pages
— Data fetching
— Nuxt Content
— Nuxt commands and deployments
— Build an application
🤯
— Anything else you want me to cover

How are we gonna do this?

- All on own and push to repo
- Work in pairs if you prefer
- Ask Questions by unmuting yourself
- Check in on break times and change direction if necessary

Creating a Nuxt.js project

```
yarn create nuxt-app <project-name>
```

Let's get started

```
cd <project-name>
yarn dev
```

Project Structure

What's in my folders?

Routing

How are routes created?

How do we link from one page to another?

Exercise

1. Create a page called about
2. Add a link to the about page and home page
3. And a link in the about page to the home page

Solution - pages/index.vue

```
<template>
  <main>
    <h1>Home page</h1>
    <NuxtLink to="/about">
      About
    </NuxtLink>
  </main>
</template>
```

pages/about.vue

```
<template>
  <main>
    <h1>About page</h1>
    <NuxtLink to="/Home">
      Home
    </NuxtLink>
  </main>
</template>
```

Images

Static:

```
<img src="my-image.jpg />
```

Assets:

```
<img src="~/assets/my-image.jpg" />
```

Images

Background Image:

```
background: url('~assets/my-image.jpg')
```

Dynamic image:

```
<img src="require(`~/assets/${image}.jpg`)" />
```

Components

How do I add a component?

nuxt.config.js

```
export default {
  components: true
}
```

Components

How do I add a component?

```
components/
  TheHeader.vue
```

```
pages/index.vue
```

```
<template>
  <div>
    <TheHeader />
  </div>
</template>
```

Components

How can I Lazy Load it?

pages/index.vue

```
<template>
  <div>
    <TheHeader />
    <LazyTheFooter />
  </div>
</template>
```

Components

What about Nested Directories?

```
components/
  base/
    Button.vue
```

```
components: {
  dirs: [
    '~/components',
      {
        path: '~/components/base/',
        prefix: 'Base'
      }
  ]
}
```

```
<BaseButton />
```

Exercise Time

1. Create a folder called base in the components folder and inside:
   — Create an Alert.vue component
   — Create a Button.vue component
2. Prefix your components to have the word 'Base' without changing the component name
3. Add both components to your page as BaseAlert and BaseButton
4. LazyLoad the Alert component
5. When you click on the Button component it will show

Solution

Folder structure

```
components/
    Base/
        Button.vue
        Alert.vue
```

Solution

nuxt.config.js

```
components: {
  dirs: [
    '~/components',
      {
        path: '~/components/base/',
        prefix: 'Base'
      }
  ]
}
```

Solution: pages/index.vue

```
<template>
  <div>
    <LazyBaseAlert></BaseAlert>
    <button v-if="!show" @click="showAlert">Click Me</button>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        show: false
      }
    },
    methods: {
      showAlert() {
        this.show = true
      }
    }
  }
</script>
```

Data Fetching

— AsnycData Hook
— Fetch Hook

The Fetch Hook

`fetch` is a hook called during server-side rendering after the component instance is created, and on the client when navigating. The fetch hook should return a promise (whether explicitly, or implicitly using `async/await`) that will be resolved:

— On the server before the initial page is rendered

— On the client some time after the component is mounted

The Fetch Hook

```
export default {
  data() {
    return {
      mountains: []
    }
  },
  async fetch() {
    this.mountains = await fetch('https://api.nuxtjs.dev/mountains').then(res =>
      res.json()
    )
  },
}
```

The Fetch Hook

It exposes `$fetchState` at the component level with the following properties:

- `pending` is a `Boolean` that allows you to display a placeholder when `fetch` is being called on client-side.
- `error` is either `null` or an `Error` thrown by the fetch hook
- `timestamp` is a timestamp of the last fetch, useful for caching with `keep-alive`

The Fetch Hook

```
<template>
  <p v-if="$fetchState.pending">Fetching mountains...</p>
  <p v-else-if="$fetchState.error">An error occurred :(</p>
  <div v-else>
    <h1>Nuxt Mountains</h1>
    <ul>
      <li v-for="mountain of mountains">{{ mountain.title }}</li>
    </ul>
  </div>
</template>
```

The Fetch Hook

You can manually call fetch in your component by calling `this.$fetch()`.

```
<button @click="$fetch">Refresh</button>
```

The Fetch Hook

You can use `keep-alive` directive in `<nuxt/>` and `<nuxt-child/>` component to save fetch calls on pages you already visited:

```
<template>
  <nuxt keep-alive />
</template>
```

Exercise

1. Create a card component, image, title, description etc
2. Use fetch to call the river API
3. Add the card component to a page
4. Add a button to refetch the data on click
5. Use fetchState to show loading message
6. Use fetchState to show error message

Nuxt API

https://api.nuxtjs.dev/rivers

Solution

```
<template>
  <p v-if="$fetchState.pending">Fetching rivers...</p>
  <p v-else-if="$fetchState.error">An error occurred :(</p>
  <div v-else>
    <h1>Nuxt Rivers</h1>
    <ul>
      <li v-for="river of rivers">
                <h2>{{ river.title }}</h2>
                <img :src="river.image" :alt="river.title" />
                <p>{{ river.description }}</p>
            </li>
    </ul>
        <button @click="$fetch">Refresh</button>
  </div>
</template>
```

Solution

```
export default {
  data() {
    return {
      rivers: []
    }
  },
  async fetch() {
    this.rivers = await fetch('https://api.nuxtjs.dev/rivers').then(res =>
      res.json()
    )
  },
}
```

AsyncData Hook

- Unlike fetch, the promise returned by the asyncData hook is resolved during route transition. This means that no "loading placeholder" is visible during client-side transitions. Nuxt will instead wait for the asyncData hook to be finished before navigating to the next page or display the error page
- asyncData simply merges its return value into your component's local state
- This hook can only be used for page-level components

AsyncData Hook

```
export default {
  async asyncData() {
    this.rivers = await fetch('https://api.nuxtjs.dev/rivers').then(res =>
      res.json()
    )
  },
}
```

Handling Errors with Fetch API

With fetch API we always get a response even if the response is not ok

We should only show the response if it is ok

```
if(response.ok){
    return response.json()
}
```

Handling Errors with Fetch API

In order to catch an error in a try catch block we must first throw the new Error

```
throw new Error(response.status)
```

Exercise

1. Add a try/catch block
2. catch your errors
3. show an error message/component

`https://api.nuxtjs.dev/rivers`

Solution

```
export default {
  async asyncData() {
    try {
            const rivers = await fetch('https://api.nuxtjs.dev/rivers').then(response => {
                if (response.ok) {
                    return response.json()
          }
                throw new Error('there was an error fetching data')
            })
            return { rivers }
    } catch (error) {
            return { error }
  },
}
```

Solution

```
<div v-if="error">There was an error</div>
<div v-else>....
```

AsyncData with Axios

Import axios package and use it.

```
import axios from 'axios'

export default {
  async asyncData() {
            const rivers = await axios.get('https://api.nuxtjs.dev/rivers').then(response => {
                    return response.data
        })
            return { rivers }
    }
}
```

AsyncData with Axios Module

Install it once and it can be used on any page

```
yarn add @nuxtjs/axios

export default {
  modules: ['@nuxtjs/axios']
}
```

AsyncData with Axios Module

— uses $ as helper

— returning data is easier

```
export default {
  async asyncData( {$axios} ) {
            const rivers = await $axios.$get('https://api.nuxtjs.dev/rivers')
            return { rivers }
    }
}
```

Fetch Hook with Axios module

```
export default {
    data() {
     return {
       rivers: []
     }
  },
  async fetch() {
            this.rivers = await this.$axios.$get('https://api.nuxtjs.dev/rivers')
            return { rivers }
    }
}
```

Exercise

Create a page that fetches data from an api using AsyncData and axios module

`https://api.nuxtjs.dev/rivers`

Solution

```
yarn add @nuxtjs/axios

export default {
  modules: ['@nuxtjs/axios']
}

export default {
  async asyncData( {$axios} ) {
            const rivers = await $axios.$get('https://api.nuxtjs.dev/rivers')
            return { rivers }
    }
}
```

Creating Dynamic pages

To create a dynamic route you need to add an underscore before the .vue file name or before the name of the directory

```
pages/_rivers.vue
```

Exercise

1. Create a dynamic page for the river details
2. Link from the river cards to the dynamic page and back

Solution

```
export default {
  async asyncData( {$axios, params} ) {
    const river = await $axios.$get(`https://api.nuxtjs.dev/rivers/${params.rivers}`)
    return { river }
    }
}
```

Nuxt content, markdown, yaml, components in Markdown

Beer app works with Content

Exercise

Re-create the beer App

Solution

https://github.com/debs-obrien/vue-toronto

http://nuxt-beers.surge.sh/

Deploying your Nuxt.js Application

Let's share our code with the web

target: 'server'

What commands are available?

target: 'static'

What commands are available?

ssr: false

What happens when we don't want ssr

Exercise

Deploy you application

Solution

lets see it live

Solution

Q and A

Question Time

The End