Introduction

In the Touring Vue Router course, we'll be exploring most of the functionality found in the Vue Router library, which allows us to create advanced navigation through our Single Page Applications using Vue.

Most of this configuration will be done inside our `/router/index.js` file, where we define our application's routes. More specifically which URL paths link up to which components and where they're defined on our screen.

To take this course I'm assuming you have basic Vue knowledge and are familiar with the concepts taught in our Intro to Vue 3 course and Real World Vue 3 course. Especially what is taught in the Vue Router Essentials lesson, which teaches how to setup Vue Router and the very basics of using it.

On our tour we'll be building out the events application you may have started building with us in our Real World Vue 3 course. We'll build in pagination, proper error handling when a page doesn't exist or the network is down, a progress bar to compensate when a page is slow to load (probably because of a slow API call), and a flash message to give our users a message that appears at the top of any page.

We'll also give an overview of much of the Vue Router syntax you'll need to build out big Vue applications.

Receiving URL Parameters

In this lesson we'll give an overview of all the different ways we can receive and parse URL data into our components with Vue Router. This will ensure we have the tools we need to build pagination in our next lesson.

🛑 Problem: How do we read query parameters off the URL?

For example, often when we write pagination, we might have a URL that looks like this: `http://example.com/events?page=4`

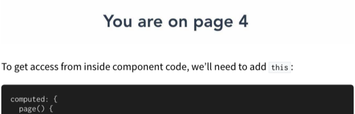
How can we get access to `page` inside our component?

✅ Solution: $route.query.page

Inside our component to read the page listing all we need to do inside our template is write:

```
<h1>You are on page {{ $route.query.page }}</h1>
```

It might look like this:



To get access from inside component code, we'll need to add `this`:

```
computed: {
  page() {
    return parseInt(this.$route.query.page) || 1
  },
}
```

🛑 Problem: What if we wanted the page to be part of the URL?

There are some cases in web development where you might want the page number to be an actual part of the url, instead of in the query parameters (which come after a question mark).

✅ Solution: Route Parameter

If you watched Vue Mastery's Real World Vue course, you might already be familiar with the solution. To solve this we'd likely have a route that looked like:

```
const routes = [
  ...
  { path: '/events/:page', component: Events },
]
```

Then inside our event component, we could access this in the template as such:

```
<h1>You are on page {{ $route.params.page }}</h1>
```

Notice that in this case we are using `$route.params` instead of `$route.query` as we did above.

Bonus: Passing Params as Props

If we want to make our component more reusable and testable, we can decouple it from the route by telling our router to pass our Param `page` as a Component prop. To do this, inside our router we would write:

```
const routes = [
  ...
  { path: '/events/:page', component: Events, props: true },
]
```

Then inside our component we would have:

```
<template>
  <h1>You are on page {{ page }}</h1>
</template>
<script>
export default {
  props: ["page"]
};
</script>
```

Notice we are declaring `page` as a prop and rendering it to the page.

🛑 Problem: Route level configuration

Sometimes we have a component we want to be able to configure at the route level. For example, if there is extra information we want to display or not.

✅ Solution: Props Object Mode

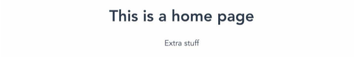
When we want to send configuration into a component from the router, it might look like this:

```
const routes = [
  {
    path: "/",
    name: "Home",
    component: Home,
    props: { showExtra: true },
  },
```

Notice the static props object with `showExtra`. We can then receive this as a prop in our component, and do something with it:

```
<template>
  <div class="home">
    <h1>This is a home page</h1>
    <div v-if="showExtra">Extra stuff</div>
  </div>
</template>
<script>
export default {
  props: ["showExtra"]
};
</script>
```

And now when I view my home page, I see:



Extra stuff

🛑Problem: How to transform query parameters?

Sometimes you may have a situation where the data getting sent into your query parameters needs to be transformed before it reaches your component. For example, you may want to cast parameters into other types, rename them, or combine values.

In our case let's assume our URL is sending in `e=true` as a query parameter, while our component wants to receive `showExtra=true` using the same component in the above example. How could we do this transform?

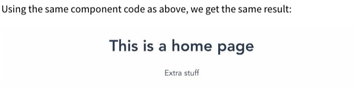
✅Solution: Props Function Mode

In our route to solve this problem we would write:

```
const routes = [
  {
    path: "/",
    name: "Home",
    component: Home,
    props: (route) => ({ showExtra: route.query.e }),
  },
```

Notice we're sending in an anonymous function which receives the `route` as an argument, then pulls out the query parameter called `e` and maps that to the `showExtra` prop.

Using the same component code as above, we get the same result:



Extra stuff

The anonymous function above could also be written like:

```
props: route => {
  return { showExtra: route.query.e }
}
```

It's a little more verbose, but I wanted to show you this to remind you that you could place complex transformations or validations inside this function.

In the next video we'll use some of these techniques to build out some pagination.