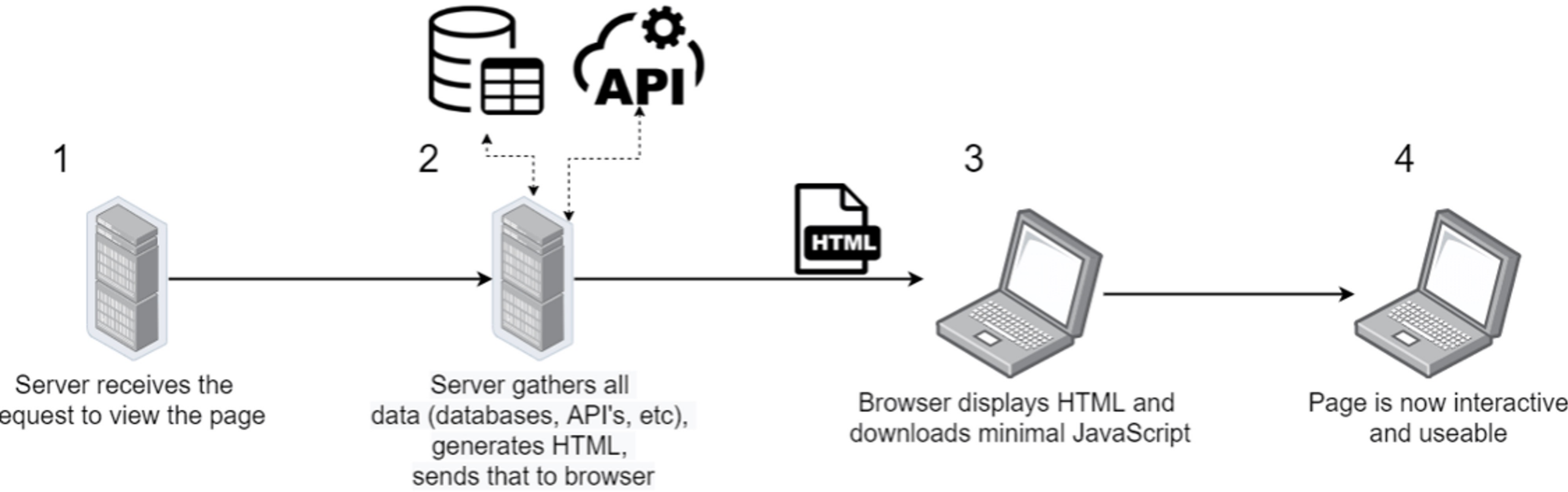
AJAX + AXIOS

Web Programming

SSR



res.render('index')

Loading

Loading

Visible but not
interactive

Complete!

CSR

Front-end Server

Back-end Server

API

1

Server sends response to browser with minimal HTML

2

Browser downloads the JavaScript

3

Browser executes JavaScript starting frontend framework and gathers required data

4

Page is now interactive and ready for visitor

Loading

Loading

Loading

Complete!

TODAY'S TOPICS

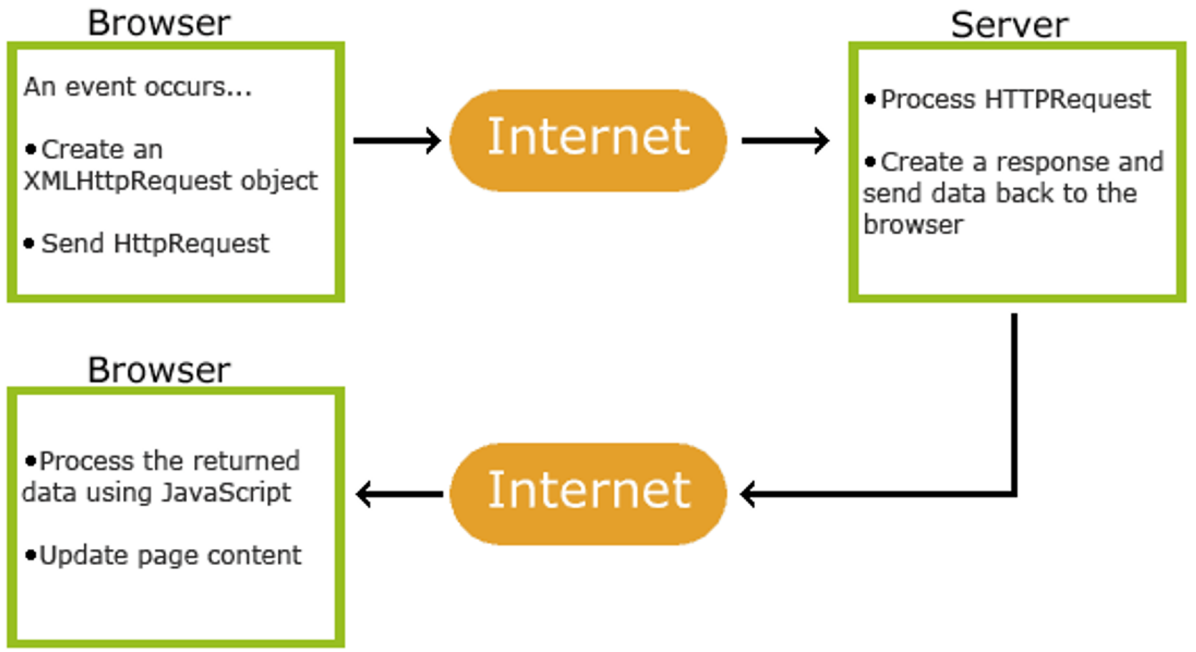
- AJAX
  - What is AJAX
  - Fetch API
- Axios
  - Consuming REST APIs
- Linking front-end with back-end
  - Vue CLI
  - Vue Router
  - Let's update the "youblog" website

WHAT IS AJAX?

- AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.
- AJAX is not a programming language.
- AJAX is a technique that uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)

HOW AJAX WORKS

1. An event occurs ... (clicking a button)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript



AJAX

AJAX is a developer's dream, because you can:

- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

Let’s see some example

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

FETCH API

- The Fetch API provides an interface for fetching resources (including across the network).
- Fetch provides a generic definition of Request and Response objects.
  - The fetch() method takes one mandatory argument, the path to the resource you want to fetch.
  - It returns a Promise that resolves to the Response to that request, whether it is successful or not.

AXIOS

Promise based HTTP client for the browser and node.js

FEATURES

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF

GET REQUESTS

```
const axios = require('axios');

// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .then(function () {
    // always executed
  });
```

Async & Await

```
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

POST REQUESTS

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

REQUEST METHOD ALIASES

- axios.request(config)
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])

For more detail:
https://www.npmjs.com/package/axios

Vue CLI

🛠️ Standard Tooling for Vue.js Development

VUE CLI

Vue Command-line Interface

VUE CLI

Vue CLI is a full system for rapid Vue.js development, providing:

1. Interactive project scaffolding via @vue/cli.
2. Zero config rapid prototyping via @vue/cli + @vue/cli-service-global.
3. A runtime dependency (@vue/cli-service) that is:
   - Upgradeable;
   - Built on top of webpack, with sensible defaults;
   - Configurable via in-project config file;
   - Extensible via plugins
4. A rich collection of official plugins integrating the best tools in the frontend ecosystem.
5. A full graphical user interface to create and manage Vue.js projects.

VUE ROUTER

https://router.vuejs.org/

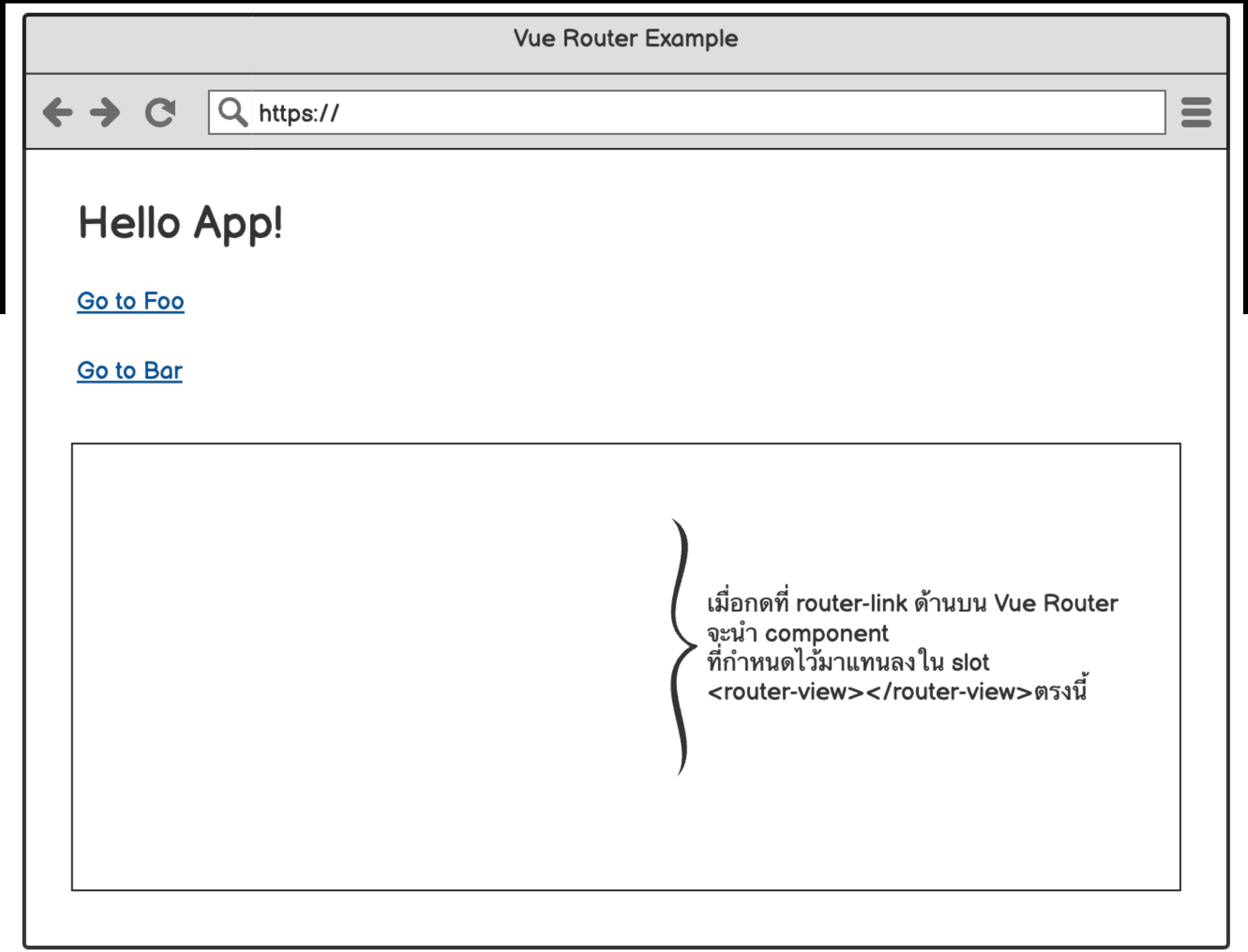
VUE ROUTER

Vue Router is the official router for Vue.js. It deeply integrates with Vue.js core to make building Single Page Applications with Vue.js a breeze.

SPA EXAMPLE

HTML

```html
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="app">
  <h1>Hello App!</h1>
  <p>
    <!-- use router-link component for navigation. -->
    <!-- specify the link by passing the `to` prop. -->
    <!-- `<router-link>` will be rendered as an `<a>` tag by default -->
    <router-link to="/foo">Go to Foo</router-link>
    <router-link to="/bar">Go to Bar</router-link>
  </p>
  <!-- route outlet -->
  <!-- component matched by the route will render here -->
  <router-view></router-view>
</div>
```


Vue Router Example
https://
Hello App!
Go to Foo
Go to Bar
เมื่อกดที่ router-link ด้านบน Vue Router
จะนำ component
ที่กำหนดไว้มาแทนลงใน slot
<router-view></router-view>ตรงนี้

JavaScript

```js
// 0. If using a module system (e.g. via vue-cli), import Vue and VueRouter
// and then call `Vue.use(VueRouter)`.

// 1. Define route components.
// These can be imported from other files
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }

// 2. Define some routes
// Each route should map to a component. The "component" can
// either be an actual component constructor created via
// `Vue.extend()`, or just a component options object.
// We'll talk about nested routes later.
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]

// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = new VueRouter({
  routes // short for `routes: routes`
})

// 4. Create and mount the root instance.
// Make sure to inject the router with the router option to make the
// whole app router-aware.
const app = new Vue({
  router
}).$mount('#app')

// Now the app has started!
```

DYNAMIC ROUTE MATCHING – PATH PARAMS

In vue-router we can use a dynamic segment in the path to achieve that:

Now URLs
like /user/foo and /user/bar
will both map to the same route.

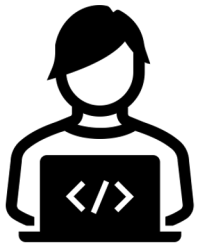
```
const User = {
  template: '<div>User</div>'
}

const router = new VueRouter({
  routes: [
    // dynamic segments start with a colon
    { path: '/user/:id', component: User }
  ]
})
```

$ROUTE.PARAMS

When a route is matched, the value of the dynamic segments will be exposed as this.$route.params in every component.

```
const User = {
  template: '<div>User {{ $route.params.id }}</div>'
}
```

$route.query:
An object that contains key/value pairs of the query string.
For example, for a path /foo?user=1, we get $route.query.user = 1.



Code Example Click Here!

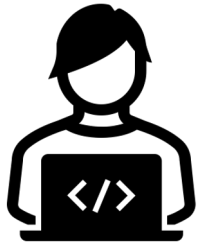
NESTED ROUTES

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:id',
      component: User,
      children: [
        {
          // UserProfile will be rendered inside User's <router-view>
          // when /user/:id/profile is matched
          path: 'profile',
          component: UserProfile
        },
        {
          // UserPosts will be rendered inside User's <router-view>
          // when /user/:id/posts is matched
          path: 'posts',
          component: UserPosts
        }
      ]
    }
  ]
})
```

Root path

Root component

Children



Code Example Click Here!

NAMED ROUTES

Sometimes it is more convenient to identify a route with a name, especially when linking to a route or performing navigations. You can give a route a name in the routes options while creating the Router instance:

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

```
<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>
```

PROGRAMMATIC NAVIGATION

Aside from using <router-link> to create anchor tags for declarative navigation, we can do this programmatically using the router's instance methods.

```
this.$router.push(location, onComplete?, onAbort?)
```

To navigate to a different URL, use this.$router.push. This method pushes a new entry into the history stack, so when the user clicks the browser back button they will be taken to the previous URL.

```
// literal string path
router.push('home')

// object
router.push({ path: 'home' })

// named route
router.push({ name: 'user', params: { userId: '123' } })

// with query, resulting in /register?plan=private
router.push({ path: 'register', query: { plan: 'private' } })
```

LET'S do the tutorial

https://github.com/it-web-pro/WEEK11-1-TUTORIAL