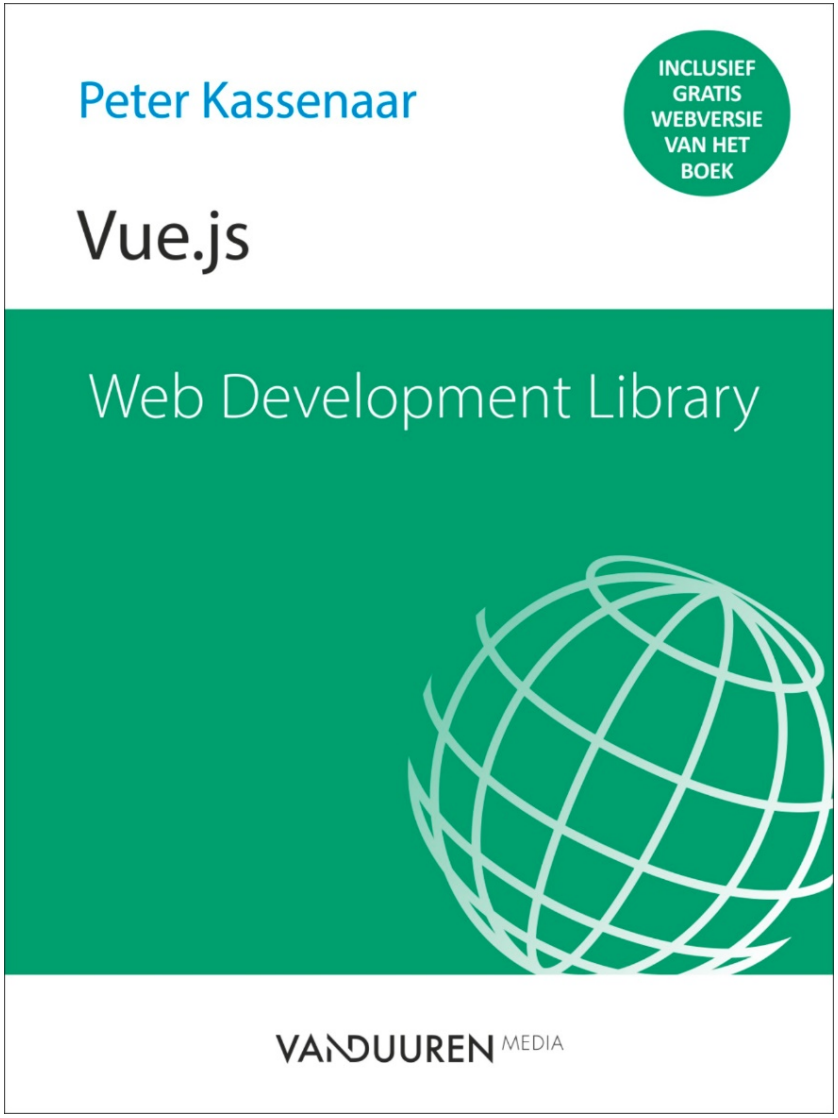
Vue Fundamentals - 05
Routing basics

intracto
IMAGINE TOMORRO_

Peter Kassenaar –
info@kassenaar.com



Chapter 6, P. 164

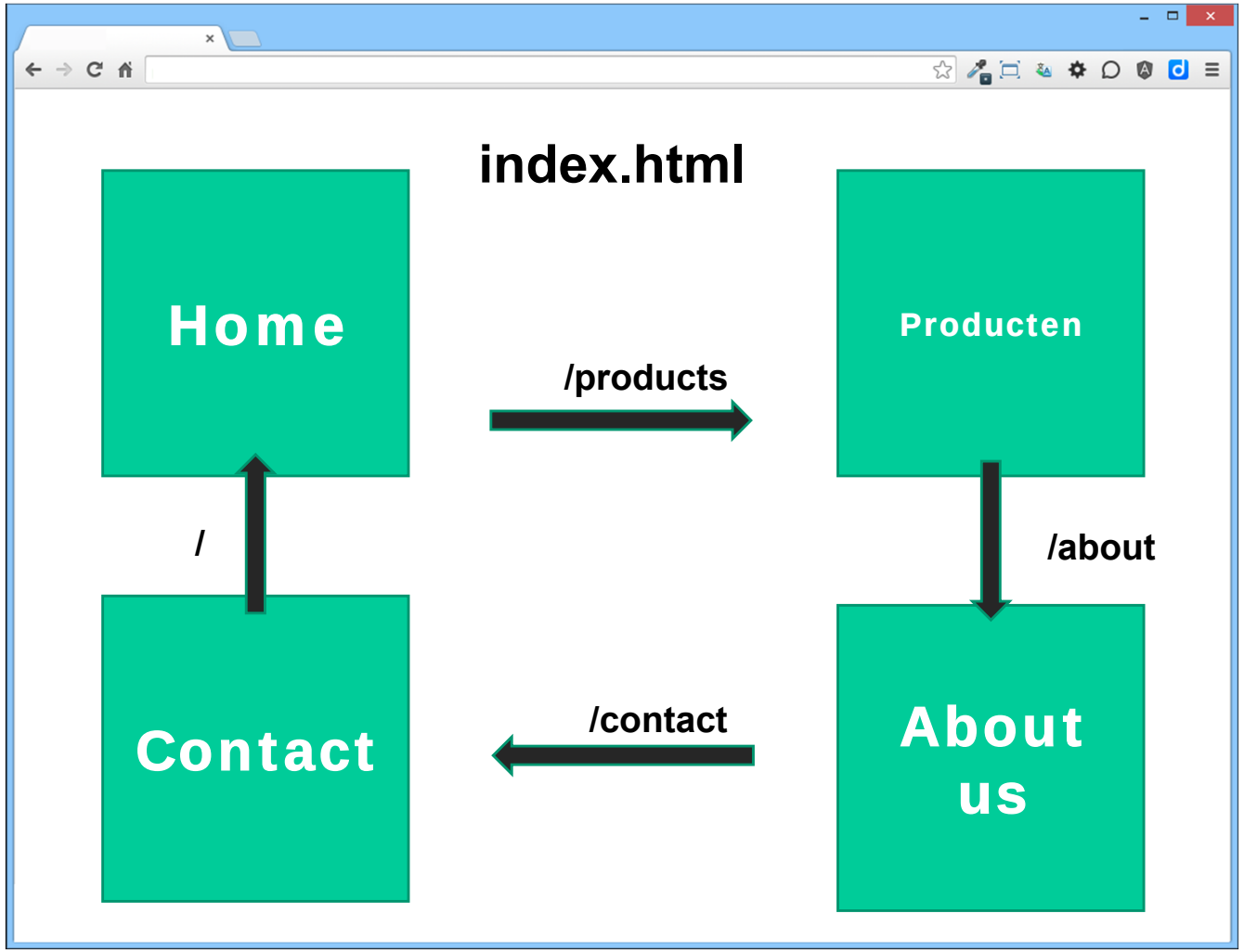
Routing

“For most Single Page Applications, it’s recommended to use the officially-supported vue-router library. For more details, see vue-router’s documentation.`”

https://github.com/vuejs/vue-router

https://router.vuejs.org/

Routing architecture and goal



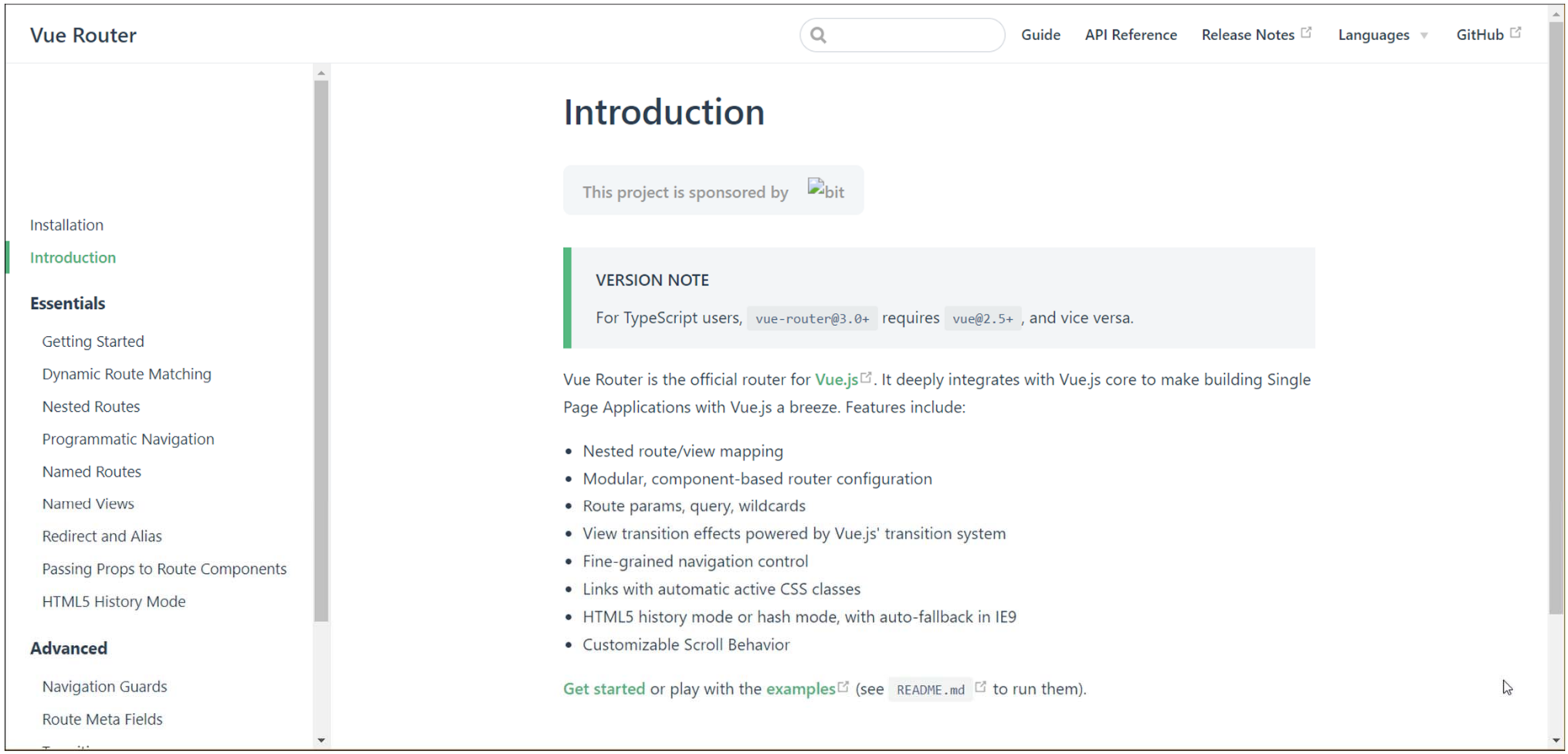
- Make use of SPA principle
- Making deep links possible

Router capabilities / characteristics

- Update URL when changing to routes/components
- Nested routes/view mapping
- Modular, component based router configuration
- Route params, query, wildcards
- View transition effects
- Link with automatic active CSS classes to denote active route
- HTML5 history mode
- Prevent navigating to a route, or navigating away

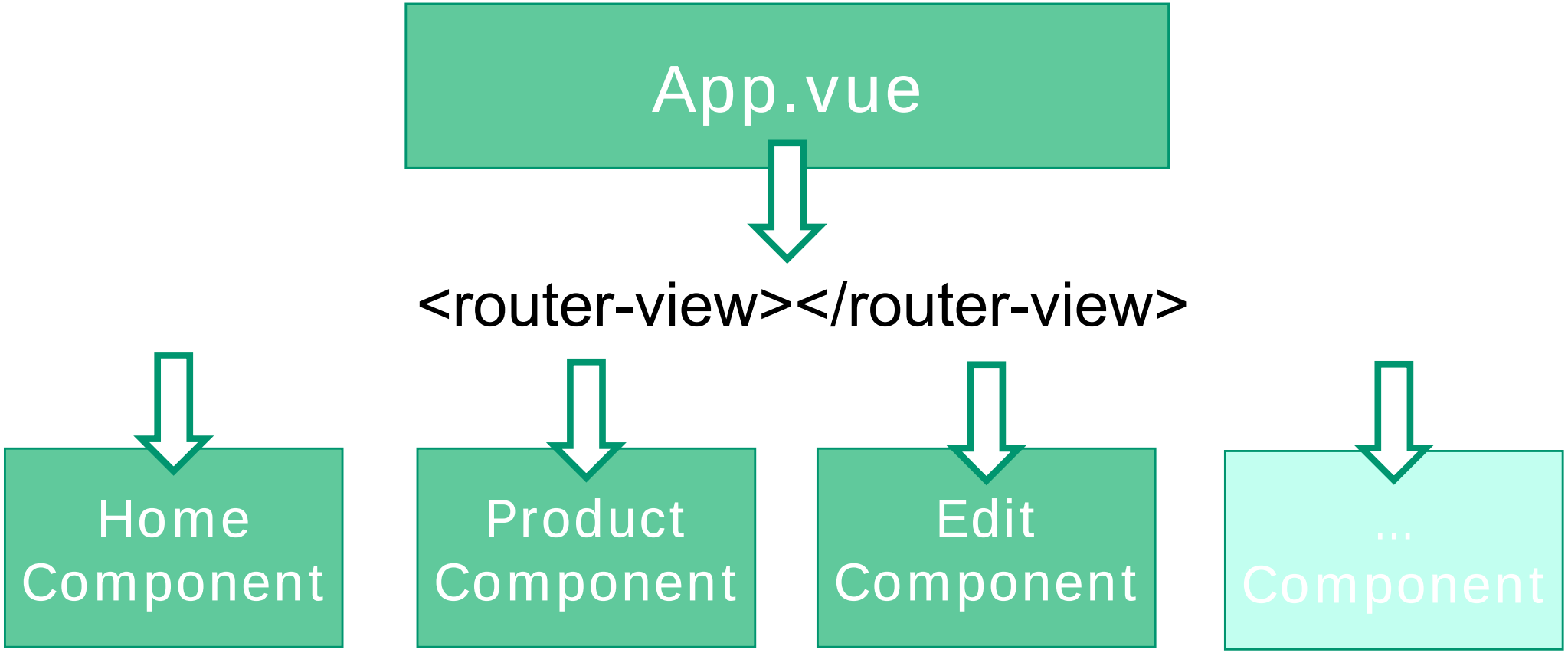
https://router.vuejs.org/

https://router.vuejs.org/



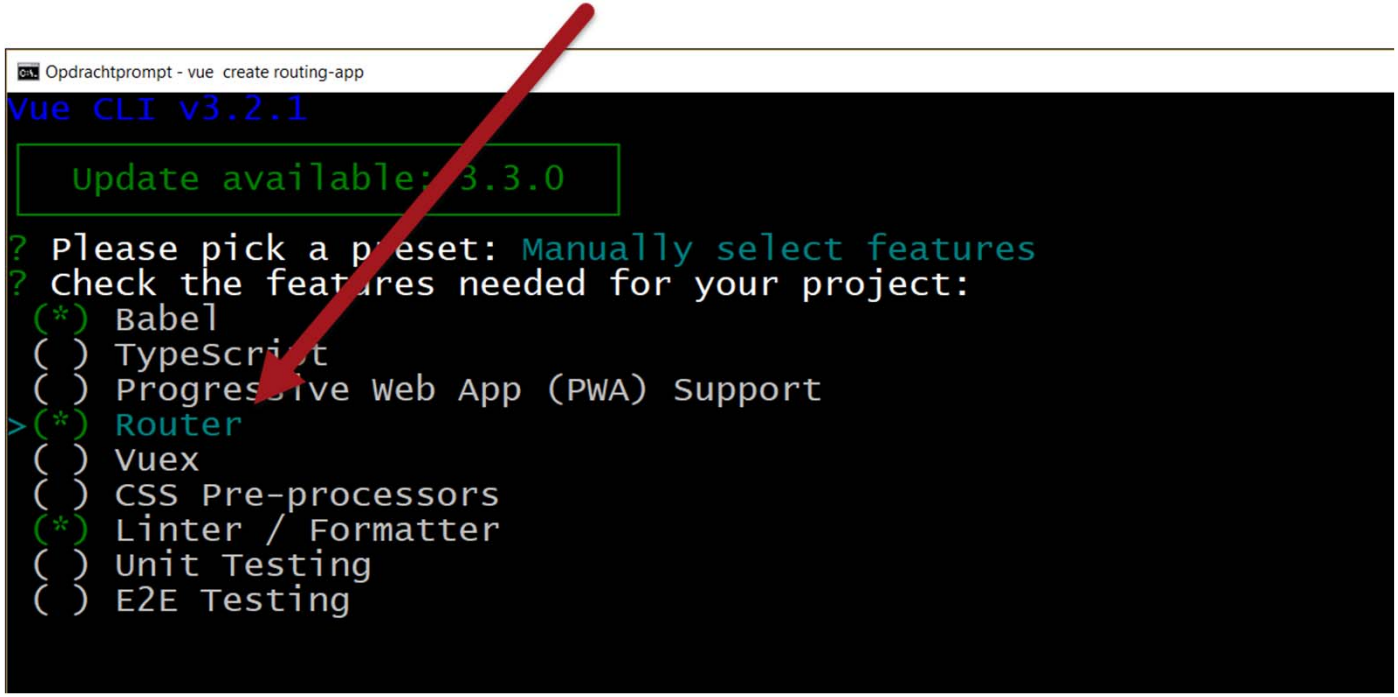
Routing – every route is a Component

- `App.vue` gets a main menu (typically in its own component :-)
- Components are injected in `<router-view></router-view>`

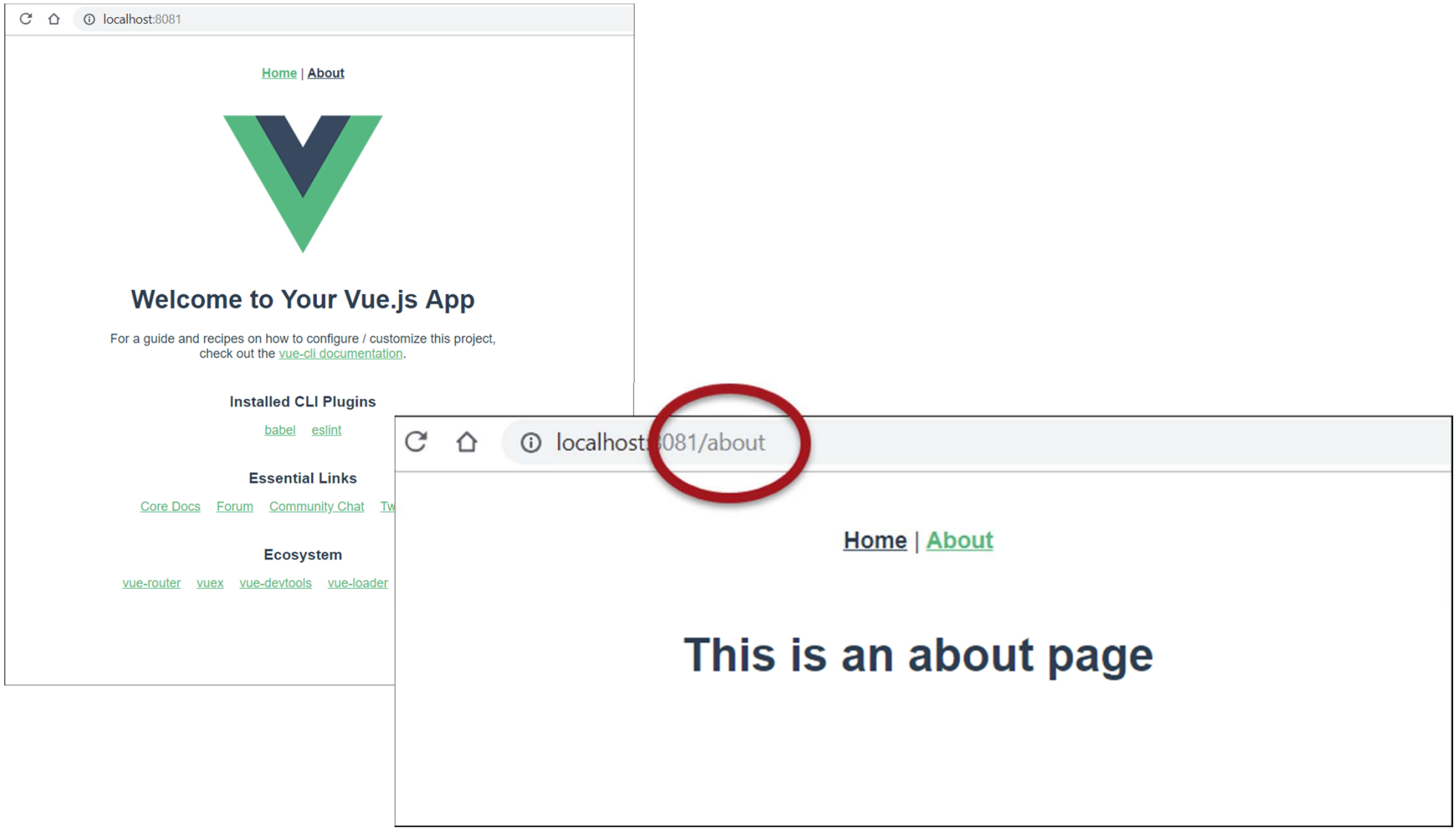


Sidestep - routing via Vue CLI

- For instance: `vue create routing-app`
- `Manually select features`
- Select `Router` (you can have multiple selections!)
- `Use history mode for router?` – typically `Yes`



Result – a default routing app



This is exactly what we are going to create manually

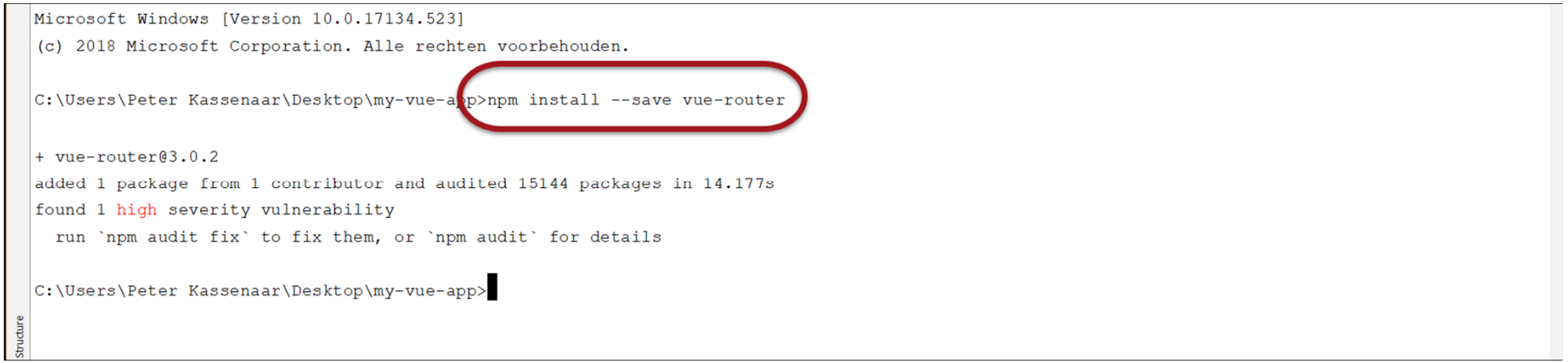
Agenda - In this module

- Adding routing to an existing app
  - Also possible – start a new app with routing from scratch, using the CLI
- Linking to routed pages
- Styling links, based on the active route
- Working with route params
- Nested router views
- More info on routing

https://router.vuejs.org/guide/

1. Installing the router to an existing app

- Install the router module: `npm install --save vue-router`
- Check if the application still runs!



2. Create router folder

- Create a `router` folder, holding all router configuration
  - Not mandatory, just for good code organization
  - Otherwise, call your file `router.js` in the root of the application
- Create an `index.js` inside `../router`
- Basic lay-out for routing table:

```
// ../router/index.js
import Vue from 'vue'
import Router from 'vue-router'

Vue.use(Router);

export default new Router({
   routes: [
      // define all routes here....
   ]
})
```

3. Define routing table

```
…
// import the required components
import VacationPicker from "../components/VacationPicker";
import AddCountry from "../components/AddCountry";
import UpdateCountry from "../components/UpdateCountry";

export default new Router({
   routes: [
      // define all routes here....
      {
         path: '/',
         name: 'home',
         component: VacationPicker
      },
      {
         path: '/add',
         name : 'add',
         component: AddCountry
      }
      …
   ]
})
```

Tell Vue which component to load when a specific route is requested

3a. Optional: lazy loading

- You can *lazy load* components, i.e. only load them once the user navigates to them
- Use the webpack `import` statement for that
- Don't forget to remove the component from the list with `import` statements at the top of the file! (if applicable)
  - Otherwise it would still *always* be loaded, and not lazy loaded.

```
{
    path: '/add',
    name: 'add',
    component: () => import('../components/AddCountry') // lazy loading
},
```

Optional w/ lazy loading

- Give the chunk / module that webpack creates a meaningful name:

```
{
    path: '/add',
    name: 'add',
    component: () => import(/* webpackChunkName: "add-component" */ '../components/AddCountry')
},
```

4. Add the routes to Vue configuration

- Add the correct `import` to `main.js`
- Add `router` to the Vue instance like so:

```
// main.js
…

// import router stuff
import router from './router'

Vue.config.productionTip = false;

new Vue({
   render: h => h(App),
   router,
}).$mount('#app');
```

5. Tell Vue where to project routed content

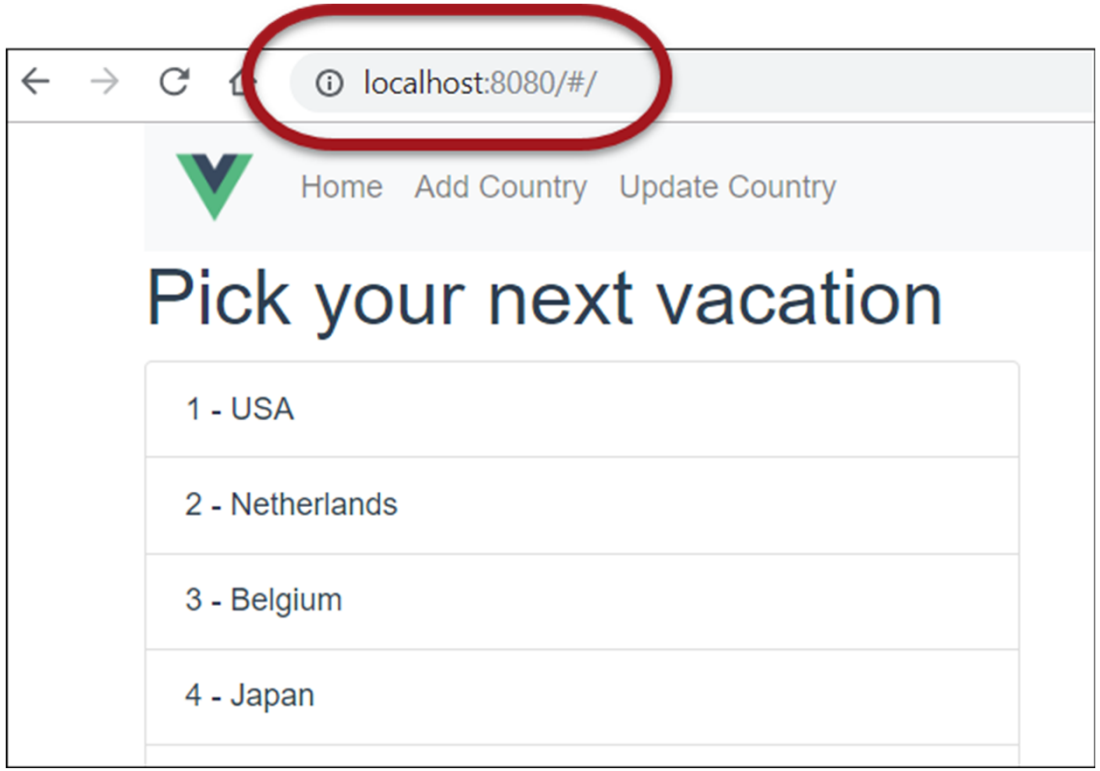
- Use `<router-view />` to tell Vue where to project the components
- You don't need the import for `VacationPicker` in `App.vue` anymore
  - We *do* need `MainNavigation`, b/c we placed it inside its own component

```
<!--App.vue-->
<template>
    <div id="app" class="container">
        <MainNavigation />
        <router-view></router-view>
    </div>
</template>

<script>
   import MainNavigation from "./components/MainNavigation";

   export default {
      name: 'app',
      components: {
         MainNavigation,
      }
   }
</script>
```

Result so far:



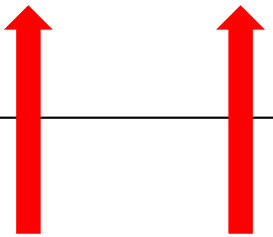
The links in the `MainNavigation` component don't work yet!

We will solve the hashtag – `localhost:8080/#/` issue later on, by using HTML5-mode routing

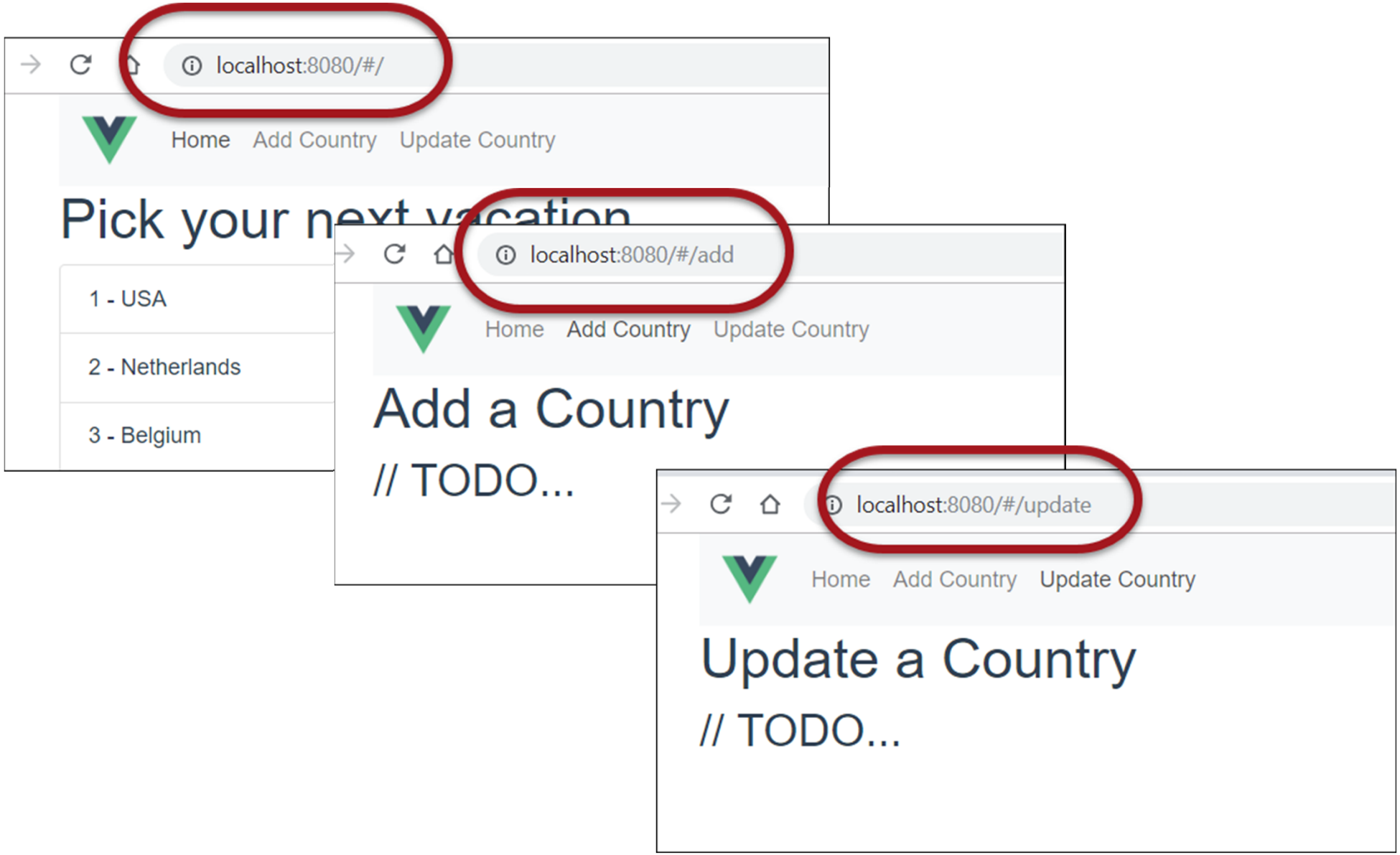
Linking to routed pages

- We can now route to pages by typing the route manually in the address bar.
  - But of course we want to achieve this by clicking links!
- No more <a href="…"> -tags!
- Use Vue-specific <router-link to="name-of-route"> tags

```
…
<li class="nav-item">
    <router-link to="/" class="nav-link">Home</router-link>
</li>
<li class="nav-item">
    <router-link to="add" class="nav-link">Add Country</router-link>
</li>
…
```



Result


localhost:8080/#/
Home
Add Country
Update Country
Pick your next vacation
1 - USA
2 - Netherlands
3 - Belgium
localhost:8080/#/add
Home
Add Country
Update Country
Add a Country
// TODO...
localhost:8080/#/update
Home
Add Country
Update Country
Update a Country
// TODO...

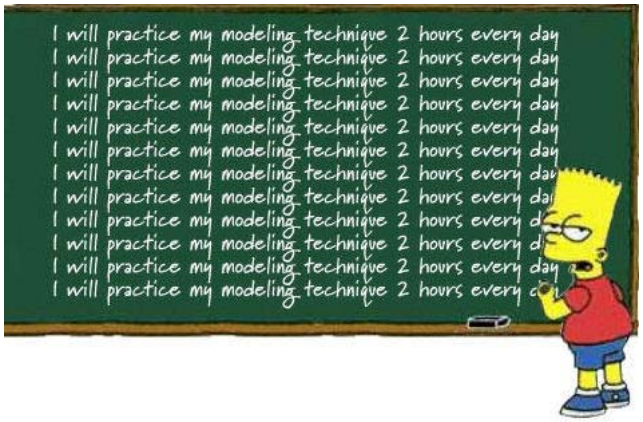
Another syntax – navigating by binding

- We also defined a `name` property for our routes
- Use the `name` to navigate by binding to `:to`
- By clicking the logo, we navigate to the home page (i.e. `VacationPicker` component)
- This is completely optional, but you see how you can also link to routes and bind via code

```
<router-link class="navbar-brand" :to="{name: 'home'}">
    <img src="../assets/logo.png" alt="Vue Logo">
</router-link>
```

Workshop

1. Add routing to your own application, using the steps in this module. OR:
2. Start a new project from scratch, using the CLI
   - Add routing via the prompt
   - Add a few components, route to them
   - Don't forget to link to the new routes/components, OR:
3. Generic example: `../300-routing-basics`
   - Add a new component
   - Add a route to that component
   - Update the `<MainNavigation />`
   - Make sure it works



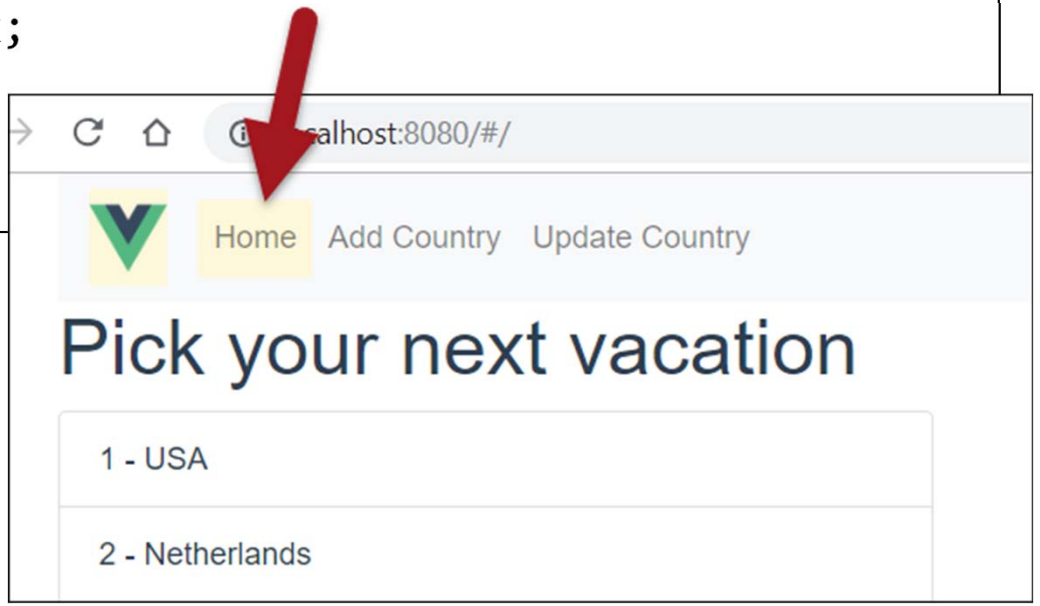
Styling the active link

Emphasizing the current route in the UI, based on a URL match

Special class names

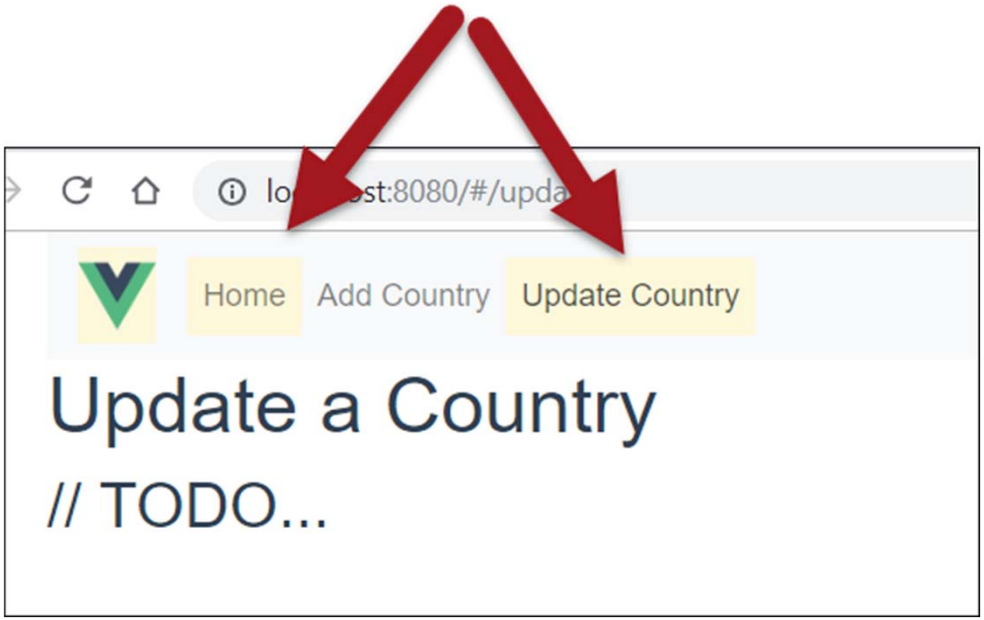
- Vue assigns a `router-link-active` class to the current active route.
- You can write a CSS-class for that
- No need to update the HTML/UI. Vue assigns it automatically.
- Of course you can add any property you want/need

```
<style scoped>
    /*Automatic assignment of this class by Vue, based on router state*/
    .router-link-active{
        background-color: cornsilk;
        color: white;
    }
</style>
```



Issue – the home link

- Vue is by default using an *inclusive match* on the route
  - Which means, the home route is *always* `active` (as every route has a slash `'/'` in it



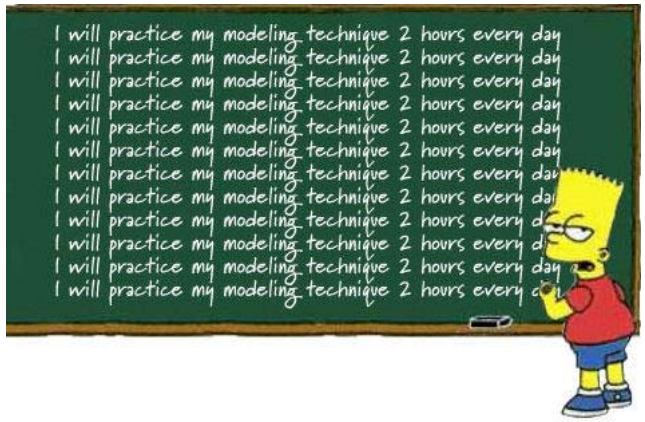
Solution: add the exact directive on links

```
<li class="nav-item">
    <router-link to="/" class="nav-link" exact>Home</router-link>
</li>
<li class="nav-item">
    <router-link to="add" class="nav-link" exact>Add Country</router-link>
</li>
<li class="nav-item">
    <router-link to="update" class="nav-link" exact>Update Country</router-link>
</li>
```

(Now only the logo still has a yellow background, b/c this links to the homepage, but IRL you will overcome this.)

Workshop

- Add a style to your active links, as described in this section.
  - Create a style for your active links
  - Add the style to the Navigation component
  - Use the exact keyword to highlight only the correct style



Navigating from code

Using the router from JavaScript, instead of HTML

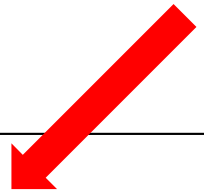
Navigating from code

- We're going to navigate to the `<CountryDetail />` component via code
- We'll use this page later on for route params
- First – create a route for it in `../router/index.js`.

```
export default new Router({
   routes: [
      // define all routes here....
      …
      {
         path: '/detail',
         name : 'detail',
         component: CountryDetail
      }
   ]
})
```

Navigate from code

- Next, define a method that uses the router in the `VacationPicker` component
- Because we injected the router in the Vue instance, we have access to the `$router` instance on components

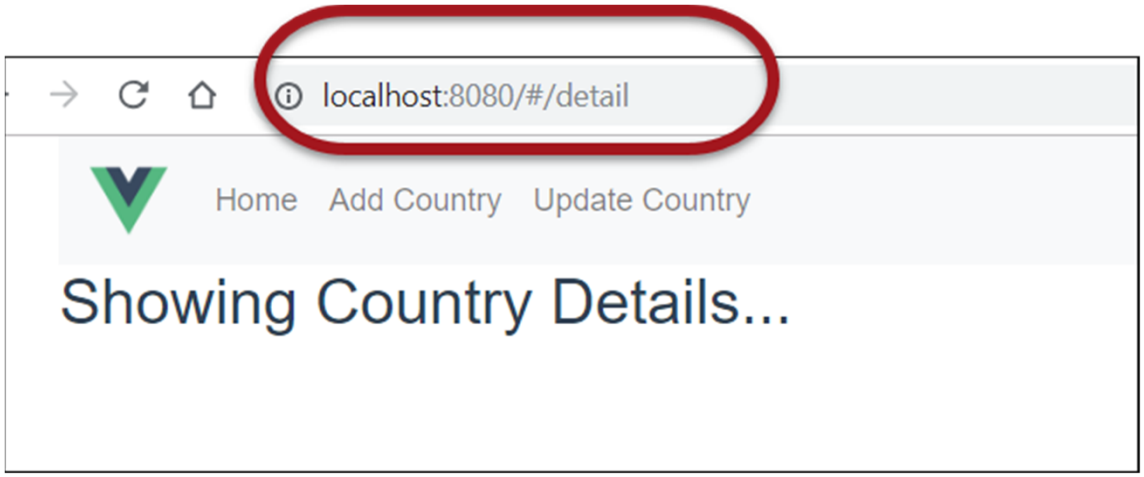


```
<li class="list-group-item"
    @click="showCountry(country)"
    v-for="country in data.countries" :key="country.id">
     …
</li>
```

```
methods:{
   // a method for navigating to the specific country from code, instead of HTML
   showCountry(country){
      console.log('navigate to ' + country.name);
      this.$router.push('/detail')
    }
}
```

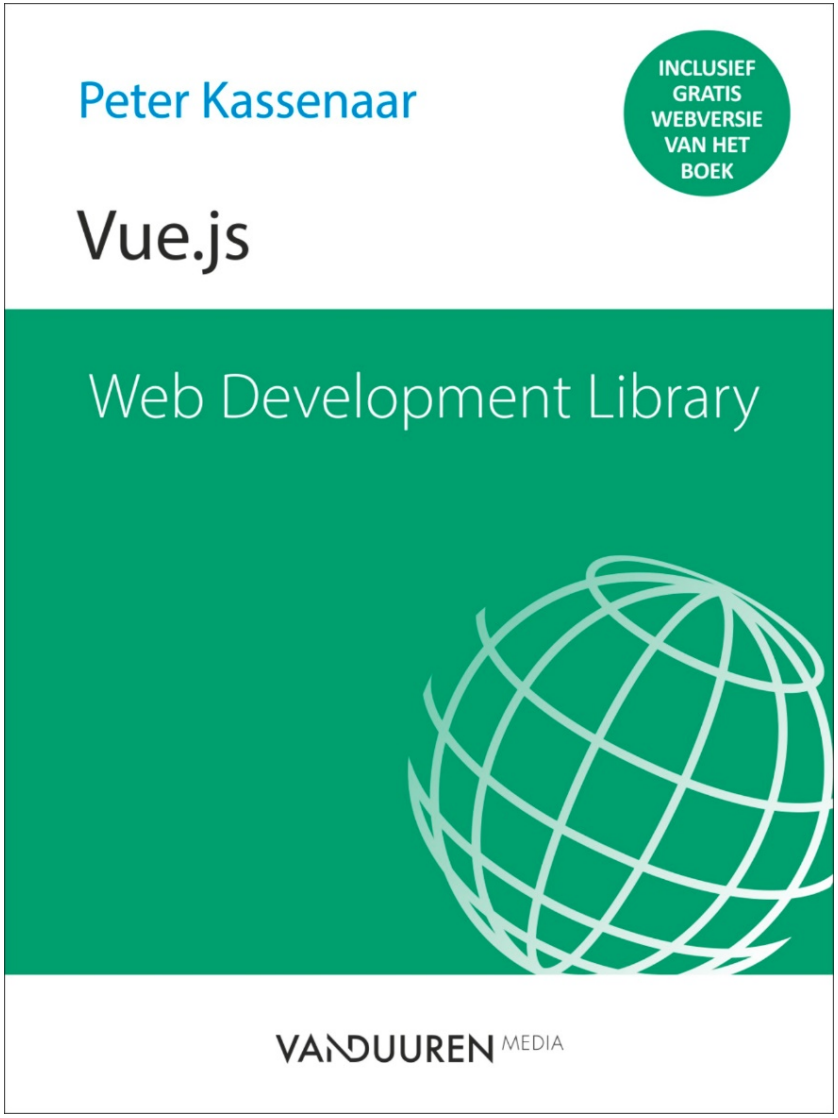
Result

- Hardcoded text for now – we're going to solve that by using route parameters.



Using Route Parameters

Passing detail parameters to the next view


Peter Kassenaar
INCLUSIEF GRATIS WEBVERSIE VAN HET BOEK
Vue.js
Web Development Library
VANDUUREN MEDIA

P.186 e.v.

So far, we're just displaying hardcoded text on the Detail page.

The goal of course, is to show data from the passed in country.

For that we use *Route Parameters*

1. Update the route

`../router/index.js` → add `/:name` and possible other parameters to the detail route

```
{
   path: '/detail/:id/:name',
   name : 'detail',
   component: CountryDetail
}
```

2. Update the <`CountryDetail /`> component, to grab the parameters from the route. We can do this for instance in a created() lifecycle hook

```
created(){
   this.id = this.$route.params.id;
   this.name = this.$route.params.name;
},
```

3. Update routerlink, to send the parameters

- Update the `<VacationPicker />` component to send the correct `id` and `name` in the URL
- Vue needs an object notation for that.
  - We can't just send a composed string like `'detail/1/Netherlands'`.

```
this.$router.push({
    name: 'detail',
    params: {
        id: country.id,
        name: country.name
    }
})
```

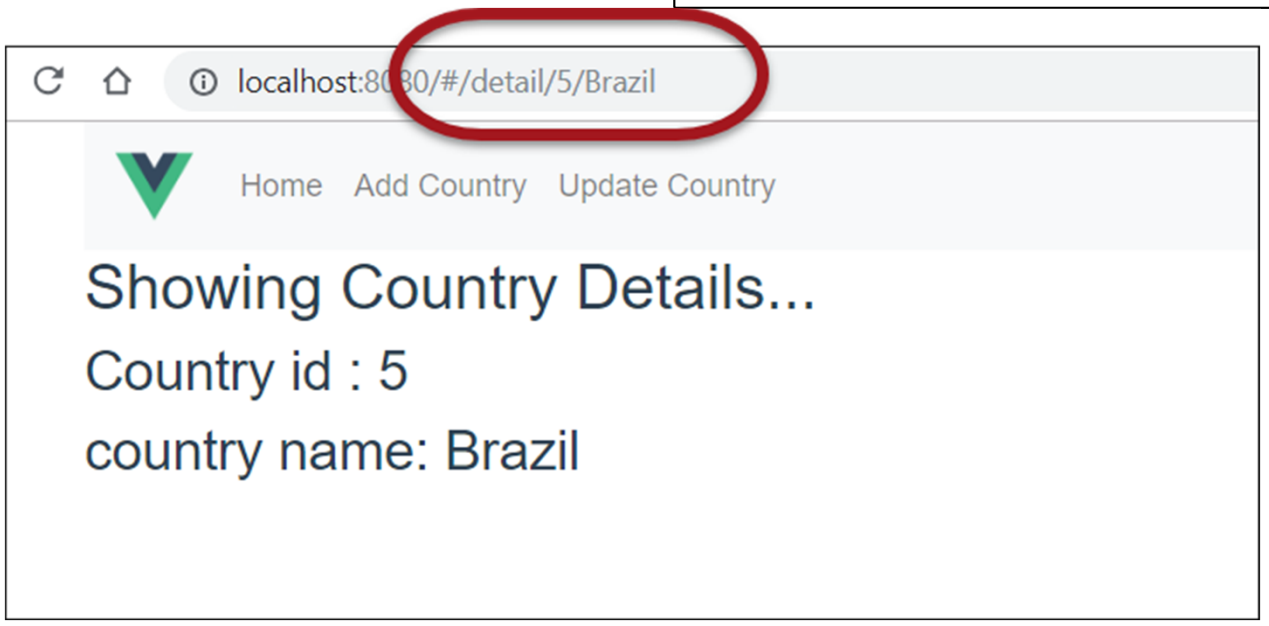
4. Show the parameter in the UI

- Start simple – just show the passed paramater in the UI.

```
created(){

   this.id = this.$route.params.id;
   this.name = this.$route.params.name;
},
```

```
<h2>Showing Country Details...</h2>
<h3>Country id : {{ id }}</h3>
<h3>country name: {{ name }}</h3>
```



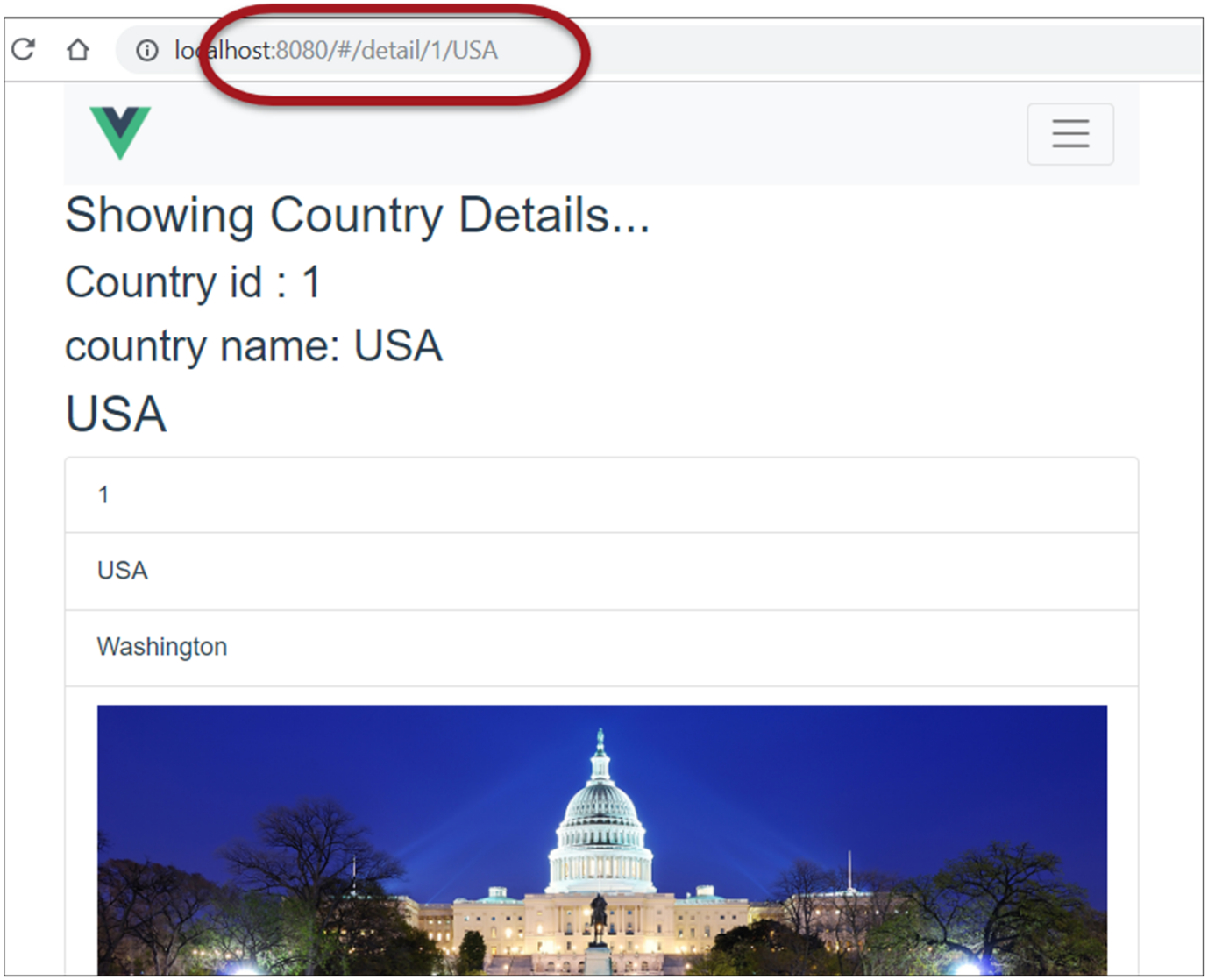
Show country data

- Grab the data (for now: local, to get the correct country)
  - We're going to fetch data from an API later on
- Look for the data based on the id

```
import data from '../data/data';
```

```
created(){
   …
   // fetch the correct country from the loaded data (+cast id-string)
    this.country = this.data.countries.find(c => c.id === +this.id);
},
```

Result


localhost:8080/#/detail/1/USA
Showing Country Details...
Country id : 1
country name: USA
USA
1
USA
Washington

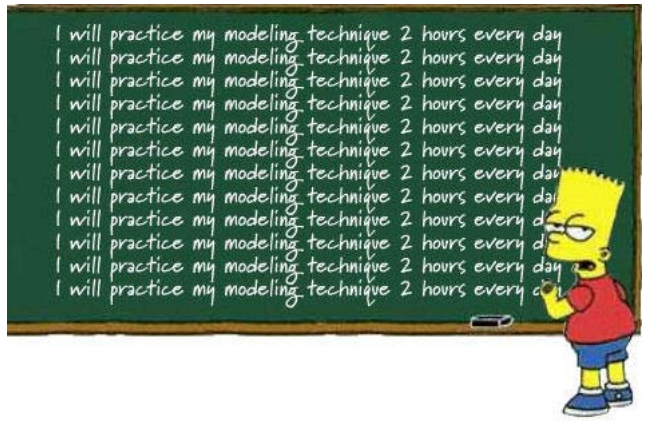
Remember:

`this.$router` gives access to the (application wide) ***router instance***

`this.$route` gives access to the ***currently selected route*** *inside a component!*

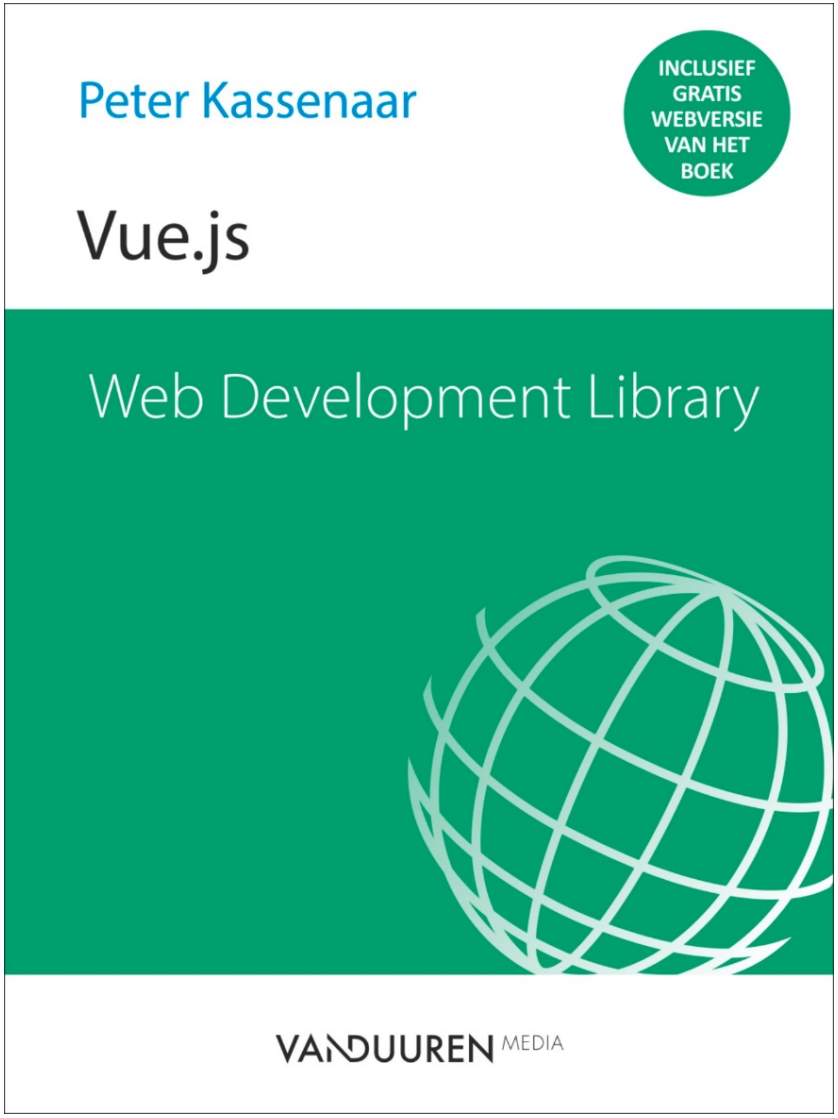
Workshop

- Add routing parameters to your own application, using the steps in this module
  - Create a parametrized route, using `/:<name>` notation
  - Use `this.$route.params` in the detail component to fetch the route parameters
  - Push a parametrized object to the `$router` when element is clicked
- Generic example: `../310-routing-parameters`



HTML 5 History mode

Getting rid of the # in the URL


Peter Kassenaar
INCLUSIEF GRATIS WEBVERSIE VAN HET BOEK
Vue.js
Web Development Library
VANDUUREN MEDIA

P.175

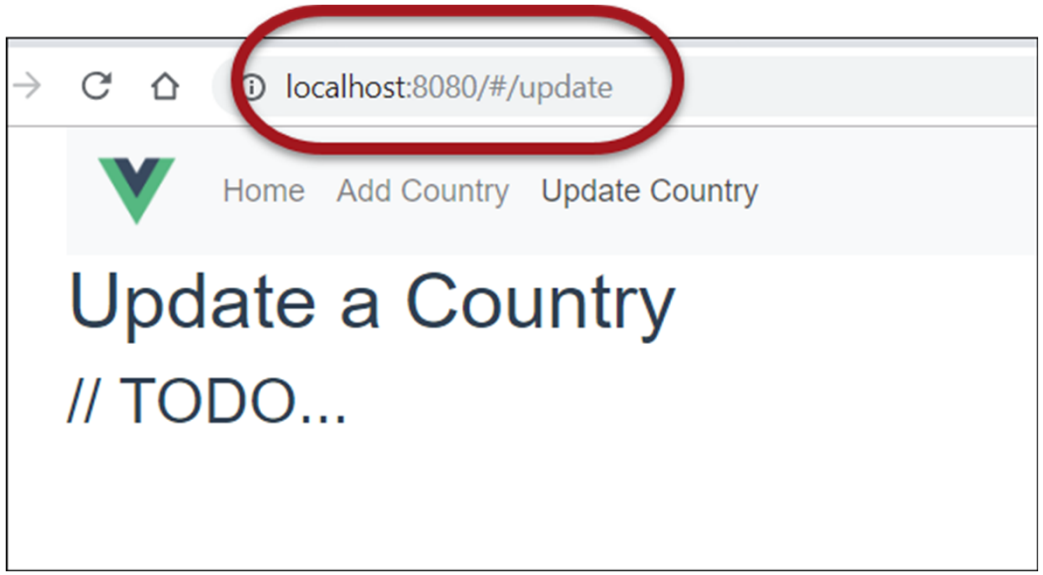
Enabling HTML 5 History mode

- No more hash sign /#/… in the URL/Address bar
- Just add a property to the router configuration
  - mode: ‘history’

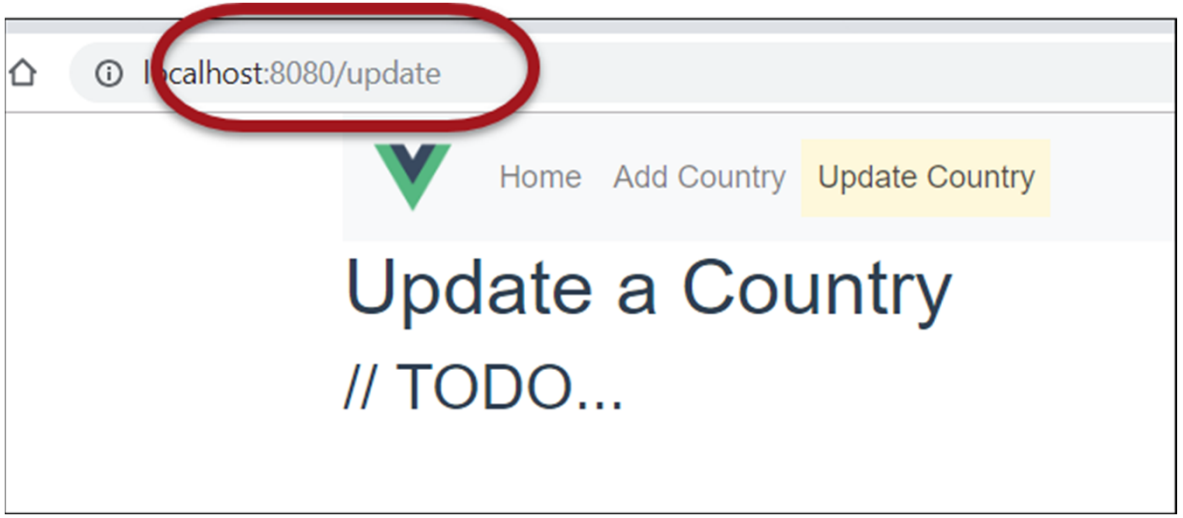
```
export default new Router({
   mode: 'history',
   routes: [
      // define all routes here....
})
```

https://router.vuejs.org/guide/essentials/history-mode.html#html5-history-mode

Before



After



Caveats on HTML History mode

- Configure your server accordingly!
  - WebPack Dev Server (used by VueJS) handles the URL correctly.
  - But other servers might actually want to fetch the requested `directory/path`!
    - IIS
    - Apache
    - Nginx
- The server should always return the `index.html` file, so Vue can handle it.
- https://router.vuejs.org/guide/essentials/history-mode.html#example-server-configurations

Check docs for your server to update configuration

Example Server Configurations

Apache

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /index.html [L]
</IfModule>
```

Instead of `mod_rewrite`, you could also use `FallbackResource`.

nginx

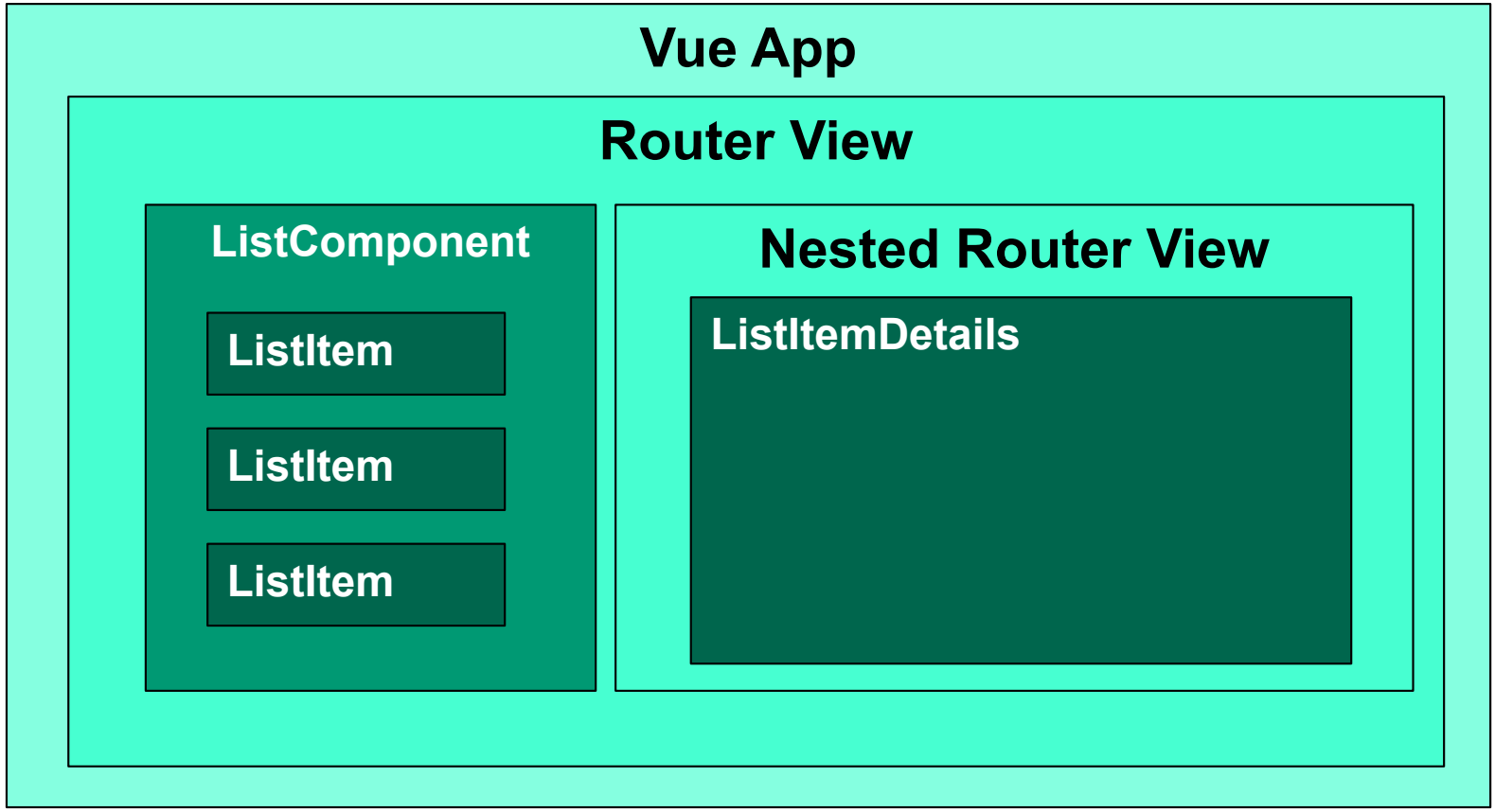
```
location / {
  try_files $uri $uri/ /index.html;
}
```

https://router.vuejs.org/guide/essentials/history-mode.html#example-server-configurations

Nested Router Views

Enabling Children: views inside views

What are nested router views?



Show router links *inside* another router view, but still update the URL

https://router.vuejs.org/guide/essentials/nested-routes.html

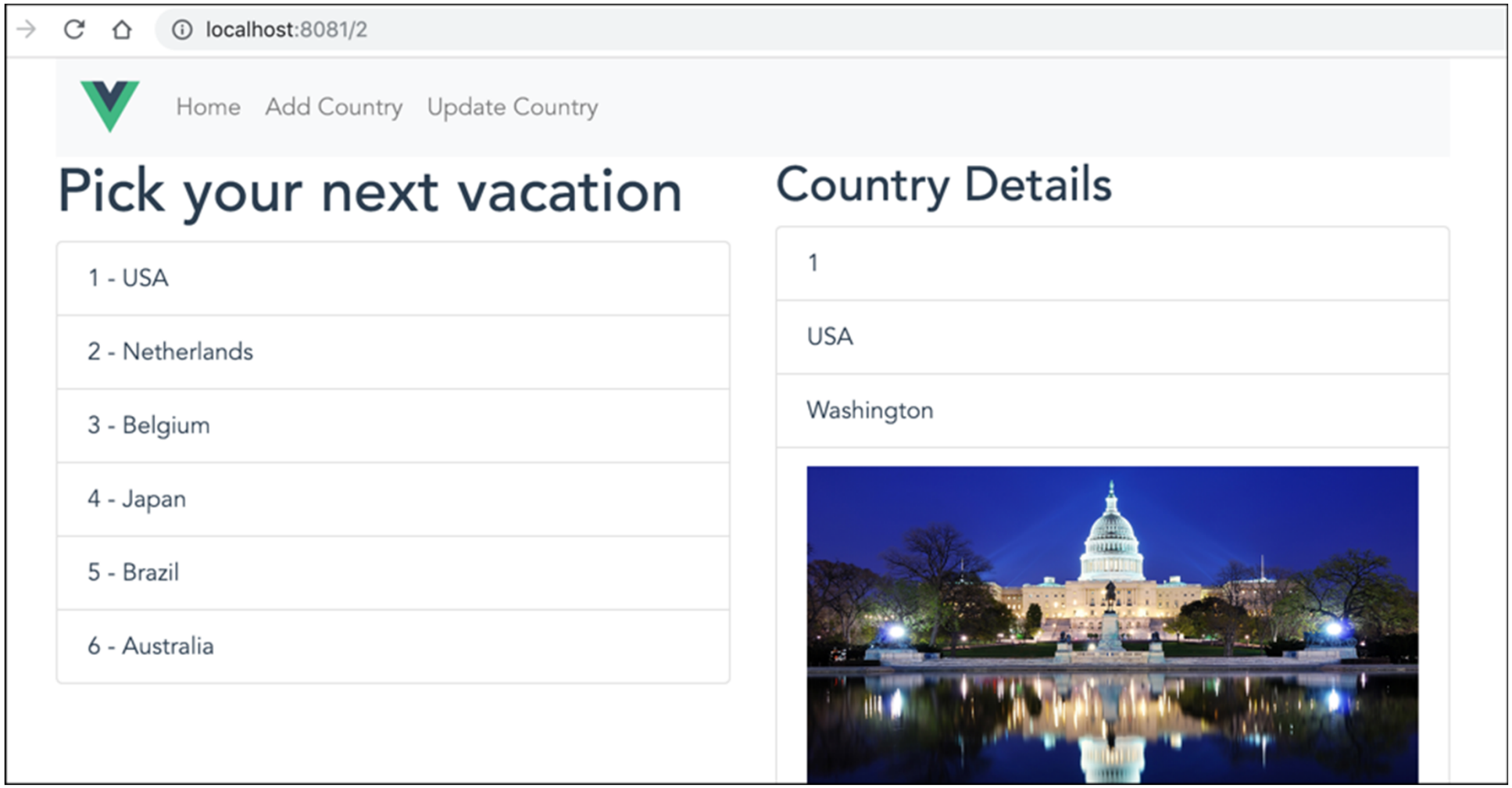
Adding a nested router

- Add a new `<router-view>` to the component you want the nested routes to appear
  - In our case: `<VacationPicker />`
- Add an array of childroutes to the main routing table
  -
```
children: [
    { <child-router-path-1> }
    { <child-router-path-2> }
]
```
- Don’t forget to remove other dynamic routes (if you have some, of course)

```
<div class="row">
    <div class="col-md-6">
        <h1>{{ header }}</h1>
        <ul class="list-group">
            <li class="list-group-item"
                @click="showCountry(country)"
                v-for="country in data.countries" :key="country.id">
            <span :id="country.id"...>
            </li>
        </ul>
    </div>
    <div class="col-md-6">
        <!--The Nested router view here-->
        <router-view></router-view>
    </div>
</div>
```

```
routes: [
    // define all routes here....
    {
        path: '/',
        name: 'home',
        component: VacationPicker,
        children: [
            {
                path: ':id',
                name: 'detail',
                component: CountryDetail
            },
        ]
    },
```

Now it works...the first time



Updating the nested view

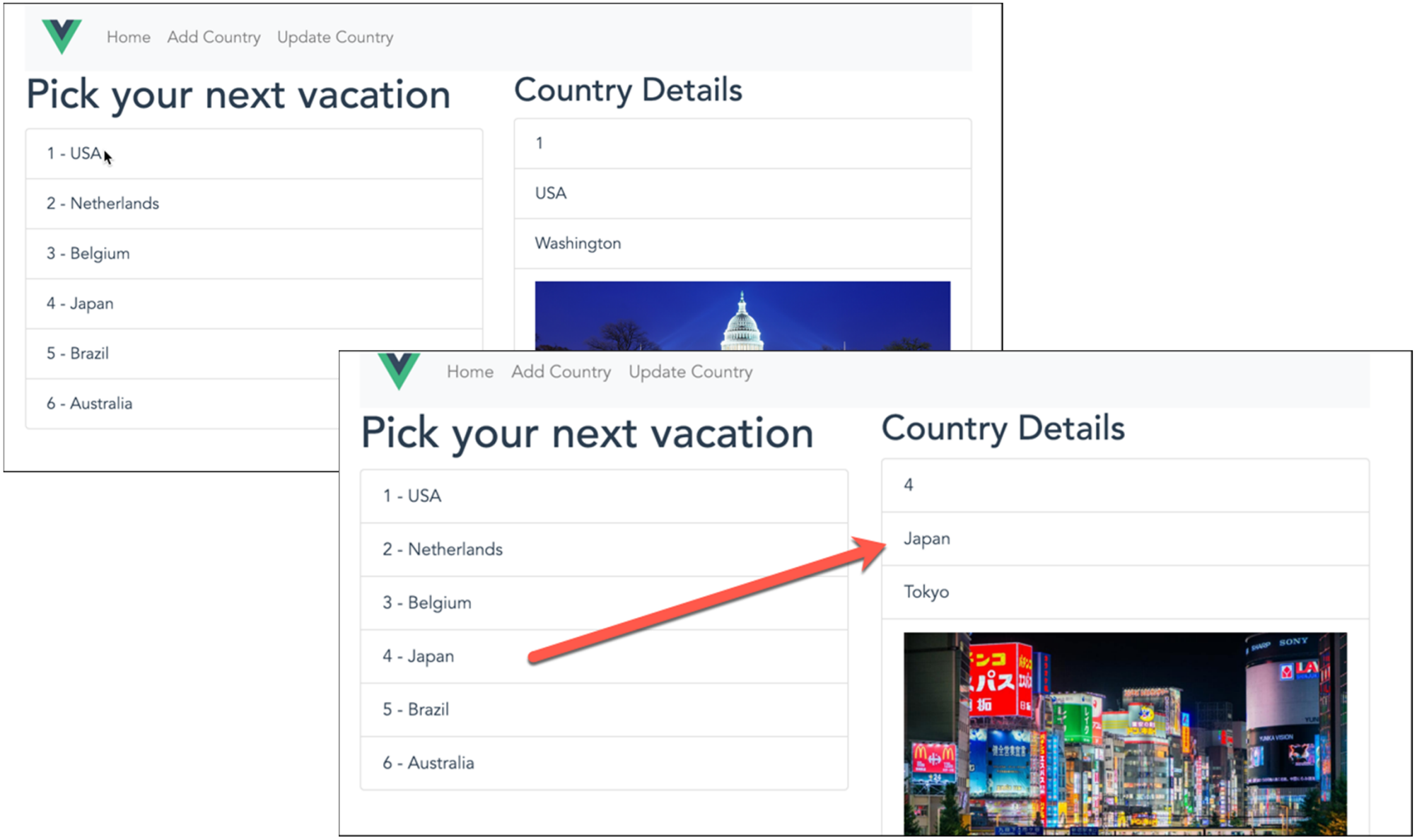
- A nested view is (by default) *not* recreated once the `$route` updates
  - Better for performance
  - You need to tell the component to watch for changes
  - Multiple solutions
- 1st: Tell the `<router-view>` to update once the route changes
  - `<router-view :key="$route.fullPath"></router-view>`
  - This works, but it is not the best for performance
  - Recreating and binding of DOM-element

Updating the nested view

- 2nd – Set a watcher on the route.
  - The component is updated on route change
  - You *need* to have an explicitly declared and initialized data variable on the component, i.e. `country : null`.

```
    data() {
        return {
            data,
            country: null
        }
    },
    watch: {
        '$route'(to, from) {
            console.log(to, from); // logging, so you can see what's going on
            this.id = to.params.id;
            this.country = this.data.countries.find(c => c.id === +this.id);
        }
    },
```

https://router.vuejs.org/guide/essentials/dynamic-matching.html#reacting-to-params-changes


Home
Add Country
Update Country
Pick your next vacation
1 - USA
2 - Netherlands
3 - Belgium
4 - Japan
5 - Brazil
6 - Australia
Country Details
1
USA
Washington
Home
Add Country
Update Country
Pick your next vacation
1 - USA
2 - Netherlands
3 - Belgium
4 - Japan
5 - Brazil
6 - Australia
Country Details
4
Japan
Tokyo

Updating nested view

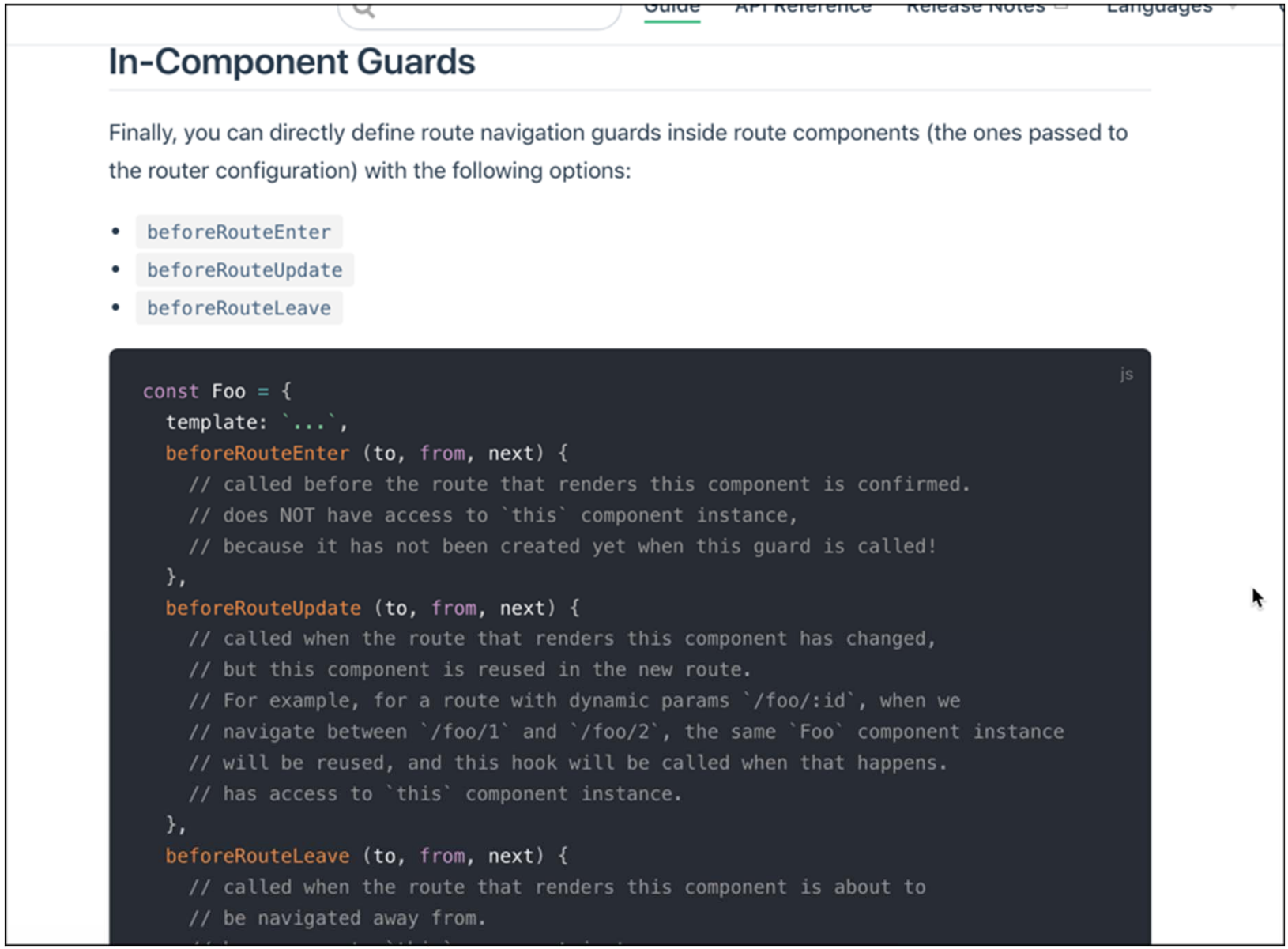
- 3rd – Use the Router ‘ in-component guard’

  `beforeRouteUpdate(to, from, next)`

  - Introduced in Vue 2.2+
  - Don’t forget to call `next()` at the end!
  - https://router.vuejs.org/guide/advanced/navigation-guards.html#in-component-guards

```
// Update nested view, Method 3 - use the beforeRouteUpdate() method,
// introduced in Vue 2.2.
beforeRouteUpdate(to, from, next) {
    this.id = to.params.id;
    this.country = this.data.countries.find(c => c.id === +this.id);
    next();
},
```

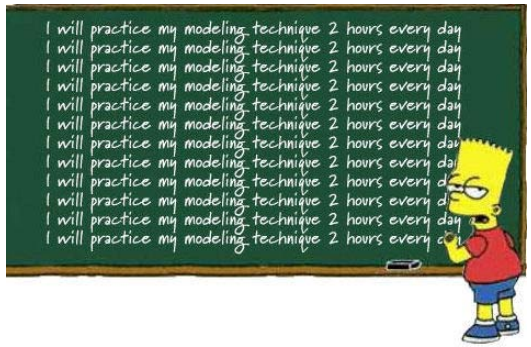
More info on In-Component guards



https://router.vuejs.org/guide/advanced/navigation-guards.html#in-component-guards

Workshop

- Use your own application, or start from `../310-route-parameters` (the *previous* example)
- Enable HTML5 History mode
- Add a nested `<router-view>`, as shown in the previous slides
  - If a country is selected, the view is updated in the same page.
  - Update the application, so that not only the `id` is shown in the URL, but also the country name.
  - Optional – update the master view, so that no `@click` event handler is used, but dynamic binding on the `<router-link :to="…">` parameter
- Generic example: `../320-nested-routing`



Checkpoint

- You know how to add routing to a project
- You can create a routing table in an `index.js`, or `router.js`
- You can use `<router-view>` and `<router-link>`
- You can add styles and conditional styles to routes
- You know how to navigate from code, by using `$router.push(…)`
- You can use route parameters (`/:<some-name>`) to create dynamic routes
- You know how to enable HTML5 history mode on the router
- You are able to use nested router views