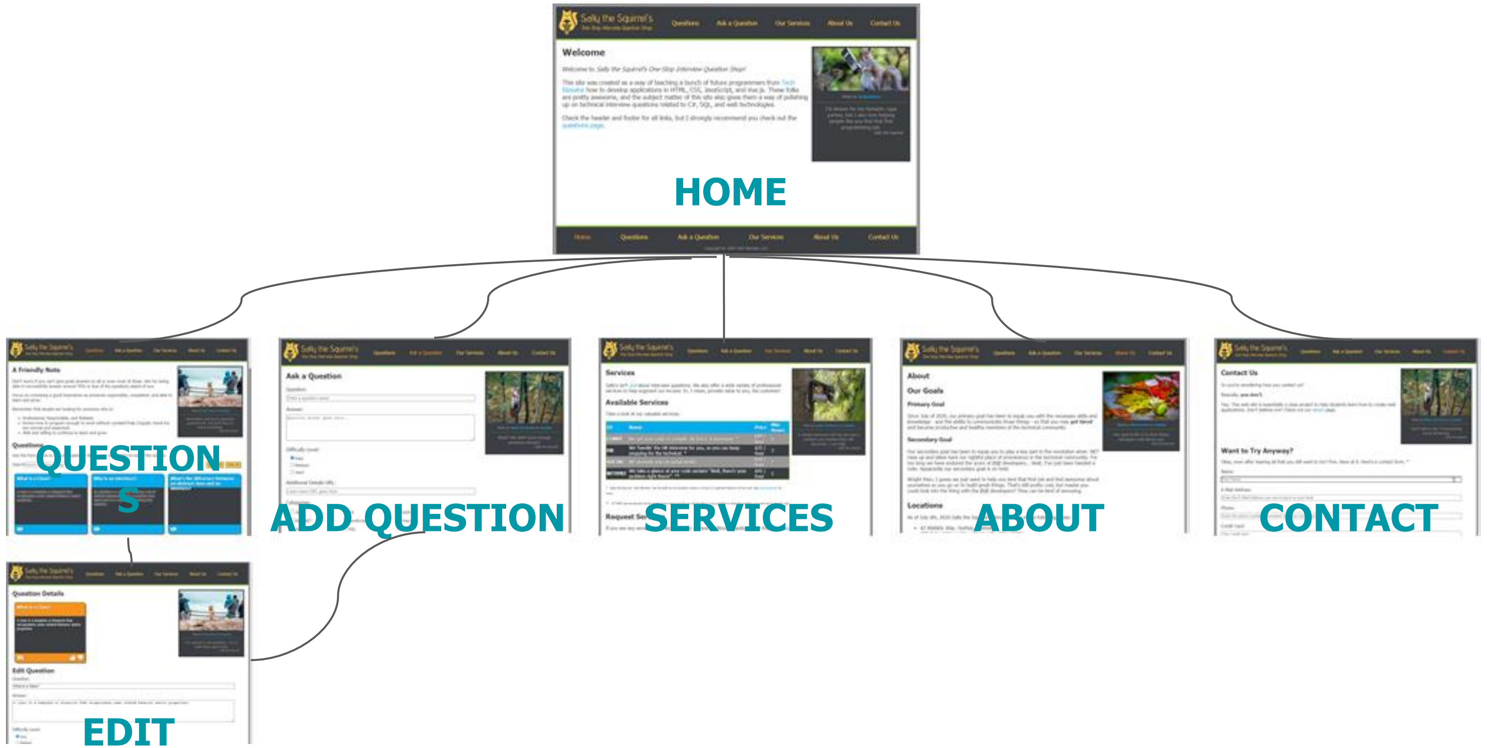
Vue Routing

Module 3: 15

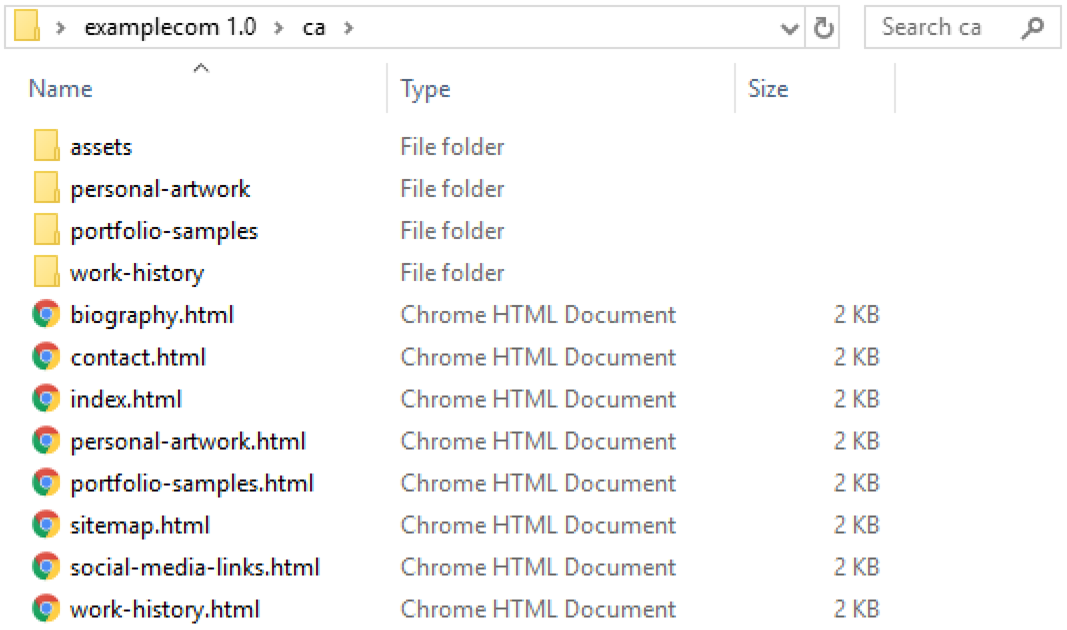
Objectives

1. The Problem with SPA
2. Vue Router
3. Views vs Components
4. Configuring Routes
5. <router-view>
6. Accessing Route Parameters
7. <router-link>
8. Redirects
9. Handling Not Found (404)
10. Navigation Guards
11. Vue LifeCycle and LifeCycle Hooks
12. Watchers

Web Navigation



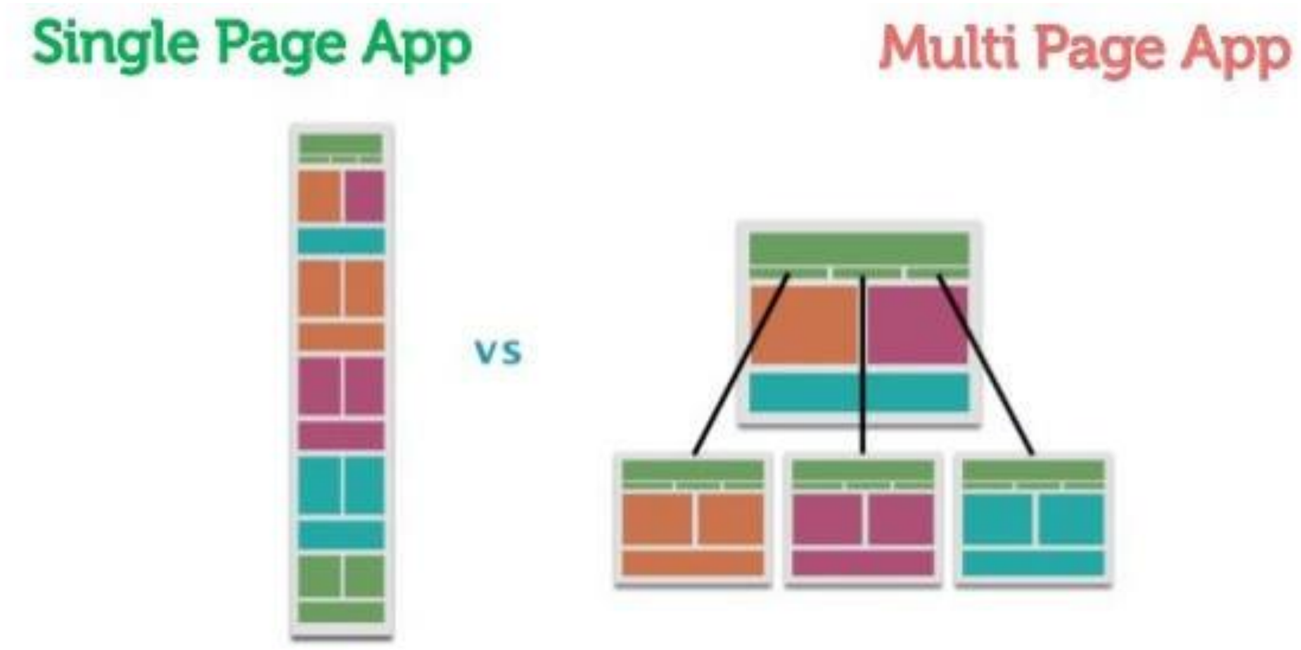
Web Navigation



Name	Type	Size
assets	File folder	
personal-artwork	File folder	
portfolio-samples	File folder	
work-history	File folder	
biography.html	Chrome HTML Document	2 KB
contact.html	Chrome HTML Document	2 KB
index.html	Chrome HTML Document	2 KB
personal-artwork.html	Chrome HTML Document	2 KB
portfolio-samples.html	Chrome HTML Document	2 KB
sitemap.html	Chrome HTML Document	2 KB
social-media-links.html	Chrome HTML Document	2 KB
work-history.html	Chrome HTML Document	2 KB

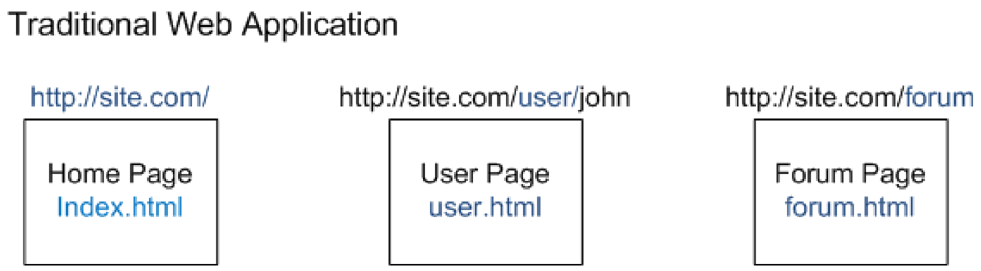
Single Page Application (SPA)

Serves a *single HTML page* and all other “pages” in the site are generated using DOM manipulation of the original HTML with new data retrieved from web API using JavaScript.

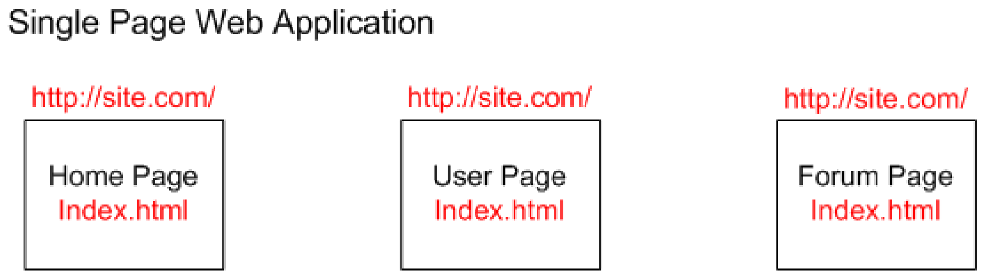


The Problem with Single Page Applications (SPA)

Browsers tracks individual pages by the HTTP Request/Responses and the resources they return. In a traditional web application each individual page is a separate resource (html/jsp) and has its URL.



In a SPA site only one resource is returned, the index.html, so there is only 1 URL for the browser to track. Since most websites have multiple pages, how can the browser track what page the user is a one, when they all use the same resource and different pages are created by API calls and DOM manipulation of that resource?



.

The Problem with SPA continued...

Answer: the browser can’t track what internal page the user is on since it sees all the pages as a single URL

The problems this creates:

1. The *Back button doesn’t work for internal pages*, instead they go “back” to the last resource, which is most likely a different site.
2. The *Refresh button doesn’t refresh the internal page*, instead it repeats the browsers last HTTP Request, which reloads the index.html, or the starting page.
3. *Browser History doesn’t track internal pages*, instead the entire visit to the site is represented in history as a single request for the initial index.html
4. *Users can’t bookmark an internal page*. If they do and come back the browser correctly sends the request, but the server does not know what the URL means so can’t load the bookmarked page, and can only respond with a 404.

SPA Solution - The Router

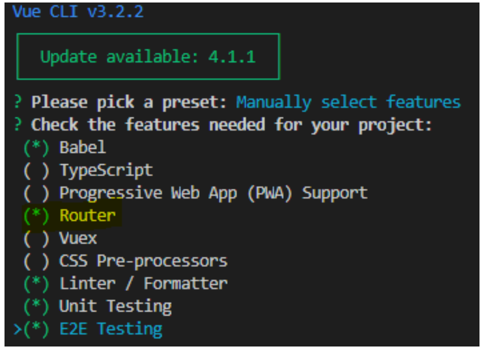
SPA Frameworks like Vue, Angular, and React solve this problem by providing a Router that allows fine-grained navigation, bookmarking of pages, and traditional use of a URL.

Vue provides the **Vue Router** for this purpose.   Documentation

The router allows definition of URL routes for “internal” pages of a Single Page Application giving users a traditional web application experience.   This works by not loading the routes on the server, but by the server still responding with the index.html when a bookmark, back button, or refresh button are used. The Vue Framework then uses the URL to load the correct View and load the data on the client side.

Adding the Vue Router to a project

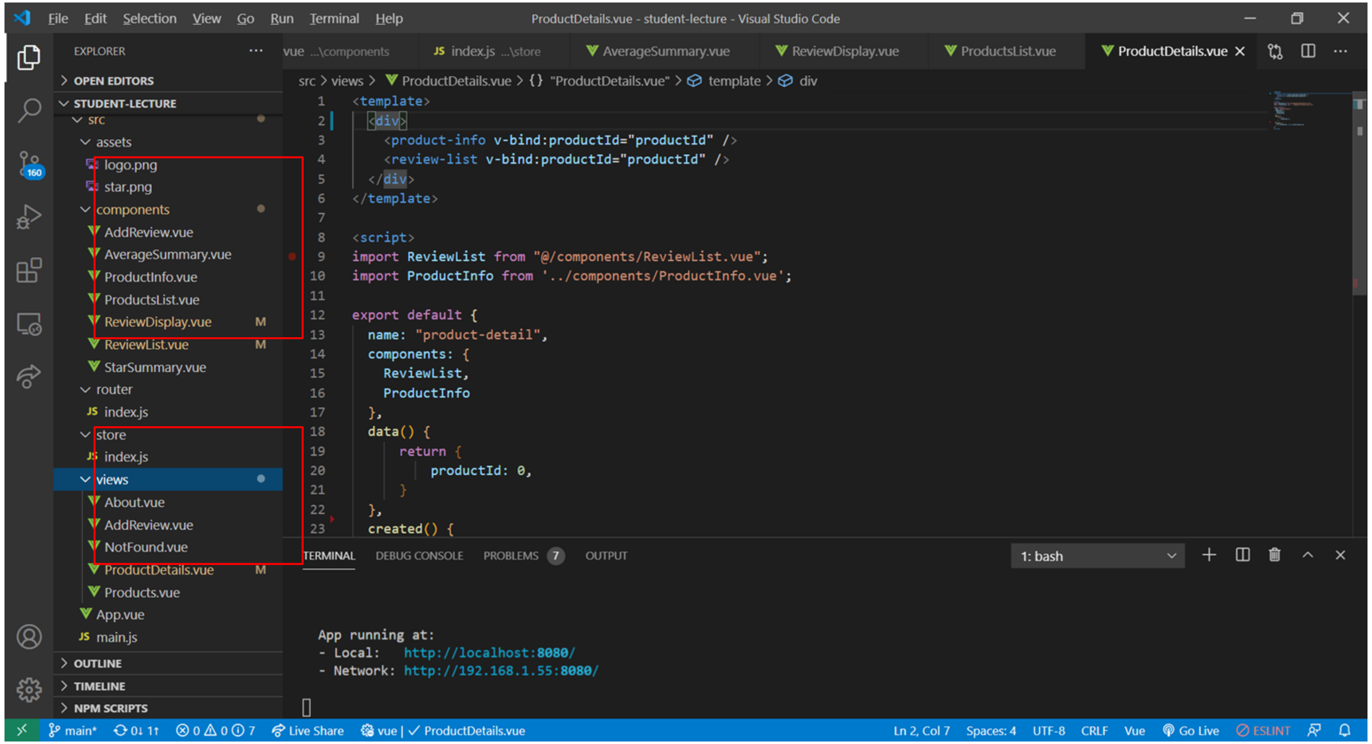
To add to a new project: when you run ***vue create*** select “Router” as an option.



```
Vue CLI v3.2.2
┌─────────────────────────┐
│  Update available: 4.1.1  │
└─────────────────────────┘
? Please pick a preset: Manually select features
? Check the features needed for your project:
 (*) Babel
 ( ) TypeScript
 ( ) Progressive Web App (PWA) Support
 (*) Router
 ( ) Vuex
 ( ) CSS Pre-processors
 (*) Linter / Formatter
 (*) Unit Testing
>(*) E2E Testing
```

To add to a an existing project. Go to the project directory with the **package.json** file in terminal. Run ***npm install***, if you have not already done so. Then run command: ***vue add router***

Vue.js Navigation



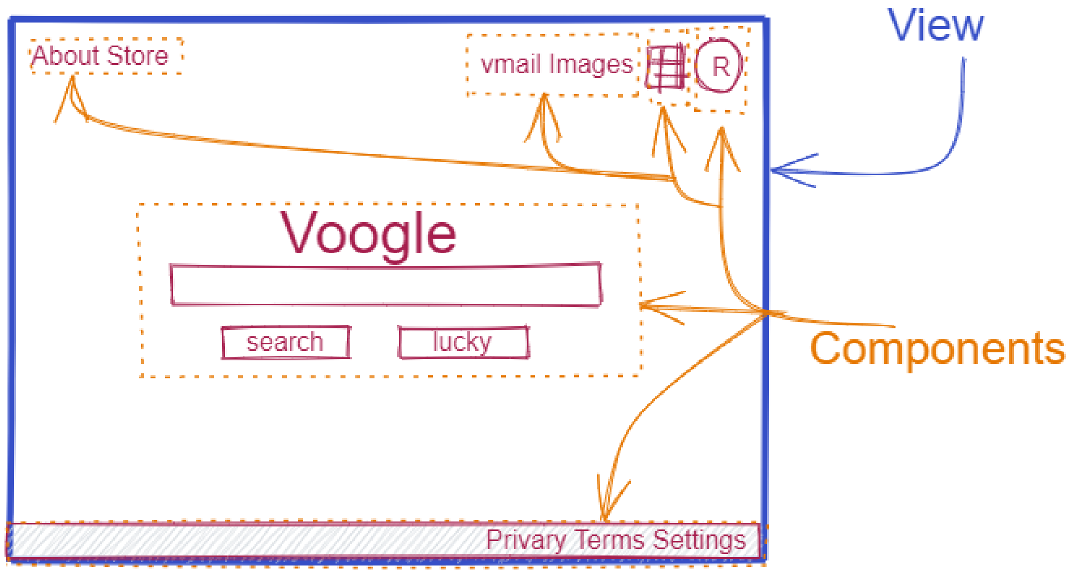
```
<template>
  <div>
    <product-info v-bind:productId="productId" />
    <review-list v-bind:productId="productId" />
  </div>
</template>

<script>
import ReviewList from "@/components/ReviewList.vue";
import ProductInfo from '../components/ProductInfo.vue';

export default {
  name: "product-detail",
  components: {
    ReviewList,
    ProductInfo
  },
  data() {
      return {
          productId: 0,
      }
  },
  created() {
```

```
App running at:
- Local:   http://localhost:8080/
- Network: http://192.168.1.55:8080/
```

Views vs Components

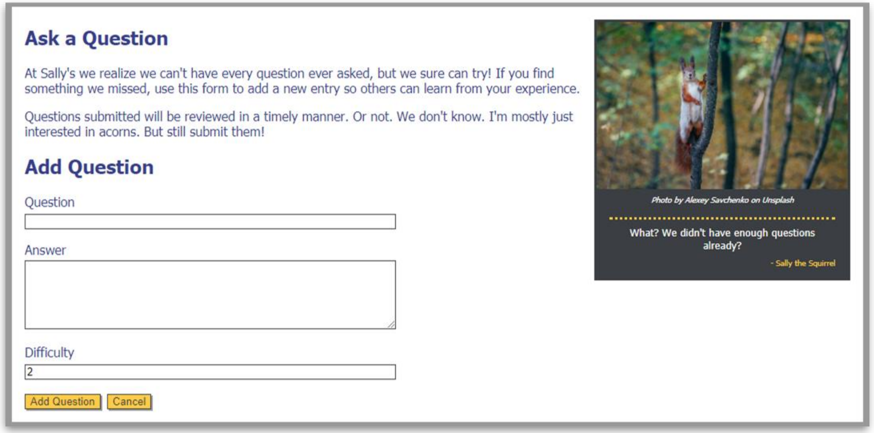
From a code perspective Views and Components are the same. The difference is organizational and how they are used with routing.

Views: represent things we will route to (aka web pages)
Components: reusable parts that will be used on Views



View vs. Components

VIEWS


Ask a Question

At Sally's we realize we can't have every question ever asked, but we sure can try! If you find something we missed, use this form to add a new entry so others can learn from your experience.

Questions submitted will be reviewed in a timely manner. Or not. We don't know. I'm mostly just interested in acorns. But still submit them!

Add Question

Question

Answer

Difficulty

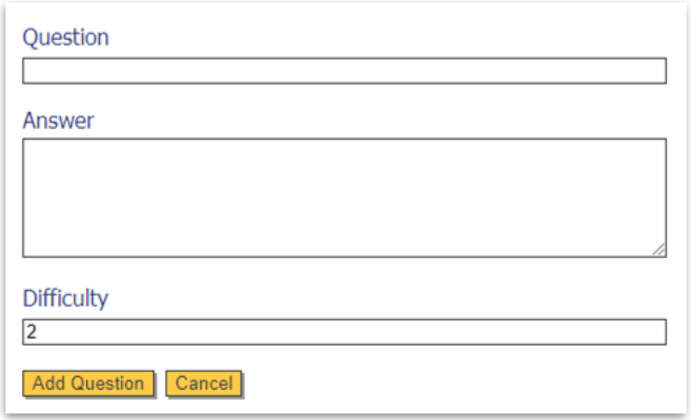
2

Photo by Alexey Savchenko on Unsplash

What? We didn't have enough questions already?

- Sally the Squirrel

COMPONENTS


Question

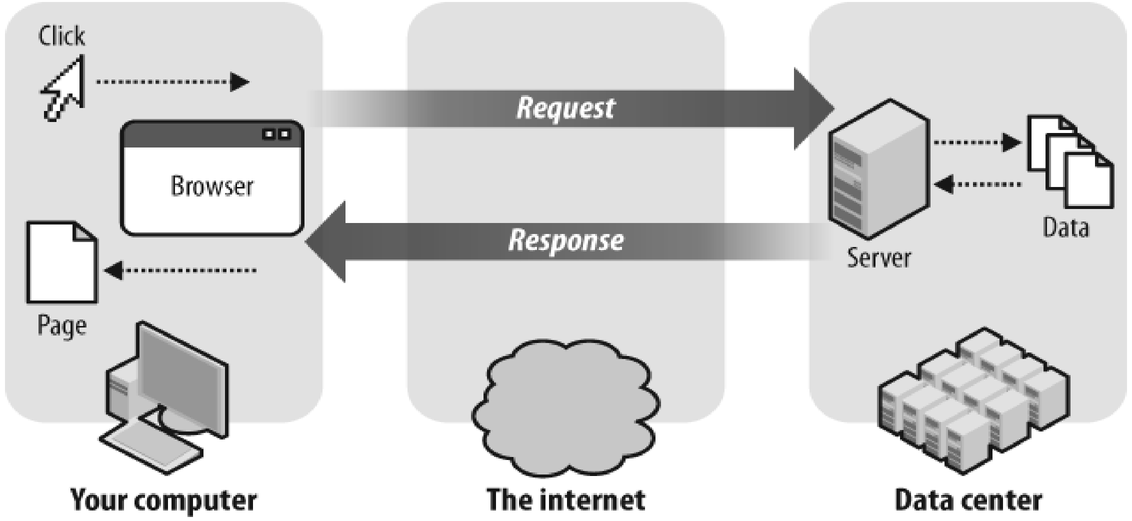
Answer

Difficulty

2

What is Routing?

- Routing allows users to be redirected to a certain component via a URL.
- Remember MVC Spring RequestMappings? Similar idea.



The src/router/index.js file overview

To define a route, an index.js file is needed.

- There is some boiler plate code beyond the scope of this class, you can carry those over for now, they are highlighted in blue.

- Our focus will be on the sections in red, which we will define.

```
import Vue from 'vue'
import Router from 'vue-router'
import Home from './views/Home.vue'
import About from './views/About.vue'
import NotFound from './views/NotFound.vue'

Vue.use(Router)
const routes = [
  {
    path: '/',          // Required
    name: 'home',       // Recommended, but not required
    component: Home    // Required
  },
  {
    path: /about',
    name: 'about',
    component: About
  }
]
const router = new VueRouter({
  routes
})
export default router
```

The index.js file in router folder: importing views

We need to first define the components a user can potentially be routed to, this can be achieved through imports:

```
import Home from ‘../views/Home.vue'
import About from ‘../views/About.vue'
import Books from ‘@/views/Books.vue'
```

Note that in this example the components are in a folder called views, this should make no difference, Home, About, and Books are VUE components.

The router/index.js file: importing components.

Next, we define the routes, these physically map the view components we imported.

```
import Home from './views/Home.vue'
import About from './views/About.vue'
import Books from ‘@/views/Books.vue'
```

```
routes: [
  {
    path: '/',
    name: 'home',
    component: Home
  },
  {
    path: '/about',
    name: 'about',
    component: About
  },
  {
    path: '/books',
    name: 'books',
    component: Books
  }
]
```

Each route is a JSON object, comprised of three key value pairs:

- **path**: what the user types into the URL
- **name**: This is how we will refer to the route within App.vue
- **component**: The component that was imported in, that the user will be redirected to

Router Configuration

The routes are defined in an array of JavaScript objects that define the path, name, and component to load.

```
routes: [
    {
      path: "/about",
      name: "about",
      component: About
    }  ]
```

- **path** is how the route will appear in the URL
- name is the routes name, like a variable name, that we use in the router-link
- component identifies the component to load. The component **name** matches the **name** given to the component when importing it into the script.

```
import About from './views/About.vue'
```

Dynamic Routes

Dynamic routes are URLs that contain variable data, such as a product id or a user id.   In SpringMVC we referred to these as PathVariables: `/users/10`

In route configuration a dynamic route can be used with a placeholder variable name identified by a colon (:)

```
    {
        path: "/user/:id",
        name: "user",
        component: User
    }
```

Dynamic Routing : Defining the Router Links

We can now define router links

```
<tr v-for="user in users" :key="user.id">
      <td><router-link :to="{name: 'users', params: {id: user.id}}">{{user.id}}</router-
link></td>
      <td>{{user.name}}</td>
</tr>
```

Template section

```
  data() {
    return {
      users: [
        {
          "id": 1,
          "name": "Leanne Graham"
        },
        {
          "id":2,
          "name": "Ervin Howell"
        }
      ]
    }
```

Script section

Here we have a v-for that will iterate through every object in the users array, each time it does so it generates a new router-link with its respective id value.

Two links are generated:

- /users/1
- /users/2

Accessing a parameter from the route

Route Parameters can be accessed using the **$route.param** object and the parameter name given in the configuration and router-link.

```
   {
        path: "/user/:id",
        name: "user",
        component: User
     }
```

```
http://localhost:8080/user/2
```

`this.$route.params.id` will be equal to 2

Creating Navigation with <router-link>

To add a navigation to another page the **<router-link>** (Documentation) tag can be used. In Vue this replaces the <a href> tag and instead of sending the request to the server will redirect to the Vue Router to be handled.

The router-link can be used with traditional routes, like above, similar to how the routes in SpringMVC @RequestMapping worked:

```
<router-link to="/about">About</router-link>
```

However, we will generally want more control over the router-link and can bind it to an object in our router configuration. This will give us one place to change the route, the router configuration, if our routes change.

Router-Link

The Router-Link can be bound to a router configuration by providing a JavaScript object with a name key that matches the routes name key in the router configuration.

Syntax: <router-link :to=”{ name: ‘route_name’ }”

```
<router-link :to="{ name: 'about' }" >About</router-link>
```

Optionally, the router link can specify a HTML tag to be rendered as:

```
          <router-link :to="{ name: 'about' }" tag=”li” >About</router-link>
```

Sending Parameters with router-link

The route object in a bound router-link can include a parameter that will be passed as part of the URL.

Route Configuration:

```
{
   path: "/user/:id",
   name: "user",
   component: User
 }
```

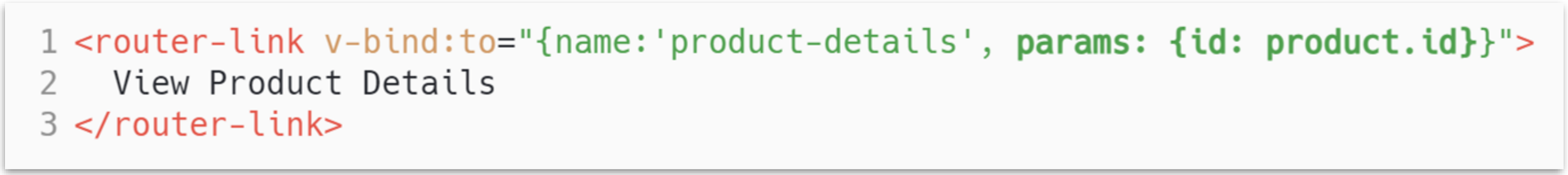
In the router-link params object, the key (id) is the name given to the parameter in the route configuration. The value (user.id) is the value we want to send for the parameter.

Router Link with Parameter

```
<router-link :to="{name: 'user', params: {id: user.id}}">
```

If user.id = 2 then the resulting URL will be: `/user/2`

Router-link with parameters – another example



```
<router-link v-bind:to="{name:'product-details', params: {id: product.id}}">
  View Product Details
</router-link>
```

Routing

SomeComponent.vue

```
<router-link v-bind:to="{name:'product-details', params: {id: product.id}}">
  View Product Details
</router-link>
```

router/index.js

```
const routes = [
  {
    path: '/',
    name: 'products',
    component: Products,
  },
  {
    path: '/products/:id',
    name: 'product-details',
    component: ProductDetails,
  },
  {
    path: '*',
    name: 'not-found',
    component: NotFound,
  }
]
```

ProductDetails.vue

```
<script>
export default {
  name: "product-detail",
  data() {
    return {
      productId: 0,
    }
  },
  created() {
    this.productId = this.$route.params.id;
  }
};
</script>
```

Router-Link: exact

The default behavior for router-link is to match the route inclusively. For example, given the route: `<router-link to="/a">` then both `/a/` or `/a` will match.

So if we want to route to the root of the site: `<router-link to="/">` then all routes ending in /, like `/a/` will match and route to the root.

Exact mode forces the router link to only route to the location if the pattern matches exactly. To enable exact matching the keyword **exact** can be added to the <router-link> tag as an attribute.

```
<router-link :to="{ name: 'home' }" tag="li" exact>Home</router-link>
```

Redirects

Redirect routes can also be configured. A redirect route is one that when visited redirects to a different route to be used.

```
routes: [
    {
      path: "/",
      redirect: { name: "users" }
    },
    {
      path: "/users",
      name: "users",
      component: Users
    }
```

When a user goes to the root route "/" in their browser, the redirect will automatically send them to the route named in the redirect, users.

Vue Lifecycle Hooks

As a Vue component is used it goes through a set lifecycle of activity including a time when it is setup, compiles the template, mounts the HTML to the DOM, updates the DOM, etc. During each of these stages it calls a method called a **lifecycle hook**. Code can be added to these methods to run custom code at each of these specific stages.   Documentation

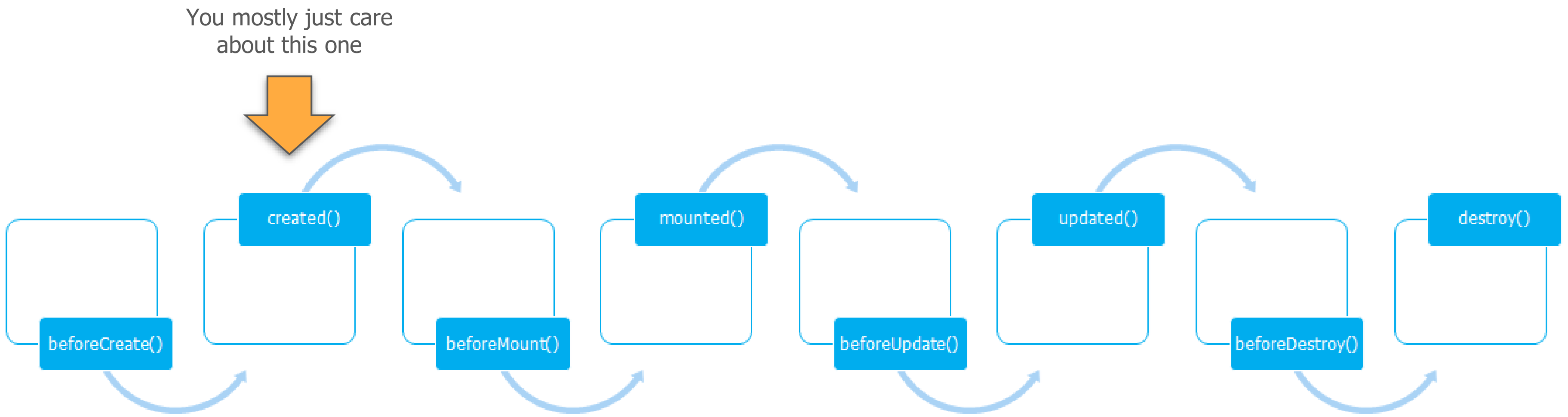
To add code for a lifecycle hook the method for it is added to the view model along with data(), methods:, computed:, etc.

```
data() {
   return {
   }
 },
created() {

 },
```

<u>Common Lifecycle Hooks</u>

created - when the page is created
mounted - when the DOM is loaded
beforeUpdate - before a DOM update is performed
updated - after a DOM update is completed
destroyed - after the page is destroyed

Life cycle events


You mostly just care about this one
created()
mounted()
updated()
destroy()
beforeCreate()
beforeMount()
beforeUpdate()
beforeDestroy()