

# Module 3

## VUE WEB SERVICES - GET

JAVA – MODULE 3, DAY 16

ELEVATE  YOURSELF





Review



# RECAP



Review



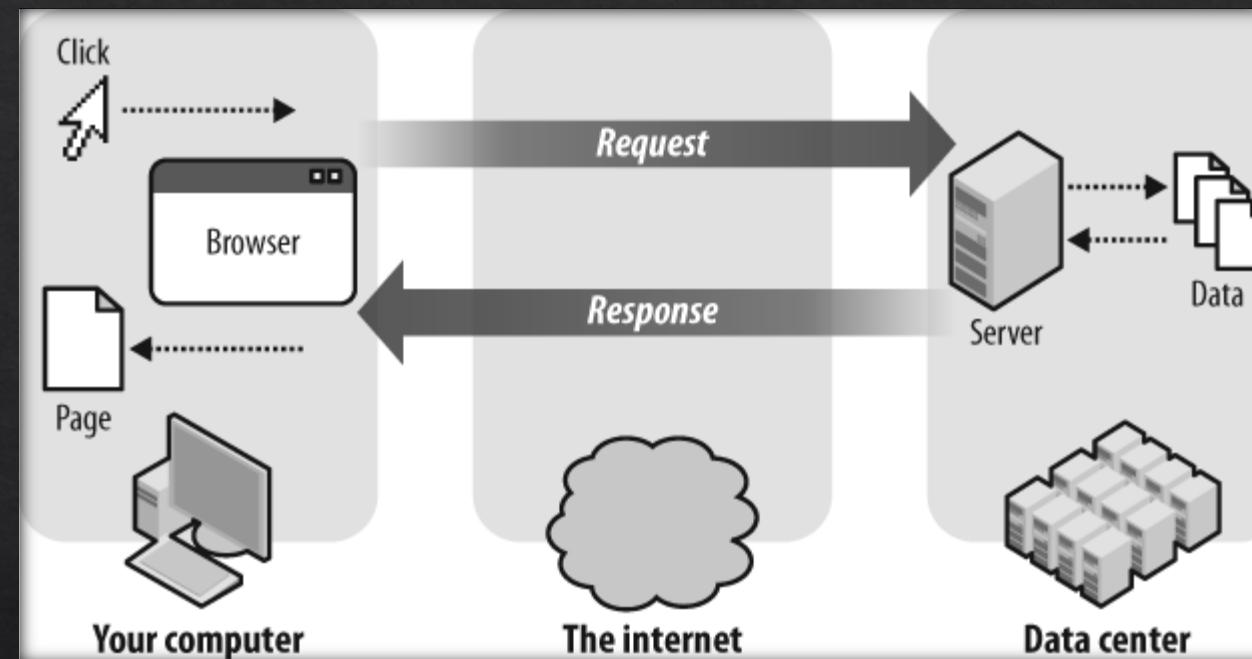
# ROUTING



Review



# WEB NAVIGATION





# VUE.JS NAVIGATION

Review





Review



# VUE.JS NAVIGATION

The screenshot shows a Visual Studio Code interface with a dark theme. The left sidebar contains a tree view of project files under 'STUDENT-LECTURE'. The 'views' folder is selected, showing components like About.vue, AddReview.vue, NotFound.vue, ProductDetails.vue, Products.vue, App.vue, and main.js. The main editor area displays the code for ProductDetails.vue:

```
<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() {
  }
}</script>
```

The terminal at the bottom shows the application running at http://localhost:8080/ and http://192.168.1.55:8080/.

Ln 2, Col 7 Spaces: 4 UTF-8 CRLF Vue Go Live ESLINT



Review



# VIEWS 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:



Note by Sally the Squirrel on Display!

What? We didn't have enough questions already?

Sally the Squirrel

## COMPONENTS

**Question**:

**Answer**:

**Difficulty**:



# INSTALLING ROUTING

Review



- **vue create my-project**

or



- **vue add router**





# DEFINING ROUTES

Review



```
1 import Home from '../views/Home.vue'
2 import About from '../views/About.vue'
3 import NotFound from '../views/NotFound.vue'
4
5 const routes = [
6   {
7     path: '/',
8     name: 'Home',           // Required
9     component: Home        // Recommended, but not required
10    },
11   {
12     path: '/About',
13     name: 'About',
14     component: About
15    },
16   {
17     path: '*',
18     name: 'NotFound',
19     component: NotFound
20    }
21 ];
```



Review



# DEFINING ROUTES

```
1 import Vue from 'vue'
2 import VueRouter from 'vue-router'
3
4 import Home from '../views/Home.vue'
5 import About from '../views/About.vue'
6 import Services from '../views/Services.vue'
7
8 Vue.use(VueRouter)
9
10 const routes = [
11   {
12     path: '/',
13     name: 'Home',
14     component: Home
15   },
16   {
17     path: '/About',
18     name: 'About',
19     component: About
20   },
21   {
22     path: '/Services',
23     name: 'Services',
24     component: Services
25   }
26 ]
27
28 const router = new VueRouter({
29   mode: 'history',
30   base: process.env.BASE_URL,
31   routes
32 })
33
34 export default router
35
```



# VUE INSTANCE

Review



```
import Vue from "vue";
import App from "./App.vue";
import router from "./router";

Vue.config.productionTip = false;

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





# WILDCARD ROUTES

Review



```
16  {
17    path: '*',
18    name: 'NotFound',
19    component: NotFound
20 }
```



# ROUTER-VIEW

Review



```
1 <template>
2   <div id="app">
3     <AppHeader />
4     <router-view /> <!-- The currently active view will be presented here -->
5     <app-footer />
6   </div>
7 </template>
```



# ROUTER-LINK

Review



```
1 <small>
2             * - this form is intended to demonstrate different input types.
3     Do not submit confidential information to untrusted sources.
4     See <router-link to="/About">site disclaimer</router-link> for more info.
5 </small>
```

```
1 <small>
2             * - this form is intended to demonstrate different input types.
3     Do not submit confidential information to untrusted sources.
4     See <router-link v-bind:to="{name: 'About'}">site disclaimer</router-link>
5     for more info.
6 </small>
```





# ROUTER-LINK STYLING

Review



.router-link-active

By default Vue adds the router-link-active class to routes.





Review



# HISTORY MODE

```
1 import Vue from 'vue'  
2 import VueRouter from 'vue-router'  
3 import Home from '../views/Home.vue'  
4  
5 Vue.use(VueRouter)  
6  
7 const routes = [  
8   {  
9     path: '/',
10     name: 'Home',
11     component: Home
12   }
13 ];
14
15 const router = new VueRouter({
16   mode: 'history',
17   base: process.env.BASE_URL,
18   routes
19 })
20
21 export default router
```



Review



# DYNAMIC ROUTES

```
1 const routes = [
2   {
3     path: '/Questions',
4     name: 'Questions',
5     component: Questions
6   },
7   {
8     path: '/Questions/:id',
9     name: 'QuestionDetails',
10    component: QuestionDetails
11  },
12  {
13    path: '/Questions/:id/Edit',
14    name: 'EditQuestion',
15    component: QuestionEdit
16  },
17  // others omitted...
18 ];
```

# ROUTER-LINK WITH PARAMS



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



# THIS.\$ROUTE & THIS.\$ROUTER

Review



```
1 created() {
2   const id = this.$route.params.id; // Grabs the route parameter named id, if it was present
3   this.question = this.$store.state.questions.find(q => q.id === id);
4
5   if (!this.question) {
6     this.$router.push({name: 'NotFound'});
7   }
8 }
```



# ROUTING



Review



## SomeComponent.vue

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



## router/index.js

```
1 const routes = [
2   {
3     path: '/',
4     name: 'products',
5     component: Products,
6   },
7   {
8     path: '/products/:id',
9     name: 'product-details',
10    component: ProductDetails,
11  },
12  {
13    path: '*',
14    name: 'not-found',
15    component: NotFound,
16  }
17 ]
```

## ProductDetails.vue

```
1 <script>
2 export default {
3   name: "product-detail",
4   data() {
5     return {
6       productId: 0,
7     }
8   },
9   created() {
10     this.productId = this.$route.params.id;
11   }
12 };
13 </script>
```





Lecture



# AGENDA

- REST Refresher
- Web Service Requests - Axios
- JavaScript Services
- A Teaser on CORS





Lecture



# REST

**REST**, or **R**epresentational **S**tate **T**ransfer, is a style of web communications involving messages that go to agreed upon *web addresses*, have a specified *verb*, and can communicate via *response codes* and *request and response bodies*.

REST is **not** an agreed-upon query language or set of specifications, more of a set of communication tools with some general guidelines.





# REST

<https://www.marvel.com/characters/thing>



```
[  
  { firstName: "Ben", lastName: "Grimm", superHeroName: "Thing"},  
  { firstName: "Jonathan", lastName: "Storm", superHeroName: "Human Torch"}]  
]
```





Lecture



# HTTP METHODS (Request)

## HTTP methods - (Request)

HTTP methods are a pre-defined set of words that each indicate different actions that a client wants to perform for a given resource. These methods are typically called "**HTTP verbs**" since they often perform some action, though some are nouns. Methods only apply to requests, and they are required.

Method	Definition
GET	The GET method requests data from the specified resource and should only be used by the client to retrieve data, not send. This is the method that's used when you use a web browser to view a website.
POST	The POST method submits data to the specified resource.
PUT	The PUT method replaces all current representations of the target resource with the request payload. You use this to "update" data, but it can "create" data if it doesn't exist.
DELETE	The DELETE method deletes the specified resource.





# HTTP METHODS (Request)



Lecture



## "HTTP verbs"

Task	Method	Path
Create a new customer	POST	/customers
Delete an existing customer	DELETE	/customers/{id}
Get a specific customer	GET	/customers/{id}
Search for customers	GET	/customers
Update an existing customer	PUT	/customers/{id}



# HTTP METHODS & RESOURCES (Request)



## “HTTP verbs”

Lecture



GET	/pet/{petId}	Find pet by ID
PUT	/pet	Update an existing pet
DELETE	/pet/{petId}	Deletes a pet
POST	/pet/{petId}/uploadImage	uploads an image





Lecture



# STATUS CODES (Response)

## HTTP Status Codes



**1XX  
INFORMATIONAL**

"OK. All Good."

**2XX  
SUCCESS**

**3XX  
REDIRECTION**

Issue with client code.

**4XX  
CLIENT ERROR**

Issue with server code.

**5XX  
SERVER ERROR**



Lecture



# STATUS CODES (Response)

HTTP STATUS CODES	
2xx Success	
200	Success / OK
3xx Redirection	
301	Permanent Redirect
302	Temporary Redirect
304	Not Modified
4xx Client Error	
401	Unauthorized Error
403	Forbidden
404	Not Found
405	Method Not Allowed
5xx Server Error	
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout

yINIGHT



# DON'T FORGET POSTMAN



**Postman** is a tool we use to simulate a web request and look at the response that comes back.

It is just one way of sending a request and is used for debugging / testing purposes.

Lecture



The screenshot shows the Postman application window. The left sidebar has options: Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area is titled 'Scratch Pad' with tabs for 'New' and 'Import'. A central panel shows a 'GET' request to 'http://localhost:3000/boards/89847'. The 'Params' tab is selected, showing a single entry: 'Key' under 'Query Params'. Below the request, the status bar shows '200 OK 1517 ms 52.87 KB | Save Response'. The response body pane displays JSON data:

```
[{"id": "1", "name": "Board 1", "list": [{"id": "1", "name": "Card 1"}, {"id": "2", "name": "Card 2"}]}, {"id": "2", "name": "Board 2", "list": [{"id": "3", "name": "Card 3"}, {"id": "4", "name": "Card 4"}]}]
```





Lecture



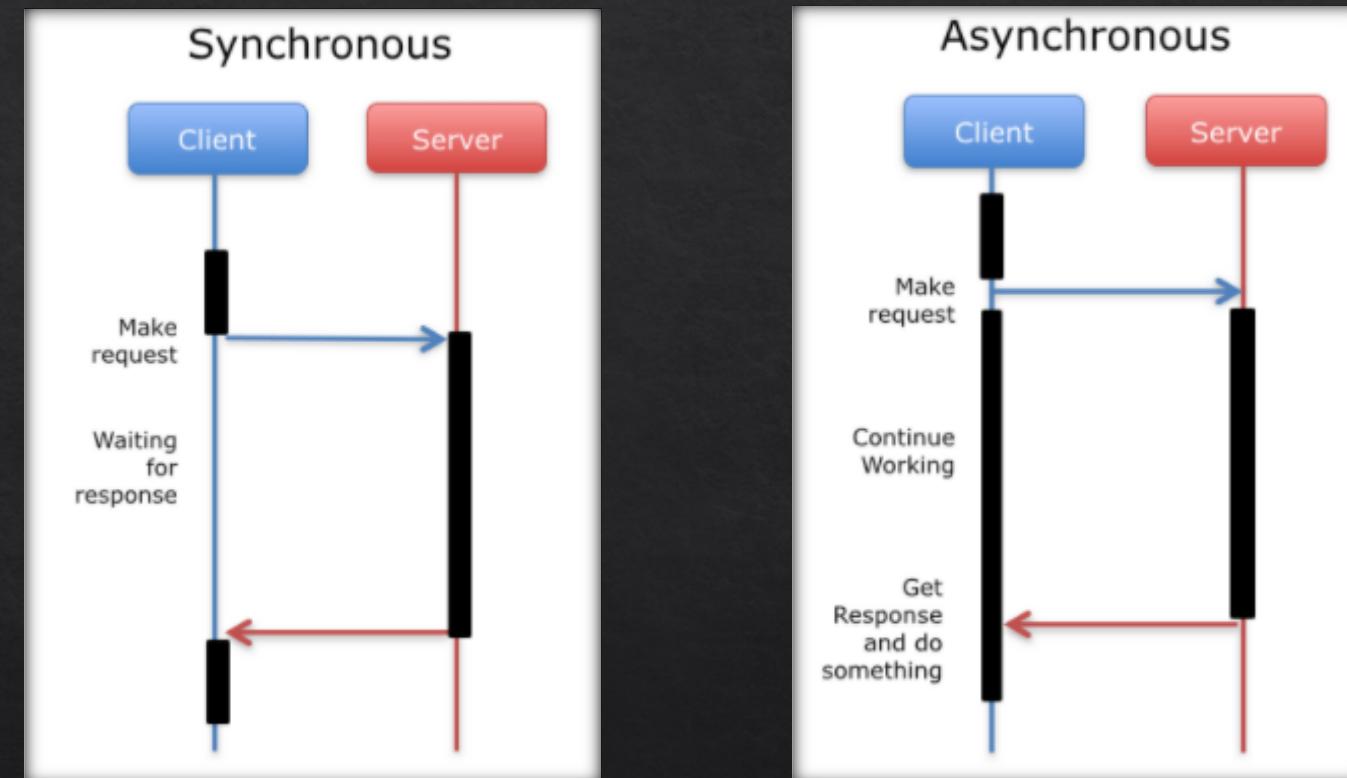
# SYNCHRONOUS & ASYNCHRONOUS



# SYNC VS ASYNC



Lecture





Lecture



# AXIOS

```
npm install axios --save
```



# AXIOS GET



Lecture



```
1 /**
2  * Gets all items on the server
3  * @returns {Promise} a promise that will complete with a list of items
4 */
5 getAllItems() {
6  // Create our Axios instance used to communicate with the server
7  const http = axios.create({
8    baseURL: 'https://some.website.com'
9  });
10
11 return http.get('/items'); // This is added to the end of baseURL specified above
12 }
```

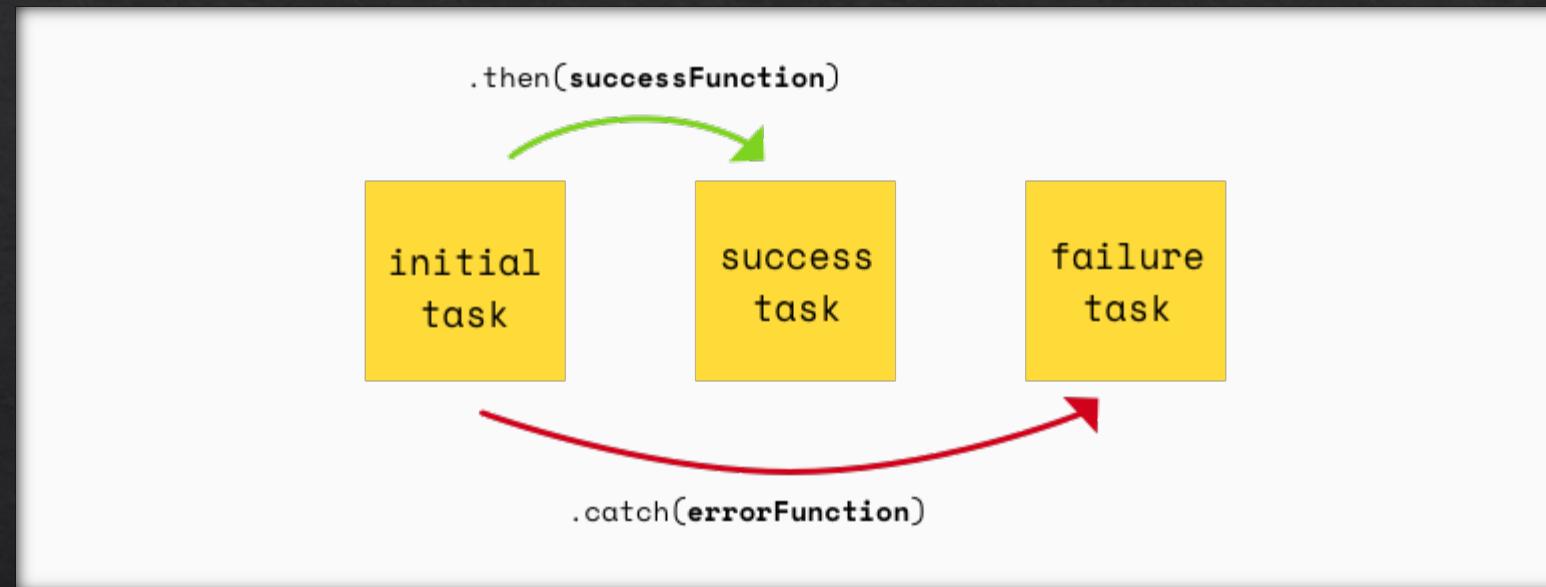




Lecture



# USING A PROMISE





Lecture



```
1 getAllItems().then(response => {
2   // response.data is loaded from the contents of the response body
3   // It's typically going to be a JavaScript object or an array of objects
4   const items = response.data;
5   this.$store.commit('ITEMS_LOADED', items);
6 });
```



# SERVICES



Lecture



```
import axios from 'axios';

//Create our Axios instance used to communicate with the server
const http = axios.create({
  baseURL: "http://localhost:3000"
});

export default { //This object is what other files will import via the import keyword

  getBoards() {
    return http.get('/boards'); //This is added to the end of baseURL specified above
  },

  getCards(boardID) {
    return http.get(`/boards/${boardID}`)
  },

  getCard(boardID, cardID) {
    return http.get(`/boards/${boardID}`).then((response) => {
      const cards = response.data.cards;
      return cards.find(card => card.id == cardID);
    })
  }

}
```



Lecture



# SERVICES

```
1 import axios from 'axios';
2
3 // Create our Axios instance used to communicate with the server
4 const http = axios.create({
5   baseURL: 'https://some.url.net'
6 });
7
8 export default { // This object is what other files will import via the import keyword
9
10  getAllItems() {
11    return http.get('/items'); // This is added to the end of baseURL specified above
12  },
13
14  getItem(id) {
15    return http.get(`/items/${id}`);
16  },
17
18  update(myItem) {
19    return http.put(`/items/${myItem.id}` , myItem);
20  },
21
22  create(myItem) {
23    return http.post('/items' , myItem);
24  },
25
26  delete(myItem) {
27    return http.delete(`/items/${myItem.id}`);
28  }
29
30};
```

```
1 import QuestionService from "./services/QuestionService.js";
2
3 QuestionService.getAllItems()
4   .then(response => console.log(response));
```



Lecture



## Lifecycle Hook



# LIFECYCLE HOOKS

```
<template>
  <ul>
    <li v-for="user in users" v-bind:key="user.id">
      {{ user.firstName }} {{ user.lastName }}
    </li>
  </ul>
</template>

<script>
  import apiService from '@/services/apiService.js';

  // You now have access to the service in your code
  export default {
    name: 'user-list',
    created() {
      apiService.list().then( (response) => {
        this.users = response.data;
      });
    }
  }
</script>
```



Lecture



# ADDITIONAL RESOURCES

- [BootcampOS](#)
- [Vue.js Documentation - \(\)](#)
- [Vue CLI – \(\)](#)



# Module 3

## Vue Web Services - GET

New Tools! Let's Code!

Vue.js  
Web Services - GET



Developer's Toolbox



Exercise HW

