# Module 3

## VUE WEB SERVICES - GET

JAVA – MODULE 3, DAY 16

ELEVATE YOURSELF

Review

# RECAP

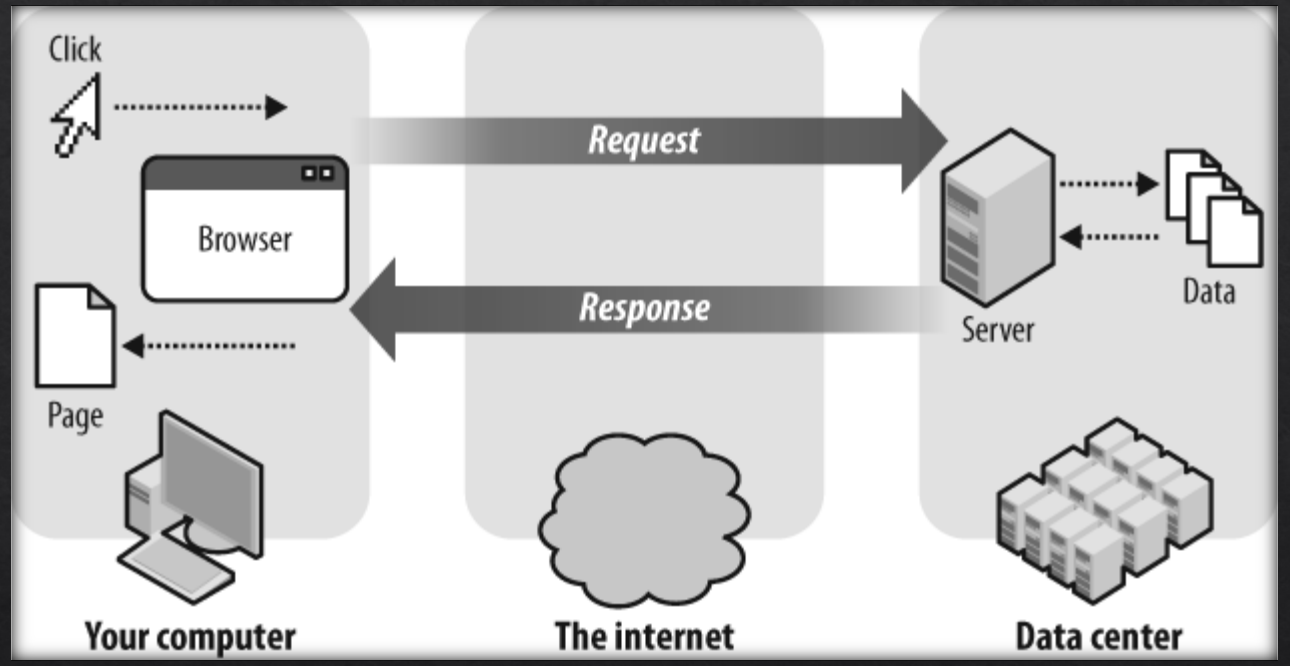
TECH ELEVATOR

Review

# ROUTING

Review

# WEB NAVIGATION



TECH ELEVATOR

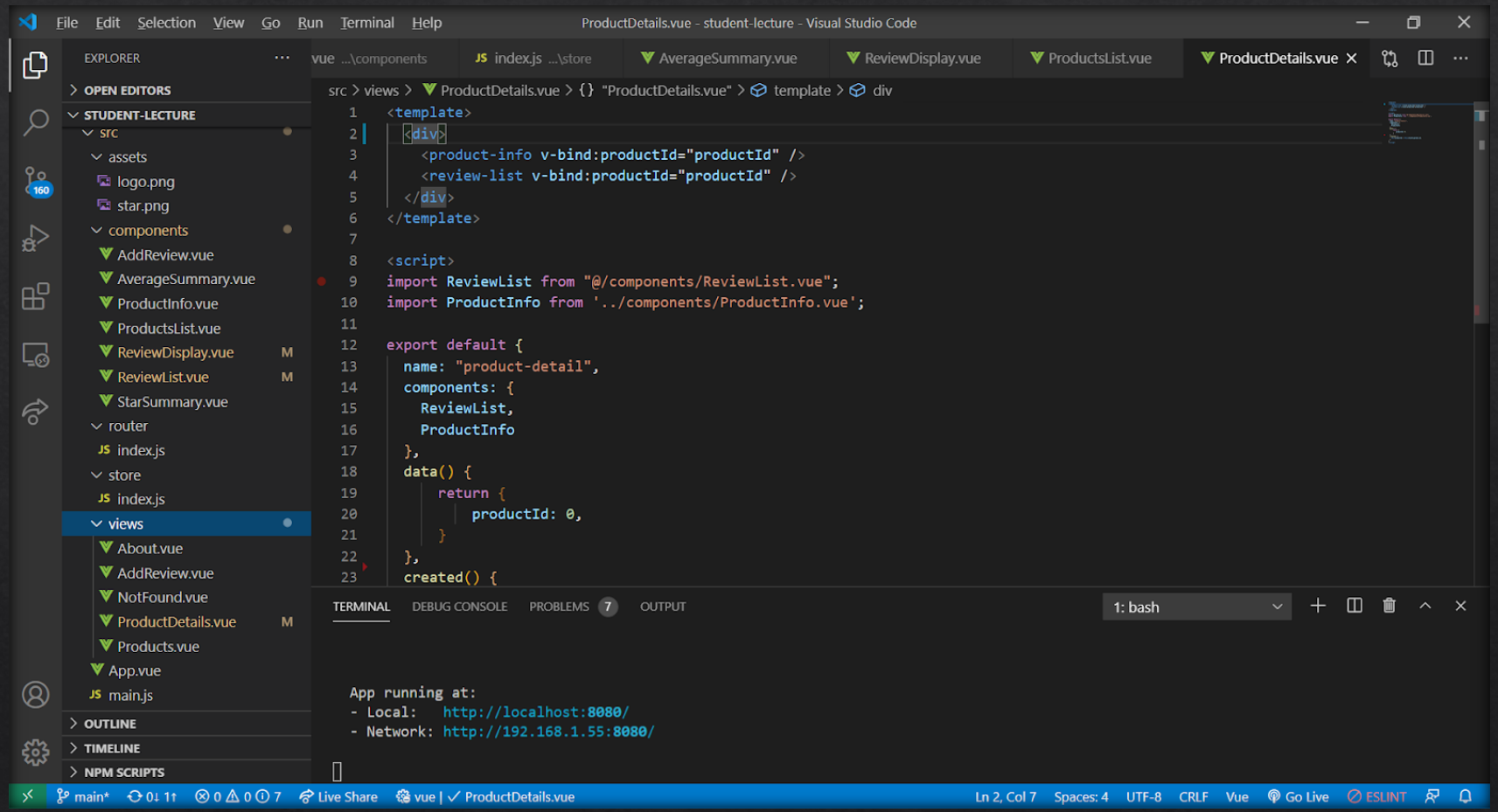
# VUE.JS NAVIGATION

Review

TECH ELEVATOR

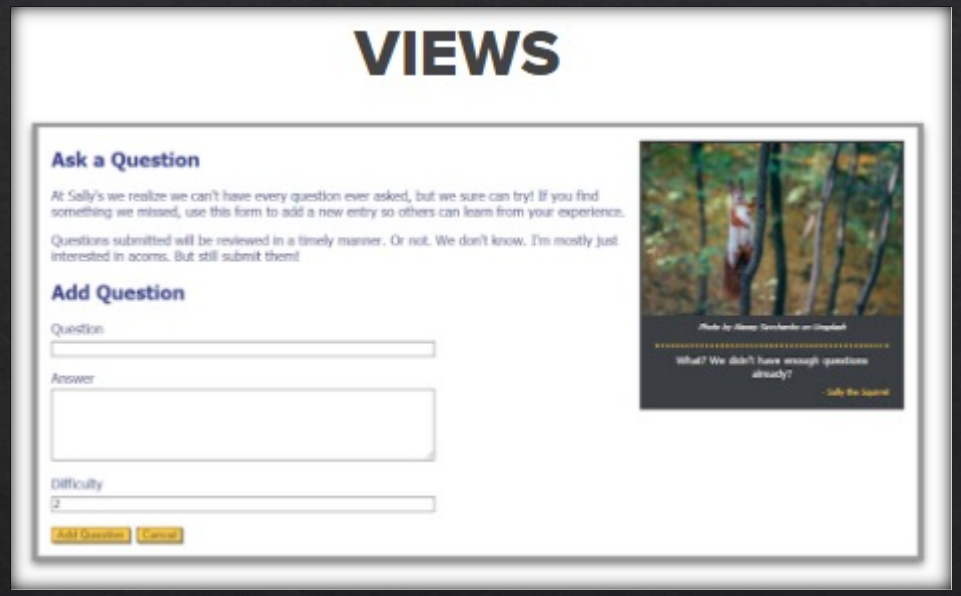
# VUE.JS NAVIGATION

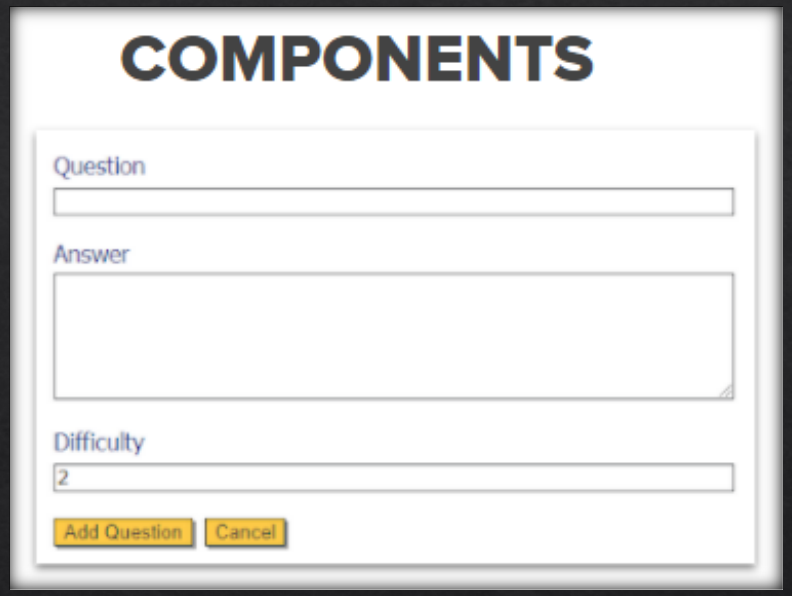
Review



Review

# VIEWS VS COMPONENTS





# INSTALLING ROUTING

- vue create my-project

or

- vue add router



# DEFINING ROUTES

```
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: '/',              // Required
8     name: 'Home',           // Recommended, but not required
9     component: Home         // Required
10  },
11  {
12    path: '/About',
13    name: 'About',
14    component: About
15  },
16  {
17    path: '*',
18    name: 'NotFound',
19    component: NotFound
20  }
21 ];
```

TECH ELEVATOR

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

```
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");
```

Review

# WILDCARD ROUTES

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



# ROUTER-VIEW

```
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>
```

Review

# ROUTER-LINK

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

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



# ROUTER-LINK STYLING

.router-link-active

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

TECH ELEVATOR

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
```

TECH ELEVATOR

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>
```

Review

# THIS.$ROUTE & THIS.$ROUTER

```
created() {
    const id = this.$route.params.id; // Grabs the route parameter named id, if it was present
    this.question = this.$store.state.questions.find(q => q.id === id);

    if (!this.question) {
        this.$router.push({name: 'NotFound'});
    }
}
```

TECH ELEVATOR

# ROUTING

Review

SomeComponent.vue

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

router/index.js

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

ProductDetails.vue

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


TECH
ELEVATOR
Lecture
AGENDA
• REST Refresher
• Web Service Requests - Axios
• JavaScript Services
• A Teaser on CORS
ELEVATE YOURSELF

# REST

**REST**, or **Re**presentational **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"}
]
```

# 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. |

Lecture

# HTTP METHODS
## (Request)

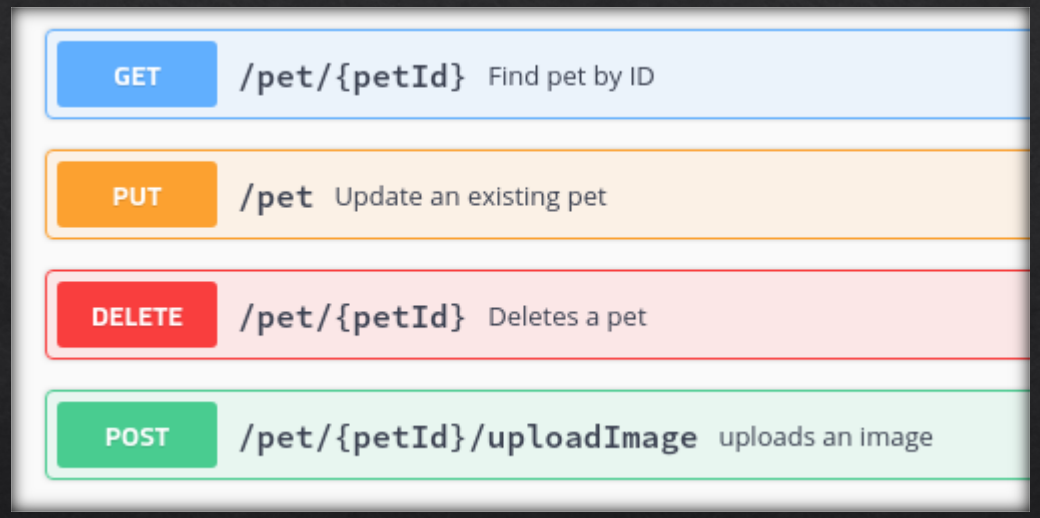
**"HTTP verbs"**

| 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} |

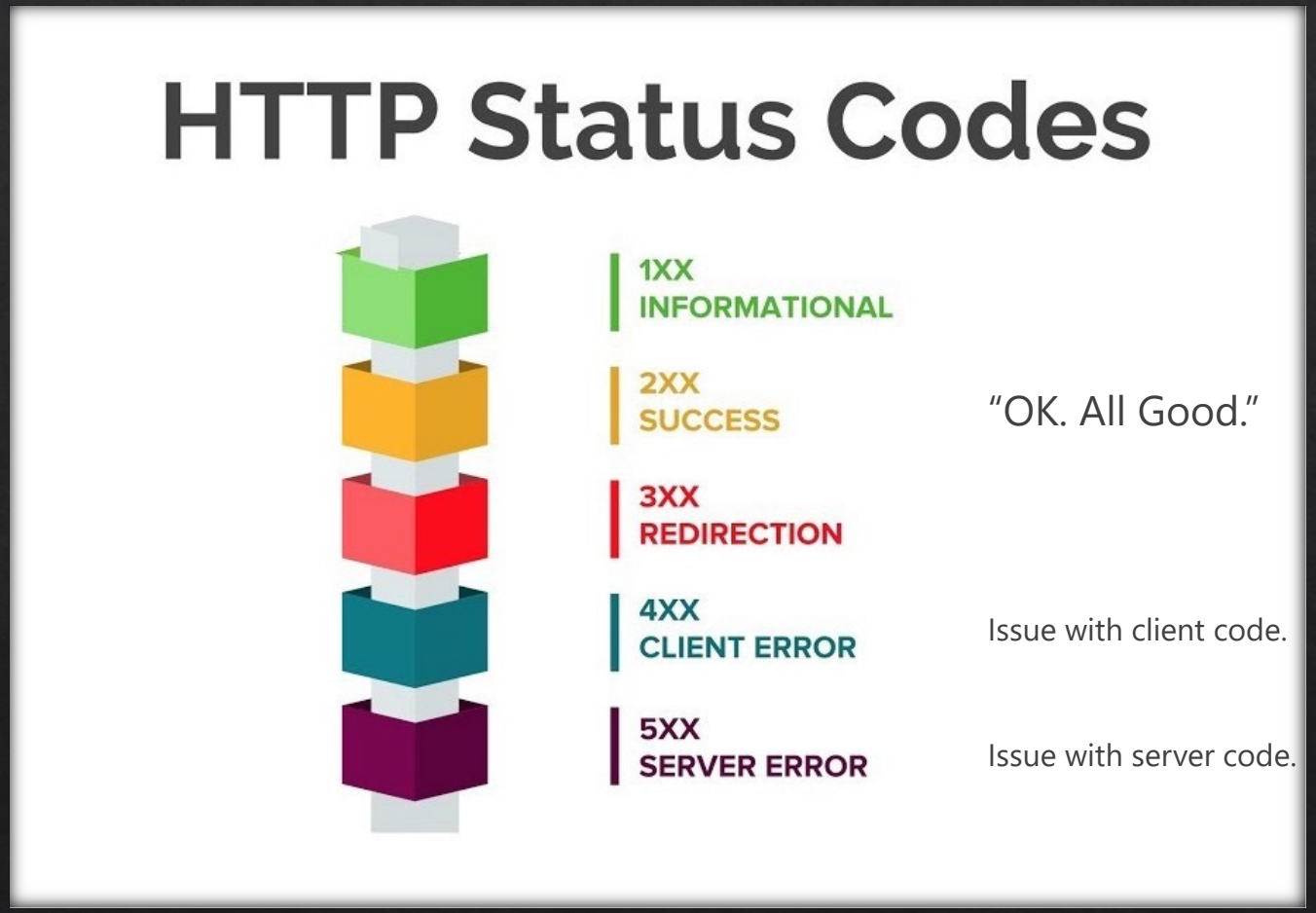
# HTTP METHODS & RESOURCES (Request)

Lecture

"HTTP verbs"



# STATUS CODES
## (Response)



# STATUS CODES
## (Response)

Lecture

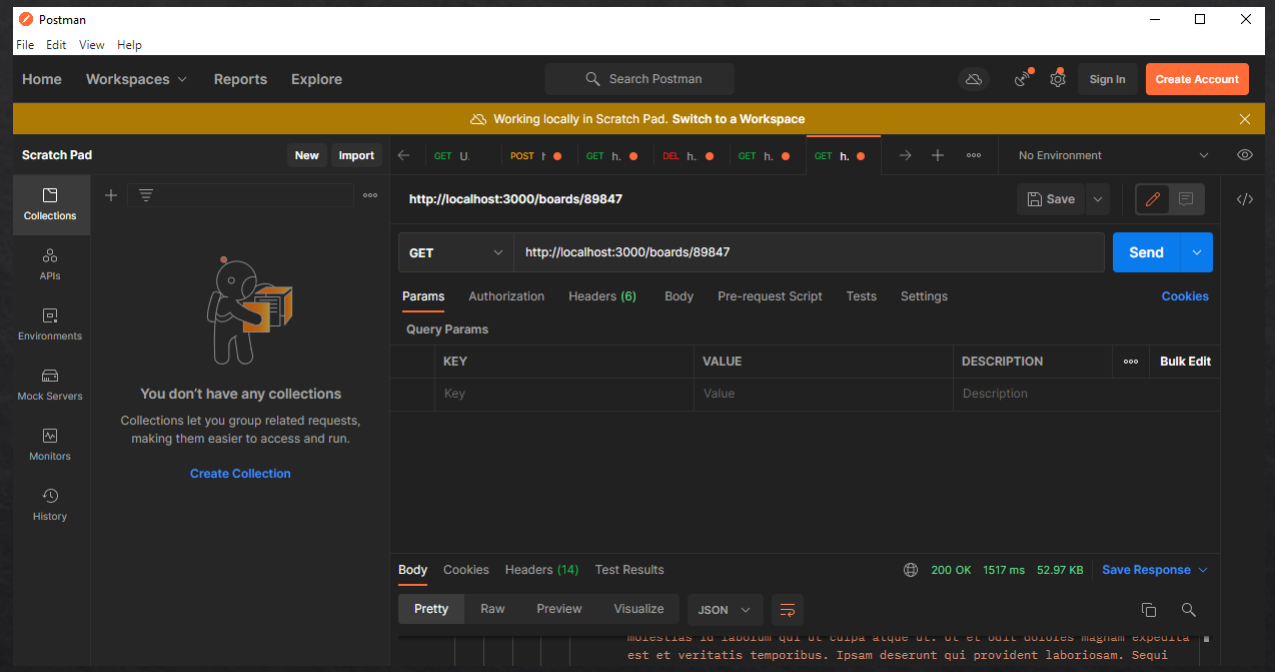
| HTTP STATUS CODES | |
|---|---|
| **2xx Success** | |
| 200 | Success / OK |
| **3xx Redirection** | |
| 301 | Permanent Redirect |
| 302 | Temporary Redirect |
| 304 | Not Modified |
| **4xx Client Error** | |
| 401 | Unauthorized Error |
| 403 | Forbidden |
| 404 | Not Found |
| 405 | Method Not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

TECH ELEVATOR

# 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.

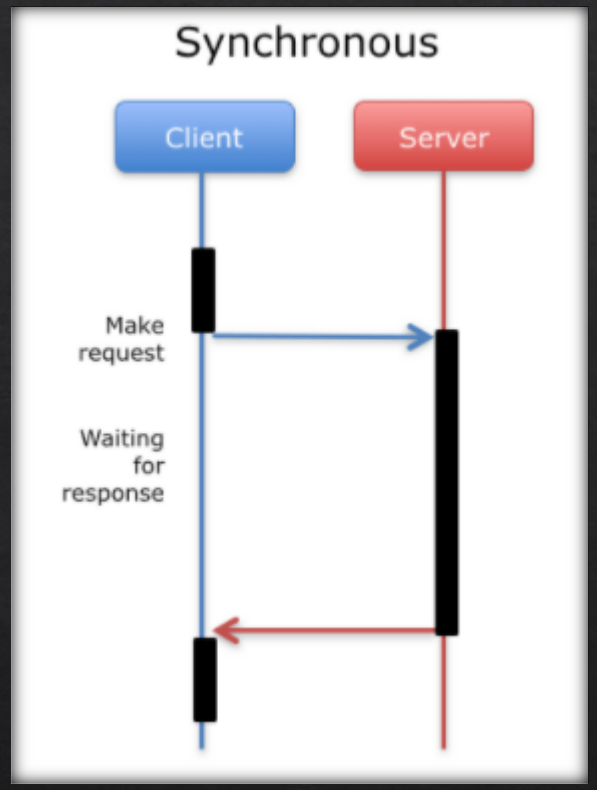


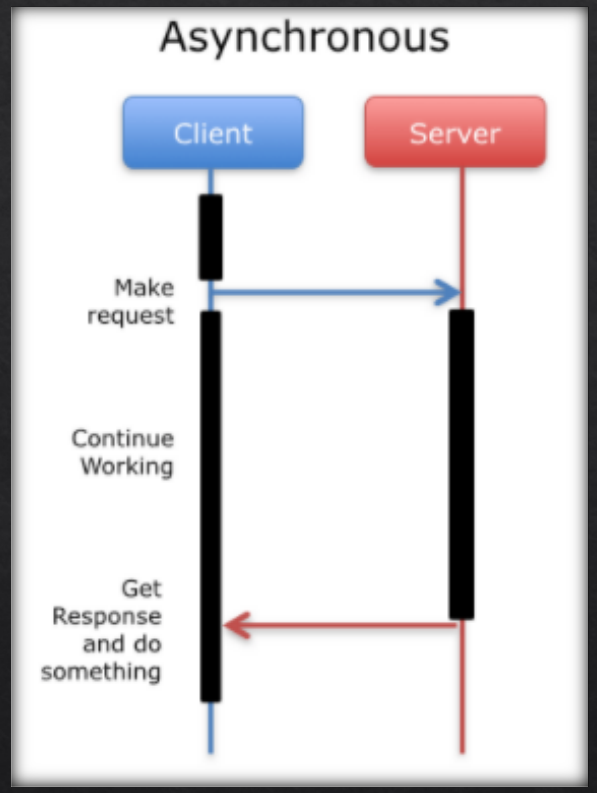

Lecture

Lecture

# SYNCHRONOUS & ASYNCHRONOUS

# SYNC VS ASYNC




Lecture

Lecture

# AXIOS

**npm install axios --save**

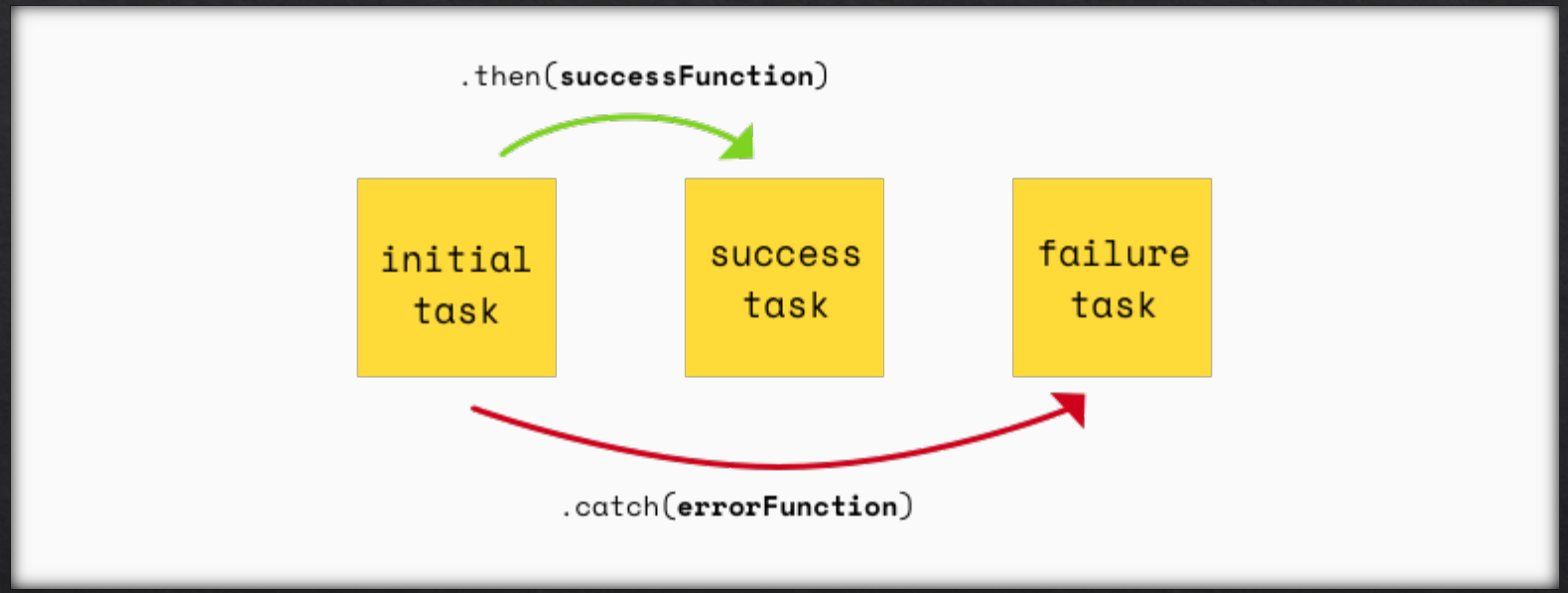
Lecture

# AXIOS GET

```
/**
 * Gets all items on the server
 * @returns {Promise} a promise that will complete with a list of items
 */
getAllItems() {
  // Create our Axios instance used to communicate with the server
  const http = axios.create({
    baseURL: 'https://some.website.com'
  });

  return http.get('/items'); // This is added to the end of baseURL specified above
}
```

# USING A PROMISE



Lecture

# AXIOS GET

```
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   });
```

Lecture

# SERVICES

```
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);
    })
  }

}
```

# 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));
```

TECH ELEVATOR

Lecture

# LIFECYCLE HOOKS

Lifecycle Hook

```
<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 - (https://vuejs.org/v2/guide/)
- Vue CLI – (https://cli.vuejs.org/)

TECH ELEVATOR

# Module 3

## Vue Web Services - GET

## New Tools! Let's Code!

**Vue.js**
**Web Services - GET**


Developer's Toolbox

Exercise HW