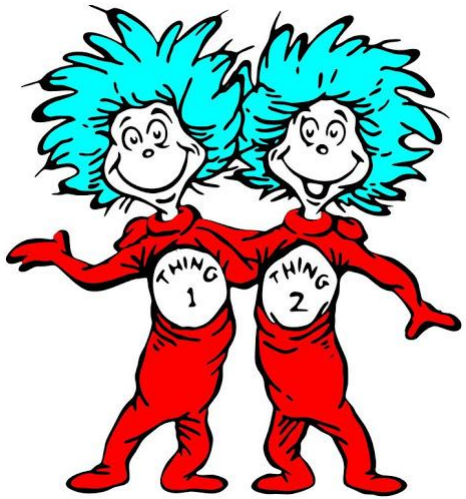


I just saw my wife trip and fall while carrying a laundry basket full of ironed clothes. I watched it all unfold!

My son told me he didn't understand cloning. I told him, 'That makes two of us.

Consuming API’s in VUE

Objectives

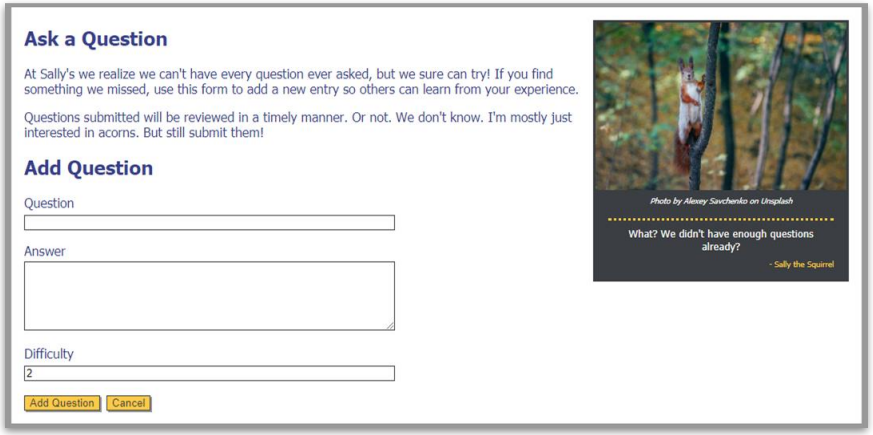
- Review typical HTTP request
- GET request
- 2xx Status Code indicates "success"
- Make an HTTP GET request using Postman and inspect the result
- Review JSON
- Use Axios
- Build a service object
- Use the Vue lifecycle hook `created()`
- Synchronous vs. asynchronous code
- What is a promise and how does it work
- Async coding techniques

But first…

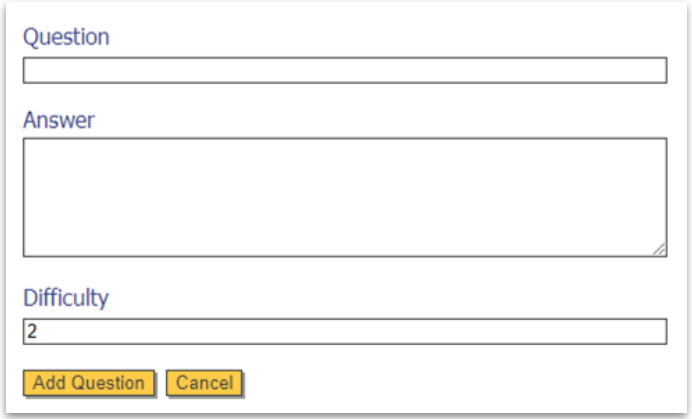
A small review!!

View vs. Components

VIEWS

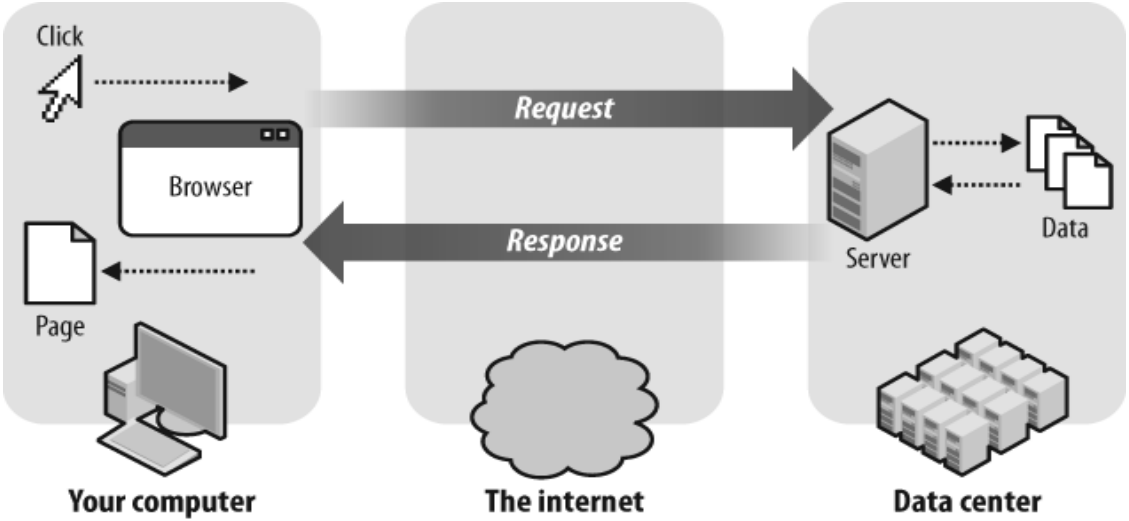


COMPONENTS



What is Routing?

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



Defining Routes

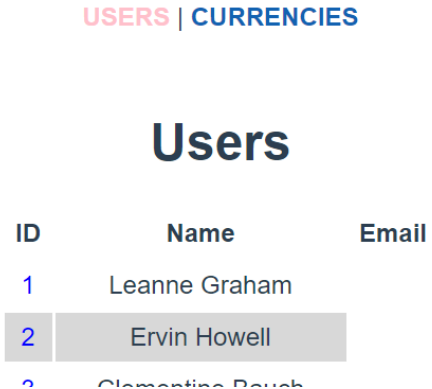
```
1  import Home from '../views/Home.vue'
2  import About from '../views/About.vue'
3  import NotFound from '../views/NotFound.vue'
4
5  const routes = [
6    {
7      path: '/',              // Required
8      name: 'Home',           // Recommended, but not required
9      component: Home        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

Router-link

```
<small>
    * - this form is a joke 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

```
    <div id="nav">
      <router-link :to="{name: 'users'}">Users</router-link> |
      <router-link :to="{ name: 'currencies' }">Currencies</router-link>
    </div>
    <router-view/>
  </div>
</template>

<style>

#nav a {
  font-weight: bold;
  color: #1262b1;
}
#nav a.router-link-active {
  color: pink;
}
```



Router-view

```
<template>
  <div id="app">
    <AppHeader />
    <router-view /> <!-- The currently active view will be presented here -->
    <app-footer />
  </div>
</template>
```

Dynamic Routes

```
const routes = [
  {
    path: '/Questions',
    name: 'Questions',
    component: Questions
  },
  {
    path: '/Questions/:id',
    name: 'QuestionDetails',
    component: QuestionDetails
  },
  {
    path: '/Questions/:id/Edit',
    name: 'EditQuestion',
    component: QuestionEdit
  },
  // others omitted...
];
```

Router-link Params

```
<section>
    <!-- Params below can be accessed from the destination page via this.$route.params.parameterName -->
    <router-link v-bind:to="{name: 'EditQuestion', params: {id: question.id}}">
        Edit this Question
    </router-link>
</section>
```

$Router.Push

```
1 saveQuestion() {
2     this.$store.commit('QUESTION_UPDATED', this.question);
3     this.$router.push({name: 'QuestionDetails', params: {questionId: this.question.id}});
4 }
```

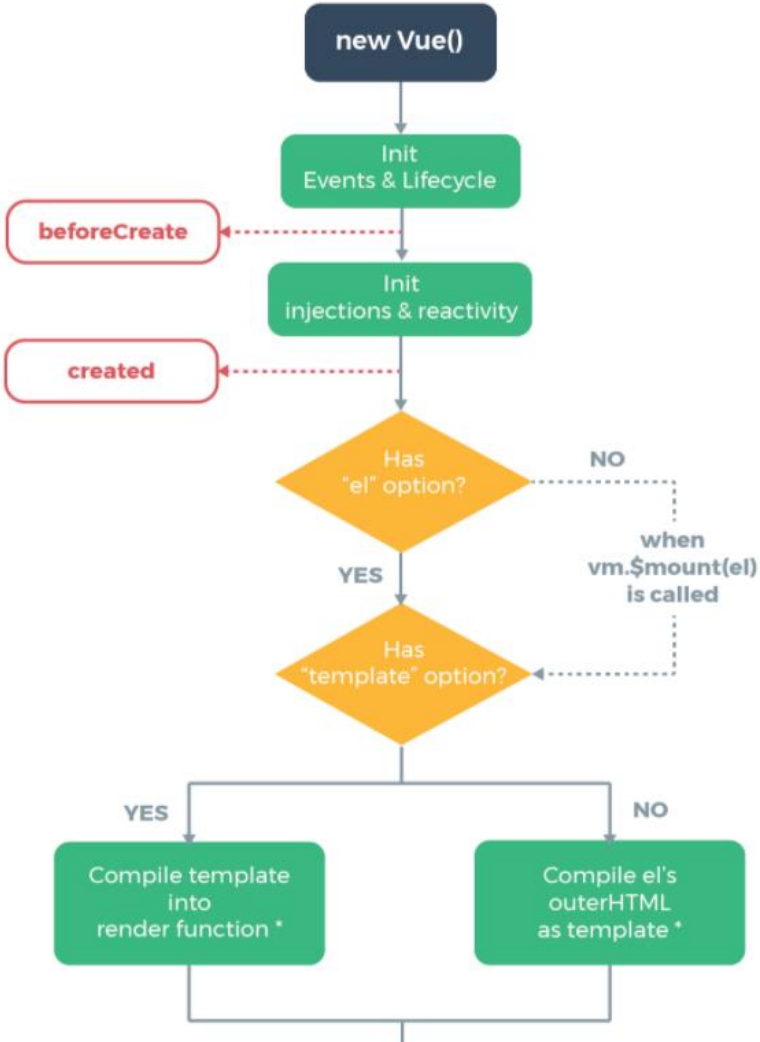
$Route.Params

```
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'});
    }
}
```

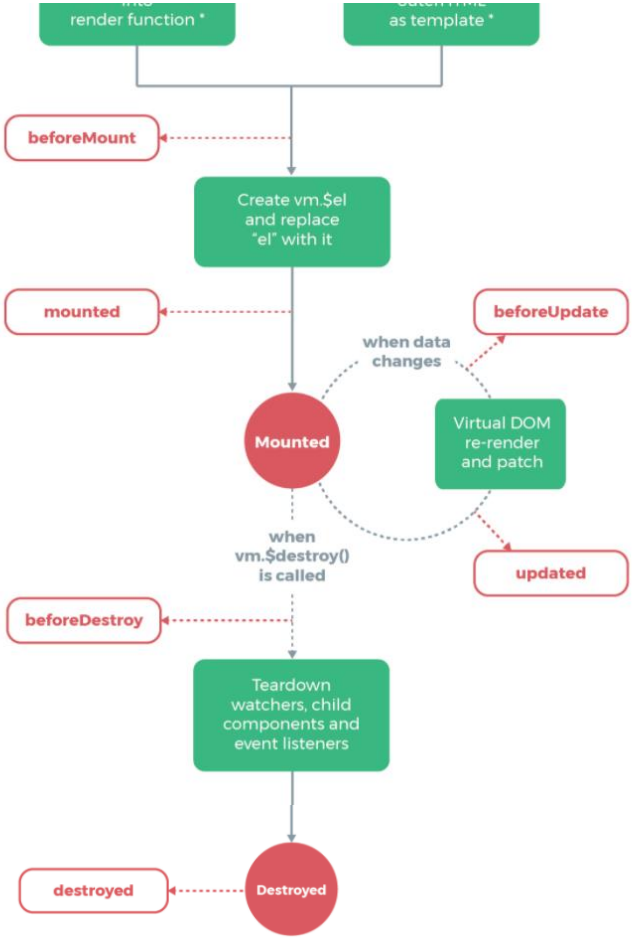
Lifecycle Events

beforeCreate( ) created( )	→	Instance is being *created*
beforeMount( ) mounted( )	→	Instance is being *mounted*
beforeUpdate( ) updated( )	→	Instance is being *updated*
beforeDestroy( ) destroyed( )	→	Instance is being *destroyed*

Lifecycle Events



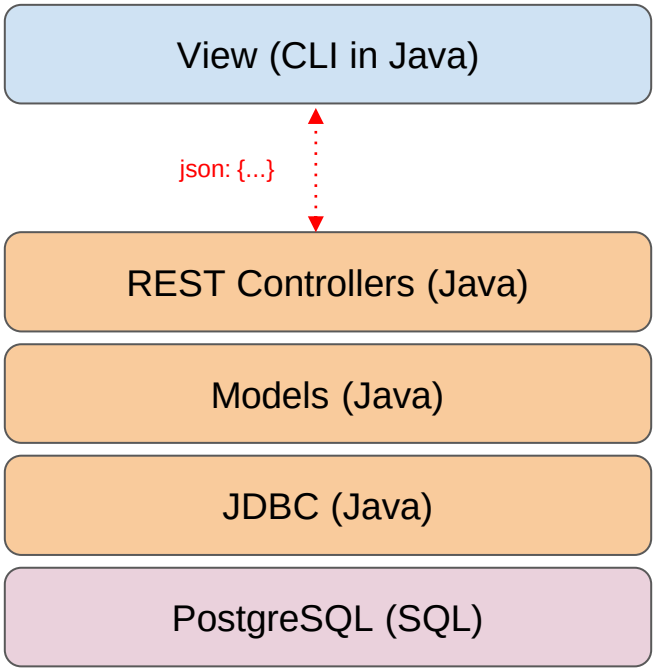
Lifecycle Events



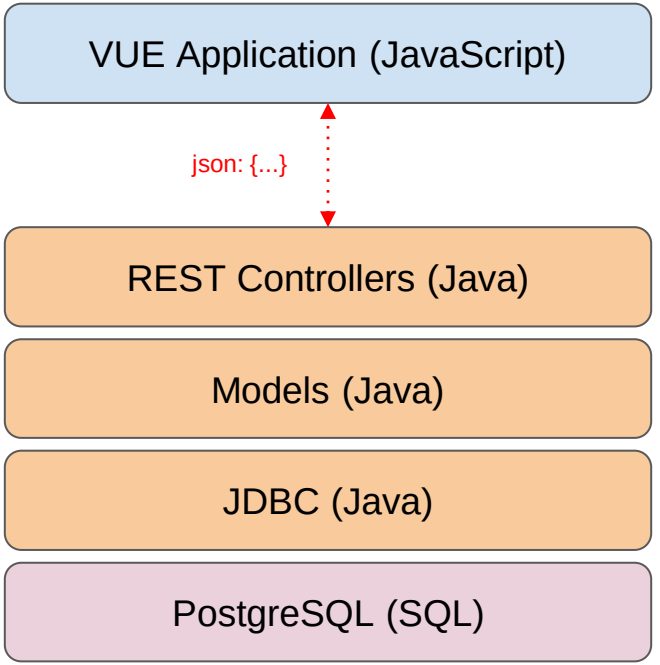
created()

```
1 // This fires after the component is created but before it renders.
2 // You can access any data, props, and $route info you need to here.
3 // This is also a good place to kick off requests for data your component will eventually need
4 created() {
5     const id = this.$route.params.id; // Grabs the route parameter named id, if it was present
6     this.question = this.$store.state.questions.find(q => q.id === id);
7
8     if (!this.question) {
9         this.$router.push({name: 'NotFound'});
10    }
11 }
```

Module 2 vs Module 3 (Comparing Stacks)

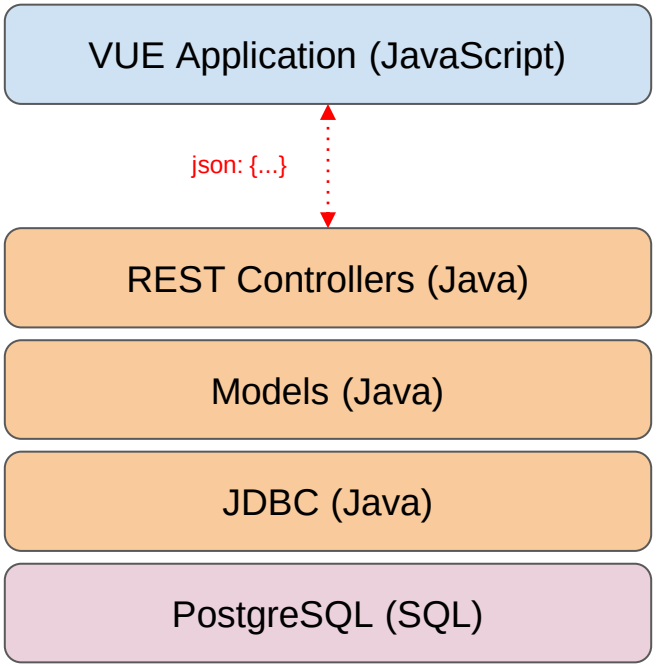


Module 2



Module 3

Consuming API's with VUE



We will use all the techniques we've learned so far in VUE to construct an application capable of consuming a REST API.

While we can use fetch syntax from Vanilla JS, we will be learning Axios for its ease.

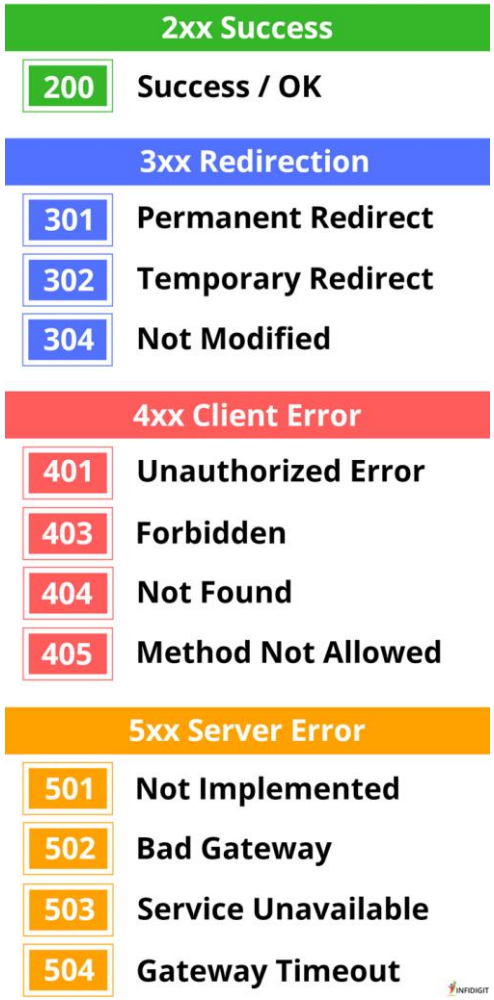
Module 3

Requests to a REST Endpoint

Recall that a REST controller can be configured to handle various types of requests. Let's review them:

- **GET**: Ideally suited to retrieve all the records from a REST endpoint.
- **GET (with path variable)**: We can configure path variables (i.e. doggo/1) to retrieve a single record of data.
- **POST**: Ideally suited for inserting new data into the data source.
- **PUT**: Ideally suited for updating an existing record within a data source.
- **DELETE**: Ideally suited for removing an existing record from the data source.

HTTP STATUS CODES

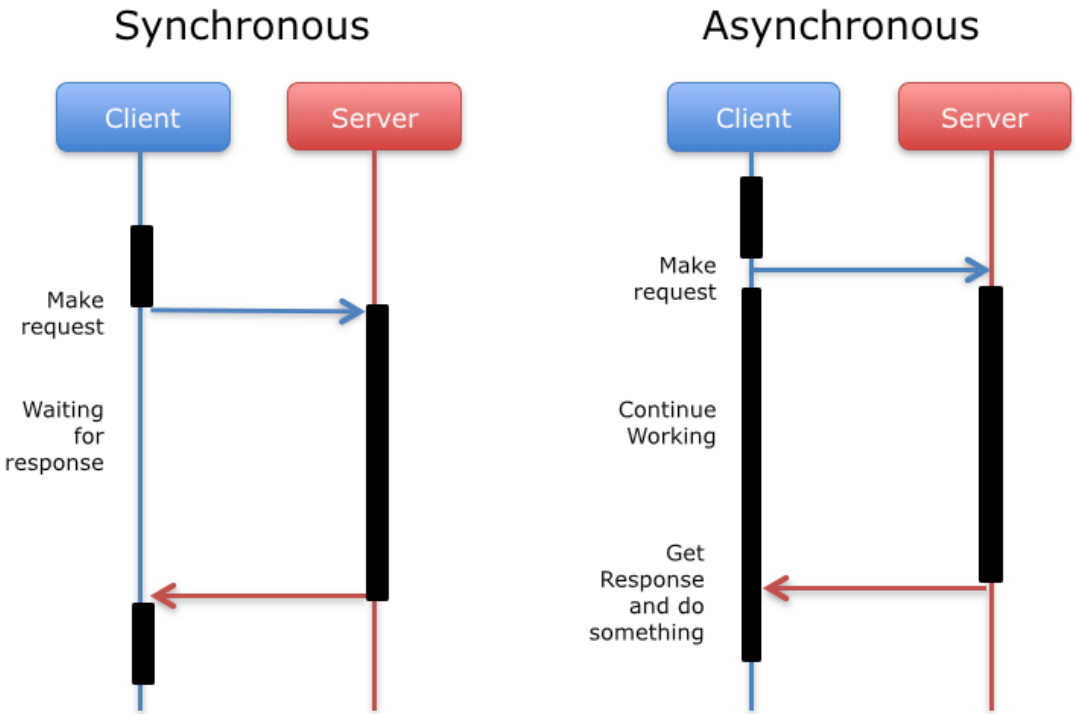


418 I’M A TEAPOT

When you help someone fix their code but you can't fix your own

Synchronous vs Asynchronous

Sync vs Async



AXIOS

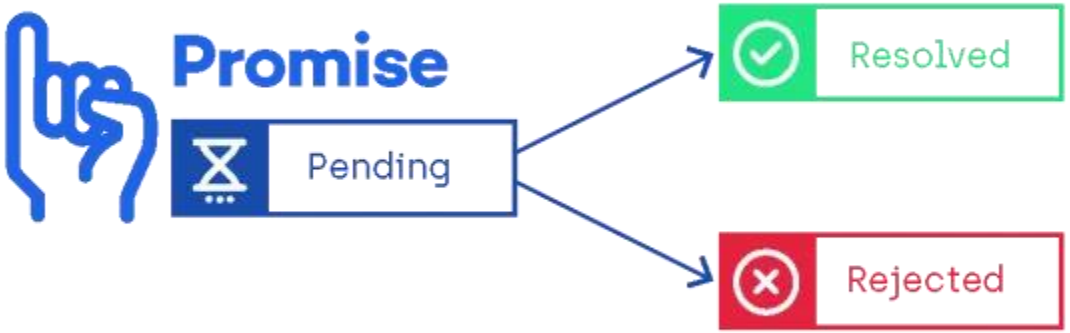
```
npm install axios --save
```

Axios Get

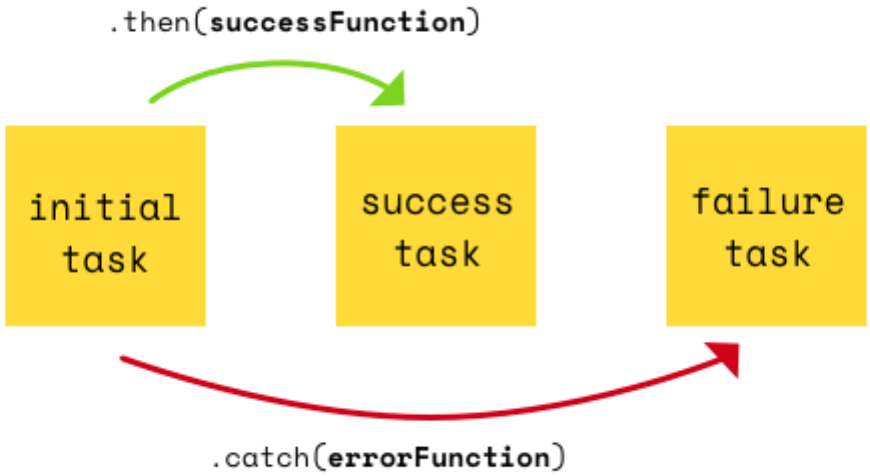
```
/**
 * Gets all items on the server
 * @returns {Promise} a promise that will complete with a list of items
 */
getAllItems() {
  // Create our Axios instance used to communicate with the server
  const http = axios.create({
    baseURL: 'https://some.website.com'
  });

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

What Is A Promise?

What Is A Promise?



Using A Promise



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
}
```

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
}
```

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

Objectives

- Review typical HTTP request
- GET request
- 2xx Status Code indicates "success"
- Make an HTTP GET request using Postman and inspect the result
- Review JSON
- Use Axios
- Build a service object
- Use the Vue lifecycle hook `created()`
- Synchronous vs. asynchronous code
- What is a promise and how does it work
- Async coding techniques

When you try to look at
the code you wrote a month ago