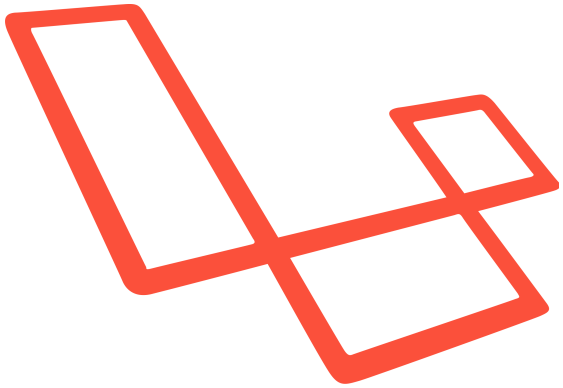
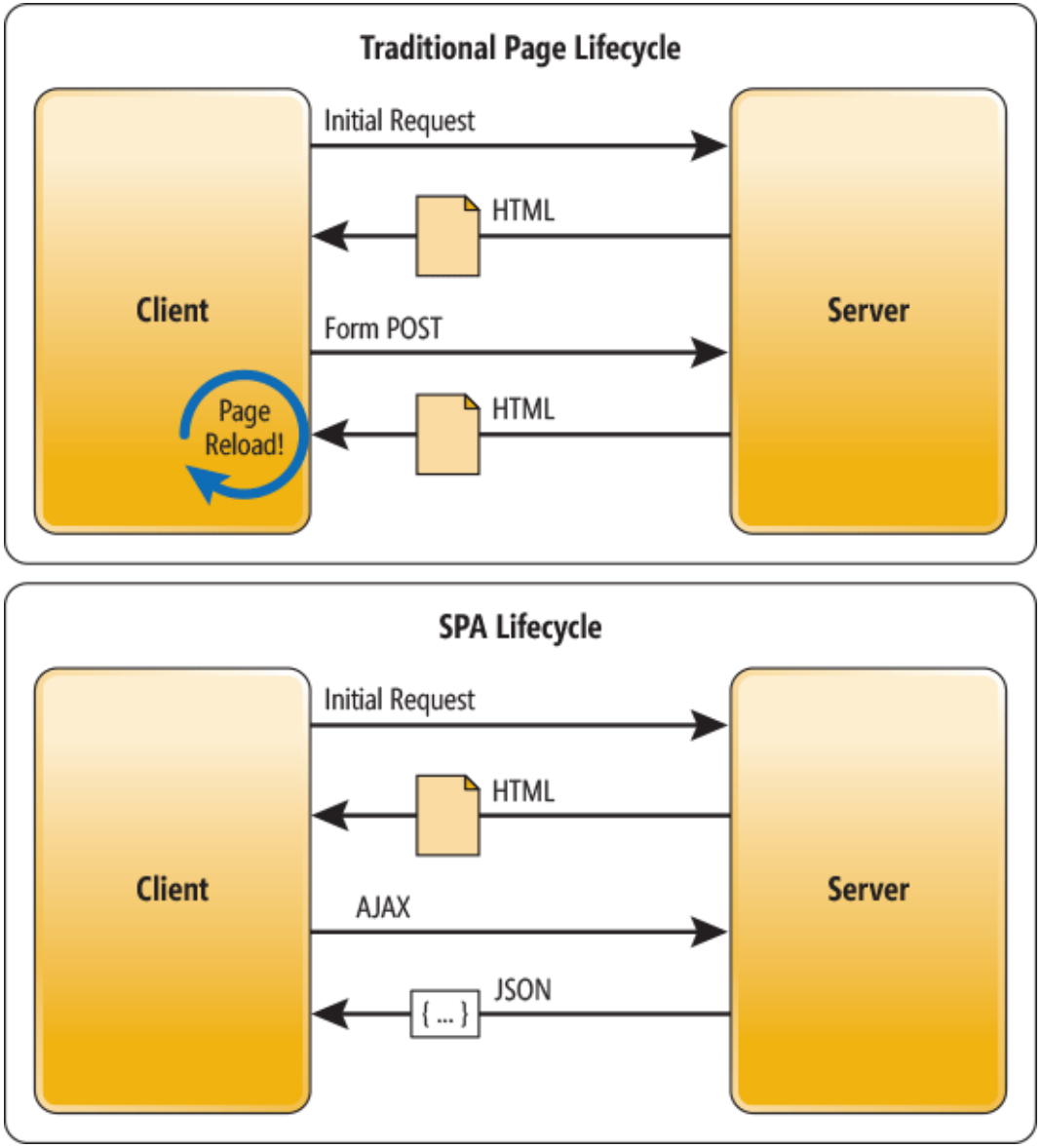


CSF304 – Web Application Development

Session 16: VueJS and Ajax and Laravel

Dr. Liam O'Reilly

Traditional Web Apps vs SPA



https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

Ajax

- AJAX (Asynchronous JavaScript and XML):
  - AJAX is not a programming language.
  - It is a technique for accessing web servers from a web page.

- We load a HTML page with JavaScript.
- The JavaScript then executes asynchronous HTTP requests in the background to communicate with servers.

- We use the data returned by AJAX to update the DOM of the webpage.

Reminder: JSON (JavaScript Object Notation)

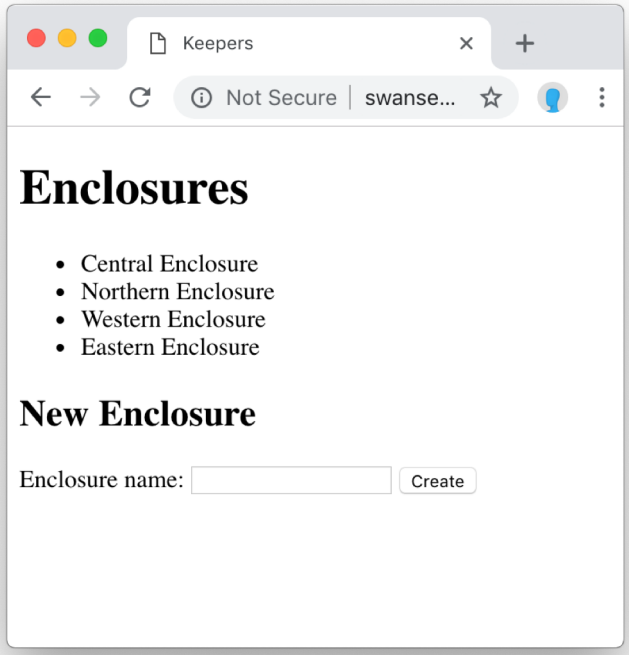
- You looked at XML and JSON in CSF205.
- JSON is a syntax for storing and exchanging data.
- Example of JSON (which has been formatted a little):

```
[
    {"id":1,"name":"Central
    Enclosure","created_at":"2018-11-27
    22:17:44","updated_at":"2018-11-27 22:17:44"},

    {"id":2,"name":"Northern
    Enclosure","created_at":"2018-11-27
    22:17:44","updated_at":"2018-11-27 22:17:44"},

    {"id":4,"name":"Eastern
    Enclosure","created_at":"2018-11-27
    22:17:44","updated_at":"2018-11-27 22:17:44"}
]
```

Today

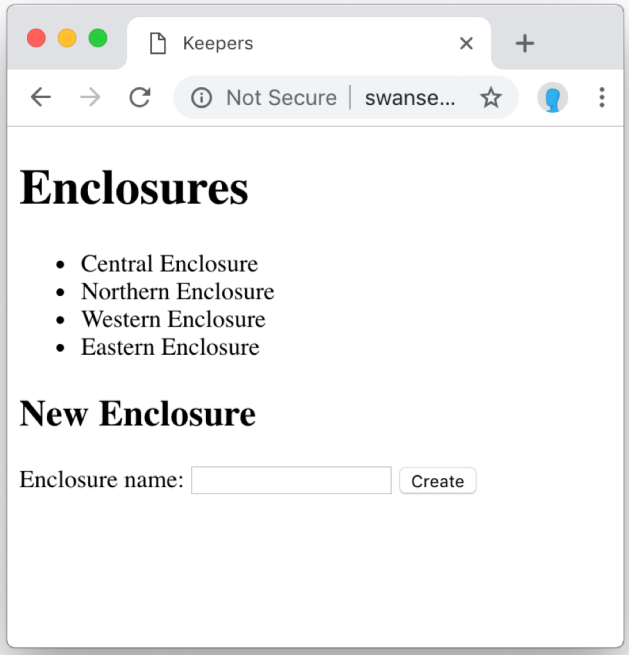
We will now create a simple demonstration of a Vue.js application which will:

- Communicate to a back-end Laravel server using an API.
- Use AJAX with JSON.
- Allow Enclosures to be listed and added to Swansea Zoo.



Overall Idea

- We will build a single Vue.js page which will:
  - Initially have no data.
  - Request all the current enclosures by issuing an AJAX request.
  - Contain a text box and a button which allows the creation of a new enclosure by issuing an AJAX request.



Step 1: Getting Laravel and Vue.js to work together

How to Get Laravel and Vue.js to Work Together

- There are a few ways to get Laravel and Vue.js to work together.
- The proper way(s) involve the use of several tools we have not looked at.
- We will go for a quick and dirty solution.

- We will be mixing PHP/Blade and HTML and JavaScript!

Setup A Route and a View

- We need a route and a view to display our Enclosures Vue.js page.
  - In routes/web.php

```
Route::get('/enclosures', 'EnclosureController@page');
```

This will display our single page for enclosures.

  - Create an Enclosure Controller:

```
php artisan make:controller EnclosureController --resource -m Enclosure
```

  - Create the page method in EnclosureController:

```
public function page()
{
    return view('enclosures.index');
}
```

Setup A Route and a View (Cont.)

- Next, we just need the actual view.

  Resources/view/enclosures/index.blade.php

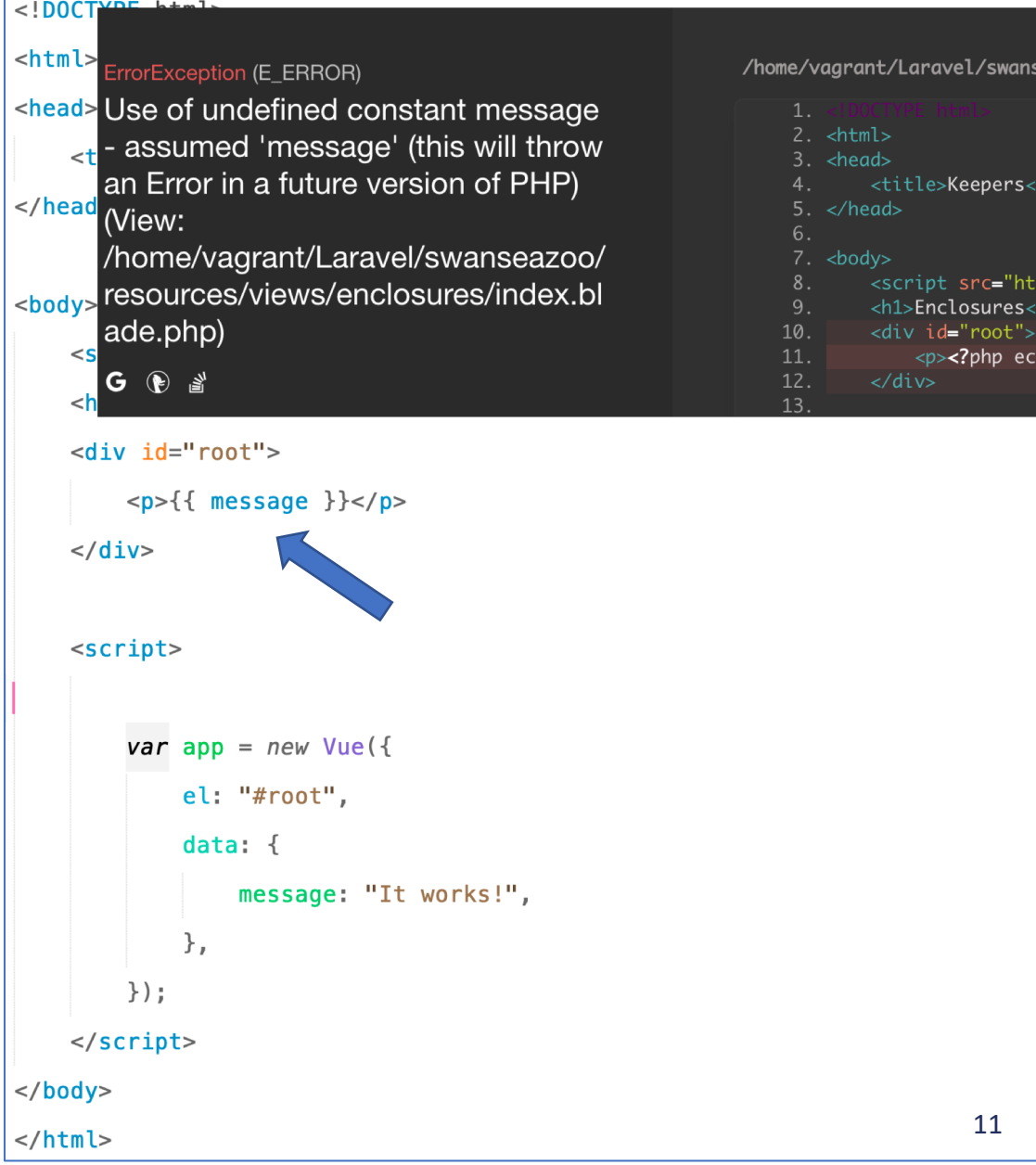
- We are not doing this properly. We will just put all the HTML and Javascript in the view file.
  - We will pull in Vue.js from a CDN.

- So lets get a simple example working.

```
<!DOCTYPE html>
<html>
<head>
    <title>Keepers</title>
</head>

<body>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <h1>Enclosures</h1>
    <div id="root">
        <p>{{ message }}</p>
    </div>

    <script>

        var app = new Vue({
            el: "#root",
            data: {
                message: "It works!",
            },
        });
    </script>
</body>
</html>
```

First Error

- Well that didn't work!
- Seems to be a problem with message.
- Think about it…
- {{ … }} is a Blade command.
  - Message is not a variable in PHP for Blade to use.
- We do not want Blade to interpret these {{ … }}
  - They should be output verbatim so Vue.js can interpret them.



Fixing First Error

- Blade allows us to use @{{ … }}
  to specify that you do not want Blade to interpret the {{ … }}.
- Blade will strip off the @ and deliver just the {{ … }}
- Cool – it works.
- Laravel and Vue.js work together.

```
<!DOCTYPE html>
<html>
<head>
    <title>Keepers</title>
</head>

<body>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <h1>Enclosures</h1>
    <div id="root">
        <p>@{{ message }}</p>
    </div>

    <script>

        var app = new Vue({
            el: "#root",
            data: {
                message: "It works!",
            },
        });
    </script>
</body>
</html>
```

Enclosures

It works!

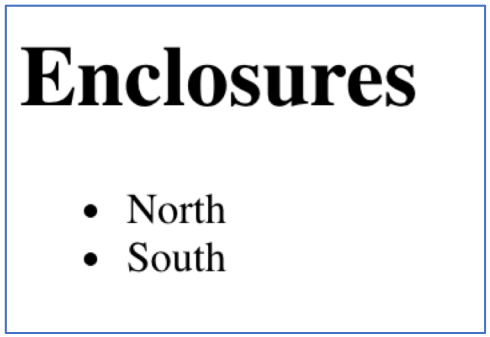
Step 2: Displaying the List of Enclosures

Lets Setup the Vue.js and HTML

```
<body>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <h1>Enclosures</h1>
    <div id="root">
        <ul>
            <li v-for="enclosure in enclosures">@{{ enclosure }}</li>
        </ul>

    </div>

    <script>

        var app = new Vue({
            el: "#root",
            data: {
                enclosures: ['North', 'South'],
            },
        });
    </script>
</body>
```

- Idea: enclosures variable will eventually hold the actual enclosure data.
- It will be populated via an AJAX request.
- But lets hard code it for now.

Result



- Okay, we have it working.
- Next:
  - Take out the hard coded data.
  - Issue an AJAX request when Vue.js starts up to get all the enclosures via an API call.

Setup The API Route

routes/api.php

```
Route::get('enclosures', 'EnclosureController@apiIndex')
    ->name('api.enclosures.index');
```

- This route is stored in the api route file.
  - All routes in here will have a /api prefix applied automatically.
  - Thus, the URI will be http://swanseazoo.test/api/enclosures
- This route should return a JSON object holding all the enclosures.

Setup The API Method in the Controller

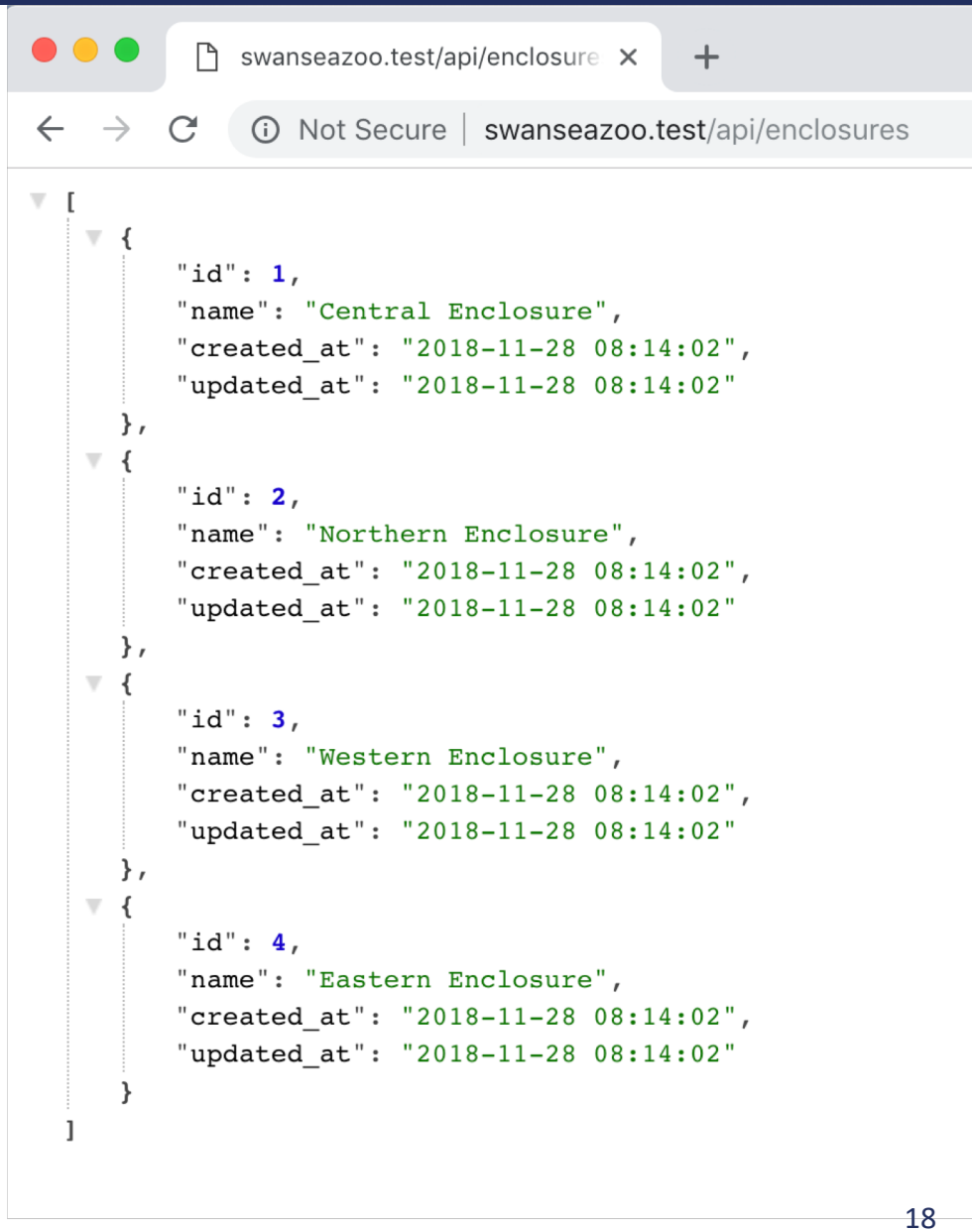
- The method should get all enclosures.
- Encode them in JSON and return them.
- Turns out this is trivial in Laravel.
- If you return a model (or collection of models) then they get changed to JSON automatically.
  - Note: previously, we passed them to a view.

EnclosureController.php:

```
public function apiIndex()
{
    $enclosures = Enclosure::all();
    return $enclosures;
}
```

Test the API Route

- As this is a GET route, we can test it easily by visiting the URI in the browser.
- Great we got JSON Data.



Axios

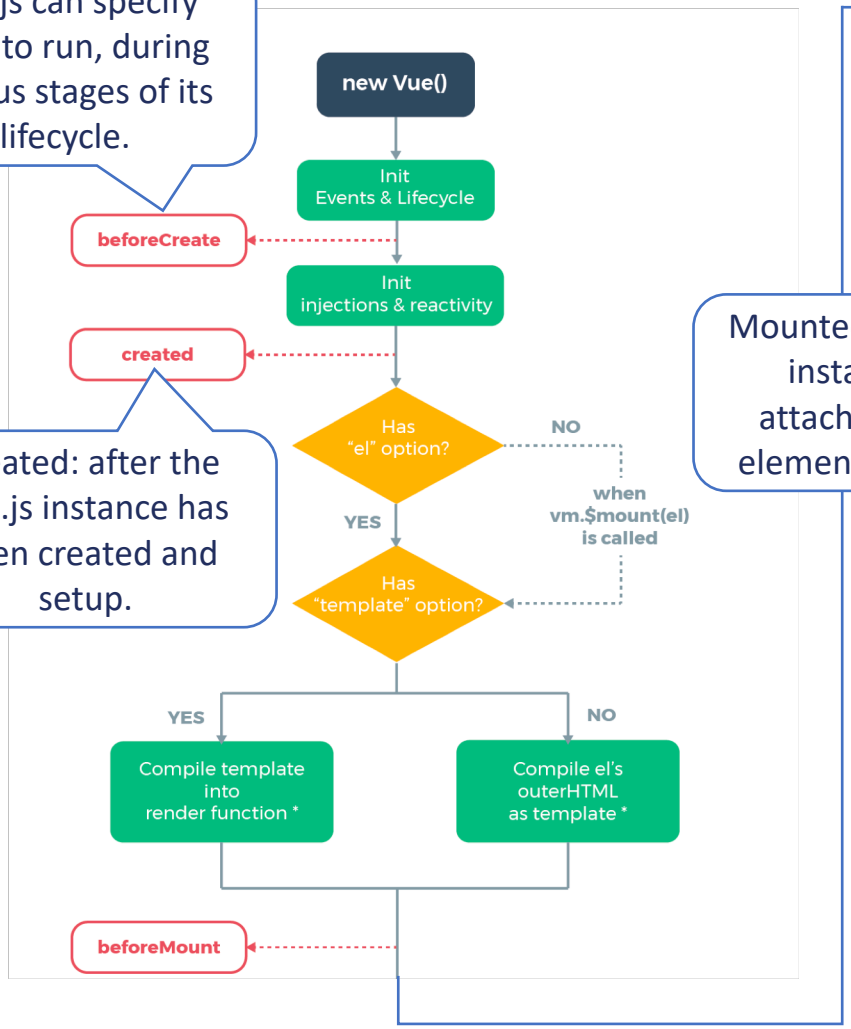
- Next, we need to issue an AJAX request to this route to get the enclosures.
- Axios is a nice library to do this. You can do it without, but it is a bit fiddly.
- We will import this using a CDN.

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

- Again, this is not really a proper way to do this.

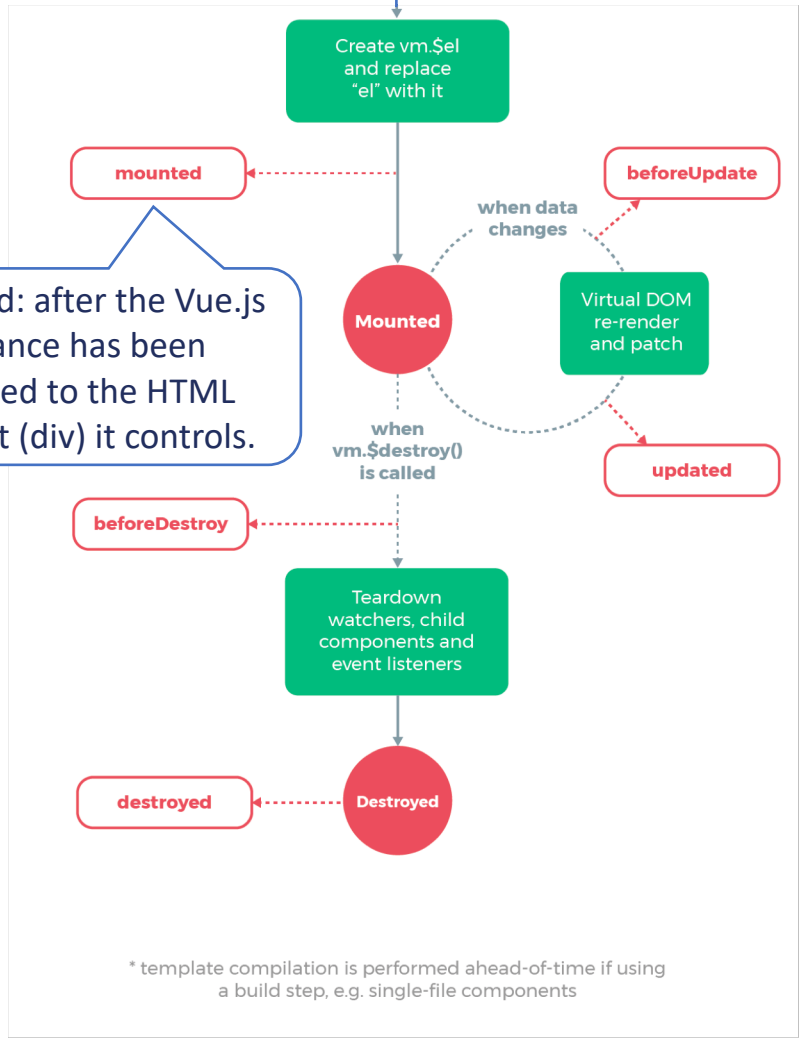
Vue.js Lifecycle

Vue.js can specify code to run, during various stages of its lifecycle.



Created: after the Vue.js instance has been created and setup.

Mounted: after the Vue.js instance has been attached to the HTML element (div) it controls.



Issuing an AJAX request

When mounted issue a GET ajax request.

Here we eave Blade calculate the route URI.

Axios is based on promises. Idea is that this code will run on a successful response from server.

We assume, on success, that the data that is returned is the JSON for all enclosures.

All we do is update our list of enclosure.

```
<script>
    var app = new Vue({
        el: "#root",
        data: {
            enclosures: [],
        },
        mounted() {
            axios.get("{{ route ('api.enclosures.index') }}")
            .then(response => {
                // handle success
                this.enclosures = response.data;
            })
            .catch(response => {
                // handle errors
                console.log(response);
            })
        },
    });
</script>
```

Error Handling

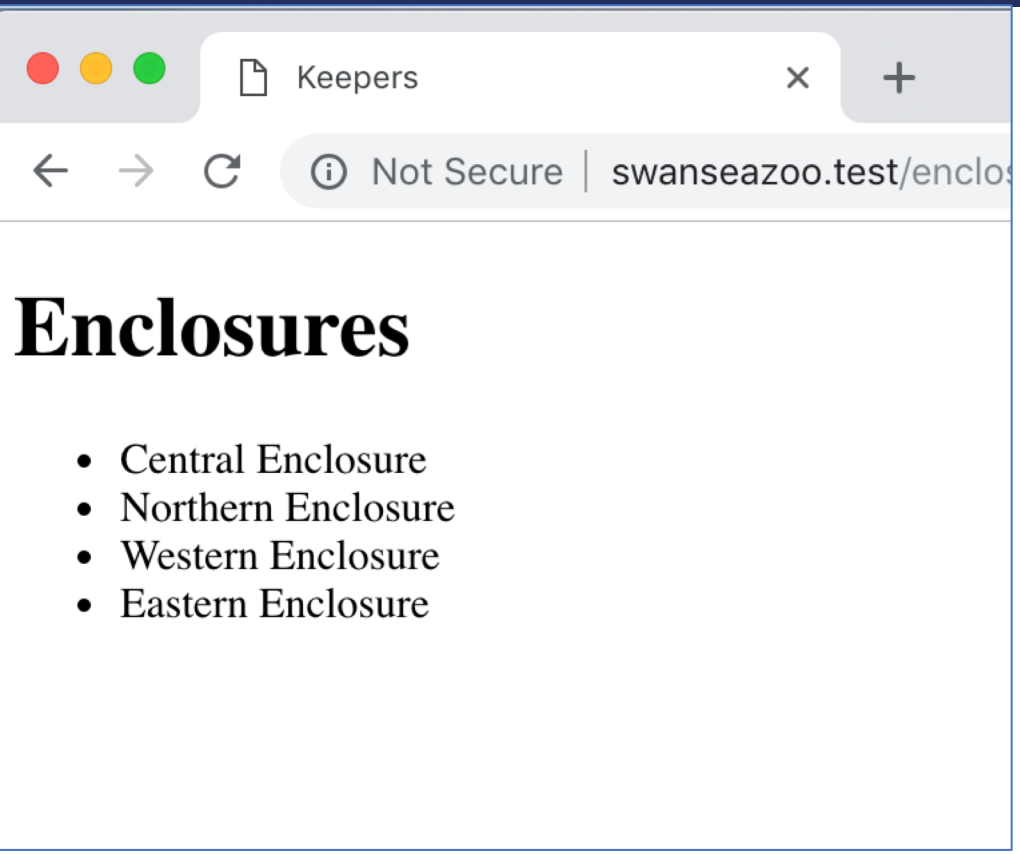
- We won’t look at errors here.
  - Errors are a MASSIVE part of this.
  - You must cater for them and use them properly.

Displaying The Data

- Next, lets update the display HTML of the page.
  - Remember, our enclosures are structured data (name, id, created_at, etc).
  - We only display the name.

```
<h1>Enclosures</h1>
<div id="root">
    <ul>
        <li v-for="enclosure in enclosures">@{{ enclosure.name }}</li>
    </ul>
</div>
```

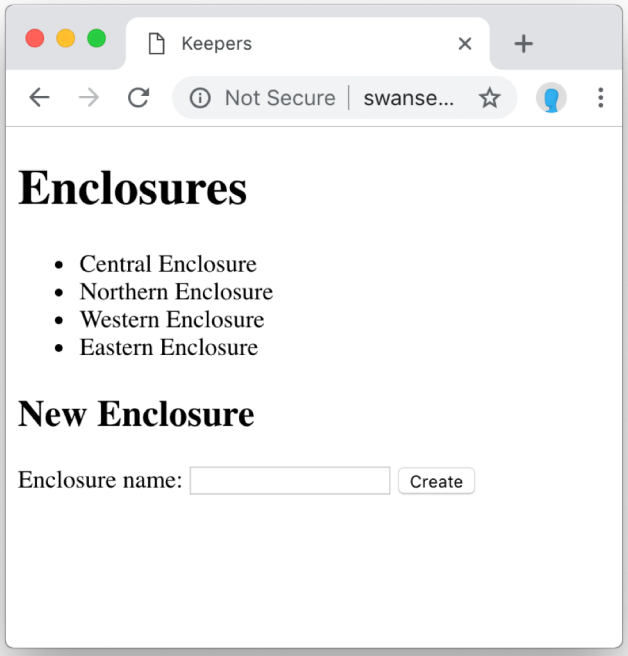
Result



- Okay, that works!
- We display a Vue.js page initially with no data.
- The Vue.js kicks off an AJAX request to retrieve data from our API.
- The retrieved data is then displayed.

Step 3: Adding New Enclosures

Next: Adding New Enclosures



- Now we want to add an input box and a button.
- When clicked we want to issue an AJAX request to create a new enclosure (and return it).
- We will then update our list in the DOM.

Setup The API Route

routes/api.php

```
Route::get('enclosures', 'EnclosureController@apiIndex')
    ->name('api.enclosures.index');


Route::post('enclosures', 'EnclosureController@apiStore')
->name('api.enclosures.store');
```

- This route is stored in the api route file.
  - All routes in here will have a /api prefix applied automatically.
- This is a post route. It should accept POST data, create a new enclosure and return it.

Setup The API Method in the Controller

- This method parses the HTML POST Data and uses it to create a new enclosure.
- It then returns the new enclosure by encoding it as JSON.
- We can test this out using various tools like Postman.
  - These allow you to issue manually crafted HTTP requests.

EnclosureController.php:

```
public function apiStore(Request $request)
{
    $e = new Enclosure;
    $e->name = $request['name'];
    $e->save();

    return $e;
}
```

Note: We skipped validation. You still need to. Do this.

You would return a JSON object that encodes your validation errors.

The JavaScript would then need to parse these and display them.

Update the Vue.js Page

```
<div id="root">
    <ul>
        <li v-for="enclosure in enclosures">@{{ enclosure.name }}</li>
    </ul>

    <h2>New Enclosure</h2>
    Enclosure name: <input type="text" id="input" v-model="newEnclosureName">
    <button @click="createEnclosure">Create</button>
</div>
```

- First, lets create an input text field and two-way bind it to a Vue.js variable.
- Then create a button which executes a method.

Update the Vue.js Page (Cont.)

- Next, we create the variable to hold the value in the text box.
- It starts of as the empty string.

```
<script>
    var app = new Vue({
        el: "#root",
        data: {
            enclosures: [],
            newEnclosureName: '',
        },
```

Update the Vue.js Page (Cont.)

- Now the main part.
- Create the method.

When activated, issue a POST request, with the value of the text box as the POST Data named ‘name’ (the name of the enclosure).

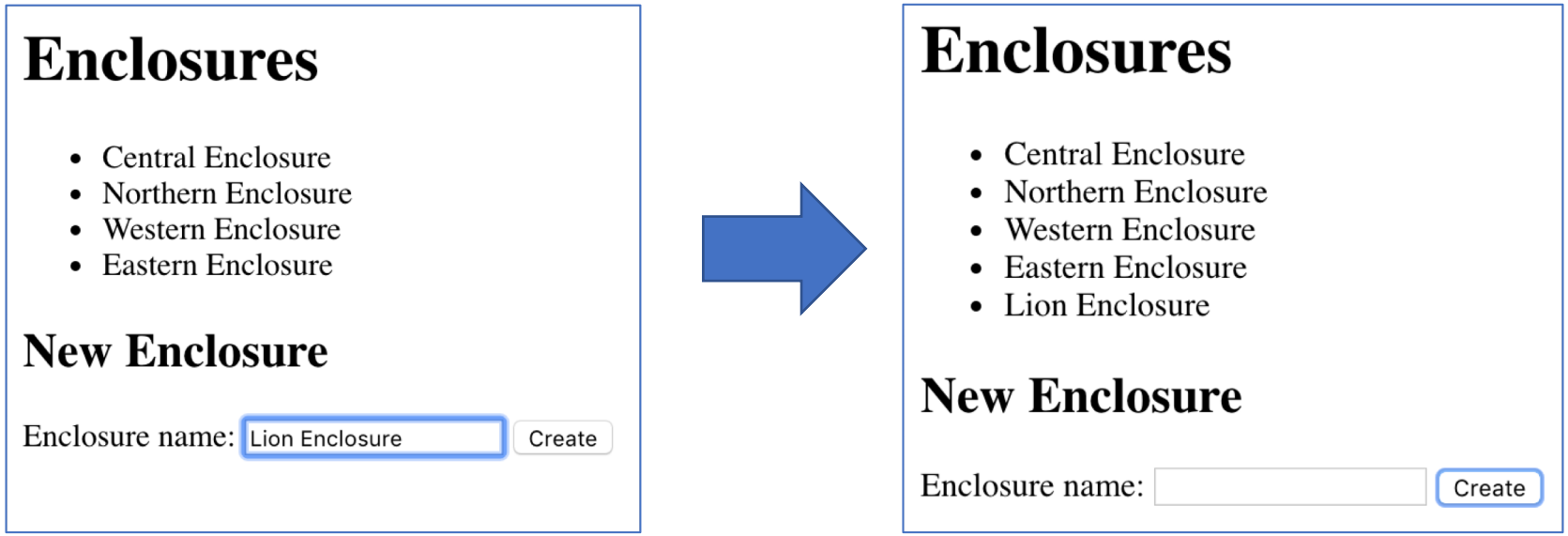
On success, assume the returned data is the new enclosure.

Add (push) it to the list.

Clear the input text box.

```
methods: {
    createEnclosure: function () {
        axios.post("{{ route ('api.enclosures.store') }}", {
            name: this.newEnclosureName
        })
        .then(response => {
            // handle success
            this.enclosures.push(response.data);
            this.newEnclosureName = '';
        })
        .catch(response => {
            // handle errors
            console.log(response);
        })
    }
}
```

Result



- So we can now add new enclosures.
- The page updates dynamically.

Whole Page (1 of 3)

```
<!DOCTYPE html>
<html>
<head>
    <title>Keepers</title>
</head>

<body>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
    <h1>Enclosures</h1>
    <div id="root">
        <ul>
            <li v-for="enclosure in enclosures">@{{ enclosure.name }}</li>
        </ul>

        <h2>New Enclosure</h2>
        Enclosure name: <input type="text" id="input" v-model="newEnclosureName">
        <button @click="createEnclosure">Create</button>
    </div>
```

Whole Page (2 of 3)

```
<script>
    var app = new Vue({
        el: "#root",
        data: {
            enclosures: [],
            newEnclosureName: '',
        },
        mounted() {
            axios.get("{{ route ('api.enclosures.index') }}")
            .then(response => {
                // handle success
                this.enclosures = response.data;
            })
            .catch(response => {
                // handle errors
                console.log(response);
            })
        },
```

Whole Page (3 of 3)

```
            methods: {
                createEnclosure: function () {
                    axios.post("{{ route ('api.enclosures.store') }}", {
                        name: this.newEnclosureName
                    })
                    .then(response => {
                        // handle success
                        this.enclosures.push(response.data);
                        this.newEnclosureName = '';
                    })
                    .catch(response => {
                        // handle errors
                        console.log(response);
                    })
                }
            }
        });
    </script>
</body>
</html>
```

Summary

- We have looked at at creating a very basic Vue.js page.
- We have done this in an ugly way to be honest.
  - You should use
    - Vue Components – these allow structuring of your JavaScript and splitting it up from the HTML.
    - Deliver VueJs, Axios, etc from our server and not a CDN.
    - We should transpile our resources – we will look at this next week.

Summary

- But you should now get the basic idea.
- Creating a SPA or just using Vue.js in places is not so hard, but it is more work.
  - You have Models, Controllers and View on your server.
  - You also end up needing controllers and views written in JavaScript/Vue.js for your front-end:
    - Controllers to manage all the AJAX.
    - View to give you reusable pieces of pages/Vue.js.
- It is a lot more work to create pages that use Vue.js
  - Massively more work for a whole SPA.
- But the results are great if done well.