We are going to remove the homes[] from the Vuex store and instead retrieve the homes using a web service.

1. Start by removing the homes[] from the Vuex store's state

2. Go to **HomeSearch.vue** component and add the homes array to the data section

```
  homes: [],
```

3. Now we need to create a service client. Under the services folder, create a HomeService.js file.

```
import axios from 'axios';

export default {

  search() {
    return axios.get('/homes')
  },

}
```

4. Now we need to import this service inside the <script> section of the HomeSearch.vue component. (inside script tag, but above export default)

```
import homeService from "../services/HomeService.js";
```

5. We need to call the service inside the created() method:

```
created() {
        homeService.search().then(response => {
          this.homes = response.data;
          this.isLoading = false;
        });
},
```

6. We need to change our filteredHomes() to NOT use the store, but instead use:

this.homes

We added it to the data section earlier.

7. Now the homes are being retrieved from the web service BUT the images are broken again. We need the DOM to load, before the image can be rendered. Functions are called after the DOM loads.

a. Create the following function in the method section:

```
methods: {
    getImageURL(pic) {
      return require('../assets/' + pic)
    }
}
```

b. Call the method from the template section above

```
<img v-bind:src="getImageURL(home.imageName)" />
```

**The images should work now.**

8. Now we are going to create a new component called **AddHome.vue**. This component will be a form that will add a home to the database.

9. Let's add a form to the AddHome.vue template

```
 <div id="addHomeform">
   <form  class="homeForm">

     <div class="form-group">
       <label for="mlsNumber">MLS Number:</label>
       <input id="mlsNumber" type="text" class="form-control" v-model="home.mlsNumber" />
     </div>

     <div class="form-group">
       <label for="address">Street Address:</label>
       <input id="address" type="text" class="form-control"
v-model="home.address.addressLine" />
     </div>

     <div class="form-group">
       <label for="city">City:</label>
       <input id="city" type="text" class="form-control" v-model="home.address.city" />
     </div>

     <div class="form-group">
       <label for="state">State:</label>
       <input id="state" type="text" class="form-control" v-model="home.address.state" />
     </div>

     <div class="form-group">
       <label for="zip">Zip:</label>
       <input id="zip" type="text" class="form-control" v-model="home.address.zip" />
     </div>

     <div class="form-group">
       <label for="price">Price:</label>
       <input id="price" type="text" class="form-control" v-model="home.price" />
     </div>

     <div class="form-group">
       <label for="bathrooms"># of Bathrooms:</label>
       <input id="bathrooms" type="text" class="form-control"
v-model.number="home.numberOfBathrooms" />
```

```
    </div>

    <div class="form-group">
      <label for="bedrooms"># of Bedrooms:</label>
      <input id="bedrooms" type="text" class="form-control"
v-model.number="home.numberOfBedrooms" />
    </div>

    <div class="form-group">
      <label for="description">Short Description:</label>
      <input id="description" type="text" class="form-control" v-model="home.shortDescription"
/>
    </div>

    <div class="form-group">
      <label for="imageName">Image Name (no path needed):</label>
      <input id="imageName" type="text" class="form-control" v-model="home.imageName" />
    </div>

    <button class="btn btn-submit">Submit</button>
    <button class="btn btn-cancel" type="cancel">Cancel</button>
  </form>
 </div>
```

10. Add a name to the script section and a data section. Explain that the errorMsg and home objects are to support the form.

```
 name: "add-home",

 data() {
  return {
    home: {
      mlsNumber: "",
      address: {
          addressLine: "",
          city: "",
          state: "",
          zip: ""
      },
      price: "",
      numberOfBathrooms: 0,
      numberOfBedrooms: 0,
      shortDescription: "",
      imageName: ""
    }
```

```
   }
  }
```

11. Add the following CSS

```
#addHomeform {

  margin-left: auto;
  margin-right: auto;
   width: 500px;

}

form input {
  width: 100%;
}

.homeForm {
  padding: 10px;
  margin-bottom: 10px;
}


.form-group {
  margin-bottom: 10px;
  margin-top: 10px;
}


.form-control {
  display: flex;
  align-items: flex-start;
  width: 100%;
  height: 30px;
  padding: 0.375rem 0.75rem;
  font-size: 1rem;
  font-weight: 400;
  line-height: 1.5;
  color: #495057;
  border: 1px solid #ced4da;
  border-radius: 0.25rem;
}
textarea.form-control {
  height: 75px;
```

```
  font-family: Arial, Helvetica, sans-serif;
}
select.form-control {
  width: 20%;
  display: inline-block;
  margin: 10px 20px 10px 10px;
}
.btn-submit {
  color: #fff;
  padding: 10px 24px;
  background-color: #38b412;
  box-shadow: 0 12px 26px 0 rgba(0,0,0,0.24), 0 17px 50px 0 rgba(0,0,0,0.19);
}

.btn-cancel {
  padding: 10px 24px;
  box-shadow: 0 12px 26px 0 rgba(0,0,0,0.24), 0 17px 50px 0 rgba(0,0,0,0.19);
}

.btn-submit:hover {

  color: #fff;
  padding: 10px 24px;
  background-color: #65f307;
  box-shadow: 0 12px 26px 0 rgba(0,0,0,0.24), 0 17px 50px 0 rgba(0,0,0,0.19);
}

.btn-cancel:hover {
  padding: 10px 24px;
  background-color: #65f307;
  box-shadow: 0 12px 26px 0 rgba(0,0,0,0.24), 0 17px 50px 0 rgba(0,0,0,0.19);
}

.status-message {
  display: block;
  border-radius: 5px;
  font-weight: bold;
  font-size: 1rem;
  text-align: center;
  padding: 10px;
  margin-bottom: 10px;
}
.status-message.success {
  background-color: #90ee90;
}
.status-message.error {
  background-color: #f08080;
}
```

12.  Now we are going to go create a View page called **AddAHome.vue**.  And embed the AddHome component we just built:

```
<template>
  <div>
    <add-home />
  </div>
</template>

<script>

import AddHome from '@/components/AddHome.vue';

export default {

  components: {
     AddHome
  }

}
</script>

<style>

</style>
```

13. Now that we have our View page and component to add a home, now we need to create a route so we can navigate to this page

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

{
     path: '/addHome',
     name: 'addHome',
     component: AddAHome,
     meta: {
       requiresAuth: false
     }
```

```
  },
```

14. Add a new router link to the nav bar

```
<router-link v-bind:to="{ name: 'addHome' }">Add Home</router-link>
```

15. Verify that everything routes appropriately through the menu…

16. Now we need to add an Axios web service client call inside HomeService:

```
addHome(home) {
  return axios.post('/homes', home)
}
```

17. Now we need to update the AddHome.vue to call the above web service. Walk through the data section again and show the v-model to the home in the data section. As data is bound the each form field, we need a way to send the populated home object to the web service.

a. First we need to import the HomeService:

```
import homeService from '../services/HomeService';
```

b. Next we need to write a function that calls the web service. We do this using a combination of the v-on submit at the form, and creating an event-handling function that will call the service

```
methods: {
  submitForm() {

    // add
    homeService.addHome(this.home)
```

```
      .then(response => {
        if (response.status === 201) {
          this.$router.push('search');
        }
      })
      .catch(error => {
        //do something
        console.log(error);
      });
  },

},
```

c. Now we call this function by using the v-on attribute on the form

```
    <form v-on:submit.prevent="submitForm" class="homeForm">
```

18. Test the form submission and debug if necessary. Everything should work and the user routed back to the search page to see the new home that was added.

19. Now we are going to hook up the cancel button. We want the cancel button to route them back to the search page. We can do this by attaching an event handler method to the Cancel button,

```
<button v-on:click="cancelForm" class="btn btn-cancel" type="cancel">Cancel</button>
```

and call the following function:

```
  cancelForm() {
    this.$router.push('search');
  },
```

20. So let's go back to the form submission, submitForm(). Point out the fact that if we get an error, we aren't really doing much. Ask the students if they can come up with a way to notify the user that an error occurred (like not being able to reach the server, or bad request)

Let the students know that the response object that is part of the promise can be checked for errors and we could check for errors and put a error message at the top of the form if it has a value..

21. So let's create a handleError() that we can call to interrogate the response

```
  handleErrorResponse(error) {
    //figure out the error message text
    if (error.response) {
      // set a variable to msg text
      this.errorMsg = 'Error adding a new home. Error: ' + error.response.status;
    }
    else if (error.request){
      //set a variable to msg text
      this.errorMsg = 'Error adding a new home. Server could not be reached.';
    }
    else {
      // catch all - return generic error msg
      this.errorMsg = 'Java Green has left you high and dry. You would be better off finding another developer who cares.';
    }

  }
```

We need to create a variable in our data section to set if we have an error

```
errorMsg: '',
```

22. Now we need to call this function from our submitForm() inside the catch block:

```
    this.handleErrorResponse(error);
```

23. Now we just need to add some code to the template to check if the errorMsg has a value or not.

```
<div class='status-message error' v-show="errorMsg != "">{{errorMsg}}</div>
```

24.  To test, we can shut off the server. Submit a request, wair a few seconds and the error message appears.