DELETE QUESTION

In this lesson we're going to make the **Delete** button work like we have in our non SPA application. To do that we need to listen to the `click` event in `QuestionExcerpt` component and hit the *delete question* API endpoint.

Alright, let's go ahead and open up our terminal then create a new branch.

```
git checkout -b lesson-61
```

After that start up the Laravel mix by running this command:

```
npm run watch
```

1. Attaching the click event on Delete button

Open the `QuestionExcerpt.vue` then find the **Delete** button. Now we no longer need the `form` tag to send *DELETE* request. Instead we're going to use `axios` library to hit the *Delete Question* endpoint. So just move the authorization stuff from the form to the `button` then remove the `form` tag.

You can then change the `type` attribute in the `button` from `submit` to `button`. And attach the `click` event to handle `destroy` method

```
<div class="ml-auto">
  <router-link ...>Edit</router-link>
  <button
      v-if="authorize('deleteQuestion', question)"
      class="btn btn-sm btn-outline-danger"
      @click="destroy">Delete</button>
</div>
```

The `destroy` method will show a confirmation dialogue before question actually removed from database like we have in `Answer` component.

We don't need to define that method, because we've already have it. Just in case you forgot you can open `js/mixins/modification.js`. It'll show a confirmation message by making use of `toast` plugin. If "YES" button being hit it'll call `delete` method which will actually remove the resource completely.

```
destroy () {
  this.$toast.question('Are you sure about that?', "Confirm", {
    // ...
    buttons: [
      ['<button><b>YES</b></button>', (instance, toast) => {

        this.delete();

        instance.hide({ transitionOut: 'fadeOut' }, toast, 'button');

      }, true],
      ['<button>NO</button>', function (instance, toast) {

        instance.hide({ transitionOut: 'fadeOut' }, toast, 'button');

      }],
    ]
  });
},
```

You might be thinking to use the `modification.js` mixin in the `QuestionExcerpt` component. But in my opinion that is not a good idea. Because you'll see in this mixin you also have `update` and other methods which useless in this component. What we really need in this component are just `destory` and `delete` method.

2. Refactoring the modification mixin

So let's make a small refactor on this `modification.js` mixin by moving the `destroy` and `delete` method into other mixin. Now we'll have these methods in this mixin.

```
// ...
export default {
    // ...
    methods: {
        edit () { ... },

        cancel () { ... },

        setEditCache () {},
        restoreFromCache () {},

        update () { ... },

        payload () {},
    }
}
```

Then let's create a new file in `mixins` directory called `destory.js`. Inside this file define `export default`. Then add `methods` property and then paste the previous methods.

```
export default {
    methods: {
        destroy () {
            this.$toast.question('Are you sure about that?', "Confirm",
{
            timeout: 20000,
            close: false,
            overlay: true,
            displayMode: 'once',
            id: 'question',
            zindex: 999,
            title: 'Hey',
            position: 'center',
            buttons: [
                ['<button><b>YES</b></button>', (instance, toast) => {

                    this.delete();

                    instance.hide({ transitionOut: 'fadeOut' }, toast,
'button');

                }, true],
                ['<button>NO</button>', function (instance, toast) {

                    instance.hide({ transitionOut: 'fadeOut' }, toast,
'button');

                }],
            ]
            });
        },

        delete () {}
    }
}
```

Back to `modification.js` mixin then use the `destory.js` mixin inside it.

```
// ...
import highlight from './highlight';
import destroy from './destroy';

export default {
    mixins: [highlight, destroy],
    // ...
}
```

3. Using the destory.js mixin in QuestionExcerpt component

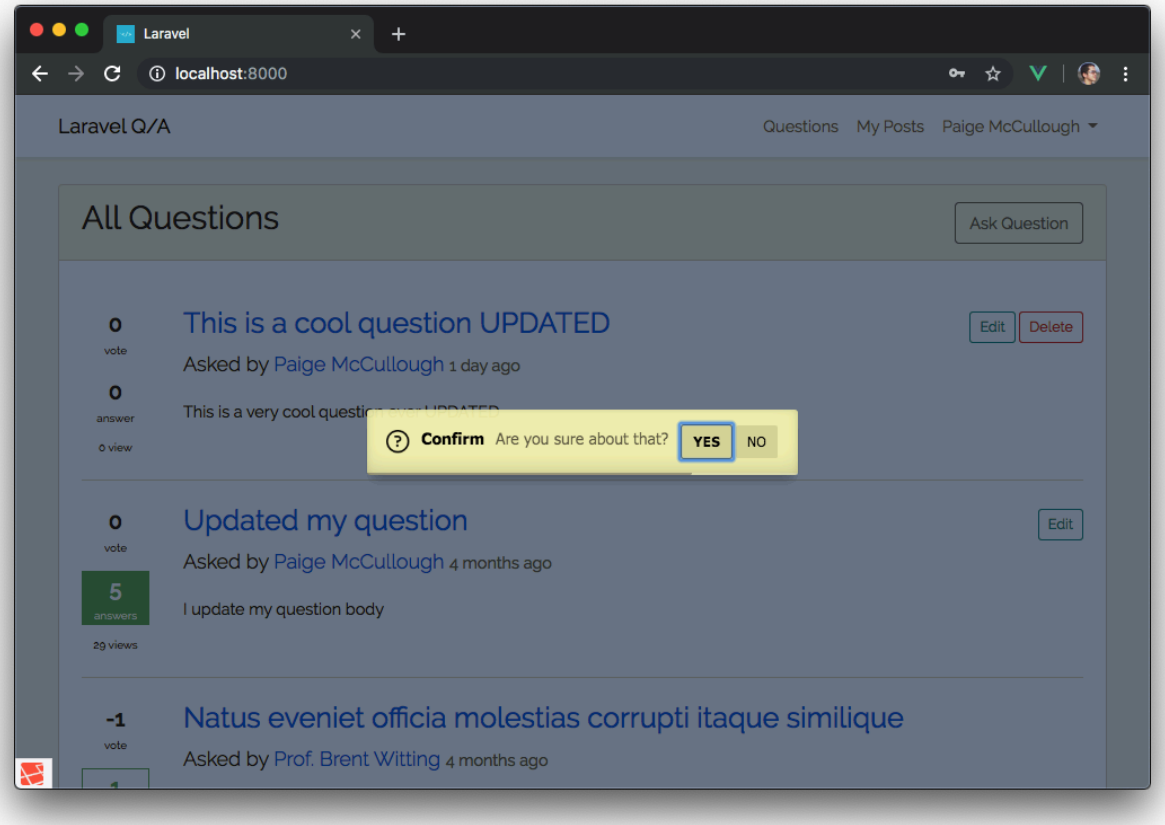
Now we can use this `destory.js` mixin in our `QuestionExcerpt` component to make the **Delete** button actually working.

Back to `QuestionExcerpt` and use the `destroy.js` mixin inside the script section.

```
import destroy from '../mixins/destroy'

export default {
    mixins: [destroy],
    // ...
}
```

Now if you save all changes then back to your browser and try to hit the **Delete** button. You'll see a nice confirmation dialog. But if you choose **YES** option there's noting happen. This because the `delete` method which is called when **YES** button being hit is just empty method by default.



So to actually remove the question from database we need to redefine (override) the `delete` method in the `QuestionExcerpt` component.

Go to `methods` property and define the `delete` method. The request method is going to be `delete` while the endpoint that is going to hit is `"/questions/" + this.question.id`. When successful we can show success message and emit `deleted` event.

```
delete () {
    axios.delete("/questions/" + this.question.id)
        .then(({data}) => {
            this.$toast.success(data.message, "Success", { timeout:
2000 });
            this.$emit('deleted');
        });
}
```

Here because we emited a `deleted` event we need to listen that in the parent component.

4. Listening the `deleted` event

Let's open the `Questions.vue` file. Then in `question-exceprt` component calling, let's listen to `deleted` event and do something. In this case let's call `remove` method and pass the `index` in.

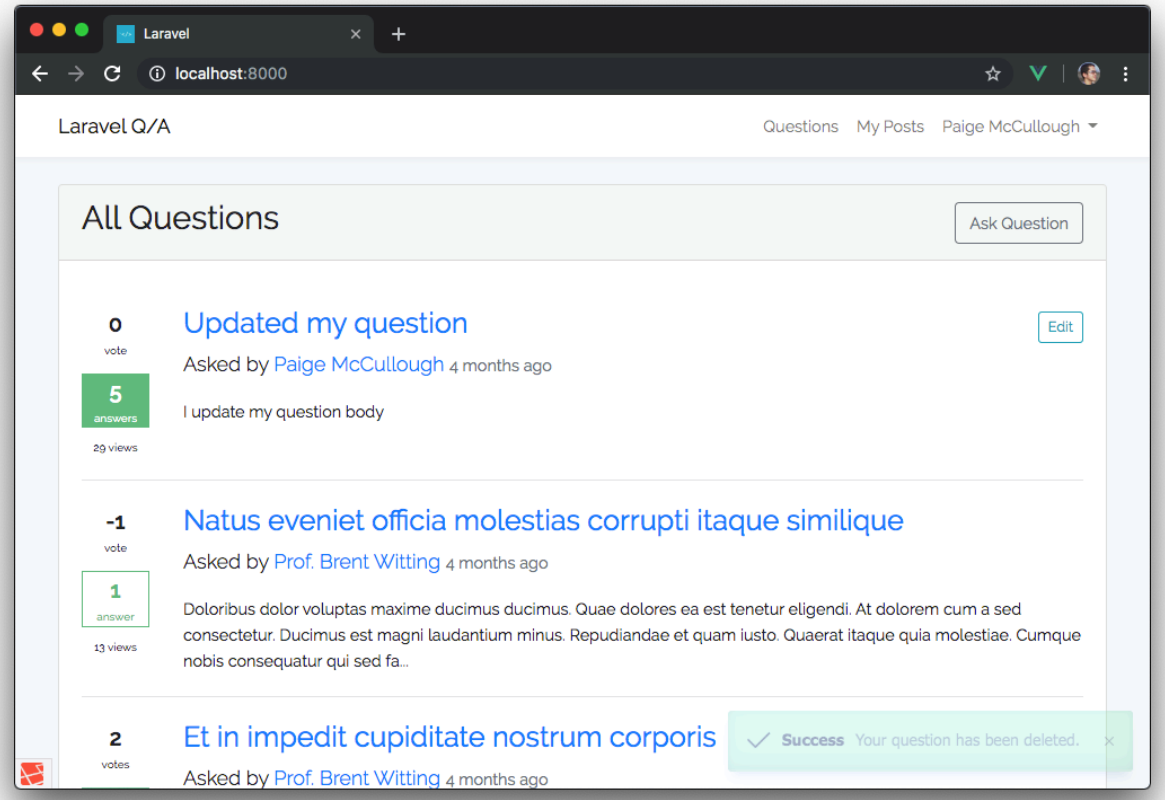
To capture the index in the loop we need to explicitly add it in the left side of the `v-for` directive.

```
<template>
    ...
    <question-excerpt
          @deleted="remove(index)"
          v-for="(question, index) in questions"
          :question="question"
          :key="question.id"></question-excerpt>
    ...
</template>
```

Now go to script section and define the `remove` method in `methods` property. In this method we simply remove the array element by the `index` specified. And we also need to increment the `count` variable.

```
remove (index) {
  this.questions.splice(index, 1)
  this.count--
}
```

Now if you save the change and try again the **Delete** button. The question will be successfully removed and you'll also see the success flash message pop up. You can also refresh the page to make sure that the question is actually removed from the database.



So now that we have a full set CRUD functionalities in our single page application. We can see all questions with pagination, we can add a new question, edit and delete existing question.

Let's move on to the next lesson where we're going to fix some issues in `QuestionPage` component. Don't forget to save all changes that you made in your git repo.

```
git add .
git commit -m "Make the delete button in QuestionExcerpt functional"
git push origin lesson-61
```