CREATING PAGINATION COMPONENT

In this lesson we're going to add a pagination in our QuestionsPage component like we have in our non spa application. But this time we're not going to use a standard pagination but instead we're going to use a simple pagination which only contains two buttons to move back and forward.

Alright, let's go ahead and open up our terminal. Then create a new branch for our work today.

```
git checkout -b lesson-58
```

After that let's tell Laravel Mix to watch file changes that we make in our frontend code.

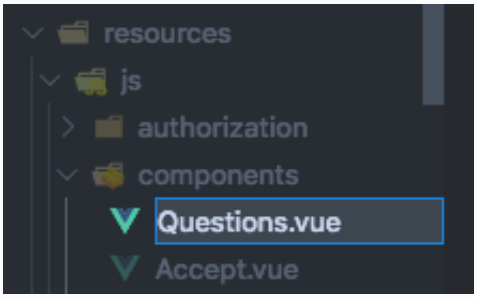
```
npm run watch
```

REFACTORING THE QUESTIONSPAGE COMPONENT

Now before moving forward, let's refactor our `QuestionsPage` component. Because in my opinion every component inside `pages` represents a single page and it acts like container for other component. So it would be better to make it as simple as possible. And that's why we need to hide the complexity in other components.

1. Creating Questions component

So let's navigate to `js/components` directory. In this directory let's create a new file. Let's call it `Questions.vue`. Basically we'll move almost everything inside the `QuestionsPage` component into this file.



2. Moving the template

Back to `QuestionsPage.vue` inside `js/pages` directory. And in this file we'll move the `card-body` element to our `Questions` component. We can do that by select and copy to clipboard these markup:

```
<div class="card-body">
  <div v-if="questions.length">
    <question-excerpt ...></question-excerpt>
  </div>
  <div v-else class="alert alert-warning">
    <strong>Sorry</strong> There are no questions available.
  </div>

  <!-- pagination goes here -->
</div>
```

Then replace it with `Questions` component.

```
<questions></questions>
```

Switch back to `components/Questions.vue`. Inside this file let's add `template` tag then add a `div` inside it. Inside the `div` we can paste the markup. We can then move the pagination comment outside the `card-body`.

```
<template>
     <div>
        <div class="card-body">
            <div v-if="questions.length">
                <question-excerpt v-for="question in questions"
:question="question" :key="question.id"></question-excerpt>
            </div>
            <div v-else class="alert alert-warning">
                <strong>Sorry</strong> There are no questions
available.
            </div>
        </div>
        <!-- pagination goes here -->
     </div>
 </template>
```

3. Moving the script

Switch back to `pages/QuestionsPage.vue`. Then cut everyting inside the `script` section. Since we called `questions` component in our template so let's import it in here.

```
<script>
import Questions from '../components/Questions.vue'

export default {
    components: { Questions }
}
</script>
```

Now we can go back to `components/Questions.vue`. Add `script` tag then paste the previous code. Since we're in the same directory with `QuestionExcerpt` component we can change the path in `import` statement to `./QuestionExcerpt.vue`.

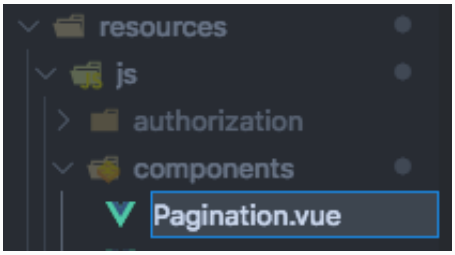
```
<script>
import QuestionExcerpt from './QuestionExcerpt.vue'

export default {
    components: { QuestionExcerpt },

    data () {
        return {
            questions: []
        }
    },

    mounted () {
        this.fetchQuestions();
    },

    methods: {
        fetchQuestions () {
            axios.get('/questions')
                 .then(({ data }) => {
                     this.questions = data.data
                 })
        }
    }
}
</script>
```

Save all changes and make sure you don't have any issue.

CREATING THE PAGINATION COMPONENT

1. Creating Pagination template

Now let's finally working with Pagination. Navigate to `resources/js` folder. Then create a brand new file called `Pagination.vue`.



In this file let's add `template` tag then add a `div` inside it. Let's add bootstrap classes classes `row` as well as `align-items-center` to make the items in center alignment.

```
<template>
    <div class="row align-items-center">

    </div>
</template>
```

Then inside the `row` div let's add 3 columns by making use the bootstrap `col` class.

```
<div class="row align-items-center">
  <div class="col"></div>
  <!-- 1st column -->

  <div class="col"></div>
  <!-- 2nd column -->

  <div class="col"></div>
  <!-- 3rd column -->
</div>
```

In the *first* column we'll add a button that is going to use to navigate to the previous page or to get the newer questions. We'll use bootstrap class `btn btn-outline-secondary` in the `button`. And the text is going to be **Newer**.

```
<div class="col">
  <button class="btn btn-outline-secondary">Newer</button>
</div>
<!-- 1st column -->
```

In the *second* column we'll show the page info. For now let's add a static text "Page 1 of 5". Let's also make the text align in center by adding the bootstrap class `text-center`.

```
<div class="col text-center">
  Page 1 of 5
</div>
<!-- 2nd column -->
```

In the *third* column we'll add a button that is going to use to navigate to the next page or to get the older questions. The button's style will be the same with **Newer** button. And the text is going to be **Older**. We also need to place the button on the right side that's why we'll add bootstrap class `text-right`.

```
<div class="col text-right">
  <button class="btn btn-outline-secondary">Older</button>
</div>
<!-- 3rd column -->
```

Here you can see the complete code in `Pagination.vue`.

```
<template>
    <div class="row align-items-center">
        <div class="col">
            <button class="btn btn-outline-secondary">Newer</button>
        </div>
        <!-- 1st column -->

        <div class="col text-center">
            Page 1 of 5
        </div>
        <!-- 2nd column -->

        <div class="col text-right">
            <button class="btn btn-outline-secondary">Older</button>
        </div>
        <!-- 3rd column -->
    </div>
</template>
```

2. Calling The Pagination in Questions component

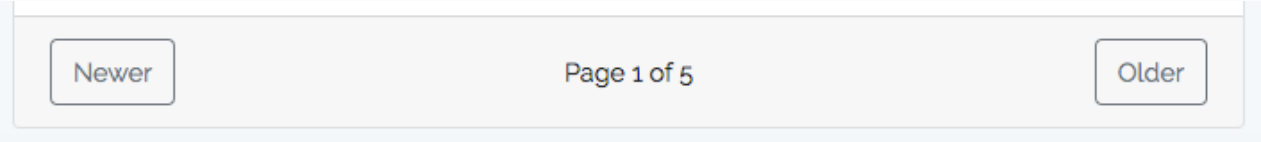
Now we can switch to `Questions` component. In this component let's import and register the `Pagination` component in the script section.

```
import QuestionExcerpt from './QuestionExcerpt.vue'
import Pagination from './Pagination.vue'

export default {
    components: {
        QuestionExcerpt,
        Pagination
    },
    // ...
}
```

Next, let's add a `card-footer` right after the `card-body`. Then add the `Pagination` inside it.

```
<div class="card-body">...</div>
<div class="card-footer">
  <pagination></pagination>
</div>
```

Now if you save the change and see in the browser. You'll see the pagination like this:



3. Pagination props

Now we have pagination component added at the bottom of our `Questions` component. The next thing that we need to do is to make it actually working. But before we do that let's go ahead and open Postman. Select the **Display all questions** request then hit the **Send** button.

In the response beside `data` you'll also see `links` and `meta` information. We can making use these information to make our pagination actually work.

```
{
    "data": [
    ],
    "links": {
        "first": "http://localhost:8000/api/questions?page=1",
        "last": "http://localhost:8000/api/questions?page=1",
        "prev": null,
        "next": null
    },
    "meta": {
        "current_page": 1,
        "from": 1,
        "last_page": 1,
        "path": "http://localhost:8000/api/questions",
        "per_page": 10,
        "to": 9,
        "total": 9
    }
}
```


Let's go back to `Pagination` component. Add `script` tag and inside it, let's define `links` and `meta` in `props` property.

```
<script>
export default {
    props: ['meta', 'links']
}
</script>
```

4. Binding meta and links from API response

Now we have defined `meta` and `links` in `Pagination` component. We can go back to `Questions` component then pass the meta and links information from our api response to the `Pagination` component.

We can do that by defining local `meta` and `links` in `data` method.

```
data () {
  return {
    questions: [],
    meta: {},
    links: {}
  }
},
```

Then in the `fetchQuestions` method, let's assign the meta and links information that we got back from our api response to our `meta` and `links` variables.

```
fetchQuestions () {
    axios.get('/questions')
            .then(({ data }) => {
                this.questions = data.data;
                this.meta = data.meta;
                this.links = data.links;
            });
}
```

Now we can go to our template and binding the `meta` and `links` to `pagination` component.

```
<pagination :meta="meta" :links="links"></pagination>
```

Alright, now let's finally make our pagination really functional. Let's do from the easiest part which is showing the *pages info* in the second column of the `Pagination` component.

SHOWING THE PAGES INFO

Back to `Pagination` component. Then in the script section, let's add `computed` property. In this property let's let's define a new method called `pagesInfo`.

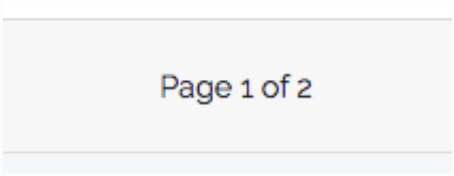
In this method we simply return a string contains "**Page 1 of 5**" like we have in the second column. We can then replace the first number (before 'of') with `${this.meta.current_page}` and replace the second number (after 'of') with `${this.meta.last_page}`.

```
computed: {
    pagesInfo () {
        return `Page ${this.meta.current_page} of
${this.meta.last_page}`
    }
},
```

Now we can jump to our template then change the static text in the second column with `pagesInfo`.

```
<div class="col text-center">{{ pagesInfo }}</div>
<!-- 2nd column -->
```

If you save the change then see in you browser. Now you'll see the pages info has changed.



If you had message "**Page 1 of 1**" you can adjust the number of `paginate` method in the `index` method of your `Api/QuestionsController`.

```
$questions = Question::with('user')->latest()->paginate(5);
```

Now we have the pages info working. The next thing that we need to do is to make the left and right buttons actually working so that we can navigate to next or previous pages.

MAKING THE LEFT AND RIGHT BUTTONS FUNCTIONAL

1. Setting up the button's state

Let's switch over to `Pagination` component then go to template. We can disable the left button (Newer) if we have the current page is the first page. Otherwise we can disable the right button (Older) if the current page is the last page.

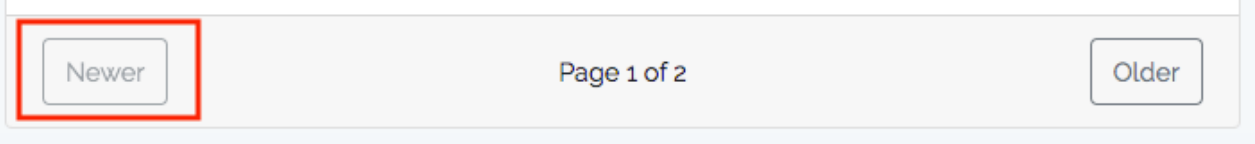
To achieve this we can making use the `disabled` attribute bindings to `isFirst` and `isLast` to respective buttons.

```
<div class="col">
    <button :disabled="isFirst" ...>Newer</button>
</div>
<!-- 1st column -->
...
<div class="col text-right">
    <button :disabled="isLast" ...>Older</button>
</div>
<!-- 3rd column -->
```

Now we can jump to script section then define the `isFirst` and `isLast` in the `computed` property. The `isFirst` method will return `true` if `current_page` in `meta` variable is equals to one. While `isLast` method will return `true` if the `current_page` is the last page. We can get the last page from `last_page` in `meta` variable.

```
computed: {
    // ...
    isFirst () {
        return this.meta.current_page === 1;
    },

    isLast () {
        return this.meta.current_page === this.meta.last_page;
    }
}
```

If you save the change then see in you browser. Now you'll see the left button being disabled because by default the current page is fist page.



OK, after we've setup the buttons's state we can now make them actually functional.

2. Switch pages functionality

In the script section of `Pagination` component, let's add `methods` property. Then inside it let's define a method called `switchPage`.

In `switchPage` method we can switch the page by making use the vue router `$router.push` method. In the `push` method call we can specify the destination information such as route name and query string.

For the rout name we can set it to `questions` so that we can go to all questions page. For the query string we can specify the page number.

```
methods: {
    switchPage () {
        this.$router.push({
            name: 'questions',
            query: {
                page: this.meta.current_page
            },
        });
    }
}
```

After `switchPage` method defined now we can define another method to actually move to the previous page. Let's call it `prev`.

Remember in the `SwitchPage` we got the `current_page` and set it to the `page`. So in the `prev` method we need to decrement the `current_page`. But before we do that we need to make sure that it's going to happen if the page is not the first page.

After that we can call the `switchPage` to actually switch the previous page.

```
methods: {
    switchPage () {...},

    prev () {
        if (! this.isFirst) {
            this.meta.current_page--;
        }
        this.switchPage();
    }
}
```

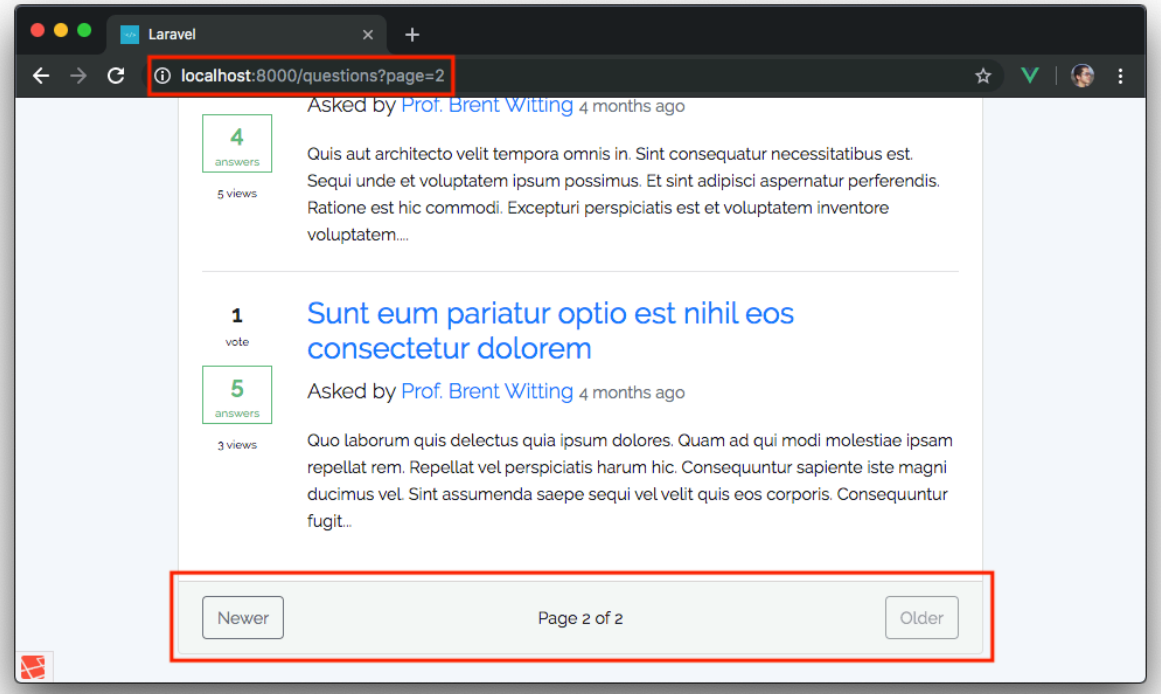
Let's define another method to actually move to the next page. Let's call it `next`. The logic for this method is almost the same with `prev` method. In this method we need to increment the `current_page` and it's going to happen if the page is not the last page.

```
methods: {
    switchPage () {...},
    prev () {...},
    next () {
        if (! this.isLast) {
            this.meta.current_page++;
        }

        this.switchPage();
    },
}
```

Now we can jump to our template. We can attach `click` event to `preve` and `next` method to the respective buttons.

```
<div class="col">
    <button :disabled="isFirst" @click="prev" ...>Newer</button>
</div>
<!-- 1st column -->
...
<div class="col text-right">
    <button :disabled="isLast" @click="next" ...>Older</button>
</div>
<!-- 3rd column -->
```

Now let's save the changes and go to the browser. If you hit the *Older* or *Newer* button you can see the url changing in the browser address bar. You'll also notice that the button become inactive when it reach at the first or the last page.



But unfortunately we can't see any changes in our table. So what's going on here? Well, to fix this issue we need to manually watch the route change.

3. Watch the route change

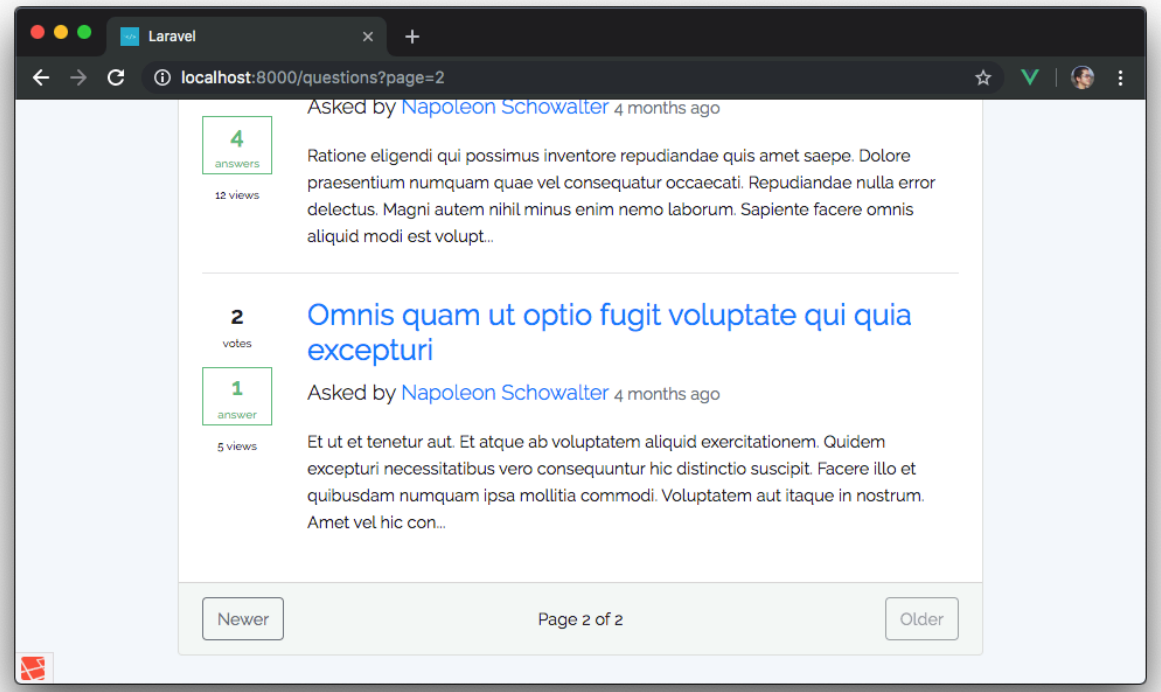
Let's switch back to `Questions` component then jump to the script. In here, let's add `watch` property. We can tell vue to watch changes whenever the route changes it should call the `fetchQuestions` method.

```
watch: {
    "$route": 'fetchQuestions'
}
```

Also in `fetchQuestions` method we need to specify in the second argument an object contains query string like so:

```
axios.get('/questions', { params: this.$route.query })
```

Alright, let's save the change and head over to the browser. Now you'll see the data on the table also changes whenever you switch the page.



SUMMARY

So now that we have pagination in our frontend working nicely. We can go back and forth without need to reload the page. In the next lesson we're going to work with Question Form and make if functional in our Single Page Application.

Alright, let's commit our work today into our git repo.

```
git add .
git commit -m "Create Pagination component"
git push origin lesson-58
```