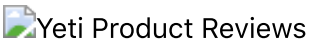
# REST API Lecture Notes

During lecture today you will be building out a product reviews application. The product at the top of the page is a Tech Elevator Yeti. On the same page you will list out a number of reviews. These reviews will come from a mockAPI and you will have the ability to list, create, read, update, and delete. During lecture you will be building out all of the REST API calls that perform those actions.

## Application Walkthrough

Before we get into building out the application It might be a good idea to run the final solution and just give a quick walkthrough of what the application does.


Yeti Product Reviews

We have a fake product page that display the Tech Elevator Yeti. This is a Vue application that will pull data from mockAPI and perform the following:

- List all reviews on the product page
- List number of reviews
- Add New Review
- Update Existing Review
    - We use the update to also Read (GET) the current review for an id
- Delete Review

## Starter Code

The starter code for this lecture is located in this directory in a folder named `product-reviews`. Your starter code looks exactly like the students starting code. All of the methods that need to perform a REST call with our mockAPI are empty and the code for each one is in this document.

## mockAPI

To follow along you can use the mockAPI clone link below to create your own copy of the Product Reviews project. I would also make sure the students do this as well. They have the link in their lecture code README.

https://www.mockapi.io/clone/5c51c520e315030014ea3415

Once you have the mockAPI cloned it might be worth walking through the schema and data before moving on to the code walkthrough.

## Lecture

The first thing you will need to do is update a variable inside of `src/components/ProductPage.vue`. You (and the students) will want to update the API_URL with your mockAPI endpoint for the reviews resource and it should looks something like this.

```
API_URL: "http://5c51c520e315030014ea3414.mockapi.io/api/reviews";
```

## List Reviews

The first thing we want to do is list out all of the review from our API in the `src/components/ProductReviews.vue` component. I would open this component and ask the students where they think the code to retrieve a list of reviews should go. If they went through the tutorial they should know that the answer is in the `created()` method. If you want to review the lifecycle hooks with the students you can walk through the documentation.

```
created() {
  // load the reviews
  fetch(this.apiURL)
    .then(response => {
      return response.json();
    })
    .then(reviews => {
      this.reviews = reviews;
    })
    .catch(err => console.error(err));
}
```

After this you should be able to see a list of reviews on the product page.

## Add Review

On the product page you will see 2 components in the template. You will notice that there is a `v-if` directive on both and the `showAddProductForm` variable is what determines what view is being shown.

```
<save-review v-if="showAddProductForm" v-on:showReviews="showReviews"
:apiURL="API_URL" :reviewID="reviewID"/>
<product-reviews
  v-if="!showAddProductForm"
  :apiURL="API_URL"
  v-on:addReview="addReview"
  v-on:editReview="editReview($event)"/>
```

When the user clicks on the add review button it emits a custom event called `addReview`.

```
<a href="#" class="add-review" v-on:click="$emit('addReview')">
  <i class="far fa-address-card"></i> Add Review
</a>
```

That is handled in the `ProductPage.vue` component. That event handler updates the `showAddProductForm` variable to true which displays our add new review form. That form is located in our `SaveReview.vue` component. I created it this way so we had one form for creating and updating a review.

```
methods: {
  addReview() {
```

```
      this.showAddProductForm = true;
    }
  }
```

In the component `SaveReview.vue` you will need to update the createReview method to POST a new review to our REST API. This might also be a good time to show how the review gets populated using the `v-model` directive.

```
createReview() {
  fetch(this.apiURL,{
    method: 'POST',
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(this.review)
  })
  .then((response) => {
    if(response.ok) {
      this.$emit('showReviews');
    }
  })
  .catch((err) => console.error(err));
}
```

### Read Review

I mentioned this earlier but instead of having 2 different forms for creating and updating a review we just have one since they both ask for the same data. When we click on the add review button a `reviewID` of 0 is passed which tells our `SaveReview` component that we are working with a new review.

When the user is looking at a list of reviews and clicks the edit button (below the review avatar) it will pass the `reviewID` of that review which tells our `SaveReview` component that we are working with an existing review. That `reviewID` variable is what drives our logic here and determines what method to call, `createReview()` or `updateReview()`.

```
saveReview() {
  this.reviewID === 0 ? this.createReview() : this.updateReview();
}
```

When we are updating an existing review we need to take the `reviewID` that was passed in and perform a GET against our API to get the current review data. Again we are going to perform this logic in the `created()` method. This gets called when the component is loaded so we need to make sure that the `reviewID` is not 0 before making the call to our API.

```
created() {
  if( this.reviewID != 0 ) {
    fetch(this.apiURL + '/' + this.reviewID)
      .then((response) => {
```

```
        return response.json();
      })
      .then((review) => {
        this.review = review;
      })
      .catch((err) => console.error(err));
  }
}
```

### Update Review

Now that we have loaded the current review you can make some changes and click save review. This will trigger the `updateReview` method and we will need to send off our updated review to our API. This is very similar to creating a new review except for the endpoint URL and the request method.

```
updateReview() {
  fetch(`${this.apiURL}/${this.reviewID}`,{
    method: 'PUT',
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(this.review)
  })
  .then((response) => {
    console.log(response);
    if( response.ok ) {
      this.$emit('showReviews');
    }
  })
  .catch((err) => console.error(err));
}
```

### Delete Review

When you are looking at a list of reviews there is a delete button under the reviews avatar. When the user clicks that the method `deleteReview` is called with the id of the review you wish to delete.

```
deleteReview(id) {
  fetch(`${this.apiURL}/${id}`,{
    method: 'DELETE',
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(this.review)
  })
    .then((response) => {
      if(response.ok) {
        // the review has been removed from the server but we need to
remove it from the local array
        // if you were to refresh the page it would be gone but that isn't
very SPA like
```

```
          const index = this.reviews.map(review => review.id).indexOf(id);
          this.reviews.splice(index,1);
        }
      })
      .catch((err) => console.error(err));
  },
```

I have some comments in the code that explain why we need to remove the review from the client & server.