Noroff
School of technology and digital media

Course Assignment - CMS - Johann Ranudd
Report

I have understood this task to be:

- Create a headless CMS with WordPress on my personal domain.
- Install the WooCommerce plugin.
- Add at least five products.
- Create a REST API with a key.
- Create a repository on github.
- Fetch data from the API.
- Create a list of products from this API.
- Create a details or product specific page using query string parameters.
- Deploy it to Netlify (I've used Vercel because of warnings).
- Submit a brief report, outlining the process and most important decisions you made over the course of the CA

The content of this report is in chronological order.

Headless CMS

Theme:
First step was to create a headless CMS. I installed WordPress (WP) on my personal domain: johann.one.
Then I created a repository that i called headless_them_johann
Link to that repo: https://github.com/johannranudd/headless_them_johann
Spelling mistake… I know.
Set the WP theme to be headless_them_johann.

WooCommerce:
Installed the WooCommerce plugin.
Added nine products with information such as:

- Name
- Price
- Image
- Shipping details (length, width, height, volume, weight)
- Categories

REST API:
Got the rest api from the WooCommerce plugin with keys.
API with keys:
https://www.johann.one/wp-json/wc/v3/products?consumer_key=ck_665f152a7ef7923e561fd71862902f11f72672c9&consumer_secret=cs_bce68a8f771bf9355c3c48d304d3e50e530e2ae0

Design

Home page:
The design was supposed to be as simple and intuitive as possible.
I've taken inspiration from my own previous projects.
Links to projects:
https://react-cart-johann-ranudd.netlify.app/
https://react-menu-johann-ranudd.netlify.app/

- Header: a logo placeholder and a cart icon displaying the amount of items in the cart.
- Footer: a simple disclaimer.
- Home page content: headline, buttons that allow filtering by category.
  List items: each product displayed in a list showing the image, name, price and an add-to-cart button.

Each product image is a link to the detail.html page that shows the specific product information.

Details page:
The details page will display the specific product selected and all its information
It will also allow the user to add a product to the cart

Cart page:
The Cart page displays each product added to the cart.
Each product has a panel that shows the name of the product, the number of this specific item added in the cart, the price of one item and the price of all items of this kind.
There are also three buttons: up and down arrows that allow the user to increase or decrease the amount of selected item(s) in the cart. And a “remove item” button that allows the user to completely remove all item(s) of this kind.

Below all products there is a section that displays the total items in the cart (how many items the user will buy).
And the total price of all items (what the user will pay).

Below this there is a form with a label, input and submission button (CTA button).
The label tells the user to input a fake credit card number. This is to make it obvious that this isn't a real eCommerce store and it's for demonstration purposes only.
The input has a placeholder that displays an example of input.
If the value of this input isn't a four digit number, the user will receive a warning that displays:
Credit card number must be 4 digits

If the form meets validation criteria, the user is then taken to the success page.

Success page:
The success page simply displays a funny image, a headline that says "SUCCESS, thank you for your purchase" and a button to continue shopping.
If the button is clicked the user is taken back to the homepage.

Loader:
The website contains a loader that spins when data is being fetched from an api. This loader has been taked from https://loading.io/css/ and is the only thing on this website that has been copied.
I was told that it was okay to do this in a previous lecture. However, if not, I could easily make one.

Code

Time for coding.

I created a repository on github, then made a continuous deployment on Netlify.
I encountered some problems with Netlify. When visiting the site, google warned that it was a deceptive site so I quickly installed an SSL certificate on my WP host, deleted the netlify domain and changed to Vercel.

Globals:

I created a global.css file to cover styling for the nav-bar, footer, buttons and other reusable things.
Also this is where fonts and color variables are imported.

There is also a utils.js file that is being used for fetching data. This way you can fetch data just by importing and calling the function: getData(url) with the variables needed as parameters.
So if you want to fetch all data you can pass in the url that fetches all data, but if you want to get specific data you could send in a different url with query string parameters.

index.html
homepage.js
homepage.css

Fetching data:
First data is fetched inside the displayData() function, like this:

```
displayData(getData(url), cart);
```

This setup looks strange but it's done this way because cart has to be a global variable.
Later the function will be called again with a different parameter

In the second example, the function is called again to fetch data but this time with a filtered array which is newCart. This is to display items of the selected category.

```
displayData(getData(url), newCart);
```

Displaying data:
displayData does a number of things.

- It takes the data and generates a dynamic html template string
- In this string there are buttons that allow the user to add items to the cart. This is done with the addToCart() function at the bottom.
- It changes the loading variable from true to false when data has been fetched which disables the spinner/loader
- It also sorts the items by id. This can be changed to any other value like:
  - Price - low/high or high/low.
  - Date item was added
  - Name

  etc..

```
cartPar.sort((a, b) => {
    const first = Number(a.id);
    const second = Number(b.id);
    if (first > second) {
      return 1;
    } else {
      return -1;
    }
  });
```

Adding items to cart:
The global cart and total variables looks like this:

```
let cart = sessionStorage.getItem('cart')
  ? JSON.parse(sessionStorage.getItem('cart'))
  : [];
let total = sessionStorage.getItem('total')
  ? JSON.parse(sessionStorage.getItem('total'))
  : 0;
```

If the cart and/or total is changed by adding an item to the cart, the variables will now go from being [] and 0 to being a sessionStorage (SS) object.
This is done by changing the cart and total variables by writing:

```
sessionStorage.setItem('total', JSON.stringify(total));
      sessionStorage.setItem('cart', JSON.stringify(cart));
```

From there it also checks the status of SS.
If SS is empty, then SS becomes the data, if SS has been changed, then keep it as it is when displayData is executed again.

```
if (cart.length === 0) {
    cart = data;
  }
  if (cartPar.length === 0) {
    cartPar = cart;
  }
```

The bulk of the work on this page is done in the displayData() function. I know I could and perhaps should split this function up into several smaller functions but **if it works don't touch it.**

To keep this report from being way too long I can say that pretty much the same logic is used when handling the cart on other pages.

details.html
details.js
details.css

To get to the details page, press the image which is an anchor tag like this

```
<a href="details.html?id=${id}">
```

Then get the query strings
```
const querystring = document.location.search;
const mySearchParams = new URLSearchParams(querystring);
const id = mySearchParams.get('id');
```

Create dynamic url:
```
const url =
`https://www.johann.one/wp-json/wc/v3/products/${id}?consumer_key=ck_665f1
52a7ef7923e561fd71862902f11f72672c9&consumer_secret=cs_bce68a8f771bf9355c3
c48d304d3e50e530e2ae0`;
```

Then fetch data with that url:
```
const data = await getData(url);
```

Destructure the data:
```
const { id, name, images, dimensions, weight, categories, price } = data;
```

Calculate volume:
```
const radiusNum = Number(dimensions.width / 2);
  const heightNum = Number(dimensions.height);
  const volumeCalc = Math.PI * radiusNum * radiusNum * heightNum;
  const volume = parseFloat(volumeCalc.toFixed());
  const dl = volume / 100;
```

Then set the HTML element to be a template literal string with dynamic data.
And add an eventlistener to the button, which updates SS.

cart.html
cart.js
cart.css

On the cart page there is no data fetching since the data is coming from SS.
Grab that data, do some math and return the numbers so that the user can see all the necessary information needed to make a purchase.

Form
The form has a very simple way of validating. Would do it differently if this wasn't a school project.
Insert a four digit fake credit card number. If false then the user gets a warning. If true then the user is taken to the success page where they can return to shopping. The cart is then emptied.

Summary

Links and sources:

Images:
Unsplash.com
All images are free to use except the one on the success page.

Inspiration and design ideas:
https://react-cart-johann-ranudd.netlify.app/
https://react-menu-johann-ranudd.netlify.app/

WooCommerce REST API:
https://www.johann.one/wp-json/wc/v3/products?consumer_key=ck_665f152a7ef7923e561fd71862902f11f72672c9&consumer_secret=cs_bce68a8f771bf9355c3c48d304d3e50e530e2ae0

Github repos:
https://github.com/johannranudd/headless_them_johann
https://github.com/johannranudd/my-coffee-mug-store

WP domain:
https://www.johann.one/
https://www.johann.one/admin

Link to project page:
https://my-coffee-mug-store.vercel.app/

Bonus

I created a version in Next.js.
Here data is pre fetched on the server and revalidated every 10 seconds.
Check it out.

Link to project:
https://next-pre-rendering-and-data-fetching.vercel.app/

Link to Github project:
https://github.com/johannranudd/next-pre-rendering-and-data-fetching