# React Shopping Cart - Workshop

Workshop for the ["JS Front-End" course @ SoftUni](https://softuni.bg/trainings/3976/js-front-end-february-2023) .

**Working with Remote Data**

For the solution of some of the following tasks, you will need to use an up-to-date version of the **local REST service**, provided in the lesson’s resources archive. You can [read the documentation here](https://github.com/softuni-practice-server/softuni-practice-server).

## Requirements

Write a **React application** that can **load**, **create**, **buy** and **remove** a list of **products** in a **shopping** **cart**. You will be given **HTML** & **CSS** resources to which you must bind the needed functionality and create several [**React Components**](https://reactjs.org/docs/react-component.html) that fire **AJAX calls** to the provided **server**.

First you need to create a **new** **React Application** using [Create React App](https://create-react-app.dev/)

Then you can start the front-end react application with the “**npm start**” or **“npm run start”** command

You also must also start the **server.js** file in the server folder using the “**node server.js**” command in another console **(BOTH THE CLIENT AND THE SERVER MUST RUN AT THE SAME TIME)**

## Server Endpoints

* [**http://localhost:3030/jsonstore/products/**](http://localhost:3030/jsonstore/products/)
* [**http://localhost:3030/jsonstore/products/**](http://localhost:3030/jsonstore/products/)**:id**

## React Setup

* Remove **all** **unnecessary** files like **App.css**, **test configuration files** and etc. (we won’t be using these)
* **Copy** & **Paste** the provided CSS file content inside i**ndex.css**
* Create **components** & **services** folders
* Inside the **components** folder add **sub-folders** **ShoppingCart**, **ShoppingCartForm** & **ShoppingCartItem**
* Create the 3 different component functions that we will render inside their respective sub-folders – **ShoppingCart.jsx**, **ShoppingCartForm.jsx** & **ShoppingCartItem.jsx**

## Shopping Cart Component

This component will be a **container** for all of the **individual products & the form**.

* Define the **state** of all products using [useState](https://reactjs.org/docs/hooks-state.html)  
  
* Create a **Total Price** variable that filters only products that are **bought** and get’s their **sum**

Text

Description automatically generated

* The Shopping Cart component should render the **HTML** for everything inside the section with class name **shopping-cart\_\_container.** Render also the **ShoppingCartForm** & **ShoppingCartItem** components that we will do next. And don’t forget the **Total Price**!
* The **ShoppingCart** component should receive every individual product as an **item prop**. We should also [render a list](https://reactjs.org/docs/lists-and-keys.html) of all products.

Text

Description automatically generated

* Render the **Shopping Cart** component inside **App.js** as a starting point to our page.

## Shopping Cart Item Component

This component should render each individual product

* It should receive an item as a [**prop**](https://reactjs.org/docs/components-and-props.html)
* It should have **handleBuyItem** & **handleRemoveItem** functions that are fired whenever the Buy button or the Remove buttons are [clicked](https://reactjs.org/docs/handling-events.html). We will send HTTP requests using those function handlers later on
* The **article** with **shopping-cart\_\_item-container** class should receive an [inline-style object](https://reactjs.org/docs/dom-elements.html#style) that changes the **text decoration** to “line-through” if the item **is bought**, otherwise set it to “none”
* Render the “**Buy**” button [conditionally](https://reactjs.org/docs/conditional-rendering.html) – only when the **isBought** property is **false**.

## Shopping Cart Form Component

This component should render the add product form

* It should **NOT** receive any props
* It should have **3 variables** defined with **useState** – **itemName**, **itemCost**, **itemImgUrl**
* It should have a **handleSubmit** function handler that fires when the “**Add**” button is **clicked**. You should add the [preventDefault](https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault) method call inside it.
* The **input fields** should change their **respective state variable.** Use the [onChange](https://upmostly.com/tutorials/react-onchange-events-with-examples) synthetic event from React.
* The “**Add**” button should be **disabled** if one of the state variables is an **empty string**!

## SVG Icons

Create an **assets folder** and inside create **3 individual svg files** for the icons

To import an icon and render it as a **React component** use the following syntax:



Graphical user interface, text

Description automatically generated

To the same for the **other 2 icons**

## Service File

Create a **products-service.js** file inside **services** **folder** that will use the [Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) and [**exports 4 functions**](https://beta.reactjs.org/learn/importing-and-exporting-components) – one for each **individual HTTP method** that we will call to the server

* (GET) All Products – **async** **function** that returns the **response json**

Graphical user interface, text

Description automatically generated

* (POST) Add Product – **async function** that receives **name, cost, imgUrl** and creates a new product with **isBought** set to **false** by default

Text

Description automatically generated

* (PATCH) Buy Product – **async function** that receives **productId** and sets the **isBought** property to **true** for that individual product

Text

Description automatically generated

* (DELETE) Remove Product – **async function** that receives **productId** and **removes** that individual product from the server  
  Text

  Description automatically generated

## List Products API Call

Send a **GET** request to the server and **list** all of the products inside **ShoppingCart.jsx** with the help of the [useEffect](https://reactjs.org/docs/hooks-effect.html) React Hook!

* Inside the hook call the **getAllProducts** function from the **products-service.js** file (don’t forget to **import** it first)
* Add callbacks for **success** & **error**
* The **success** callback should **set the state** with the new **products**
* The **error** callback should just **log** the error on the **console** (for now)
* Add an **empty array** as a **second parameter** to the **useEffect** hook. That is the hook’s dependency array. An empty array means that it will **only fire once**!

## Add Product API Call

Send a **POST** **request** to the server and add the new product inside the **ShoppingCartForm.jsx**

* Inside the **handleSubmit** function handler call the **addProductToCart** **async** **function** from the service file
* Pass in the **state** **variables** as **parameters** to the service function
* Add callbacks for **success** & **error**
* The **success** callback should **clear the input fields!**
* The **error** callback should just **log** the error on the **console** (for now)

## Fetch All Products Again

As you can see the **newly created product** is not displayed because the **useEffect** has to be **executed** **again** & that will send a **new HTTP request** to get all of the products. Find a way for the **child component** to interact with it’s **parent** so it can tell it to **execute** the **useEffect** again.

***Hint: Think about the dependency array of the useEffect***

## Buy Product API Call

Send a **PATCH** request to the server inside the **ShoppingCartItem.jsx** file

* Inside **handleBuyItem** function call the **buyProduct** async function from the service file and pass in the **product id**
* Add callbacks for **success** & **error**
* The **success** callback should somehow **execute the useEffect on the parent component**
* The **error** callback should just **log** the error on the **console** (for now)

## Remove Product API Call

Send a **DELETE** request to the server inside the **ShoppingCartItem.jsx** file

* Inside **handleRemoveItem** function call the **removeProduct** async function from the service file and pass in the **product id**
* Add callbacks for **success** & **error**
* The **success** callback should somehow **execute the useEffect on the parent component**
* The **error** callback should just **log** the error on the **console** (for now)

## (Bonus Task) Notifications

Add **notification pop ups** when a **product** has been **created**, **bought** or **removed**. Also add notification in all of your **error callbacks**

Install & Use the [react-toastify](https://www.npmjs.com/package/react-toastify) npm package. Read through the documentation and figure out how to use it!