DTEK Online Store – Design

https://rob-das-react.azurewebsites.net/

2023-Jul-23

Rob Das

DTEK Consulting Services Ltd.

This is the design document for the DTEK Online Store web application. The target audience is: developers.

This document focusses on how the application was designed. There is a separate document for maintenance and support called *DTEK Online Store – Maintenance*. For information on how the application is used, there is a separate document called *DTEK Online Store – User Manual*.

Specification of Features

This section describes the high level, application requirements.

What does the application do?

The application presents the products that can be purchased; and provides a way to place an order and pay for it. The goal is to make it as easy as possible for you to browse and buy products.

How will the user use it?

The user will browse products; and select products by adding them to a shopping cart. When they have finished browsing, they can purchase the products in their shopping cart.

What are the features that are important to the users?

- browsing a list of products
- adding products to their shopping cart
- seeing a list of items that have been added to the shopping cart so far
- seeing the total cost of all items added to the shopping cart so far
- removing items from their shopping cart at any time before the final purchase
- making the final purchase of all items in the cart

The Final Purchase

The most significant event in the application is the purchase. This involves the payment of the grand total, including sales tax, for everything in the shopping cart, which must be paid in full. All or nothing. This event finalizes the purchase of the current contents of the shopping cart, at the time that the event was initiated. After the purchase is initiated by the user, the user can make no further changes to the shopping cart.

It may be useful to define a business object called an *order*. An order could provide a way to further track the list of items in the final purchase until they have been delivered. However, the order object may be redundant since the receipt or invoice should contain the list of items in the final purchase.

An important distinction is made between information tracked on products available and on products purchased. Information about available products changes over time, but information about the products in a final purchase is recorded and cannot be changed later without invalidating the purchase. The products in a final purchase are listed on the receipt.

Data Model

This section begins by identifying all information that is potentially useful and may be of interest. It uses this to form a comprehensive business data model, and a suggested technical representation. This may or may not be the final data model for the application, but it provides a starting-off point.

All business Information that may be of Interest:

- list of products
- product
- product details
- shopping cart
- shopping cart summary (number of products, total cost)
- products that are in the shopping cart
- product name
- quantity (number of items of a particular product)
- item cost / unit price
- total cost of all items of a particular product in the shopping cart
- total cost of all products in the shopping cart
- tax
- grand total
- final purchase
- shipping information (customer name, address)
- payment information (payment method, credit card info)
- receipt

Business Data Objects

The information of interest can be conveniently grouped into data objects in many ways. Here are some possible ways that can help provide features that are important to the users.

Available Products

List of all products that can be purchased from the store.

- product name
- product details
- unit price
- product image

Chosen Products

List of products added to the shopping cart.

- product name

- product details
- unit price
- quantity

Shopping Cart

- chosen products
- number of different products
- total cost of products
- total tax
- grand total

Final Purchase

- list of products to be shipped
- quantity of each product to be shipped
- cost of each product
- total cost of products
- total tax
- grand total paid by customer
- receipt

Shipping Information

- customer name
- address

Payment Information

- payment method
- card number
- name on card
- card expiration date
- cvv/cvc security code

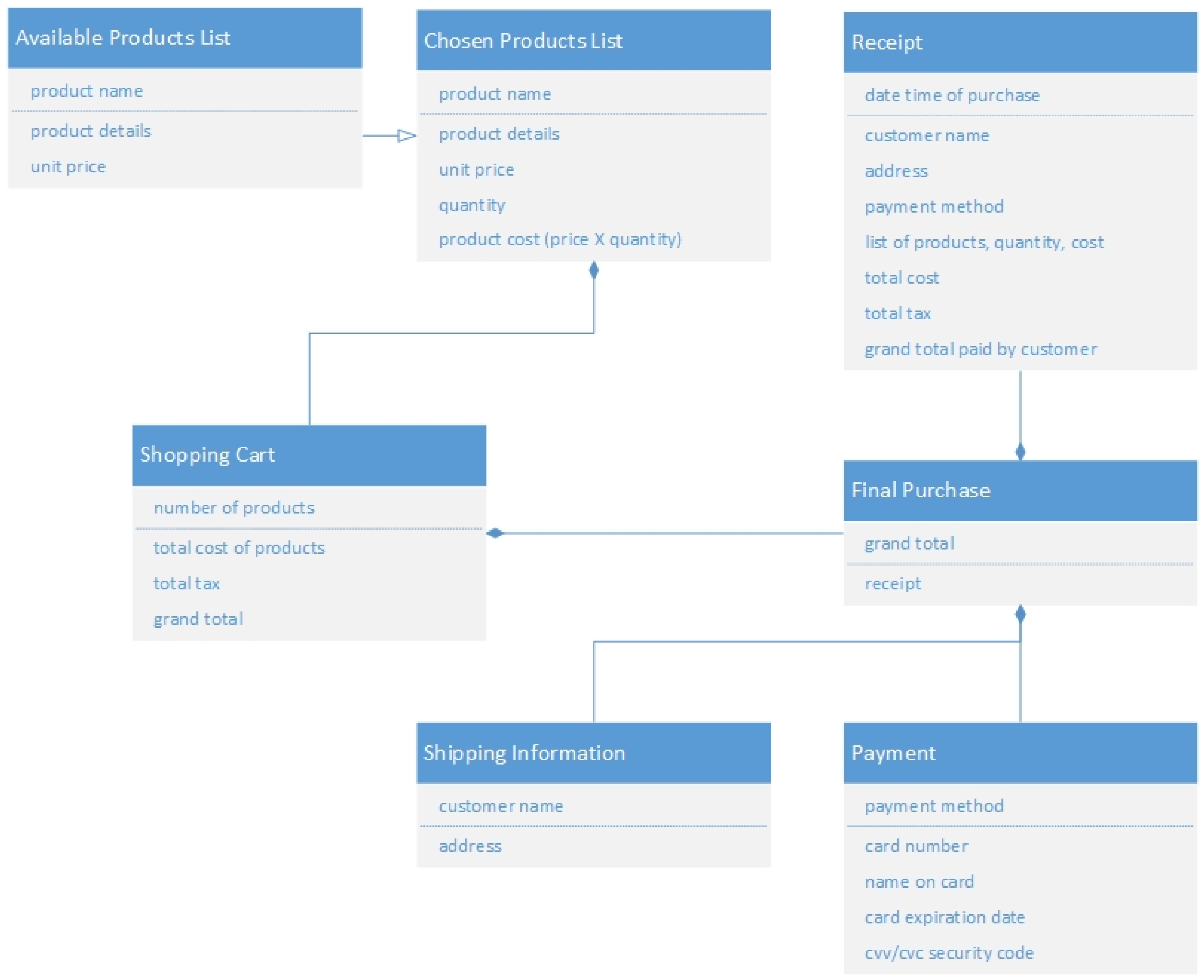
Receipt

Printed document showing the following information:

- date and time of purchase
- customer name
- address
- payment method
- list of products to be shipped

- quantity of each product to be shipped
- cost of each product
- total cost of products
- total tax
- grand total paid by customer

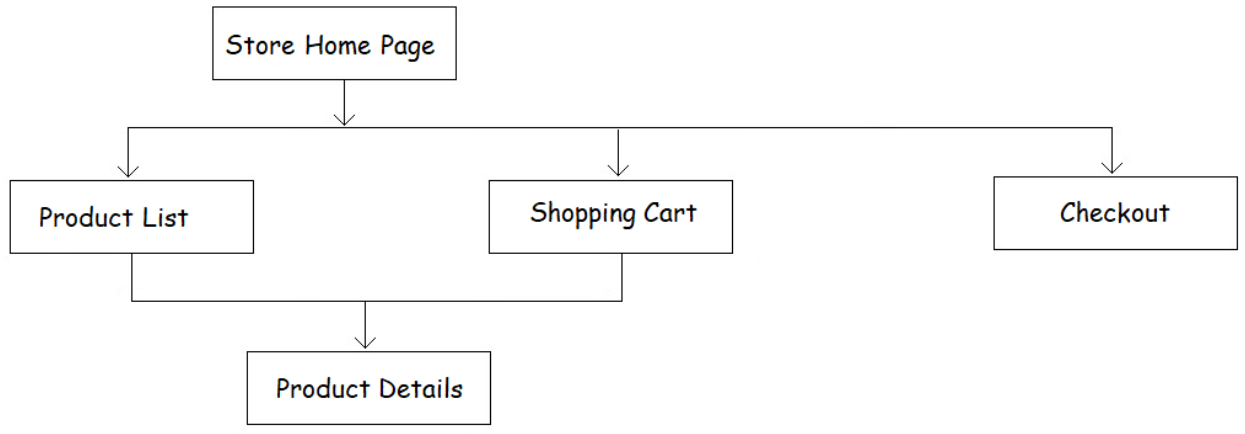
Business Object Model



JSON Object Schema

```
{
  "availableProductList": [
    {
      "id": "string",
      "productName": "string",
      "productDetails": "string",
      "productImagePath": "string",
      "unitPrice": "number"
    },
    ...
  ],

  "shoppingCart": {

    "chosenProductList": [
      {
        "id": "string",
        "productName": "string",
        "productDetails": "string",
        "unitPrice": "number",
        "quantity": "number",
        "productTotal": "number"
      },
      ...
    ],

    "totalCostOfProducts": "number",
    "tax": "number",
    "grandTotal": "number"
  },

  "finalPurchase": {

    "shippingInformation": {
      "customerName": "string",
      "address": "string"
    },

    "payment": {
      "paymentMethod": "string",
      "cardNumber": "string",
      "nameOnCard": "string",
      "cardExpirationDate": "string",
      "securityCode": "string"
    },

    "receipt": {
      "receiptNumber": "string",
      "purchaseDateTime": "string",
      "customerName": "string",
      "address": "string",
      "paymentMethod": "string",

      "productList": [
        {
          "productName": "string",
          "quantity": "number",
          "productTotal": "number"
        },
        ...
      ],

      "totalProductCost": "number",
      "tax": "number",
      "grandTotal": "number"
    }
  }
}
```

Site Map



Pages

Store Home Page

- landing page for the store application
- servers as the application root
- shows website common header including:
    - Logo (however users may be put off, if the first thing they see is a corporate logo – it may be enough to put a small logo in the footer)
    - shopping cart summary (number of items, and total cost)
    - the navigation menu
- contains the SPA routing component that shows the current page
- shows website common footer including:
    - small logo
    - web app name, version number, one line description
    - copyright

Product List

- access to all products being sold
- ability to select and add product to shopping cart

Product Details

- unit price
- provides detailed information on a specific product
- ability to add product to shopping cart

Shopping Cart

- shows list of products that the user has chosen to buy
- for each product in shopping cart
    - product name
    - +/- buttons to increase or decrease product quantity
    - quantity (product count)
    - cost (= product unit price * quantity)
- totals
    - total product cost
    - total tax
    - total shipping
    - grand total
- checkout button

Checkout

- provides way to make final payment on total contents of shopping cart
- allows user to go back and make further changes before final payment
- shipping information
- upon final payment, generates receipt and clears shopping cart

Navigation

The *Store Home Page* contains all other pages, and the application navigation menu, which allows user to navigate between

- Product List
- Shopping Cart
- Checkout

The *Product Details* page is only accessible by clicking on a product in the *Product List* page, or by clicking on a product in the *Shopping Cart*.

From the *Product Details* page, the user can still use the application navigation menu to navigate to

- Product List
- Shopping Cart
- Checkout

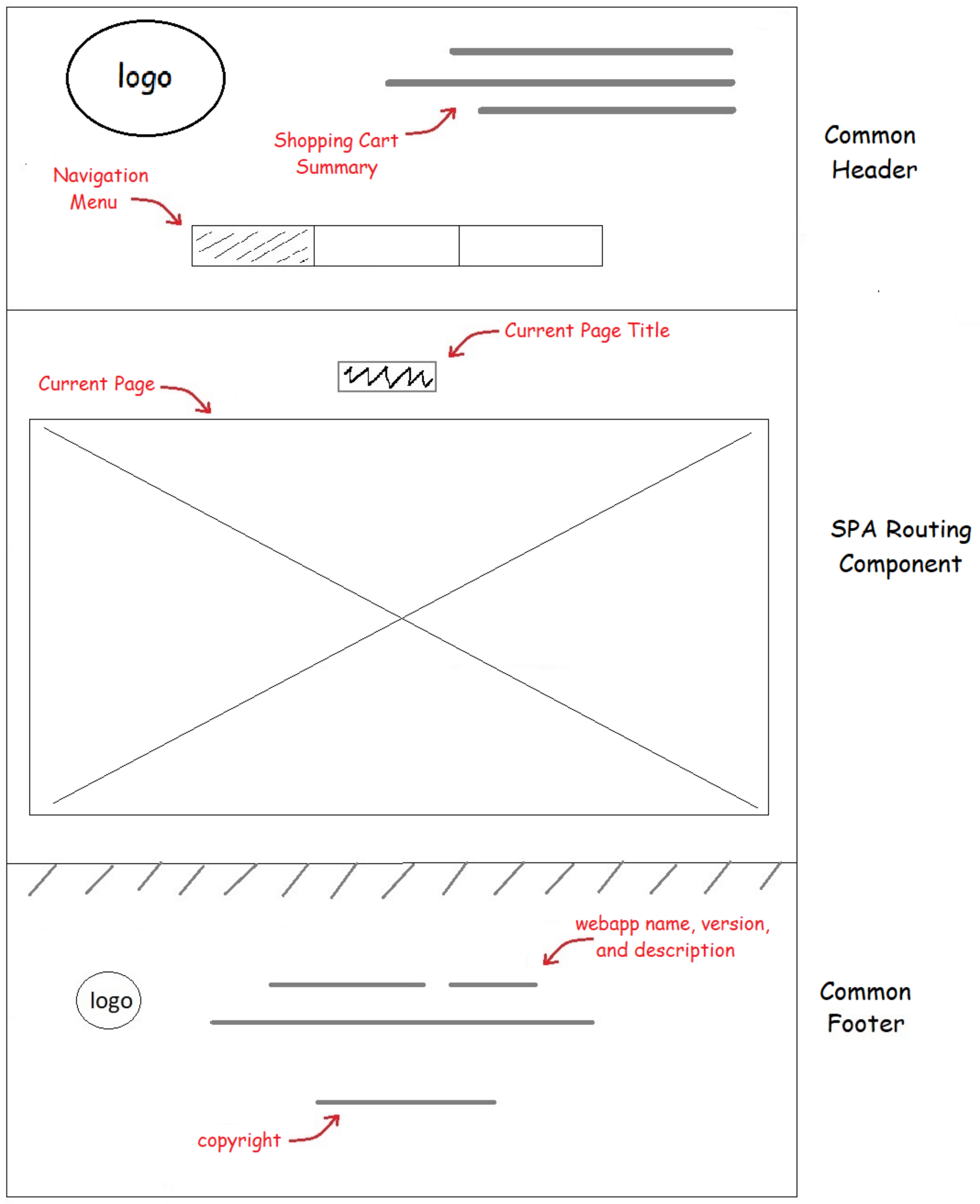
The *Shopping Cart* page has a checkout button that navigates to the checkout page. This is the same as clicking the checkout page menu item.

From the *Checkout* page, the user can still use the navigation menu to go back and make further changes before final payment. After final payment, all data is cleared from the shopping cart and the website resets to the starting state.

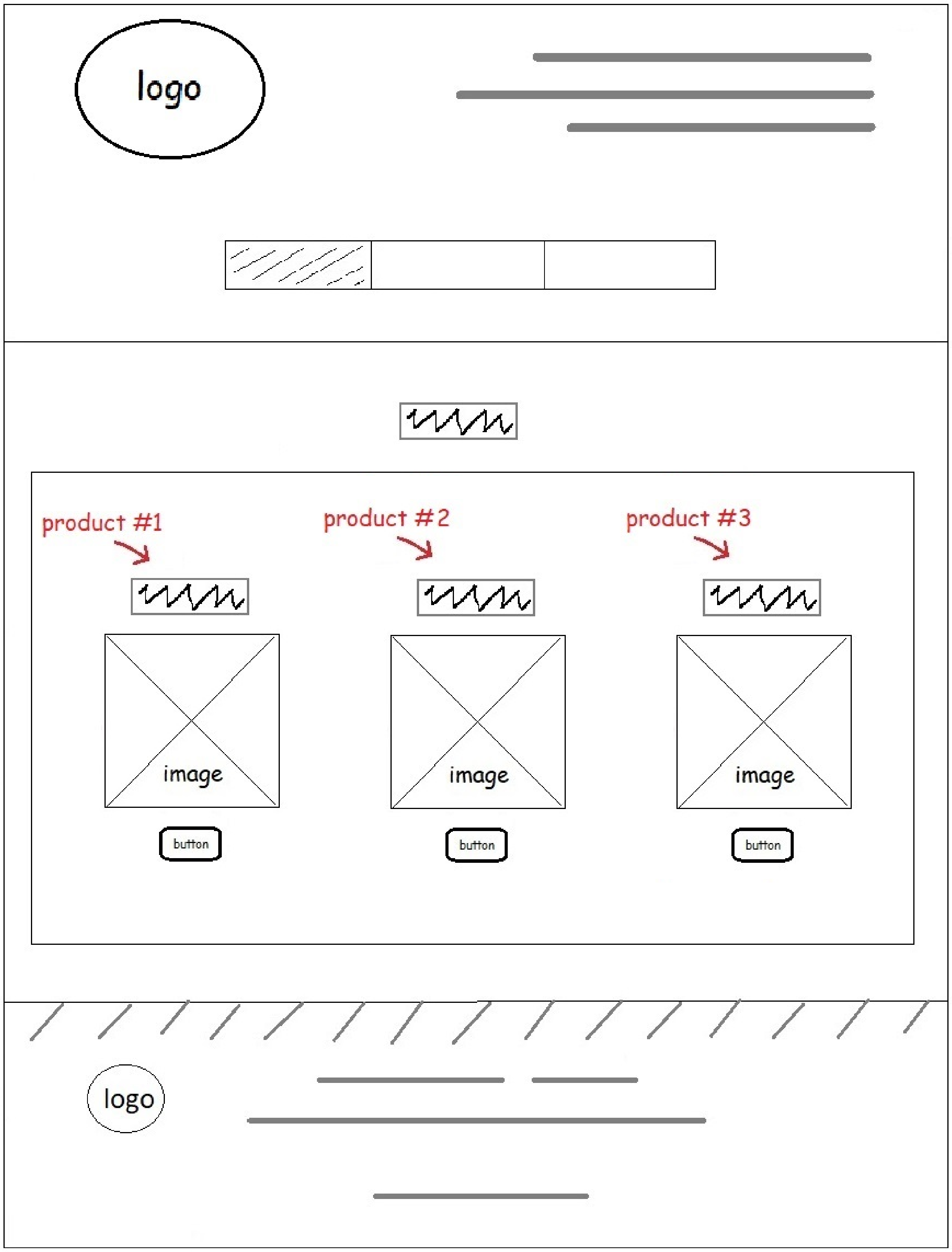
Wireframe Layouts

This section uses wireframes to design the main features of the website. It gives an idea of the website's functionality, before considering visual design details, like styling and color schemes.

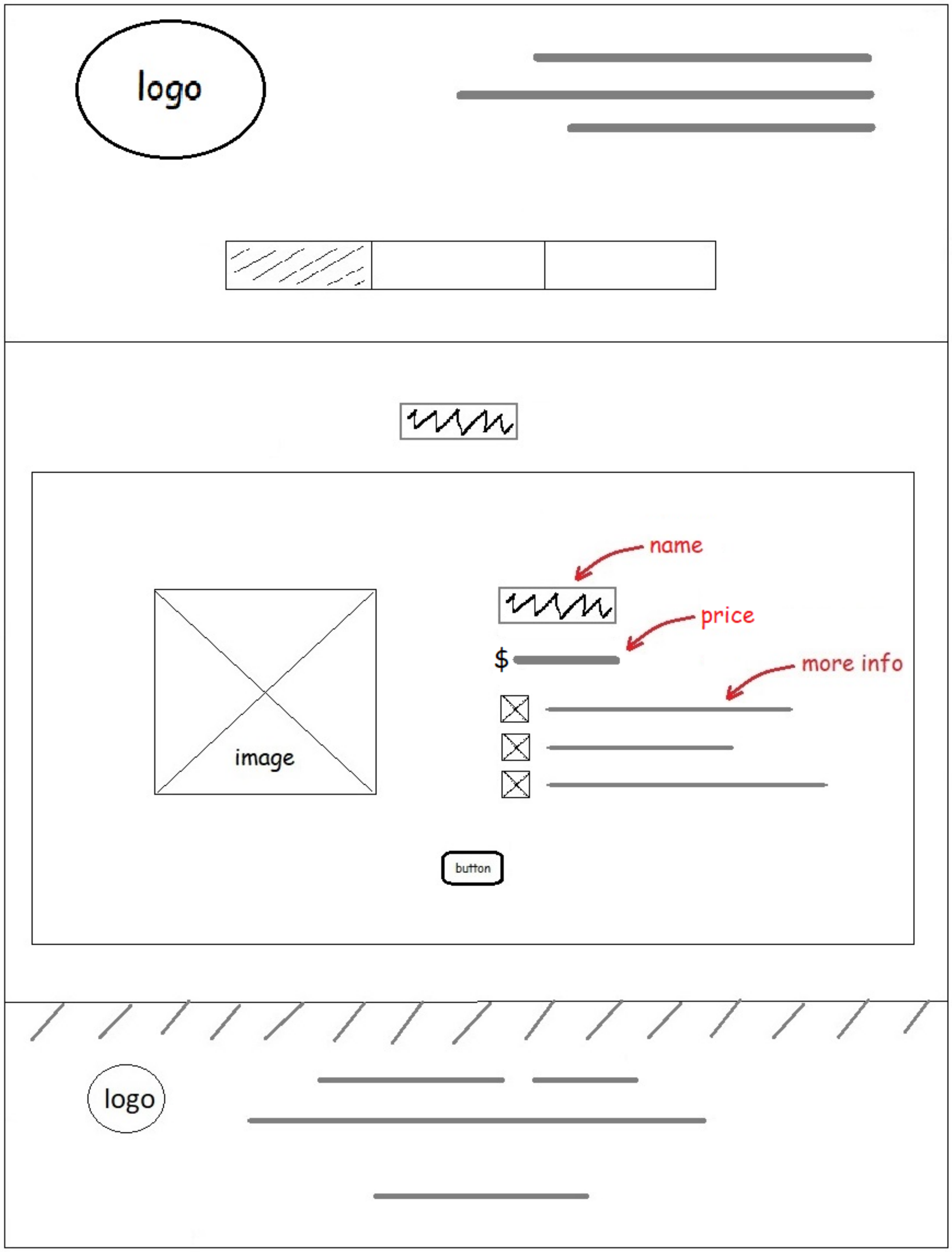
Store Home Page



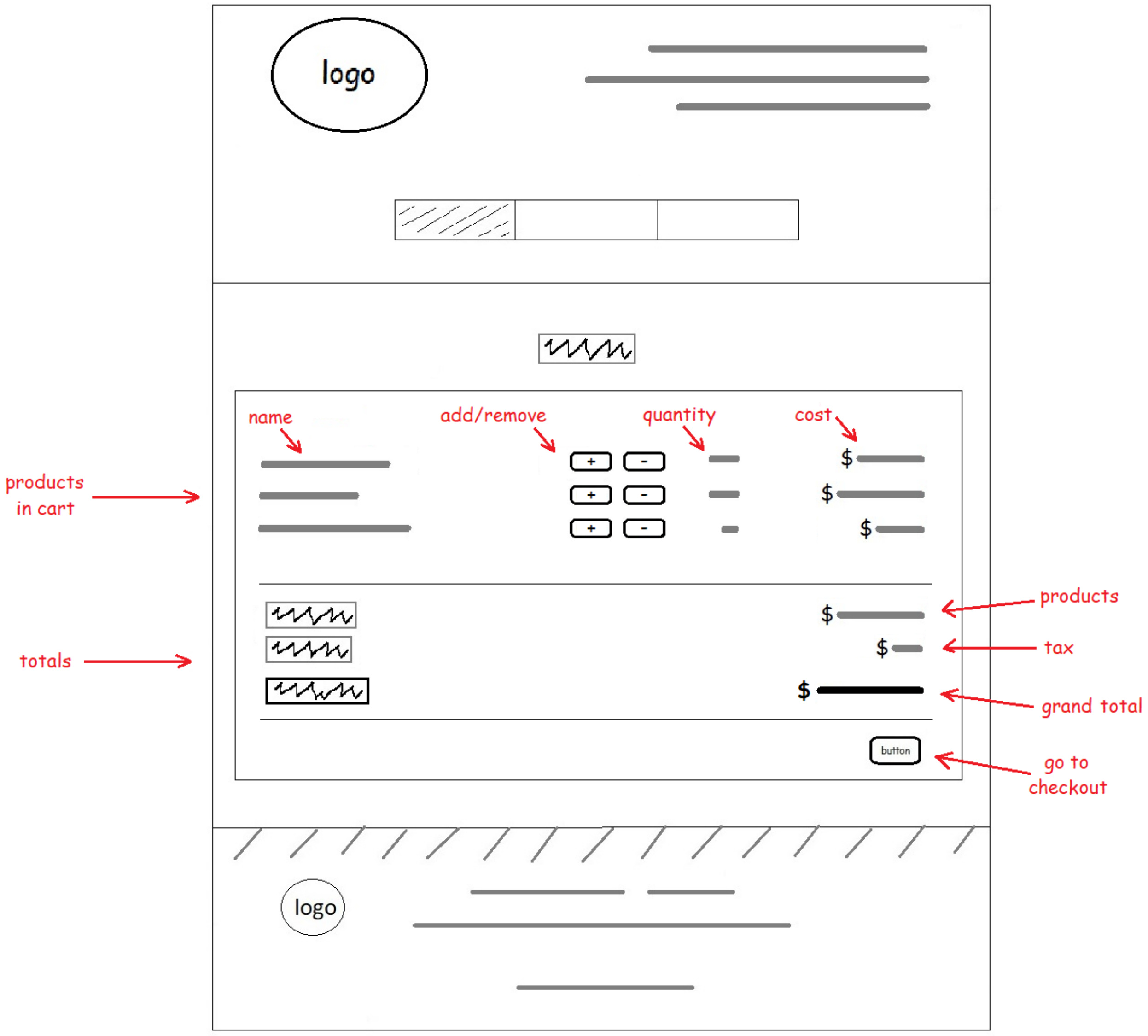
Product List



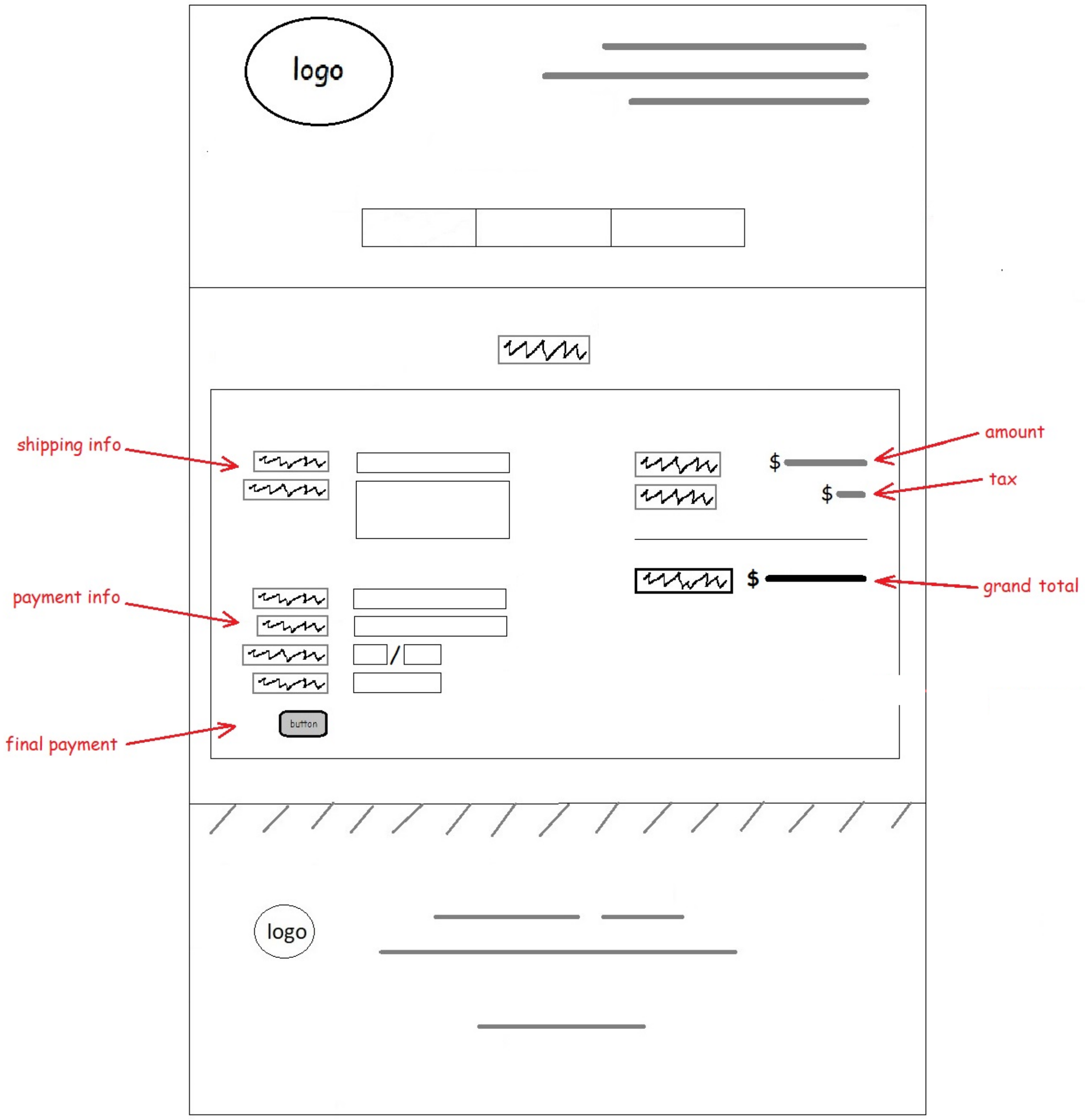
Product Details



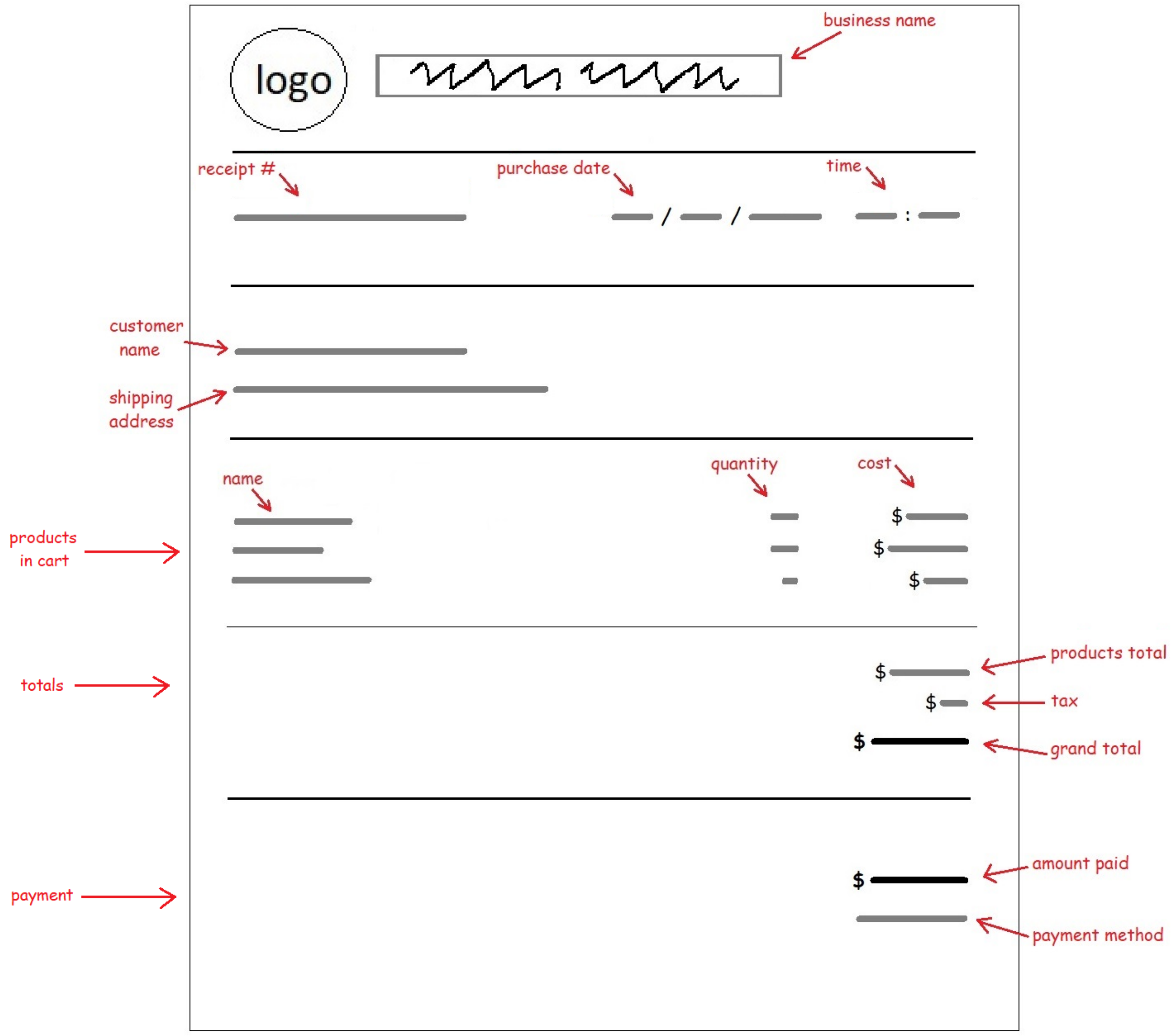
Shopping Cart



Final Checkout



Receipt


business name
logo
receipt #
purchase date
time
customer name
shipping address
name
quantity
cost
products in cart
totals
products total
tax
grand total
payment
amount paid
payment method

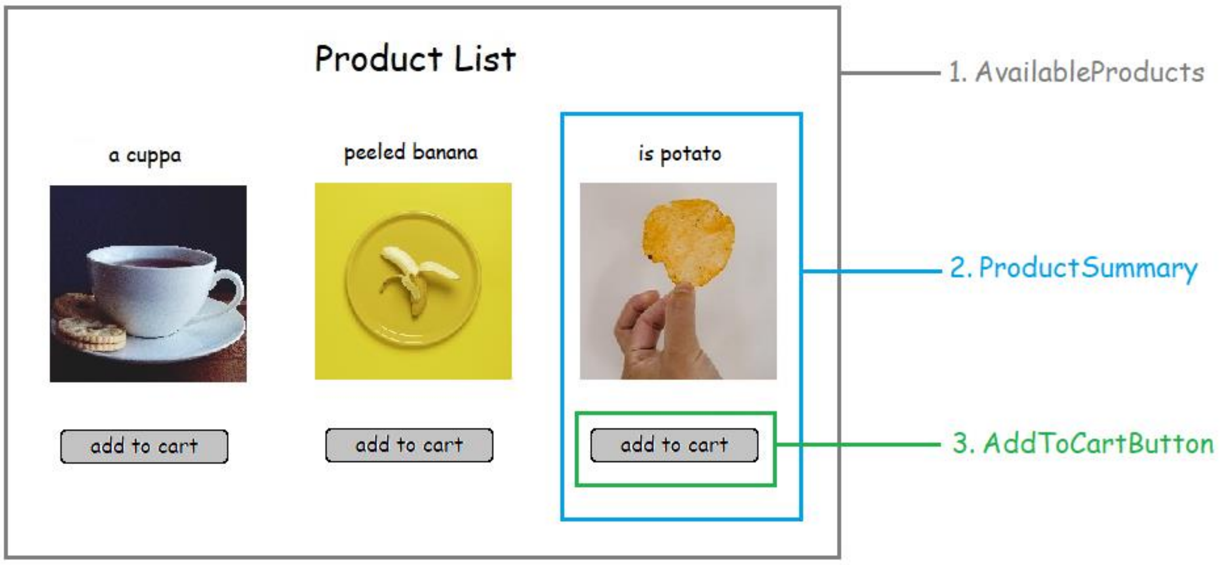
Implementation – React

The Product List page will be our starting point. Following the steps recommended by React (see the references at the end of this document), we will document the steps to implement just the Product List page here. Then we will use this as a guide to implement the rest of the application's components.

Step 1: Break the UI into a component hierarchy

Product List

The Product List page can be broken down into the following 3 components.



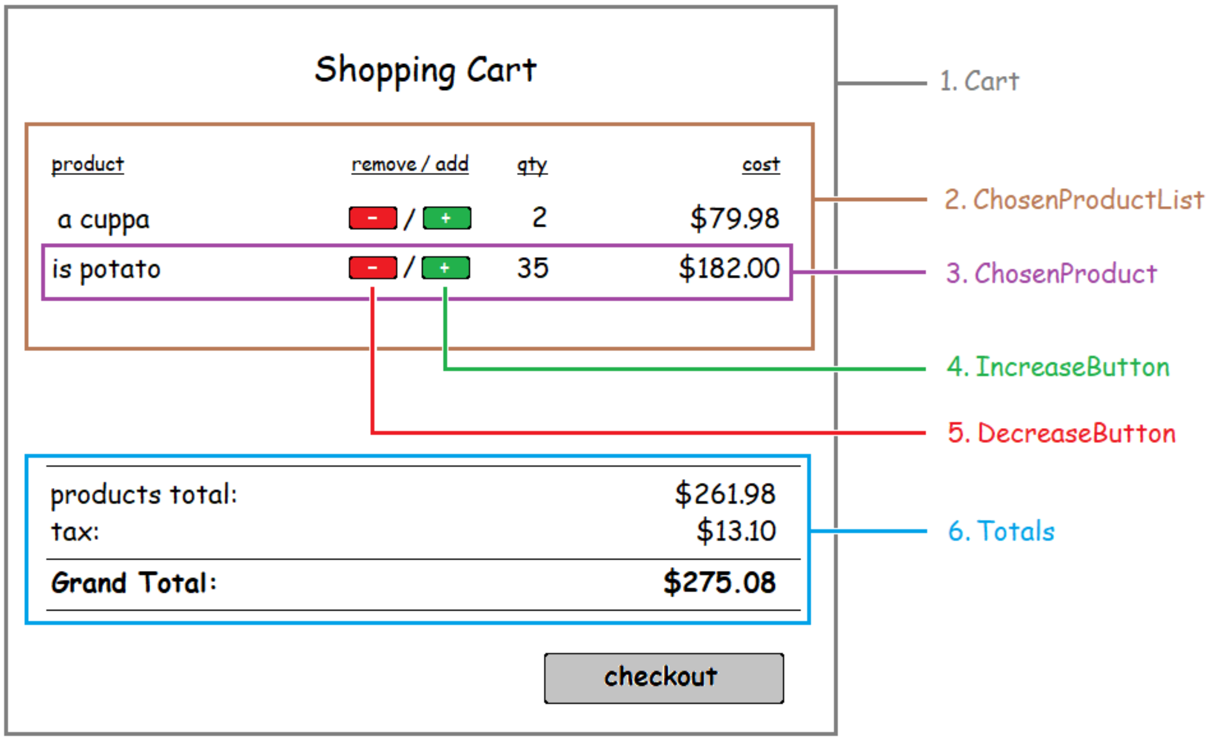
1. AvailableProducts shows the list of available products, from which to browse and buy.
2. ProductSummary shows a summary of an individual product.
3. AddToCartButton is a button to add an individual product to the shopping cart.

The components form the following hierarchy.

- AvailableProducts
  - ProductSummary
    - AddToCartButton

Cart

The Cart page can be broken down into the following 6 components.



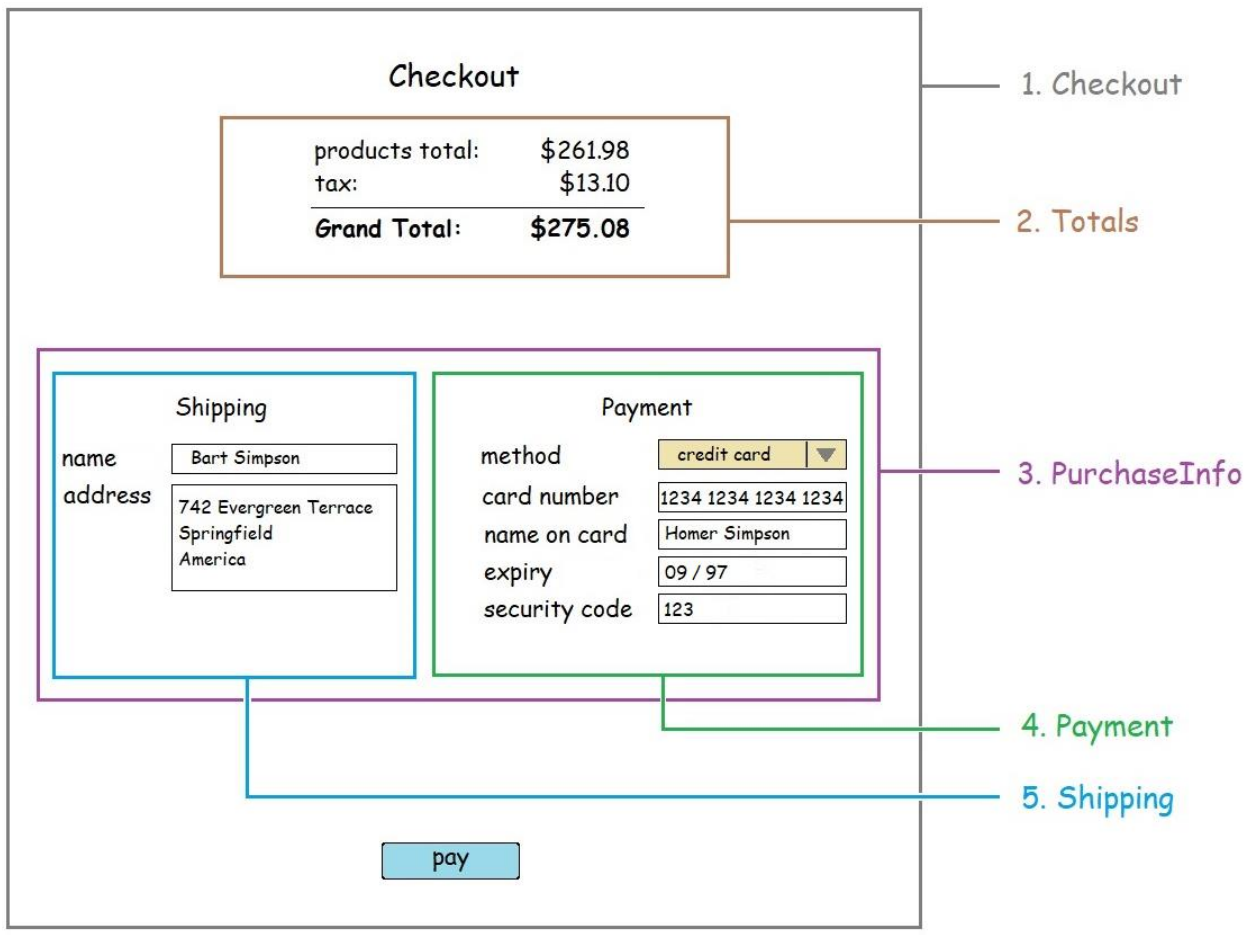
1. Cart shows the contents of the shopping cart so far, and a link to the checkout page
2. ChosenProductList shows a list of chosen products
3. ChosenProduct shows an individual chosen product
4. IncreaseButton is a button to increase the quantity of a chosen product
5. DecreaseButton is a button to decrease the quantity of a chosen product
6. Totals shows the total for all products, tax, and a grand total

The components form the following hierarchy.

- Cart
  - ChosenProductList
    - ChosenProduct
      - IncreaseButton
      - DecreaseButton
  - Totals

Checkout

The Checkout page can be broken down into the following 5 components.



1. Checkout is for making the final purchase, which can be initiated using the *pay* button.
2. Totals shows the total for all products, tax, and a grand total.
3. PurchaseInfo is where the user provides the information needed to make the final purchase.
4. Payment is for the user to provide payment information.
5. Shipping is for the user to provide shipping information.

The components form the following hierarchy.

- Checkout
  - Totals
  - PurchaseInfo
    - Payment
    - Shipping

The Checkout page is intended for a one-time transaction based on the current content of the shopping cart.

Step 2: Build static versions in React

Continuing with the design methodology recommended at https://react.dev/learn/thinking-in-react, in this step we build a version that renders the UI from the data model without adding any interactivity.

To keep things as basic as possible, all content pages are coded and tested as separate apps, and the code for each content page app is in its own App.js and index.css.

Product List

See Appendix 1 at the end of this document, for the static version source code of Product List.

Shopping Cart

See Appendix 2 at the end of this document, for the static version source code of Shopping Cart.

Checkout

See Appendix 3 at the end of this document, for the static version source code of Checkout.

Step 3: Find the minimal but complete representation of UI state

From step 2, here is the data used by the 3 main content pages (the step 2 source code can be found in the appendices, at the end of this document).

Product List Page Data

The product list page has an array of products that the user can buy.

```
const AVAILABLE_PRODUCT_LIST = [
  {
    "id": 1,
    "productName": "a cuppa",
    "productDetails": "A cup of tea and a bickie.",
    "productImageFile": "./assets/fancyacuppa_sm.jpg",
    "unitPrice": 39.99,
  },
  ...
];
```

Shopping Cart Page Data

The shopping cart page has an array of products that the user has chosen.

```
const CHOSEN_PRODUCTS = [
  {
    "id": "1",
    "productName": "a cuppa",
    "productDetails": "A cup of tea and a bickie.",
    "unitPrice": 39.99,
    "quantity": 2,
  },
  ...
];
```

Checkout Page Data

The checkout page has an array of products the user has chosen to buy and the information needed to complete the final purchase consisting of shipping information and payment information.

```
const CHOSEN_PRODUCTS = [
  {
    "id": "1",
    "productName": "a cuppa",
    "productDetails": "A cup of tea and a bickie.",
    "unitPrice": 39.99,
    "quantity": 2,
  },
  ...
];
```

```
const FINAL_PURCHASE = {
  "shippingInformation": {
    "customerName": "Bart Simpson",
    "address": "742 Evergreen Terr.\nSpringfield\nAmerica\n"
  },

  "payment": {
    "paymentMethod": "credit card",
    "cardNumber": "0000 0000 0000 0000",
    "nameOnCard": "Homer Simpson",
    "cardExpirationDate": "09/97",
    "securityCode": "000"
  },
};
```

To determine which pieces of data are state,

https://react.dev/learn/thinking-in-react#step-3-find-the-minimal-but-complete-representation-of-ui-state

The pieces of data to consider are:

1. available products
2. chosen products
3. final purchase

Which of these are state? Identify the ones that are not:

- Does it remain unchanged over time? If so, it isn't state.
- Is it passed in from a parent via props? If so, it isn't state.
- Can you compute it based on existing state or props in your component? If so, it definitely isn't state!

What's left is probably state.

Let's go through the pieces of data one by one.

1. available products are obtained from an external data source and remains unchanged over time, so it **isn't state**.
2. chosen products have a quantity property which is under the control of the user, so it **is state**.
3. all properties of the final purchase are under the control of the user, so it **is state**.

State

Chosen products and final purchase are state.

Step 4: Identify where your state should live

Chosen products are referenced by all three pages, so it should live in a common parent, the store home page.

Final purchase is only referenced on the checkout page so perhaps it should live there. However, since the user can navigate back and forth to the checkout page, the final purchase data should persist. If the checkout page resets its state every time, the final purchase should also live in the store home page.

Step 5: Add inverse data flow

Assuming the checkout page resets its state every time we navigate there, the state will live in the store home page.

Available products are read from an external source, which at present is just a file containing *availableProducts* Json object, exported as a JavaScript object called data. Here is a sample:

```
const data = {

    availableProducts: [
        {
            "id": "1",
            "productName": "cuppa",
            "productDetails": "A cup of tea and a bickie. A wonderfully soothing afternoon break.",
            "productImageSource": "https://www.pexels.com/photo/close-up-photography-of-cup-of-coffee-near-biscuits-1143760/",
            "downloadDate": "2023-Oct-03",
            "productImageFile": "/assets/fancyacuppa_sm.jpg",
            "unitPrice": 39.99,
        },
        {
            ...
        },
    ],
};

export default data;
```

The *useState* hook is used to define the state in the store home page for the shopping cart contents, *cartContents*, the shipping information, *shippingInformation*, and the payment information, *payment*.

```
const [cartContents, setCartContents] = useState(chosenProducts);
const [shippingInformation, setShippingInformation] = useState(emptyShippingInformation);
const [payment, setPayment] = useState(emptyPayment);
```

These state items can then be passed into the components that need them, as props. Adding and removing items from the shopping cart contents, are done in the store home page by functions named *onAdd* and *onRemove*. These functions are passed as props to any component that needs to add or remove products from the *cartContents* objects. This keeps the logic for adding and removing in one place. For example, the Store component calls the shopping cart component, *Cart*, via the react router as follows:

```
<Route path="/cart" element={<Cart chosenProducts={cartContents} onAdd={onAdd} onRemove={onRemove} />} />
```

Product Details Page

The product details page will take a route parameter of the product id and render details for that specific product. Route parameters are discussed in the *Net Ninja Full React Tutorial #25* (see the References section at the end of this document).

The react router entry for the product details page is in the Store component as follows:

```
<Route
  path="/product/:id"
  element={
    <ProductDetails
      availableProducts={availableProducts}
      onAdd={onAdd}
    />
  }
/>
```

Appendix 1 – static version of source code: Product List

Here is the source code for the static version of the *Product List* React component. The static version is a simplified intermediate prototype, created during the recommended development process described at https://react.dev/learn/thinking-in-react#step-2-build-a-static-version-in-react

To keep things simple, all JavaScript code is in one file, App.js, and all CSS is in one file, index.css

App.js

```
const AVAILABLE_PRODUCT_LIST = [
  {
    "id": 1,
    "productName": "a cuppa",
    "productDetails": "A cup of tea and a bickie.",
    "productImageFile": "./assets/fancyacuppa_sm.jpg",
    "unitPrice": 39.99,
  },
  {
    "id": 2,
    "productName": "peeled banana",
    "productDetails": "A sweet yellow partially peeled fruit.",
    "productImageFile": "./assets/bananana_sm.jpg",
    "unitPrice": 25.99,
  },
  {
    "id": 3,
    "productName": "is potato",
    "productDetails": "Just eat; I promise you will like.",
    "productImageFile": "./assets/ispotato_sm.jpg",
    "unitPrice": 5.20,
  },
];

function App() {
  return (
    <div className="App">
      <AvailableProducts
        availableProductList={AVAILABLE_PRODUCT_LIST} />
    </div>
  );
}

function AvailableProducts({ availableProductList }) {
  return (
    <>
      <h1>Product List</h1>
      <div className="flex-container">

        {availableProductList.map((item) => (
          <div key={item.id}>
            <ProductSummary product={item} />
          </div>
        ))}

      </div>
    </>
  );
}

function ProductSummary({ product }) {
  return (
    <>
      <h3>{product.productName}</h3>
      <div>
        <img
          src={product.productImageFile}
          alt={product.productName}
          title={product.productDetails} />
      </div>
      <AddToCartButton product={product}></AddToCartButton>
    </>
  );
}

function AddToCartButton({ product }) {

  const onAdd = (product) => {
    console.log(`add to cart was clicked for product ${product.productName}`);
  };

  return (
    <button onClick={() => onAdd(product)}>add to cart</button>
  );
}

export default App;
```

index.css

```
body {
  margin: 0;
  font-family: comic-sans-ms, badaboom, cursive, Arial, Helvetica, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

.flex-container {
  display: flex;
  justify-content: space-around;
}

.App {
  text-align: center;
}

button {
  font-family: comic-sans-ms, badaboom, cursive, Arial, Helvetica, sans-serif;
  font-weight: bold;
}

img {
  margin: 1.5em;
}
```

Appendix 2 – static version of source code: Shopping Cart

Here is the source code for the static version of the *Shopping Cart* React component. The static version is a simplified intermediate prototype, created during the recommended development process described at https://react.dev/learn/thinking-in-react#step-2-build-a-static-version-in-react

To keep things simple, all JavaScript code is in one file, App.js, and all CSS is in one file, index.css

App.js

```
const CHOSEN_PRODUCTS = [
  {
    "id": "1",
    "productName": "a cuppa",
    "productDetails": "A cup of tea and a bickie.",
    "unitPrice": 39.99,
    "quantity": 2,
  },
  {
    "id": "2",
    "productName": "is potato",
    "productDetails": "Just eat; I promise you will like.",
    "unitPrice": 5.20,
    "quantity": 35,
  },
];

function App() {
  return (
    <div className="App">
      <Cart
        chosenProducts={CHOSEN_PRODUCTS} />
    </div>
  );
}

function Cart({ chosenProducts }) {

  const gotoCheckout = (products) => {
    console.log("Go to checkout was requested.");
  }

  return (
    <>
      <h1>Shopping Cart</h1>

      <table className="center-element shopping-cart-width product-list-table">
        <thead>
          <tr>
            <th className="table-column-heading table-text-column">product</th>
            <th className="table-column-heading text-right"> remove / add </th>
            <th className="table-column-heading table-number-column">qty</th>
            <th className="table-column-heading table-number-column">cost</th>
          </tr>
        </thead>
        <tbody>
          <ChosenProductList
            chosenProducts={chosenProducts} />
        </tbody>
      </table>
      <table className="center-element shopping-cart-width solid-top solid-bottom
totals-table">
        <tbody>
          <Totals
            chosenProducts={chosenProducts} />
        </tbody>
      </table>
      <div className="center-element shopping-cart-width text-right">
        <button
          className="checkout-button"
          onClick={() => gotoCheckout(chosenProducts)}>checkout</button>
      </div>
    </>
  );
}

function ChosenProductList({ chosenProducts }) {

  return (
    <>
      {chosenProducts.map((product) => (
        <tr key={product.id}>
          <ChosenProduct product={product} />
        </tr>
      ))}
    </>
  );
}

function ChosenProduct({ product }) {

  return (
    <>
      <td className="table-text-column">{product.productName}</td>
      <td className="text-right">
        <DecreaseButton product={product} /> / <IncreaseButton product={product}
/>
      </td>
      <td className="table-number-column">{product.quantity}</td>
      <td className="table-number-column">${(product.unitPrice *
product.quantity).toFixed(2)}</td>
    </>
  );
}
```

```
function DecreaseButton({ product }) {

  const decreaseQty = (product) => {
    console.log(`${product.productName} decreased`);
  };

  return (
    <>
      <button className="decrease-qty-button" onClick={() =>
decreaseQty(product)}> - </button>
    </>
  );
}

function IncreaseButton({ product }) {

  const increaseQty = (product) => {
    console.log(`${product.productName} increased`);
  };

  return (
    <>
      <button className="increase-qty-button" onClick={() =>
increaseQty(product)}> + </button>
    </>
  );
}

function Totals({ chosenProducts }) {

  const productsPrice = chosenProducts.reduce((acc, cur) => { return acc +
(cur.quantity * cur.unitPrice) }, 0);
  const taxPrice = productsPrice * 0.05;
  const totalPrice = productsPrice + taxPrice;

  return (
    <>
      <tr>
        <td className="table-text-column">products total</td>
        <td className="table-number-column">${productsPrice.toFixed(2)}</td>
      </tr>
      <tr>
        <td className="table-text-column">tax</td>
        <td className="table-number-column">${taxPrice.toFixed(2)}</td>
      </tr>
      <tr className="text-strong">
        <td className="table-text-column">Grand Total</td>
        <td className="table-number-column">${totalPrice.toFixed(2)}</td>
      </tr>
    </>
  );
}

export default App;
```

index.css

```
body {
  margin: 0;
  font-family: comic-sans-ms, badaboom, cursive, Arial, Helvetica, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

.App {
  text-align: center;
}

.center-element {
  margin-left: auto;
  margin-right: auto;
}

.table-text-column {
  text-align: left;
}

.table-number-column, .text-right {
  text-align: right;
}

.product-list-table, .totals-table {
  margin-bottom: 1.0em;
}

.table-column-heading {
  text-decoration: underline;
}
```

```css
.text-strong {
  font-weight: bold;
}

.shopping-cart-width {
  width: 90%;
}

.solid-top {
  border-top: solid;
}

.solid-bottom {
  border-bottom: solid;
}

button {
  border-radius: 4px;
}

.checkout-button {
  padding: 0.2em 1em;
  margin: 0.5em 0.0em;
  font-size: large;
  font-family: comic-sans-ms, badaboom, cursive, Arial, Helvetica, sans-serif;
  font-weight: bold;
  background-color: lightblue;
}

.increase-qty-button, .decrease-qty-button {
  padding: 0.0em 0.8em;
  font-size: large;
  margin-top: 0.5em;
}

.increase-qty-button {
  color: white;
  background-color: green;
}

.decrease-qty-button {
  color: white;
  background-color: red;
}
```

Appendix 3 – static version of source code: Checkout

Here is the source code for the static version of the *Checkout* React component. The static version is a simplified intermediate prototype, created during the recommended development process described at https://react.dev/learn/thinking-in-react#step-2-build-a-static-version-in-react

To keep things simple, all JavaScript code is in one file, App.js, and all CSS is in one file, index.css

App.js

```
const FINAL_PURCHASE = {
  "shippingInformation": {
    "customerName": "Bart Simpson",
    "address": "742 Evergreen Terr.\nSpringfield\nAmerica\n"
  },

  "payment": {
    "paymentMethod": "credit card",
    "cardNumber": "1234 1234 1234 1234",
    "nameOnCard": "Homer Simpson",
    "cardExpirationDate": "09/97",
    "securityCode": "123"
  },
};

const CHOSEN_PRODUCTS = [
  {
    "id": "1",
    "productName": "a cuppa",
    "productDetails": "A cup of tea and a bickie.",
    "unitPrice": 39.99,
    "quantity": 2,
  },
  {
    "id": "2",
    "productName": "is potato",
    "productDetails": "Just eat; I promise you will like.",
    "unitPrice": 5.20,
    "quantity": 35,
  },
];

function App() {
  return (
    <div className="App">
```

```
      <Checkout />
    </div>
  );
}

function Checkout() {
  const onPay = () => {
    console.log("you pressed pay");
  }

  return (
    <>
      <h1>Checkout</h1>

      <table className="center-element checkout-totals-width totals-table">
        <tbody>
          <Totals
            chosenProducts={CHOSEN_PRODUCTS} />
        </tbody>
      </table>

      <PurchaseInfo finalPurchase={FINAL_PURCHASE} />
      <button
        className="pay-button"
        onClick={() => onPay()}>
        pay
      </button>
    </>
  );
}

function Totals({ chosenProducts }) {
  const productsPrice = chosenProducts.reduce((acc, cur) => { return acc +
(cur.quantity * cur.unitPrice) }, 0);
  const taxPrice = productsPrice * 0.05;
  const totalPrice = productsPrice + taxPrice;

  return (
    <>
      <tr>
        <td className="table-text-column">products total</td>
        <td className="table-number-column">${productsPrice.toFixed(2)}</td>
      </tr>
      <tr>
        <td className="table-text-column">tax</td>
        <td className="table-number-column">${taxPrice.toFixed(2)}</td>
      </tr>
      <tr className="text-strong">
        <td className="table-text-column">Grand Total</td>
        <td className="table-number-column">${totalPrice.toFixed(2)}</td>
      </tr>
    </>
  );
}

function PurchaseInfo({ finalPurchase }) {
  return (
    <>
      <div className="flex-container">
        <Shipping shippingInformation={finalPurchase.shippingInformation} />
        <Payment paymentInformation={finalPurchase.payment} />
      </div>
    </>
  );
}

function Shipping({ shippingInformation }) {
  return (
    <>
      <fieldset>
        <legend>Shipping</legend>
        <label htmlFor="cname">name</label>
        <input
          className="right"
          type="text"
          id="cname"
          value={shippingInformation.customerName}
          onChange={e => { shippingInformation.customerName = e.target.value; }}
          placeholder="Your name.." />
        <br />
        <label htmlFor="caddress">address</label>
        <textarea
          className="right"
          id="caddress"
          rows="4"
          cols="20"
          wrap="hard"
          value={shippingInformation.address}
          onChange={e => { shippingInformation.address = e.target.value; }} />
      </fieldset>
    </>
  );
}

function Payment({ paymentInformation }) {
  return (
    <>
      <fieldset>
        <legend>Payment</legend>
        <label htmlFor="method">method</label>
        <select
          className="right"
          id="method"
          value={paymentInformation.paymentMethod}
          onChange={e => { console.log(e.target.value); }}>
          <option value="credit card">credit card</option>
          <option value="paypal">pay pal</option>
          <option value="mypay">my pay</option>
        </select>
        <br />
        <label htmlFor="cardnumber">card number</label>
        <input
          className="right"
          type="text"
          id="cardnumber"
          value={paymentInformation.cardNumber}
          onChange={e => { paymentInformation.cardNumber = e.target.value; }}
          placeholder="Your name.." />
        <br />
        <label htmlFor="nameOnCard">name on card</label>
        <input
          className="right"
          type="text"
          id="nameOnCard"
          value={paymentInformation.nameOnCard}
          onChange={e => { paymentInformation.nameOnCard = e.target.value; }}
          placeholder="Your name.." />
        <br />
        <label htmlFor="cardExpirationDate">expiry</label>
        <input
          className="right"
          type="text"
          id="cardExpirationDate"
          value={paymentInformation.cardExpirationDate}
          onChange={e => { paymentInformation.cardExpirationDate =
e.target.value; }}
          placeholder="Your name.." />
        <br />
        <label htmlFor="securityCode">security code</label>
        <input
          className="right"
          type="text"
          id="securityCode"
          value={paymentInformation.securityCode}
          onChange={e => { paymentInformation.securityCode = e.target.value; }}
          placeholder="Your name.." />

      </fieldset>
    </>
  );
}

export default App;
```

Index.css

```
body {
  margin: 0;
  font-family: comic-sans-ms, badaboom, cursive, Arial, Helvetica, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

.App {
  text-align: center;
}

.center-element {
  margin-left: auto;
  margin-right: auto;
}

input, textarea, select {
  font-family: comic-sans-ms, badaboom, cursive, Arial, Helvetica, sans-serif;
  margin-bottom: 0.3em;
}
```

```css
.table-text-column {
  text-align: left;
}

.table-number-column, .text-right {
  text-align: right;
}

.product-list-table, .totals-table {
  margin-bottom: 1.0em;
}

.text-strong {
  font-weight: bold;
}

.checkout-totals-width {
  width: 40%;
}

button {
  border-radius: 4px;
}

.checkout-button, .pay-button {
  padding: 0.0em 1.0em 0.5em 1.0em;
  margin: 0.5em 0.0em;
  font-size: large;
  font-family: comic-sans-ms, badaboom, cursive, Arial, Helvetica, sans-serif;
  font-weight: bold;
  background-color: lightblue;
}

.flex-container {
  display: flex;
}

/*
The third row of the totals table contains the grand total,
so we put a solid thin line at the top of that table row to imply
that the grand total is the business "bottom-line".
See:
  https://stackoverflow.com/questions/20872200/giving-a-border-to-an-html-table-
row-tr
*/
```

```
table.totals-table { border-collapse: collapse; }
table.totals-table tr:nth-child(3) { border-top: solid thin; }

fieldset { border-width: thin; }
fieldset {width: 100%; margin: 20px;}
fieldset .right {float: right; width: 55%;}
fieldset .right {float: right;}
fieldset label {float:left;}
fieldset label {text-align: right;}
fieldset legend {margin-bottom: 0.2em;}
```

REFERENCES

- DTEK Online Store – Documentation
  https://rob-das-win.azurewebsites.net/docs/OnlineStoreUserManual.pdf
  https://rob-das-win.azurewebsites.net/docs/OnlineStoreDesign.pdf
  https://rob-das-win.azurewebsites.net/docs/OnlineStoreMaintenance.pdf

- Design Approach
  https://www.youtube.com/watch?v=W8smyf1eHFk
  https://www.youtube.com/watch?v=pN92rnO_n5U

- Implementation with React
  https://react.dev/learn/thinking-in-react

- Using the react router to pass data between pages
  https://stackoverflow.com/questions/52238637/react-router-how-to-pass-data-between-pages-in-react
  https://stackoverflow.com/questions/72880492/react-state-on-multiple-pages
  https://stackoverflow.com/questions/52253003/how-to-redirect-one-page-to-another-page-in-reactjs
  https://youtu.be/Udbgwn1K0o8?si=dkUUATHSeLeMnHcb
  https://www.google.com/search?q=react+router+sharing+state+between+pages&rlz=1C1CHZN_enCA944CA944&oq=react+router+sharing+state+between+pages&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIHCAEQABiiBNIBDzIxNTk1MzQzNThqMGoxNagCALACAA&sourceid=chrome&ie=UTF-8

- Net Ninja Full React Tutorial #25
  https://youtu.be/t7VmF4WsLCo?si=mC9Pd5n18u0J5A7e