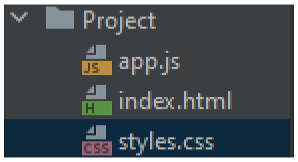
# Restaurant Application project:

Your task is to create a complete restaurant application project with JavaScript and html based on the tasks before and material on the JavaScript Essentials Moodle course.

Your GitHub repository should be already set up from the previous tasks.
Update your progress on GitHub via the instructions provided in the project readme.md. Commit and push your changes after each task is done.

**Getting started**

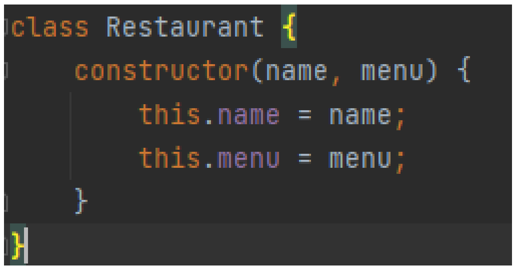
Create a new empty project inside your IDE of choice. Create a new folder with the following files: app.js, index.html and styles.css like following:



```
Project
    app.js
    index.html
    styles.css
```

**Task 1: Creating the classes**

Create a new class inside app.js called Restaurant: The restaurant class should have a constructor which sets following properties: name and menu.



```
class Restaurant {
    constructor(name, menu) {
        this.name = name;
        this.menu = menu;
    }
}
```

Create a similar classes called FoodItem and a User
The foodItem should have following properties:

- name
- price
- description

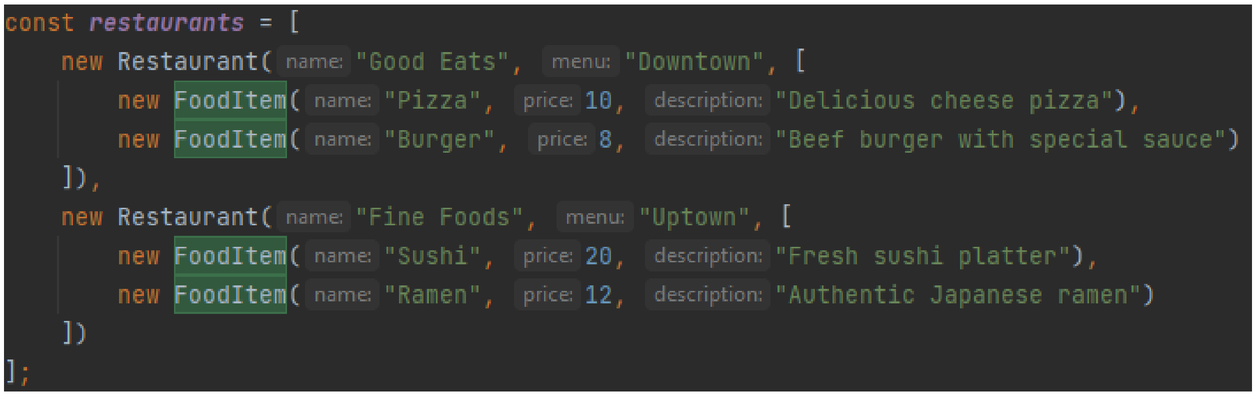
And the User class should have following properties:

- name
- accountBalance

**Task 2: Populate the classes.**

Create a new variable called **restaurants**. It should be an array of the new objects of the Restaurant class. The restaurants should have a object of type FoodItem()

Assign values of your choice for each property: name, price and description.



```
const restaurants = [
    new Restaurant( name: "Good Eats",  menu: "Downtown", [
        new FoodItem( name: "Pizza",  price: 10,  description: "Delicious cheese pizza"),
        new FoodItem( name: "Burger",  price: 8,  description: "Beef burger with special sauce")
    ]),
    new Restaurant( name: "Fine Foods",  menu: "Uptown", [
        new FoodItem( name: "Sushi",  price: 20,  description: "Fresh sushi platter"),
        new FoodItem( name: "Ramen",  price: 12,  description: "Authentic Japanese ramen")
    ])
];
```

Create a similar object for the user. It should be a new User() object with a name and account balance.

Commit and push your changes and move to the next step.

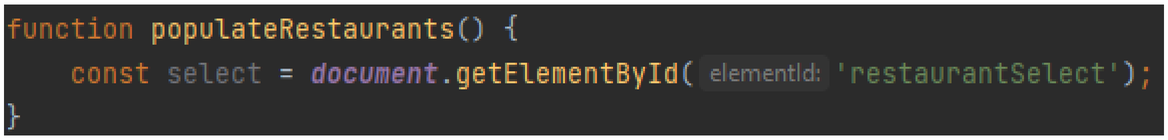
**Task 3: Creating the HTML elements:**
You can copy paste the following HTML template to your index.html file to get started:

```
<!DOCTYPE html>
<html lang="en">
   <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <link rel="stylesheet" href="styles.css">
   <title>Food Ordering App</title>
</head>
<body>
<h1>Food Delivery App</h1>
<div id="app">
   <section>
       <h2>Select a Restaurant</h2>
       <select id="restaurantSelect"></select>
   </section>
   <section>
       <h2>Menu</h2>
       <div id="menuItems"></div>
   </section>
   <section>
       <h2>Order Summary</h2>
       <div id="orderSummary"></div>
   </section>
</div>

<script src="app.js"></script>
</body>
</html>
```

It sets the initial elements, files and scripts to be used in the html file. Note that the elements contain id values such as `<div id="orderSummary">` These values will be later used to assign these elements to scripts, so do not change them.

**Task 4: Populating the HTML elements**

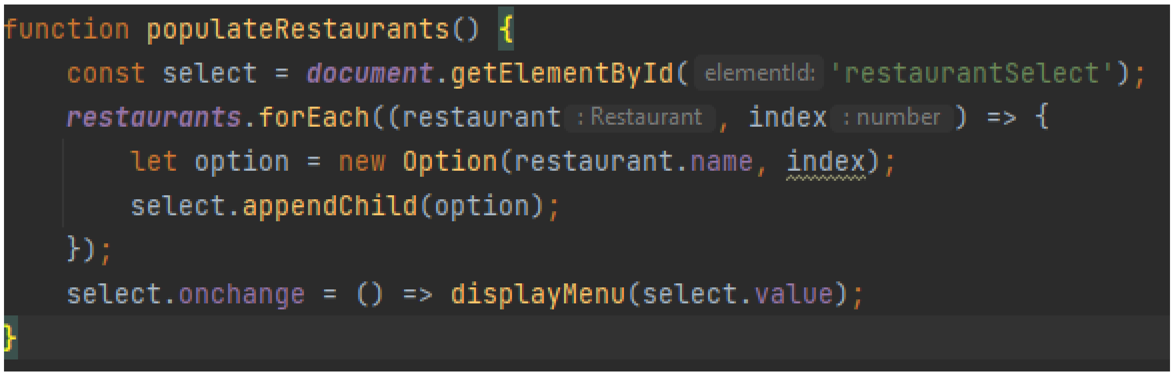
We need to now populate the created html elements with the data we have created. Create a function inside app.js called populateRestaurants(). This function should first select the html element with id “restaurantSelect” from the index.html.



```
function populateRestaurants() {
    const select = document.getElementById('restaurantSelect');
}
```

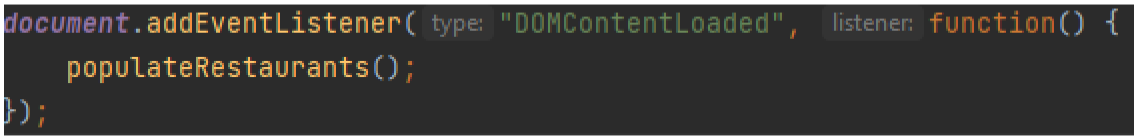
The document.getElementId() can be used to select html elements by id.

We then need to map the created restaurant element to the id. The restaurantSelect is a “<select>” element which needs a value and a index for each option so that the elements inside it will be selectable and their value can be outputted. The appendChild() method can be used to create new html elements inside a parent element: in this case we are creating <option> elements for the <select> element with our data inside. We wrap this into a forEach() function which will go through each element in our restaurant array and assign the restaurant and index values to it.
Finally we need to create function which sets the displayMenu() value to be filled with our data.



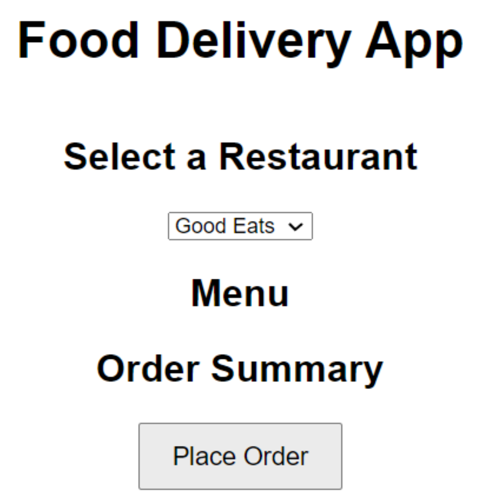
```
function populateRestaurants() {
    const select = document.getElementById( elementId: 'restaurantSelect');
    restaurants.forEach((restaurant : Restaurant , index : number ) => {
        let option = new Option(restaurant.name, index);
        select.appendChild(option);
    });
    select.onchange = () => displayMenu(select.value);
}
```

To make sure that the populateRestaurants() runs each time we load our page, we need to create a function  which listens to if our content on our page is loaded. Add the following function which listens for content loading on our page and calls the populateRestaurants() function to app.js :



```
document.addEventListener( type: "DOMContentLoaded",  listener: function() {
    populateRestaurants();
});
```

You should now have a selector where you can see your created restaurant options and select them like below. Commit and push your changes with git and proceed to the next task.



Food Delivery App

Select a Restaurant

Good Eats

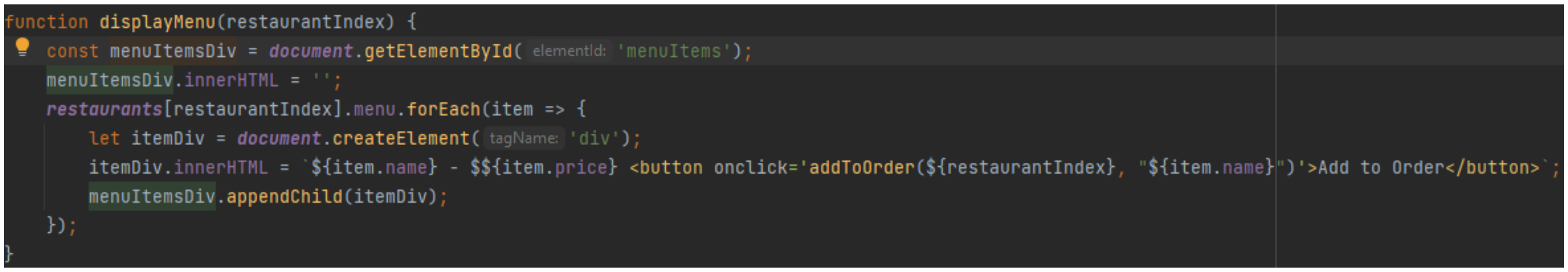
Menu

Order Summary

Place Order

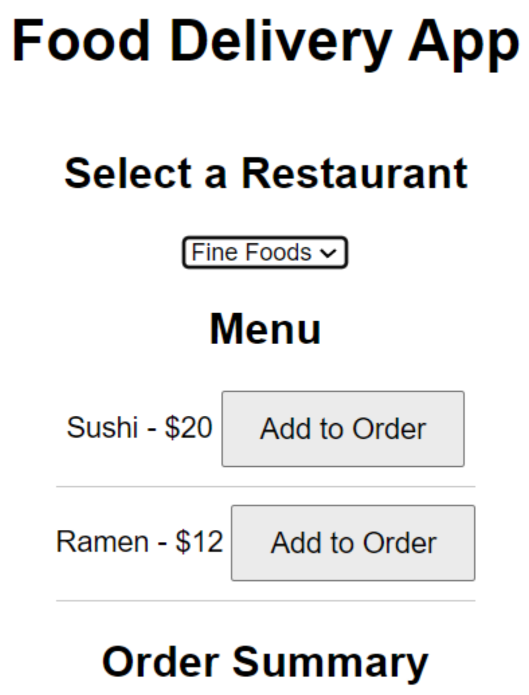
**Task 5: Displaying menu data**

We now need to create a new function called displayMenu(). This function should be similar to the last one. We need to select an element from the html document called “menuItems” and append our own data to it. We will want to create <button> for each element. The html <button> element takes a parameter called onClick() which can call a function when it is pressed. We will now assign a function with parameters inside it which will be created in the next part. Inside a html element code we can use ${} to pass direct variable values to functions and elements.



```
function displayMenu(restaurantIndex) {
    const menuItemsDiv = document.getElementById('menuItems');
    menuItemsDiv.innerHTML = '';
    restaurants[restaurantIndex].menu.forEach(item => {
        let itemDiv = document.createElement('div');
        itemDiv.innerHTML = `${item.name} - $${item.price} <button onclick='addToOrder(${restaurantIndex}, "${item.name}")'>Add to Order</button>`;
        menuItemsDiv.appendChild(itemDiv);
    });
}
```

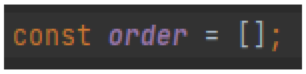
You can now view your page by opening the index.html: You should be able to select a restaurant you created and see the menu items inside it like below. Commit and push your changes to GitHub and move to the next task.



**Task 6: Creating Orders**

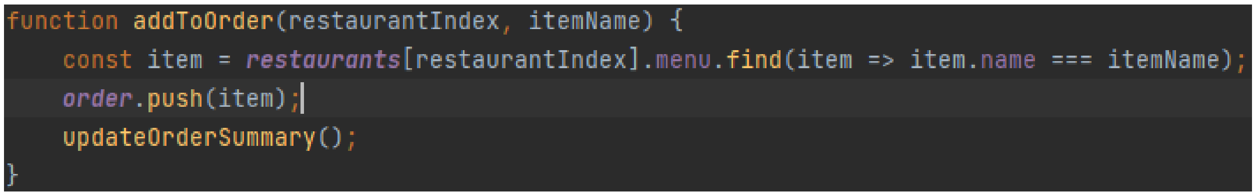
Now that we have restaurant and menu items showing on our page, we need a way to create orders.

We first need to create a empty array for orders:



```
const order = [];
```

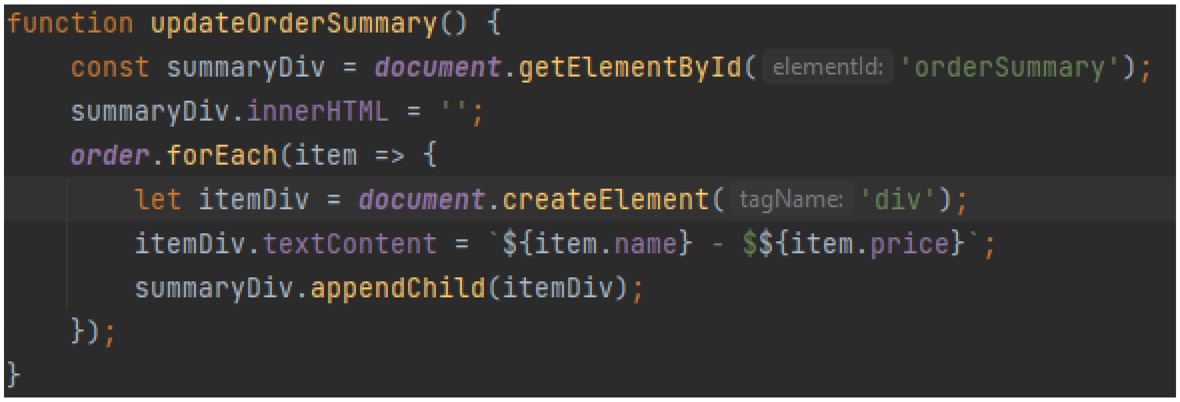
We need to create a new function called addToOrder which will push an new object to this array of the type of our selection.



```
function addToOrder(restaurantIndex, itemName) {
    const item = restaurants[restaurantIndex].menu.find(item => item.name === itemName);
    order.push(item);
    updateOrderSummary();
}
```

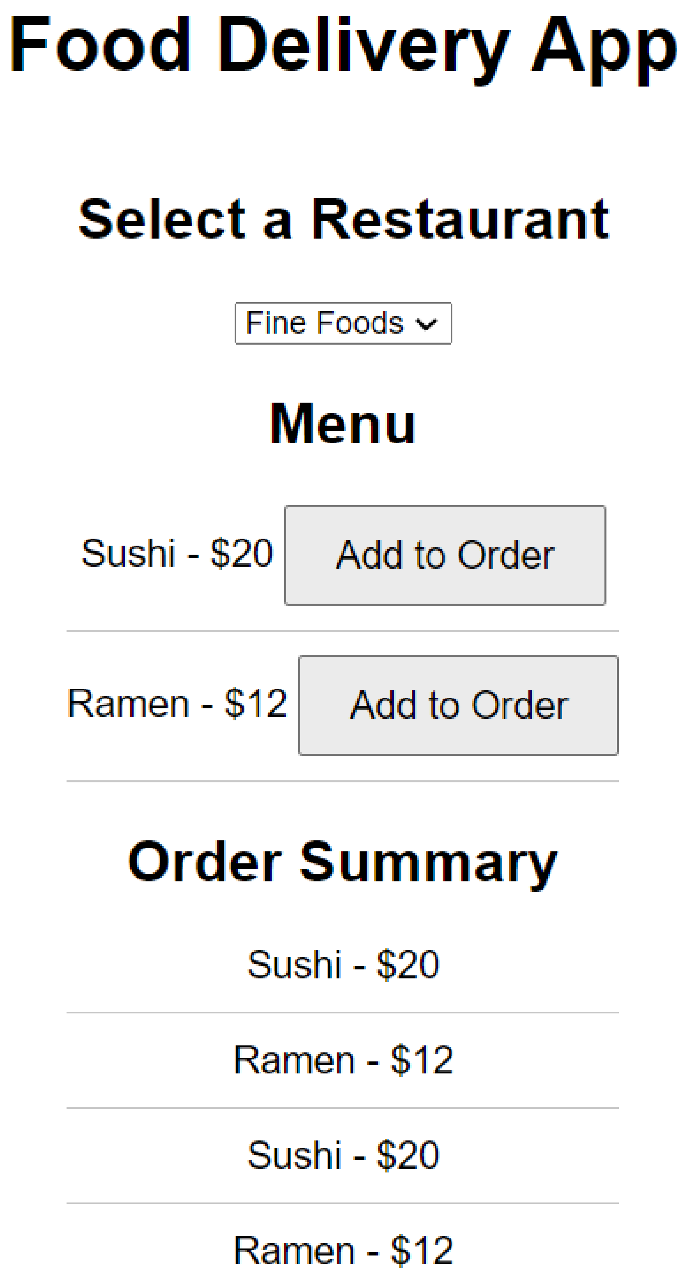
Note the use of the methods find() and push(). With these we can first find an item from the restaurants array which matches our selection's name and then add this information to the order[] array. We also need to update the pages html elements accordingly, so we call a function updateOrderSummary(); which we'll create next.

Once again we will need to select an html element and populate it accordingly with child elements. This time we will populate it with elements from the order array.



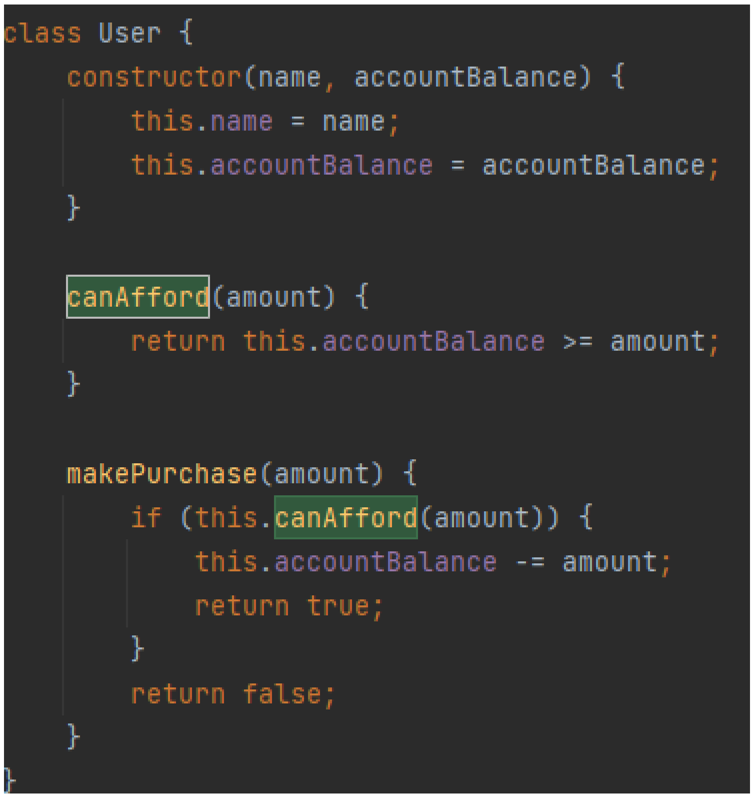
```
function updateOrderSummary() {
    const summaryDiv = document.getElementById( elementId: 'orderSummary');
    summaryDiv.innerHTML = '';
    order.forEach(item => {
        let itemDiv = document.createElement( tagName: 'div');
        itemDiv.textContent = `${item.name} - $${item.price}`;
        summaryDiv.appendChild(itemDiv);
    });
}
```

You can now view the changes in the html document. You should be able to add items to the order summary and see them update. Commit and push your changes to GitHub and proceed to the next task.



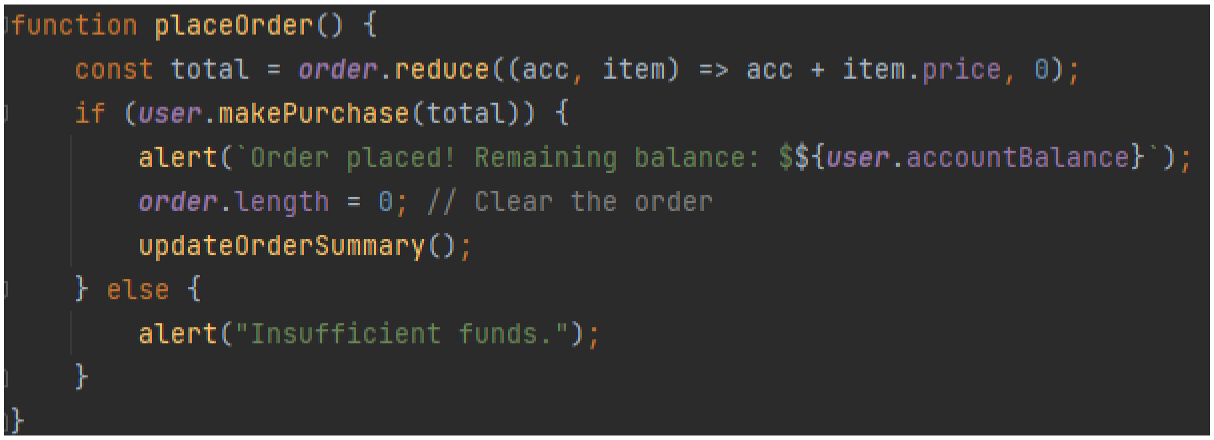
**Task 7: Placing the order**

We will now create a function which will check the users balance and buy the items that we have selected. First we need to update the User class with a few new functions: canAfford() and makePurchase().



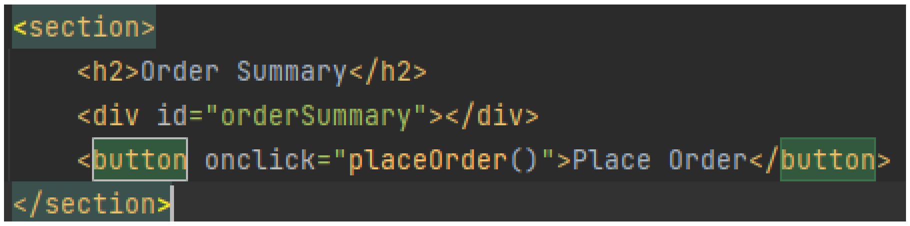
```
class User {
    constructor(name, accountBalance) {
        this.name = name;
        this.accountBalance = accountBalance;
    }

    canAfford(amount) {
        return this.accountBalance >= amount;
    }

    makePurchase(amount) {
        if (this.canAfford(amount)) {
            this.accountBalance -= amount;
            return true;
        }
        return false;
    }
}
```

We then need to create a new function based on these. It should remove the price of the items from the accounts balance and cancel the return “insufficient funds” if the balance is not enough. The alert() method can be used to display a window alert to notify the user of changes. We also want to clear the orders array and update the orderSummary() each time a purchase is made.



```
function placeOrder() {
    const total = order.reduce((acc, item) => acc + item.price, 0);
    if (user.makePurchase(total)) {
        alert(`Order placed! Remaining balance: $${user.accountBalance}`);
        order.length = 0; // Clear the order
        updateOrderSummary();
    } else {
        alert("Insufficient funds.");
    }
}
```

Lastly, we want to add a button element to the last <section> element in our index.html where we can call the placeOrder() function and make our purchase.



```
<section>
    <h2>Order Summary</h2>
    <div id="orderSummary"></div>
    <button onclick="placeOrder()">Place Order</button>
</section>
```

You should now have an application where you can select restaurants and buy items from. Check that all of the functionality works in your application as intended and commit and push the changes to github.

**Task 8: Add your own functionality**

Add at least 3 more features of your choice to the application.
Some examples of preferred functionality:

- Ability to add and view currency of the user
- Ability to add restaurants/food items from the user interface
- Additional properties to the food items / restaurant classes with html elements to display the data.

**Task 9: Styling**

You can copy/paste the following base to your styles.css file.

```
body {
   font-family: Arial, sans-serif;
}

section {
   margin-bottom: 20px;
}

#menuItems div, #orderSummary div {
   margin-top: 10px;
   border-bottom: 1px solid #ccc;
   padding-bottom: 10px;
}

button {
   padding: 10px 20px;
```

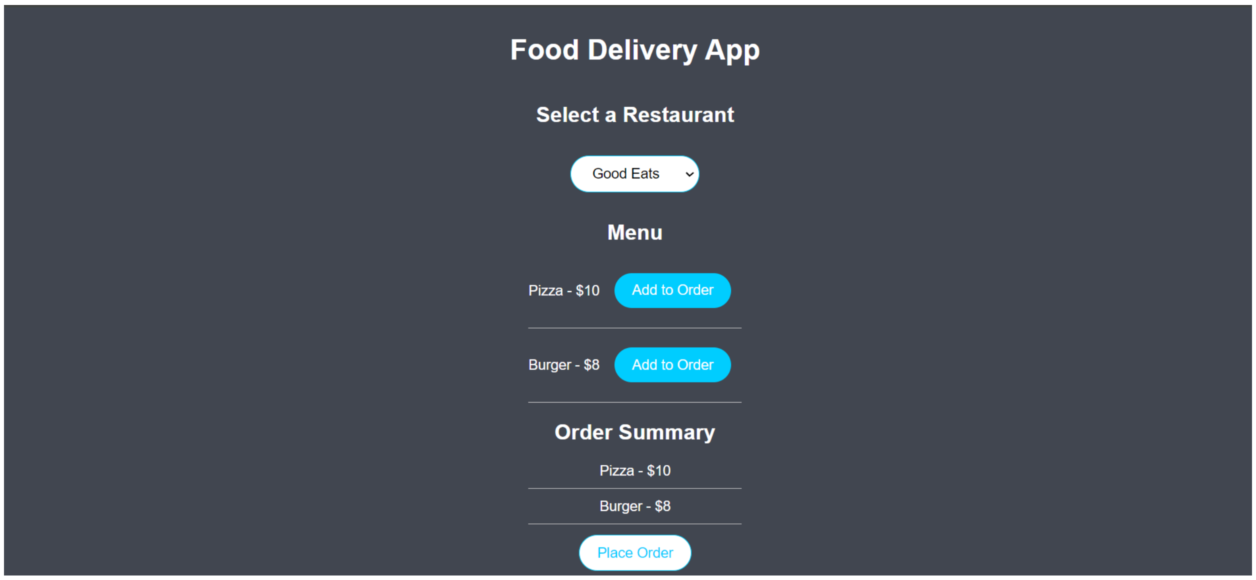
```
    font-size: 16px;
    cursor: pointer;
}
```

Add your own styling to the page. You can find basic instructions for css styling on here: https://www.w3schools.com/css/

The styling should follow at least the following rules:

- All of the content should be centered on the page
- Proper padding and margins on all of the elements.
- Style the button and selector inputs with your own styles and colors.

Your finished project should look something like this with your own functionality added:
Commit and push all your code to GitHub.


Food Delivery App
Select a Restaurant
Good Eats
Menu
Pizza - $10
Add to Order
Burger - $8
Add to Order
Order Summary
Pizza - $10
Burger - $8
Place Order