Instrukcja

Laboratorium 8

Ćwiczenie – przygotowanie

Stwórz **fork** repozytorium otrzymanego przez prowadzącego.

https://github.com/ruljin/laboratory-classes-8

Następnie, sklonuj to repozytorium ze swojego **GitHub**. Wykonanie tych kroków, pozwoli Ci na rozpoczęcie pracy nad ćwiczeniem. Pamiętaj, żeby wszelkie zmiany wypychać na bieżąco na **zdalne repozytorium**, dzięki temu, nawet jeśli nie skończysz pracy na czas, na bieżąco będziesz wypychać swoje zmiany lokalne na **zdalne repozytorium**, dzięki czemu możesz uzyskać zaliczenie, dzięki cząstkowym rozwiązaniom.

Podstawą do oceny zadania z ćwiczeń jest **oddanie linku do repozytorium** poprzez **zadania Teams** w wyznaczonym przez prowadzącego **terminie**. Drugim kryterium, jest oddanie **działającego** zadania. Zadanie, które się nie uruchamia **nie będzie oceniane**.

Repozytorium powinno zawierać w zależności od zadania pliki we formacie **.html**, **.ejs**, **.js**, **.css** lub też **.json**.

Osoby, których zadania niosą znamiona pracy grupowej (np. kilku autorów commitów) otrzymają ocenę **niedostateczną**. Jeżeli struktura zadania istnieje, to jeżeli zadanie tego nie wymaga nie należy jej modyfikować, rozbudowywać lub też dodawać nowe elementy do struktury repozytorium/projektu.

Zadanie

Celem bieżącego ćwiczenia jest **poprawienie niedociągnięć** oraz **błędów w aplikacji**, ulepszenie jej funkcjonalności, a także podniesienie jakości wizualnej i ogólnego odbioru poprzez poprawę interfejsu użytkownika (**UI**) oraz doświadczenia użytkownika (**UX**). Działania mają charakter usprawniający i optymalizujący istniejące rozwiązania.

Poprawki w nawigacji dla views/partials/navigation.ejs

Celem tego etapu jest poprawienie działania i wyglądu elementu nawigacji odpowiadającego za wyświetlanie liczby produktów w koszyku.

- uaktywnij **navigation__badge** koszyka zmieniając jego **display: none** na **display: block**,
- uwarunkuj wyświetlanie **navigation__badge** od wartości **cartCount**. Wyświetlaj go jedynie, gdy jego wartość jest **większa od 0**,
- popraw przekazywanie **cartCount** do widoków. Przejrzyj wszystkie kontrolery, widoki i upewnij się, że do funkcji **res.render()** przekazywana jest wartość **cartCount**. W razie potrzeby, zmodyfikuj odpowiednio metody kontrolerów.

Przeniesienie logiki dodawania produktów do controllers/productsController.js

W tym kroku należy uporządkować odpowiedzialność za dodawanie produktów, przenosząc logikę związaną z dodaniem nowego produktu do odpowiedniego kontrolera.

- dodaj metodę **addProduct** w **productsController**, która korzysta z **Product.add()** do dodania nowego produktu. Przekieruj użytkownika na stronę **/products/new** z kodem przekierowania **FOUND** (wykorzystaj odpowiednie źródło),
- zaktualizuj **routing** w **routes/products.js**. Zastąp istniejące przypisanie ścieżki **POST „/add”**,
- usuń zbędne importy, jeśli nie są już nigdzie używany. Przejrzyj inne kontrolery i pliki powiązane z tym procesem, aby usunąć nieużywane już referencje.

Usuwanie produktu wraz z jego wystąpieniami w koszyku

W celu poprawnego usuwania produktu z systemu, należy zadbać również o jego usunięcie z koszyka, jeśli został wcześniej dodany przez użytkownika.

- w pliku **models/Cart.js** zaimplementuj metodę statyczną **deleteProductByName**, która czyści produkt w koszyku, jeśli znajduje się w nim produkt o danej nazwie,
- zmodyfikuj metodę **deleteByName** w modelu **Product** tak, aby po usunięciu produktu z kolekcji **products**, wywoływał **Cart.deleteProductByName()** by usunąć produkt również z koszyka,
- zadbaj o poprawne importy.

Uzupełnienie i korekta styli dla views/partials/card.ejs

Celem tego etapu jest poprawienie stylów dla widoku kart produktów, w szczególności linków i przycisków, które dotychczas były nieczytelne lub źle ostylowane.

- dodaj nowe klasy stylów do **public/css/main.css**:

```css
.card__link {
  max-height: 45px;
  display: inline-block;
  padding: 0.75rem 1.5rem;
  background-color: #4caf50;
  color: white;
  border: none;
  border-radius: 0.3rem;
  cursor: pointer;
  font-size: 1.1rem;
  transition: background-color 0.3s ease, transform 0.2s ease;
  text-decoration: none;
  margin-right: 1rem;
}

.card__link:hover {
  background-color: #388e3c;
  transform: translateY(-2px);
}

.card__link:active {
  background-color: #2e7d32;
  transform: translateY(0);
}

.card__button {
  padding: 0.75rem 1.5rem;
  background-color: #dc3545;
  color: white;
  border: none;
  border-radius: 0.3rem;
  cursor: pointer;
  font-size: 1.1rem;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.card__button:hover {
  background-color: #c82333;
  transform: translateY(-2px);
}

.card__button:active {
  background-color: #b21f2d;
  transform: translateY(0);
}
```

- zastosuj odpowiednie klasy w **views/partials/card.ejs**. Upewnij się, że **<button>** ma klasę **card__button** a **<a>** ma klasę **card__link**,
- sprawdź i popraw ewentualne literówki lub niespójności klas w **main.css** oraz **card.ejs**. Upewnij się, że nie występują literówki lub błędne nazwy klas (np. **card_price**, **card-btn** itp.)
- zamień je wszędzie na zgodne z aktualnym nazewnictwem: **.card__price**, **card__button** itp..

Uzupełnienie widoku views/product.ejs, stylów oraz obsługi dodawania do koszyka

W tym kroku, należy rozszerzyć widok **product.ejs**, dodać brakujące style, zabezpieczyć widok, a także rozbudować funkcjonalność dodawania produktu do koszyka.

- dodaj element **<p>** wyświetlający cenę produktu z klasą **product__price**,
- dodaj element **<button>** o treści **Add to Cart**, który wywołuje metodę **addToCart(productName)** i klasach **product__button** i **product__button—add**,
- dodaj do **<button>** odpowiedzialnego za usuwanie klasy **product__button** i **product__button—delete**,
- zabezpiecz widok na wypadek braku danych,
- dodaj brakujące style do **public/css/main.css**

```
.product__title {
  font-size: 1.5rem;
  margin-bottom: 0.5rem;
  color: #2c3e50;
}

.product__description {
  color: #666;
  margin-bottom: 1rem;
}

.product__price {
  font-size: 1.2rem;
  font-weight: 700;
  color: #4caf50;
}

.product__button {
  padding: 0.75rem 1.5rem;
  color: white;
  border: none;
  border-radius: 0.3rem;
  cursor: pointer;
  font-size: 1.1rem;
  transition: background-color 0.3s ease, transform 0.2s ease;
  width: 100%;
}

.product__button--delete {
  background-color: #dc3545;
}
```

```css
.product__button--delete:hover {
  background-color: #c82333;
  transform: translateY(-2px);
}

.product__button--delete:active {
  background-color: #b21f2d;
  transform: translateY(0);
}

.product__button--add {
  background-color: #4caf50;
}

.product__button--add:hover {
  background-color: #388e3c;
  transform: translateY(-2px);
}

.product__button--add:active {
  background-color: #2e7d32;
  transform: translateY(0);
}
```

- dodaj (w osobnym pliku lub we widoku **product.ejs**) metodę **addToCart()**, która wywoła **fetch** do **„/cart/add”** i metodą **POST**. W przypadku otrzymania sukcesu (**response.ok**) dokona przeładowania okna,
- dodaj **routing POST** dla **„/card/add”** w **/routes/cart.js**. W odpowiedzi, kontroler powinien zwrócić status **OK** (wykorzystaj odpowiednie źródło) bez przekierowania,
- zmodyfikuj **cartController.addProductToCart**, tak by wyszukiwała za pomocą **Product.findByName** produktu, który ma dodać a następnie przekazywała go do **Cart.add**. Dodaj zabezpieczenia na niekompletny obiekt na poziomie metody **add**,
- zmodyfikuj metodę **Cart.add** w **models/Cart.js** tak, aby przyjmowała bezpośrednio **product**, nie pobierała już produktu samodzielnie i była odporna na błędne dane,
- usuń zbędny kod (usuń stare wersje metody **Cart.add**, przejrzyj i oczyść kontrolery/routingi z nieużywanych fragmentów kodu, zwłaszcza starego sposobu dodawania produktu do koszyka).