Wzorzec MVC w tworzeniu aplikacji internetowych

Laboratorium 6

Ćwiczenia – Przygotowanie

Ćwiczenie przewiduje **trzy** (**3**) zadania. Utwórz nowy projekt w swoim **edytorze kodu** lub **IDE**, w którym utworzysz odpowiednio **cztery katalogi**:

- zad1,
- zad2,
- zadX,
- (…),
- zad3.

Gdzie **X** to numer porządkowy, każdego następnego zadania od **pierwszego** do **następnego**, aż do ostatniego. Zrobienie tej czynności na początku, ułatwi Ci zarządzanie projektem podczas zajęć, więc warto byś ten krok wykonał na starcie. Katalogi przeznaczone są na zawartość w postaci **plików wymaganych** do uruchomienia **konkretnego zadania**. Utworzone pliki, muszą mieć odpowiedni format. W zależności od zadania, będzie to plik we formacie **.js**, **.html**, **.ejs** lub też **.json**.

PRZYPOMNIENIE: Zadania są rozwiązywane **w podanej kolejności** (czyli najpierw **zadanie 1** a następnie **zadanie 2** itd.). Każde następne zadanie powinno wykorzystywać **kod z poprzedniego zadania** jako kod początkowy.

Odpowiednie **zadanie** (czyli zadanie zawarte w **reprezentującym je katalogu** np. **zad1**) zawiera **jedynie** taką ilość kodu, jaka jest potrzebna na rozwiązanie danego podpunktu.

Oddawanie zadań, które niosą znamiona np. rozwiązania **następnego punktu** w niewłaściwym zadaniu, może rzutować na ocenę końcową. Dodatkowo, oddawanie zadań:

- **kompletnych** lub tzw. **komplementarnych** (zawierających rozwiązanie na **wszystkie podpunkty**) jako np. jedynie rozwiązanie,
- oddanie zadania w sposób inny niż jako **link do repozytorium** (za pomocą **zadań teams** – dopuszczalne jest dołączenie ewentualnie **dokumentu tekstowego** zawierającego **link do repozytorium**),
- **po terminie**,
- niezgodnego z **zaproponowaną strukturą** (czyli innej niż z **podziałem zadań z ćwiczenia na katalogi** lub **podziałem zadań z ćwiczenia na branche**),

Będzie skutkowało otrzymaniem oceny **2.0.**

Zadanie 1

Stwórz folder projektu **mvc-lab-6** (jeżeli jeszcze tego nie zrobiłeś lub nie dzielisz zadań na **branch'e**) a następnie zainicjuj projekt **Node.JS** i zainstaluj następujące zależności:

- **express**,
- **express-session**,
- **ejs**.

Utwórz plik startowy aplikacji o nazwie **app.js** w głównym katalogu Twojego projektu oraz zaimportuj w nim potrzebne moduły (tj. **express**, **express-session**, **body-parser** oraz **path**). Skonfiguruj aplikację **express** w następujący sposób:

- ustaw katalog plików statycznych, wskazując na katalog **public**,
- skonfiguruj silnik widoków na **ejs**,
- skonfiguruj sesję z użyciem **session** (**const session = require('express-session')**),
- dodaj middleware **bodyParser** do obsługi zapytań typu **urlencoded**.

W kolejnym kroku dodaj **routing** dla obsługi błędów (plik **./routes/error**) i upewnij się, ze **requesty** są przekierowywane do tego **routera** za pomocą **app.use(„*", errorRoutes)**. Ustaw aplikację na nasłuchiwanie na porcie określonym w stałej **PORT** i dodaj logowanie w konsoli (**'Server is running on http://localhost:${PORT}'**), aby potwierdzić uruchomienie serwera.

Stwórz katalog **views** a w nim plik **not-found.ejs**, który:

- będzie korzystał z arkusza styli **styles.css** z katalogu **public**,
- będzie miał dynamicznie ładowany tytuł strony poprzez zmienną **title** przekazywaną w kontrolerze do widoku,
- w **<h1>** wyświetli treść **„404 – Page Not Found"**,
- w **<p>** wyświetli treść **„Sorry, the page you are looking for does not exist."**,
- w **<a href="/">** wyświetli treść „**Back to Home"**.

Następnie, stwórz katalog **routes** a w nim plik **error.js**, w którym zaimportuj potrzebne w nim moduły (tj. **express** oraz ../**controllers/error**) oraz:

- utwórz instancję **routera Express**,
- ustaw **router**, aby używał kontrolera błędów dla każdego żądania, które nie pasuje do innych zdefiniowanych tras (wykorzystaj w tym celu **callback** - **errorController.getNotFoundPage**),
- eksportuj **router**.

W kolejnym kroku dodaj **public/css** a w nim plik **styles.css**, który uzupełnij następująca treścią:

```css
body {
  font-family: Arial, sans-serif;
  padding: 20px;
  background-color: #f4f4f4;
}

h1,
h2 {
  color: #333;
}
```

```css
a {
  display: inline-block;
  margin-top: 20px;
  color: #0077cc;
}

input[type="number"],
select {
  margin-top: 10px;
  margin-bottom: 10px;
  display: block;
}

button {
  display: block;
  width: 120px;
  padding: 10px 20px;
  background-color: #0077cc;
  color: white;
  border: none;
  cursor: pointer;
}

button:hover {
  background-color: #005fa3;
}

button:disabled {
  background-color: #8ab6d6;
}
```

Stwórz katalog **controllers** a w nim plik **error.js** w którym:

- stwórz funkcję **getNotFoundPage**, która zwróci status **404** oraz wyrenderuje widok **not-found** o wartości zmiennej **title**, **„Page Not Found”**,
- wyeksportuj tą funkcję (upewnij się, ze używasz do tego notacji **module.exports = { NAZWA_FUNKCJI }**, aby móc używać tej funkcji w innych częściach aplikacji, np. w **routerach.**

UŻYJ KODU PONIŻEJ (w app.js) DLA SKONFIGUROWANIA SESJI:

```
app.use(
  session({
    secret: "secret",
    resave: false,
    saveUninitialized: true,
  })
);
```

Zadanie 2

Skopiuj **katalog** poprzedniego zadania (**zad1**) i zmień jego nazwę na **zad2** (jeżeli nie dzielisz zadań na **branch'e**). Następnie, zaktualizuj plik **app.js**:

- dodaj importy dla nowych modułów tras (**./routes/book** oraz **/routes.user**),
- zintegruj nowo dodane moduły tras z aplikacją (żądania do **/user** mają być obsługiwane przez **userRoutes** a żądania związane z książkami, czyli bez filtrowania przez **bookRoutes**).

PAMIĘTAJ, ŻE KOLEJNOŚĆ USTAWIANIA ROUTINGU, MA ZNACZENIE (ZWŁASZCZA - W PRZYPADKU BRAKU FILTROWANIA).

W kolejnym kroku stwórz plik **views/set-user-session.ejs**, który:

- będzie korzystał z arkusza styli **styles.css** z katalogu **public**,
- będzie miał dynamicznie ładowany tytuł strony poprzez zmienną **title** przekazywaną w kontrolerze do widoku,
- w **<h1>** wyświetli treść **„Set User"**,
- w zależności od długości tablicy **users** (przekazywanej jako zmienna do widoku) wyświetli:
    - gdy jest większa niż **0**, formularz o wartości **action="/user/set"** i **method="POST"**:
        - w ** **, wyświetli **<label>** o treści **„User"**,
        - w tym samym ** ** pod **<label>** będzie miał **<select name="userId">**, który w zależności od długości tablicy **users** wyświetli <**option value="USER_ID">** o treści **USER_LOGIN**,
        - w **<button>**, wyświetli treść „**Set user"**.
    - w innej sytuacji:
        - w **<p>**, wyświetli treść **„Sorry, no users has been found."**.
- w **<p>** wyświetli treść **„Sorry, the page you are looking for does not exist."**,
- w **<a href="/">** wyświetli treść „**Back to Home"**.

Oraz **views/books.ejs**, który:

- będzie korzystał z arkusza styli **styles.css** z katalogu **public**,
- będzie miał dynamicznie ładowany tytuł strony poprzez zmienną **title** przekazywaną w kontrolerze do widoku,
- w **<h1>** wyświetli treść **„Books"**,
- w zależności od **userId** (przekazywanej jako zmienna do widoku) wyświetli:
    - gdy **nie istnieje**:
        - w **<p>** wyświetli treść **„Sorry, you cannot access library without login into the system."**.
    - w innej sytuacji:
        - w **<ul>** dla tablicy **books** (przekazywanej jako zmienna do widoku) wyświetli:
        - dla każdego elementu **<li>** a w nim **<a href="/books/<%= book.id %">** wyświetli treść **book.title**,
        - a jako kontynuacja tego samego **<li>** wyświetli treść **by book.author**.
- w **<a href="/user/set">** wyświetli treść **„Log in"**.

W PRZYPADKU book.title ORAZ book.author CHODZI O WARTOŚĆ ARGUMENTU OBIEKTU DO KTÓREGO DOSTĘP UZYSKUJEMY W RAMACH ITERACJI TABLICY - books.

Następnie, stwórz **routes/user.js**, w którym:

- utwórz instancję **routera Express**,
- ustaw **router**, aby używał kontrolera **userController.getSetUserSession**, dla metody **GET** oraz ścieżki **„/set”**,
- ustaw **router**, aby używał kontrolera **userController.setUserSession**, dla metody **POST** oraz ścieżki **„/set”**,
- eksportuj **router**.

Oraz **routes/book.js**, w którym:

- utwórz instancję **routera Express**,
- ustaw **router**, aby używał kontrolera **bookController.getBooksList**, dla metody **GET** oraz ścieżki **„/”**,
- eksportuj **router**.

W następnym kroku, utwórz katalog **models** a w nim plik **Book.js**, który:

- zdefiniuj klasę **Book** z konstruktorem przyjmującym parametry **id**, **title**, **author**, **year** oraz opcjonalny parametr **available**, który domyślnie ustawiony jest na **true**,
- dodaj statyczną metodę tej klasy **getAll**, która zwraca tablicę wszystkich książek,
- stwórz pod kodem klasy tablicę **books** i wypełnij ją pięcioma elementami, będącymi **instancją** klasy **Book**,
- eksportuj klasę **Book**.

Oraz **models/User.js**, w którym:

- zdefiniuj klasę **User** z konstruktorem przyjmującym parametry **id**, **login**, **borrowedBooks**, który domyślnie ustawiany jest na **[]**,
- dodaj statyczną metodę tej klasy **getAll**, która zwraca tablicę wszystkich użytkowników,
- stwórz pod kodem klasy tablicę **users** i wypełnij ją pięcioma elementami, będącymi **instancją** klasy **User**,
- eksportuj klasę **User**.

W kolejnym kroku, stwórz **controllers/user.js**, w którym:

- zaimportuj model **User** (**const User = require(„../models/User”)**,
- stwórz funkcję **getSetUserSession**, która wykorzysta kod poniżej:

```
const getSetUserSession = (request, response) => {
  const users = User.getAll();
  response.render("set-user-session", { title: "Set User Session", users });
};
```

- stwórz funkcję **setUserSession**, która wykorzysta kod poniżej:

```
const setUserSession = (request, response) => {
  request.session.userId = request.body.userId;
  response.redirect("/");
};
```

- wyeksportuj te funkcje, (upewnij się, ze używasz do tego notacji **module.exports = { NAZWA_FUNKCJI }**, aby móc używać tej funkcji w innych częściach aplikacji, np. w **routerach.**

Oraz **controllers/book.js**, w którym:

- zaimportuj model **Book**,

- stwórz funkcję **getBooksList**, która wyciągnie z sesji uzytkownika **userId** (do którego dostęp możesz uzyskać w ramach **request.session.userId**), wyciągnie wszystkie książki (za pomocą modelu **Book**), wyrenderuje widok **books** o wartości zmiennej **title**, **"Books"** oraz przekaze do widoku **userId** oraz **books** (pod takimi samymi nazwami).

Zadanie 3

Skopiuj **katalog** poprzedniego zadania (**zad2**) i zmień jego nazwę na **zad3** (jeżeli nie dzielisz zadań na **branch'e**). Następnie, stwórz **views/success.ejs**, który:

- będzie korzystał z arkusza styli **styles.css** z katalogu **public**,
- będzie miał dynamicznie ładowany tytuł strony poprzez zmienną **title** przekazywaną w kontrolerze do widoku,
- w **<h1>** wyświetli treść **„Success"**,
- w **<p>** wyświetli treść poprzez zmienną **message** przekazywaną w kontrolerze do widoku,
- będzie **includować**, **partials/back.ejs**.

Oraz zmodyfikuj **set-user-session.ejs**, **not-found.ejs** tak, żeby **includowały**, **partials/back.ejs** a **books.ejs** tak, żeby **includował**, **partials/login.ejs**. Wyeliminuj z tych plików kod, który wejdzie w skład tych **partials**. Następnie utwórz **partials/back.ejs**, który:

- będzie agregował fragment kodu **Back to Home**.

Oraz stwórz **partials/login.ejs**, który:

- będzie agregował fragment kodu **Log in**.

Następnie, utwórz **views/book-details.ejs**, który:

- będzie korzystał z arkusza styli **styles.css** z katalogu **public**,
- będzie miał dynamicznie ładowany tytuł strony poprzez zmienną **title** przekazywaną w kontrolerze do widoku,
- w **<h1>** wyświetli treść **„${book.title} by ${book.author}"**,
- w **<p>** wyświetli treść **„Year: ${book.year}**,
- w **<p>** wyświetli treść **„Status: XXX"**, gdzie **XXX** wynosi **Available** gdy **book.avaialable** jest **true** i w przypadku **false**, wynosi **Borrowed**,
- **<form>** o wartośći **action="<% bok.available ? '/books/borrow' + book.id : '/books/return/' + book.id %>"** i **method="POST"**,
- **<button>** o wartośći **type="submit"** i warunku ustawiającym przycisk na **disabled** gdy **!book.available**, o treści **Borrow Book**,
- **<button>** o wartośći **type="submit"** i warunku ustawiającym przycisk na **disabled** gdy **!book.available && didUserBorrowTheBook,** o treści **Return Book**.
- będzie **includować**, **partials/back.ejs**.

W kolejnym kroku, zaktualizuj plik **routes/book.js**:

- dodaj ustawienie **router**, aby używał kontrolera **bookController.getBookDetails**, dla metody **GET** oraz ścieżki **„/books/:id"**,
- dodaj ustawienie **router**, aby używał kontrolera **bookController.postBookBorrow**, dla metody **POST** oraz ścieżki **„/books/borrow/:id"**,
- dodaj ustawienie **router**, aby używał kontrolera **bookController.postBookReturn**, dla metody **POST** oraz ścieżki **„/books/return/:id"**,

- dodaj ustawienie **router**, aby używał kontrolera **bookController.getBookBorrowSuccess**, dla metody **GET** oraz ścieżki **„/books/borrow/success”**,
- dodaj ustawienie **router**, aby używał kontrolera **bookController.getBookReturnSuccess**, dla metody **GET** oraz ścieżki **„/books/return/success”**.

W następnym kroku, zaktualizuj plik **models/User.js**:

- dodaj publiczną metodę **borrowBook** przyjmującą jako argument **book**, która dodaje do **borrowedBooks** tej instancji obiektu **User**, nową wypożyczoną książkę,
- dodaj publiczną metodę **returnBook** przyjmującą jako argument **bookId**, która usuwa z **borrowedBooks** tej instancji obiektu **User**, wypożyczona książkę,
- dodaj publiczną metodę **findBorrowedBookById** przyjmująca jako argument **bookId**, która zwraca **true/false** w zależności czy książka o danym **id** znajduje się w **borrowedBooks** tej instancji obiektu **User**.

Oraz zaktualizuj plik **models/Book.js**:

- dodaj publiczną metodę **borrow**, która zmienia **available** tej instancji obiektu **Book** na **false**,
- dodaj publiczną metodę **return**, która zmienia **available** tej instancji obiektu **Book** na **true**.

Na koniec, zaktualizuje **controllers/book.js**:

- dodaj metodę **getBookDetails**, która pobiera szczegóły konkretnego użytkownika (z sesji) oraz **id** książki z przekazanego przez ścieżkę parametru a następnie renderuje widok szczegółów książki, przekazując informacje o książce i statusie jej wypożyczenia pod nazwami zmiennych **book** oraz **didUserBorrowTheBook**,
- dodaj metodę **postBookBorrow**, która sprawdza dostępność książki i czy użytkownik (po id z sesji) istnieje a następnie, jeżeli użytkownik istnieje aktualizuje dostępność książki (na **false**), dodaje ją do **borrowedBooks** użytkownika i przekierowuje na widok **success** (**„/books/borrow/success”**),
- dodaj metodę **getBookBorrowSucces**, która renderuje stronę sukcesu po pomyślnym wypożyczeniu książki, przekazując za pomocą zmiennej **message**, treść **„Book borrowed successfully”**,
- dodaj metodę **postBookReturn**, która sprawdza dostępność książki i czy użytkownik (po id z sesji) istnieje a następnie, jeżeli użytkownik istnieje aktualizuje dostępność książki (na **true**), usuwa ją z **borrowedBooks** użytkownika i przekierowuje na widok **success** (**„/books/return/success”**),
- dodaj metodę **getBookReturnSuccess**, która renderuje stronę sukcesu po pomyślnym oddaniu książki, przekazując za pomocą zmiennej **message**, treść **„Book returned successfully”**,

PAMIĘTAJ, ZE WSZYSTKIE POTRZEBNE DO ZREAZLIZOWANIA ZADAN METODY DOTYCZACE KSIAZKI I UZYTKOWNIKA MASZ JUŻ W ICH MODELACH. WYKORZYSTAJ JE W KONTROLERACH/KONTROLERZE.