Warsztaty IV
Paczkolab

v.1.4

Coders Lab
SZKOŁA PROGRAMOWANIA

Cel warsztatów

- Celem warsztatów jest napisanie backendu aplikacji do nadawania paczek.
- Zadaniem tego kodu jest połącznie plików wyglądu frontend z bazą danych wykorzystując API REST oraz wzorca Active Record.
- Pliki z wyglądem aplikacji frontend są już gotowe i znajdują się w repozytorium do warsztatów.

- Projekt ma być napisany obiektowo.
- Do każdej z klas mają być napisane testy.
- Projekt jest grupowy (zespoły dwuosobowe).
- Osoby równolegle piszą klasy i testy do tych klas a kod wysyłają na zdalne repozytorium tym samym synchronizując swoją pracę.

Strony wyglądu frontend

Główna **/index.html**

Zawiera listę linków do stron aplikacji, które służą do realizacji wszystkich funkcji.

Podstrony zawierające listy:

- użytkowników /template/view/user.html
- adresów /template/view/address.html
- paczek /template/view/parcel.html
- rozmiarów /template/view/size.html

Pliki html zawierają formularze z danymi poszczególnych wpisów w bazie umożliwiające również aktualizację i usunięcie.

Podstrony umożliwiające dodanie wpisów:

- użytkowników /template/add/user.html
- adresów /template/add/address.html
- paczek /template/add/parcel.html
- rozmiarów /template/add/size.html

W/w strony zawierają formularze umożliwiające dodanie nowych wpisów do bazy danych.

Pliki z katalogu /template znajdują się w udostępnionym repozytorium do warsztatu.

Opis działania frontendu znajduje się na następnym slajdzie.

Opis działania frontend’u

Do każdego pliku HTML dołączony jest plik z kodem w java script.

Pliki te mieszczą się w katalogu /template/js

Głównym zadaniem dołączonego kodu jest przechwycenie zdarzenia z formularza i wysłanie danych pod odpowiedni adres REST zwany endpointem.

Lista adresów wraz z opisem znajduje się w dalszej części prezentacji.

Pliki java script wykorzystują asynchroniczną komunikację z serwerem AJAX czyli bez przeładowania strony.

Adresy, na które są wysyłane zapytania są wpisane na stałe w kodzie co oznacza, że należy ściśle się ich trzymać projektując serwer REST.

Wszystkie prowadzą do jednego pliku router.php.

Kod PHP zawarty w nim musi odpowiednio interpretować zapytania AJAX i zgodnie z opisem zwracać wartości.

Wykonanie tego pliku jest jednym z zadań warsztatu.

Opis działania aplikacji w PHP

Projekt bazy danych zawiera 4 tabele o nazwach:

- User
- Address
- Parcel
- Size

Ich opis znajduje się na następnym slajdzie.

W katalogu /class znajdują się pliki PHP odpowiedzialne za komunikację z bazą danych zgodnie z zasadą Active Record.

Każdy z tych plików implementuje interfejs zawarty w pliku /class/interface/action.php

Dzięki temu będziemy mogli w ten sam sposób odnosić się do klas pisząc kod w pliku routingu.

Pliki komunikujące się z bazą powinny zawierać:

- ➢ Funkcje SET i GET do wszystkich atrybutów (do id tylko GET).
- ➢ Konstruktor nastawiający id na -1, a resztę danych zerujący.
- ➢ Funkcję load(id).
- ➢ Funkcję save().
- ➢ Funkcję update().
- ➢ Funkcję delete().
- ➢ Funkcję statyczną loadAll().

Opis bazy danych

Adresy
Każdy adres posiada miasto, kod, ulicę oraz numer mieszkania.

Użytkownicy
Mają przypisany adres oraz pola: imię, nazwisko, ilość kredytów, hasło.

Paczki
Posiadają przypisanego użytkownika, adres oraz wielkość.

Wielkość paczki
Jest znakowym określeniem wielkości np. XL a także posiada swoją cenę (odpowiada kredytom użytkownika).

Opis działania aplikacji w PHP

Działanie poszczególnych metod jest zgodne z Active Record czyli:

- **load**(id) – wczytuje z bazy pojedynczy rząd
- **update**() – zapisuje obiekt do tabeli (jako zmiany wcześniej istniejącego rzędu tej tabeli)
- **save**() – zapisuje obiekt do tabeli (jako nowy rząd)
- **delete**() – usuwa obiekt z tabeli (czyli usuwa rząd o id takim samym jak obiekt).

Jako statyczną metodę dodamy wczytanie wszystkich rzędów z bazy danych o nazwie **loadAll**().

Posłuży ona do wyświetlenia wszystkich wpisów danego typu (danej klasy) na frontendzie.

Poza interfejsem, klasy dziedziczą po klasie abstrakcyjnej /class/abstract/database.php.

Statyczne pole $connection, w niej zawarte, zawiera połączenie do bazy danych.

Pozwoli to w jednym miejscu tworzyć połączenie i przekazać je statycznie do wszystkich klas.

Połączenie jest tworzone w pliku /config.php.

W nim również znajdują się dane dostępowe, które należy uzupełnić.

Opis działania serwera REST

Plik router.php ma pełnić zadanie serwera REST.

Oznacza to, że powinien obsłużyć zapytania wysyłane do niego na odpowiednie endpointy.

Opis akcji i adresów url (endpointów) do nich znajduje się w dalszej części prezentacji w slajdach z tytułem „Tabela endpointów”.

Dla każdej klasy reprezentującej tabelę w bazie danych adresy te są inne ale akcje realizują identyczne zadania jak np. wyświetlenie wszystkich wpisów w bazie lub dodanie nowego.

Kod w pliku routingu korzysta z klas reprezentujących tabele i na ich podstawie przygotowuje dane dla plików frontendu np. wyświetla json ze wszystkimi rzędami.

Aby wszystko działało prawidłowo bardzo ważne jest **zachowanie konwencji nazewniczej** w klasach.

Przykładowo kolumny z tabeli adresów powinny się nazywać city, code, street, flat gdyż kod java script odnosi się do nich właśnie po takich nazwach.

```
(address.city + ' ' + address.code + ' ' +  address.street + ' ' +
                                    address.flat);
```

Przykład współdziałania serwera REST i frontendu

Wyświetlenie listy wszystkich użytkowników

Użytkownik otwiera stronę główną index.html i wybiera LISTA UŻYTKOWNIKÓW

Zostanie przekierowany do pliku frontendu /template/view/user.html, który zawiera pustą tabelę.

Tabela w momencie załadowania strony zostanie uzupełniona danymi z bazy dzięki asynchronicznemu zapytaniu. Wykonuje to funkcja **loadUserView**() zawarta w pliku /template/js/view/viewUser.js, która wysyła zapytanie metodą GET na adres router.php/user/

W pliku router.php znajduje się kod, który wywoła statyczną metodę na klasie reprezentującą użytkownika, o nazwie **loadAll()** wczytująca wszystkie rzędy z bazy użytkowników.

Dane te po wczytaniu zostaną zamienione do formatu json i wyświetlone.

Kod ten w formacie json zostanie odebrany w odpowiedzi na zapytanie AJAX a następnie umieszczony w tabeli w kolejnych wierszach.

Przykład współdziałania serwera REST i frontendu

Edycja danych użytkownika

Użytkownik otwiera stronę główną index.html i wybiera LISTA UŻYTKOWNIKÓW
Spośród wszystkich wpisów wybiera jeden do edycji.
Po modyfikacji danych znajdujących się w formularza zostają one zapisane w bazie danych. (Update istniejącego wpisu)

W momencie zatwierdzenia wysłania danych w formularzu kod js w pliku /template/js/view/viewUser.js „przechwytuje” zdarzenie i za pomocą Ajax wysyła zapytanie metodą PUT na adres router.php/user/id (zamiast id podstawiany jest numer id edytowanego użytkownika).

W pliku **router.php** znajduje się kod, który z adresu wywołania zapytania pobierze nazwę **klasy** oraz **id**.

Dzięki metodzie wywołania (PUT) będzie wiadomo, że wpis w tabeli o nazwie **klasy** i o danym **id** należy zaktualizować nowymi danymi zawartymi w zapytaniu zamiast np. wyświetlić (metoda GET) lub usunąć (metoda DELETE).

Przygotowanie

Zadanie 1. Przygotowanie

- Dobierzcie się w pary.
- Osoby w parze piszą na zmianę kod i testy do kodu.
- W repozytorium do warsztatów znajdują się pliki z wyglądem stron w HTML wraz z plikami Java Scipt oraz pliki PHP.
- Zapoznajcie się z logiką działania frontendu biorąc pod uwagę przykład wyświetlenia wszystkich użytkowników opisany w poprzednim slajdzie.
- Zapoznajcie się z plikami PHP w szczególności interfejsem, klasą abstrakcyjną i configiem.
- W parze załóżcie jedno repozytorium na GitHub (musi się znajdować na jednym z kont, bez znaczenia na którym).
- Dodajcie pliki z otrzymanego repozytorium (frontend) do nowo utworzonego przez Was.
- Podepnijcie swoje nowe projekty do repozytorium i zobaczcie, czy działa (np. przez dodanie pliku readme na GitHubie i ściągnięciu go na oba komputery).

Przygotowanie

Zadanie 2. Połączenie do bazy danych

- Na swoich komputerach uzupełnijcie plik z danymi do połączenia się z bazą danych.
- Dane do połączenia każdy powinien trzymać w osobnym pliku config.php, który nie może się znajdować w repozytorium.
- Użyjcie do tego .gitignore.
- Upewnijcie się że połączenie jest nawiązane i nie zwraca błędów.

Przed przystąpieniem do wykonywania kolejnych zadań upewnijcie się, że rozumiecie działanie frontendu a także interfejsu i klasy abstrakcyjnej.

Klasa Address

Zadanie 3a. Klasa Address (pierwsza osoba)

- Dopisz do pliku z klasą **Address** (w katalogu /class/Address.php) prywatne właściwości:
  - id
  - city
  - code
  - street
  - flat

- Napisz setery i getery do nich (do id tylko geter).
- Przygotuj tabelę w bazie danych z powyższymi polami.
- Napisz kod realizujący zadania zgodne z metodami z interfejsu.
- Napisz kod w pliku routingu, który poprawnie przetworzy zapytania z tabeli endpointów korzystając m.in. z metod wcześniej napisanych.

Zadanie 3b. Testy do klasy Address (druga osoba)

- Przed przystąpieniem do napisania testów napisz kod w pliku routingu, który poprawnie przetworzy zapytania z tabeli endpointów i wywoła metody z plików klas.
- Wykorzystaj tzw. zaślepki do metod z interfejsu.
- Stwórz w katalogu /tests plik, w którym dodasz wszystkie testy do klasy **Address**.
- Testy mają sprawdzić m.in:

a) Czy można zapisać nowy adres.
b) Czy można uaktualnić adres.
c) Czy można usunąć adres, który istnieje w bazie danych.
d) Czy można pobrać adres, który istnieje w bazie.
e) Czy można pobrać wszystkie adresy z bazy za pomocą statycznej metody loadAll.

- Dodaj własne testy nie opisane powyżej.

Tabela endpointów klasa Address

Metoda HTTP	ADRES	CO ROBI?
GET	router.php/address/	Wyświetla listę wszystkich adresów w formacie json.
POST	router.php/address/	Tworzy nowy adres na podstawie danych przekazanych z formularza i zapisuje do bazy danych.
GET	router.php/address/id	Wyświetla informacje o adresie o podanym id.
PUT	router.php/address/id	Zmienia informacje o adresie o podanym id.
DELETE	router.php/address/id	Usuwa adres o podanym id z bazy danych.

Synchronizacja

Zadanie 4. Synchronizacja

- Wyślijcie swój kod do wspólnego repozytorium.
- W przypadku wystąpienia konfliktu wspólnie ustalcie sposób rozwiązania i ponownie wyślijcie kod.
- Pamiętajcie o modyfikacji swoich baz danych np. poprzez dodanie do repozytorium pliku dump.sql.
- Jeżeli ukończysz swoją klasę lub testy wcześniej niż druga osoba, to pomóż jej w pracy.
- Uruchomcie testy i sprawdźcie czy przechodzą. Jeśli nie to wspólnie przeanalizujcie rozwiązania programistyczne jakie zastosowaliście i usuńcie błędy w kodzie lub w testach.
- Następnie do repozytorium dodajcie fix (kod naprawiający ten błąd).
- Na końcu sprawdźcie czy działają strony z frontendu dotyczące adresów.

Klasa User

Zadanie 5a. Klasa użytkownika (druga osoba)

- Dopisz do klasy reprezentującej użytkownika o nazwie **User** (w katalogu /class/User.php) prywatne właściwości:
  - id
  - address_id
  - name
  - surname
  - credits
  - password
- Napisz setery i getery do nich (do id tylko geter).
- Przygotuj tabelę w bazie danych z powyższymi polami. Pole address_id jest kluczem zewnętrznym do tabeli Address kolumny id.
- Napisz kod realizujący zadania zgodne z do metod z interfejsu.
- Napisz kod w pliku routingu, który poprawnie przetworzy zapytania z tabeli endpointów korzystając m.in. z metod interfejsu.

Zadanie 5b. Testy do klasy użytkownika (pierwsza osoba)

- Przed przystąpieniem do napisania testów napisz kod w pliku routingu, który poprawnie przetworzy zapytania URL (tabela endpointów).
- Wywoła metody z plików klas.
- Wykorzystaj tzw. zaślepki do metod z interfejsu.
- Stwórz w katalogu /tests plik, w którym dodasz wszystkie testy do klasy **User**.
- Testy mają sprawdzić m.in:

a) Czy udało się zapisać nowego użytkownika do bazy danych.
b) Czy można uaktualnić dane użytkownika.
c) Czy można usunąć użytkownika, który istnieje w bazie danych.
d) Czy można pobrać dane użytkownika, który istnieje w bazie.
e) Czy można pobrać wszystkich użytkowników za pomocą statycznej metody loadAll.

- Dodaj własne testy nie opisane powyżej.

Tabela endpointów klasa User

Metoda HTTP	ADRES	CO ROBI?
GET	router.php/user/	Wyświetla listę wszystkich użytkowników w formacie json.
POST	router.php/user/	Tworzy nową książkę na podstawie danych przekazanych z formularza i zapisuje ją do bazy danych.
GET	router.php/user/id	Wyświetla informacje o użytkowniku o podanym id.
PUT	router.php/user/id	Zmienia informacje o użytkowniku o podanym id.
DELETE	router.php/user/id	Usuwa użytkownika o podanym id z bazy danych.

Synchronizacja

Zadanie 6. Synchronizacja

- Wyślijcie swój kod do wspólnego repozytorium.
- W przypadku wystąpienia konfliktu wspólnie ustalcie sposób rozwiązania i ponownie wyślijcie kod.
- Pamiętajcie o modyfikacji swoich baz danych np. poprzez dodanie do repozytorium pliku dump.sql.
- Jeżeli ukończysz swoją klasę lub testy wcześniej niż druga osoba, to pomóż jej w pracy.
- Uruchomcie testy i sprawdźcie czy przechodzą. Jeśli nie to wspólnie przeanalizujcie rozwiązania programistyczne jakie zastosowaliście i usuńcie błędy w kodzie lub w testach.
- Następnie do repozytorium dodajcie fix (kod naprawiający ten błąd).
- Na końcu sprawdźcie czy działają strony z frontendu dotyczące adresów.

Klasa Parcel

Zadanie 7a. Klasa Parcel (pierwsza osoba)

- Dopisz do klasy **Parcel** prywatne właściwości:
  - id
  - address_id
  - size_id
  - user_id

- Napisz setery i getery do nich (do id tylko geter).
- Przygotuj tabelę w bazie danych z powyższymi polami. Pola address_id, size_id oraz user_id są kluczami zewnętrznymi.
- Napisz kod realizujący zadania zgodne z założeniami active record do klasy.
- Napisz kod w pliku routingu, który poprawnie przetworzy zapytania z tabeli endpointów korzystając m.in. z metod interfejsu.

Zadanie 7b. Testy do klasy Parcel (druga osoba)

- Przed przystąpieniem do napisania testów napisz kod w pliku routingu, który wywoła metody z plików klas.
- Wykorzystaj tzw. zaślepki do metod z interfejsu.
- Stwórz odpowiedni plik z testami.
- Napisz testy, które sprawdzą zapis, update i wczytanie wszystkich wpisów z bazy danych dotyczących paczek czyli dokładnie tak jak poprzednio.
- Rozwiń testy o sprawdzenie relacji czyli:

a) Czy możliwy jest zapis podając nie istniejące id adresu, id rozmiaru i/lub id użytkownika.
b) Czy można usunąć adres, rozmiar i/lub użytkownika z bazy danych jeśli istnieje wpis o paczce

Po skończeniu wykonajcie procedurę synchronizacji.

Tabela endpointów klasa Parcel

Metoda HTTP	ADRES	CO ROBI?
GET	router.php/parcel/	Wyświetla listę wszystkich paczek w formacie json.
POST	router.php/parcel/	Tworzy nową paczkę na podstawie danych przekazanych z formularza i zapisuje do bazy danych.
GET	router.php/parcel/id	Wyświetla informacje o paczce o podanym id.
PUT	router.php/parcel/id	Zmienia informacje o paczce o podanym id.
DELETE	router.php/parcel/id	Usuwa paczkę o podanym id z bazy danych.

Klasa Size

Zadanie 8a. Klasa Size (druga osoba)

- Dopisz do klasy **Size** prywatne właściwości:
  - id
  - size
  - price

- Napisz setery i getery do nich (do id tylko geter).
- Przygotuj tabelę w bazie danych z powyższymi polami.
- Napisz kod realizujący zadania zgodne z założeniami active record do klasy.
- Napisz kod w pliku routingu, który poprawnie przetworzy zapytania z tabeli endpointów.

Zadanie 8a. Klasa Size (pierwsza osoba)

- Przed przystąpieniem do napisania testów napisz kod w pliku routingu, który wywoła metody z plików klas.
- Wykorzystaj tzw. zaślepki do metod z interfejsu.

- Napisz testy, które sprawdzą:

a) Zapis nowego rozmiaru.
b) Uaktualnienie danych rozmiaru.
c) Usunięcie rozmiaru z bazy danych.
d) Usunięcie rozmiaru w przypadku gdy znajduje się powiązana z nią paczka.

Po skończeniu wykonajcie procedurę synchronizacji

Tabela endpointów klasa Size

Metoda HTTP	ADRES	CO ROBI?
GET	router.php/size/	Wyświetla listę wszystkich rozmiarów w formacie json.
POST	router.php/size/	Tworzy nowy rozmiar na podstawie danych przekazanych z formularza i zapisuje do bazy danych.
GET	router.php/size/id	Wyświetla informacje o rozmiarze o podanym id.
PUT	router.php/size/id	Zmienia informacje o rozmiarze o podanym id.
DELETE	router.php/size/id	Usuwa rozmiar o podanym id z bazy danych.

Epilog

- Pod koniec pamiętajcie o tym, żeby osoba, która nie ma repozytorium, zrobiła fork na swoje konto GitHuba.
- Dzięki temu będzie miała takie samo repozytorium u siebie.
- Osoba, która to zrobi, musi też pamiętać o zmianie adresu **origin** w swoim repozytorium, dzięki temu będzie commitować do repozytorium swojego, a nie do koleżanki lub kolegi.
- W razie problemów poproście wykładowcę.

- Warsztat można już samodzielnie rozwinąć o dodatkowe funkcje jak np.:

- Logowanie z użyciem hasła. Użytkownik może wtedy pracować tylko na własnych danych.
- Płatności kredytami za nadanie paczki. Użytkownik może nadać paczkę jeśli ma wystarczającą ilość kredytów. Po nadaniu paczki ilość jest automatycznie odejmowana z ogólnej puli.
- Dodać obsługę błędów z użyciem kodów odpowiedzi HTTP oraz prezentacji ich na frontendzie.