Marta Jaworska

EFA-DU1

L1

FD 143920

email: maartajaworska@gmail.com

Nazwa i link do repozytorium: Caarmell/Fundacja

Adres do aplikacji: <http://mjaw.ayz.pl/>

Załączniki: baza danych, kod aplikacji

Data: 18.10.2018 r.

Przedmiot: Aplikacje Internetowe

Rodzaj zajęć: laboratorium

**Fundacja dla zwierząt**

***Spis treści***

[**Streszczenie:**](#_qz88lusrzd7b) **3**

[**Autoryzacja:**](#_raixius5wxmn) **3**

[Logowanie do bazy danych:](#_2jw5di6hy3cx) 3

[Logowanie do aplikacji:](#_zaajwteulxpj) 3

[**Założenia i technologie:**](#_ncmbux4s1bvc) **4**

[Założenia funkcjonalne:](#_qa3mukk6dnrj) 4

[Założenia niefunkcjonalne:](#_si2yam2o5ju8) 5

[Wykorzystane technologie:](#_el8oco35jo9f) 5

[**Diagramy UML:**](#_fhqyqacjupwj) **5**

[Diagram przypadków użycia dla administratora.](#_is0zt99bjza9) 5

[Diagram przypadków użycia dla niezarejestrowanego użytkownika.](#_kpw948yk2rm3) 6

[Diagram przypadków użycia dla zarejestrowanego użytkownika.](#_ci46dnksiyr1) 6

[Diagram przypadków użycia dla pracownika.](#_kxx0h2by1kmi) 7

[**Front-end i back-end aplikacji:**](#_7qut2do3t054) **7**

[Front-end:](#_ro9602byk5vn) 7

[Back-end:](#_2slpyp6up5ht) 8

[**Harmonogram prac:**](#_m71g05jangey) **9**

[**Model danych:**](#_lpu0pd3u39nd) **11**

[**Struktura aplikacji:**](#_at1mqspn8ju5) **11**

[Model:](#_14o1p04t5p63) 11

[Widok:](#_2oemha6fuh2o) 12

[Kontroler:](#_g2uqpcicnr5) 13

[**Opis funkcjonalności:**](#_nbvthqnefvl1) **14**

[**Unikalne linki:**](#_rodjxgoykxva) **25**

[**Problematyka:**](#_8m01itpc4e6c) **25**

### Streszczenie:

Wykonanie prostej aplikacji, która mogłaby posłużyć dla fundacji dla zwierząt. Dołączenie logowania, dzięki czemu każdy użytkownik mógłby posiadać konto. Konto dla pracowników umożliwiłoby wprowadzenie zmian, do których mają uprawnienia. Aplikacja ma za zadanie umożliwić komunikację między pracownikami, a osobami zainteresowanymi losem zwierząt. Wprowadzenie możliwości wstępnej rezerwacji zwierzaka mogłoby usprawnić pracę fundacji. Realizowaną aplikację można rozwijać o różnego rodzaju funkcjonalności. Z powodu licznych problemów nie udało się wykonać wszystkich zamierzonych funkcjonalności. Wprowadzenie wyszukiwarki umożliwiającej szybsze znalezienie pupila, a także podział na gatunki zwierząt mogłyby usprawnić znajdowanie konkretnego podopiecznego fundacji.

### Autoryzacja:

##### Logowanie do bazy danych:

W celu uzyskania dostępu do bazy danych proszę o kontakt z autorem.

##### Logowanie do aplikacji:

W poniższej tabeli przedstawione zostały loginy oraz hasła, a także rodzaje kont oraz uprawnienia. Uprawnienia zostały wypełnione tylko dla konta pracownik, ponieważ według założeń projektu tylko pracownik ma mieć różne uprawnienia. Należy pamiętać, że konto administratora służy tylko do dodawania nowych pracowników, edycji ich uprawnień oraz usuwanie pracowników. Z konta administratora nie ma możliwości wprowadzenia zmian dotyczących zwierząt. Nie ma też możliwości dodanie konta administratora z poziomu aplikacji. W celu dodania konta administratora należy wpisać je bezpośrednio do bazy danych. Istotnym elementem jest fakt, że w przypadku dodawania użytkowników z poziomu bazy danych należy skorzystać z generatora kodowania za pomocą algorytmu bcrypt. W przypadku, gdy wpisze się niezakodowane hasło do bazy danych, nie będzie możliwości zalogowania się.

*Uprawnienia:*

**1** - dodawanie zwierząt, **2** - edycja danych zwierząt ( w tym przenoszenie do adoptowanych/zmarłych) **3** - usuwanie zwierząt **4** - edycja dodatkowych informacji (dodawanie, usuwanie, edycja wpisów, dodawanie, usuwanie zdjęć, w tym zmiana na zdjęcie główne) **5** - dodawanie długów zwierząt

**6** - edycja długów zwierząt **7** - akceptacja rezerwacji

Rezerwacje nie zostały zrealizowane ze względu na liczne błędy, jakie wystąpiły podczas tworzenia projektu, dlatego uprawnienie to nie pokazuje nic.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NR** | **LOGIN (EMAIL)** | **HASŁO** | **TYP KONTA** | **UPRAWNIENIA** |
| 1 | admin@admin.com | admin | administrator | BRAK |
| 2 | test@test.com | test1 | pracownik | 1,2,3,4,5,6,7 |
| 3 | anna@anna.com | test1 | pracownik | 4 |
| 4 | kamil@kamil.com | test1 | pracownik | 2, 4, 6 |
| 5 | sara@sara.com | test1 | gość | BRAK |

### Założenia i technologie:

##### Założenia funkcjonalne:

ADMINISTRATOR

* dodawanie nowych pracowników
* usuwanie pracowników
* nadawanie uprawnień pracownikom

PRACOWNICY

* logowanie
* edycja danych osobowych
* dodawanie nowych podopiecznych
* edycja podopiecznych
* usuwanie podopiecznych / przeniesienie do zakładki “adoptowane” lub “tęczowy most” (zmarłe)
* akceptacja rezerwacji pupila
* dodawanie, edycja, usuwanie newsów
* dodawanie, usuwanie zdjęć
* zmiana głównego zdjęcia
* dodawanie, edycja długów pupili

ZAREJESTROWANI UŻYTKOWNICY

* logowanie
* edycja danych osobowych
* rezerwacja pupila
* dodawanie pupili do ulubionych - łatwiejsze przeglądanie pupili
* przeglądanie zarezerwowanego pupila, ulubionych
* wyświetlanie wszystkich informacji o podopiecznych (zdjęcia, tekst, w tym przebyte choroby, licznik pieniążków na podopiecznego), przy każdym pupilu opcja “dodaj do ulubionych” bądź gdy już taki pupil jest w ulubionych to “usuń z ulubionych”

GOŚCIE

* wyświetlanie informacji o podopiecznych (zdjęcia, tekst, w tym przebyte choroby, licznik pieniążków na podopiecznego)
* rejestracja

##### Założenia niefunkcjonalne:

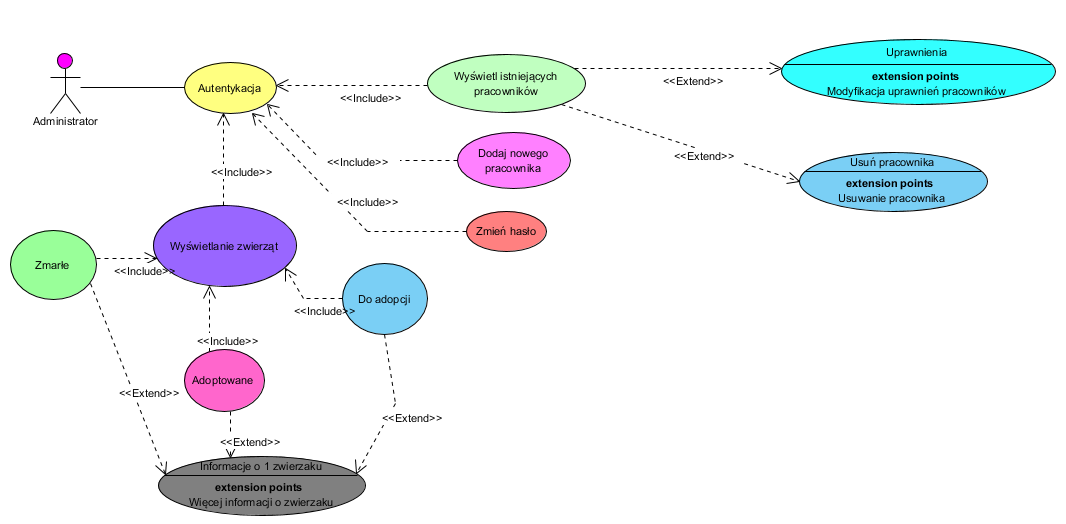
* szybkość - szybkie zmiany na serwerze, akceptowalne dodawanie zdjęć do 10 sekund, wczytywanie do 5 sekund,
* prostota w korzystaniu - intuicyjny widok, z którym poradzi sobie każda osoba,
* rozmiar - aplikacja, która może być zainstalowana na serwerze o najmniejszej pojemności,
* niezawodność - prawdopodobieństwo błędnego wykonania 0%, wszystkie parametry podczas testów zostały sprawdzone i nie pojawiły się żadne błędy,
* przenośność - projekt może być wykorzystany na każdej platformie, co oznacza, że 100% kodu jest niezależne od platformy docelowej. Należy odpowiednio skonfigurować framework, aby móc wprowadzać zmiany w projekcie.

##### Wykorzystane technologie:

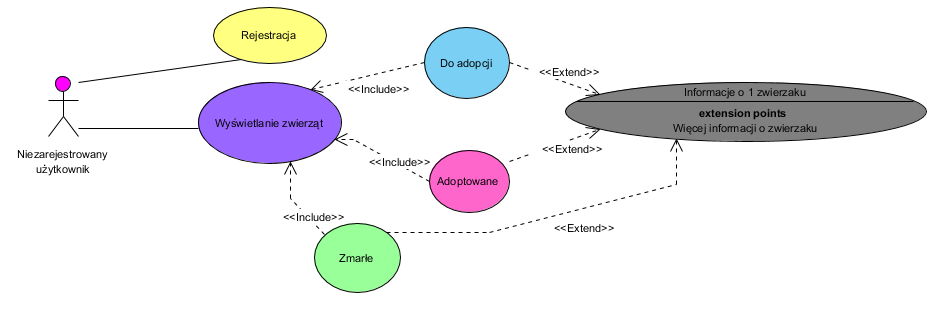
* baza danych MySQL
* języki: PHP, JavaScript, CSS, HTML
* frameworki: Bootstrap, Symfony
* system szablonów Twig

### Diagramy UML:

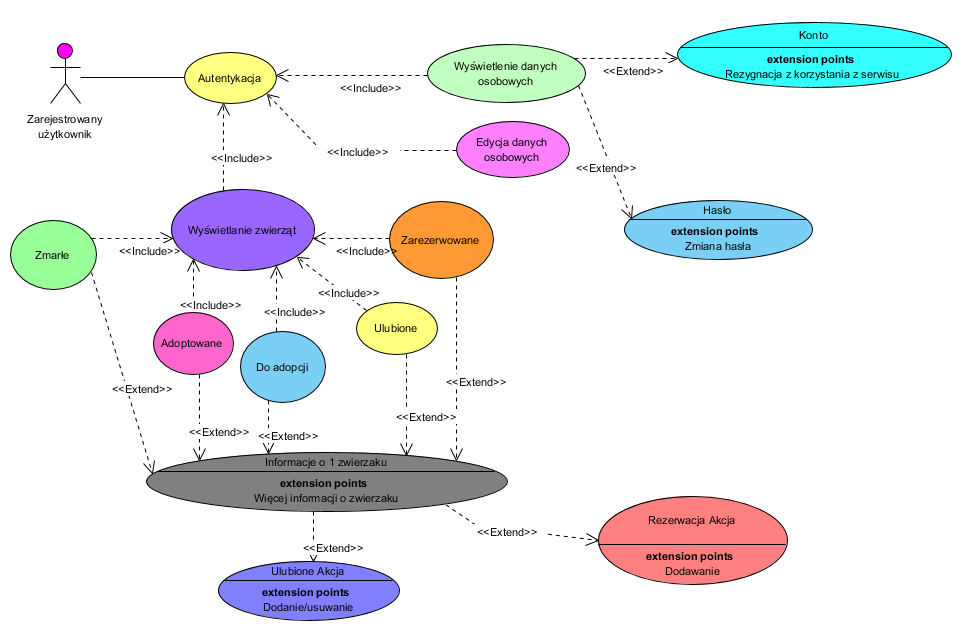
##### Diagram przypadków użycia dla administratora.



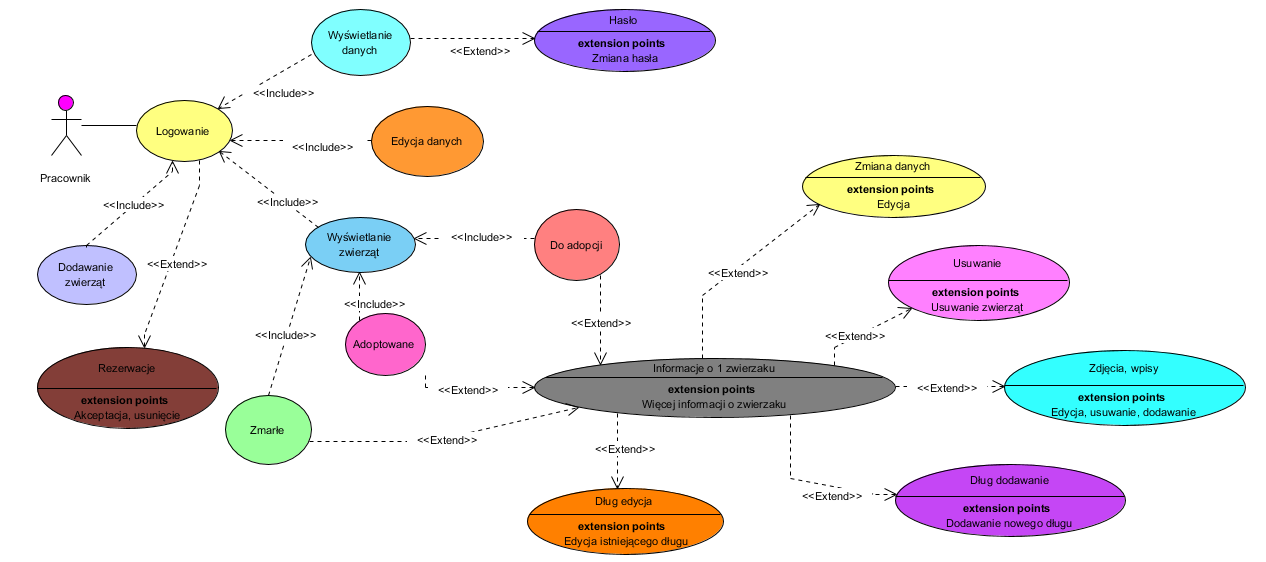
##### Diagram przypadków użycia dla niezarejestrowanego użytkownika.



##### Diagram przypadków użycia dla zarejestrowanego użytkownika.



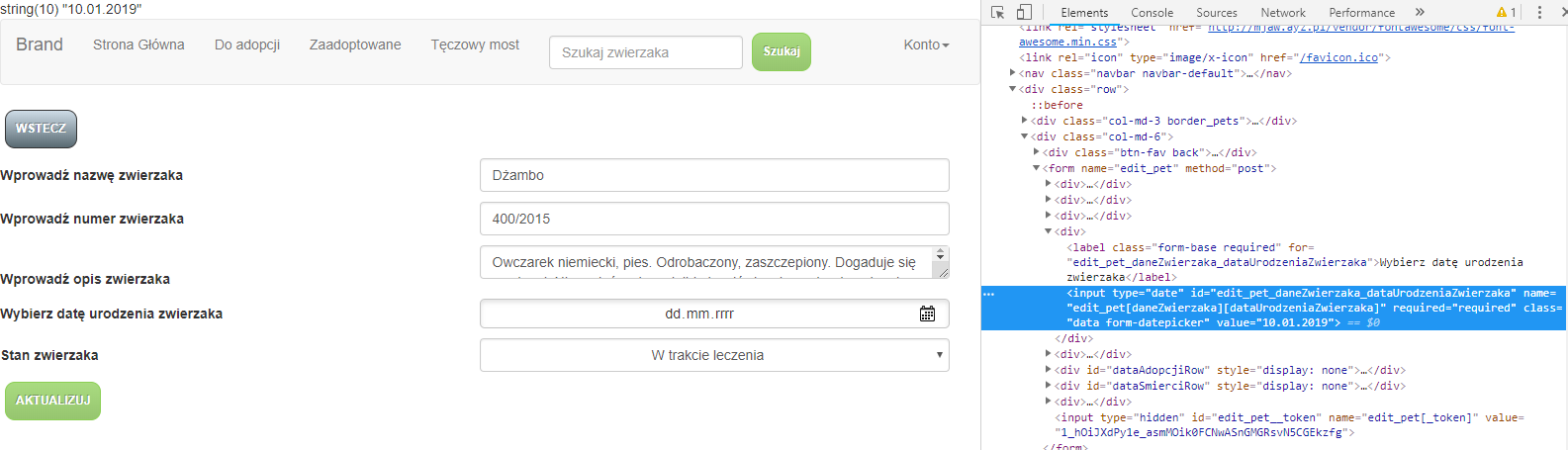
##### Diagram przypadków użycia dla pracownika.



### Front-end i back-end aplikacji:

##### Front-end:

Ze względu na rozbudowaną aplikację oraz pewne problemy front-end nie został rozszerzony o framework Angular. We front-endzie wykorzystano framework Bootstrap, który nadał wyglądy formularzom. Dodatkowo w przypadku uzupełniania formularza dodaj zwierzaka lub formularza edytuj zwierzaka został wykorzystany język JavaScript, który w zależności, jaki jest wybrany stan zwierzaka (do adopcji/w trakcie leczenia czy adoptowany, czy odszedł) wyświetla lub nie dodatkowe daty adopcji oraz śmierci pupila. Pojawił się problem z wyświetlaniem daty, ponieważ wykorzystano DatePicker. Problem polegał na tym, że w inpucie odpowiedzialnym za datę była ustawiona prawidłowa data, natomiast wyświetlało się jako dd.mm.rrrr, co w dalszym etapie generowało błąd walidacyjny “wypełnij to pole”. Takie komunikaty wyświetlały się tylko w tych formularzach, gdzie była możliwość edytowania zwierzaka, czy też rezerwacji. W przypadku, gdy użytkownik wpisał datę w DatePicker’ze formularz był aktualizowany prawidłowo. Na poniższym zdjęciu przedstawiony jest występujący problem. U góry skorzystano również z funkcji var\_dump, aby sprawdzić, co dokładnie znajduje się w zmiennej oraz jakiego jest typu.



W celu rozwiązania tego problemu należało użyć daty w odpowiednim formacie. Format, który jest akceptowany przez DatePicker to “rrrr-mm-dd”, a w projekcie próbowano wpisywać w formacie “dd-mm-yyyy” lub “dd.mm.yyyy”.

Aby uniknąć pisania wielu takich samych plików z wyglądów postanowiono skorzystać z możliwości, jakie daje twig. Plik dotyczący wyświetlania więcej informacji dla zwierząt jest jeden, natomiast wewnątrz jego sprawdzane są wartości, jakie zostały ustawione w kontrolerze. Jeżeli zostanie wykryte konto pracownika to wyświetlane są inne informacje, niż w przypadku konta gościa. Konto osoby, która nie jest zalogowana oraz konto administratora wyświetlają podstawowe informacje o zwierzaku. Projekt został napisany, w taki sposób, aby jak największą liczbę kodu wykorzystać ponownie. Oznacza to, że edycja zwierzaka oraz dodanie zwierzaka, które zawierają takie same elementy z pewną różnicą, generowane są przy wykorzystaniu pewnej tej samej części szablonu formularza.

##### Back-end:

Back-end został w całości zrealizowany, w taki sposób, w jaki miał być. Postanowiono napisać wszystkie zamierzone funkcjonalności. Największym problemem okazało się logowanie. W Symfony do logowania należy wykorzystać specjalną klasę, którą najłatwiej jest stworzyć w linii poleceń poprzez wykorzystanie Doctrine. Dzięki odpowiedniemu tutorialowi udało się dodać możliwość logowania do aplikacji. Więcej informacji dotyczących logowania znajduje się w rozdziale dotyczącym problematyki, tam również podany jest adres strony, na której znajduje się przewodnik. Kontroler, który ma najwięcej zadań to PetController. W przypadku wyświetlania informacji dotyczących jednego konkretnego zwierzaka sprawdzane jest, czy został ustawiony adres email. Adres email pozwala na rozpoznanie, z jakiego rodzaju konta użytkownik próbuje wyświetlić informacje o pupilu. W przypadku, gdy jest to zwykły, niezalogowany użytkownik nie posiada on adresu email, dlatego inne wartości zwracane są do widoku. W widoku sprawdzane jest, które z wartości zostały ustawione i rozpoczyna się odpowiednie wyświetlanie dla każdego użytkownika. Innym dobrym przykładem jest ReservationController. Należało sprawdzić, czy dany zwierzak został już zarezerwowany lub czy osoba zalogowana ma zarezerwowanego pupila. Założenie było takie, aby każdy zalogowany użytkownik mógł mieć maksymalnie 1 zarezerwowanego pupila. Do pracownika odpowiedzialnego za akceptację rezerwacji trafia imię i nazwisko rezerwującego i po spełnieniu ustalonych warunków pracownik może potwierdzić lub usunąć rezerwację. W przypadku zarządzania rezerwacjami pracownik ma do wyboru stran: akceptuj lub usuń. Jeżeli wybraną opcją będzie akceptuj to dane w bazie zostaną zaktualizowane, w przypadku usuń taka rezerwacja zostanie usunięta.

Zrealizowane zostały podstawowe funkcjonalności takie jak wyświetlanie danych konta, zmiana hasła. Próbowano wprowadzić opcję usunięcia konta, jednak z pewnych powodów nie jest ona dobrze przeprowadzona. W momencie kliknięcia na przycisk Usuń konto pojawia się błąd. Po odświeżeniu strony błąd znika (czasami trzeba dwukrotnie odświeżyć). Zauważono, że w momencie usunięcia konta użytkownik nie jest wylogowywany stąd pojawił się taki problem. Próbowano wylogować użytkownika po usunięciu danych, co generowało podobny błąd. Wylogowanie użytkownika przed usunięciem danych nie pozwalało na wykreślenie użytkownika z bazy danych. Funkcjonalność ta została przesunięta na zrobienie w przypadku, gdy zostanie więcej czasu. Była to dodatkowa opcja.

### Harmonogram prac:

Przewidywany czas wykonania poszczególnych zadań został zaznaczony kolorem **niebieskim**, natomiast rzeczywisty czas zaznaczony jest kolorem **zielonym**.

* Utworzenie projektu w Symfony 20 min 20 min
* Utworzenie bazy danych 4 godziny 3 godziny
* Podpięcie bazy danych do projektu 1 godzina 30 godzin

*Opis został przedstawiony w problematyce, pojawiła się pewna niezgodność w wersji i dokumentacji Symfony.*

* Funkcjonalności:
  + Zwierzęta:
    - Wyświetlanie adoptowanych 1 godzina 1 godzina
    - Wyświetlanie zmarłych 1 godzina 30 minut
    - Wyświetlanie zwierzaków do 1 godzina 30 minut  
      adopcji
    - Wyświetlanie więcej informacji o 1 godzina 3 godziny  
      konkretnym pupilu

*Duża ilość informacji potrzebnych do wyświetlenia spowodowała zwiększenie ilości czasu.*

* + - Dodawanie zwierzaka 1 godzina 90 minut
    - Edycja zwierzaka 1 godzina 1 godzina
  + Pracownicy:
    - Dodawanie pracownika 1 godzina 1 godzina
    - Edycja uprawnień pracownika 30 minut 30 minut
    - Wyświetlanie wszystkich 30 minut 30 minut  
       pracowników
  + Ulubione zwierzaki:
    - Dodawanie pupila do ulubionych 1 godzina 30 minut
    - Usunięcie pupila z ulubionych 30 minut 15 minut
    - Wyświetlanie wszystkich 30 minut 15 minut  
      ulubionych zwierząt
  + Zarezerwowane zwierzaki:
    - Dodawanie rezerwacji 1 godzina 20 minut
    - Zatwierdzenie rezerwacji 1 godzina 20 minut
    - Przedłużanie czasu rezerwacji 30 minut 10 minut
    - Wyświetlanie wszystkich 1 godzina 30 minut  
       rezerwacji
    - Wyświetlanie konkretnej 30 minut 15 minut  
      rezerwacji
  + Konto:
    - Logowanie 30 minut 6 godzin

*Do logowania potrzebna była specjalna klasa user przez co trzeba było w znacznym stopniu przebudować projekt.*

* + - Rejestracja 30 minut 30 minut
    - Wyświetlanie danych 1 godzina 30 minut  
      osobowych
    - Edycja danych osobowych 1 godzina 30 minut
    - Zmiana hasła 30 minut 20 minut
    - Usunięcie konta 20 minut 40 minut

*Usunięcie konta nie udało się zrealizować tak, aby poprawnie działało, co zostało opisane w problematyce.*

* + Dodatkowe informacje o pupilach:
    - Dodawanie wpisu 1 godzina 30 minut
    - Edycja wpisu 1 godzina 30 minut
    - Usuwanie wpisu 30 minut 10 minut
  + Dodatkowe zdjęcia pupili:
    - Dodawanie zdjęcia 1 godzina 30 minut
    - Usuwanie zdjęcia 30 minut 10 minut
    - Zmiana zdjęcia na główne 30 minut 3 godziny
  + Dług pupila:
    - Dodawanie długu 1 godzina 30 minut
    - Edycja długu 30 minut 20 minut
* Wyglądy
  + Bootstrap 10 godzin 30 godzin

*Okazało się, że front-end jest bardzo pracochłonny, jak się nie ma wprawy i zmiany samych przycisków potrafiły zająć bardzo dużo czasu. Zbyt duża ilość kodu, a także zmęczenie wpływało na częstsze robienie błędów, przez co naprawa ich zajęła więcej czasu.*

* + Angular i JavaScript 10 godzin 1 godzina

*Frameworka Angular nie udało się wprowadzić ze względu na brak czasu. JavaScript został wprowadzony tylko do wyświetlania dodatkowych opcji w formularzu np. daty adopcji w przypadku wybrania stanu “adoptowany”.*

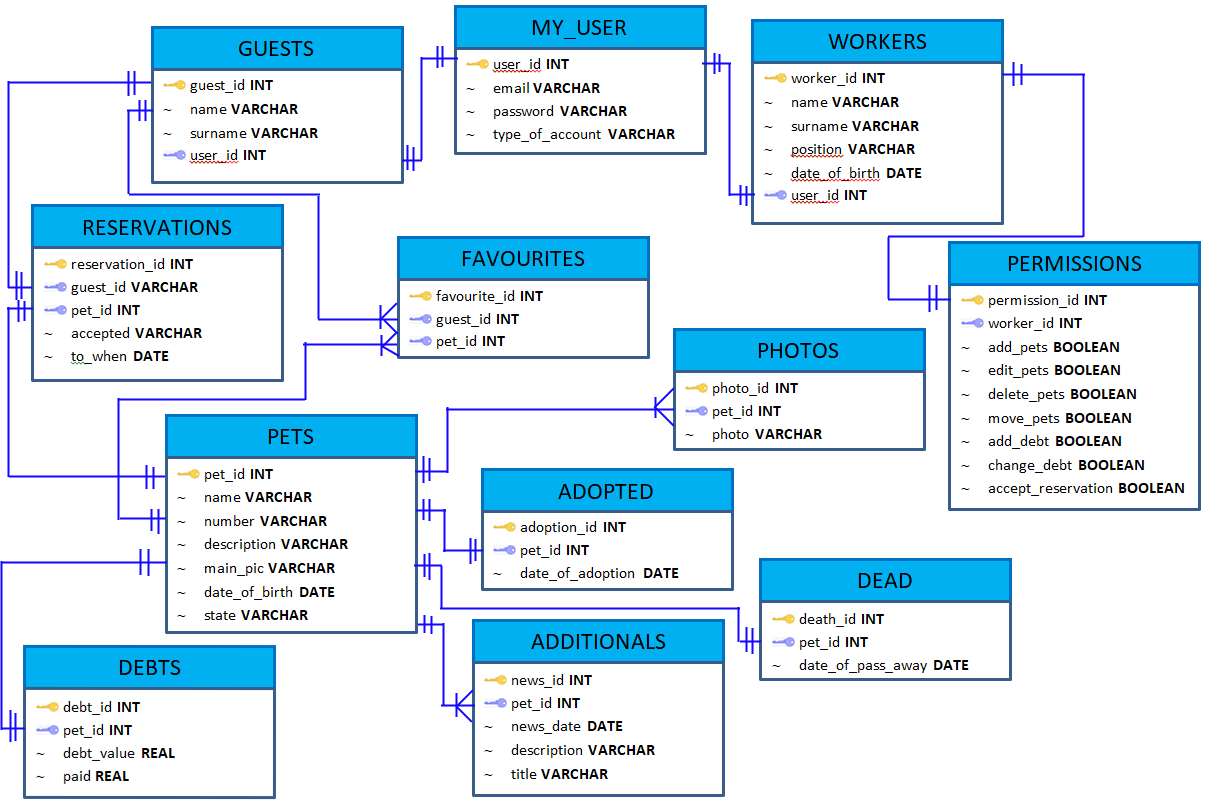
* Testy 4 godziny 3 godziny
* Dokumentacja 5 godzin 20 godzin

*Dokumentacja okazała się być bardzo pracochłonna, opis najważniejszych elementów, wstawienie odpowiednich listingów spowodowało, że zostało to znacznie wydłużone.*

Bardzo duża część funkcjonalności została zrealizowana. Nie udało się wprowadzić frameworka Angular. Znaczne opóźnienie w oddaniu projektu spowodowane było wersją Symfony, do której nie było dokumentacji. Próby naprawienia błędów a w końcu zainstalowanie najnowszej wersji zabrało ponad miesiąc czasu. Korzystanie z Symfony jest bardzo przyjemne, kod pisze się szybko tylko należy nauczyć się pewnych podstaw. Dodatkowym atutem są szablony Twig, które znacznie przyspieszyły napisanie plików widoku. Korzystając z czystego php projekt można byłoby zrealizować w krótszym czasie, jednak nie byłby on w żaden sposób zabezpieczony oraz nie można by było zobaczyć większości błędów. Postanowiono zostawić projekt z opcją Symfony debuger.

### Model danych:

Model danych przedstawiony został na poniższym diagramie ERD.



### Struktura aplikacji:

##### Model:

Model w aplikacji to klasy występujące w katalogu “src/Entity”, które zostały utworzone przez Doctrine. Klasy te zawierają wszystkie informacje dotyczące pól w bazie danych. Dodatkowo każda z klas zawiera metody gettery i settery, czyli funkcję ułatwiające pobieranie i ustawianie pewnych wartości.

* Additionals - klasa zawierająca informacje dodatkowe, czyli wszystkie wpisy, jakie zostały dodane do każdego zwierzaka.
* Adopted - klasa przechowująca id zwierząt, które zostały adoptowane oraz datę ich adopcji.
* Dead - klasa zawierająca id oraz datę śmierci zwierząt.
* Debts - klasa, która przechowuje wartość długu zwierzaka oraz kwotę, jaka została spłacona. W przypadku większych aplikacji mogłyby być podpięte płatności, dzięki czemu w łatwiejszy sposób każdy zwierzak miałby możliwość zbierania na siebie pieniędzy.
* Favourites - klasa zawierająca id pupila oraz zalogowanego użytkownika. Dzięki takiemu rozwiązaniu użytkownik w szybszy sposób może przeglądać ulubione zwierzaki.
* Guests - klasa zawierająca informacje dotyczące użytkownika, czyli jego imię i nazwisko. Można by było rozwinąć ją o pesel i wprowadzić dodatkową opcję ankiet przed adopcyjnych, które byłyby wyświetlane po zaakceptowaniu przez pracownika rezerwacji. Zadaniem użytkownika byłoby wypełnienie ankiety, ale nie podawanie danych osobowych. Dane osobowe wymagane do uzupełnienia w ankiecie przedadopcyjnej byłyby dołączane do pliku, a całość przesyłana na maila odpowiedniego pracownika. Po wysłaniu ankiety na koncie pracownika wyświetlana byłaby notatka z imieniem i nazwiskiem osoby, która wysłała ankietę.
* MyUser - klasa użytkownik służąca do logowania. Oprócz getterów oraz setterów zawiera inne funkcję niezbędne dla frameworka Symfony w celu przeprowadzenia logowania.
* Permissions - klasa przechowująca uprawnienia pracowników.
* Pets - klasa przechowująca dane o zwierzaku, datę urodzenia, nazwę, numer, nazwę zdjęcia głównego oraz stan zwierzaka (adoptowany, odszedł, do adopcji, do leczenia).
* Photos - klasa przechowująca id zwierzaka oraz nazwę dodatkowego zdjęcia.
* Reservations - klasa zawierająca rezerwacje, w tym id użytkownika, id zwierzaka, datę do kiedy jest rezerwacja ustawianą przez pracownika oraz stan (adoptowany i nieznany).
* Workers - klasa przechowująca dane pracowników, w tym imię, nazwisko, datę urodzenia. Klasa ta mogłaby przechowywać więcej informacji, dzięki czemu w łatwy sposób można by było zarządzać urlopami, a także wystawiać umowy dla pracowników. W klasie tej znajduje się również stanowisko, jakie obejmuje pracownik.

##### Widok:

Widok w Symfony to pliki z rozszerzeniem \*.twig, a także niektóre napisane w języku PHP, które dotyczą formularzy.

Formularze, które znajdują się w folderze “src/Form/Type”

* AddPetType - formularz do dodawania zwierząt.
* AddWorkerType - formularz do dodawania pracowników.
* ChangeDebtType - formularz, który daje możliwość dodawania kwoty do długu oraz dodawania spłaconej kwoty.
* ChangePasswordType - formularz do zmiany hasła.
* ChangePermissionType - formularz do zmiany uprawnień pracownika.
* DebtType - formularz do dodawania długu zwierzaka.
* EditPetType - formularz do edycji zwierzaka.
* EditReservationType - formularz do zmiany daty rezerwacji, w przypadku, gdy taka rezerwacja wygaśnie.
* EditUserType - formularz do zmiany danych pracownika i zarejestrowanego użytkownika.
* NewsType - formularz do dodawania i edycji wpisów.
* PermissionsType - formularz, który jest wykorzystywany w formularzu odpowiedzialnym za zmianę uprawnień oraz dodawania pracownika.
* PetInfoType - formularz wykorzystywany w formularzu edycji oraz dodawania zwierząt.
* PhotosType - formularz do dodawania dodatkowych zdjęć.
* ReservationType - formularz wyświetlający zarządzanie rezerwacją, która jest do zatwierdzenia.

Szablony widoków, które znajdują się w folderze “templates”.

* addDebt.html.twig - wygląd odpowiedzialny za dodanie długu zwierzaka.
* addNews.html.twig - wygląd odpowiedzialny za dodanie wpisu
* addPet.html.twig - dodawanie zwierząt.
* addPhotos.html.twig - dodawanie dodatkowych zdjęć.
* addWorker.html.twig - dodawanie pracownika.
* base.html.twig - szablon podstawowy, po którym wszystkie szablony dziedziczą.
* changePassword.html.twig - szablon do zmiany hasła.
* changePermissions.html.twig - zmiana uprawnień pracownika.
* editDebt.html.twig - edycja długu zwierzaka.-
* editPets.html.twig - edycja pupila.
* editReservation.html.twig - przedłużenie czasu rezerwacji, gdy jest do usunięcia.
* editUserData.html.twig - wyświetlanie danych, które są do edycji dla pracownika i użytkownika zarejestrowanego.
* mainpage.html.twig - strona główna z aktualnościami.
* myaccount.html.twig - strona dotycząca konta użytkownika.
* reservationState.html.twig - zarządzanie rezerwacją, która jest do zatwierdzenia.
* showAdoptedPets.html.twig - pokazanie zaadoptowanych zwierząt.
* showDeadPets.html.twig - pokazanie zmarłych zwierząt.
* showFavourite.html.twig - pokazanie zwierząt, które zostały dodane do ulubionych.
* showPet.html.twig - wyświetlenie informacji o konkretnym pupilu.
* showPets.html.twig - szablon odpowiedzialny za wyświetlenie wszystkich zwierząt dostępnych do adopcji.
* showReservation.html.twig - szablon pokazujący wszystkie rezerwacje pracownikowi, który ma odpowiednie uprawnienia.
* showReservationGuest.html.twig - pokazywanie swojej rezerwacji konkretnemu zalogowanemu użytkownikowi.
* showUserData.html.twig - wyświetlenie danych osobowych.
* showWorkers.html.twig - pokazanie wszystkich pracowników.

##### Kontroler:

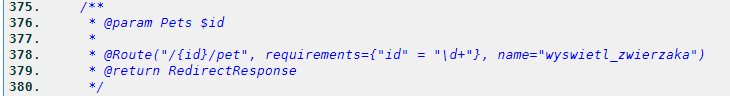
Wszystkie kontrolery przechowywane są w folderze “src/Controller”. Każdy z nich odpowiada za inną klasę z modelu. W każdym kontrolerze są funkcje, które są odpowiedzialne za odpowiednią akcję. Kontroler wywoływany jest po przejściu na dany adres. Przed każdą funkcją zapisuje się drogę, dzięki czemu Symfony wie, z której funkcji powinien skorzystać. Kontroler pobiera informację z bazy danych, a następnie przekazuje je do widoku. W przypadku formularzy odpowiada on za “nasłuchiwanie” i wykonanie odpowiedniej akcji po naciśnięcia przycisku.

* DebtController - kontroler obsługujący zdarzenia dotyczące długów, czyli dodawanie, wyświetlanie i edycja.
* FavouriteController - kontroler, który obsługuje ulubione zwierzaki, czyli dodawanie i usuwanie z listy ulubionych oraz wyświetlanie ulubionych zwierząt dla danego użytkownika.
* MyController - kontroler, który powstawał w fazie testów. W nim napisane są 3 funkcje. Pierwsza z nich dotyczyła początkowego sprawdzenia, czy dodawanie zwierząt oraz wyświetlanie działa poprawnie. Druga funkcja służy do wyświetlania strony głównej. Trzecia funkcja dotyczy akcji, jakie dany użytkownik może wykonać, czyli wyświetlanie przycisków do danych odnośnie konta, edycji konta, w przypadku pracownika dodawanie zwierząt i wyświetlanie rezerwacji, w przypadku użytkownika wyświetlanie ulubionych i rezerwacji, a w przypadku administratora dodawanie pracownika, wyświetlanie wszystkich pracowników oraz zmienienie hasła.
* NewsController - w tym kontrolerze możliwe jest dodawanie, edytowanie, usuwanie wpisu, a także wyświetlanie wszystkich wpisów dotyczących danego zwierzaka. Wyświetlanie ma za zadanie działać w podobnym stylu, jak działają aktualności na większościach stron. Podczas dodawania wpisu pobierana jest aktualna data.
* PetController - kontroler odpowiedzialny za odpowiednie pobranie danych z bazy w celu wyświetlenia odpowiednich informacji dla danego typu użytkownika. W przypadku, gdy użytkownikiem jest pracownik kontroler sprawdza, jakie ma uprawnienia, dzięki czemu każdy pracownik po wejściu w konkretnego zwierzaka widzi odpowiednią dla niego akcję np edycję danych zwierzaka, czy też dodawanie zdjęć. Dodatkowo kontroler ten jest odpowiedzialny za wyświetlanie wszystkich zwierząt do adopcji, adoptowanych oraz zmarłych. W tym kontrolerze jest obsługa dodawania i edycji zwierząt. Jest to kontroler, który zawiera najwięcej linii kodu i jest odpowiedzialny za najwięcej zadań.
* PhotosController - kontroler, który umożliwia dodanie zdjęć dodatkowych, usuwanie zdjęć dodatkowych, a także możliwość zmienienia zdjęcia na główne.
* ReservationController - kontroler odpowiedzialny za dodawanie rezerwacji, zarządzanie rezerwacjami, czyli akceptowanie oraz usuwanie rezerwacji, a także wyświetlenie rezerwacji dla konkretnego użytkownika oraz wyświetlenie wszystkich rezerwacji pracownikowi posiadającemu odpowiednie uprawnienia.
* SecurityController - kontroler odpowiedzialny za logowanie, wylogowywanie, rejestrację, zmianę hasła, edycja danych użytkownika, wyświetlanie danych użytkownika i usunięcie konta.
* WorkerController - zadaniem tego kontrolera jest obsługa funkcji dotyczących pracowników, w tym dodawanie, wyświetlanie wszystkich, usuwanie, zmiana uprawnień.

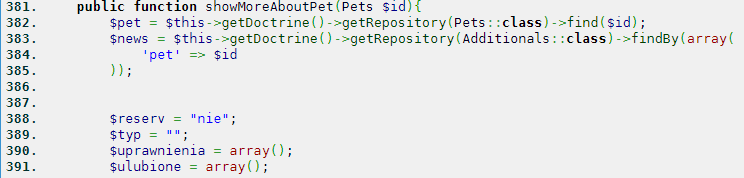
### Opis funkcjonalności:

W tym rozdziale przedstawiona zostanie jedna funkcjonalność, jaką jest wyświetlanie więcej informacji o zwierzaku. Aby wyświetlić informację o zwierzaku należy odpowiednio przygotować 2 pliki. Pierwszy z nich to kontroler - PetController, a drugi to widok - showPet.html.twig. Kontroler korzysta z klasy Pet znajdującej się w “src/Entity”, czyli klasy modelu. W pierwszej kolejności należy napisać kontroler, aby wiedzieć, jakie elementy będą potrzebne w widoku. Opisywany kontroler można znaleźć w “src/Controller/PetController.php” opisywana funkcjonalność zaczyna się od 375 linii kodu. Ostatnia linia w pliku to ostatnia linia funkcjishowMoreAboutPets(). Jest to najdłuższa funkcja ze względu na ilość rzeczy, jakie muszą być wyświetlone.

Na poniższym listingu linie od 375 do 380 to ustawienie odpowiednich wartości, tak aby kontroler wiedział, kiedy jest wywoływany po przejściu przez użytkownika na daną stronę. W linii 376 informuje się, że wymagany do działania tej funkcji jest parametr. W linii 4 ustawia się ścieżkę dostępu do zasobu. Jeżeli w linii 376 występuje informacja o parametrze, to taka sama informacja musi znaleźć się w linii 378. Dalej w linii 378 można ustawić, jakiego typu powinien być parametr - w tym przypadku liczba. Ostatnim parametrem funkcji Route jest nazwa, która jest wykorzystywana w różnych miejscach, między innymi w widoku na akcji na guziku. Linia 379 informuje jakiego typu będzie zwracany parametr funkcji.



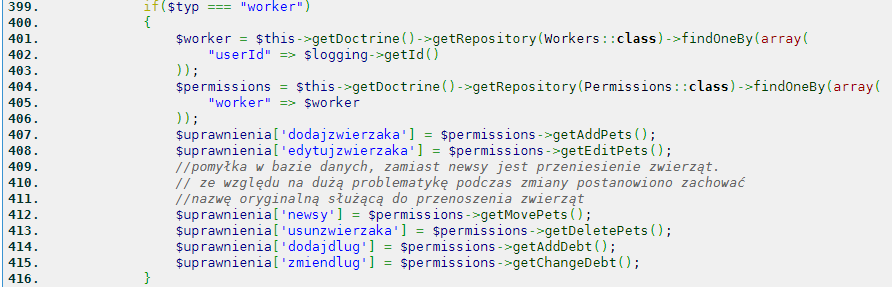
Następnie rozpoczyna się funkcja. W linii 382 pobiera się z bazy danych informacje o zwierzaku, którego id jest równe przekazanemu parametrowi do funkcji. Linie 383-385 odpowiadają za pobranie wpisów dla klucza obcego pet. Linia 388 i 389 to zmienne, które będą wykorzystywane w dalszym etapie oraz w widoku, dlatego powinny zostać ustawione. Linia 390 i 391 to utworzenie tablic.



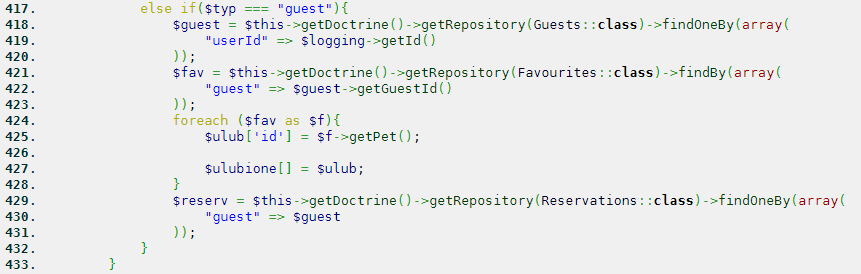
Dalej jest instrukcja warunkowa, która sprawdza, czy został ustawiony użytkownik. Jeżeli otrzyma się wartość true to w linii 395 ustawiane są dane logowania z klasy MyUser dla pola “email”. Linia 398 odpowiada za pobranie wartości pola dotyczącej typu konta.



Kolejnym etapem jest sprawdzenie, czy typ konta to pracownik. W tym celu wykorzystana została instrukcja warunkowa if else if. Jeżeli w linii 399 otrzyma się wartość true to dalej w linii 401 pobiera się pracownika tabeli Workers przez klucz, który łączy się z tabelą MyUser. Każdy pracownik ma ustawione uprawnienia więc linie 404-406 służą za pobranie informacji o nich. Tablica uprawnienia jest tablicą asocjacyjną, co oznacza, że pod konkretną nazwą jest dana wartość. Linie 407, 408 oraz od 412 do 415 służą wpisaniu pod danymi nazwami w tablicy uprawnień odpowiednich wartości. W liniach 409 - 411 znajduje się informacja o drobnej pomyłce, jaką było ustawienie jednego pola jako informację o przenoszeniu zwierząt.



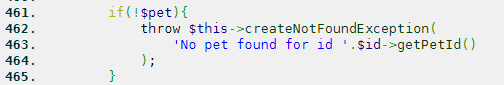
Następnie w linii 417 sprawdza się, czy typ konta to gość, czyli zarejestrowany użytkownik. Istnieje jeszcze typ konta admin. Linie 418 - 420 to pobranie danych użytkownika, linie 421 - 423 pobranie danych o ulubionych zwierzętach danego użytkownika. W linii 424 jest pętla foreach, która dla każdego elementu pobiera id zwierzaka i wstawia do tablicy ulubione. Linie 429 - 432 odpowiadają za pobranie informacji o rezerwacji danego użytkownika.Linia 433 to zamknięcie instrukcji warunkowej z linii 393.



Dalej w linii 435 sprawdzane jest, czy użytkownik ma jakąś rezerwację, jeżeli nie to wykonuje się linia 436 w przeciwnym wypadku linia 438. Kolejnym krokiem jest pobranie informacji o rezerwacji danego pupila, aby uniknąć sytuacji, że kilka osób zarezerwuje tego samego zwierzaka. Są to linie 440 - 442 oraz sprawdzenie czy zwierzak posiada rezerwację - linie 444 - 447. Następnie pobierane są dodatkowe zdjęcia zwierzaka w liniach 449 - 451 oraz dług zwierzaka, w liniach 453 - 455. Linie 457 do 459 odpowiadają za utworzenie 3 różnych tablic, które przydadzą się w dalszym etapie. Tablice te będą również potrzebne w widoku.



Następnie należy się upewnić, czy zwierzak o danym id istnieje w bazie. Jeżeli nie zgłaszany jest wyjątek.



Teraz zaczyna się sprawdzenie wszystkich potrzebnych wartości. Wykorzystana została instrukcja warunkowa if else if do sprawdzenia, jakie wartości zostały ustawione dla danego zwierzaka. Ze względu na to, że takich bloków kodu jest 8 zdjęcia listingi będą obcinane o te linie kodu, które wystąpiły wcześniej. Dodatkowo pojawi się stosowna informacja np. linie 120-150 odpowiadają liniom 20-50. Linia 466 sprawdza informuje o tym, że tylko jeden obiekt został ustawiony, czyli obiekt dotyczący zwierzaka ($pet). Linie 467 do 474 odpowiadają za pobranie informacji o danym pupilu. Następnie w linii 475 sprawdzany jest stan zwierzaka. Stany, w jakich może znajdować się pupil to:

*- adoptowany - leczony - zmarł - dostępny*

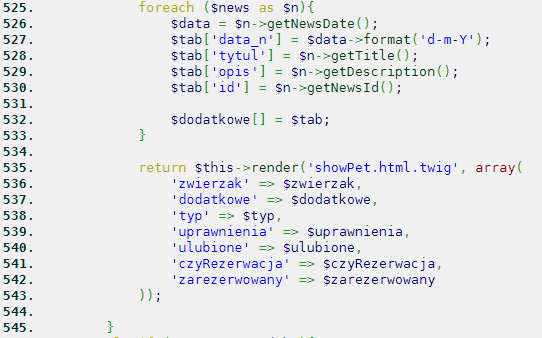
Jeżeli stan jest adoptowany to wykonuje się pobranie obiektu z tabeli Adopted, którego klucz obcy pet jest równy id zwierzaka. Następnie w linii 479 pobierana jest wartość pola odnoszącego się do daty adopcji. Linia 480 odpowiada za wpisanie do tablicy asocjacyjnej daty w odpowiednim formacie. Format ‘d-m-Y’ spowoduje wyświetlenie daty w następujący sposób “08-01-2019”. Linie 482 - 488 są wykonywane analogicznie do linii 475 - 481, z tą różnicą, że sprawdzane jest czy stan zwierzaka to “odszedł” oraz pobierana wartość pola daty śmierci jest z tabeli Dead. Linia 489 to przypisanie tablicy asocjacyjnej do tablicy zwierzak, jest to linia opcjonalna, która nie musiała być użyta. Linia 491 odpowiada za wygenerowanie odpowiedniego widoku, w liniach 492 - 497 przekazywane są wszystkie wartości do formularza, jakie zostały ustawione.



Jeżeli powyższy kod się nie wykona to sprawdzane jest, czy obiekty pet oraz news nie są puste.



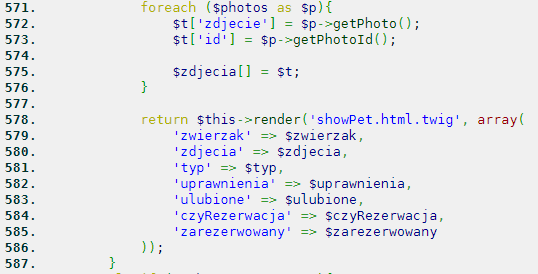
W przypadku, gdy spełnione jest wyrażenie wykonywany jest jego blok kodu. Linie od 501 do 524 odpowiadają liniom od 467 do 489. Wykonywana jest pętla foreach i dla każdego wpisu dotyczącego zwierzaka pobierane są odpowiednie wartości, takie jak data wprowadzenia wpisu, tytuł, opis oraz identyfikator wpisu. Identyfikator wpisu potrzebny jest w celu usuwania oraz edytowania treści. Są to linie od 526 do 530. Linia 532 jest linią obowiązkową. Ma ona za zadanie przypisanie do kolejnego klucza w tablicy (numeracja od 0) wartości, którą jest tablica asocjacyjna. Jeżeli nie będzie użyte takie zastosowanie to będzie możliwy dostęp do tylko ostatniego wpisu. Linia 535 odpowiada za wygenerowanie tego samego wyglądu, co powyżej. Natomiast w liniach 536 do 542 ustawiane są zmienne, które będą wykorzystywane w szablonie widoku.



Jeżeli wyrażenie nie było spełnione to sprawdzane jest kolejne, które informuje o tym, że obiekty pet oraz photos nie są puste. Linie od 547 do 569 są dokładnie takie same, jak linie 467 - 489.



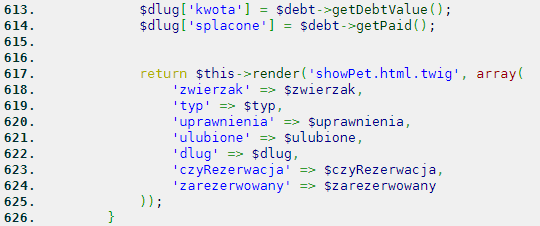
W dalszej w pętli foreach dla każdego elementu pobierana jest nazwa zdjęcia oraz jego id, które potrzebne jest do zmiany na główne zdjęcie oraz usunięcia zdjęcia. Linia 575 to przypisanie do kolejnego klucza w tablicy zdjecia, która jest numerowana od 0, wartości będącej tablicą t. Linia 578 to wywołanie konkretnego widoku, a kolejne linie odpowiadają a przekazanie odpowiednich parametrów do widoku.



Jeżeli powyższy blok nie został spełniony to sprawdza się, czy ustawione obiekty to zwierzak i dług. linie 589 - 612 odpowiadają liniom 467 - 489.



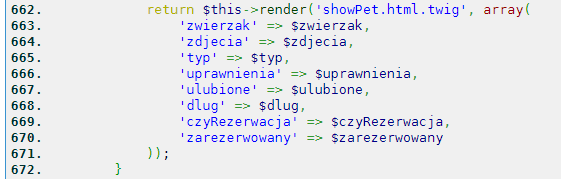
Linia 613 to wpisanie wartości długu zwierzaka, a kolejna linia dotyczy wartości spłaconej. Nie jest pobierane id, ponieważ użytkownik nie ma możliwości edycji długu. Następnie w linie 617 generowany jest odpowiedni widok, a niżej przekazywane parametry, które będą wyświetlane w szablonie.



Ten blok pozwala sprawdzić, czy obiekty posiadające wartość to pet, photos i debt. Jeżeli tak to wykonywany jest ten blok. Linie 628 - 650 odpowiadają liniom 467 - 489. Linie 652 - 658 odpowiadają liniom 571 - 577, a linie 659 - 661 odpowiadają liniom 613 - 615.



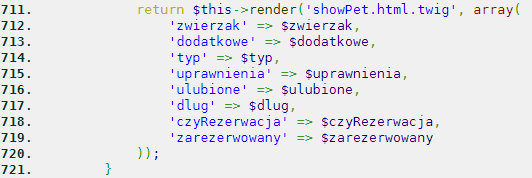
W linii 662 generowany jest odpowiedni wygląd, a niżej przedstawione są parametry, jakie zostały przekazane do widoku.



Kolejny blok pozwala sprawdzić, czy ustawione obiekty to pet, debt i news. Linie 674 - 696 odpowiadają liniom 467 - 489, linie 698 - 707 odpowiadają liniom 525 - 534, linie 708 - 710 odpowiadają liniom 613 - 615.



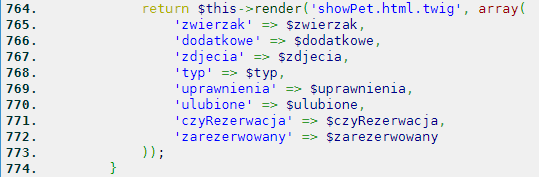
Następnie kontroler uruchamia odpowiedni widok i przekazuje do niego odpowiednie parametry.



Dalszym sprawdzanym blokiem jest blok, którego obiekt debt jest pusty. Linie 723 - 745 odpowiadają liniom 467 - 489, linie 747 - 756 odpowiadają liniom 525 - 534, linie 757 - 763 odpowiadają liniom 571 - 577.



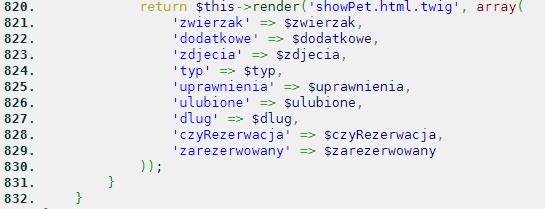
Renderowany jest właściwy widok i przekazywane są do niego parametry.



W przypadku, gdy żaden z powyższych bloków nie został spełniony oznacza to, że wszystkie obiekty posiadają wartość. Linie 776 - 798 są takie same jak 467 - 489, 800 - 809 odpowiadają liniom 525 - 534, linie 810 - 816 są równe liniom 571 - 577, a linie 817 - 819 odpowiadają liniom 613 - 615.

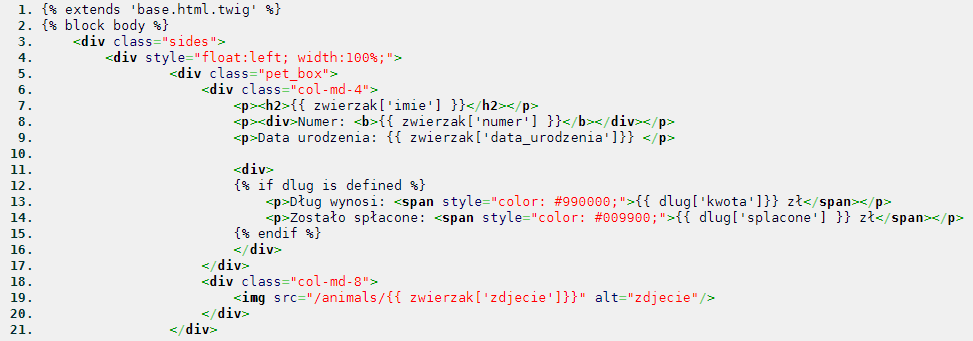


Ostatni etap to uruchomienie odpowiedniego widoku z parametrami, które mają być przekazane. Linia 832 kończy opisywaną funkcję.

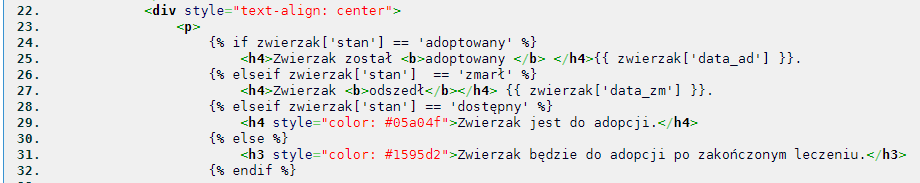


Drugi plik jest widokiem. Jest to “templates/showPet.html.twig”.

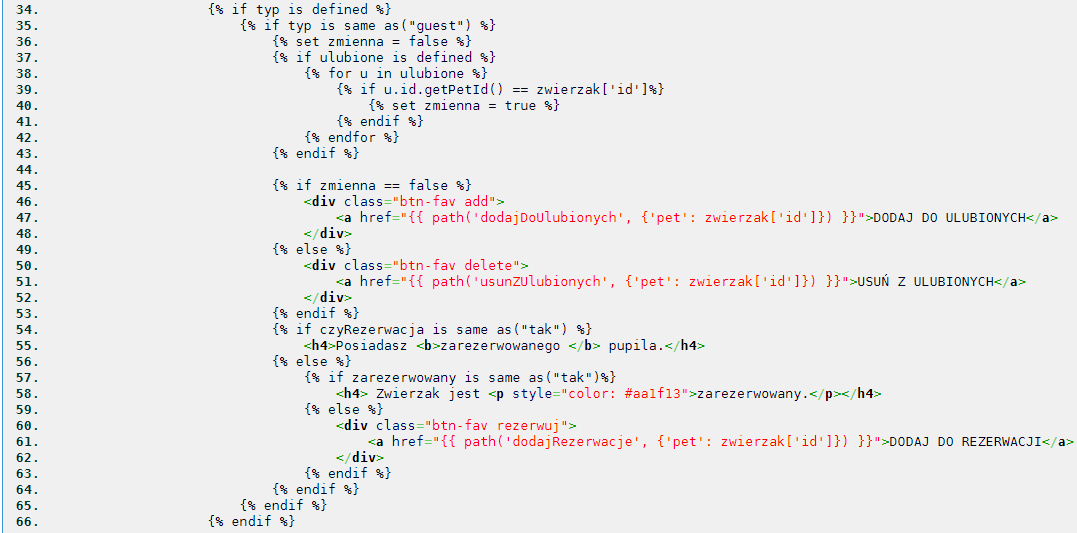
Linie pierwsza to wczytanie pliku podstawowego, który jest rozszerzany przez dodawanie elementów w blokach. Linia 2 nadpisuje blok body w pliku ‘base.html.twig’. Linia 7, 8, 9 to pobranie wartości z kontrolera. W linii 12 sprawdzane jest, czy zmienna dlug została zdefiniowana, jeżeli tak to linia 13 i 14 odpowiadają za pobranie wartości.



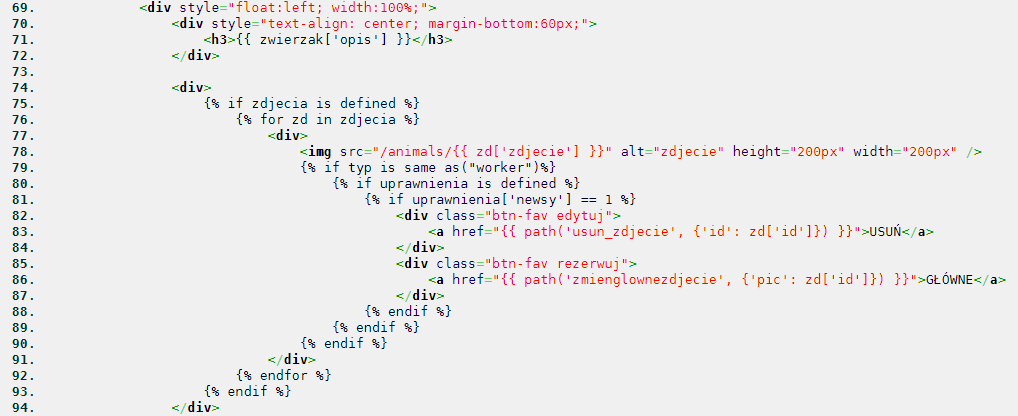
Następnie linia 24 odpowiada za sprawdzenie, czy stan zwierzaka to ‘adoptowany’, jeżeli tak to wypisywana jest data adopcji, w przeciwnym razie sprawdzane jest czy stan jest równy ‘zmarł’, jeżeli tak to wyświetlana jest informacja z datą śmierci. Jeżeli żaden z powyższych stanów się nie sprawdził to patrzy się, czy stan jest równy ‘dostępny’. Jak tak to ustawiany jest odpowiedni kolor i informacja, w przeciwnym razie stan jest ‘leczony’, więc kolor i informacja są inne niż przy dostępnym zwierzaku.



Dalej w linii 34 sprawdza się, czy typ został zdefiniowany w kontrolerze. Jeżeli tak to sprawdza się, czy typ konta to guest. Jeżeli wszystko się zgadza to ustawia się zmienną pomocniczą na false, a następnie sprawdza, czy zmienna ulubione została zdefiniowana. Jak tak to przestawia się zmienną pomocniczą na true. Można było w kontrolerze pobrać ulubionego zwierzaka, który ma opisany klucz obcy pet oraz guest. Dzięki temu uniknęłoby się pętli for. Ze względu na to, że pobrane wartości z bazy danych dotyczyły wszystkich zwierzaków, które były dodane do ulubionych przez konkretnego użytkownika należało wykonać pętlę for i znaleźć interesującego pupila. W dalszej części sprawdzane jest, czy ustawiona zmienna pomocnicza jest false. Jak tak to wyświetla się przycisk DODAJ DO ULUBIONYCH, w przeciwnym razie USUŃ Z ULUBIONYCH. Linia 54 to sprawdzenie czy zmienna rezerwacja jest równa “tak”, jeżeli tak to wyświetla się informacja “Posiadasz zarezerwowanego pupila”, jak nie to sprawdza się, czy pupil był już przez kogoś zarezerwowany, jeżeli tak to wyświetla się taka informacja, a jak nie to pojawia się przycisk DODAJ DO REZERWACJI. Co ważne użytkownik nie może usunąć rezerwacji.



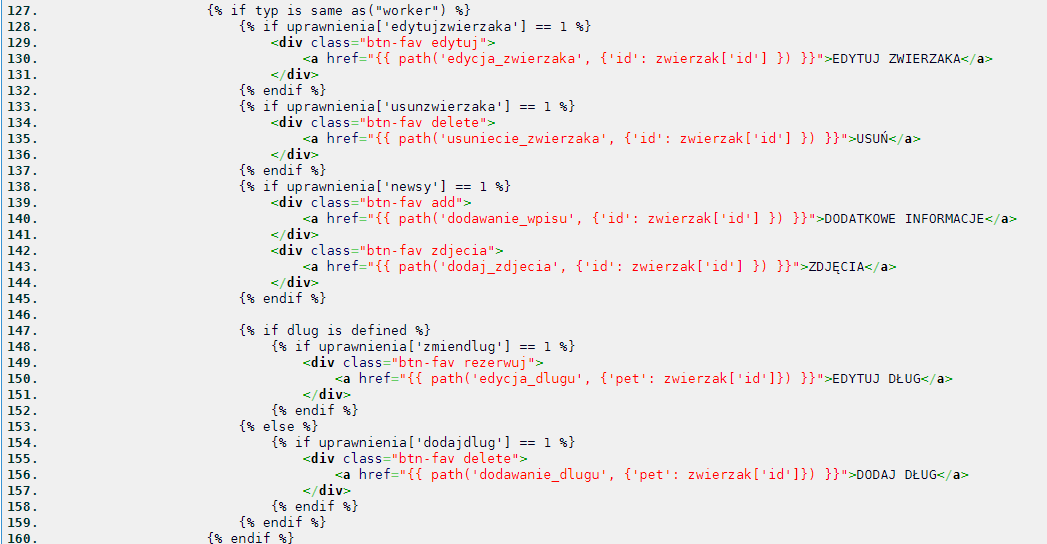
Kolejny etap to linia 71, która pobiera opis zwierzaka i go wyświetla. Linia 75 sprawdza, czy zmienna zdjęcia jest zdefiniowana, jeżeli tak to wykonuje się pętla for ( w innych językach zapisana jako foreach). Wyświetlane jest zdjęcie i wewnątrz pętli sprawdzane jest, czy typ konta to pracownik. Jeżeli tak sprawdzane są jego uprawnienia do modyfikacji zdjęć. Gdy pracownik posiada odpowiednie uprawnienia to zostaną wyświetlone przyciski USUŃ oraz GŁÓWNE.



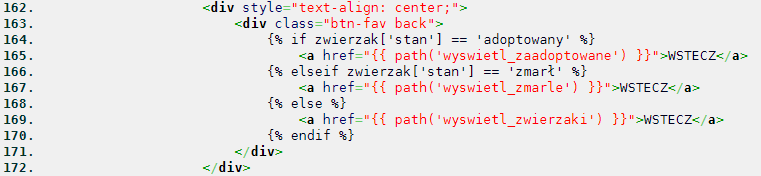
Następna część to sprawdzenie w linii 98, czy zmienna dodatkowe odpowiedzialna za wyświetlanie wpisów jest zdefiniowana, jeżeli tak to wykonuje się pętla for (w innych językach zapisywana jako foreach), w której wyświetlane są odpowiednio elementy takie jak data, tytuł i opis. Dalej należy sprawdzić, czy typ konta to pracownik, jeżeli tak to sprawdzane są jego uprawnienia do wpisów i w przypadku, gdy uprawnienia się zgadzają zostają wyświetlone przyciski EDYTUJ oraz USUŃ.



Przedostatni sprawdzany etap to typ konta. Jeżeli danego pupila wyświetlił pracownik to trzeba mu udostępnić przyciski zgodnie z jego uprawnieniami. W 147 linijce sprawdza się, czy zmienna dług została zdefiniowana, jeżeli tak to dla pracowników, którzy mają możliwość edytowania długu wyświetla się odpowiedni przycisk, w przeciwnym razie dla takich pracowników wyświetla się przycisk z opcja dodania długu zwierzęcia.



Ostatnim elementem jest zobaczenie stanu zwierzaka, aby można było cofnąć się do odpowiedniej listy zwierząt.



### Unikalne linki:

Poniżej znajduje się 5 linków, do których widoku ma niezarejestrowany użytkownik, aby zobaczyć funkcjonalności należy zalogować się na jedno z kont. Konto administratora, gościa oraz pracownika posiadają inne widoki. Po sprawdzeniu w tabeli na stronie 2 uprawnień danego pracownika można większość rzeczy zobaczyć, po wejściu w danego zwierzaka, a także po kliknięciu w zakładkę konto, a następnie Moje konto.

* <http://mjaw.ayz.pl/pets> - zwierzaki do adopcji
* <http://mjaw.ayz.pl/adopted> - zwierzaki zaadoptowane
* <http://mjaw.ayz.pl/dead> - zwierzaki, które odeszły
* <http://mjaw.ayz.pl/login> - logowanie
* <http://mjaw.ayz.pl/register> - rejestracja

### Problematyka:

Pierwszy napotkany problem wystąpił podczas instalacji frameworka Symfony 3.4.18. Instalacja została przeprowadzona zgodnie z zaleceniami na oficjalnej stronie Symfony. Do instalacji został użyty Symfony Installer. Problem z instalacją udało się rozwiązać poprzez zainstalowanie composer’a i uruchomienie odpowiedniej komendy.

Innym problemem okazała się instalacja dostępnej wersji testowej na githubie. Podczas instalacji wystąpił błąd informujący o brakujących parametrach. Aby rozwiązać ten problem należy kilkakrotnie kliknąć klawisz Enter, co spowoduje ustawienie domyślnych wartości i projekt zostanie zainstalowany.

Kolejny problem pojawił się podczas próby uruchomienia pierwszej strony. Wszystko zostało zrealizowane zgodnie z instrukcją wideo dostępną na oficjalnej stronie dokumentacji. W PHPStorm po dodaniu katalogu src do “Sources Root” tworzy się automatycznie przestrzeń nazw. W Symfony 3 utworzona przestrzeń nazw dla Controllera wyglądała następująco:

**namespace** Controller;

Po próbie uruchomienia strony pojawiał się błąd informujący o tym, że kontroler został znaleziony, ale klasa kontrolera już nie lub problem jest z przestrzenią nazw. Na wielu stronach rozwiązaniem tego problemu było sprawdzenie, czy plik rozpoczyna się od “<?php” lub czy nazwa pliku jest taka sama jak nazwa klasy. Żadne z powyższych rozwiązań nie pomogły. Dopiero zmiana przestrzeni nazw na:

**namespace** App\Controller;

spowodowała poprawne uruchomienie się projektu.

Następny problem dotyczył widoków. Podczas próby uruchomienia Template Service pojawił się błąd. Po wykonaniu polecenia, jakim było przejście do folderu var kliknięcie prawym przyciskiem myszy na cache i wybranie opcji Mark Directory as Excluded powinna być możliwość wpisania takiej linijki kodu:

$templating = $this->container->get('templating');

Niestety ‘templating’ nie pojawił się odpowiedni element przez co przy każdej próbie uruchomienia projektu pojawiał się następujący błąd:

Próbowano doinstalować potrzebne elementy poprzez composer jednak nie przyniosło to zamierzonego efektu. Dopiero przejście do pliku framework.yaml znajdującego się w config->packages i dodanie tam poniższego fragmentu:

**templating**:

**engines**:

twig

całkowicie rozwiązało problem i umożliwiło dalsze korzystanie.

W dokumentacji do Symfony 3 znajdują się informacje o umieszczeniu styli, skryptów, zdjęć itp w folderze web. W przypadku Symfony 3.4 taki folder nie został utworzony, bo jego utworzeniu i dodaniu plików nie będą one widoczne w taki sposób, w jaki jest to opisane w dokumentacji. Aby uzyskać satysfakcjonujący widok należy odpowiednie foldery umieścić w folderze public. Folder public odpowiada za to samo, co we wcześniejszych wersjach Symfony 3 folder web.

Kolejną napotkaną trudnością było odpowiednie utworzenie bazy danych. Utworzono dwa pliki, które były opisane w dokumentacji. Okazało się, że wszelkie zmiany dotyczące nazwy użytkownika, hasła, nazwy nowej bazy danych należało uzupełnić w pliku .env, który znajduje się w głównym katalogu projektu. Pojawił się inny problem, dotyczący kodowania znaków. Po zmianie w Doctrine a także w pliku konfiguracyjnym w XAMPP nie uzyskano kodowania utf-8. Postanowiono zostawić ten problem, ze względu na dużą ilość czasu, jaka została mu poświęcona.

Aktualizacja XAMPP. Postanowiono uaktualnić XAMPPa do wersji, z jakiej można korzystać z Symfony 4.\*. Ze względu na liczne dane jakie były w starej wersji zainstalowano XAMPPa w innym folderze. Pojawiły się pewne problemy z uruchomieniem apache więc zdecydowano się na odinstalowanie wszystkich wersji XAMPP i zainstalowanie najaktualniejszej. Okazało się, że nowa wersja, nie może zostać uruchomiona, bo została ścieżka do nieistniejącego folderu. Rozwiązanie tego problemu polegało na przejście do services.msc i tam wyszukanie odpowiednich nazw serwisów. W tym przypadku “Apache 2.4” oraz “mysql”. Następnie należało uruchomić cmd jako administrator i wpisać komendę sc delete “Apache 2.4” oraz sc delete “mysql”. W dalszym kroku wystarczyło już uruchomić XAMPP i zainstalować wymagane serwisy.

Następne problemy wynikły z połączenia bazy danych z projektem. Zbudowanie bazy danych można wykonać na wiele sposobów. Wystarczy edytować plik .env a następnie w konsoli wpisać komendę

php bin/console doctrine:database:create

co spowoduje utworzenie bazy danych. Ze względu na fakt, iż baza danych była już wcześniej zapisana postanowiono zaimportować tabele do nowej bazy danych. Po wykonaniu tego zabiegu należało wpisać dwie komendy, aby wykreować klasy ułatwiające połączenie z bazą danych.

php bin/console doctrine:mapping:import 'App\Entity' annotation --path=src/Entity

Powyższa komenda spowoduje utworzenie klas wraz ze wszystkimi polami, których nazwy pokrywają się z nazwami w bazie danych. Dodatkowo utworzone zostaną adnotacje umożliwiające bibliotece Doctrine łatwe powiązanie pól w klasie z tymi z bazy danych.

php bin/console make:entity --regenerate App

Ostatnie wykonane polecenie służy do aktualizacji klas o gettery i settery, czyli specjalne metody, które umożliwią łatwe pobieranie i ustawianie odpowiednich elementów. W późniejszych etapach utworzenie klasy umożliwiającej logowanie się spowodowało wiele błędów. Pierwszy problem to wykorzystanie istniejącej tabeli do logowania. Najszybszym sposobem na logowanie w symfony było utworzenie klasy użytkownika. Skorzystano ze strony, w której krok po kroku opisany był proces logowania ([*https://symfonycasts.com/screencast/symfony-security/make-user*](https://symfonycasts.com/screencast/symfony-security/make-user)). Zanim przejdzie się do wykonywania poleceń należy sprawdzić, z jaka wersja serwera bazy danych jest wykorzystywana w projekcie, ponieważ może pojawić się błąd, który uniemożliwi utworzenie klasy. Ważnym elementem jest usunięcie błędnych migracji. Jeżeli tego się nie zrobi to po poprawieniu danych w projekcie nie będzie możliwe uzyskanie poprawnego wyniku. W przypadku, gdy po próbie wgrania migracji pojawia się błąd taka migracja jest do usunięcia. Następnie wprowadza się błędy, wywołuje polecenie do utworzenia kolejnej, właściwej migracji i wykonuje polecenie, które wgra migrację. Po wprowadzeniu klasy użytkownik do bazy danych zdecydowano się na usunięcie tabeli odpowiedzialnej za logowanie. Rozszerzono klasę użytkownik o pola, jakie występowały w tabeli logowanie. Pojawił się spory problem przy usuwaniu tej tabeli. Tabele workers i guests miały połączenie poprzez klucz obcy z tabelą logging. Postanowiono wyeksportować plik z bazy danych, wprowadzić poprawki, a następnie go zaimportować do bazy danych i do projektu. Wydawało się, że wszystko się udało. Jednak po wprowadzeniu pewnych zmian na serwerze okazało się, że nie jest możliwa ani rejestracja użytkownika, ani logowanie. Po długich poszukiwaniach znaleziono problem. Klasa użytkownik z bazy danych różniła się znacząco od klasy użytkownik utworzonej przez Doctrine w konsoli. Postanowiono dołączyć implementację potrzebnego interfejsu oraz metody, jednak błąd logowania i rejestracji w dalszym ciągu występował. Zdecydowano się na przywrócenie właściwej wersji projektu na serwerze, sprzed wprowadzenia możliwości logowania. Zadanie to nie było łatwe. Krok po kroku tworzono nowe pliki zaczynając od bazy danych. Tutaj pojawił się kolejny problem. Pliki zgodne z diagramem ERD powstały po zaimportowaniu istniejącej bazy danych do projektu. Przed importem z bazy danych postanowiono utworzyć bazę za pomocą Doctrine. Niestety różnica między plikami była duża. Obecna wersja projektu nie posiada połączenia z tabelą użytkownik. Dwie tabele (workers i guests), które powinny być połączone mają pole “user\_id”, do którego zapisywana jest wartość znajdująca się w tabeli MyUser. Nie znaleziono skutecznego sposobu na utworzenie klucza obcego między tabelami, a z powodu braku czasu postanowiono nie eksperymentować, ponieważ mogłoby się to skończyć zmarnowanym czasem na przywróceniu aplikacji do poprawnego działania.