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**

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 proszę o kontakt z autorem.

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 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

Wykorzystane technologie:

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

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.

Diagramy UML:

Diagram przypadków użycia dla administratora.

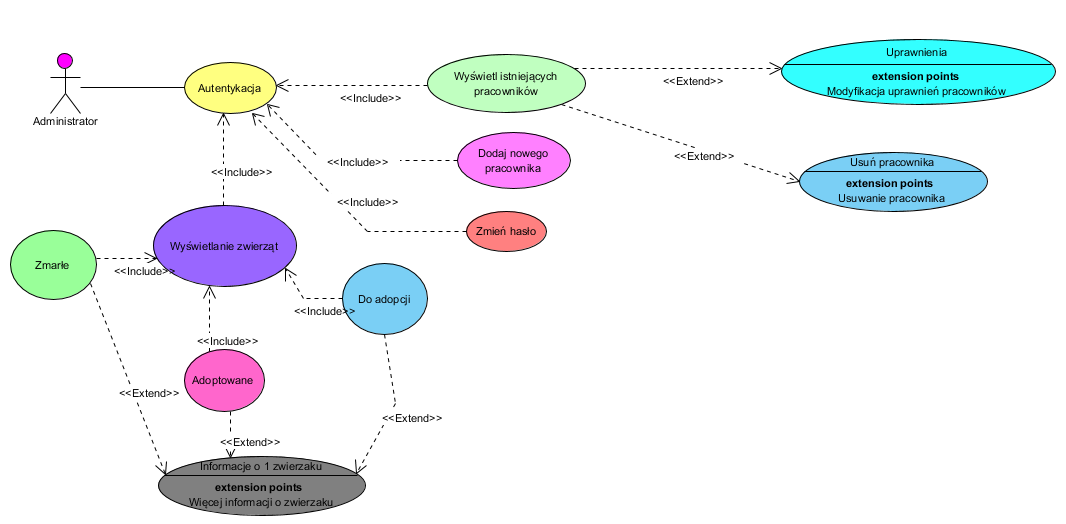


Diagram przypadków użycia dla niezarejestrowanego użytkownika.

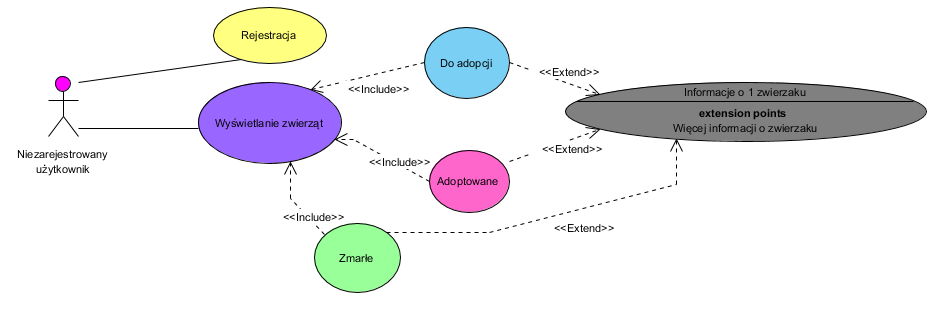


Diagram przypadków użycia dla zarejestrowanego użytkownika.

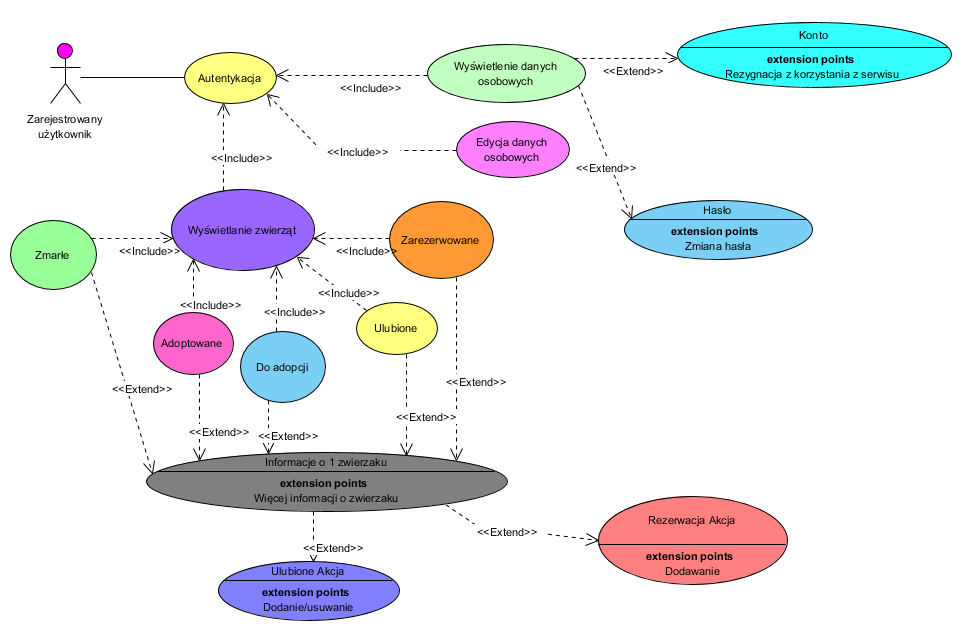
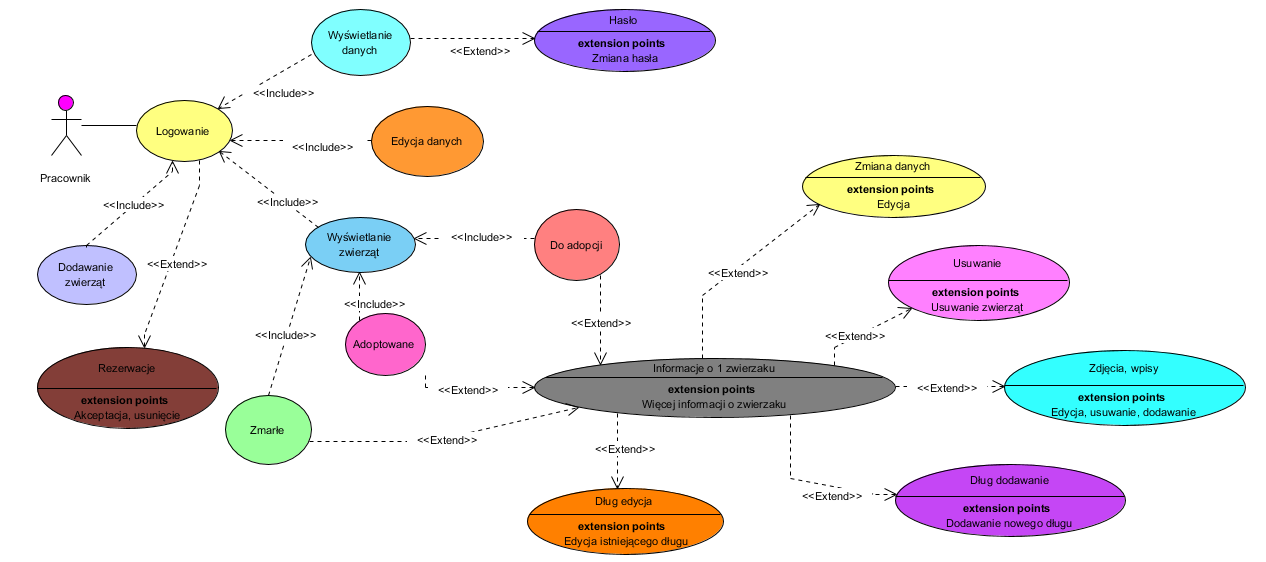
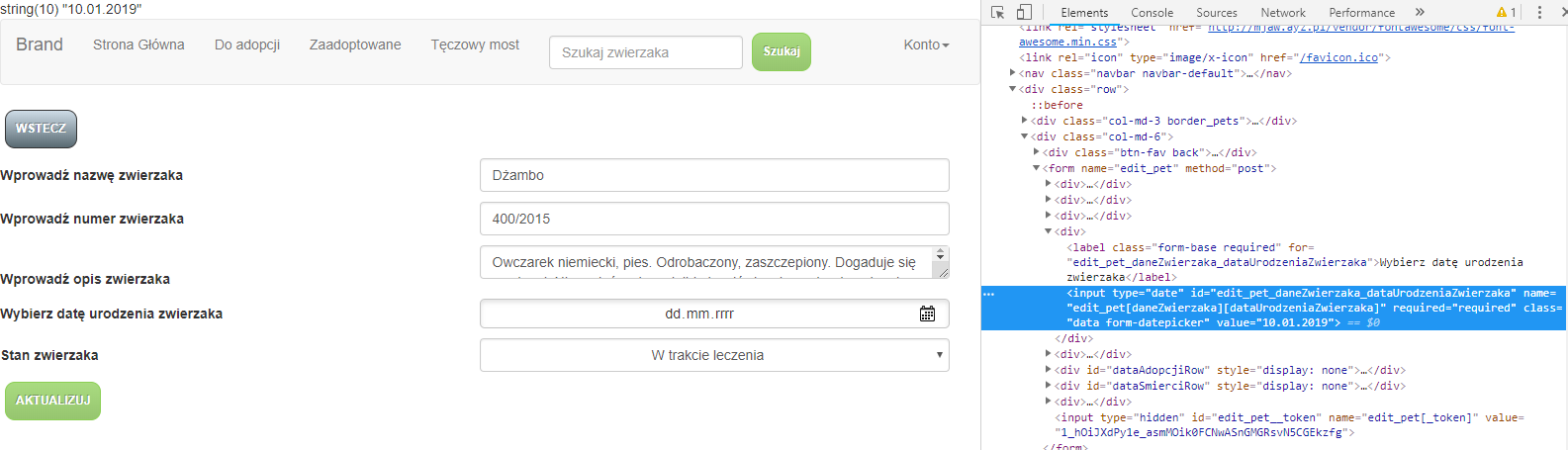


Diagram przypadków użycia dla pracownika.



Front-end:

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:

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.

Na zakończenie przygotowanie opisu przygotowania jednej pełnej funkcjonalności w aplikacji lub wybranego elementu. W ramach opisu odnosimy się do własnego kodu i konkretnie wskazujemy (pliki, przestrzenie nazw, klasy, metody) oraz numerujemy listingi wskazując którą linię opisujemy, aby ułatwić procedurę weryfikacji.

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

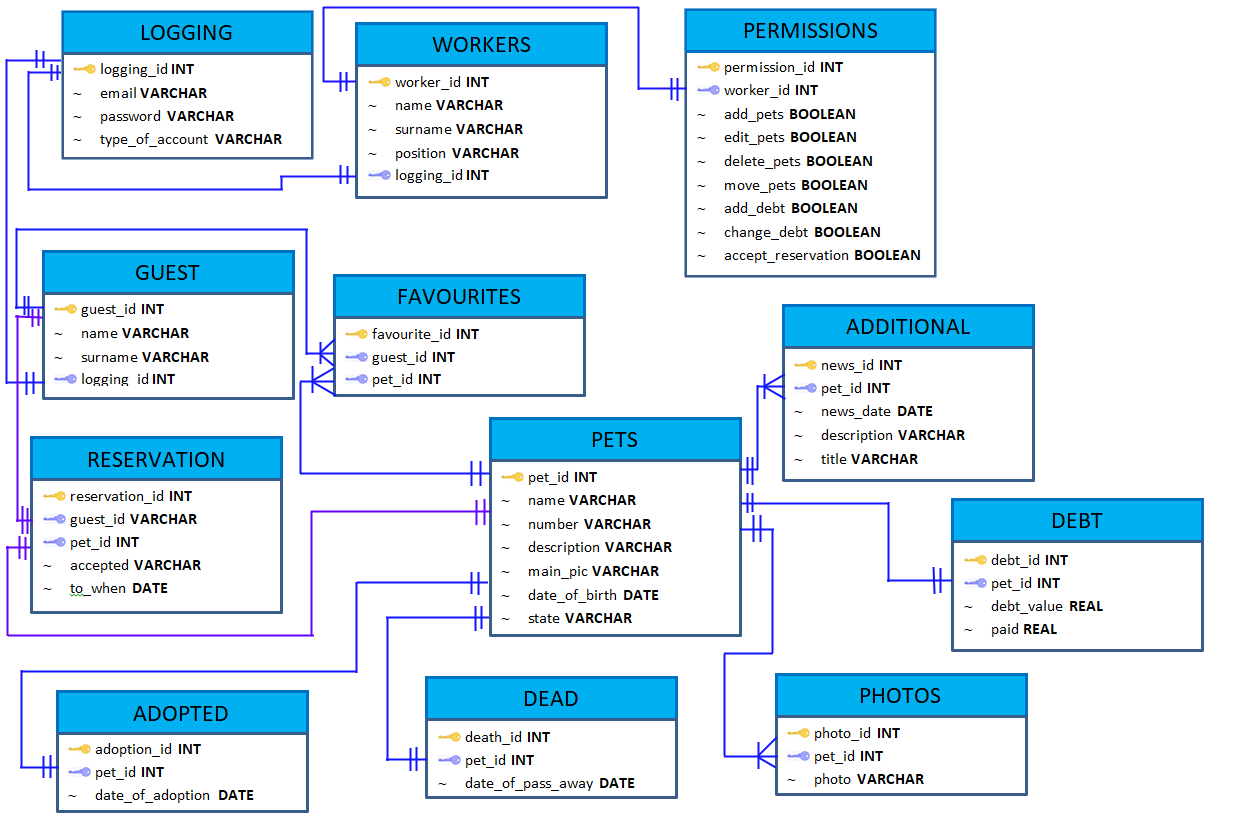
Powyższa komenda spowoduje utworzenie klas wraz ze wszystkimi polami, których nazwy pokrywają się z nazwami w bazie danych. Dodatkowo utworzone zostaną adnotacje umożliwiające bibliotece Doctrine łatwe powiązanie pól w klasie z tymi z bazy danych.

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.

Co zostało wykonane:

diagram ERD bazy danych,



Podczas pracy nad bazą danych postanowiono dodać jedno pole w tabeli WORKERS. To pole to data urodzenia, diagram zostanie zaktualizowany po wprowadzeniu wszystkich poprawek.

Elementy jakie działają:

* wyświetlanie zwierząt - lista zwierząt, każde zwierzę z imieniem, dostępnością, numerem, datą urodzenia oraz przyciskiem więcej, który umożliwia przejście do podglądu konkretnego zwierzaka
* dodawanie pracownika
* w podglądzie konkretnego zwierzaka można:
  + dodać dodatkowe informacje
  + edytować każdy wpis,
  + usunąć wpis,
  + zmieniać dane zwierzaka,
  + dodać kolejne zdjęcie,
  + usunąć konkretne zdjęcie,
  + usunąć zwierzaka

Problemy do rozwiązania:

* podczas wprowadzania zwierzaka w momencie wybrania opcji z listy (odszedł/adoptowany) powinna pojawić się opcja wprowadzenia daty śmierci/adopcji. Po rozwiązaniu tego problemu zostanie naprawiony sposób dodawania/edycji zwierząt, które zmarły lub są adoptowane. W tej chwili wyświetlanie działa pod warunkiem, że na sztywno wpiszę się w bazie danych informację o dacie.
* przeniesienie zdjęcia z jednego folderu do drugiego, tak aby można było w łatwy sposób zmienić zdjęcie główne.

Następny etap:

* dodanie obsługi długów do zwierząt,
* rejestracja użytkowników
* logowanie pracowników i użytkowników
* edycja danych osobowych pracowników i użytkowników
* wyświetlenie pracowników (lista pracowników)
* usunięcie pracowników / konta użytkownika
* zmiana uprawnień pracownika
* umożliwienie dodania do ulubionych
* umożliwienie dodania rezerwacji
* akceptacja rezerwacji (edycja)
* usunięcie rezerwacji
* wyświetlanie ulubionych

Dalszy etap:

* wprowadzenie JavaScript
* wyglądy