Projekt indywidualny,

Małgorzata Szwed

**Elektroniczny system obsługi siłowni: Twoja siłownia, online**

**16 września 2014**

Najnowsza wersja jest dostępna online: https://gym-online.herokuapp.com/

**Cel (i ogólny opis)**

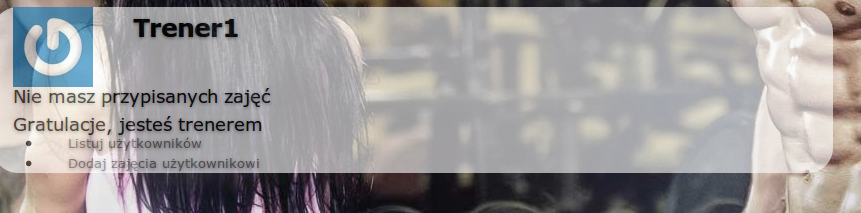
Celem projektu jest stworzenie dynamicznej aplikacji webowej we frameworku Ruby on Rails do obsługi elektronicznego systemu zarządzania siłownią.

W systemie współegzystują dwa rodzaje użytkowników:

-trenerzy

-osoby ćwiczące/kursanci

Trenerzy układają każdemu kursantowi plan treningowy (ćwiczenie/udział w zorganizowanych zajęciach) dostosowany do jego potrzeb i aktualnej kondycji. Analizują postępy kursantów. Prowadzą konkretne zajęcia (plan zajęć).

(panel trenera)

Mogą wpisać w dziennik danego kursanta konkretne zajęcia fizyczne. Aby to zrobić muszą podać:

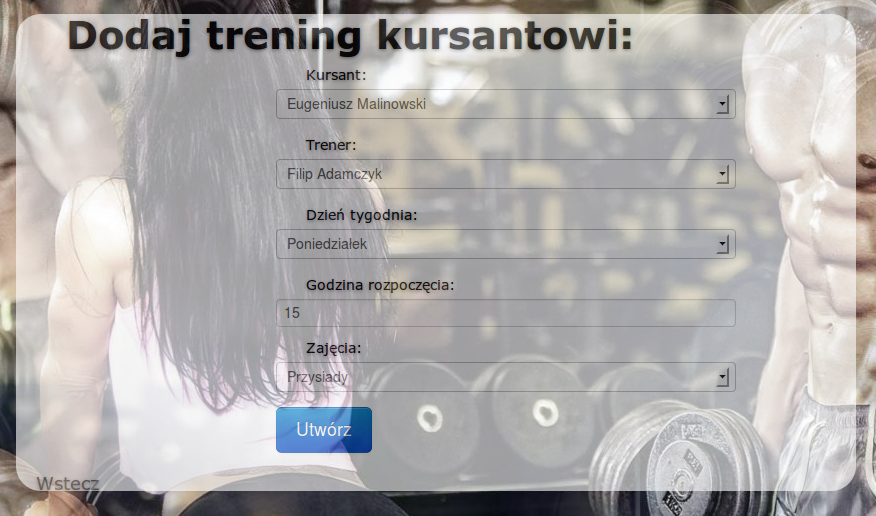
-kursanta (imie i nazwisko)

-dzień tygodnia w planie

-godzine rozpoczęcia

-rodzaj zajęć

Trener może także zobaczyć listę wszystkich kursantów w danej siłowni oraz wejść w profil każdego kursanta i zobaczyć jego zajęcia (aby nie spowodować kolizji i aby najlepiej zoptymalizować trening - nie przeciążyć kursanta).



(trener -> dodawanie nowych zajęć dla kursanta; w oparciu o CRUD)

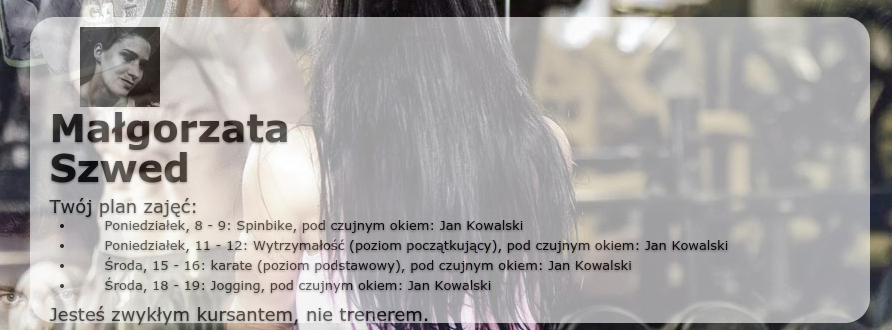
Kursant – ma wgląd do planu treningowego na dany tydzień/miesiąc/okres. Monitoruje swoje postępy (monitorowanie odbywa się w sposób standardowy dla monitorowania postępów w świecie rzeczywistym. Kursant nanosi odpowiednie dane - np. wagę, maksymalną szybkość chwilową, maksymalną wagę ciężaru, jaką może udźwignąć - na kratki powiązane z datą. Proces monitorowania odbywa się poza systemem elektronicznym) i urealnia termin osiąganych celów. Przy rejestracji wprowadza:

-imie

-nazwisko

-adres email

-hasło

(widok kursanta z aktualnym, posortowanym po dacie i godzinie planie treningowym oraz zdjęciem - obsługa zdjęcia przez serwis zewnętrzny: graviator)

Taka wizja projektu daje możliwość znacznego jego rozbudowywania. Na cele uczelniane powstanie jednak tylko podstawowa funkcjonalność.

**Dokładny opis, które części kodu zostały napisane przeze mnie lub zmodyfikowane** na bazie wygenerowanych standardowo plików znajduję się w kodzie źródłowym. Jest to najbardziej przejrzysta forma prezentacji tych danych, gdyż powtarzanie całego kodu w dokumentacji mija się z celem a szczegółowy opis kodu w oderwaniu od kodu nie jest oczywiście wygodny przy czytaniu.

Dokumentacje wygenerowaną na podstawie komentarzy z kodu dołączam do tego pliku (z opisem wszystkich nieoczywistych lub niestandardowych bytów/funkcji/klas). Tag “S” na początku komentarza oznacza, że najbliższy fragment był napisany/modyfikowany przeze mnie.

Metody generowane automatycznie w kontrolerach:

-index: odpowiada za wyświetlenie (wylistowanie) wszystkich rekordów dla danego modelu

-show: wyświetlenie szczegółów dot. konkretnego modelu

-new: możliwość dodania nowego rekordu opisanego przez konkretny model (powiązany z danym konrolerem)

-edit: edycja konkretnego rekordu

-create: funkcja odbiera dane z formatki tworzącej nowy rekrod i zapisuje go w bazie.

-update: odbiera dane od formatki aktualizującej dany rekord i modyfikuj go w bazie

-destroy: odpowiada za usunięcie rekordu z bazy

W przypadku, gdy konstrukcje powtarzają się wielokrotnie, są one opisane tylko raz (w pliku najmniejszym według nazwy w porządku leksykograficznym).

Cały kod dot. modeli (kat. models) tworzony łasnoręcznie (bez dalszych ostrzeżeń w kodzie).

Standardowo tworzone są widoki:

-edit

-index

-new

-show

Co odpowiada oczywiście (i jest automatycznie powiązane) z opisanymi wcześniej funkcjami w konkretnych kontrolerach. Powiązanie jest wyspecyfikowane w pliku routes.rb.

Widoki związane z json’em nie zostały modyfikowane przeze mnie (są automatycznie wygenerowane).

Z kat. config modyfikowałam tylko routes.rb.

schema.rb - plik generowany “niby” automatycznie, ale zawiera opis, który został podany całkowicie explicite przeze mnie podczast stosowanie odpowiednich komend do tworzenia modeli w bazie.

development.sqlite3 - plik zawiera deweloperską (a’la testową) wersje bazy danych (nie produkcyjną/serwerową).

spec - przyjmijmy dla uproszczenia, że całkowiecie wg. railstutorial

vendor/assets/images - trzy proponowane obrazy na główne tło strony

**Typowe, często używane konstrukcje:**

nazwaModeluWBazie.all - pobiera wszystkie rekordy opisywane przez dany model z bazy

zmienna = nazwaModeluWBazie.new(... parametry opisujące rekord ...) - tworzy nowy, pusty rekord dla danego modelu i jeszcze nie zapisuje go

zmiennaJakoModel.save - zapisuje dany rekord w bazie

private - informuje, że kod następujący po nim to funkcje prywatne. Nie mogą być wywołane z zewnątrz.

MS - skrót od Małgorzata Szwed

A - skrót od autor

konstrukcje ze znakiem zapytania - sprawdzają, czy warunek jest spełniony: zwracają tak lub nie (bool).

zmienna ||= P\_Wartość - jeśli zminna jest pusta, zostanie na nią przypisana P\_Wartość.

setZBazyDanych.count - rozmiar setu z bazy danych

has\_many: - mówi, że dany model posiada wiele innych modeli

belongs\_to: mówi, że dany model należy do danego modelu

dependent: - opisuje sposób zachowania tzn. co mamy zrobić z danym rekordem, gdy rekord z nim powiązany został usunięty. Możemy ten rekord także usunąć (jeśli pole to jest wymagane, to nie mamy innej opcji).

# - rozpoczyna komentarz jednoliniowy w kodzie ruby

/\* \*/ - kom. wieloliniowy w kodzie css/sass

<!-- .... --> -komentarz w pliku html

resources: - powoduje wygenerowanie w pliku routes.rb (tam też jest obecny ten “napis”) podstawowych powiązań między funkcjami w danym kontrolerze, widokami a ścieżkami (adresami z przeglądarki)

**Funkcje (w skrócie):**

-tworzenie użytkownika

-CRUD dla diet, trenerów, (użytkowników - wspomniane wcześniej), zajęć, wpisów w planie użytkowników

-trener może układać plan kursantowi

-wyświetlenia swojego planu

-zmiana danych osobowych w tym zdjęcia

-dostęp do polecanych przez siłownie diet

-dostęp do informacji o trenerach

-kontakt z trenerami

**Opis (bardziej szczegółowe dopowiedzenie)**

Nie zaprojektowano specjalnych algorytmów do optymalizacji planu zajęć kursanta. Trener przed dodaniem zajęć do planu kursanta ma obowiązek zapoznać się z jego aktualnym planem treningowym. Zajęcia są dodawane ręcznie, co odpowiada niejako wpisaniu notatki w dzienniku kursanta.

Siłownia dostarcza swoim kursantą podstawowej wiedzy odnośnie najbardziej skutecznych diet. Dieta nie jest układana według żadnych algorytmów. Z punktu widzenia naszego projektu, zostaje po prostu pobrana informacja z bazy i wyświetlona na stronie. Kursant podejmuje decyzje o podjęciu danej diety (lub nie) po odbyciu osobistej rozmowy z trenerem (poza systemem elektronicznym).

(diety polecane przez siłownie - dane wyświetlane dynamicznie z bazy danych)

Analiza postępów kursanta ma miejsce poza systemem elektronicznym.

**Wymagania do uruchomienia aplikacji:**

-system (np. mac os x, linux, windows)

-interpreter języka ruby

-framework: ruby on rails

-przeglądarka internetowa (polecany Firefox lub Chrome)

-dostęp do terminala

-zainstalowane gemy z aplikacji w odpowiednich wersjach

-baza danych: sqlite (dla wersji developerskiej) oraz postgreSQL dla produkcyjnej

-(przydatne, nie wymagane) system wersjonowania git

**Potrzebne/Wykorzystane programy:**

-Git - system wersjonowania

-Rvm

-interpreter języka ruby

-rails

-postgresql - produkcyjna baza danych

-sqlite - developerska, lokalna baza danych

-rubyMine - IDE programistyczne

-vim - normalny edytor tekstu

-firefox - przeglądarka

-pentadactyl - potężna wtyczka do szybkiej obsługi firefoxa

-firebug - standardowa funkcjonalność FF (a także chroma itp.) pomagająca webdeveloperą w modelowaniu wyglądu strony. Wybieramy element wizualnie i w łatwy sposób widzimy, do jakiej klasy on należy, jakie ma id, jakie reguły z arkusza styli są do niego zastosowane (w takim sensie pomaga programistą aplikacji webowych) itd.

-Microsoft Word - do stworzenia dokumentacji

-heroku - platforma webowa obsługująca deploying aplikacji RoR

**Zastosowana platforma:**

-komputer lokalny: Ubuntu Linux (Linux lastop 3.13.0-35-generic #62-Ubuntu), 4GB ramu, ok. 2.3 GHz, 2 rdzenie, Intel Core 2 Duo, Asus LV50N

-platforma webowa: heroku

**Skrótowa instrukcja (jak uruchomić aplikacje lokalnie):**

-(jeśli nie mamy) instalacja przeglądarki internetowej (polecane FireFox lub Chrome)

-instalacja interpretera rubiego i railsów (w zależności od systemu). Dla windowsa polecam rails installer: http://railsinstaller.org/en

-(opcjonalnie) instalacje gita

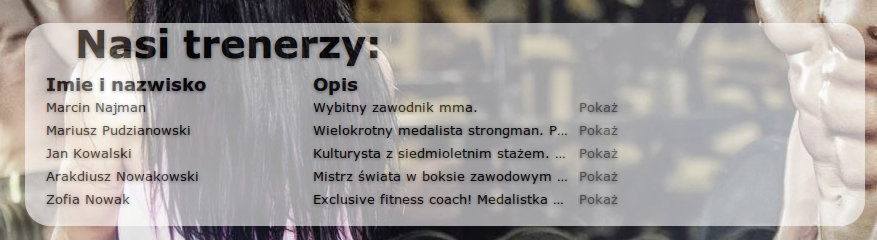
-wchodzimy do katalogu z aplikacją (katalog należy rozpakować z archiwum)

-w terminalu (dla lokazjiacji folderu aplikacji) piszemy: rails s

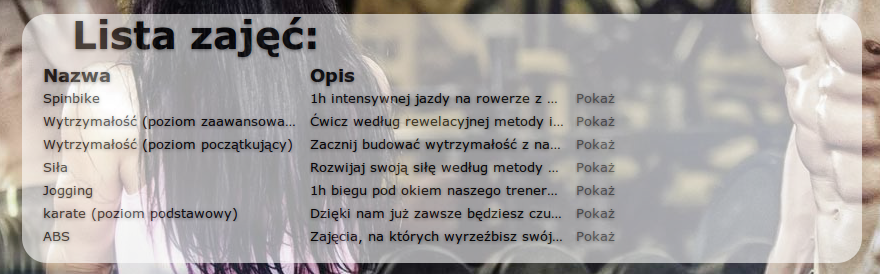
-wchodzimy w: http://127.0.0.1:3000

-podziwiamy portal

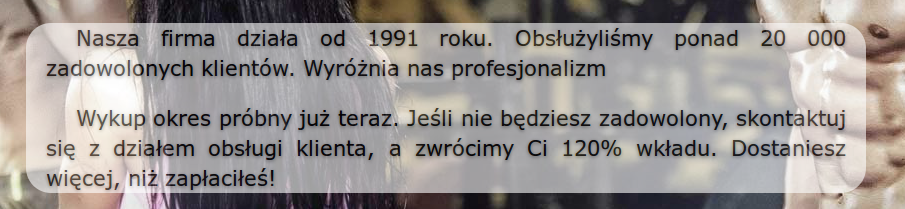
**Dodatkowe zrzuty ekranu:**



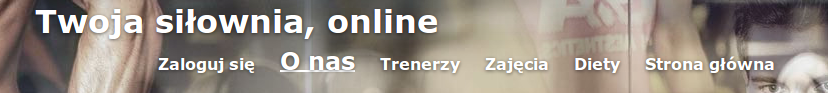
(licencjonowani trenerzy naszej siłowni wraz z opisem)



(przykładowe zajęcia oferowane przez siłownie)



(krótki opis siłowni)



(menu główne w formie bloku z dynamicznie zmieniającym się “wskazaniem” na aktualne podstronę)

**Szczegółowa informacja jak jest zbudowana aplikacja**

jest to typowa aplikacja we frameworku ruby on rails. Struktura katalogów jest standardowa.

Opisuje ją szczegółowo w pliku README.md.

Z rzeczy, na które należy zwrócić uwagę, gdyż zostały wygenerowane w trakcie tworzenia projektu (oczywiscie przy użyciu narzędzia rails generate scaffold/model/controller):

-scaffold (model + controller + widoki) dla:

--użytkownika (user)

--diet (diet)

--dni tygodnia (weekday) - prosta tabela zawierające polskie nazwy dni tygodnia

--trenera (coach)

--zajęcia (activity) - np. bieganie z krótkim opisem

--zajęcia wpisane w planie (activity\_record3) - tabela łącząca id użytkownika, trenera oraz dnia tygodnia wraz z wpisem godzinowym

-controller

--sessions (sesje) - do obsługi system logowania

Oczywiście, w skończonej aplikacji nie zawsze wszystkie wygenerowane przez funkcje scaffold funkcjonalności będą używane, a wiele jest zmodyfikowanych.

**Opis modeli w bazie danych:**

Tabele:

-activities: opisuje czynności/aktywności fizyczne np. spinbike, bieganie (krótki opis + dłuższy)

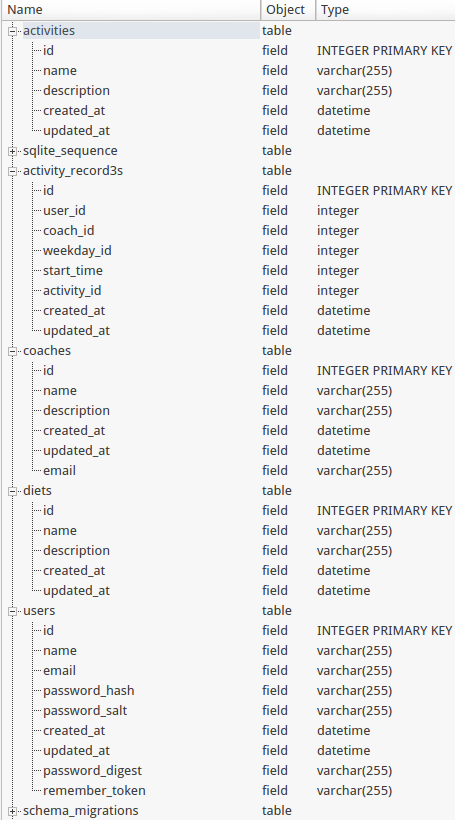
-activity\_record3: stanowi pojedynczy wpis w dzienniku kursanta. Łączy krusanta z trenerem i konkretną aktywnościa dla danej daty.

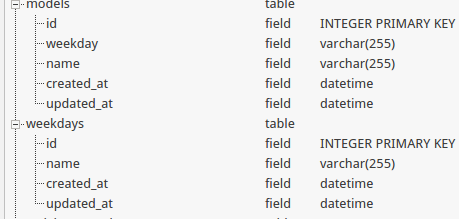
-coaches: trenerzy: imie, nazwisko, opis, email

-diety: opis diet (nazwa + krótki opis)

-users: odpowiada za użytkowników. Standardowe, podstawowe informacje (imie, nazwisko, email, hasło)

-weekdays: tłumaczenie nazw dni tygodnia na język polski





Przy czym kolumny created\_at oraz updated\_at są stworzone automatycznie przez RoR i są transparentne dla programisty.

Wycinki ekranu są zrobione przez program shutter i pochodzą z programu SQLLite, który wyświetla developerską strukturę bazy danych. Produkcyjna baza stoi na postreSQL.

**Jak działa aplikacja?**

Działanie opiera się na dobrze poznanym i szeroko stosowanym wzorcu projektowym: model-widok-kontroler.

Pozwolę sobie opisać jego działanie na przykładnie naszej aplikacji:

kiedy użytkownik pierwszy raz otwiera stronę, aplikacja szuka w pliku konfiguracyjnym routes.rb, jaki kontroler i jaka funkcja (a poprzez funkcje także i widok) odpowiada danej ścieżce (w tym przypadku ścieżce głównej, domyślnej). Ta sama procedura tyczy się każdej innej ścieżki (istotnie, wszystkie dozwolone ścieżki wraz z metodami dostępu są wyszczegółowione w pliku routes.rb).

Następnie akcja zostaje przekazana do odpowiedniego kontrolera i odpowiedniej funkcji. Rozpoczyna się kalkulacja niezbędnych parakterów do wyświetlenia strony (charakterystyczny mechanizm np. dla django albo RoR) - kalkulowanie logiki podstrony aplikacji. Po zakończeniu, zostajemy przekierowani do konkretnego widoku, który może kożystać z wyliczonych wcześniej zmeinnych.

Istnieje jeden główny schemat wyglądu widoków (applications.html.erb) - podmieniane są tylko bloki treści/kontekstu (ramka po środku strony) oraz tytuł.

Więcej informacji na temat działania wzorca projektowego model-widok-kontroler w aplikacji pisanej pod Ruby on Rails można znaleźć np. tutaj: https://www.railstutorial.org/book/demo\_app#sec-mvc\_in\_action

**Które cześći kodu pochodzą z innych źródeł?**

Część kodu została wygenerowana automatycznie (patrz: rails generate ...) (opisywałam to w części dot. standardowej struktury katalogów).

Wspierałam się także książką dostępna na railstutorial.org (możliwe, że można znaleźć trochę podobieństw do zaprezentowanej tam aplikacji) oraz stackoverflow. Nie mogę jednak powiedzieć, jakoby jakieś fragmenty były bezmyślnie kopiowane. Cały kod (nie wygenerowany automatycznie) napisałam własnoręcznie w oparciu o powyższe materiały.

Grafika tła pochodzi z serwisu flickr (dla hasła “gym workout”) i prawa autorskie pozwalają jej obróbke i dystrybucje.

Poniżej opisuje najważniejsze schematy:

app/assets/javascripts - pliki wygenerowane

app/assets/stylesheets - pliki wygenerowane, custom.css.scss w dużej części podobny do proponowanego na railsturorial.org, własne modyfikacje

app/assets/mailers - standarowe

kontrolery, widoki, modele, migracje - własnoręcznie lub modyfikowane po szablonowym utworzeniu

config/routes.rb - plik standardowo generowany, lecz w dużej części modyfikowany

lib, db (poza migracjami), log, public - standardowe

spec - własnoręcznie w oparciu o railtutorial

Generalnie, wszystkie pliki, które nie są wygenerowane automatycznie przez narzedzia dostarczone do RoR (a więc pliki standardowe - o których wspominałam wcześniej przy okazji standardowej struktury aplikacji) zostały utworzone przeze mnie. Ponadto, modyfikowałam większość plików standardowych, aby dostosować je pod moją aplikacje.

**Szczegółowa dokumentacja:**

Etap 1

Nauka rubyiego I frameworku ruby on rails.

<http://www.railstutorial.org/book>

Instalacja oraz konfiguracja niezbędnego oprogramowania

Etap 2

Wizualizacja początkowa projektu; wstępny zarys; decyzje architektoniczne.

Etap 3

Skrótowy opis działań:

-stworzenie pustej aplikacji railsowej (rails new gym)

-skonfigurowanie Gemfile

-założenie repozytorium git I powiązanie z githubem

-dostosowanie gitignore (wyspecyfikowanie, które pliki mają być automatycznie dodawane, gdy wywołamy komendę git add na zawierającym je katalogu, a które pliki mają być ominięte. Nie chcemy dodawać do repozytorium np. plików binarnych)

-ustawienie frameworków do testów automatycznych (rspec, spork, guard) oraz TDD/BDD (Capybara, Cucumber)

-powiązanie repozytorium z heroku, wysłanie strony na publiczny serwer

-stworzenie stron statycznych (główna, o nas)

-dostosowanie wyglądu (scss, twitter bootstrap), dobranie obrazów

-napisanie testów do stron statycznych sprawdzających zawartość

-wygenerowanie plików konfiguracyjnych (modelu, kotrolera, …) dla diet, zajęć oraz trenerów (rails generate scaffold …)

-dostosowanie praw dostępu do wygenerowanych stron (routes.rb)

-wpisanie przykładowych informacji na publiczny serwer heroku do bazy postgresql

-zablokowanie praw edycji od strony anonimowego użytkownika

-obsługa logowania

-obsługa gravataru

-obsługa planu zajęć

**Wnioski:**

Po długiej pracy udało się stworzyć portal dokładnie według założeń projektowych.

**Pozdrawiam,**

Małgorzata Szwed