



Ruby on Rails

WPROWADZENIE



**Fundusze
Europejskie**
Wiedza Edukacja Rozwój

Unia Europejska
Europejski Fundusz Społeczny



Wymagania

Ruby 2.5.5.1: <https://rubyinstaller.org/downloads/>

RubyMine: <https://www.jetbrains.com/ruby/>

Postgres: <https://www.postgresql.org/download/windows/>

Utworzenie projektu

Uruchamiamy RubyMine

Tworzymy nowy projekt

Wybieramy Rails Application

Rails Version -> Install Rails Gem...

Wybieramy wersję 5.2.3

Zaznaczamy „Preconfigure for selected database”

Wybieramy postgresql

Konfiguracja Bazy Danych

1. Otwieramy plik config/database.yml
2. W sekcji default dodajemy wpisy z informacją o nazwie użytkownika i haśle, podanym podczas instalacji postgresql
3. Alternatywnie możemy utworzyć nowego użytkownika w postgresql

```
default: &default  
  [...]  
username: postgres  
password: admin  
  [...]
```

Konfiguracja Bazy Danych c.d.

1. Tworzymy bazę danych poleceniem `rake db:create`
2. Tools->Run Rake Task...
3. Wpisujemy `db:create` i zatwierdzamy

Uruchomienie

1. Run->Run->Development: [NazwaProjektu]
2. Otwieramy przeglądarkę i wchodzimy pod adres localhost:3000
3. Powinniśmy zobaczyć podobną stronę:



Yay! You're on Rails!



Rails version: 5.2.3

Ruby version: 2.5.3 (x64-mingw32)

Pliki projektu – katalogi

app – zawiera główne pliki aplikacji, m.in. kontrolery, modele i widoki

bin – pliki binarne Ruby on Rails

config – pliki konfiguracyjne aplikacji, bazy danych i niektórych bibliotek

db – pliki związane z bazą danych, m.in. pliki migracji

lib – dodatkowe biblioteki

log – katalog logów aplikacji

public – specjalny katalog, pliki w nim umieszczone będą publicznie udostępnione w aplikacji webowej

test – testy jednostkowe aplikacji

tmp – pliki tymczasowe

vendor – dodatkowe pliki od odrębnych dostawców

Zarządzanie zależnościami - biblioteki



Biblioteki nazywamy
gemami



Wszystkie automatycznie
zarządzane zależności
wpisujemy do pliku
Gemfile



Aby zainstalować
zależności uruchamiamy
polecenie „bundle
install”



Tools->Bundler->Install

Katalog aplikacji - app

assets – zawiera tzw. assety: pliki Javascript, CSS, grafiki itp.

controllers – pliki kontrolerów

helpers – dodatkowe moduły, w których umieszczamy funkcje pomocnicze

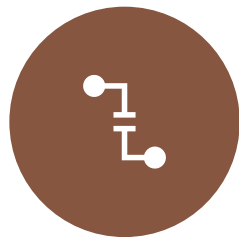
mailers – klasy do zarządzania wysyłaniem e-maili

models – klasy modeli

views – pliki widoków, domyślnie w formacie html.erb



ENVIRONMENTS
PLIKI KONFIGURACYJNE
KAŻDEGO ZE ŚRODOWISK:
PRODUKCYJNE,
DEWELOPERSKIE, TESTOWE



INITIALIZERS
PLIKI KONFIGURACYJNE
POWIĄZANE Z NIEKTÓRYMI
BIBLIOTEKAMI



LOCALES
PLIKI TŁUMACZEŃ DLA
KAŻDEGO JĘZYKA
OBSŁUGIWANEGO PRZEZ
APLIKACJĘ



APPLICATION.RB
USTAWIENIA APLIKACJI
WSPÓLNE DLA WSZYSTKICH
ŚRODOWISK



DATABASE.YML
KONFIGURACJA POŁĄCZENIA Z
BAZĄ DANYCH



ROUTES.RB
USTAWIENIA DOSTĘPNYCH
ENDPOINTÓW

Pliki konfiguracyjne

Wzorzec MVC

Model-View-Controller

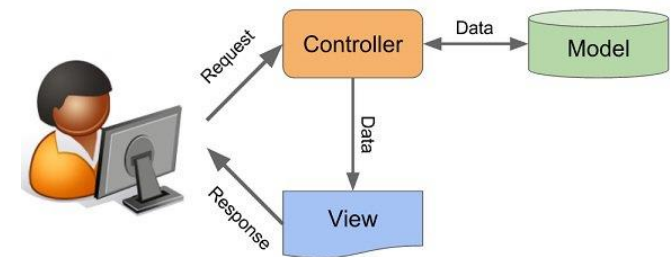
Jeden z popularnych wzorców projektowych, wykorzystywany w aplikacjach Ruby on Rails

Określa zasady tworzenia aplikacji

Kontroler otrzymuje zapytanie od użytkownika, komunikuje się z Modelem w celu pobrania danych, a następnie przekazuje je do Widoku, który wyświetla dane użytkownikowi

Tylko Kontroler powinien komunikować się z Modelem

Brak komunikacji pomiędzy Widokiem a Modelem



Generatory

Ruby on Rails posiada wiele przydatnych generatorów, znacznie ułatwiających i przyspieszających pracę

Pozwalają one wygenerować często używane pliki i fragmenty kodu, takie jak:

- Kontrolery
- Widoki
- Modele
- Migracje

Scaffold

Tools -> Run Rails Generator...

Wpisujemy *scaffold* i zatwierdzamy

Podajemy nazwę modelu i jego pola

Utworzymy model Studenta z następującymi polami: imię, nazwisko, wiek, data urodzenia

Wpisujemy:

Student first_name:string surname:string age:integer birthdate:date

Zatwierdzamy

Scaffold

W logach możemy zobaczyć, jakie pliki zostały utworzone

Można je podzielić na kilka kategorii:

- Plik migracji
- Plik kontrolera
- Plik modelu
- Pliki widoków
- Pliki skryptów JavaScript oraz CSS
- Pliki testów jednostkowych

Migracja

`/db/migrate/[data]_create_students.rb`

Określa operacje, jakie mają być przeprowadzone na bazie danych

Zawiera nazwę nowej tabeli (w liczbie mnogiej), a także nazwy i typy wszystkich pól (kolumn)

Strukturę bazy danych powinniśmy zmieniać wyłącznie za pomocą migracji

Model

`/app/models/student.rb`

Zawiera definicję klasy modelu

Domyślnie jest pusta – wszystkie pola powiązane są z bazą danych

Tutaj możemy dodawać ograniczenia na pola, a także metody

Kontroler

`/app/controllers/students_controller.rb`

Zawiera definicję dostępnych metod – endpointów

Domyślnie zawiera metody pozwalające na:

- Wypisanie wszystkich studentów
- Dodanie nowego studenta
- Edycję studenta
- Wyświetlenie studenta o podanym identyfikatorze
- Usunięcie studenta

Widoki

`/app/views/students/`

Definiują to, co użytkownik może zobaczyć i zrobić w aplikacji

Określają, co ma zostać wyświetlone na konkretnych stronach

Automatycznie wygenerowane pozwalają na wykonywanie operacji opisanych w kontrolerze

Automatycznie wygenerowany format nie jest ładny, ale posiada wszystkie potrzebne informacje

Routes

/config/routes.rb

Został dodany jeden wpis:

```
resources :students
```

Oznacza, że w aplikacji dostępny jest „zasób” studenci, który obsługuje wszystkie podstawowe metody (dodawanie, wyświetlanie, edycja itp.)

Adresy są automatycznie wygenerowane i „ukryte” pod tym zapisem

Uruchomienie

Zanim sprawdzimy działanie aplikacji, musimy uruchomić migrację na bazie danych

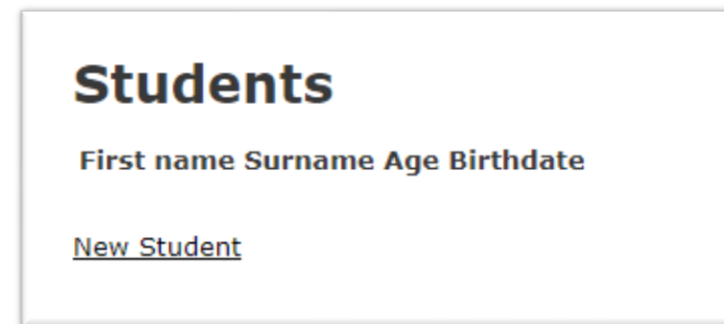
Tools->Run Rake Task...->db:migrate

W bazie danych zostanie utworzona nowa tabela Students

Uruchamiamy aplikację: Run->Run->Development: [NazwaProjektu]

Wchodzimy pod adres <http://localhost:3000/students>

Powinniśmy zobaczyć coś podobnego:



Dodanie studenta

Klikamy [New Student](#)

Podajemy dane nowego studenta i klikamy *Create Student*

Wracamy do listy przyciskiem *back*

Powinniśmy zobaczyć nowo dodanego studenta na liście:

Students			
First name	Surname	Age	Birthdate
Jan	Kowalski	22	2014-07-22

[Show](#) [Edit](#) [Destroy](#)

[New Student](#)

Określanie wymagań na pola

Otwieramy plik modelu (/app/models/student.rb)

Dodajmy wymaganie na minimalną i maksymalną długość imienia

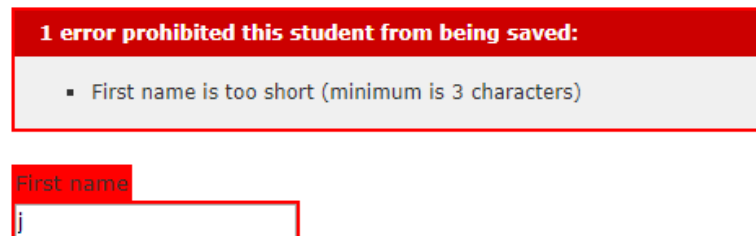
Określmy minimalną na 3, a maksymalną na 30

Wewnątrz klasy dopisujemy:

```
validates_length_of :first_name, minimum: 3, maximum: 30
```

Spróbujmy teraz ponownie utworzyć studenta, ale z imieniem krótszym niż 3 znaki

Zakończy się to komunikatem błędu:



1 error prohibited this student from being saved:

- First name is too short (minimum is 3 characters)

First name

Wymaganie podania nazwiska

Obecnie możemy utworzyć studenta, nie podając jego nazwiska, co może być niewskazane w pewnych sytuacjach

Dodajmy wymaganie podania nazwiska do klasy modelu:

```
validates :surname, presence: true
```

Teraz próba utworzenia studenta bez podania nazwiska zakończy się niepowodzeniem:

1 error prohibited this student from being saved:

- Surname can't be blank

First name

Surname

Obliczanie wieku studenta

Gdy data urodzenia jest podana, wiek studenta może zostać obliczony automatycznie

Dodajmy metodę, która policzy wiek studenta na podstawie jego daty urodzenia

Zapiszemy ją w klasie modelu studenta:

```
def compute_age
  self.age = ((Time.zone.now - birthdate.to_time) / 1.year.seconds).floor
end
```

Teraz musimy ją wywołać podczas zapisywania danych nowego studenta:

```
before_save :compute_age
```

Teraz możemy sprawdzić, czy nowy student będzie miał automatycznie policzony wiek

Studenci o podanym wieku

Dodajmy metodę, która pozwoli nam wypisać wszystkich studentów w podanym wieku

Metodę dodamy w klasie kontrolera (/app/controllers/students_controller.rb)

Metoda przyjmie jeden parametr – poszukiwany wiek

W metodach kontrolera parametry pobieramy z zapytania – z hasha params:

```
def search_by_age  
  desired_age = params[:age]  
  
end
```

Studenci o podanym wieku c.d.

Znając wymagany wiek, możemy wyszukać studentów w bazie danych:

```
def search_by_age
  desired_age = params[:age]

  students = Student.where(age: desired_age)
end
```

Studenci o podanym wieku c.d.

Teraz pozostało zwrócić znalezionych studentów

Zwrócimy ich w formacie json – jest to format, który posłuży do komunikacji z aplikacją mobilną

```
def search_by_age
  desired_age = params[:age]

  students = Student.where(age: desired_age)

  render json: students
end
```

Studenci o podanym wieku c.d.

Aby przetestować nową metodę, musimy dodać ją do pliku routes.rb

Najpierw podajemy typ metody: GET, POST, PUT, DELETE

Następnie jej adres – endpoint

Na końcu podajemy, jaka metoda z kontrolera ma zostać wywołana

Ważne, aby nowy wpis dodać **przed wpisem** *resources* **:students**

```
get "students/search_by_age" => "students#search_by_age"
```

Studenci o podanym wieku c.d.

Sprawdzamy działanie metody

W tym celu najpierw utworzymy kilku studentów o różnym wieku

Następnie wchodzimy pod adres:

http://localhost:3000/students/search_by_age?age=5

Powinniśmy zobaczyć wszystkich studentów w wieku 5 lat:

```
[{"id":4,"first_name":"Jan","surname":"Kowalewski","age":5,"birthdate":"2014-07-22","created_at":"2019-07-22T10:45:26.468Z","updated_at":"2019-07-22T10:51:34.543Z"}]
```

Co dalej

Każdy szanujący się serwis webowy powinien mieć obsługę logowania

Nie ma sensu pisać tego samodzielnie, do tego istnieją biblioteki, np. Devise:

<https://github.com/plataformatec/devise>

Warto także dodać panel administracyjny, aby w łatwy sposób zarządzać danymi aplikacji:

https://github.com/sferik/rails_admin

<https://i.stack.imgur.com/jKOn7.jpg>

Źródła