

## **Программа обучения стажеров на базе ROR.**

1. ВВЕДЕНИЕ.
2. ОБУЧЕНИЕ ОСНОВАМ.
  - 2.1. Знакомство с Ruby.
  - 2.2. Знакомство с Rails.
3. ПОСТРОЕНИЕ ENGLI.
  - 3.1. Создание приложение Rails.
  - 3.2. Создания контроллера.
  - 3.3. Создание первой страницы.
  - 3.4. Создание первой модели.
  - 3.5. Создание кастомных миграций.
  - 3.6. Немного о gems.
  - 3.7. Добавление пользователя.
  - 3.8. Знакомство с helper.
  - 3.9. Ассоциации.
  - 3.10. Еще один полезный gem.
  - 3.11. Добавление примеров к фразам.
  - 3.12. Накидаем фразе лайков!
  - 3.13. Дадим авторам понять что они клевые.
4. ЗАКЛЮЧЕНИЕ

## **ВВЕДЕНИЕ**

Эта программа обучения рассчитана на обучение базового функционала и синтаксиса построения приложений на Ruby on Rails.

По окончании курса вы будете иметь понятие о синтаксисе Ruby и архитектуре приложений Rails.

Идея приложения очень проста, помогать группе людей совершенствовать свой разговорный английский, а именно это будет набор фразовых выражений, которые будут иметь категории, примеры, лайки.

Итог данного курса будет работающее ROR приложение в production (Heroku).

Пример финального приложения:

<https://dunice-trainees-rails-cource.herokuapp.com>

## ОБУЧЕНИЕ ОСНОВАМ

### Знакомство с Ruby.

Есть прекрасный ресурс который поможет разобраться в синтаксисе Ruby - <https://www.codecademy.com/learn/ruby>

### Знакомство с Rails.

На этом же ресурсе можно ознакомиться с основными возможностями Rails - <https://www.codecademy.com/learn/learn-rails>

## ПОСТРОЕНИЕ ENGLI.

### Создание приложение Rails.

Для начала вам нужно установить ruby на вашу машину и git, если вы этого не сделали раньше.

После этого установить gem rails глобально.

Во всем этом вам поможет данный ресурс - [http://guides.rubyonrails.org/getting\\_started.html](http://guides.rubyonrails.org/getting_started.html), но есть одно замечание, нам не нужен sqlite3, мы используем postgresql. Следовательно вам нужно установить его на вашу машину, если вы этого не сделали ранее. После установления postgresql вы можете использовать замечательную команду которая создаст Rails приложение с postgresql db.

После проделанных операций вам нужно открыть app/config/database.yml и изменить default на

**default:** &default

**adapter:** postgresql

**encoding:** unicode

**user:** имя вашего postgres пользователя

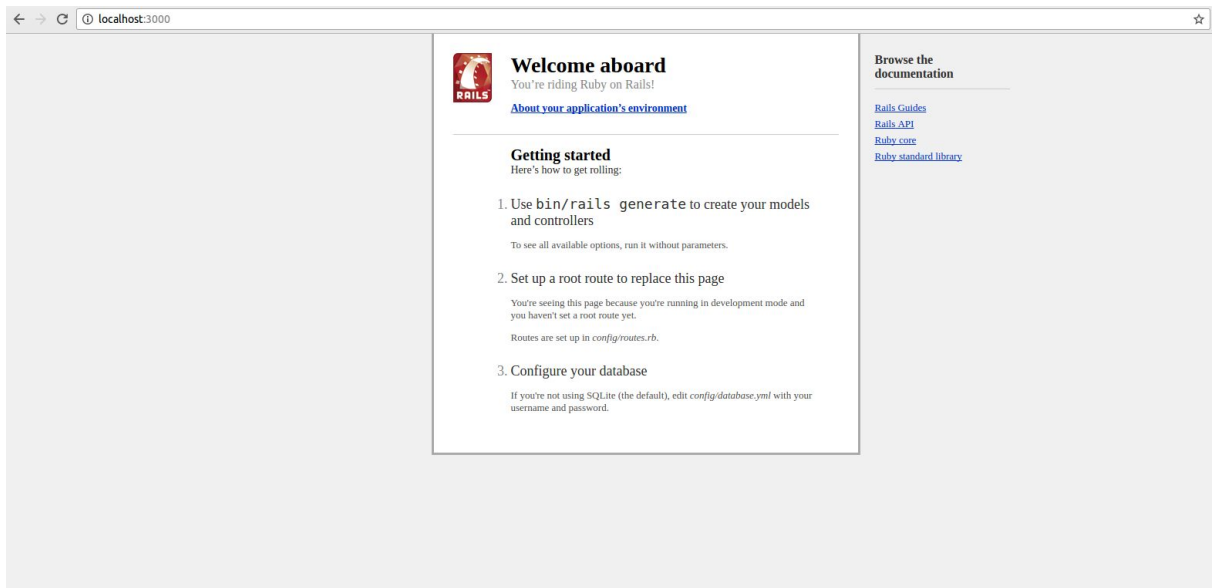
**password:** пароль вашего postgres пользователя

**pool:** 5

После чего создать базы данных командой rake db:create.

В итоге мы получили шаблон приложения с созданными датабазами (Rails создает две базы, первая development вторая test).

Теперь вы можете увидеть плоды своих трудов, запустив команду rails s (rails start) и зайдя в браузер ввести в адресную строку <http://localhost:3000/>, вы должны получить вот такую страницу.



## Создание контроллера.

Первой целью будет создание статических страниц, для этого нам нужно создать контроллер StaticPagesController и action hello.

`rails g controller static_pages hello` поможет вам в этом.

Подробнее о команде rails generate (g):

[http://guides.rubyonrails.org/command\\_line.html](http://guides.rubyonrails.org/command_line.html).

Посмотрев на созданные файлы можно увидеть созданный файл `static_pages_helper.rb` в папке `helper`, о котором мы поговорим позже, и папку `static_pages` в папке `views` в которой есть файл `hello.html.erb`.

## Создание первой страницы.

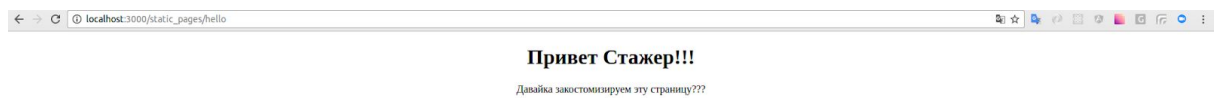
За отображение контента в Rails отвечают `views`, в нашем случае это `hello.html.erb`.

Чтобы страница отображалась в браузере нужно прописать к ней путь, для этого откройте файл `routes.rb` в папке `config` и вы можете увидеть уже созданный для этого путь `get 'static_pages/hello'`.

Теперь можно посмотреть что получилось по адресу [http://localhost:3000/static\\_pages/hello](http://localhost:3000/static_pages/hello).

Нам осталось добавить нужный контент к странице и все готово. Для этого в папке `app/views` откройте файл `hello.html.erb` и поменяйте его содержимое на удобное вам.

У меня получилось так



А теперь тебе нужно сделать эту страницу корневой.

### Создание первой модели.

Создание простых страниц мы не ограничимся, нам нужно иметь таблицу всех фраз и страницу к ней.

Для этого нам нужно сгенерировать модель, для фраз.

```
rails g model phrase phrase:string translation:string
```

Создаст модель Phrases и миграцию для создания таблицы phrases в нашей базе.

А теперь зайдите в файл миграции и посмотрите на синтаксис.

Подробнее о моделях:

[http://guides.rubyonrails.org/active\\_record\\_basics.html](http://guides.rubyonrails.org/active_record_basics.html)

Подробнее о миграциях:

[http://edgeguides.rubyonrails.org/active\\_record\\_migrations.html](http://edgeguides.rubyonrails.org/active_record_migrations.html)

После чего нужно запустить миграции, для создания таблицы в нашей базе командой `rake db:migrate`.

Теперь вы можете открыть базу используя VStudio и посмотреть на структуру.

Теперь нам нужно создать контроллер с actions `:new` `:create` `:index` для этого, под название `phrases`.

После этого создать пути к нашим action via config/routes

<http://guides.rubyonrails.org/routing.html>

Теперь нам нужно создать форму для создания наших фраз, для этого файл `views/new.html.erb` переделаите в такой вид

```
<%= form_for(@phrase) do |f| %>
  <% if @phrase.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@phrase.errors.count, "error") %> prohibited this phrase from being saved:</h2>

      <ul>
        <% @phrase.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :phrase %><br>
    <%= f.text_field :phrase %>
  </div>
  <div class="field">
    <%= f.label :translation %><br>
    <%= f.text_field :translation %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Теперь, при попытке посетить <http://localhost:3000/phrases/new> мы должны ловить ошибку, это происходит потому что переменная `@phrase` не определена, все что нам нужно это в action `:new` в `PhrasesController` объявить ее

```
def new
  @phrase = Phrase.new()
end
```

Теперь вы должны получить следующее



The screenshot shows a web browser window with the address bar displaying `localhost:3000/phrases/new`. Below the address bar, there is a form with two input fields. The first field is labeled "Phrase" and the second field is labeled "Translation". Below these fields is a button labeled "Create Phrase".

Теперь нам осталось написать логику для создания, обычно это выглядит так:

```

class PhrasesController < ApplicationController
  def index
    end

  def new
    @phrase = Phrase.new()
    end

  def create
    @phrase = Phrase.new(phrase_params)
    if @phrase.save
      flash[:notice] = 'Phrase has been created'
      redirect_to phrases_path
    else
      flash[:danger] = @phrase.errors.full_messages.to_sentence
      render :new
    end
  end

  private

  def phrase_params
    params.require(:phrase).permit(:phrase, :translation)
  end
end

```

Я думаю логика ясна, мы делаем пробный экземпляр нашей модели и потом пробуем его сохранить. Если наша новая фраза сохранена мы показываем сообщение что она сохранена, а если нет то мы возвращаемся обратно на форму с текстом ошибки.

Теперь нужно добавить отображение *flash\_messages* на страницу для этого в `views/layouts/application.html.erb` вставим

```

<% flash.each do |key, value| %>
  <div class="alert alert-<%= key %>"><%= value %></div>
<% end %>
До
<%= yield %>

```

И теперь мы получаем сообщение с успешным сохранением:

Phrase has been created

## Phrases#index

Find me in app/views/phrases/index.html.erb

Но как можно догадаться сейчас нет никаких причин для того чтобы фраза не сохранилась, для этого нам нужно добавить валидации



Подробнее о валидациях:

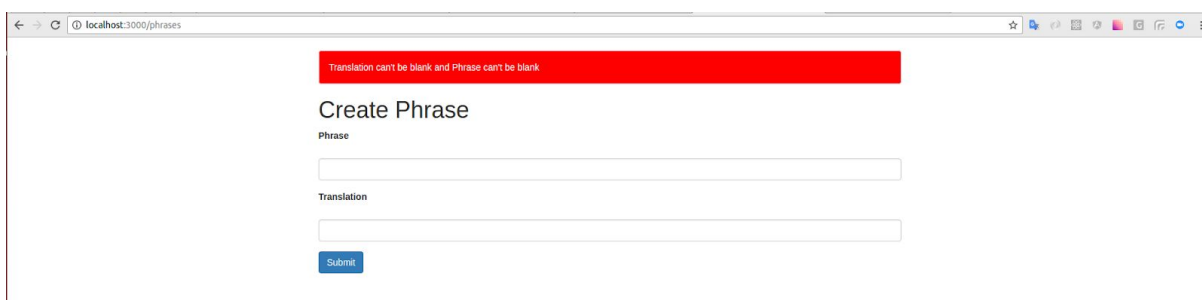
[http://guides.rubyonrails.org/active\\_record\\_validations.html](http://guides.rubyonrails.org/active_record_validations.html)

### CHALLENGE:

Это будет плохо если мы позволим сохранять фразы без самой фразы или без перевода, поэтому нужно добавить валидацию на эти два поля.

1. Добавить валидации на поля `:phrase`, `:translation` (присутствие, уникальность для фразы).
2. Установить `bootstrap` и закласовать форму.

После добавления валидаций и не долгих манипуляций со стилями при сохранении фразы с пустыми значениями мы получим:



The screenshot shows a web browser window at `localhost:3000/phrases`. At the top, a red error banner displays the message "Translation can't be blank and Phrase can't be blank". Below this, the form is titled "Create Phrase". It contains two input fields: "Phrase" and "Translation", both of which are empty. A blue "Submit" button is located at the bottom of the form.

Теперь когда мы удостоверились в валидности нашего поста, он будет сохранен, теперь, как было сказано выше, после сохранения поста мы будем переходить на страницу со всеми постами.

За отображение всех фраз у нас отвечает action `:index` в нашем контроллере, так давайте напишем запрос, обычно это делается так:

```
class PhrasesController < ApplicationController
  def index
    @phrases = Phrase.all
  end
end
```

А теперь настроим отображение их на странице, примерно так:

```

<h1>Listing phrases</h1>

<table class="table">
  <tr>
    <th>Id</th>
    <th>Phrase</th>
    <th>Translation</th>
  </tr>

  <% @phrases.each do |phrase| %>
    <tr>
      <td><%= phrase.id %></td>
      <td><%= phrase.phrase %></td>
      <td><%= phrase.translation %></td>
    </tr>
  <% end %>
</table>

```

## CHALLENGE:

Как вы можете видеть, ничего сложного нету, мы используем нашу переменную `@phrases` из контроллера, которая является Active Record Collection и делаем по ней цикл, который возвращает наши фразы, но было бы круто использовать партиал для возвращения каждой фразы.

1. Вынести отрисовку каждой фразы в партиал.
2. Сделать страницу с таблицей фраз корневой.

Подробнее о партиалах:

[http://guides.rubyonrails.org/layouts\\_and\\_rendering.html](http://guides.rubyonrails.org/layouts_and_rendering.html)

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/create\\_partial\\_for\\_phrase](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/create_partial_for_phrase)

## Создание кастомных миграций.

Теперь пришло время добавить категории к наши фразам:

- Actions
- Time
- Productivity
- Apologies
- Common

Для этого как вы уже поняли из название параграфа нам нужно создать миграцию на добавление этой колонку в нашу таблицу фраз.

Так как вы уже читали о миграциях (ссылка выше), вы должны быть в состоянии добавить новую колонку в таблицу (**NOTICE: колонка должна иметь формат integer и дефолтное значение 0**)

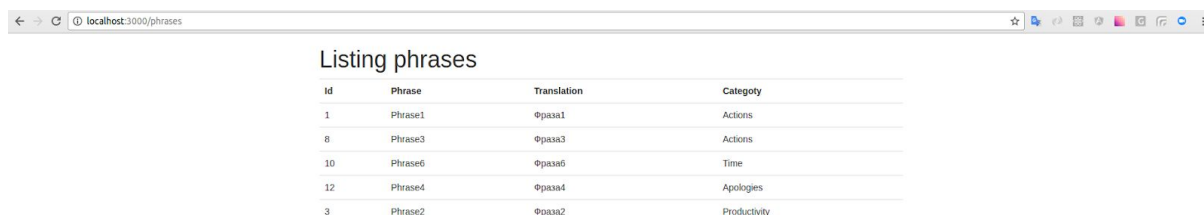
После того как вы добавили новую колонку в таблицу, вам нужно добавить новый html tag в нашу форму, это будет select.

Как только вы разобрались с тегом вам нужно добавить новый параметр в список разрешенных для создания новой фразы в PhrasesController метод :phrase\_params.

Как только вы справились с этим вам нужно написать валидацию на доступный значения для записи. (**TIP: inclusion**)

Осталось дело за малым, добавить к нашей view колонку category и значение к каждой строке в партиале. Как можно заметить, значения отображаются в цифрах, что нас не устраивает, для этого вы можете использовать :enum.

Теперь у вас таблица должна иметь похожую структуру:



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/phrases'. The page content is titled 'Listing phrases' and contains a table with four columns: 'Id', 'Phrase', 'Translation', and 'Category'. The table lists five entries with IDs 1, 8, 10, 12, and 3. The translations are 'Фраза1', 'Фраза3', 'Фраза6', 'Фраза4', and 'Фраза2' respectively. The categories are 'Actions', 'Actions', 'Time', 'Apologies', and 'Productivity'.

Id	Phrase	Translation	Category
1	Phrase1	Фраза1	Actions
8	Phrase3	Фраза3	Actions
10	Phrase6	Фраза6	Time
12	Phrase4	Фраза4	Apologies
3	Phrase2	Фраза2	Productivity

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/add\\_category\\_to\\_phrase](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/add_category_to_phrase)

## Немного о gems.

Для начала, если вы еще не имеете представления что такое RubyGem, вам следует прочесть эту статью <https://ru.wikipedia.org/wiki/RubyGems>.

На данный момент наш gemfile включает совсем немного гемов которые мы туда поместили, давайте это исправим.

Первый gem в нашем списке - **slim-rails**:

<https://github.com/slim-template/slim-rails>

Второй это - **will\_paginate**:

[https://github.com/mislav/will\\_paginate](https://github.com/mislav/will_paginate)

Он хорош в связке с этим гемом, который использует bootstrap стили для пагинации

[https://github.com/bootstrap-ruby/will\\_paginate-bootstrap](https://github.com/bootstrap-ruby/will_paginate-bootstrap)

### CHALLENGE:

Задача на этот параграф добавить эти gems в наш gemfile и начать их активно использовать. На пагинацию нужно установить дефолтное значение 10 экземпляров на страницу.

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/new\\_gems](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/new_gems)

## Добавление пользователя.

Если ты добрался до этой части, то ты уже имеешь базовое понятие о views, controller, model, migration и gem. И теперь я покажу тебе gem который поможет сделать это все “В два клика”, для создания нашего пользователя.

**gem device** - <https://github.com/plataformatec/devise>

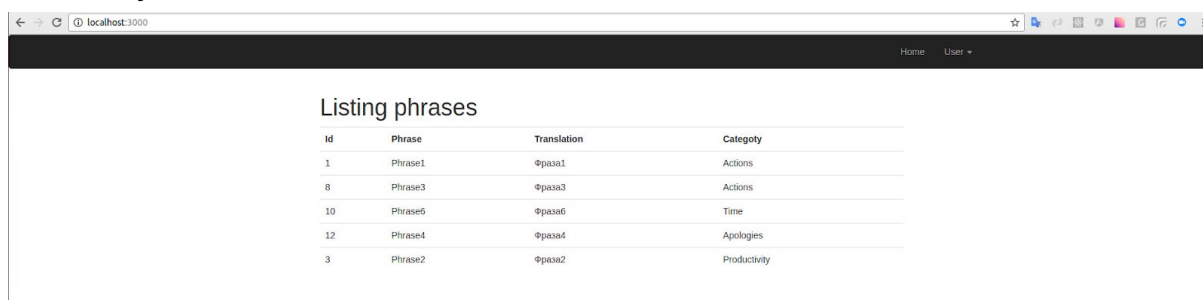
Этот гем имеет хорошую документацию, так что никакого труда тебе не составит сделать все самостоятельно, ну а если составит, то ссылка на ветку как всегда будет в конце (NOTICE: нужно добавить username к

таблицу пользователей и добавить поле в форму, ну и не забываем навесить валидацию на присутствие и уникальность)

Теперь, при попытке зайти на страницу тебя должно редиректить на логинку, это все из за строки `before_action :authenticate_user!` в `application_controller.rb` круто, да?

Но как же быть если мы захотим разлогиниться, посетить таблицу фраз или страницу hello? Постоянно вводить url в адресную строку браузера уж слишком уныло, надо сделать navbar! К слову navbar тоже должен лежать в партиале, так что приступим!

Вот что у меня вышло:



The screenshot shows a web browser window with the address bar at localhost:3000. The page has a dark header with 'Home' and 'User' links. The main content area displays a table titled 'Listing phrases'.

Id	Phrase	Translation	Category
1	Phrase1	Фраза1	Actions
8	Phrase3	Фраза3	Actions
10	Phrase6	Фраза6	Time
12	Phrase4	Фраза4	Apologies
3	Phrase2	Фраза2	Productivity

Но после того как мы разлогиниваемся нас редиректит на форму регистрации и мне откровенно не нравится ее вид, так что займемся этим.

### CHALLENGE:

1. Переделать все фомы в slim формат.
2. Создать страницу с пользователя(NOTICE: добавь просто username на страницу).
3. По той же аналогии создать страницу с таблицей всех пользователей! Не забудь добавить ссылку в навбар!

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/creation\\_users](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/creation_users)

### Знакомство с helper.

Как вы можете заметить на скрине выше таблица выглядит хорошо за одним исключением, даты действительно информативны, но нам не нужна такая детальная информация, нам нужно форматировать дату, для этого можно использовать rails helper.

Подробнее о хелперах:

<http://api.rubyonrails.org/classes/ActionController/Helpers.html>

#### CHALLENGE:

1. Создать метод `date_formatter` в `ApplicationHelper` который будет форматировать дату в формат: 'Sunday 30 Apr 2017'

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/date\\_formatter](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/date_formatter)

### Ассоциации.

Анализируя сделанную работу тебе - стажер можно гордиться собой, но это еще не все, нам нужно привязать каждую фразу к пользователю, ведь это логично чтобы у каждой фразы в нашей базе был автор, но для начала нужно немного про это почитать.

Подробнее об ассоциациях:

[http://guides.rubyonrails.org/association\\_basics.html](http://guides.rubyonrails.org/association_basics.html)

#### CHALLENGE:

1. Сгенерировать миграцию на добавление `:user_id` поля в таблицу фраз (TIP: я привык для этого пользоваться `reference` в миграциях).

После запуска миграции можно обнаружить в таблицу phrases поле :user\_id, теперь нужно объявить эту ассоциацию в моделях фраз и пользователя, после чего немного изменить логику создания фраз, примерно таким образом:

```
class PhrasesController < ApplicationController
  def index
    @phrases = Phrase.paginate(:page => params[:page])
  end

  def new
    @phrase = Phrase.new()
  end

  def create
    @phrase = current_user.phrases.new(phrase_params)
    if @phrase.save
      flash[:notice] = 'Phrase has been created'
      redirect_to root_path
    else
      flash[:danger] = @phrase.errors.full_messages.to_sentence
      render :new
    end
  end

  private

  def phrase_params
    params.require(:phrase).permit(:phrase, :translation)
  end
end
```

После этого можно добавить ссылку на форму создания фраз в нашу таблицу фраз:

Home Users User +

Phrase has been created

Listing phrases

Id	Phrase	Translation	Category
1	Phrase1	Фраза1	Actions
8	Phrase3	Фраза3	Actions
10	Phrase6	Фраза6	Time
12	Phrase4	Фраза4	Apologies
3	Phrase2	Фраза2	Productivity
20	BLAH	Болтовня	Actions
21	BLAH2	Болтовня2	Actions

Add new phrase

Ну и было бы логично отображать username автора в колонке Author(**TIP: используй includes(:user) и прочти, что собственно делает :includes**).

Ну выглядит неплохо, как по мне:

Home Users User ▾				
Listing phrases				
Id	Phrase	Translation	Category	Author
20	BLAH	Болтовня	Actions	Dmitry
21	BLAH2	Болтовня2	Actions	Dmitry
1	Phrase1	Фраза1	Actions	Dmitry
10	Phrase6	Фраза6	Time	Dmitry
3	Phrase2	Фраза2	Productivity	Dmitry
12	Phrase4	Фраза4	Apologies	Dmitry
8	Phrase3	Фраза3	Actions	Dmitry
<a href="#">Add new phrase</a>				

Но всегда же можно что то улучшить, например вместо простого отображения username пользователя можно отображать ссылку на созданную нами ранее страницу пользователя.

### CHALLENGE:

1. Добавить вместо простого username ссылку на страницу пользователя.
2. Добавить ссылку к таблице пользователей.

Я думаю у тебя это не вызвало никаких трудностей, теперь было бы логично отображать фразы пользователя, чью страницу мы посетили.

### CHALLENGE:

1. Добавить таблицу фраз к странице пользователя без отображения колонки Author(цикл по коллекции фраз должен быть в странице пользователя, но использовать парциал *views/phrases/\_phrase.slim*).

Что ж, неплохо, но что если ты понимаешь что создал фразу с неверным переводом, в этом случае я вижу два возможных решения: поменять перевод и удалить фразу.

### CHALLENGE:

1. Добавить колонку Actions к таблице фраз и напротив каждой фразы отображать edit | delete ссылки, edit должна вести к форме изменения фразы, которая после изменения должна показывать сообщение об успешном изменении и редиректить обратно на страницу пользователя, а delete должна удалить фразу и вернуть опять же на страницу пользователя и показать сообщение что фраза была успешно удалена). TIP: тебя может смутить использование одной и



той же строки: `@phrase = Phrase.find_by(id: params[:id])` в `:edit` и в `:update` методах, так что настало время написать свой `:before_action` для `PhrasesController` для определения фразы, этот экшн должен вызываться также и для `:destroy` метода, так что выгода со всех сторон.

2. Также имеет смысл вынести форму для фразы в отдельный партиал, тк мы будем использовать ту же форму и для изменения фразы.

Теперь пришло время обезопасит фразы пользователя от других пользователей, в этом долгом и нелегком пути мы пока можем сделать минимум, а именно добавить `:before_filter` к `:edit`, `:update` и `:destroy` методам, который будет проверять автора фразы и сравнивать его с `current_user`, для этого тебе нужно написать метод экземпляра `:is_author?` `current_user` в модели `Phrase` у меня это получилось примерно так:

Метод используемый в `before_filter`:

```
def check_user!  
  unless @phrase.author? current_user  
    flash[:danger] = 'You don\'t author of phrase, go away!'  
    redirect_to(:back)  
  end  
end
```

Метод модели:

```
def author?(user)  
  self.user == user  
end
```

Выглядит неплохо!

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/associations](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/associations)

Еще один полезный gem.

Все выглядит хорошо, кроме URL к нашей фразе, было бы круто использовать вместо `:id` - `:phrase`, не так ли? В этом несложном деле нам может помочь `gem friendly_id`:

[https://github.com/norman/friendly\\_id](https://github.com/norman/friendly_id)

### CHALLENGE:

1. Закостомизировать URL для фраз (TIP: для этого тебе придется либо удалить все фразы вручную либо вручную заполнить поле `:slug`).

### Добавление примеров к фразам.

Иметь фразу на английском и ее перевод на русском хорошо, но было бы прекрасно иметь примеры использования этой фразы в быту, но как мы знаем примеров использования фраз может быть неограниченное количество, так зачем себя ограничивать!?!

CHALLENGE: Создать модель `Example`, таблица `examples` должна содержать поля:

- `example`
- `user_id`
- `phrase_id`

Как можно заметить мы имеем поле `:phrase_id`, что собственно логично и так же мы имеем поле `:user_id`, назначение которого я объясню позже.

Наша задача сейчас заключается в том чтобы добавить отношения между фразой и примером и между примером и пользователем.

После добавления отношений мы должны изменить форму создания фразы, но только для создания, для обновления она не должна измениться, как можно догадаться нам нужно добавить поле `example`.

### CHALLENGE:

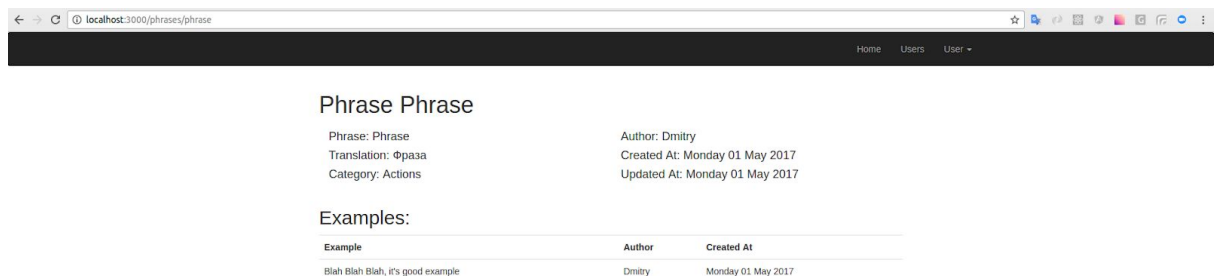
1. Добавить input в форму создания фразы, но не для изменения фразы и не позволять создание фразы без примера, user\_id для примера должен быть *current\_user* id.

Теперь было бы логично добавить страницу фразы с таблице всех примеров

### CHALLENGE:

1. Создать страницу для каждой фразы.
2. Создать простое отображение фразы, перевода, автора, категории, когда создано и когда изменено (с использованием хелпер метода для форматирования).
3. Добавить таблицу с примерами (пример, автор и дата создания).
4. Добавить пагинацию к таблице примеров.
5. Добавить ссылку на эту страницу на поле фразы в таблицах фраз.

Не судите строго, я не дизайнер:



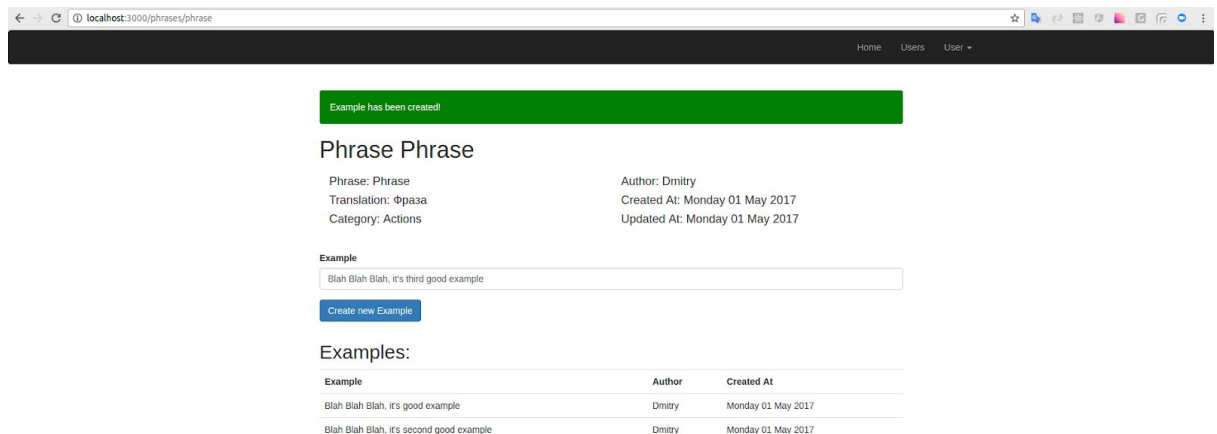
Как было сказано выше, мы должны быть не ограничены в количестве фраз, так давайте добавим форму создания фразы на страницу, выше таблицы.

### CHALLENGE:

1. Создать контроллер ExamplesController.
2. Создать action на создание.
3. Создать роут для этого(TIP: почитай про resources).
4. Валидировать поле :example по уникальности относительно каждой фразы.

5. После сабмита показывать сообщение удачное/неудачное и возвращать на эту же страницу.

У меня вышло примерно так:

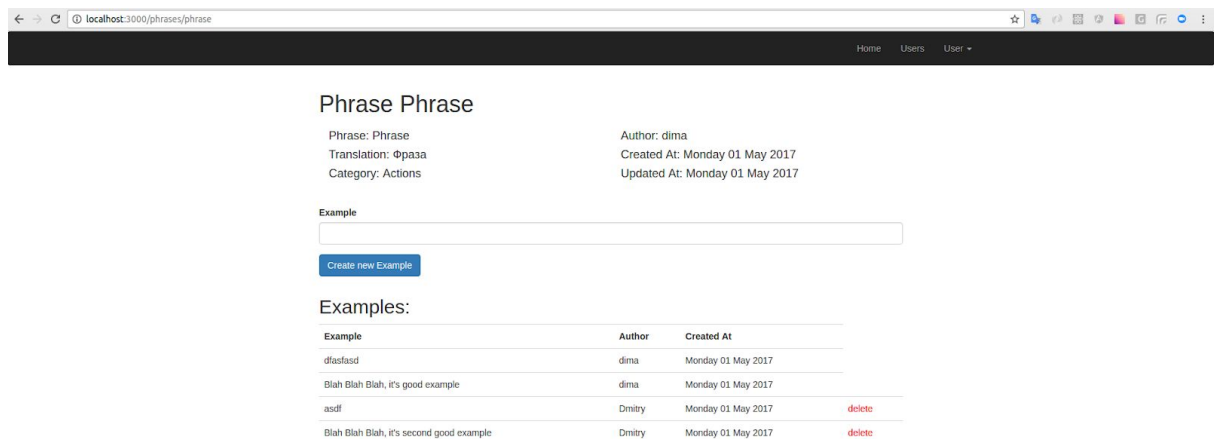


А теперь я думаю иметь возможность удалить пример была бы актуальна, но давайте дадим эту возможность автору примеру ну и конечно же автору фразы.

### CHALLENGE:

1. Создать action в ExamplesController :destroy.
2. Показывать сообщение об успешном удалении примера.
3. Создать :before\_filter для проверки что пользователь либо автор фразы либо автор примера, если нет угрожай ему что вычислишь по ip!
4. Показывать ссылку на удаление только авторам, для этого создай helper метод для проверки в ExamplesHelper, который принимает атрибут example, вы можете подумать что для определения автора фразы можно использовать метод из модели фраз и это правильно, но что делать с примером? Создавать такой же метод в модели примеров? Не не не, тут нам на помощь приходят ActiveSupport::Concern (<http://api.rubyonrails.org/classes/ActiveSupport/Concern.html>), это нам позволит вынести наш метод туда и использовать в обеих моделях (имя консерн: SharedMethods).

У меня это вышло так:



В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/add\\_examples\\_to\\_phrase](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/add_examples_to_phrase)

### Накидаем фразе лайков!

Пришло время для конкуренции в знании английского языка и мы можем добавить пользователям такую возможность, мы добавим лайки к фразам и будем отображать суммарное количество лайков в таблице пользователей, чтобы все видели кто царь горы!

Наш первый шаг поискать в сети уже готовое для этого решение, люблю готовые решения, и вот что удалось найти: `gem acts_as_votable`.

[https://github.com/ryanto/acts\\_as\\_votable](https://github.com/ryanto/acts_as_votable)

По мне отличное решение и тазов немало, начнем осваивать.

Как вы можете заметить этот `gem` генерирует свою собственную таблицу `vote` в нашей ДБ, и вы можете увидеть `:polymorphic => true` в миграции, это означает, что эта таблица будет полиморфной.

Подробнее о полиморфных таблицах:

[http://guides.rubyonrails.org/association\\_basics.html#polymorphic-associations](http://guides.rubyonrails.org/association_basics.html#polymorphic-associations)

### CHALLENGE:

1. Установить и настроить **gem acts\_as\_votable**.
2. Создать путь с `:post` запросом в контроллер `PhrasesController`.
3. Создать метод в контроллере в котором в зависимости от параметра мы будем решать лайк это или дизлайк.
4. Отрисовать голосовалку в таблицах фраз.
5. Запретить юзеру голосовать за свои фразы(`:before_filter` будет к месту).

Ну как то так:

### Listing phrases

Id	Phrase	Translation	Category	Author	Votes
35	<a href="#">Phrase</a>	Фраза	Actions	<a href="#">dima</a>	<div><div></div><div>0</div><div></div></div>
36	<a href="#">Phrase666</a>	Дьявольская фраза	Actions	<a href="#">Dmitry</a>	<div><div></div><div>0</div><div></div></div>

[Add new phrase](#)

Выглядит неплохо! Как ты можешь помнить таблицы `votes` полиморфная, так почему же нам не добавить голосовалку и к примерам тоже?

### CHALLENGE:

1. Добавить голосовалку к примерам.
2. Запретить пользователям голосовать за свои примеры.
3. Сделать отображение фраз и примеров в таблице отсортированным по лайкам.

Теперь наши фразы и примеры имеют лайки, но ценность их разная, для этого я предлагаю добавить поле `:arma` к пользователям в таблицу и учитывать каждый лайк и дизлайк.

### CHALLENGE:

1. Добавить поле :carma к пользователям(TIP: установи дефолтное значение = 0).
2. За каждый лайк/дизлайк поставленный пользователем, ему присваивается 1 carma point.
3. За каждый лайк поставленный фразе, ее автору присваивается 4 carma points.
4. За каждый дизлайк поставленный фразе, ее автору отнимается 2 carma points.
5. За каждый лайк поставленный примеру, его автору присваивается 2 carma points.
6. За каждый дизлайк поставленный примеру, у его автора отнимается 1 carma point.
7. Добавить в отображение таблицы пользователей поле карма в котором будет счет кармы, сортировать пользователей нужно по карме.

NOTICE!: вынеси метод подсчета в наш консерн, метод будет принимать параметры: лайк/дизлайк и current\_user! Так же логика для голосования в контроллерах ух должна получится схожа, так что вынесем повторяющиеся части в ApplicationController.

Вот что у меня получилось:

### Listing users

Id	Username	Email	Created At	Updated At	Carma
7	dima	ddd@bk.ru	Sunday 30 Apr 2017	Monday 01 May 2017	8
6	Dmitry	dmitriy.gusev@duniceedge.com	Sunday 30 Apr 2017	Monday 01 May 2017	4
8	test	dddd@bkddd.ru	Monday 01 May 2017	Monday 01 May 2017	0

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/likes\\_d\\_islikes](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/likes_d_islikes)

Дадим авторам понять что они клевые.

Все мы пользуемся вконтакте и всем нам приятно видеть что нас лайкают, а как мы это видим? Правильно, уведомления, так давай построим такую систему и для нашего приложения.

Для использования системы уведомлений настоятельно рекомендую **gem public\_activity**: [https://github.com/chaps-io/public\\_activity](https://github.com/chaps-io/public_activity)

#### CHALLENGE:

1. Настроить гем.
2. Дополнить таблицу activities полем readed.
3. Создать контроллер и прописать путь к :index методу.
4. В меню навбара добавить ссылку на Notifications.
5. К каждой нитификации слева добавить индикатор прочитано/не прочитано.
6. Добавить индикатор в навбар, символизирующий что у юзера есть непрочитанные уведомления.

Вышло здорово, но теперь нам надо как то после страницы обновлять поле :readed, тем самым отмечая что все уведомления были прочитаны.

#### CHALLENGE:

1. Создать и прописать в роутах action :read\_all с :put запросом который обновляет поле :readed для непрочитанный уведомления current\_user.
2. Используя coffeescript отправлять аjax на action :read\_all после посещения страницы уведомлений.

В случае фэйла вот ветка:

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_course/tree/notifications](https://github.com/DmitriyGysev/dunice_trainees_rails_course/tree/notifications)

## ЗАКЛЮЧЕНИЕ



Вы проделали большую работу, длиною примерно в 3 недели и можете собой гордиться.

Давайте подведем краткий итог пройденного материала:

1. Вы познакомились с синтаксисом Ruby и Rails на [codecademy.com](https://www.codecademy.com).
2. Вы научились создавать шаблон Rails application.
3. Познакомились с контроллерами.
4. Узнали как в Rails делать страницы.
5. Познакомились с slim синтаксисом.
6. Узнали что такое Model & Migration.
7. Познакомились с несколькими полезными gems, которых большое множество.
8. Научились как сделать авторизацию пользователей.
9. Узнали что такое Helper и с чем его едят.
10. Узнали как создавать ассоциации между моделями.
11. Как создавать инстансы с дочерними инстансами в один клик.
12. Узнали что такое полиморфные модели.
13. Познакомились с синтаксисом coffeescript.

Как можно заметить получилось 13 пунктов, как говорят ‘чертова дюжина’ :)

Но на этом не стоит останавливаться и у меня есть к тебе еще парочка заданий

### FINAL CHALLENGE:

1. Добавить friendly\_id ко всем моделям в системе.
2. Сделать несколько новых полей в таблицу пользователя (минимум 3, :string, :integer, :boolean).
3. Добавить валидации на новые поля в пользовательской таблице.
4. На данном этапе при попытке удаления фразы с лайками должна бросаться ошибка, тк фраза имеет лайки или дизлайки, так же схема и с примерами, надо это исправить.
5. Настроить подтверждение авторизации через email.

6. Настроить отправку email автору если его фразу или пример лайкнули каждый пятый раз.

Что ж, вот и настало время прощаться, спасибо за внимание!

[https://github.com/DmitriyGysev/dunice\\_trainees\\_rails\\_cource](https://github.com/DmitriyGysev/dunice_trainees_rails_cource)

**Dmitry Gusev**

Software developer Dunice Edge

<https://github.com/DmitriyGysev>

[dmitriy.gusev@duniceedge.com](mailto:dmitriy.gusev@duniceedge.com)

Skype: edge.gusev