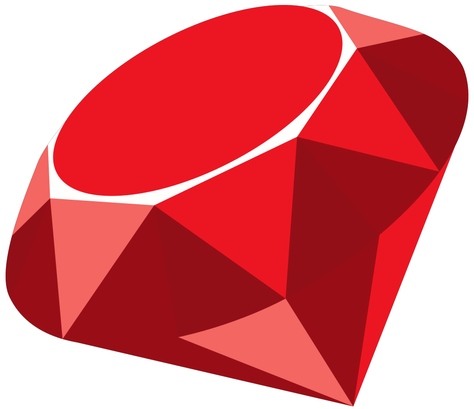
PROTOTYPING TO PRODUCTION USING RUBY ON RAILS



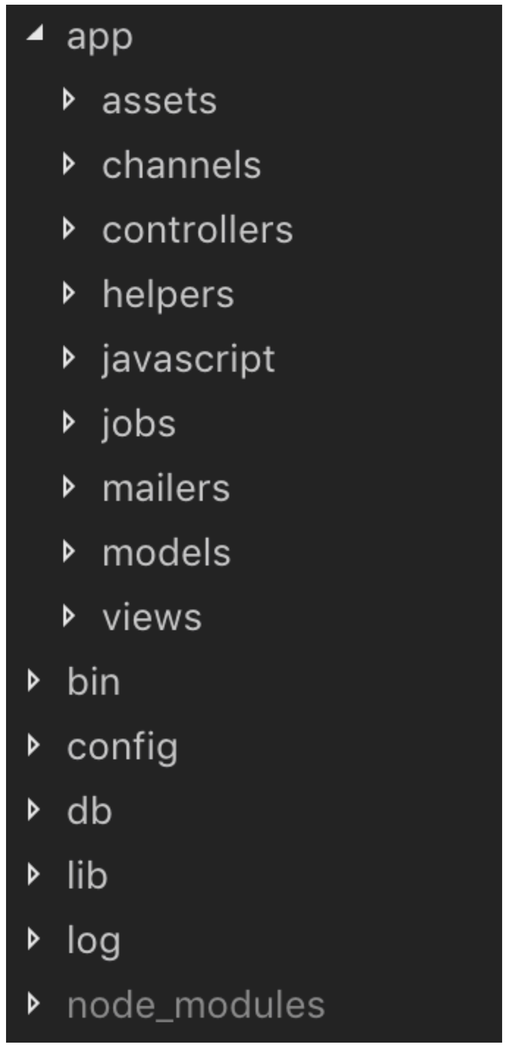
JONATHAN FLECKENSTEIN

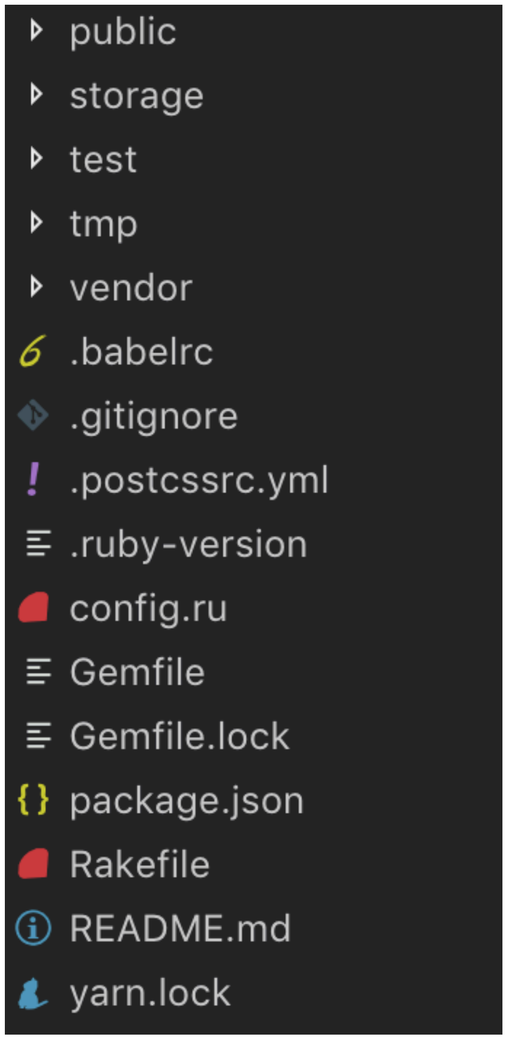
- Software Engineer
- @jonfleck
- github.com/fleck

BUILDING AN APPLICATION TO CONDUCT A SURVEY

- Create a survey
- Add answers to the survey
- Attach an image to the survey
- Create a user to take the survey

`rails new ~/Sites/cposc --database=postgresql --webpack`





```
invoke  active_record
create    db/migrate/20181124212150_create_polls.rb
create    app/models/poll.rb
invoke    test_unit
create      test/models/poll_test.rb
create      test/fixtures/polls.yml
invoke  resource_route
 route    resources :polls
invoke  scaffold_controller
create    app/controllers/polls_controller.rb
invoke    erb
create      app/views/polls
create      app/views/polls/index.html.erb
create      app/views/polls/edit.html.erb
create      app/views/polls/show.html.erb
create      app/views/polls/new.html.erb
create      app/views/polls/_form.html.erb
invoke    test_unit
create      test/controllers/polls_controller_test.rb
create      test/system/polls_test.rb
invoke    helper
create      app/helpers/polls_helper.rb
invoke      test_unit
invoke    jbuilder
create      app/views/polls/index.json.jbuilder
create      app/views/polls/show.json.jbuilder
create      app/views/polls/_poll.json.jbuilder
invoke  assets
invoke    js
create      app/assets/javascripts/polls.js
invoke    scss
create      app/assets/stylesheets/polls.scss
invoke  scss
create    app/assets/stylesheets/scaffolds.scss
```

SCAFFOLDING

- `rails g scaffold Poll name:string`
- `rails g scaffold Answer name:string poll:references`
- Routes
- Controllers with index, show, new, edit, create, update, and destroy actions
- Stubs out tests
- Gives you basic views for your actions
- Creates database migrations

LETS START OUR SERVER AND SEE WHAT WE HAVE

- `rails s`
- http://localhost:3000

MIGRATIONS

- All database changes in rails are managed through migrations
- Create our database `rails db:create`
  - Runs the SQL to create the database
- Let's run the pending migrations `rails db:migrate`
  - Updates or creates schema.rb
  - Runs SQL to creates or updates tables according to migrations

SCHEMA.RB VS MIGRATIONS

- Schema.rb exists for when you're deploying a new instance of your application (setting up new team members development environment, creating a new instance of an application for a customer). You can use it instead of replaying all of your migrations
  - Represents the current state of your database
- Migrations are for updating an existing instance of an application with the latest database changes
- Let's try starting the application again

ROUTES.RB

```ruby
Rails.application.routes.draw do
  resources :answers
  resources :polls
end
```

- http://localhost:3000/rails/info/routes
- You can also get to the same page by visiting a non existing route
- In this case nesting the routes would probably make the most sense

```ruby
Rails.application.routes.draw do
  resources :polls do
    resources :answers
  end
end
```

RAILS ACTIVE_STORAGE:INSTALL

- Generates the a migration that adds the active storage tables to the database
- Active storage makes adding files to your models easy
- Has built in adapters for:
  - S3
  - Azure Storage
  - Google Cloud Storage
  - Local Disk Storage
- Mirror Service to upload files to a secondary service

ADDING CREDENTIALS FOR A STORAGE SERVICE

- Rails ships with encrypted credential management
- `EDITOR="code --wait" rails credentials:edit`
  - Opens your encrypted credentials.yml in an editor you specify
- Put the key in an environment variable and access your credentials using
  - `Rails.application.credentials.amazon`
- Now you only have to worry about managing 1 secure environment variable when deploying code

ADDING AN IMAGE TO THE POLL

- Add this to the poll model

```
  has_one_attached :image
```

- In the view you can resize an image using variant

```
<%= image_tag @poll.image.variant(resize: 100) %>
```

- The form helper to upload the image

```
<%= form.file_field :image %>
```

WE NEED USERS TO TAKE THE POLL

- The Devise gem makes authentication simple
- Surprise! More generators:
  - `rails generate devise:install`
  - `rails generate devise user`
  - `rails db:migrate`
- http://localhost:3000/users/sign_up
- http://localhost:3000/users/edit

TAKING THE SURVEY

- `rails g scaffold Result poll:references user:references answer:references`
- `rails db:migrate`
- http://localhost:3000/results/new

LETS MAKE THIS A LITTLE MORE USER FRIENDLY

- Make a select of consisting of all our polls

```
<%= collection_select(:result, :poll_id, Poll.all, :id, :name) %>
```

- Don’t allow user_id in params

```
params.require(:result).permit(:poll_id, :user_id, :answer_id)
```

- Set the user from current_user, a helper provided by devise

```
@result.user = current_user
```

- Add a home page route and controller

```
 root to: ‘home#index’
```

```
class HomeController < ApplicationController
  def index; end
end
```

IT'S NOT A WEBSITE UNLESS IT HAS JAVASCRIPT....

- When we select a poll we'll make an AJAX call to fetch the corresponding answers for that poll
- Create file app/javascript/packs/results.js

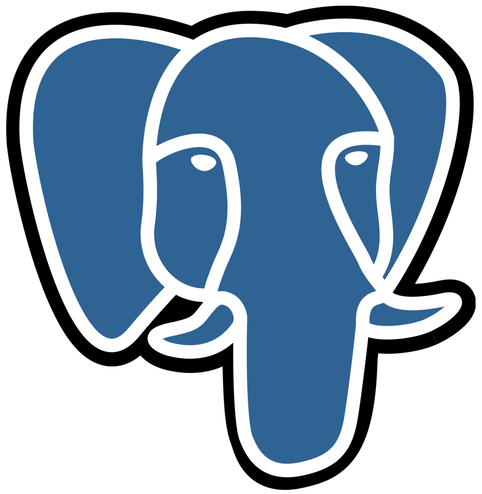
```
const updateAnswers = async (e, answers) => {
  const response = await fetch(`/polls/${e.target.value}/answers.json`)
  const newAnswers = await response.json()

  empty(answers)

  newAnswers.forEach(newAnswer => addAnswer(newAnswer, answers))
}
```

- By adding .json or setting the content-type header to application/json the model will be serialized to JSON!
- Include this file with

```
<%= javascript_pack_tag 'results' %>
```



WHY POSTGRESQL?

- Full text search (that doesn't suck)
  - Trigram, Double Metaphone and other techniques to provide good search results
- JSON columns
  - Allows you to mix NoSQL in where needed without having to run a separate NoSQL service 😎

```
SELECT * FROM users WHERE other_data->>'age' = 30;
```

```
SELECT other_data->>'age' FROM users;
```

USING POSTGRES SEARCH IN RAILS

- pg_search gem
- Search on a single model

```
pg_search_scope :search_by_name, :against => :name
```

```
Poll.search_by_name('some name')
```

- Search against multiple models

```
multisearchable :against => [:title, :author]
```

WHERE'S THE SQL?

```
<div class="field">
  <%= form.label :poll_id %>
  <%= collection_select(:result, :poll_id, Poll.all, :id, :name) %>
</div>
<div class="field">
  <%= form.label :answer_id %>
  <%= collection_select(:result, :answer_id, Poll.first.answers, :id, :name) %>
</div>
```

- Poll.all
  - SELECT "polls".* FROM “polls"
- Poll.first.answers
  - SELECT "polls".* FROM "polls" ORDER BY "polls"."id" ASC LIMIT 1
  - SELECT "answers".* FROM "answers" WHERE "answers"."poll_id" = 1
- All queries are output to console

TESTING IN RAILS

- Added method to poll model

```
  def has_answer_that_starts_with(letter)
    answers.any? { |a| a.name.first == letter }
  end
```

- Rails provides fixtures to aid in testing

FIXTURES

- test/fixtures/polls.yml

```
favorite_system_programming_language:
  name: Favorite System Programming Language
```

- test/fixtures/answers.yml

```
one:
  name: Rust
  poll: favorite_system_programming_language
two:
  name: Go
  poll: favorite_system_programming_language
```

- We can then test the poll model

```
  test "finding if any answers start with a given letter" do
    poll = polls(:favorite_system_programming_language)
    assert poll.has_answer_that_starts_with('R')
  end
```

PUBLISHING APPLICATION TO ‘PRODUCTION’

- `heroku create`
- `heroku addons:create heroku-postgresql:hobby-dev`
- `heroku config:set RAILS_MASTER_KEY=??????`
- `git push --set-upstream heroku master`
- `heroku run rails db:migrate`