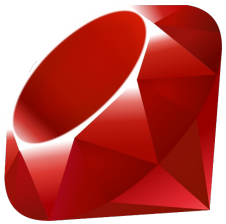
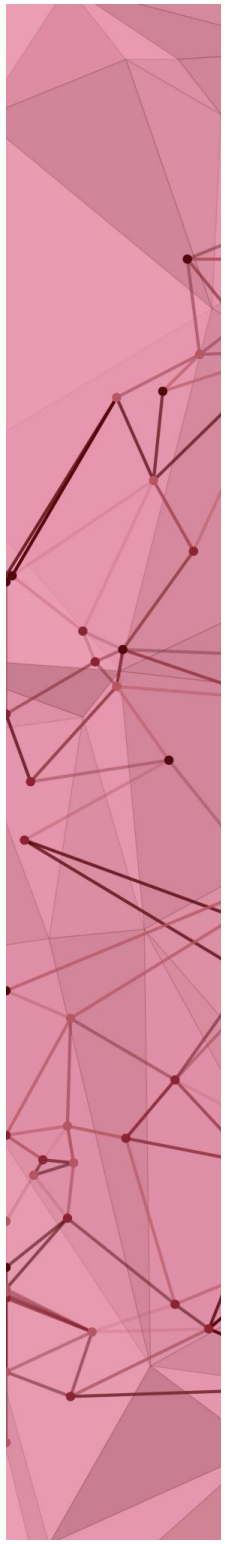
CSCI 2320
Web Programming:
Ruby on Rails



Mohammad T. Irfan

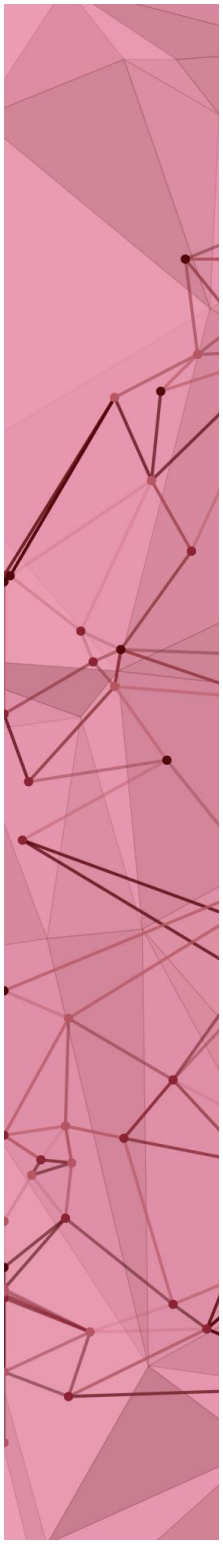
Plan

- Model-View-Controller (MVC) framework of web programming
- Ruby on Rails



Ruby on Rails

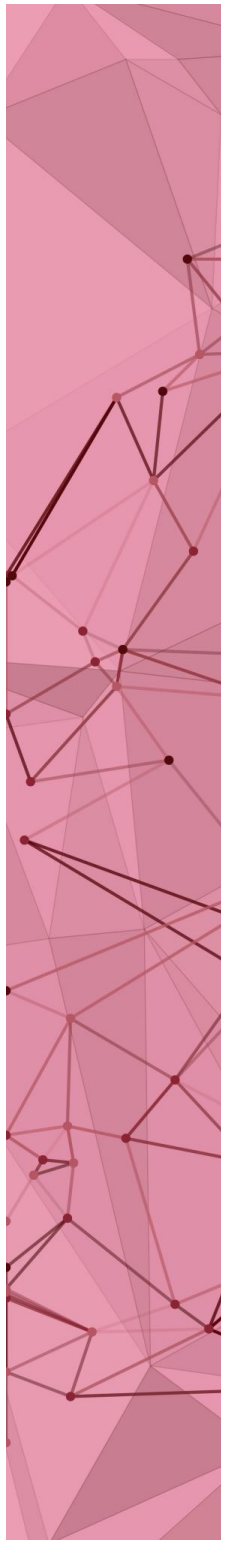
- Developed by David Hansson – released 2004
- MVC architecture
  - MVC by Trygve Reenskaug, 1979
  - GUI for Smalltalk
- Learning Resources
  - Quick guide
    http://guides.rubyonrails.org/getting_started.html
  - Best online book
    https://www.railstutorial.org/book



Interview of David H. Hansson

“Ruby Is Closer to Human Thought Than to Code”

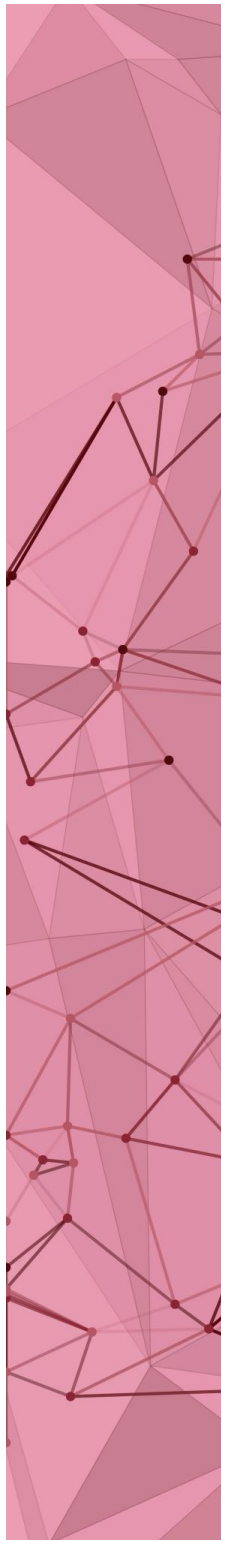
https://bigthink.com/videos/ruby-is-closer-to-human-thought-than-to-code/



Ruby on Rails – MVC framework

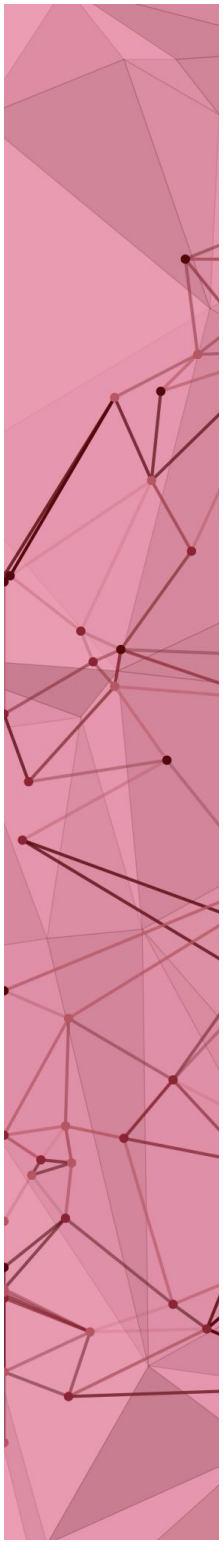
Goal: Decouple the three parts of an application–

Model, View, Controller



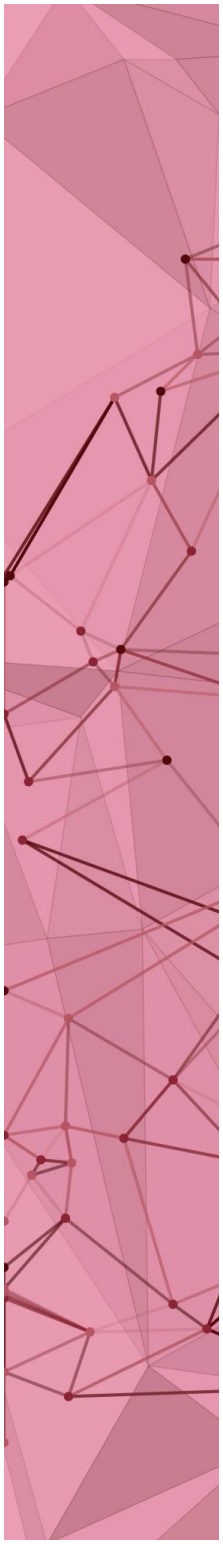
Model

- Database (DB)
- Constraints on data
- Object Relational Mapping (ORM)
  - Maps DB tables to classes, rows to objects
  - Called ActiveRecord



View

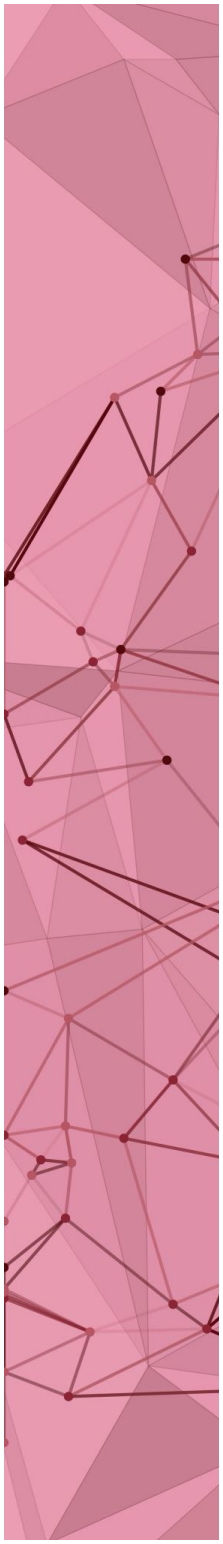
- Prepares and presents results for users
- Templates
  - XHTML
  - XML
  - Javascript



Controller

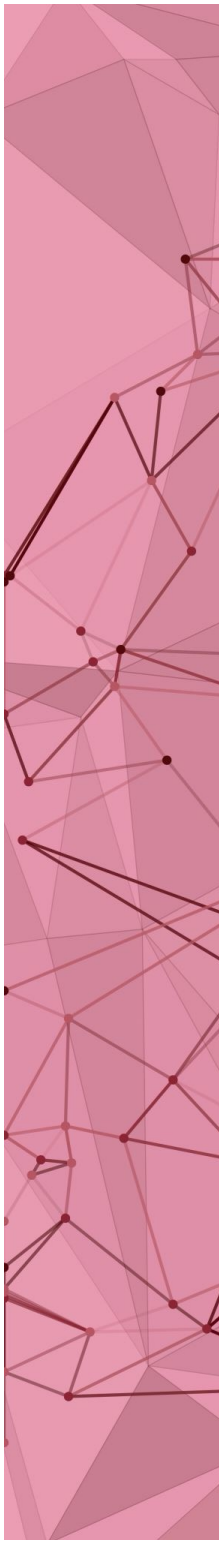
- Takes user input
- Consults with model
- Directs the view

The basic code is auto-generated



Getting started

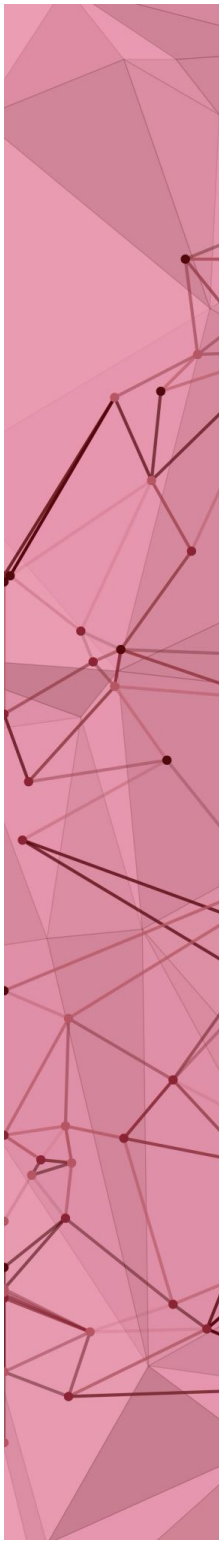
- Terminal command from the parent folder
  - rails new projectName
  - Windows users: open Gemlock.lock file, change “sqlite 3 (1.3.8-...)” to “sqlite 3 (1.3.8)”
- Error related to Gemfile?
  - cd to project folder
  - Execute command: bundle install
- Start the server
  - Open the newly created project folder in VS Code
  - Execute terminal command: rails server (or, rails s)
  - No need to restart the server when you edit code
- Open a browser and go to http://localhost:3000/

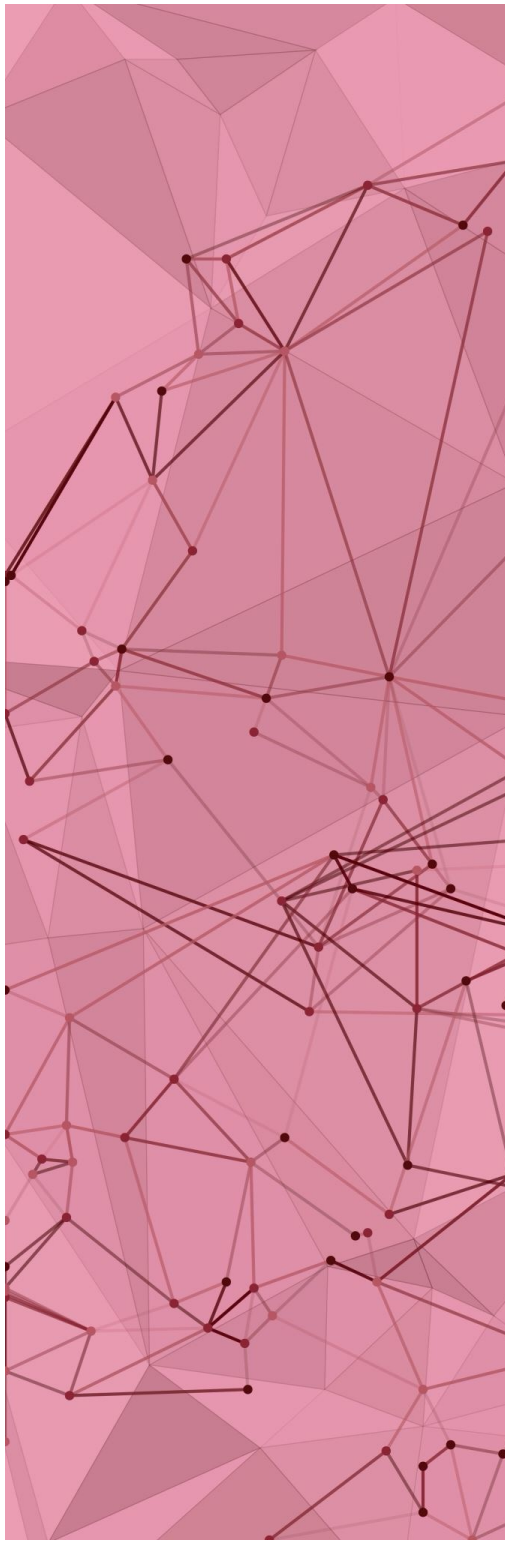


Browse the project folders

App

- models
- views
- controllers





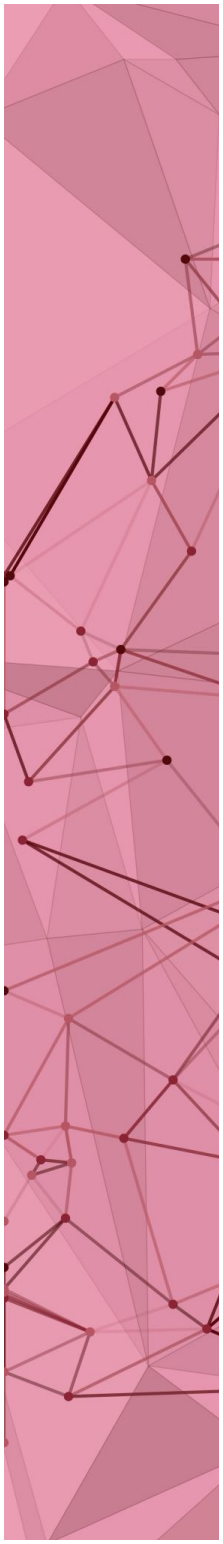
Creating a new website

1. Create its own controller
2. Add pages to it later on

Website with dedicated controller

Initial setup: Create a project (rails new ProjectName) and open the new project folder in VS Code

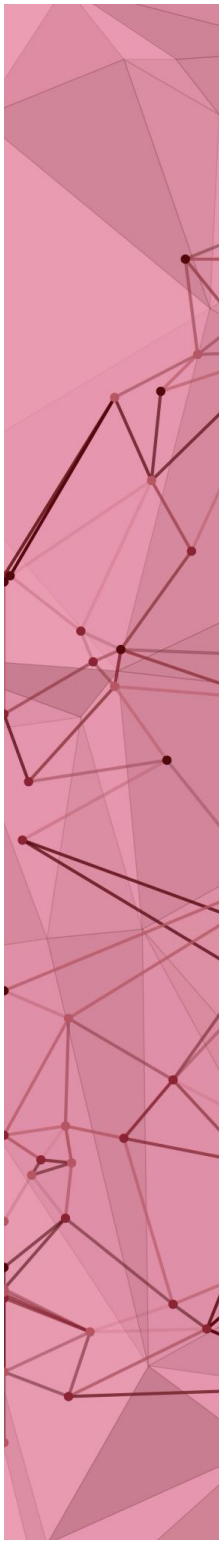
- Command
  - rails generate controller MyHomePage home contact --no-test-framework
- Controller class
- Views



Test the pages

- http://localhost:3000/my_home_page/home
- http://localhost:3000/my_home_page/contact
- http://localhost:3000/my_home_page
  - Error
  - Look into config/routes.rb

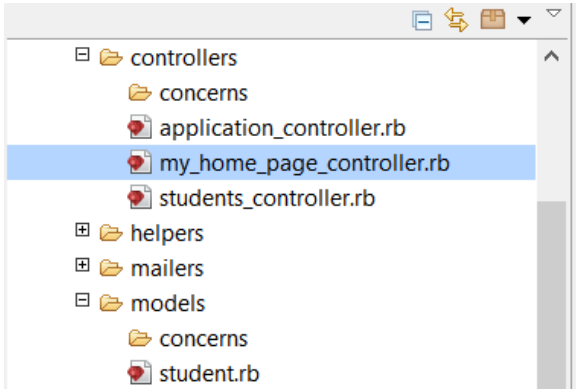
```
get 'my_home_page/home'
get 'my_home_page/contact'
```



How it works:

http://localhost:3000/my_home_page/home

- Router routes to MyHomePageController controller
- The home method of the Controller class is executed first
  - Empty for now
- Then the corresponding view is executed
  - home.html.erb
  - You may edit it as you like




```
1  class MyHomePageController < ApplicationController
2    def home
3    end
4
5    def contact
6    end
7  end
8
```

Add a page without adding new controller

- First, modify config/routes.rb by adding

  get "my_home_page/projects"

- Modify the controller class in my_home_page_controller.rb

  def projects

  end

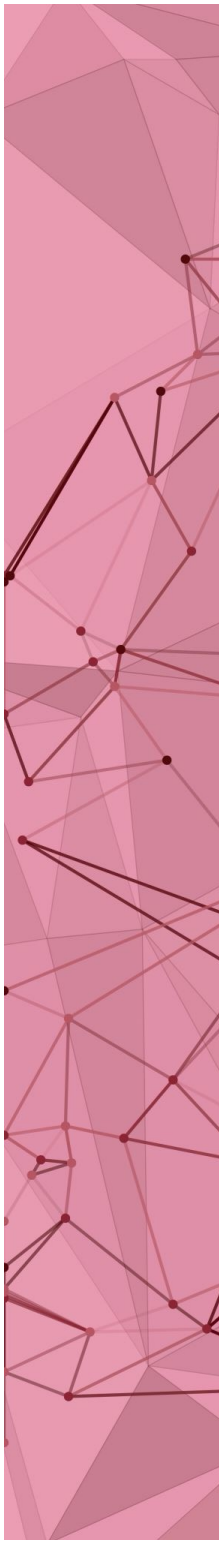
- Create view
  - Add a new projects.html.erb file in views/my_home_page folder
- Any content:

  <h1>Here are my Ruby projects</h1>

  <%=image_tag("ruby_logo.png", size:"200")%>

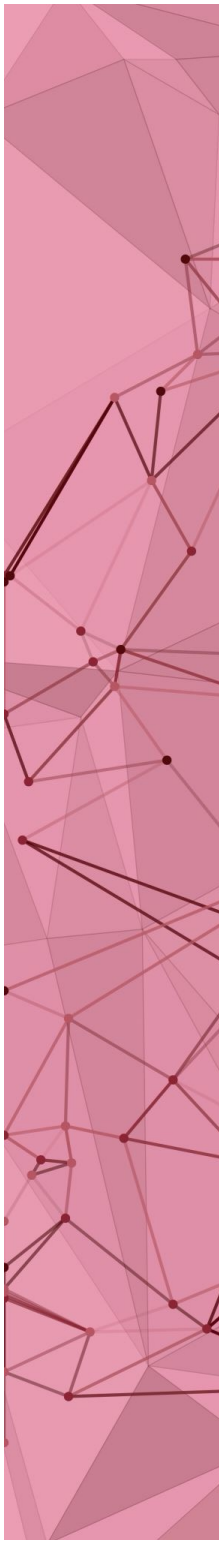
Drag & drop it in app/assets/images

More here: http://guides.rubyonrails.org/layouts_and_rendering.html



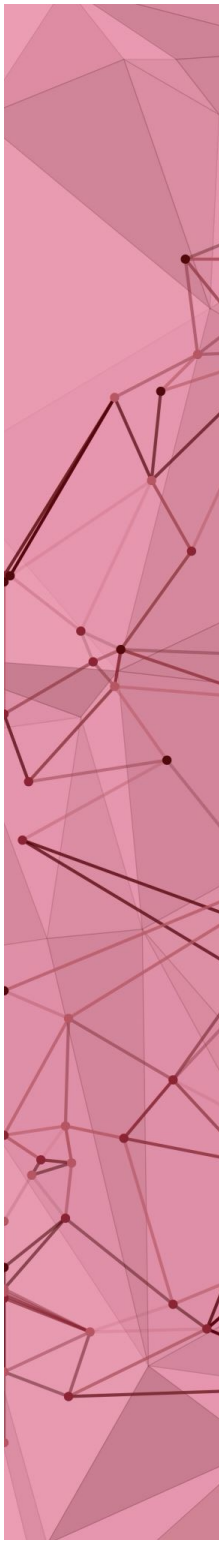
Rails architecture

- Representational State Transfer (REST)
  - Roy Fielding (2000) – “architectural style”
- Clients communicate with web service
  - Limited number of verbs
  - Resources (nouns) – identified by URI
- Rails
  - Nouns: objects in ORM
  - Verbs: create, read, update, delete (CRUD)
- HTTP
  - Nouns: URL
  - Verbs: GET, POST, PATCH, DELETE



REST is stateless (or memoryless)!

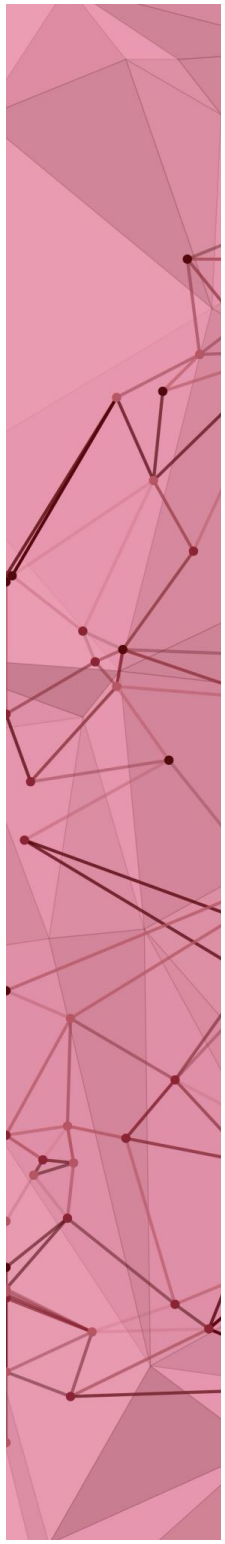
- Every new request creates a new controller object
- All prior controller objects wiped out
- No data transfer from one request to next
  - Way around: database, cookies



Building an auction app from scratch

Plan

- Rails web application
  - A more involved example
  - Without “scaffolding”
  - Understand flow of control
- Problem: web service with database connectivity – auction
  - Input: name and bid amount
  - Store bid information in database
  - Output: show all bids in sorted order



Welcome to the auction!

Google this picture
Download it
(Copy to
app/assets/images)

Your Name:

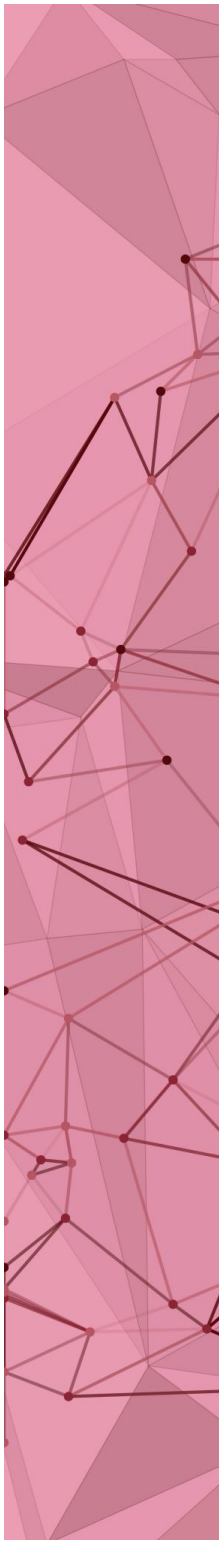
Your Bid: $

Enter Bid

Bidder	Bid
David Kerrigan	250.0
Alice Lin	200.0
Rob Johnson	100.0
David Parkes	50.0
Bob Mitchell	25.0
Just Kidding	0.5

Start of workflow

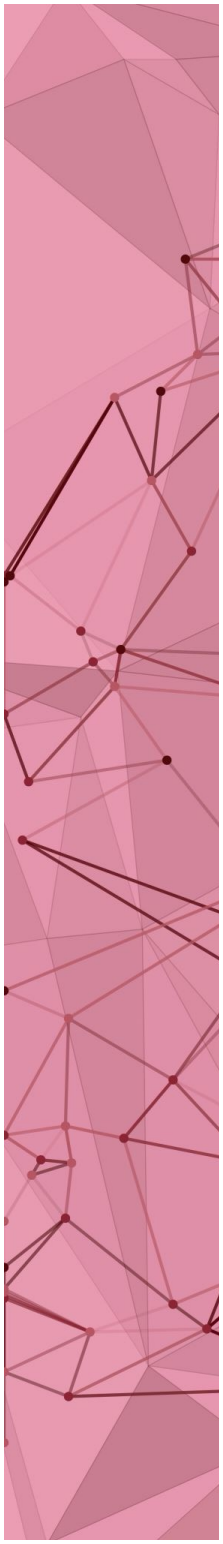
1. Open a terminal and cd to your Rails folder
2. Create an application
   - rails new AuctionApp
3. Open the AuctionApp folder on VS Code
4. Create a controller – the only controller
   - rails generate controller AuctionApp index
5. Start the server (below, s stands for server)
   - rails s



Routing – config/routes.rb

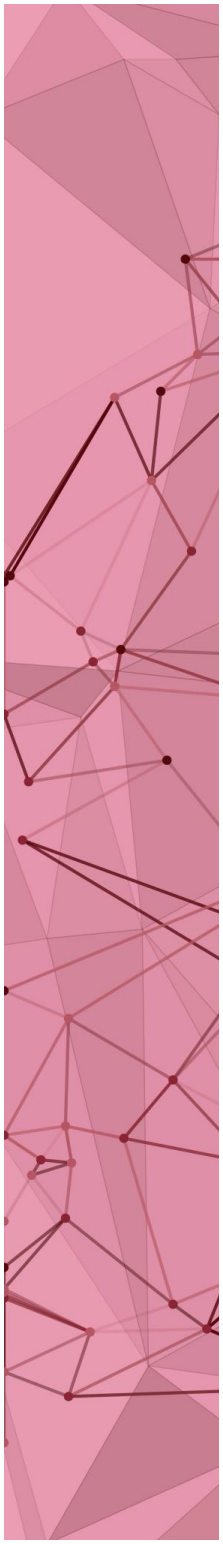
- Make the index page the root (http://localhost:3000)
  - *root "auction_app#index"*
- Other routing information (previously these were done automatically when you said *resources :students*)
  - get "/auction_app" **=> "auction_app#index"**
  - get "/" **=> "auction_app#index"**
  - post "/" **=> "auction_app#enterBid"**

```
1  Rails.application.routes.draw do
2     root "auction_app#index"
3     get "auction_app" => "auction_app#index"
4     get "/" => "auction_app#index"
5     post "/" => "auction_app#enterBid"
6  end
```



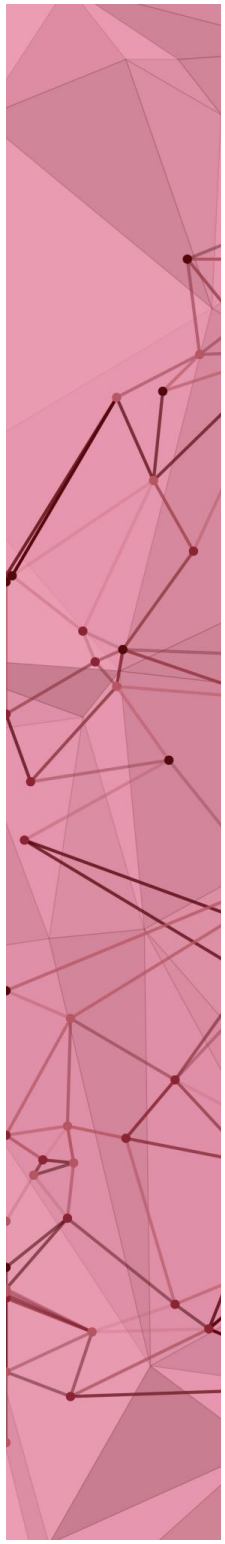
Model – without scaffolding

- Open a new terminal in VS Code
- Create a model: ORM
  - rails generate model Bid bidder:string amount:float
- Create actual database table
  - bundle exec rake db:migrate #creates DB table bids
- We don't want a separate controller for this (want to use auction_app_controller)
  - Don't say *resources :bids* in routes.rb
  - If you say so, it will *automatically* (without writing it explicitly in *routes.rb*) map HTTP get, post, etc. to *index*, *create*, etc. methods of the bids_controller.rb which we don't have!



Controller

- Action for the “Enter Bid” button
  - auction_app/enterBid: enterBid method in auction_app_controller
- Next: write this method
  - This is the method that will be called when the “submit” button is pressed
  - You are allowed to pick any name for the method
  - The name must match with the router though!



```ruby
class AuctionAppController < ApplicationController
  protect_from_forgery with: :null_session
  def index
    puts "-------------- In Index ---------------"
    @allBids = Bid.all
    puts "# of bids = #{@allBids.size}"
    @allBids = @allBids.sort_by {|bid| [-bid.amount,bid.bidder]}
  end

  def enterBid
    puts "------------- In Enter Bid -------------"
    bidder = params[:bidderInput]
    amount = params[:amountInput].to_f
    map = {"bidder" => bidder, "amount" => amount}
    newRow = Bid.new(map)
    respond_to do |format|
      if newRow.save
        puts "Success!"
        format.html{redirect_to auction_app_url}
      else
        format.html{redirect_to "/"} #Can create an error page
      end
    end
  end
end
```

Controller

Method name

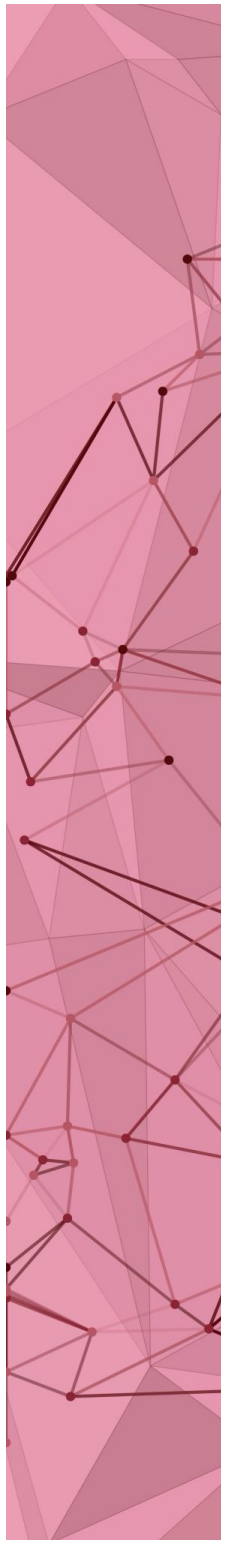
Argument: responder obj.

Body

More: http://bit.ly/1p1L8AR

Other DB functions

- newRow.save
- newRow.update
- newRow.destroy
- Bid.find(map)



View

Create the view:
HTML way or ERB way

Add to change size:
, width: "300"

Your image name could be different!

```
1  <h1>Welcome to the auction!</h1>
2  <p> <%= image_tag "starry.jpg"%> </p>
3  <!-- equivalent html code
4  <form name="bidInput" action="/" method="post">
5      <div>
6          <p>Your Name: <input type="text" name="bidderInput"></p>
7          <p>Your Bid: <input type="text" name="amountInput"></p>
8          <p><input type="submit" value="enter_bid"></p>
9      </div>
10 </form>
11 -->
12 <%= form_tag do%>
13     <p>Your Name: <%= text_field_tag(:bidderInput) %> </p>
14     <p>Your Bid: $ <%= text_field_tag(:amountInput)%> </p>
15     <p> <%= submit_tag "Enter Bid"%> </p>
16 <% end %>
17
```

View (continued)

```
18 <table >
19     <thead>
20         <tr>
21             <th> Bidder </th>
22             <th> Bid </th>
23         </tr>
24     </thead>
25
26     <tbody>
27         <% @allBids.each do |bid| %>
28             <tr>
29                 <td> <%= bid.bidder %> </td>
30                 <td> <%= bid.amount %> </td>
31             </tr>
32         <% end %>
33     </tbody>
34 </table>
```

Welcome to the auction!

Your Name:

Your Bid: $

Enter Bid

Bidder	Bid
David Kerrigan	250.0
Alice Lin	200.0
Rob Johnson	100.0
David Parkes	50.0
Bob Mitchell	25.0
Just Kidding	0.5

To see the actual database files:

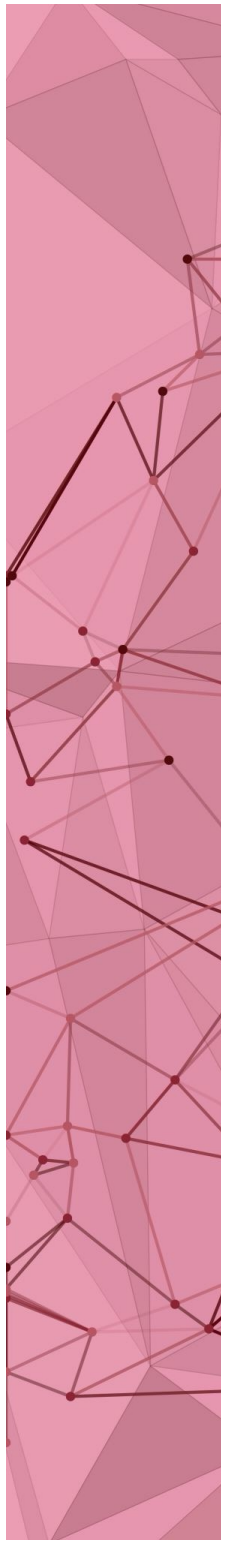
1. cd to the storage folder
2. command:
   sqlite3 development.sqlite3
   > .tables
   > select * from bids;

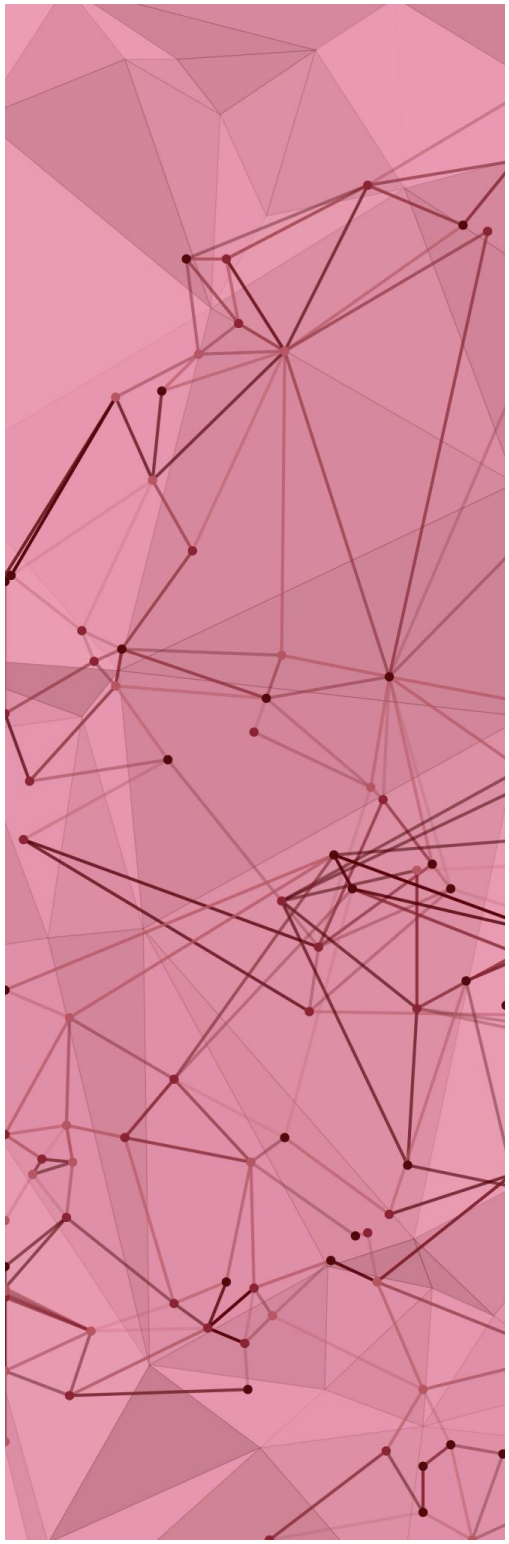
```
11|Rob Johnson|100.0|2014-11-20 03:05:43.528
12|Alice Lin|200.0|2014-11-20 03:06:33.16803
13|David Kerrigan|250.0|2014-11-20 03:07:14.
14|David Parkes|50.0|2014-11-20 03:07:54.221
15|Bob Mitchell|25.0|2014-11-20 03:08:12.811
16|Just Kidding|0.5|2014-11-20 03:08:44.7282
```

Flow of control

- localhost:3000
  ➔ routes.rb routes it to auction_app_controller's index method
  ➔ shows output of index.html.erb

- Enter data in form and press "Enter Bid" button ➔ routes.rb routes it auction_app_controller's enterBid method (why not the index method?)
  ➔ Redirects to homepage





Multiple Forms

with

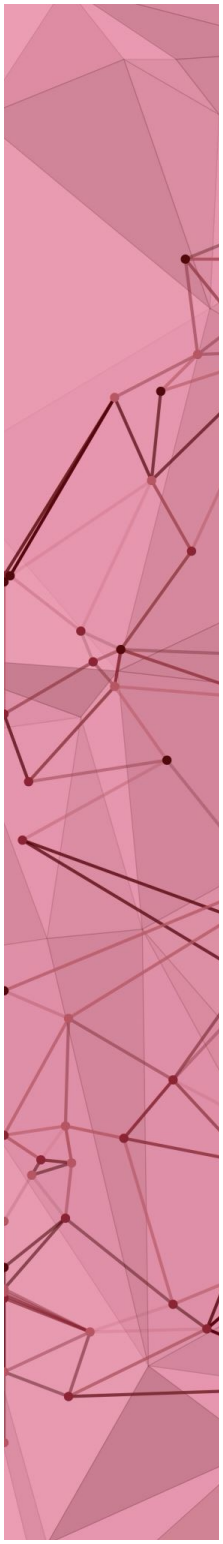
One Controller

One Post Handler

Auction App

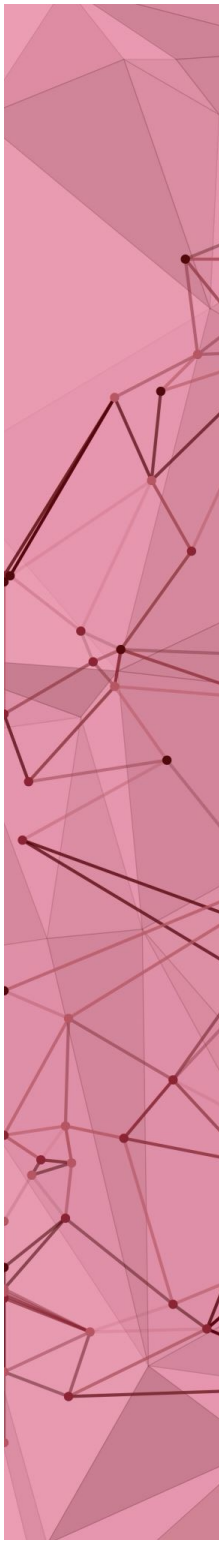
Create a new button to find the leader

1. View: add embedded Ruby (erb) code for new form [alternative: HTML]
2. routes.rb: Enter the name of a new method to handle all posts
3. Controller: Implement the new post-handler method
   - Also implement a method for finding the leader



View (index.html.erb)

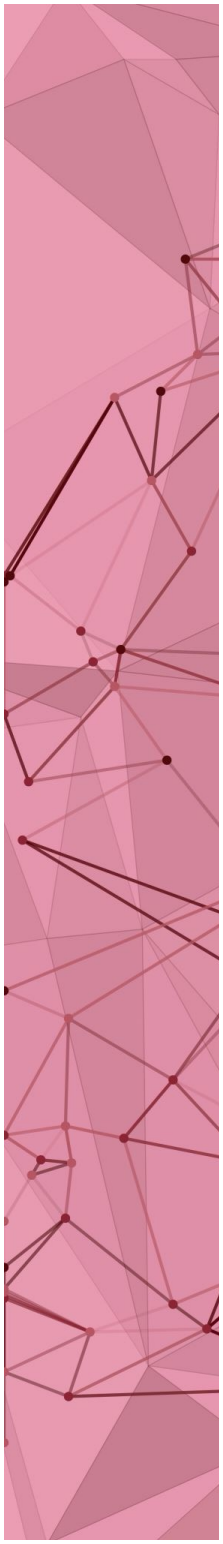
```
35  <!-- Another form on the same page:
36       Find the leader's name -->
37  <%= form_tag do%>
38       <p> Who's leading the acution now?
39            <%= submit_tag "Get Leader"%>
40       </p>
41  <% end %>
42
```



routes.rb

```
1  Rails.application.routes.draw do
2     root "auction_app#index"
3     get "/auction_app"=>"auction_app#index"
4     get "/"=>"auction_app#index"
5     #post "/"=>"auction_app#enterBid"
6     post "/"=>"auction_app#handlePost"
```

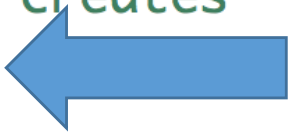
Next: add methods to the controller class



```ruby
  def getLeader
    puts "-------------- In Get Leader ----------------"
    #Need to sort again, because every request creates
    #a new instance of Controller class (why?)
    @allBids = Bid.all
    @allBids = @allBids.sort_by {|bid| [-bid.amount,bid.bidder]}
    puts "Leader: #{@allBids[0].bidder}"
    respond_to do |format|
        format.html {redirect_to auction_app_url}
    end
  end

  def handlePost
    if params[:commit] == "Enter Bid"
      enterBid
    elsif params[:commit] == "Get Leader"
      getLeader
    end
  end

end #end of class AuctionAppController
```



Welcome to the auction!

Your Name:

Your Bid: $

Enter Bid

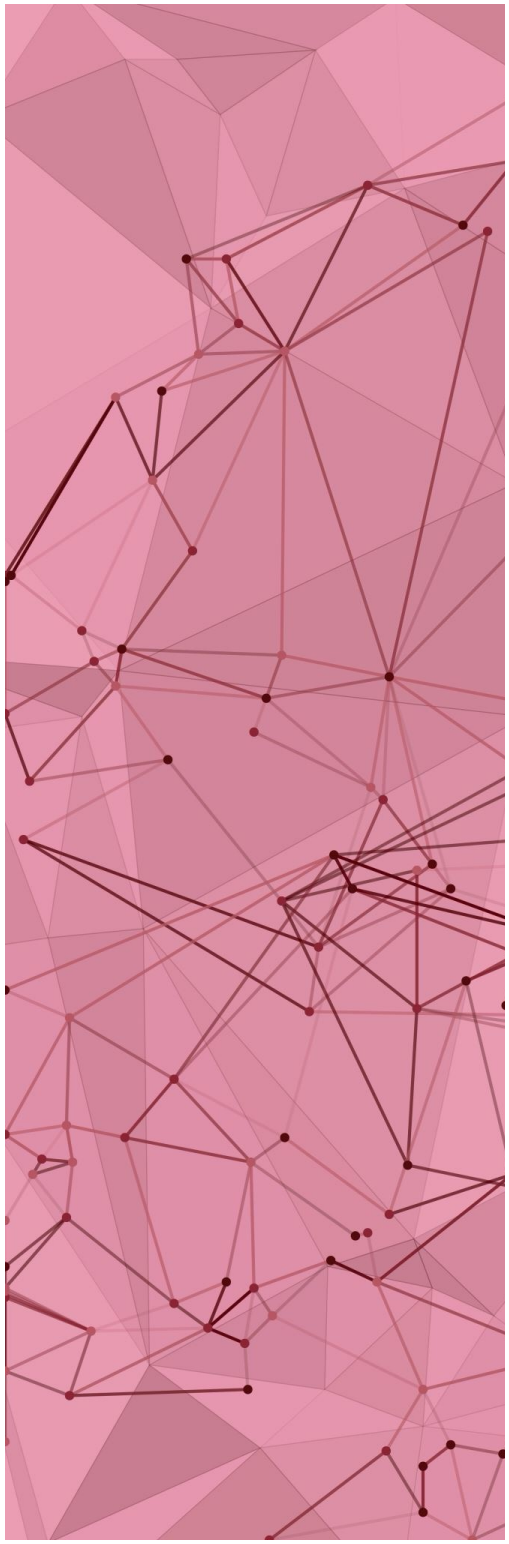
Click

Bidder	Bid
Bob	200.0
Alice	100.0
David	50.0
Clint	25.0

Who's leading the acution now? Get Leader

Message from the Rails Server:
note how post is handled

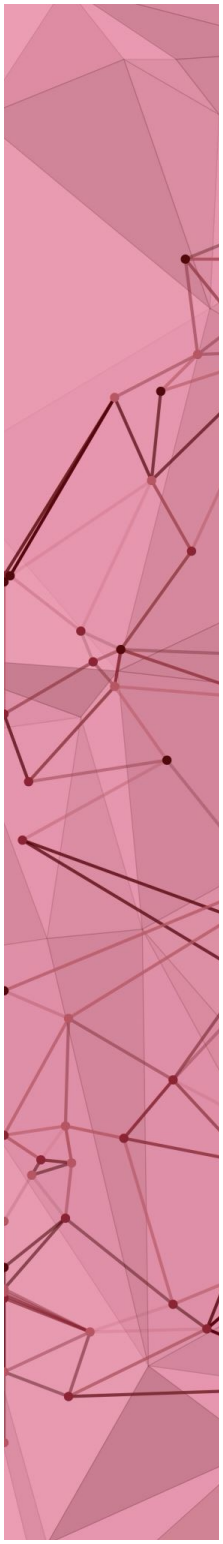
```
Started POST "/" for ::1 at 2017-11-28 01:47:26 -0500
Processing by AuctionAppController#handlePost as HTML
  Parameters: {"utf8"=>"✓", "authenticity_token"=>"PObsUzvwMXuw0/28xwUISw+h
Ny3NfkMGmV4JVy/yzqxLVQGgoPjoKaZbvorKSOw==", "commit"=>"Get Leader"}
--------------- In Get Leader ---------------
  Bid Load (0.2ms)  SELECT "bids".* FROM "bids"
Leader: Bob
Redirected to http://localhost:3000/auction_app
Completed 302 Found in 6ms (ActiveRecord: 0.4ms)
```

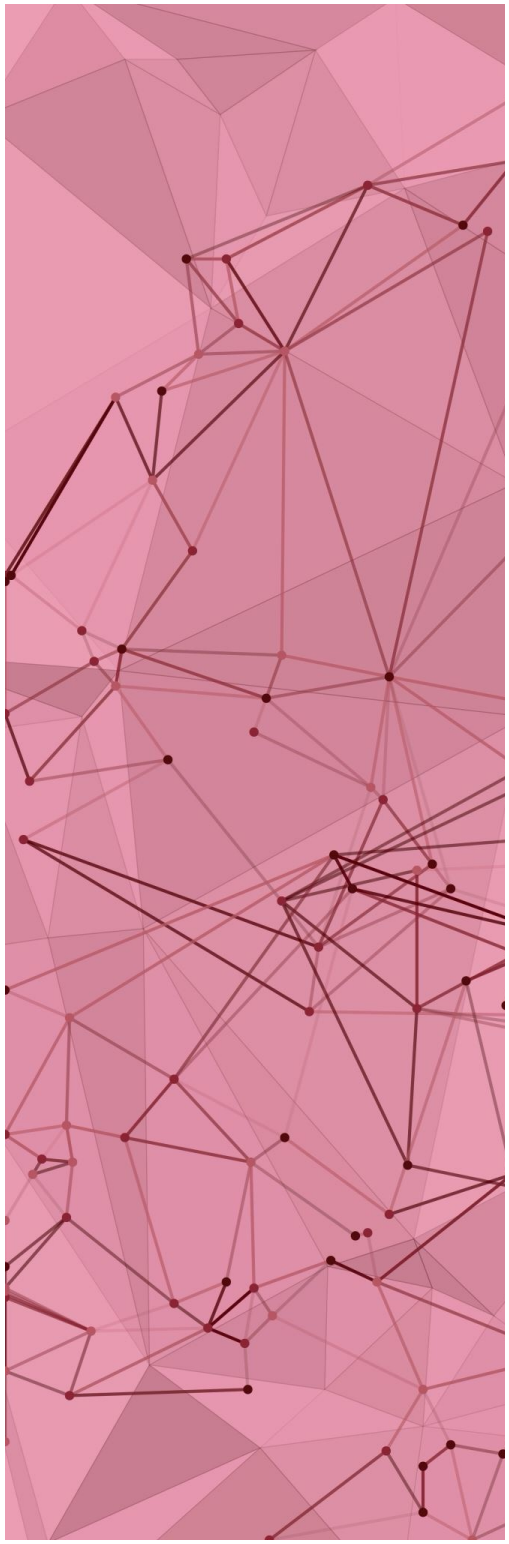


Other data operations

Active records

- Create new object or equivalently new row in a table
- Update existing object/row
- Delete existing object/row
- Tutorial:
  - https://guides.rubyonrails.org/active_record_basics.html





Scaffolding (optional)

Quick way of creating a database project

Scaffolding

- Fast process of generating start-up codes
- First, design a schema

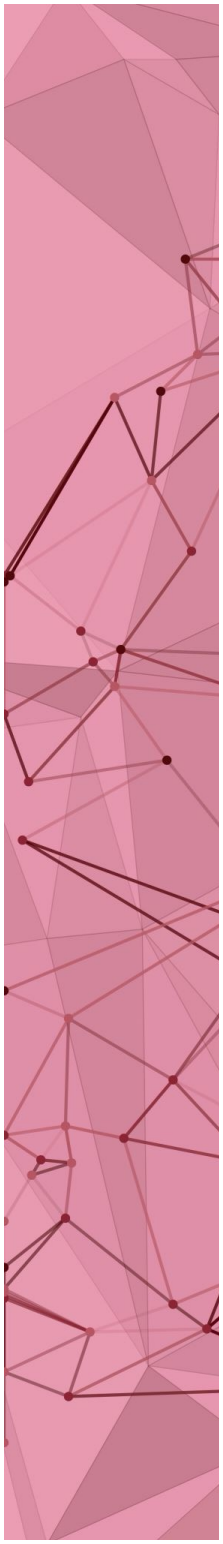
bID	name	email
B01224	Bob	bob@bowdoin.edu
…	…	…
…	…	…

students

- Command for ORM
  - rails generate scaffold Student bID:string name:string email:string
- Other useful commands: rails destroy scaffold ... (delete a previous ORM)

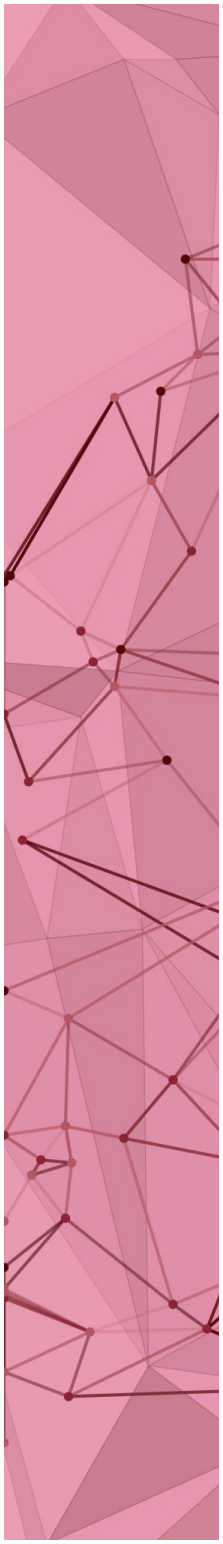
Migrate model to DB

- Command for migration
  - bundle exec rake db:migrate
  - Reverse is: rake db:rollback (don't run it now)
- "rake"
  - Ruby's make: configure, make, make install
- "bundle exec": executes the rake script (db:migrate) in the context of the project's Gemfile



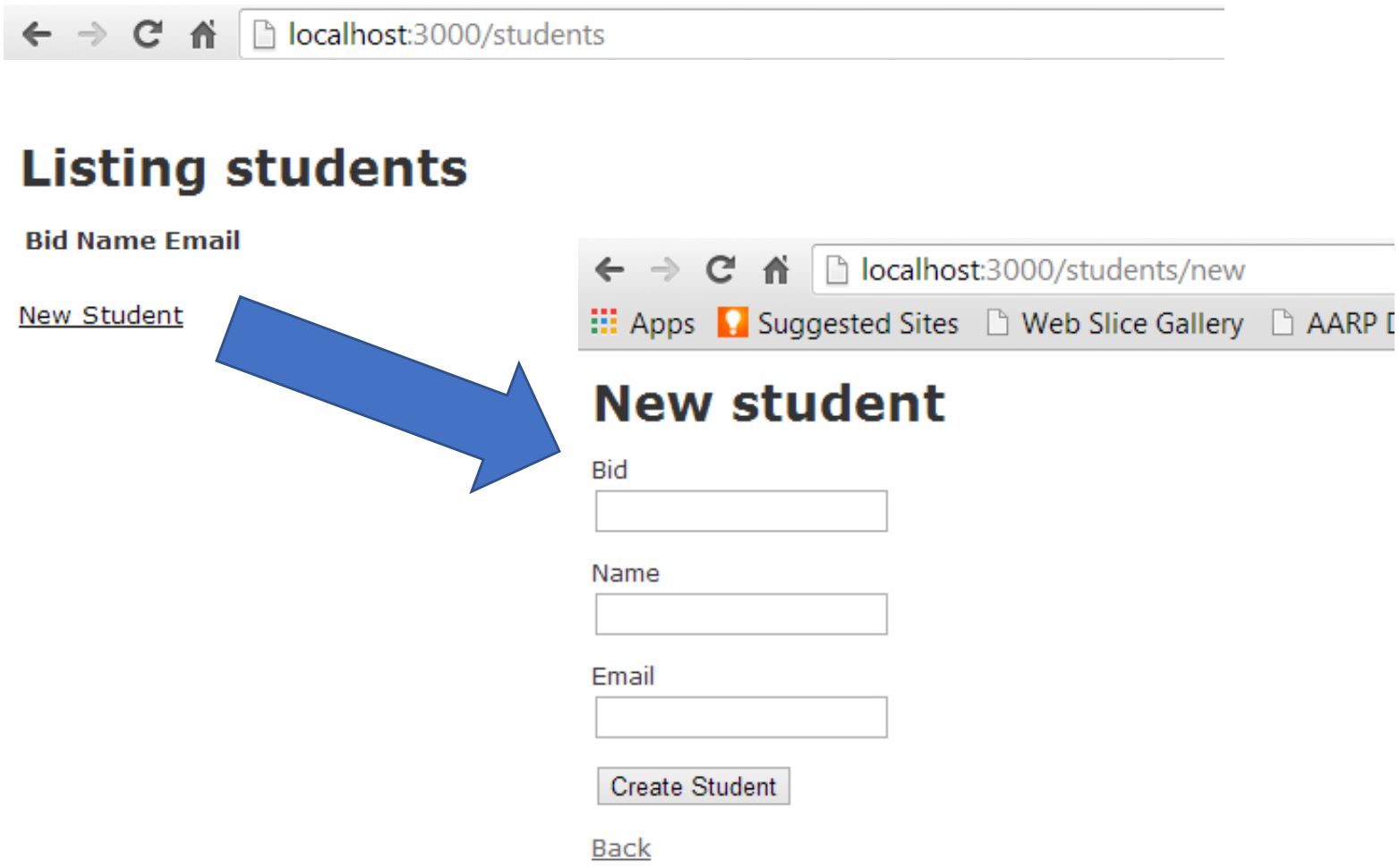
View the webpage

- Command: rails s
  - s is shortcut for server
- Go to http://localhost:3000 on web browser
  - No surprise there



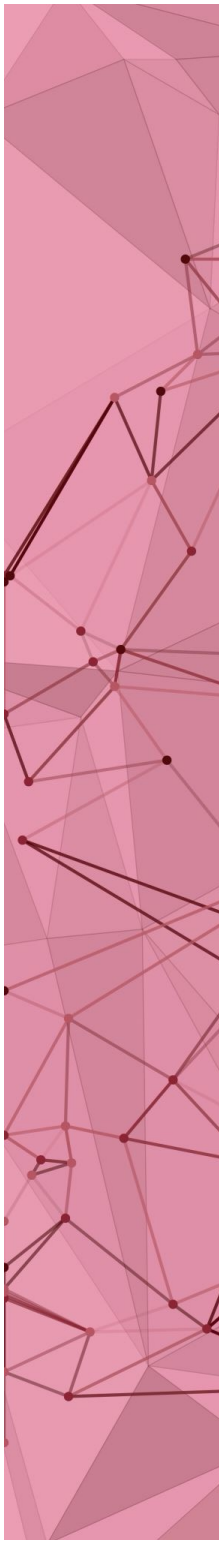
Automatically created form

Navigate to http://localhost:3000/students



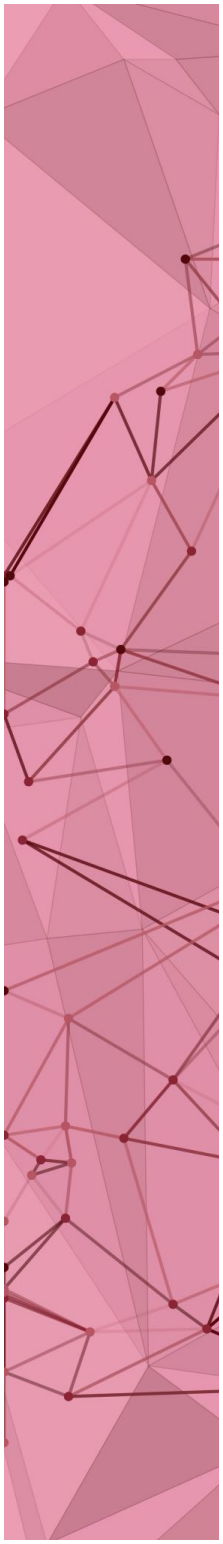
Where's the database?

- Location information: config/database.yml
- Usually in the storage folder
- To work on the database from the terminal:
  - cd to the storage folder
  - Command: sqlite3 development.sqlite3



What's going on?

1. Browser: http://localhost:3000/students
2. <Ruby Router> routes to students_controller.rb
3. students_controller.rb gets data from database table students (using ORM)
4. students_controller.rb feeds data to View<index.html.erb> within the students view (erb = embedded Ruby)
5. index.html.erb produces a nice html file and gives it to students_controller.rb
6. students_controller.rb sends that html file to browser.



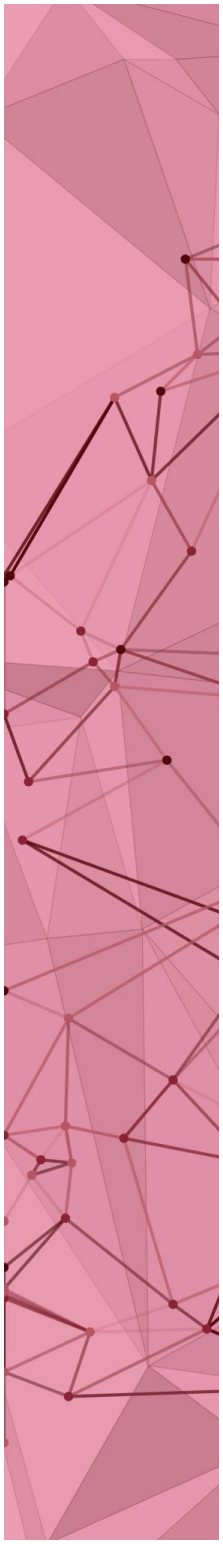
Rails router

- config/routes.rb
  - resources :***students***
- Routes to app/controllers/students_controller.rb

- **class StudentsController < ApplicationController**

```
# GET /students/new
def new
        @student = Student.new
 end
```

Student class is in models/ student.rb



Controller

- **class StudentsController < ApplicationController**

```
# GET /students/new
def new
        @student = Student.new
 end
```

Student class is in models/ student.rb