**The Comment App**

**------------------------------------------------------------PHASE I-------------------------------------------------------------------**

**Recreate the article project with all the functionality as before**

**Now lets add images to your project with “paperclip” gem**

*>>gem install paperclip*

*>>bundle install*

*>>bundle show paperclip*

*\*\*you may have to add it to your gemfile directly*

**Now add a migration that add the paperclip fields to your project. Then look at the migration that was added so you know what fields you are working with.**

*>>rails g migration AddPaperclipToProject*

*>>rake db:migrate*

*\*\*make sure when you migrate you see that the files were added*

**Then update your project model to have an image associated with it.**

*has\_attached\_file :image*

*validates\_attachment\_content\_type :image, :content\_type => [“image/jpg”,”image/jpeg”,”image/png”]*

**You must also modify your helper. Remember String Parameters**

*module ProjectsHelper*

*def project\_params*

*params.require(:project).permit(:title, :description, :tag\_list, :image)*

*endend*

**You must also modify your \_form. We need to tell it to use multipart data. This is needed any time people can upload a file. Then add the paragraph for your image to be uploaded**

*<%= form\_for(@article, html: {multipart: true}) do |f| %>*

*<p>*

*<%= p.label :image, "Attach Image"%></p>br>*

*<%= p.file\_field :image %>*

*</p>*

*\*\*Challenge: explore other styles and sizes, etc that you can do with paperclip*

**But what if we want more that one image? We’d have to create a model and controller with “has\_many”. We’ll do this later.**

**Generate a comment model with an author of type string and body of type text**

*>>rials g model Comment author\_name:string body:text project:references*

*\*\*Note the project reference. This is a rails conventions since each project has\_many comments*

**Migrate your model**

*>>you figure it out (redo the article app a few times if you don’t know)*

**Now lets take advantage of relational databases. Notice your comment model belongs to Project. Now your project model should be modified so it has\_many :comments**

**\*\*Rails Convention: project\_id**

**Now the comments that belong to a project can be referenced and edited via console:**

*p = Project.first*

*p.comments.new*

**Now display comments in our show.html.erb and add a partial called \_comment.html.erb**

*belongs\_to :project*

**Test it in the console and create a few comments for an article**

*proj = Project.first*

*proj.comments.new*

*you play with it*

*Note: proj.reload may come in handy*

**------------------------------------------------------------PHASE II-------------------------------------------------------------------**

**Now lets add authentication with sorcery**

*>>you figure it out (redo the article app a few times if you don’t know)*

User App with added Gem (DEVISE)

Processing Secure Payments

Routes and Routing

* Shallow Routing
  + builds routes with the least amount of information necessary to create the route

**Migrations** db/migrate/(some\_time\_stamp)\_create\_articles.rb

* Two methods
  + Up and down replaced by change
* T.timestamps
  + Create two columns for us created\_at, updated\_at

What’s the difference between: <%= %> and <% %> (one will output, one will be hidden)?

More on “Strong Parameters”

The [Github](https://github.com/blog/1068-public-key-security-vulnerability-and-mitigation) incident.

1. Rails has a class method attr\_protected that is used to specify attributes that can not be part of mass-assigntment.

Now the admin value is not accessible for mass-assignment. Attackers can not update this value in url or through a form.

2. The other way to protect all models in your app from mass-assignment is to setconfig.active\_record.whitelist\_attributes to true in config/application.rb

Now all model’s attributes are not accessible for mass-assignment. But if we want some attributes to be accessible for mass-assignment, we can use the attr\_accessible method

attr\_accessible is just opposite of attr\_protected. When an attribute passed to the attr\_accessible method, it becomes available for mass-assignment

In Rails 3.2.3 config.active\_record.whitelist\_attributes is true by default and in all models attr\_accessible method is used for whitelisting the attributes.

**\*\*Challenges:**

**STOP: YOU SHOULD NOW ALL UNDERSTAND**

1. **How to add comments to your projects application using “belongs\_to” and “has\_many”**