How to Build a Dropbox-like Sharing App using Ruby on Rails

By: Phyo Wai

In this Premium Nettuts+ tutorial, we'll learn how to build a file-sharing web application, like Dropbox, using Ruby on Rails.

Introduction

The enormous success of DropBox clearly shows that there's a huge need for simple and fast file sharing.

Our app will have the following features:

- simple user authentication
- upload files and save them in Amazon S3
- create folders and organize
- share folders with other users

Throughout the tutorial, I will point out different ways that we can improve our app. Along the way, we'll review a variety of concepts, including Rails scaffolding and AJAX.

Step 0 Getting Ready

> "Make sure that you upgrade to Rails 3 to follow along with this tutorial."

Before we begin, ensure that you have all of the followings installed on your machine:

- Ruby 1.9.+
- Rails 3+
- MySQL 5

If you're brand new to Ruby and Rails, you can start the learning process here. We will be making use of Terminal (Mac), or the Command Line, if you're a Windows user.

Step 1 Create a New Rails App

First, we need to create a Rails app. Open Terminal, and browse to your desired folder. We'll call our app, "ShareBox." Type the following into Terminal.

view plaincopy to clipboardprint?

```
1. Rails new sharebox -d mysql
```

> "The `-d mysql` option is added to ensure that the app uses MySQL as its database, since, by default, it will use sqlite3."

This will create a file and folder structure, similar to what's shown below.

We now must make sure that we have the correct gems installed. Open the "Gemfile" from your Rails directory, and replace all of the code inside with the following:

view plaincopy to clipboardprint?

```
1. source 'http://rubygems.org'
2.
3. gem 'Rails', '3.0.3'
4. gem 'ruby-mysql'
```

> "The Gemfile is the file where you put all the necessary gems required to run this specific app. It helps you organize the gems you would need."

Since we are using Rails 3 and MySQL, those two gems are added to the Gemfile. Then we need to run the following command in the Rails "sharebox" directory using Terminal. If you are not in the "sharebox" directory, type "cd sharebox" to go into the folder.

```
1. bundle install
```

The command `bundle install` will install all gems defined in the Gemfile if they haven't already been installed.

Next, let's make sure that we have a working database connection, and the database, "sharebox," set up. From the 'sharebox' folder, open "config/database.yml." Change the development and test environment settings, like so:

view plaincopy to clipboardprint?

```
development:
  adapter: mysql
  encoding: utf8
  reconnect: false
  database: sharebox_development
  pool: 5
  username: root
  password:
  socket: /tmp/mysql.sock

test:
  adapter: mysql
  encoding: utf8
  reconnect: false
  database: sharebox_test
  pool: 5
  username: root
  password:
  socket: /tmp/mysql.sock
```

Of course, replace your own MySQL connection details, accordingly. Additionally, you might need to change the socket and add `host` to make it work. Run the following command in the Terminal (under the "sharebox" folder).

view plaincopy to clipboardprint?

```
rake db:create
```

"If you don't see any feedback once this command has completed, you are good to go."

Now if you do, you'll probably need to change the database.yml file to match your system's MySQL connection settings.

Let's review our application in the browser. Run the following in Terminal:

```
Rails server
```

You'll see something like the following in your Terminal.

Now, fire up your favourite browse such as Firefox or Chrome, and type this in the address bar:

```
http://localhost:3000/
```

You should see the Rails' default home page as follows:

Step 2 Create User Authentication

Next up, we'll create basic user authentication.

We'll use the fantastic devise gem to help with our user authentication. We can also use Ryan's nifty-generator to help with our layout and view helpers.

Add those two gems to your Gemfile at the bottom:

view plaincopy to clipboardprint?

```
#for user authentication
gem 'devise'

#for layout and helpers generations
gem "nifty-generators", :group => :development
```

"Don't forget to run `bundle install` to install those gems on your system."

Once the gems are installed, let's start by installing some layouts using nifty-generator. Type the following into Terminal to generate the layout files in the "sharebox" directory.

view plaincopy to clipboardprint?

```
Rails g nifty:layout
```

"Rails g' is short-hand for `Rails generate`.

It will ask you if you want to overwrite the "layouts/application.html.erb". Press "Y" and Enter key to proceed. It will then create a few files which we will be using in a while.

Now let's install "devise". Type this in Terminal:

view plaincopy to clipboardprint?

```
1. Rails g devise:install
```

This command will install a handful of files, but it'll also ask you to perform three things manually.

First, copy the following line, and paste it into "config/environments/development.rb". Paste it just before the last "end."

view plaincopy to clipboardprint?

```
1. config.action_mailer.default_url_options = { :host => 'localhost:3000' }
```

Secondly, we have to set up the root_url in "config/routes.rb" file. Open the file and add:

view plaincopy to clipboardprint?

```
1. Sharebox::Application.routes.draw do
2.
3.   root :to => "home#index"
4. end
```

We'll skip the last step as it's been done by the nifty generator.

Now, let's create our first model, `User`, using "devise".

view plaincopy to clipboardprint?

```
1. Sharebox::Application.routes.draw do
2.
3.   root :to => "home#index"
4. end
```

This will create a few files within your Rails directory. Let's have a quick look at the migration file, "db/migrate/[datetime]_devise_create_users.rb". It should look something along the lines of:

view plaincopy to clipboardprint?

```
1. class DeviseCreateUsers < ActiveRecord::Migration
2.   def self.up
3.     create_table(:users) do |t|
4.       t.database_authenticatable :null => false
5.       t.recoverable
6.       t.rememberable
7.       t.trackable
8.
9.       # t.confirmable
10.      # t.lockable :lock_strategy => :failed_attempts, :unlock_strategy => :both
11.      # t.token_authenticatable
12.
13.
14.      t.timestamps
15.    end
16.
17.    add_index :users, :email,                :unique => true
18.    add_index :users, :reset_password_token, :unique => true
19.    # add_index :users, :confirmation_token,   :unique => true
20.    # add_index :users, :unlock_token,         :unique => true
21.  end
22.
23.  def self.down
24.    drop_table :users
25.  end
26. end
```

Devise contains built-in features ready for use out of the box. Among these great features, we'll be using:

- `database_authenticatable`: authenticates users.
- `recoverable`: lets the user recover or change a password, if needed.
- `rememberable`: lets the user "remember" their account on their system whenever they login.
- `trackable`: tracks the log in count, timestamps, and IP address.

The migration also adds two indexes for "email" and "reset_password_token".

We won't change much, but we will add a "name" field to the table, and remove all commented lines as follows:

view plaincopy to clipboardprint?

```
class DeviseCreateUsers < ActiveRecord::Migration
  def self.up
    create_table(:users) do |t|
      t.database_authenticatable :null => false
      t.recoverable
      t.rememberable
      t.trackable

      t.string :name

      t.timestamps
    end

    add_index :users, :email,                :unique => true
    add_index :users, :reset_password_token, :unique => true
  end

  def self.down
    drop_table :users
  end
end
```

Next, run `rake db:migrate` in the Terminal. This should create a "users" table in the Database. Let's check the User model at "app/models/user.rb," and add the "name" attribute there. Also, let's add some validation as well.

view plaincopy to clipboardprint?

```
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :token_authenticatable, :confirmable, :lockable and :timeoutable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  # Setup accessible (or protected) attributes for your model
  attr_accessible :email, :name, :password, :password_confirmation, :remember_me

    validates :email, :presence => true, :uniqueness => true
end
```

Before we move on, we need to quickly add a Home controller with an Index action for the root url. Create a file "home_controller.rb" in the "app/controllers" folder. Next, add the followings inside the file for Index action:

view plaincopy to clipboardprint?

```
class HomeController < ApplicationController

  def index

  end
end
```

Don't forget; we have to create a view for this as well. Create a folder, named "home" inside "app/views/," and add an "index.html.erb" file there. Append the following to this file.

```
This is the Index.html.erb under "app/views/home" folder.

"We need to remove/delete the 'public/index.html' file to make root_url work correctly."
```

Now let's try creating a user, and then sign in/out to get the feel for devise's magic. You need to restart the Rails server (by pressing Ctrl+C and running 'Rails server') to reload all the new installed files from devise. Then go to http://localhost:3000 and you should see:

If we browse to `http://localhost:3000/users/sign_up`, which is one of the default routes already built-in by Devise, to sign up users, you should see this:

Notice how the `name` field is not there yet? We need to change the view to add the name. Since Devise views are hidden by default, we must unhide them with the following command.

view plaincopy to clipboardprint?

```
Rails g devise:views
```

Open "app/views/devise/registrations/new.html.erb" and edit the file to add the "name" field to the form.

view plaincopy to clipboardprint?

```
<h2>Sign up</h2>

<%= form_for(resource, :as => resource_name, :url => registration_path(resource_name)) do |f| %>
  <%= devise_error_messages! %>

  <p><%= f.label :email %><br>
  <%= f.text_field :email %></p>

  <p><%= f.label :name %><br>
  <%= f.text_field :name %></p>

  <p><%= f.label :password %><br>
  <%= f.password_field :password %></p>

  <p><%= f.label :password_confirmation %><br>
  <%= f.password_field :password_confirmation %></p>

  <p><%= f.submit "Sign up" %></p>
<% end %>

<%= render :partial => "devise/shared/links" %>
```

If you refresh the page, you should now see:

If you fill in the form and submit it to create a new user, you'll then see the following image, which means you have successfully created the user, and have logged in.

To sign out, go to "http://localhost:3000/users/sign_out" and you'll be signed out with this flash message.

You can then sign back in at "http://localhost:3000/users/sign_in" with your login credentials:

We now have the basic authentication working. The next step is for tidying up the layout and look and feel of the pages along with the links.

Step 3 Add Basic CSS

> "The 'app/views/layouts/application.html.erb' layout file will be applied to all view pages unless you specify your own layout file."

Before we do anything, change the title of each page. Open up "app/views/layouts/application.html.erb" file and revise the title like so:

view plaincopy to clipboardprint?

```
<title>ShareBox | <%= content_for?(:title) ? yield(:title) : "File-sharing web app" %></title>
```

That'll make the titles look like "ShareBox | File-sharing web app", "ShareBox | Upload files" and so on.

Next, we'll add "Sign In" and "Sign out" links into the same application.html.erb layout file. In the body tag, just before the "container" `div`, add this:

view plaincopy to clipboardprint?

```
<div class="header_wrapper">
    <div class="logo">
        <%= link_to "ShareBox", root_url %>
    </div>
</div>
```

This markup/code will add a Logo (text) with a clickable link. We'll add the CSS in a bit. Now place this snippet of html with Ruby code for showing User login, log out links. Make sure you add this in the "header_wrapper" `div` after the "logo" `div`.

view plaincopy to clipboardprint?

```
<div id="login_user_status">
    <% if user_signed_in? %>
        <%= current_user.email  %>
        |
        <%= link_to "Sign out", destroy_user_session_path %>
    <% else %>
        <em>Not Signed in.</em>
        <%= link_to 'Sign in', new_user_session_path%>
        or
```

```
10.         <%= link_to 'Sign up', new_user_registration_path%>
11.     <% end %>
12. </div>
```

"The `user_signed_in?` method determines whether a user has logged in."

The paths `destroy_user_session_path`, `new_user_session_path` and `new_user_registration_path` are the default paths provided by devise for Signing out, Signing in and Signing up, respectively.

Now let's add the necessary CSS to make things look a bit better. Open the "public/stylesheets/application.css" file, and insert the following CSS. Make sure you replace the "body" and "container" styles as well.

view plaincopy to clipboardprint?

```
1.  body {
2.    background-color: #EFEFEF;
3.    font-family: "Lucida Grande","Verdana","Arial","Bitstream Vera Sans",sans-serif;
4.    font-size: 14px;
5.  }
6.
7.  .header_wrapper {
8.      width: 880px;
9.      margin: 0 auto;
10.     overflow:hidden;
11.     padding:20px 0;
12. }
13. .logo a{
14.     color: #338DCF;
15.     float: left;
16.     font-size: 48px;
17.     font-weight: bold;
18.     text-shadow: 2px 2px 2px #FFFFFF;
19.     text-decoration:none;
20. }
21. #container {
22.   width: 800px;
23.   margin: 0 auto;
24.   background-color: #FFF;
25.   padding: 20px 40px;
26.   border: solid 1px #BFBFBF;
27. }
28. #login_user_status {
29.     float:rightright;
30. }
```

If you refresh the home page in your browser, you'll see:

Step 4 Uploading Files

We'll be using the Paperclip gem to help us upload files. Add this gem in Gemfile as follows, and run "bundle install" in the Terminal.

view plaincopy to clipboardprint?

```
1.  #for uploading files
2.  gem "paperclip", "~> 2.3"
```

Once installed, we'll create our File table. However, since the word "File" is one of the Reserved Words, we'll just use the term "Asset" for the files.

One user can upload multiple files, so we need to add `user_id` to the `Asset` model. So run this in the Terminal to scaffold the `Asset` model.

view plaincopy to clipboardprint?

```
1.  Rails g nifty:scaffold Asset user_id:integer
```

Next, go to the newly created migration file, within the "db/migrate/" folder, and add the index for the `user_id`.

view plaincopy to clipboardprint?

```
1.  add_index :assets, :user_id
```

Run `rake db:migrate` in Terminal to create the Asset table.

Then we need to add relationships to both the `User` and `Asset` table. In "app/models/user.rb" file, add this.

view plaincopy to clipboardprint?

```
1. has_many :assets
```

In "app/models/asset.rb", add this:

view plaincopy to clipboardprint?

```
1. belongs_to :user
```

Now it's time to use those relationships in the controller for loading the relevant resources(assets) for each logged in user. Within "app/controllers/assets_controller.rb", alter the code, like so:

view plaincopy to clipboardprint?

```
1. class AssetsController < ApplicationController
2.   before_filter :authenticate_user!  #authenticate for users before any methods is called
3.
4.
5.   def index
6.     @assets = current_user.assets
7.   end
8.
9.   def show
10.    @asset = current_user.assets.find(params[:id])
11.  end
12.
13.  def new
14.    @asset = current_user.assets.new
15.  end
16.
17.  def create
18.    @asset = current_user.assets.new(params[:asset])
19.    ...
20.  end
21.
22.  def edit
23.    @asset = current_user.assets.find(params[:id])
24.  end
25.
26.  def update
27.    @asset = current_user.assets.find(params[:id])
28.    ...
29.  end
30.
31.  def destroy
32.    @asset = current_user.assets.find(params[:id])
33.    ...
34.  end
35. end
```

In the code above, we're making sure that the asset(s) requested is, in fact, owned or created by the current_user (logged-in user), since "current_user.assets" will give you assets which are "belonged to" the user.

Then let's run Paperclip's migration script to add some necessary fields in Asset model. We'll name the main file field as "uploaded_file". So type this in Terminal.

```
1. Rails g paperclip asset uploaded_file
```

Run "rake db:migrate" to add the fields.

Now that we have added the "uploaded_file" field in the table, we then need to add it in the model. So in the "app/models/user.rb" Model, add the "uploaded_file" field, and define it as "attachment," using Paperclip.

view plaincopy to clipboardprint?

```
1. attr_accessible :user_id, :uploaded_file
2.
3.
4. belongs_to :user
5.
6. #set up "uploaded_file" field as attached_file (using Paperclip)
7. has_attached_file :uploaded_file
8.
9. validates_attachment_size :uploaded_file, :less_than => 10.megabytes
10. validates_attachment_presence :uploaded_file
```

> "`validates_attachment_size` and `validates_attachment_presence` are validation methods provided by Paperclip to let you validate the file uploaded. In this case, we are checking if a file is uploaded and if the file size is less than 10 MB."

We use `has_attached_file` to identify which field is for saving the uploaded file data. It's provided by Paperclip anyway.

Before we test this out, let's change the view a bit to handle the file upload. So open up "app/views/assets/_form.html.erb" file and insert the following.

view plaincopy to clipboardprint?

```
<%= form_for @asset, :html => {:multipart => true} do |f| %>
  <%= f.error_messages %>
    <p>
      <%= f.label :uploaded_file, "File" %><br>
      <%= f.file_field :uploaded_file %>
    </p>
  <p><%= f.submit "Upload" %></p>
<% end %>
```

In the `form_for` method, we have added the html attribute, "multipart," to be `true`. It's always needed to post the file to the server correctly. We've also removed the `user_id` field from the view.

Now it's time to test things out.

> "Restarting the Rails server is necessary each time you add and install a new gem."

If you visit "http://localhost:3000/assets/new", you should see something like:

Once you upload a file, and go to "http://localhost:3000/assets", you'll see:

It's not quite right, is it? So let's change the "list" view of the assets. Open "app/views/assets/index.html.erb" file and insert:

view plaincopy to clipboardprint?

```
<% title "Assets" %>
<table>
  <tr>
    <th>Uploaded Files</th>
  </tr>
  <% for asset in @assets %>
    <tr>
      <td><%= link_to asset.uploaded_file_file_name, asset.uploaded_file.url %></td>
      <td><%= link_to "Show", asset %></td>
      <td><%= link_to "Edit", edit_asset_path(asset) %></td>
      <td><%= link_to "Destroy", asset, :confirm => 'Are you sure?', :method => :delete %></td>
    </tr>
  <% end %>
</table>
<p><%= link_to "New Asset", new_asset_path %></p>
```

As you can see, we have updated the header to "Uploaded Files," and have added the link instead of User Id. `asset` now has the `uploaded_file` accessor to load the file details. One of the methods available is "url," which gives you the full url of the file location. Paperclip also saves the file name in the name of the field we defined ("uploaded_file" in our case) plus "_file_name". So we have "uploaded_file_file_name" as the name of the file.

With that in mind, let's quickly refactor this to have a nicer name for the actual file name. Place this method in the Asset model located at "app/models/asset.rb".

view plaincopy to clipboardprint?

```
def file_name
    uploaded_file_file_name
end
```

This code should allow you to use something like "asset.file_name" to get the uploaded file's actual file name. Make sure you update the index.html.erb page with the new file_name method, as exemplified below.

view plaincopy to clipboardprint?

```
<%= link_to asset.uploaded_file_file_name, asset.uploaded_file.url %>
```

Now if we refresh the page "http://localhost:3000/assets", you'll see:

If you then click the link in the file, you'll be taken to a url, like "http://localhost:3000/system/uploaded_files/1/original/Venus.gif?1295900873".

"By default, Paperclip will save the files under your Rails root at `system/[name of the file field]s/:id/original/:file_name`."

But These Links Are Public!

One of our goals for this application is to secure the uploaded files from unauthorized access. But as you can see, the link is quite open to the public, and you can download it even after logging out. Let's fix this.

First, we should think of a nice url for the file to be downloaded from. "system/uploaded_files/1/original/" looks nice, right?

"So we will make the url like this 'http://localhost:3000/assets/get/1' which looks a lot neater."

Let's change the `Asset` model to store the uploaded file in the "assets" folder rather than "system" folder. Change the code like so:

view plaincopy to clipboardprint?

```
1. has_attached_file :uploaded_file,
2.              :url => "/assets/get/:id",
3.              :path => ":Rails_root/assets/:id/:basename.:extension"
```

The option `:url` is for the url you would see in the address bar to download the file, whereas the `path` is for the place to actually store the file on your machine.

The `:Rails_root` is the physical root directory of your Rails app, such as "C:\web_apps\sharebox" (windows) or "/Users/phyowaiwin/Sites/apps/sharebox" (mac). The `:id` is the id of the Asset file record. The `:basename` is the file name (without extension) and `:extension` is for the file extension.

We are essentially instructing Paperclip to store the uploaded files in the "assets" folder, under the Rails root folder. The url would be the same as if we wanted: "http://localhost:3000/assets/get/1".

Now let's destroy the file at *http://localhost:3000/assets* and re-upload the file to *http://localhost:3000/assets/new*.

When you click on the file link, it should take you to a url like *http://localhost:3000/assets/get/2* which should display this error.

This is due to the fact that we haven't added any routes to handle this. Secondly, there isn't an action or controller to take care of the file download. Let's create a route first within the `config/routes.rb` file.

view plaincopy to clipboardprint?

```
1. #this route is for file downloads
2. match "assets/get/:id" => "assets#get", :as => "download"
```

This means, when the url is accessed, it should route to the `get` action at `assets_controller`. This also gives up the `download_url()` named-route. Let's make use of this route quickly in `app/views/assets/index.html.erb`:

```
1. <%= link_to asset.file_name, download_url(asset) %>
```

Okay, now we need to add the `get` action to `app/controllers/assets_controller.rb`.

view plaincopy to clipboardprint?

```
1. #this action will let the users download the files (after a simple authorization check)
2. def get
3. asset = current_user.assets.find_by_id(params[:id])
4. if asset
5.     send_file asset.uploaded_file.path, :type => asset.uploaded_file_content_type
6. end
7. end
```

"`find_by_id(:id)` will return null if no record with an `id` is found, whereas `find(:id)` will raise an exception if no record is found."

Above, we first grab the `id` and find it within the current_user's own assets. We then determine if the asset is found. If it is, we use the Rails method `send_file` to let the user download the file. `send_file` takes a "path" as a first parameter and file_type (content_type) as the second.

Now if you click the file link on the page, you should be able to download the file.

This is now much better. The download link is fairly secure now. You can now copy the URL of the download link (eg., *http://localhost:3000/assets/get/2*), logout from the application and go the link again. It should ask you to login first before you can download the file. Even after you've logged in, if you attempt to download a file which is not yours, you'll receive an error for now.

We'll put a nice message for users who are trying to access other people files. In the same `get` action of `assets_controller`, change the code like so:

view plaincopy to clipboardprint?

```
1. asset = current_user.assets.find_by_id(params[:id])
2. if asset
3.     send_file asset.uploaded_file.path, :type => asset.uploaded_file_content_type
```

```
4. else
5.     flash[:error] = "Don't be cheeky! Mind your own assets!"
6.     redirect_to assets_path
7. end
```

Above, we've added a flash message. To try this out, login to the system, and browse to a URL, like *http://localhost:3000/assets/get/22* (which you shouldn't have access to). As a result, you will be presented with this:

Further Potential Improvements

The above method for letting the users download files will most likely be fine for an application with a small number of users and traffic. But once you get more and more requests (for download), the Rails app will suffer from using the default `send_file` method, since the Rails process will read through the entire file, copying the contents of the file to the output stream as it goes.

However, it can be easily fixed by using `X-Sendfile`. It's available as an apache module. You can read more about how Rails handle downloading files with `send_file` here.

Step 5 Integrate Amazon S3

> "Now we will review how to store the files on Amazon S3 instead of your local machine. You may `skip` this step and move on to Step 6 if you don't require this functionality."

Since our application is for storing files and sharing them, we shouldn't ignore the opportunity to use Amazon S3. After all, most of us are already using it, aren't we?

Storing files on your local machine (or your server) will limit the application, when it comes to data storage. Let's make this app use S3 to store the files to prevent some headaches which might come up later.

As always, we don't want to waste time reinventing the wheel. Instead, why not use the brilliant gem, called `aws-s3` to help Paperclip use Amazon S3 to store the files. Let's add that to our Gemfile, and run `bundle install` in Terminal.

view plaincopy to clipboardprint?

```
1. #for Paperclip to use Amazon S3
2. gem "aws-s3"
```

Next, let's change the Asset model for the paperclip configuration to instruct it to store the files in S3. Go to `app/models/asset.rb` and change the `has_attached_file`, as below:

view plaincopy to clipboardprint?

```
1. has_attached_file :uploaded_file,
2.             :path => "assets/:id/:basename.:extension",
3.             :storage => :s3,
4.             :s3_credentials => "#{Rails_ROOT}/config/amazon_s3.yml",
5.             :bucket => "shareboxapp"
```

> "The Paperclip version we are using doesn't seem to create the `bucket` if it doesn't exist. So you need to `create the bucket first` on Amazon S3 first."

`:storage` says that we'll store the files in S3. You should put your bucket name in the `:bucket` option. Of course, S3 requires the proper credentials in order to upload the files. We'll provide the credentials in the `config/amazon_s3.yml` file. Create this file, and insert credentials. An example file is provided below:

```
1. development:
2.   access_key_id: your_own_access_key_id
3.   secret_access_key: your_own_secret_access_key
4.
5. staging:
6.   access_key_id: your_own_access_key_id
7.   secret_access_key: your_own_secret_access_key
8.
9. production:
10.  access_key_id: your_own_access_key_id
11.  secret_access_key: your_own_secret_access_key
```

> "The bucket namespace is shared by all users of the Amazon S3 system. So make sure that you use a `unique bucket name`."

If you restart your Rails server and upload a file, you should see the file within the S3 bucket. I use the amazing S3fox Firefox plugin to browse my S3 folders and files.

However, if you go to the index page of the `assets` controller and click on the newly uploaded file, you'll be faced with this error.

This error notes that the application can't find the file at *assets/3/Saturn.gif*. Although we have specified that we should -- within Paperclip's configuration -- use S3 as storage, we are still using the `send_file` method, which only sends the local files from your machine (or your server), not from a remote location, like Amazon S3.

Let's fix this by changing the `get` action in the assets_controller.rb

view plaincopy to clipboardprint?

```
1. def get
2.   asset = current_user.assets.find_by_id(params[:id])
3.   if asset
4.     #redirect to amazon S3 url which will let the user download the file automatically
5.     redirect_to asset.uploaded_file.url, :type => asset.uploaded_file_content_type
6.   else
7.     flash[:error] = "Don't be cheeky! Mind your own assets!"
8.     redirect_to assets_path
9.   end
10. end
```

We have just redirected the link to Amazon's S3 file. Let's see this in action by clicking the file link again on the index page of the assets. You will then be redirected to the actual location of the file stored on S3 such as *http://s3.amazonaws.com/shareboxapp/assets/3/Saturn.gif?1295915759*.

> "Exposing a downloadable link from your Amazon s3 is never a good thing, unless you intend to actually share it with others."

As you can see, this time the downloadable links aren't secure anymore. You can even download the files without logging in. We need to do something about it.

There are a couple of options available to us.

- We can try to download the file from S3 to the server (or your machine) first, and, in the background, send the file to the user.
- We could stream the file from S3 while opening it in a Rails process to server the data (file) back to the user.

We'll use the first option for now to progress quickly, as the second one is far more advanced to get it right, which will be outside of the scope of this tutorial.

So let's change the `get` action again in assets_controller.rb.

view plaincopy to clipboardprint?

```
1. def get
2.   asset = current_user.assets.find_by_id(params[:id])
3.
4.   if asset
5.     #Parse the URL for special characters first before downloading
6.     data = open(URI.parse(URI.encode(asset.uploaded_file.url)))
7.
8.     #then again, use the "send_data" method to send the above binary "data" as file.
9.     send_data data, :filename => asset.uploaded_file_file_name
10.
11.    #redirect to amazon S3 url which will let the user download the file automatically
12.    #redirect_to asset.uploaded_file.url, :type => asset.uploaded_file_content_type
13.  else
14.    flash[:error] = "Don't be cheeky! Mind your own assets!"
15.    redirect_to root_url
16.  end
17. end
```

We've added a couple of lines here. The first one is used for parsing the Amazon s3 url strings (for special characters like spaces etc) and opening the file there, using the "`open`" method. That will return the binary data, which we can then send as a file back to the user, using the `send_data` method in the next line.

If we return and click the file link again, you'll get the download file, as shown below, without seeing the actual Amazon s3 url.

Well that's better, right? Not quite. Because we are opening the file from Rails first, before passing it to the user, you could face a significant delay before you can download the file. But we will leave things as-is for simplicity's sake.

In the next step, we'll list the files properly on the home page.

Step 6 Show Files on the Home Page

Let's list the files nicely on the home page. We will show the files when the user goes to the `root_url` (which is at *http://localhost:3000*). Since the `root_url` actually directs to the `Index` action of the `Home` controller, we'll change the action now.

view plaincopy to clipboardprint?

```
1. def index
2.     if user_signed_in?
3.       @assets = current_user.assets.order("uploaded_file_file_name desc")
4.     end
5. end
```

Above, we loaded the *@assets* instance variable with the current_user's own assets -- if the user is logged in. We also order the files by the "file name".

Next, in `app/views/home/index.html.erb`, insert the following:

view plaincopy to clipboardprint?

```
<% unless user_signed_in? %>
  <h1>Welcome to ShareBox</h1>
  <p>File sharing web application you can't ignore.</p>

<% else %>

  <div class="asset_list_header">
    <div class="file_name_header">File Name</div>
    <div class="file_size_header">Size</div>
    <div class="file_last_updated_header">Modified</div>
  </div>
  <div class="asset_list">
    <% @assets.each do |asset| %>
      <div class="asset_details">
        <div class="file_name"><%= link_to asset.file_name, download_url(asset) %></div>
        <div class="file_size"><%= asset.uploaded_file_file_size %></div>
        <div class="file_last_updated"><%= asset.uploaded_file_updated_at %></div>
      </div>
    <% end %>
  </div>
<% end %>
```

We've added a condition to check if the user has logged in or not. If the user has not, he/she will see:

If the user has logged in, we'll use the *@assets* we set in the controller to loop through it to show the `File Name`, `File Size` and `Last modified date time`. Paperclip provides file_size information in the form of *[field_name]_file_size*. This will provide you with the total `bytes` of the file. What we mean by `Last modified date time` here is the time when the file was uploaded. Paperclip also provides it as *[field_name]_updated_at*.

> "Without the proper styling, what we have here looks quite ugly. So let's add some CSS."

Now, open the `public/stylesheets/application.css` file, and add the following styles to the bottom of the file.

view plaincopy to clipboardprint?

```
.asset_list_header, .asset_list, .asset_details {
    width:800px;
    font-weight:bold;
    font-size:12px;
    overflow:hidden;

}
.file_name_header, .file_name {
    width:350px;
    float:left;
    padding-left:20px;
}
.file_size_header, .file_size {
    width:100px;
    float:left;
}
.file_last_updated_header, .file_last_updated {
    width:150px;
    float:left;
}
.asset_list {
    padding:20px 0;
}
.asset_details {
    font-weight:normal;
    height:25px;
    line-height:25px;
    border:1px solid #FFF;
    width:790px;
    color:#4F4F4F;
}

.asset_details a, .asset_details a:visited {
```

```
    text-decoration:none;
    color:#1D96EF;
    font-size:12px;
}
```

I won't go much into the details here, as it exceeds the scope of this tutorial.

As you can see, the file size and last modified fields look quite weird at the moment. So let's add a method in the Asset model to help with File Size info. Go to `app/models/asset.rb` file and add this method.

view plaincopy to clipboardprint?

```
def file_size
    uploaded_file_file_size
end
```

This method acts as an alias for `uploaded_file_file_size`. You can also call *asset.file_size* instead. While we're here, let's make the *bytes* more readable. On the `app/views/home/index.html.erb` page, change the *asset.uploaded_file_file_size* to:

view plaincopy to clipboardprint?

```
number_to_human_size(asset.file_size, :precision => 2)
```

We have just used the `number_to_human_size` view helper from Rails to help with file size info.

Before we refresh the page to see the changes, let's add the following lines to the `config/environment.rb` file.

view plaincopy to clipboardprint?

```
#Formatting DateTime to look like "20/01/2011 10:28PM"
Time::DATE_FORMATS[:default] = "%d/%m/%Y %I:%M%p"
```

Always restart the Rails server when you make a change within your environment files."

Restart the Rails server and refresh the home page.

From the home page, we can't do much, except to view the files. Let's add an Upload button to the top.

view plaincopy to clipboardprint?

```
<% unless user_signed_in? %>
    <h1>Welcome to ShareBox</h1>
    <p>File sharing web application you can't ignore.</p>

<% else %>
    <div id="menu">
      <ul id= "top_menu">
          <li><%= link_to "Upload", new_asset_path %></li>
      </ul>
    </div>
    ...
<% end %>
```

The upload button is linked to creating a new asset_path as we wanted. Now we'll add the following CSS into the `application.css` file.

view plaincopy to clipboardprint?

```
#menu {
    width:800px;
    padding:0 20px;
    margin:20px auto ;
    overflow:hidden;
}
#menu ul{
    padding:0;
    margin:0;
}
#menu ul li {
    list-style:none;
    float:left;
    display:block;
    margin-right:10px;

```

```
}
#menu ul li a, #menu ul li a:visited{
    display:block;
    padding:0 15px;
    line-height:25px;
    text-decoration:none;
    color:#45759F;
    background:#EFF8FF;
    border:1px solid #CFEBFF;
}
#menu ul li a:hover, #menu ul li a:active{
    background:#DFF1FF;
    border:1px solid #AFDDFF;
}
```

If you refresh the page, it should look like the following:

Let's next customize the `app/views/assets/new.html.erb`.

view plaincopy to clipboardprint?

```
<% title "Upload a file" %>

<%= render 'form' %>

<p><%= link_to "Back to List", root_url %></p>
```

In next step, we'll learn how to create folders.

Step 7 Create Folders

> We need to organize our files and folders. The app needs to provide the ability to allow users to create folder structures, and upload files within them.

We can create folders virtually on the views by using the database table(model), called Folder. We won't be actually creating the folders on the file system or on the Amazon S3. We'll basically make the concept and add the feature effortlessly.

Let's begin by creating the `Folder` model. Run the following Scaffold command in the Terminal:

view plaincopy to clipboardprint?

```
Rails g nifty:scaffold Folder name:string parent_id:integer user_id:integer
```

With that command, we've added `name` for the name of the folder, and `user_id` for the relationship with the user. One user *has many* folders and one folder *belongs to* a user. We've also added a `parent_id` for storing the nested folders.

That `parent_id` field is also a necessity for the gem "acts_as_tree," which we will use to help with our nested folders. Now open the newly created migration file and add the following database indexes:

view plaincopy to clipboardprint?

```
add_index :folders, :parent_id
add_index :folders, :user_id
```

Run "`rake db:migrate`" to create the Folder model.

Then go to the `User` (`app/models/user.rb`) model to update this.

view plaincopy to clipboardprint?

```
has_many :folders
```

And go to Folder (`app/models/folder.rb`) model to update it as well.

view plaincopy to clipboardprint?

```
belongs_to :user
```

Let's add the `acts_as_tree` gem to the Gemfile.

view plaincopy to clipboardprint?

```
#For nested folders
gem "acts_as_tree"
```

We need to add this in the Folder model as part of the set up for `acts_as_tree`.

view plaincopy to clipboardprint?

```
class Folder < ActiveRecord::Base
  acts_as_tree
  ...
end
```

Now, run `bundle install`, and restart your Rails server.

> "`acts_as_tree` allows you to use methods, like `folder.children` to access sub-folders, and `folder.ancestors` to access root folders."

Next, within `app/controllers/folders_controller.rb`, change the following, which allows for securing the folders for the user who owns them:

view plaincopy to clipboardprint?

```
class FoldersController < ApplicationController
 before_filter :authenticate_user!

 def index
   @folders = current_user.folders
 end

 def show
   @folder = current_user.folders.find(params[:id])
 end

 def new
   @folder = current_user.folders.new
 end

 def create
   @folder = current_user.folders.new(params[:folder])
   ...
 end

 def edit
   @folder = current_user.folders.find(params[:id])
 end

 def update
   @folder = current_user.folders.find(params[:id])
   ...
 end

 def destroy
   @folder = current_user.folders.find(params[:id])
   ...
 end
end
```

Above, we added `before_filter :authenticate_user!` to the top, which requires users to first login in order to gain access. Secondly, we changed all the `Folder.new` or `Folder.find` to `current_user.folders.new` to make sure the user is viewing/accessing the folder that he or she owns.

Let's change the view for this. Open `app/views/folders/_form.html.erb`.

view plaincopy to clipboardprint?

```
<%= form_for @folder do |f| %>
 <%= f.error_messages %>
 <p>
   <%= f.label :name %><br />
   <%= f.text_field :name %>
 </p>
 <p><%= f.submit "Create Folder" %></p>
<% end %>
```

Here, we've removed the two fields ("user_id" and "parent_id").

You can now see the list of folders at *http://localhost:3000/folders*. You can also create new folders at *http://localhost:3000/folders/new*. Next, we'll put those folders on the home page.

Step 8 Display Folders on the Home Page

To display folders on the home page, we need to load the folders in an instance variable from the controller. Go ahead and open `app/controllers/home_controller.rb` to add this in the `index` action.

view plaincopy to clipboardprint?

```
def index
  if user_signed_in?
    #load current_user's folders
    @folders = current_user.folders.order("name desc")

    #load current_user's files(assets)
    @assets = current_user.assets.order("uploaded_file_file_name desc")
  end
end
```

We've added a line to load the user's folders into the *@folders* instance variable.

Then, on the `app/views/home/index.html.erb` page, update the code, like the following, to list the folders.

view plaincopy to clipboardprint?

```
<div class="asset_list">
  <!-- Listing Folders -->
  <% @folders.each do |folder| %>
    <div class="asset_details folder">
      <div class="file_name"><%= link_to folder.name, folder_path(folder) %></div>
      <div class="file_size"> - </div>
      <div class="file_last_updated"> - </div>
    </div>
  <% end %>

  <!-- Listing Files -->
  <% @assets.each do |asset| %>
    <div class="asset_details file">
      <div class="file_name"><%= link_to asset.file_name, download_url(asset) %></div>
      <div class="file_size"><%= number_to_human_size(asset.file_size, :precision => 2) %></div>
      <div class="file_last_updated"><%= asset.uploaded_file_updated_at %></div>
    </div>
  <% end %>
</div>
```

We've used the *@folders* variable to list the folders and link them to *folder_path*. Now, if you refresh the home page, you should see something like:

Though, it doesn't seem obvious which one is a folder, and which is a file. Since we've already added the CSS classes `file` and `folder`, we will only have to grab some images and use them in the stylesheet as background images.

You can use any images you like. For this tutorial, we'll use some of these images. Download them and place them in the `public/images` folder. Next, let's add some CSS to `application.css`.

view plaincopy to clipboardprint?

```
.folder  {
    background:url("../images/folder.png") no-repeat scroll left center transparent;
}

.file {
    background:url("../images/file.png") no-repeat scroll left center transparent;
}
.folder.asset_details:hover, .shared_folder.asset_details:hover, .file.asset_details:hover  {
    border:1px solid #DFDFDF;
}
.folder.asset_details:hover  {
    background:url("../images/folder.png") no-repeat scroll left center #EFEFEF;
}
.shared_folder  {
    background:url("../images/shared_folder.png") no-repeat scroll left center transparent;
}
.shared_folder.asset_details:hover  {
    background:url("../images/shared_folder.png") no-repeat scroll left center #EFEFEF;
}
.file.asset_details:hover  {
    background:url("../images/file.png") no-repeat scroll left center #EFEFEF;
}
```

If you refresh the home page, you should see folder and file icons aligned next to each folder and files you have.

It's missing "New Folder" button, though. Let's add it next to the "Upload" button, like below:

view plaincopy to clipboardprint?

```
1. <li><%= link_to "New Folder", new_folder_path %></li>
```

Great! What we have here is looking good. Next, we'll learn how to create nested folders, and display breadcrumbs.

Step 9 Handle Nested Folders and Create Breadcrumbs

In this step, we'll allow users to create folders inside other folders, using the `acts_as_tree` gem that we installed in Step 8.

Although we'll provide breadcrumbs navigation at the top of every page, we'll make the *URL* simple. Let's create a new route in the `config/routes.rb` file. Put this line near the top of the page.

view plaincopy to clipboardprint?

```
1. match "browse/:folder_id" => "home#browse", :as => "browse"
```

With this route, we'll now have URLs like *http://localhost:3000/browse/23* to see the folder (with id 23). It will direct to the *browse* action of the *home* controller. This folder might be nested within another, but we won't care that in the URL. Instead, we'll deal with it in the controller. Add the following *browse* in the Home controller.

view plaincopy to clipboardprint?

```
1.  #this action is for viewing folders
2.  def browse
3.    #get the folders owned/created by the current_user
4.    @current_folder = current_user.folders.find(params[:folder_id])
5.
6.    if @current_folder
7.
8.      #getting the folders which are inside this @current_folder
9.      @folders = @current_folder.children
10.
11.     #We need to fix this to show files under a specific folder if we are viewing that folder
12.     @assets = current_user.assets.order("uploaded_file_file_name desc")
13.
14.     render :action => "index"
15.   else
16.     flash[:error] = "Don't be cheeky! Mind your own folders!"
17.     redirect_to root_url
18.   end
19. end
```

`@current_folder.children` will give you all the folders of which the `@current_folder` is the *parent*.

Ok, let's look at the code above, line by line. First, we determine if this folder is owned/created by the current_user and put it into `@current_folder`. Then, we get all the folders which are inside that folder.

Since we are going to use the `app/views/home/index.html.erb` view to render this, we need to provide the `@assets` variable. At the moment, we are listing all assets owned by the current_user, not the ones under that folder. We'll take care of that in a bit.

Finally, if you're trying to view a folder not owned by you, you'll receive a similar message like before, and will be redirected back to the home page.

A folder should have many files (assets), and a file should belongs to a folder. So we need to add a `folder_id` to the Asset model. Run this command to create a migration file.

view plaincopy to clipboardprint?

```
1. Rails g migration add_folder_id_to_assets folder_id:integer
```

This will create a new migration file in the `db/migrate/` folder. Inside of this file, we'll add the index for `folder_id`, like so.

view plaincopy to clipboardprint?

```
1. class AddFolderIdToAssets < ActiveRecord::Migration
2.   def self.up
3.     add_column :assets, :folder_id, :integer
4.     add_index :assets, :folder_id
5.   end
6.
```

```
  def self.down
    remove_column :assets, :folder_id
  end
end
```

Next, run `rake db:migrate`. This should add a new column, `folder_id` into the Assets table. Add the new field to the Asset model (`app/models/asset.rb`). Also, add a relationship to the Folder model.

view plaincopy to clipboardprint?

```
attr_accessible :user_id, :uploaded_file, :folder_id

belongs_to :folder
```

view plaincopy to clipboardprint?

```
has_many :assets, :dependent => :destroy
```

This time, we've added `:dependent => :destroy`; this instructs Rails to *destroy* all the assets which belong to the folder... once the folder is *destroyed*.

> "If you are confused about *destroy* and *delete* in Rails, you might want to read this."

Back to the `browse` action in the Home controller, let's change the way we set the `@assets` variable.

view plaincopy to clipboardprint?

```
#this action is for viewing folders
def browse
  ...

  #show only files under this current folder
  @assets = @current_folder.assets.order("uploaded_file_file_name desc")

  ...
end
```

The `browse` action is now good to go. But, we need to revisit the

index

action again, because we don't want to display all files and folders on the index page once you are logged.

> "We wish to display only the *root* folders which have no parents, and only the files which are not under any folders."

To achieve this, let's change the `index` action, as follows:

view plaincopy to clipboardprint?

```
def index
  if user_signed_in?
    #show only root folders (which have no parent folders)
    @folders = current_user.folders.roots

    #show only root files which has no "folder_id"
    @assets = current_user.assets.where("folder_id is NULL").order("uploaded_file_file_name desc")
  end
end
```

`Folder.roots` will give you all root folders which have no parents. This is one of the useful scopes provided by the `acts_as_tree` gem.

We also put a condition on the assets to grab only the ones which have no `folder_id`.

We've got the structure to show files and folders correctly. We now have to change the links to these folders to go to `browse_path`. On the Home index page, change the link, like so:

view plaincopy to clipboardprint?

```
<!-- Listing Folders -->
...
        <div class="file_name"><%= link_to folder.name, browse_path(folder) %></div>
...
```

Let's start by creating a new route within the `routes.rb` to enable the creation of "sub folders."

view plaincopy to clipboardprint?

```

#for creating folders insiide another folder
```

```
3. match "browse/:folder_id/new_folder" => "folders#new", :as => "new_sub_folder"
```

Above, we've added a new route, called *new_sub_folder*, which will allow for URLs, like *http://localhost:3000/browse/22/new_folder*. This will reference the `new` action of the `Folder` controller. We'll essentially be creating a new folder under the folder (with id 22 or so).

Now, we have to make a `New Folder` button that will direct to that route, if you are within another folder (ie. if you are at URLs like *http://localhost:3000/browse/22*). Let's add a conditional change on the Home index page to the `New Folder button`.

view plaincopy to clipboardprint?

```
1. <% if @current_folder %>
2.     <li><%= link_to "New Folder", new_sub_folder_path(@current_folder) %></li>
3. <% else %>
4.     <li><%= link_to "New Folder", new_folder_path %></li>
5. <% end %>
```

We determine if the `@current_folder` exists. If it does, we know we are within a folder, thus, making the link direct to `new_sub_folder_path`. Otherwise, it'll still go to `normal new_folder_path`.

Now, it's time to change the `new` action in the Folder controller.

view plaincopy to clipboardprint?

```
1. def new
2.   @folder = current_user.folders.new
3.   #if there is "folder_id" param, we know that we are under a folder, thus, we will essentially create a subfolder
4.   if params[:folder_id] #if we want to create a folder inside another folder
5.
6.     #we still need to set the @current_folder to make the buttons working fine
7.     @current_folder = current_user.folders.find(params[:folder_id])
8.
9.     #then we make sure the folder we are creating has a parent folder which is the @current_folder
10.    @folder.parent_id = @current_folder.id
11.  end
12. end
```

We set the `parent_id` equal to the current_folder's `id` for the folder we are going to create.

view plaincopy to clipboardprint?

```
1. <%= form_for @folder do |f| %>
2.  <%= f.error_messages %>
3.  <p>
4.    <%= f.label :name %><br>
5.    <%= f.text_field :name %>
6.  </p>
7.    <%= f.hidden_field :parent_id %>
8.  <p><%= f.submit "Create Folder" %></p>
9. <% end %>
```

Next, we should take care of the `create` action to redirect to a correct path, once the folder has been saved.

view plaincopy to clipboardprint?

```
1. def create
2.   @folder = current_user.folders.new(params[:folder])
3.   if @folder.save
4.     flash[:notice] = "Successfully created folder."
5.
6.     if @folder.parent #checking if we have a parent folder on this one
7.       redirect_to browse_path(@folder.parent)  #then we redirect to the parent folder
8.     else
9.       redirect_to root_url #if not, redirect back to home page
10.    end
11.  else
12.    render :action => 'new'
13.  end
14. end
```

This code ensures that the user is correctly redirected to the folder he/she was browsing before creating the folder.

Lastly, let's correct the "Back to List" link on the Folder creation page. This file is located at `app/views/folders/new.html.erb`. Open it and change it, as follows:

view plaincopy to clipboardprint?

```
1. <% title "New Folder" %>
```

```
2.
3. <%= render 'form' %>
4.
5. <p>
6. <% if @folder.parent %>
7.   <%= link_to "Back to '#{@folder.parent.name}' Folder", browse_path(@folder.parent) %>
8. <% else %>
9.   <%= link_to "Back to Home page", root_url %>
10. <% end %>
11. </p>
```

The page should now have the necessary links, such as `Back to 'Documents' Folder`.

Now, let's create basic breadcrumb navigation for the pages.

Create a partial file, called `_breadcrumbs.html.erb` within the `app/views/home/` folder, and insert the following code:

view plaincopy to clipboardprint?

```
1. <div class="breadcrumbs">
2.   <% if @current_folder #checking if we are under any folder %>
3.     <%= link_to "ShareBox", root_url %>
4.     <% @current_folder.ancestors.reverse.each do |folder| %>
5.       » <%= link_to folder.name, browse_path(folder) %>
6.     <% end %>
7.      »  <%= @current_folder.name %>
8.   <% else #if we are not under any folder%>
9.     ShareBox
10.  <% end %>
11. </div>
```

"`@current_folder.ancestors` provides us with a folders array, which contains all the parent folders of the `@current_folder`."

In the code above, we are first determining if we are under any folders. If not (ie. if we are on the home page the first time), we simply show the text (not a link), "ShareBox."

Otherwise, we display a "ShareBox" link, so that users can return to the home page easily. Then, we use the `ancestors` method to get the parents folder and reverse it to display them correctly.

We finish it with the current_folder's name, which we display as plain text. Because the user is under that folder already, it doesn't need to be a link.

Now, let's add a bit of CSS to the applications.css file.

view plaincopy to clipboardprint?

```
1. .breadcrumbs {
2.   margin:10px 0;
3.   font-weight:bold;
4. }
5. .breadcrumbs a {
6.   color:#1D96EF;
7.   text-decoration:none;
8. }
```

Finally, we have to call the partial in both the home and folder creation pages. Go to `app/views/home/index.html.erb`, and add the following, just after the "menu" `div`.

view plaincopy to clipboardprint?

```
1. <%= render :partial => "breadcrumbs" %>
```

Also, add the following to `app/views/folders/new.html.erb`, just after the "title" line.

<%= render :partial => "home/breadcrumbs" %>

Note that you have to pass the "home" directory to call the partial from the "folders" directory, because they are both under the same directory. Let's see some screen shots of the breadcrumbs in action!

This breadcrumb navigation should do quite well, for our needs. Next, we'll add the ability to upload and store files inside a folder.

Step 10 Uploading Files to a Folder

Similar to creating a sub folder, we'll use the same type of route for uploading a (sub) file under a folder. Open `routes.rb`, and add:

view plaincopy to clipboardprint?

```
#for uploading files to folders
match "browse/:folder_id/new_file" => "assets#new", :as => "new_sub_file"
```

This will create url structures, like *http://localhost:3000/browse/22/new_file*. It will also hit the `new` action of Asset controller.

Change the `Upload` button on the home page to direct to the correct url, if we are browsing a folder. So edit the "top_menu" link, as demonstrated below:

view plaincopy to clipboardprint?

```
<ul id= "top_menu">
    <% if @current_folder %>
      <li><%= link_to "Upload", new_sub_file_path(@current_folder) %></li>
      <li><%= link_to "New Folder", new_sub_folder_path(@current_folder) %></li>
  <% else %>
      <li><%= link_to "Upload", new_asset_path %></li>
      <li><%= link_to "New Folder", new_folder_path %></li>
  <% end %>
</ul>
```

This code will make the `Upload` button direct to `new_asset_path`, if there's no `@current_folder`. If there is, on the other hand, it will direct to `new_sub_file_path`.

Next, change the `new` action of the Asset controller.

view plaincopy to clipboardprint?

```
def new
  @asset = current_user.assets.build
  if params[:folder_id] #if we want to upload a file inside another folder
   @current_folder = current_user.folders.find(params[:folder_id])
   @asset.folder_id = @current_folder.id
  end
end
```

Let's make a quick change to `app/views/assets/new.html.erb`. We need to have the correct "Back" url, as well as the breadcrumbs.

view plaincopy to clipboardprint?

```
<% title "Upload a file" %>
<%= render :partial => "home/breadcrumbs" %>

<%= render 'form' %>

<p>
<% if @asset.folder %>
  <%= link_to "Back to '#{@asset.folder.name}' Folder", browse_path(@asset.folder) %>
<% else %>
  <%= link_to "Back", root_url %>
<% end %>
</p>
```

This code should look quite similar to Folder's new page. We've added the breadcrumbs partial, and have personalized the `Back to` link to direct to the parent folder.

Then, in the `app/views/assets/_form.html.erb` file, add the hidden field, `folder_id`.

view plaincopy to clipboardprint?

```
<%= form_for @asset, :html => {:multipart => true} do |f| %>
 <%= f.error_messages %>

  <p>
    <%= f.label :uploaded_file, "File" %><br>
    <%= f.file_field :uploaded_file %>
   </p>
    <%= f.hidden_field :folder_id %>
 <p><%= f.submit "Upload" %></p>
<% end %>
```

Also, in the `Asset` controller, change the `create` action, like so:

view plaincopy to clipboardprint?

```
def create

  @asset = current_user.assets.build(params[:asset])
  if @asset.save
    flash[:notice] = "Successfully uploaded the file."

    if @asset.folder #checking if we have a parent folder for this file
      redirect_to browse_path(@asset.folder)  #then we redirect to the parent folder
    else
      redirect_to root_url
    end
  else
    render :action => 'new'
  end
end
```

This code will ensure that the user is redirected back to correct folder, once the upload is complete.

You should now be able to upload files to specific folders. Give it a try to make sure things work, before moving forward.

Step 11 Add Actions to Files and Folders

Now that we have files and folders listed nicely, we need to be able to edit/delete/download/share them. Let's start by creating some links on the home page.

> "We need to allow users to do three things to Folders: `Share`, `Rename` and `Delete`. However, for files, we'll only allow the user to perform two things: `Download` and `Delete`"

Return to the home page, and edit the folder list, as follows:

view plaincopy to clipboardprint?

```
<!-- Listing Folders -->
<% @folders.each do |folder| %>
    <div class="asset_details folder">
      <div class="file_name"><%= link_to folder.name, browse_path(folder) %></div>
      <div class="file_size">-</div>
      <div class="file_last_updated">-</div>
      <div class="actions">
        <div class="share">
          <%= link_to "Share" %>
        </div>
        <div class="rename">
          <%= link_to "Rename" %>
        </div>
        <div class="delete">
          <%= link_to "Delete" %>
        </div>
      </div>
    </div>
<% end %>
```

Let's also do the same for the file list:

view plaincopy to clipboardprint?

```
<!-- Listing Files -->
<% @assets.each do |asset| %>
      <div class="asset_details file">
      <div class="file_name"><%= link_to asset.file_name, download_url(asset) %></div>
      <div class="file_size"><%= number_to_human_size(asset.file_size, :precision => 2) %></div>
      <div class="file_last_updated"><%= asset.uploaded_file_updated_at %></div>
      <div class="actions">
        <div class="download">
          <%= link_to "Download" %>
        </div>
        <div class="delete">
          <%= link_to "Delete" %>
        </div>
      </div>
    </div>
<% end %>
```

We've added a download and delete links, above. Why not add some CSS to make these links look a bit nicer? Add the following to application.css

view plaincopy to clipboardprint?

```
.actions {
    float:rightright;
    font-size:11px;
}
.share, .rename, .download, .delete{
    float:left;
}
.asset_details .share a,.asset_details .rename a,.asset_details .download a,.asset_details .delete a{
    border: 1px solid #CFE9FF;
    font-size: 11px;
    margin-left: 5px;
    padding: 0 2px;
}
.asset_details .share a:hover,.asset_details .rename a:hover,.asset_details .download a:hover,.asset_details .delete a:hover{
    border:1px solid #8FCDFF;
}

.asset_details .delete a{
    color:#BF0B12;
    border:1px solid #FFCFD1;
}

.asset_details .delete a:hover{
    color:#BF0B12;
    border:1px solid #FF8F93;
}
```

Delete Files

Now, let's get those action links working. We'll begin with the Asset (File) delete links. Change the file `delete` link on the home page, like so:

view plaincopy to clipboardprint?

```
<%= link_to "Delete", asset, :confirm => 'Are you sure?', :method => :delete %>
```

This will pop up a confirmation message, once you click the link asking you if you are sure you want to delete it. Then it'll direct you to the destroy action of the Asset controller. So let's change that action to redirect:

view plaincopy to clipboardprint?

```
def destroy
  @asset = current_user.assets.find(params[:id])
  @parent_folder = @asset.folder #grabbing the parent folder before deleting the record
  @asset.destroy
  flash[:notice] = "Successfully deleted the file."

  #redirect to a relevant path depending on the parent folder
  if @parent_folder
   redirect_to browse_path(@parent_folder)
  else
   redirect_to root_url
  end
end
```

"Note that we need to get the `@parent_folder` of the file before it is deleted."

Delete Folders

Now it's time to change the folder delete link on the home page as follows:

view plaincopy to clipboardprint?

```
<%= link_to "Delete", folder, :confirm => 'Are you sure to delete the folder and all of its contents?', :method => :delete %>
```

"Remember: whenever we delete a folder, we also need to delete the contents (files and folders) within it."

And here's the Destroy action of the Folder Controller. So we'll edit the code there.

view plaincopy to clipboardprint?

```
def destroy
  @folder = current_user.folders.find(params[:id])
  @parent_folder = @folder.parent #grabbing the parent folder

  #this will destroy the folder along with all the contents inside
  #sub folders will also be deleted too as well as all files inside
  @folder.destroy
  flash[:notice] = "Successfully deleted the folder and all the contents inside."

  #redirect to a relevant path depending on the parent folder
  if @parent_folder
   redirect_to browse_path(@parent_folder)
  else
   redirect_to root_url
  end
end
```

This is the sam as we did in the Asset controller. Note that when you destroy a folder, all files and folders will be automatically be destroyed, because we have defined `:dependent => :destroy` in the Folder model for all files. Also, `acts_as_tree` will destroy all child folders, by default.

Download Files

Creating the "Download File" link is an easy one. Change the `Download` link in the files list on the home page.

view plaincopy to clipboardprint?

```
<%= link_to "Download", download_url(asset) %>
```

The `download_url(asset)` is already used on the file name link; so it shouldn't be a surprise for you.

Rename Folders

To rename a folder is to edit it. So we need to redirect the link to direct to the Edit action of the Folder controller. Let's do that with a new route to handle nested folder ids too. We can create this new route within the `routes.rb` file.

view plaincopy to clipboardprint?

```
#for renaming a folder
match "browse/:folder_id/rename" => "folders#edit", :as => "rename_folder"
```

Next, change the Rename link in the Folder list on the home page.

view plaincopy to clipboardprint?

```
<%= link_to "Rename", rename_folder_path(folder) %>
```

In the Folder Controller, change the Edit action, like so:

view plaincopy to clipboardprint?

```
def edit
  @folder = current_user.folders.find(params[:folder_id])
  @current_folder = @folder.parent    #this is just for breadcrumbs
end
```

"`@current_folder`, in the Edit action might not make sense at first, but we need the instance variable for displaying the breadcrumbs correctly."

Next, update the `app/views/folders/edit.html.erb` page:

view plaincopy to clipboardprint?

```
<% title "Rename Folder" %>
<%= render :partial => "home/breadcrumbs" %>

<%= render 'form' %>

<p>
<% if @folder.parent %>
  <%= link_to "Back to '#{@folder.parent.name}' Folder", browse_path(@folder.parent) %>
<% else %>
  <%= link_to "Back", root_url %>
<% end %>
</p>
```

We've simply added the breadcrumbs, and have updated the title and the `Back to` link.

Now we only need to change the "Create Folder" button to "Rename Folder." So, change `app/views/folders/_form.html.erb` to:

view plaincopy to clipboardprint?

```
<%= form_for @folder do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :name %><br>
    <%= f.text_field :name %>
  </p>
  <%= f.hidden_field :parent_id %>
  <p>
<% if @folder.new_record? %>
    <%= f.submit "Create Folder" %>
<% else %>
    <%= f.submit "Rename Folder" %>
<% end %>
  </p>
<% end %>
```

The page should now look like the following image, when you click to rename a folder.

Step 12 Sharing Folders Across Users

"In this ShareBox app, we need to make folders shareable. Any contents (Files and/or Folders) inside a shared folder will be accessible by each shared users."

Creating a Structure

We'll make the Sharing process easy with a few simple steps. The most likely scenario when a user shares a folder is:

- A user clicks on the *Share* link for a folder
- A dialog box will pop up to let the user enter email addresses to share the folder with
- The user may add an additional message to the invitation, if they wish
- Once the user invites a person (an email address), we'll store this information in the DB to inform the system to let that email address holder have access to the Shared folder.
- An email will be sent to the shared user to inform the invitation.
- Then the shared user signs in (or signs up first and then signs in) and he/she will see the shared folder.
- The shared user cannot perform any actions on the shared folders, other than downloading the file(s) inside them.

We need a new model handle the Shared Folders. Let's call it, `SharedFolder`. Run the following model generation script in the Terminal.

view plaincopy to clipboardprint?

```
Rails g model SharedFolder user_id:integer shared_email:string shared_user_id:integer folder_id:integer message:string
```

The `user_id` is for the owner of the shared folder. The `shared_user_id` is for the user to whom the owner has shared the folder to. The `shared_email` is for the email address of the shared_user. The `folder_id` is obviously the folder being shared. The `message` is for optional message to be sent in the invitation email.

In the newly created migration file (under `db/migrate` folder), add the following indexes:

view plaincopy to clipboardprint?

```
class CreateSharedFolders < ActiveRecord::Migration
  def self.up
    create_table :shared_folders do |t|
      t.integer :user_id
      t.string :shared_email
      t.integer :shared_user_id
      t.integer :folder_id
      t.string :message

      t.timestamps
    end

    add_index :shared_folders, :user_id
    add_index :shared_folders, :shared_user_id
    add_index :shared_folders, :folder_id
  end

  def self.down
    drop_table :shared_folders
  end
```

```
21. end
```

Now, run `rake db:migrate`. It'll create the DB table for you.

In the SharedFolder model, located at `app/models/shared_folder.rb`, add the following:

view plaincopy to clipboardprint?

```
1. class SharedFolder < ActiveRecord::Base
2.   attr_accessible :user_id, :shared_email, :shared_user_id,  :message,  :folder_id
3.
4.   #this is for the owner(creator) of the assets
5.   belongs_to :user
6.
7.   #this is for the user to whom the owner has shared folders to
8.   belongs_to :shared_user, :class_name => "User", :foreign_key => "shared_user_id"
9.
10.  #for the folder being shared
11.  belongs_to :folder
12. end
```

> "We are linking to the `User` model twice from the `ShareFolder` model. For the `shared_user` connection, we need to specify which class and which foreign key is used, because it's not following the Rails default conventions to link to `User` model."

Now in the `User` model, let's add those two relationships:

view plaincopy to clipboardprint?

```
1. #this is for folders which this user has shared
2. has_many :shared_folders, :dependent => :destroy
3.
4. #this is for folders which the user has been shared by other users
5. has_many :being_shared_folders, :class_name => "SharedFolder", :foreign_key => "shared_user_id", :dependent => :destroy
```

Then, in the Folder model, add this relationship.

view plaincopy to clipboardprint?

```
1. has_many :shared_folders, :dependent => :destroy
```

Creating View for Sharing

> "We are going to use jQuery and jQuery UI to create the invitation form for sharing the folders."

First off, we use jQuery instead of Prototype to prevent the possibility of CSRF attacks. By default, Rails uses the Prototype JS library to help with that.

We need to first download the jQuery version of the sort from here. You should download the zip file, extract it and copy the `jquery.Rails.js` file and paste it in our Rails app `public/javascripts` folder.

Then, we have to use that JS file instead of the default one. Open the `app/views/layout/application.html.erb` file, and change the following in the "head" section.

view plaincopy to clipboardprint?

```
1. <head>
2.   <title>ShareBox |<%= content_for?(:title) ? yield(:title) : "Untitled" %></title>
3.   <%= stylesheet_link_tag "application" %>
4.
5. <!-- This is for preventing CSRF attacks. -->
6.   <%= javascript_include_tag "jquery.Rails" %>
7.
8.
9.   <%= csrf_meta_tag %>
10.  <%= yield(:head) %>
11. </head>
```

We need to download jQuery UI from here. Make sure that you choose the `Redmond` theme to match our colors.

Once you've downloaded it, copy the *jquery-1.4.4.min.js* and *jquery-ui-1.8.9.custom.min.js* files and paste them into the `public/javascripts` folder.

Also copy the entire *redmond* folder from the CSS folder into `public/stylesheets`

Next, we need to load the files in our layout application file (`app/views/layouts/application.html.erb`) in the "head" section.

view plaincopy to clipboardprint?

```
1. <head>
2.   <title>ShareBox |<%= content_for?(:title) ? yield(:title) : "Untitled" %></title>
```

```
    <%= stylesheet_link_tag "application", "redmond/jquery-ui-1.8.9.custom" %>
<%= javascript_include_tag "jquery-1.4.4.min", "jquery-ui-1.8.9.custom.min" %>
<%= javascript_include_tag "application" %>

<!-- This is for preventing CSRF attacks. -->
  <%= javascript_include_tag "jquery.Rails" %>


  <%= csrf_meta_tag %>
  <%= yield(:head) %>
</head>
```

The `application.js` file is also being loaded here. We'll put our own JavaScript code in that file shortly.

We'll now create a jQuery UI dialog box to help with the invitation form. We'll make it load when the Share button is clicked.

First, open the `app/views/home/index.html.erb` page, and append the following near the bottom of the page, just before the last `<%= end %>`.

view plaincopy to clipboardprint?

```
<div id="invitation_form" title="Invite others to share" style="display:none">
    <% form_tag '/home/share' do -%>
        <label for="email_addresses">Enter recipient email addresses here</label><br />
        <%= text_field_tag 'email_addresses', '", :class => 'text ui-widget-content ui-corner-all'%>
        <br /><br />
        <label for="message">Optional message</label><br />
        <%= text_area_tag 'message','",  :class => 'text ui-widget-content ui-corner-all'%>
        <%= hidden_field_tag "folder_id" %>
    <% end -%>
</div>
```

Above, we've created an HTML form with one text field for an email address, and a textarea for the optional message. Also note that we have added a hidden field, called "folder_id," which we'll make use of to post the form later.

The *div* is hidden by default, and will be revealed in a dialog box when the Share button is clicked.

We are going to place the trigger in our JavaScript code. Before we do that, though, we need to have the specific *folder id* and *folder name* for the one that the user wants to share. To retrieve that, we need to change the Share folder link in the list of the Folder actions. On the home page in the folder list, revise the Share link, like this:

view plaincopy to clipboardprint?

```
<%= link_to "Share", "#", :folder_id => folder.id, :folder_name => folder.name %>
```

Now, the link will be generated:

view plaincopy to clipboardprint?

```
<a folder_name="Documents" folder_id="1" href="#">Share</a>
```

Every Share link for each folder will now have a unique folder_name and folder_id attribute.

We'll next create the trigger event in the `public/javascripts/application.js` file:

view plaincopy to clipboardprint?

```
$(function () {
    //open the invitation form when a share button is clicked
    $( ".share a" )
        .button()
        .click(function() {
            //assign this specific Share link element into a variable called "a"
            var a = this;

            //First, set the title of the Dialog box to display the folder name
            $("#invitation_form").attr("title", "Share '" + $(a).attr("folder_name") + "' with others" );

            //a hack to display the different folder names correctly
            $("#ui-dialog-title-invitation_form").text("Share '" + $(a).attr("folder_name") + "' with others");

            //then put the folder_id of the Share link into the hidden field "folder_id" of the invite form
            $("#folder_id").val($(a).attr("folder_id"));

            //Add the dialog box loading here

            return false;
        });
```

```
});
```

This code specifies that, whenever each link with a CSS class of "share" is clicked, we'll trigger an anonymous function. Refer to the comments above for additional details.

Now, we need to add the following code in the place of `//Add the dialog box loading here` to actually load the dialog box.

view plaincopy to clipboardprint?

```
//the dialog box customization
$( "#invitation_form" ).dialog({
    height: 300,
    width: 600,
    modal: true,
    buttons: {
        //First button
        "Share": function() {
            //get the url to post the form data to
            var post_url = $("#invitation_form form").attr("action");

            //serialize the form data and post it the url with ajax
            $.post(post_url,$("#invitation_form form").serialize(), null, "script");

            return false;
        },
        //Second button
        Cancel: function() {
            $( this ).dialog( "close" );
        }
    },
    close: function() {

    }
});
```

The dialog box should be loaded with the specified width and height, and it also has two buttons: Share and Cancel. When the Share button is clicked, we'll perform two actions. First, we store the Form's action (which is "home/share") into a variable, called `post_url`. Secondly, we post the form, using AJAX, to the `post_url` after serializing the form data.

> "The last parameter value *script* of AJAX method, *$.post()*, instructs Rails to respond when the AJAX request has completed."

Before we test this out in our browser, let's add a bit of styling in the application.css file.

view plaincopy to clipboardprint?

```
.ui-button-text-only .ui-button-text {
    font-size: 14px ;
    padding: 3px 10px !important;
}
.share .ui-button-text-only .ui-button-text {
    padding: 0 2px !important;
    font-size:11px;
    font-weight:normal;
}
#invitation_form{
    font-size:14px;
}
#invitation_form label{
    font-weight:bold;
}
#invitation_form input.text {
    width:480px;
    height:20px;
    font-size:12px;
    color:#7F7F7F;
}
#invitation_form textarea {
    width:480px;
    height:100px;
    font-size:12px;
    color:#7F7F7F;
}
```

Since we don't currently have any route matched for *home/share*, submitting the form will silently fail in the background, as it's an AJAX request.

We'll create a new route in the `routes.rb` file.

view plaincopy to clipboardprint?

```
#for sharing the folder
match "home/share" => "home#share"
```

This route will direct to the Share action of the Home controller. So let's create that action now.

view plaincopy to clipboardprint?

```
#this handles ajax request for inviting others to share folders
def share
    #first, we need to separate the emails with the comma
    email_addresses = params[:email_addresses].split(",")

    email_addresses.each do |email_address|
      #save the details in the ShareFolder table
      @shared_folder = current_user.shared_folders.new
      @shared_folder.folder_id = params[:folder_id]
      @shared_folder.shared_email = email_address

      #getting the shared user id right the owner the email has already signed up with ShareBox
      #if not, the field "shared_user_id" will be left nil for now.
      shared_user = User.find_by_email(email_address)
      @shared_folder.shared_user_id = shared_user.id if shared_user

      @shared_folder.message = params[:message]
      @shared_folder.save

      #now we need to send email to the Shared User
    end

    #since this action is mainly for ajax (javascript request), we'll respond with js file back (refer to share.js.erb)
    respond_to do |format|
      format.js {
      }
    end
end
```

Here, we are splitting the email addresses with a comma, since we can share one folder with multiple email addresses. Then, we loop through each email address to create a *SharedFolder* record. We make sure we have the correct user id (owner) and folder id.

> "Getting a shared user is a bit tricky. The email address holder may or may not be an account holder of this ShareBox app."

To compensate, we'll grab the `shared_user` only when we can find the email address holder. If we can't find it, we'll leave it as null in the `shared_user_id` for now. We'll take care of that in a bit. We'll also save the optional message.

Let's now create the "share.js.erb" file in the `app/views/home` folder, as the AJAX request will be expecting a JavaScript response.

view plaincopy to clipboardprint?

```
//closing the dialog box
$("#invitation_form").dialog("close");

//making sure we don't display the flash notice more than once
$("#flash_notice").remove();

//showing a flash message
$("#container").prepend("<div id="flash_notice">Successfully shared the folder</div>");
```

We close the dialog box first. Then, we remove any existing Flash messages, and, finally, we display a new Flash message to inform the user that the folder has been shared.

But I want the *shared* folder icons to be different from the normal folder icon. First, I need to know if a folder is shared or not. We can add a quick method to the Folder model to help with this task.

view plaincopy to clipboardprint?

```
#a method to check if a folder has been shared or not
def shared?
    !self.shared_assets.empty?
end
```

> We can determine if `a_folder` is shared or not by calling `a_folder.shared?`, which returns a boolean.

Next, we need to know which folder element to change in our jQuery. So we can add a `folder_id` to every folder line. Within home/index page, update the following line:

view plaincopy to clipboardprint?

```
<div class="asset_details folder">
```

...and replace it with:

view plaincopy to clipboardprint?

```
<div class="asset_details <%= folder.shared? ? 'shared_folder' : 'folder' %>" id="folder_<%= folder.id %>">
```

This assigns a new CSS class, called "shared_folder," to the `div` if the folder is shared. Also we add a folder_id to let jQuery dynamically change the icons by switching the CSS class.

In fact, we already added the CSS class, "shared_folder" earlier. So it should look something like this once you share a folder.

Add the following two lines in the `share.js.erb` to change the folder icons dynamically after the Ajax request.

view plaincopy to clipboardprint?

```
//Removing the css class 'folder'
$("#folder_<%= params[:folder_id] %>").removeClass("folder");

//Adding the css class 'shared_folder'
$("#folder_<%= params[:folder_id] %>").addClass("shared_folder");
```

Before we go back and work on the email section, we need to place an "after_create" method in the User model to be executed every time a new user is added. This will be to sync the new user id with the SharedFolder's `shared_user_id` if the email address is matched. We add this in the User model.

view plaincopy to clipboardprint?

```
after_create :check_and_assign_shared_ids_to_shared_folders

#this is to make sure the new user ,of which the email addresses already used to share folders by others, to have access to those folders
def check_and_assign_shared_ids_to_shared_folders
    #First checking if the new user's email exists in any of ShareFolder records
    shared_folders_with_same_email = SharedFolder.find_all_by_shared_email(self.email)

    if shared_folders_with_same_email
      #loop and update the shared user id with this new user id
      shared_folders_with_same_email.each do |shared_folder|
        shared_folder.shared_user_id = self.id
        shared_folder.save
      end
    end
end
```

Here, we get the new user id for the purpose of putting it in the `shared_user_id` of the SharedFolder object if any of the records have the same email address as the new user's. That should keep it synced with the non-users who you are trying to share your folder with, on ShareBox.

Step 13 Sending Emails to Shared Users

Now, we'll review how to send emails when a user shares a folder with others.

> "Sending emails in Rails 3 is quite easy. We'll use Gmail, since you can easily and quickly create Gmail accounts to test this bit of the tutorial."

First, we need to add the SMTP settings to the system. So, create a file named "setup_mail.rb" within the `config/initializers` folder. Append the following settings to the file:

view plaincopy to clipboardprint?

```
ActionMailer::Base.smtp_settings = {
:address              => "smtp.gmail.com",
:port                 => 587,
:domain               => "gmail.com",
:user_name            => "shareboxapp",
:password             => "secret",
:authentication       => "plain",
:enable_starttls_auto => true
}
```

> "Don't forget to restart the Rails server to reload these settings."

We next need to create mailer objects and views. Let's call it, "UserMailer". Run the following generator:

```
Rails g mailer user_mailer
```

This command will create the `app/mailers/user_mailer.rb` file among others. Let's edit the file to create a new email template.

view plaincopy to clipboardprint?

```
class UserMailer < ActionMailer::Base
  default :from => "shareboxapp@gmail.com"

  def invitation_to_share(shared_folder)
    @shared_folder = shared_folder #setting up an instance variable to be used in the email template
    mail( :to => @shared_folder.shared_email,
          :subject => "#{@shared_folder.user.name} wants to share '#{@shared_folder.folder.name}' folder with you" )
  end
end
```

You can set the default values with the `default` method there. We'll basically create a method, which accepts the `shared_folder` object and pass it on to the email template, which we'll create next.

Now, we have to create the email template. The file name should be the same as the method's name you have in the UserMailer. So let's create the `invitation_to_share.text.erb` file under `app/views/user_mailer` folder.

We name it as `invitation_to_share.text.erb` to use Text-based emails. If you'd like to use HTML-based emails, name it, `invitation_to_share.html.erb`

Insert the following wording.

view plaincopy to clipboardprint?

```
Hey,

<%= @shared_folder.user.name %> has shared the "<%= @shared_folder.folder.name %>" folder on Sharebox.

Message from <%= @shared_folder.user.name %>:
"<%= @shared_folder.message %>"

You can now login at <%= new_user_session_url %> and view the folder if you have an account already. If you don't have one,  you can sign up here at <%= new_user_registra

Have fun,

Sharebox
```

To make this work, we add the following code to the `share` action of the Home controller where we left space to send emails.

view plaincopy to clipboardprint?

```
#now send email to the recipients
UserMailer.invitation_to_share(@shared_folder).deliver
```

Now, if you share a folder, it'll send an email to the shared user. Once you click the "Share" button within the dialog box, you might have to wait a few seconds or so to let the system send an email to the user before you see the flash message.

Step 14 Giving Shared Folder Access to Other Users

Thus far, even though a user can share his folder with others, they will not see his folder yet on their ShareBox pages."

To fix this, we need to display the shared folders to the shared users on the home page. We definitely have to pass a new instance variable loaded with these shared folders to the Index page.

Update your index action in the Home controller, like so:

view plaincopy to clipboardprint?

```
def index
  if user_signed_in?
    #show folders shared by others
    @being_shared_folders = current_user.shared_folders_by_others

    #show only root folders
    @folders = current_user.folders.roots
    #show only root files
    @assets = current_user.assets.where("folder_id is NULL").order("uploaded_file_file_name desc")
  end
end
```

In this code, we've added the `@being_shared_folders` instance variables. Note that we don't currently have the relationship, called `shared_folders_by_others`, in the `User` model; so, let's add that now.

view plaincopy to clipboardprint?

```
#this is for getting Folders objects which the user has been shared by other users
has_many :shared_folders_by_others, :through => :being_shared_folders, :source => :folder
```

Now let's show that shared folders list on the home/index page. Insert the following code, just before the normal Folder listing:

view plaincopy to clipboardprint?

```
<!-- Listing Shared Folders (the folders shared by others) -->
<% @being_shared_folders.each do |folder| %>
  <div class="asset_details <%= folder.shared? ? 'shared_folder' : 'folder' %>" id="folder_<%= folder.id %>">
    <div class="file_name"><%= link_to folder.name, browse_path(folder) %></div>
    <div class="file_size">-</div>
    <div class="file_last_updated">-</div>
    <div class="actions">
    </div>
  </div>
<% end %>

<!-- Listing Folders -->
...
```

"Note that we won't provide any action links for Shared Folders, since they don't belong to the shared user."

This should work fine on the home page. You should now be able to see others' shared folders at the top of the pages.

But if you're in a subfolder (of your own), you will receive a `nil object` error for `@being_shared_folders`. We need to fix that, and can do so in the `browse` action of the `Home` controller. Visit that page, and add:

view plaincopy to clipboardprint?

```
#this action is for viewing folders
def browse
  #making sure that this folder is owned/created by the current_user
  @current_folder = current_user.folders.find(params[:folder_id])

  if @current_folder
    #if under a sub folder, we shouldn't see shared folders
    @being_shared_folders = []

    ...
  else
    ...
  end
end
```

This code ensures that we don't wish to see folders shared by others. Those folders should only exist at the Root level. The above should fixes this issue.

Viewing a Folder Shared by Others

Now if you try to browse into a folder shared by others, you'll get this error:

This is due to the fact that the system thinks you are trying to gain access to a folder, for which you don't have the proper privileges. The reason we are seeing this error, instead of the "*Don't be cheeky!*" message, is because we have used the `find()` method instead of `find_by_id()`, while getting the `@current_folder` in the `browse` action of `Home` controller.

Now once again, we have to rethink the logic. We need to set the `@current_folder` for folders shared by others, but we also must pass some sort of flag to the views, which specifies that this folder is shared by others. That way, the views can restrict the access privileges -- such as deleting folders, etc.

To handle this, wee need a method for the `User` model, which determines if the user has "Share" access to the folder specified. So let's add this first before we change it in the `Browse` action.

view plaincopy to clipboardprint?

```
#to check if a user has acess to this specific folder
def has_share_access?(folder)
  #has share access if the folder is one of one of his own
  return true if self.folders.include?(folder)

  #has share access if the folder is one of the shared_folders_by_others
  return true if self.shared_folders_by_others.include?(folder)

```

```
    #for checking sub folders under one of the being_shared_folders
    return_value = false

    folder.ancestors.each do |ancestor_folder|

      return_value = self.being_shared_folders.include?(ancestor_folder)
      if return_value #if it's true
        return true
      end
    end

    return false
  end
```

Again, this method determines if the user has share access to the folder. The user has share access if he/she owns it. Alternatively, the user has share access if the folder is one of the user's Folders shared By others.

If none of the above return `true`, we still need to check one last thing. Although the folder you are viewing might not exactly be the folder shared by others, it could still be a subfolder of one of the folders shared by others, in which case, the user does, indeed, have share access.

So the code above checks the folder's ancestors to validate this.

Back to the browse action of the `Home` controller; insert the following code there:

view plaincopy to clipboardprint?

```
def browse
  #first find the current folder within own folders
  @current_folder = current_user.folders.find_by_id(params[:folder_id])
  @is_this_folder_being_shared = false if @current_folder #just an instance variable to help hiding buttons on View

  #if not found in own folders, find it in being_shared_folders
  if @current_folder.nil?
    folder = Folder.find_by_id(params[:folder_id])

    @current_folder ||= folder if current_user.has_share_access?(folder)
    @is_this_folder_being_shared = true if @current_folder #just an instance variable to help hiding buttons on View

  end

  if @current_folder
    #if under a sub folder, we shouldn't see shared folders
    @being_shared_folders = []

    #show folders under this current folder
    @folders = @current_folder.children

    #show only files under this current folder
    @assets = @current_folder.assets.order("uploaded_file_file_name desc")

    render :action => "index"
  else
    flash[:error] = "Don't be cheeky! Mind your own assets!"
    redirect_to root_url
  end
end
```

This code accomplishes two main things:

- Assigns the `@current_folder`, even if you are under a subfolder of a folder shared by others
- Assigns the flag, `@is_this_folder_being_shared`, to pass it to the views. This will inform us as to whether the `@current_folder` is a folder shared by others or not.

Downloading the Files from Folders Shared By Others

If you attempt to download a file from a folder shared by others, you'll be met with the "*Don't be cheeky!*" message. So, let's adjust the `get` action of the `Asset` controller now.

view plaincopy to clipboardprint?

```
def get
  #first find the asset within own assets
  asset = current_user.assets.find_by_id(params[:id])

  #if not found in own assets, check if the current_user has share access to the parent folder of the File
  asset ||= Asset.find(params[:id]) if current_user.has_share_access?(Asset.find_by_id(params[:id]).folder)

```

```
8.  if asset
9.    #Parse the URL for special characters first before downloading
10.   data = open(URI.parse(URI.encode(asset.uploaded_file.url)))
11.   send_data data, :filename => asset.uploaded_file_file_name
12.   #redirect_to asset.uploaded_file.url
13. else
14.   flash[:error] = "Don't be cheeky! Mind your own assets!"
15.   redirect_to root_url
16. end
17. end
```

Above, we've added a line to assign the `asset` variable if the user has shared access to the folder of that asset (file).

Restricting Actions Available to Shared Users

We need to restrict access to the top Buttons: "Upload" and "New Folder". Open the `app/views/home/index.html.erb` file and alter the `top_menu` ul list, like so:

view plaincopy to clipboardprint?

```
1.  <% unless @is_this_folder_being_shared %>
2.    <ul id= "top_menu">
3.      <% if @current_folder %>
4.        <li><%= link_to "Upload", new_file_path(@current_folder) %></li>
5.        <li><%= link_to "New Folder", new_sub_folder_path(@current_folder) %></li>
6.      <% else %>
7.        <li><%= link_to "Upload", new_asset_path %></li>
8.        <li><%= link_to "New Folder", new_folder_path %></li>
9.      <% end %>
10.   </ul>
11. <% else %>
12.   <h3>This folder is being shared to you by <%= @current_folder.user.name %></h3>
13. <% end %>
```

We're using the `@is_this_folder_being_shared` variable to determine if the current_folder is indeed a folder shared by others or not. If it is, we'll hide them and display a message.

On this same page, near the `@folders` list, adjust the actions, as shown below:

view plaincopy to clipboardprint?

```
1.  <div class="actions">
2.    <div class="share">
3.      <%= link_to "Share", "#", :folder_id => folder.id, :folder_name => folder.name unless @is_this_folder_being_shared%>
4.    </div>
5.    <div class="rename">
6.      <%= link_to "Rename", rename_folder_path(folder) unless @is_this_folder_being_shared%>
7.    </div>
8.    <div class="delete">
9.      <%= link_to "Delete", folder, :confirm => 'Are you sure to delete the folder and all of its contents?', :method => :delete unless @is_this_folder_being_shared%>
10.   </div>
11. </div>
```

This code restricts actions on the subfolders of a folder shared by others.

Next, on the `Delete` action of the file, add:

view plaincopy to clipboardprint?

```
1.  <div class="delete">
2.    <%= link_to "Delete", asset, :confirm => 'Are you sure?', :method => :delete unless @is_this_folder_being_shared%>
3.  </div>
```

This ensures that the actions are now nicely secure for the shared users.

Conclusion

Although there are certain plenty of additional things to cover to transform this app into a full-blown file sharing web site, this will provide with a solid start.

We've implemented each essential feature of a typical file sharing, including creating (nested) folders, and using emails for invites.

In terms of the techniques we've used in this tutorial, we covered several topics, ranging from uploading files with Paper clip to the use of jQuery UI for the modal dialog box and sending post requests with AJAX.

This was a massive tutorial; so, take your time, read it again, work along with each step, and you'll be finished in no time!