Quora Clone Part 2: Creating Users, Authentication

Time	~ 3/4 day
Learning Goals	• Deeper understanding of MVC and Active Record associations • Integrating user authentication system and managing views for user authorizat

Set Up Instructions

1. Fork the repo from: https://github.com/NextAcademy/quora-clone
2. Clone the repo to your computer: `$ git clone <repo-link>`
3. `cd` into the app folder. Run `$ bundle install` to install the necessary gems.
4. Launch Your App Server by typing `$ shotgun config.ru` in the command line. You should now be able to visit the skeleton app at http://localhost:9393

Build Your User Stories

Before you start building the question and answer features, let's build out these few user stories:

1. A `user` can register
2. A `user` can log in and log out
3. A `user` profile page

It's good practice not to work on the master branch. Remember to `git checkout -b feature/user` and work on the branch for this feature!

Story #1: Building User Signup

Let's create a feature that allows a new visitor to create an account.

Design a DATABASE SCHEMA and MIGRATION file

When signing up as a user, I should enter my full name, my email, and a password. What fields should there be in the `users` table? A `user` will be logging in by submitting their `email` and `password` somewhere. Run the relevant rake tasks to create your user migration file.

Set up MODEL

The model should be simple. However, by now, you should have built at least one app, so let's go a step further! Let's think about data validity and security.

Quora Clone Part 2: Creating Users, Authentication

Ask yourself:

- What if a user tries to input an email or password that is invalid in terms of format or length?
- What if 2 users key in the same email?
- Should passwords be of a minimum length?
- What if hackers managed to get hold of read access to your entire database? Should you store plain text password in the database? A lot of users use the same passwords for all their web services. Perhaps we should encrypt or do one-way password hashing?
- What if a user inputs an email that has invalid format such as `nextacademy.com` or `joshgmail.com`?

Key things to do:

1. Use Active Record validations.
2. DO NOT store plain text password. Check out this documention on `has_secure_password` and read up on why we should use Bcrypt. (Check out this video on “how not to store passwords”).

Create a VIEW

Create a signup page for users to sign up using their email and password. The view file could look something like this:

```
<h2>Sign up for an account!</h2>

<form method="post" action="/signup">

  <input type=text placeholder="Email" name="user[email]"><br>

  <input type=password placeholder="Password" name="user[password]"></br>

  <input type=submit value="Sign up">

</form>
```

Set up ROUTE and Create a CONTROLLER

We’ll also need corresponding controllers for users to POST `/signup` as a user. If the user sent a valid email and password, store their data in the database.

Quora Clone Part 2: Creating Users, Authentication

Start with this template for your users controller (`users_controller.rb`):

```
# In users_controller.rb
post '/signup' do
  user = User.new(params[:user])
  if user.save
    # what should happen if the user is saved?
  else
    # what should happen if the user keyed in invalid date?
  end
end
```

Extra note:

Note that the form had the name in the format `name="user[email]"` and `name="user[password]"`. In the controller, we can use `User.new(params[:user])` instead of `User.new(email: params[:user][:email] ...)`. This means using the `name="user[email]"` format automatically transforms the value sent by the form into a hash. Note that this is mostly specific to Rails (and Sinatra). But if you do raw Ruby or any other language that doesn't follow this convention, it's not always the case that the params will be converted into a nested hash.

Story #2: User can log in and log out

Now that we have set up a user sign up system, let's setup a feature such that the user can log in with their valid email and password and log out.

The flow

```
User visits log in page ->
```

Quora Clone Part 2: Creating Users, Authentication

```
User keys in email and password ->

User hits submit ->

Our app should check if they keyed in the correct credentials ->

If yes, log the user in. If not, show an error.
```

We can start with the view or controller or model. As you build more apps, you will eventually find a sequence that works best for you.

Login form for your VIEW

```
<h2>Login</h2>

<form method=post action=/login>

  <input type=text placeholder="Email" name="user[email]"><br>

  <input type=password placeholder="Password" name="user[password]"></br>

  <input type=submit value="Login">

</form>
```

You may also want to add buttons that link to your login page.

Login ROUTES and CONTROLLER

Let's set up a route and controller that listens to the form. How do we know whether they submitted the correct password or not?

```
post '/login' do

 # apply a authentication method to check if a user has entered a valid email and password

 # if a user has successfully been authenticated, you can assign the current user id to a session

end



post '/logout' do
```

Quora Clone Part 2: Creating Users, Authentication

```
# kill a session when a user chooses to logout, for example, assign nil to a session
# redirect to the appropriate page
end
```

To understand more about `sessions`, read the [using sessions section of the Sinatra documentation](). Next, ask for help from a staff member if it's still unclear what you need to store and how you store it.

Extend your MODEL

If you haven't already, let's move the logic of authenticating a user out of the CONTROLLER into our MODEL. Best practice dictates that business logics should live in your MODEL or CONCERNS and not within your CONTROLLER. Let's write a method that authenticates a user!

```
post '/login' do
# apply a authentication method to check if a user has entered a valid email and password
# if a user has successfully been authenticated, you can assign the current user id to a session
end


post '/logout' do
# kill a session when a user chooses to logout, for example, assign nil to a session
# redirect to the appropriate page
end
```

Story #2a: Enhancing experience

Should we show the log in button to a user who is already logged in? Let's write a method to check if a user is currently log in.

Using a helper

Quora Clone Part 2: Creating Users, Authentication

Wouldn't it be great to remember a logged in user when navigating between pages? Let's create some convenience/helpers methods we can use. We shall put these methods in users' helper at `app/helpers/user.rb`. It should contain a method that works like this:

```
helpers do
 # This will return the current user, if they exist
 # Replace with code that works with your application
 def current_user
   if session[:user_id]
     @current_user ||= User.find_by_id(session[:user_id])
   end
 end

 # Returns true if current_user exists, false otherwise
 def logged_in?
   !current_user.nil?
 end
end
```

Your VIEW and CONTROLLER can now call current_user to get the currently authenticated user, if they exist! Now with those helper methods, hide the `log in` button and page if a already logged in user tries to do it!

Story #3: Profile Page

Create a profile page for each user. In the controller, you would have something that look like this:

Quora Clone Part 2: Creating Users, Authentication

```
get '/users/:id' do
 # some code here
end
```

We will leave it to you to create the view for the profile page. You can later add on the questions and answers posted by the users on this page, and customize it to suit your design.

Once you are done, congratulations, you can now merge back to your master branch and move on to the next feature!