Lecture 5 Authentication

Deliverables

- [ ] Add a signup feature to API
- [ ] Add a login feature to API to include authentication
- [ ] Add a users show route that will authenticate a users session on the front end as "/me"
- [ ] Add a logout feature to API

Important Talking Points

- Cookies
- Sessions
- Authentication
- Stateless HTTP

The current problem we are aiming to solve is that HTTP is stateless. It does not remember important information, particularly about a user, between requests. Each request is handled individually.

To solve this error, we set sessions and cookies that are shared between the server and client and vise versa to keep track of certain information, including our users login.

Configuring Rails app to use cookies and sessions

- Inside `config/application.rb`

```
module MyApp
  class Application < Rails::Application
    config.load_defaults 6.1
    config.api_only = true

    config.middleware.use ActionDispatch::Cookies
    config.middleware.use ActionDispatch::Session::CookieStore

    config.action_dispatch.cookies_same_site_protection = :strict #
ensures that cookies are only shared on the same domain.
  end
end
```

- Inside `app/controllers/application_controller.rb` give access to cookies to all subsequent controllers

```
class ApplicationController < ActionController::API
  include ActionController::Cookies
end
```

/signup

The user is here because they have not set up an account. Essentially, we are going to create a new user object with the submitted form data.

The flow:

1. Client: At `'/signup'`, render a signup form.
2. Client: When user submits the form, make a `POST` fetch request to the endpoint `/signup`, with form data included.
3. Server: `users#create` will validate the newly created user object. If valid, return a serialized user with a status of ok. If invalid, provide error messages and failure status.

Setup:

- Inside Gemfile, comment back in `bcrypt` and run `bundle update`

`bcrypt` allows us to call on the `has_secure_password` which adds multiple methods to the user model for reading, writing and encrypting passwords.

```
password=
password_confirmation=
authenticate
```

- Inside User model add:

```
has_secure_password
```

Encrypted passwords will be stored inside of a special column called `password_digest`.

1. Let's create a new column in the `users` table: `password_digest`

```
rails g migration AddPasswordDigestToUsers password_digest
```

run `rails db:migrate`

Add `password` to `UsersController` strong params

```
  def user_params
      params.permit(:username, :email, :password, :password_confirmation)
  end
```

WARNING: Do not serialize user objects with password. Must be excluded. No changes are necessary to the current `UserSerializer`

Add an endpoint to handle a signup request:

```
get '/signup', to: "users#create"
```

At the moment, we have a `current_user` method inside the ApplicationController that is hardcoded to return a user object. We want to make this more dynamic based on the stored session. Let's update `current_user` to:

```
    def current_user
        User.find_by(id: session[:user_id])
    end
```

Update `create` in UsersController:

```
  def create
    user = User.create(user_params)
    session[:user_id] = user.id # this is the piece that logs a user in
and keeps track of users info in subsequent requests.
    render json: user, status: :created
  end
```

On the front end:

```
import React, { useState } from "react";

const SignupForm = () => {
  const [formData, setFormData] = useState({
    username: "",
    password: "",
    email: "",
  });

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value,
    });
  };
  function handleSubmit(e) {
    e.preventDefault();

    const userCreds = { ...formData };

    fetch("/users", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(userCreds),
    }).then((res) => {
      if (res.ok) {
        res.json().then((user) => {
          setCurrentUser(user);
        });
      } else {
        res.json().then((errors) => {
          console.error(errors);
        });
      }
    });
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="username">Username:</label>
      <input
        id="username-signup-input"
        type="text"
        name="username"
        value={formData.username}
        onChange={handleChange}
      />
      <label htmlFor="email">Email:</label>
      <input
        id="email-signup-input"
        type="text"
        name="email"
        value={formData.email}
        onChange={handleChange}
      />
      <label htmlFor="password">Password:</label>
      <input
        id="password-signup-input"
        type="password"
        name="password"
        value={formData.password}
        onChange={handleChange}
      />
      <button type="submit">Submit</button>
    </form>
  );
};

export default SignupForm;
```

/me

This is a route that will return currently logged in user if one exists. It will do so by checking the `current_user` method defined inside ApplicationController. On the frontend, we will use this when the app initially mounts to determine if user already has access to an authenticated app, or if they need to login/sign up.

The flow:

1. Client: When a user first lands on the app, a fetch request will be made to check if user has already been authenticated.
2. Client: A GET fetch request will be made to `/me` to acquire if a current user exists.
3. Server: The users show action is responsible for rendering a response based on the `current_user` return value.
4. Client: If client is not authenticated, the authentication process will begin by rendering either a login or signup form.

Define an endpoint for this feature:

```
get '/me', to: "users#show"
```

Inside `UsersController` we will check to see if current_user is returning a currently logged in user. This is to ensure that our user does not need to follow authentication again.

```
def show
  if current_user
    render json: current_user, status: :ok
  else
    render json: "No current session stored", status: :unauthorized
  end
end
```

Front end code:

```
import { useState, useEffect } from "react";
import { BrowserRouter as Router } from "react-router-dom";
import LoggedIn from "./LoggedIn";
import LoggedOut from "./LoggedOut";

const App = () => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [currentUser, setCurrentUser] = useState(null);

  useEffect(() => {
    fetch("/me").then((res) => {
      if (res.ok) {
        res.json().then((user) => {
          setCurrentUser(user);
```

```
            setIsAuthenticated(true);
          });
        }
      });
    }, []);

    if (!isAuthenticated) {
      return <div></div>;
    }
    return (
      <div className="app">
        <Router>{false ? <LoggedIn /> : <LoggedOut />}</Router>
      </div>
    );
  };

  export default App;
```

/login

For login and logout, we will use a special controller, `SessionsController`

```
rails g controller sessions create destroy
```

Add the login endpoint to the routes:

```
post "/login", to: "sessions#create"
```

In the `SessionsController`

```
class SessionsController < ApplicationController
  def create
    user = User.find_by(username: params[:username])
    if user&.authenticate(params[:password])
      session[:user_id] = user.id
      render json: user, status: :ok
    else
      render json: "Invalid Credentials", status: :unauthorized
    end
  end
end
```

```
import React, { useState } from "react";

const LoginForm = () => {
  const [formData, setFormData] = useState({
    username: "",
    password: "",
  });

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value,
    });
  };
  const handleSubmit = (event) => {
    event.preventDefault();
    fetch("/login", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(formData),
    }).then((res) => {
      if (res.ok) {
        res.json().then((user) => {
          setCurrentUser(user);
        });
      } else {
        res.json().then((errors) => {
          console.error(errors);
        });
      }
    });
  };

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="username">Username:</label>
      <input
        id="username-input"
        type="text"
        name="username"
        value={formData.username}
        onChange={handleChange}
      />
      <label htmlFor="password">Password:</label>
      <input
        id="password-input"
        type="password"
        name="password"
        value={formData.password}
        onChange={handleChange}
      />
      <button type="submit">Submit</button>
    </form>
  );
};
```

```
export default LoginForm;
```

/logout

Add the logout endpoint to the routes:

```
delete "/logout", to: "sessions#destroy"
```

In the `SessionsController`

```
class SessionsController < ApplicationController
...

  def destroy
    session.delete :user_id
  end

end
```

Front end code:

```
const handleLogout = () => {
  fetch('/logout', {method: "DELETE"})
  .then(res => {
        if (res.ok) {
          setCurrentUser(null)
        }
      })
}
...
<button onClick={handleLogout}>Logout</button>
```