Full Stack Cat App

Knitting Two Projects Together

* This exercise knits together two previous projects, a bcrypt demo to full stack login exercise and a full stack cat app.
* The cat app does not have a way to remember if a user has been to the site before, therefore adding a login gives the app the additional functionality of being able to remember a user.

Make repository

* Go to gitHub and make a new repository. Call it something like cat-login. Copy the http of the new repository to your clipboard.
* On your computer open the finished full stack cat app and open the terminal. Type in git push and then the address of the new repository. Go to gitHub and make sure there is something in the repo.
* On your computer open gitBash or the command line. Type in git clone --mirror and the address of the new repo on gitHub (it should still be on your clipboard). If you are using git bash you might have to left click and paste.
* Go to the root folder (it should be user) and then your user name (you should have a choice of username or public) in the username folder find the created repo. Cut and paste it to the desktop.
* Copy and paste the .env file from the full stack cat app to the new cat login app. Change the name of the database to start out with a clean database. i.e.
MONGODB_URI=mongodb://localhost/backendcrud to
MONGODB_URI=mongodb://localhost/backend-n-stuff
* In terminal type in npm i . The node modules should appear.
* Type in npm run dev. The app should start.

Knit Together the package.json

* Open the package.json on the new cat-login repo.
* In the dependencies object, erase the dependencies. It should look like this:
```
"dependencies": {  },
  "devDependencies": {
    "dotenv": "^8.2.0",
    "nodemon": "^2.0.4"
  }
```
* Open the login repo on your computer and copy and paste the dependencies from the login repo to the cat login repo. This should give you all the dependencies you need to tie the two projects together. Now in the cat-login your dependencies should look like this:
```
"dependencies": {
    "bcryptjs": "^2.4.3",
    "connect-flash": "^0.1.1",
    "ejs": "^3.1.3",
    "express": "^4.17.1",
    "express-ejs-layouts": "^2.5.0",
    "express-session": "^1.17.1",
    "mongoose": "^5.9.22",
    "passport": "^0.4.1",
    "passport-local": "^1.0.0"
  },
  "devDependencies": {
    "dotenv": "^8.2.0",
    "nodemon": "^2.0.4"
  }
```
* Erase the node_modules
* In the terminal run npm i
* Close VSC and re-open the repo, and all your dependencies should be installed
* Run npm run dev. It should still work. Save to gitHub.
%%%%%%%%%%%% Just a Reminder %%%%%%%%%%%%%%%%%%%%%%%%%%%
* In the terminal type in git add . (The period is important).
* Type in git commit -m "knitting together the package.json"
* Type in git push
* If prompted (and you may not be since the repo is cloned) type in password. You will not be able to see what you are typing.

Changing out the server

* In the new cat-login repo, delete the server (it is easier to do this part outside of VSC)
* Copy the server in the login exercise repo and paste it in the cat-login repo.
* Open up the server in VSC and change the app.set('layout', 'layouts/layout'); to app.set('layout', 'layouts/page');
* In the login exercise the primary page was called "layouts" but in the full stack cat app the primary page was called 'page'.
* Close the VSC server and open the whole file with VSC. Open the terminal and type in npm run dev. It should start and then fail. It should not be able to find the passport module.
* Copy and paste the entire passport folder from the login exercise to the new cat-login repo. Close out the server with control C and start again with npm run dev. It should start and then fail because it cannot find the user module.
* Copy and paste the User.js module from the login-exercise models folder into the cat-login models folder. The cat-login models folder should now have CatFancier.js and User.js in it.
* Kill the server and start it again. It should start and fail because it wants a secret and it wants the uri to be a string (and not undefined -- so picky!). To solve the first problem, go into your .env file and enter a secret. Like so: SECRET=catsarehorrible . You can enter any secret you would like. To solve the second problem, in the server at the mongoose.connect include the

process.env.MONGODB_URI (because that is how it is written in the .env). The mongoose.connect should now look like this:

```
mongoose.connect(process.env.DATABASE_URI || process.env.MONGODB_URI, {
   useNewUrlParser: true,
   useUnifiedTopology: true,
   useFindAndModify: false,
   useCreateIndex: true
 },
```

* Your server should now be working!
* Go to localhost:3000 and you should now see the opening page of the full stack cat app. Close the page.
* Close the server and save to gitHub.

Knit together Routes

* Copy and paste the auth folder from the login exercise routes folder into the cat-login routes folder. cat-login should now have two folders, auth and site.
* Open up index.js in the login exercise routes folder. Copy everything except for the module.exports. Past it into the index.js of the cat-login routes folder. Eliminate any redundant entries.
* cat-login routes/index.js should now look like this:

````

const router = require('express').Router();
const updateRoutes = require('./update');
const siteRoutes = require('./site');
const authRoutes = require('./auth');
const CatFancier = require('../models/CatFancier');
const User = require('../models/User');
const {
   ensureAuthenticated
} = require('../passport/auth');

router.use('/update', updateRoutes);
router.use('/restOfSite', siteRoutes);
router.use('/', authRoutes);

router.get('/landing/:id', ensureAuthenticated, (req, res) => {
   User.findById({ _id : req.params.id}, (err, data) => {
      if (err) {
         console.log(err);
      } else {
````

```
            let name = data.name;
            let email = data.email;
            let id = data._id;
            res.render('layouts/landing', {
                name,
                email,
                id
            });
        }
    });
});
router.get('/', (req, res) => {
    res.render('layouts/form');
});

router.post('/catimage', (req, res) => {
    let name = req.body.name;
    let age = req.body.age;
    let catFancier = new CatFancier({ name, age });
    catFancier.save((err, data) => {
        if (err) {
            console.log(err);
        } else {
            let name = data.name;
            let age = data.age;
            let id = data._id;
            res.render('layouts/other-page', {
                name,
                age,
                id
            });
        }
    });
});

router.post('/third-page/:id', (req, res) => {
    let src = req.body.src;
    let id = req.params.id;
    CatFancier.findByIdAndUpdate({ _id : id}, { $set: { favoriteCatImg : src } }, { new: true }, (err,
data) => {
        if (err) {
            console.log(err);
        } else {
            let id = data.id;
            CatFancier.findById({ _id : id }, (err, data) => {
                if (err){
                    console.log(err);
                }else{
                    let name = data.name;
                    let age = data.age;
                    let fci = data.favoriteCatImg;
                    let id = data._id;
                    res.render('layouts/third-page', { name, age, fci, id });
                }
            });
        }
    });

});

router.get('/delete/:id', (req, res) => {
    let id = req.params.id;
    CatFancier.findByIdAndRemove({ _id: id }, (err, data) => {
        if (err) {
            console.log(err);
        } else {
            let name = data.name;
            res.render('layouts/delete.ejs', { name });
        }
    });
});


module.exports = router;

````

* If you run it, the server should run, but if you open up localhost:3000 it should stop because it cannot render the 'login' view.
* Kill the server and save to gitHub.

Add the CSS

* To make the next section easier, copy and past the styles and the fonts from the login exercise into the cat-login. In the body section, remove the margins. Do not override the cat-login styles. Make sure the paths are the same in page.ejs.

Knitting Together the Views

* In cat-login/views/layouts make a new folder called 'login'. Copy and paste all the ejs files EXCEPT layout.ejs from the login exercise/views/layouts into the new views/layouts/login folder.
* In cat-login/views copy and past all the partial ejs files from the login exercise into the cat-login partials file.
* Open cat-login in VSC, go to routes/auth/index.js this is your first route. Change the render from 'layouts/login' to 'layouts/login/login' (to reflect the new path).
* Go to views/layouts/login/login.ejs and change the path of the partial from '../partials/passport-message' to '../../partials/passport-message'
* If you start the server and go to localhost:3000 the login should now render (but not really work). If not, adjust the paths until it renders (but does not really work).
* In cat-login go through the routes, and in the render sections painstakingly re-work all of the ejs paths to reflect these changes then save to gitHub.
* The login function should now work as expected. If the logout button does not work, it's href might need to be changed to '/' instead of '/auth/logout' in views/login/landing.ejs .
* Do not forget about the '/landing/:id' route in views/routes/index.js.
* Test a few times to make sure that all views render. When they do, save to gitHub.

Rendering the Full Stack Cat App Once the User is Logged In

* Now that it is rendering properly, go to cat-login routes/index.js router.get'/landing/:id' and change the view from 'layouts/login/landing' to 'layouts/form' . This view is the first 'page' of the full stack cat app.
* The site should now work, after a fashion. The user can log in and then go to the cat site. The user can update his/her information for the login in the login part of the site and enter his/her name in the cat section. These can be updated as can the cat picture.
* The CSS at this point should be a crime against nature, especially if the login has the original css.

Some Clean Up

* In views layouts/login delete landing.ejs (the cat app is the new landing page).
* In routes/index.js comment out the get'/' route. DO NOT comment out the router.use('/') route. If everything continues to work as expected, delete the route.

Fix CSS

* In style.css change the form and input (and input submit etc.) into classes such as .user-form .user-input .user-input[type=submit]
* Copy over the .user-input[type=submit] (both of them) and change them to [type=button] for the other login/register buttons.

* In views/layouts/login add the classes to all of the elements.
* The atrocities of the login/cat page css should now be remedied. If this is so (please check), save to gitHub.

Where To Go From Here

* The sites are now knitted together, but only superficially.
* We are saving the user name in two places
* None of the site routes, other than the landing routes, are ensure-authenticated.
* Why do we need to check to see if the user info should be updated twice? This is not really a realistic flow. Furthermore, as it stands now, the user has to give his or her name and age everytime he/she visits the site - even though he/she is logged in.
* If we had the user age + the user birthday we could calculate the age in the database (or we could just leave it at the age).
* Do we want two models, so that the user password would stay with the User model? If that is the case, we will have to store the userId in the catFancier model in case we want to delete the User model (or whatever).
* Once the site is integrated and working, it needs to be styled.