React Journalling App

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | App |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

**Components**

CreatePost

EditPost

Posts

Post

**Props**

posts list

post data

**ToDo List**

collection: comments

document: post

res.data ALWAYS returns an array. With an array of one object, you will have to have to extract that object from the array with a map() or [0] and then make reference to it using the dot operator

Mongoose:

findOne() returns an aboject

find() returns an array – DO NOT use if you only want one thing, as it requires more code to pull out the things you want from an object embedded in an array!

Comments section variables

|  |  |  |  |
| --- | --- | --- | --- |
| commentsId | APPEARS IN COMMENTFORM BUT NOT IN PARENT - MISTAKE |  |  |
| commentObj | The object with the single written comment, contains the commentId made with uuid.v4() |  |  |
| articleId | The match param | Post.js | CommentForm:  comentItem: |
| commentObjectId | The object of the comment document | Post.js |  |
| comments | The comments array from the get request | Post.js |  |
| commentData | The comment object from the get req | ReplyForm.js |  |
| commentId | The property on each object of the comments array, to enable us to reply to it | CommentForm.js |  |
| commentsObj | The concstructed object posted to mongo consisting of the comments, articleId and closed prop. | CommentForm.js |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Get all comments | Post.js | get | A request to /comments/:commentId – a document is a comment object linked to a post |
| Post the first comment on an article | Post.js | post | Post to /comments – create a comment object for the first comment on an article |
| Adding a comment to an article | Post.js | put | If the previous comments object grabbed from the get request is zero – PUT to document id |
| Grab the comments again?? | replyForm | get | Get the comments object in order to update the replies |
| Add a reply | replyForm | put | Update replies using commentId, then PUT to document id |

|  |  |
| --- | --- |
| Post.js | |
| commentItem | commentForm |
| replyForm |  |

Do confuse:

**articleID** -used to add comments to the comment object – how could we identify which comment doc belongs to the article if not, as the \_id is random

**fullCommentObject.\_id** – grabbed after making the call to article, and then used to update the:

comments doc with a new comments and new replies

**Todo list 22/7**

saved articles

saved comments

random pages (about, terms and conditions)

user login

nodemailer to create account page

create account page

**Todo List 26/7**

messaging upon post submission + other user actions

make posts display successfully on webpage

l

signup form – put request + validation

complete signup – choosing topics your are interested in + images on list

User session using passport

Notes about PassportJS

strategies are methods of logging in. PassportJS provides about 7 or so, which allow for Oauth, FB, Google etc. The standard email/username – password is defined under “local strategy” by PassportJS.

Method:

you add: passport.use(‘local’, function(req, res)… to the route app.post(‘/login, passport.use(…). When the route is called, the local strategy you defined, is called. The xml-form body is not grabbed using req.body, but implicity passed in (if you want to change the fields from username and password, you have to specify it using “usernamefield: blablaemail”.

The local strategy takes a callback where (username, password, done) are all passed in. You have to use your own password comparison system inside the local strategy. Then importantly, you run an if – else block at the end, which returns either the user if successful, or null if unsuccessful. If null is return, passport automatically returns “unauthorized”. And it is really as easy as that – passport on your login route, that calls the local strategy you defined somewhere else in your program. Lastly you must initialise passport using app.use() → app.use(passport.initialize());

upon initialising, passport grabs the location of your strategy, ready to be accessed when the login route is triggered...

IMORTANT: in order for the local strategy to be triggered when the route is run, you must export the strategy as a function which allows “passport” to be passed in. We then import the file into the file where we have our route, and then pass that file’s passport instance into it., sharing the same instance, when passport.use(‘local’...) is run, “passport” is also passed into the local strategy function as the strategy is thus run, which then returns to the instance the result: null or user.

**Sessions**

https://medium.com/@brendt\_bly/simple-mern-passport-app-tutorial-4aec2105e367

we have to install express-session with passport. Once we pass session into app.use(), every time we make a request to the server, it sends back a session cookie to our browser, that we can view by sending back req.session. Passport works by piggybacking on this sessino variable and adding auth info to it. SavedUnitialized should be set to false, so it doesn’t send back an id to the browser

we can view the sessino cookie with: req.session.id

**Questions**

how does the cookie persist across sessions – how do I keep a user logged in if they navigate to different pages?:

requests using json values?

What sets the cookie initially in passport?

**Notes**

standard checkAuth path is (‘/’) in componentDidMount, for protected pages. What about “ensureAuth” on routes in the backend then?

likes

Create a separate foler for auth components

create utils directory

how to empty the state when form is sent (and default is prevented?)

**Consumers and providers**

It seems that…

Context is a place where global context is stored

providers are tags which are rendered, whereby you can set the state, which will be stored in the context

consumers, are tags where you can gain access to the values inside of the context.

For this reason consumers are sometimes found inside of providers – there are places where we want to both render the values and also update them in the global context

**You need 3 main components:**

1. **Context.js** – create the context, establish the defaults to be exported
2. **App.js** – define the toggle methods and set them onto the buttons. The state is set within app and then passed into the provider. Wrap all your components in the provider (or the ones that cause the context to change, passing in the state. Providers don’t render anything
3. **MyComponent.js** – consumes the context using this.context or a consumer

*this.context → contextType ??*

componentDidMount() {

let value = this.context;

/\* perform a side-effect at mount using the value of MyContext \*/

}

Recently we have covered Legal Matters, Famous Fehler, and grammar topics from Business Grammar No Problem. There has been little consistency of attendance from class to class meaning pages 50-53 in Legal Matters have been repeated various times, and I would have to make a quick change to the lesson plan as the attending student(s) that day had already done the particular topic.

**Recommendations**

continue with Legal Matters from 54 onwards

Do an article from the newsletter.

Page 287 in theBERF (“used to”)

one Famous Fehler – the last one that none of the students have done

Don’t be afraid to repeat any of the topics they have already done

have all your functions that update the state in AppJs, then with a provider in that component, pass in the state into the provider. This will then update the state in the oontext, which can be consumed from any point

To mutate the state with the reducer hook, begin by creating a component that returns a provider. This provider should take both children {props.children} and another context as props

Main functions handling context manipulation should be at the top level of the

**The state can or can’t be changed inside the Provider?**

App is executed every time the page reloads, as it is at the top of the application – this is useful to set the context each time

The state *can* be changed in the provider using a function that is passed through to the consumer.

If you upddate the provider, it will affect only those components below it in the hierarchy.

What does “below it in the hierarchy” mean, in this context?

You cannot pass vaues into the provider and they will be updated in the consumer .. right?

**General process**

* create a controller class where you implement the provider with children in the middle
* wrap <App> in the controller class so it is at the top of the hierarchy
* pass your functions throughout your program

for this model – important: your imports MUST be wrapped inside of curly brackets – WHY? Just like Browser router etc. this is because you are grabbing properties of the main object. When you grab the provider class, you are actually grabbing what it renders, ??

basically the provider gets declared once in your program – it is the place where the values are fed down through the tree into the children. If you want to modify a value, you can pass a function reference down with the value props, that, when fired, modifies the values in the provider. However, what is not possible, is to redeclare the provider further down the tree, pass in new values from the local state, and have it update throughout all the cosumers in the rest of the program throughout the tree. This is because in React data Is designed to flow in only 1 direction- top to bottom.

**Context within mounting components**

so how can we pass a function down from the provider into a component. That fires a function when that component mounts – e.g. every time the page is refreshed?

**2/8 To do**

decorate congratulations sign up page

change image on complete signup page

login modal forgotpassword

forgot password forgot link decorate

recovery email page separate out rows – spinner in container

reset password – psinner in middle

spinners

**4/8**

link articles to authors

create an author page

profile page - edit

portect routes

**5/8**

sidebar links

**6/8**

create 2 users with 1 article each

author: username fixed

subscriptions

saved articles (db push)

read articles history

user articles

post preview page

“average read” based on article length

Profile avatar in comments and subscriptions list

comments avatars and names linked to profile pages

likes button – only 1 like per page then push to db

**HOW TO MAKE PROTECTED ROUTE WORK properly??**

how to make props work...

**9/8**

image for users all around site

saved articles – toggle array route, not articleContext

need to be logged in warning modal

**10/8**

comments

YT sidebar

**Comments**

**loop through comments and pass into commentItem**

**pass each of the replies into ReplyItem**

**set up a replyForm next to each replyItem**

the replyform should pass back a: commentId to know which mongo object to target. It can then append the comment onto the end of the replies array in that object

it also needs the username of the reply

**Remaining: Sunday**

edit post

comment numbers

article created

user created

personalized feed

user thumbnail link to profile

saved articles

front page when not logged in

other pages niceify – topics, popular,

**This weekend**

popular page

protected routes

clean up code

spinners

more articles

error 404

error login

css file

handle errors

**Homepage**

featuredThumbnail

centerThumbnail

popularHeaderPost

TopicHeaderPost

topicThumbnail

popularThumbnail

personalizedThumbnail

following thumbnail

popularFeedItem

personalizedFeedItem

authorAvatar

then…

saved posts bookmark on:

personalized

topic

peronlized homepage

redirect on own profile

fix createPost

**Monday**

~~history, saved – spinner~~

~~fix deletehistory function~~

~~mobile reponsive homepage~~

~~protect typed in routes~~

~~change title – journalling NOT blogging~~

tidy up backend

add pages

push to github

**Final Stretch...**

push both to heroku

tidy up website

fix quiz app

Unit tests on both

put onto website & CV

apply!