Billing System

Monday, April 26, 2021 9:49 PM

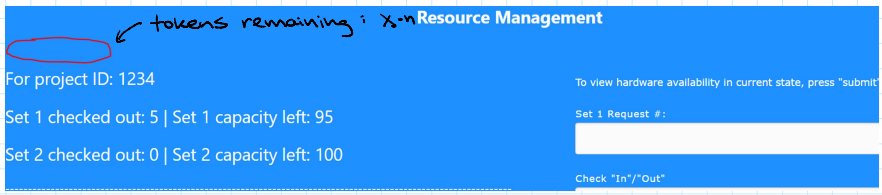
1) Each user given an initial X tokens

app.py

```python
@app.route("/api/register", methods=["POST"])
def register():
    try:
        # pull form submission data.
        # encrypt sensitive information like email and password
        email = request.json["email"].lower()
        email_hashed = bcrypt.hashpw(str.encode(email), salt).decode()
        password = request.json["password"]
        confirm_password = request.json["confirm_password"]
        name = request.json["name"]

        # confirm the password the user chooses is strong
        if len(policy.test(password)) > 0:
            return jsonify({"error": "Password is not strong enough. Minimums: 8 characters, 1 uppercase, 1 number, 1 special"})

        # make sure the user confirms the password properly
        if password != confirm_password:
            return jsonify({"error": "Passwords do not match"})

        # check to see if user already exists; else, communicate with user that an
        # account already exists
        if Account_Info.find_one({"email": email_hashed}) is not None:
            return jsonify({"error": "User with that email already exists"})
        else:
            Account_Info.insert_one({"name":name, "email":email_hashed, "password":pbkdf2_sha256.hash(password)})

        return jsonify({"success": True})
```

add "tokens": X to the dictionary inserted to the user DB

2) Add a view of tokens on checkin/checkout page



a) app.py

def hardware():

⋮

```python
    if check1 == "out":
        used1 = result.get("used1") + int(set1val)
        cap1 = result.get("cap1") - int(set1val)

    if check1 == "in":
        used1 = result.get("used1") - int(set1val)
        cap1 = result.get("cap1") + int(set1val)

    if check2 == "out":
        used2 = result.get("used2") + int(set2val)
        cap2 = result.get("cap2") - int(set2val)

    if check2 == "in":
        used2 = result.get("used2") - int(set2val)
        cap2 = result.get("cap2") + int(set2val)

    if cap1 > 100 or cap1 < 0:
        return jsonify({"error": "Set 1 value invalid"})
    if cap2 > 100 or cap2 < 0:
        return jsonify({"error": "Set 2 value invalid"})

    Hardware_Info.update_one({"projectid": id}, {"$set": { "used1": int(used1), "used2": int(used2), "cap1": int(cap1), "cap2": int(cap2)}})
    result = Hardware_Info.find_one({"projectid": id})
    del result['_id']
    return result
```

user = Account_Info.find_one({"email": email of current user})

tokens = user["tokens"]

add a key "tokens" <-> value tokens

b) Hardware.jsx

add <h1 style={divStyle}> tokens left: {this.state.hw.tokens} </h1>

```jsx
            </div>
            <h1 style={divStyle}>For project ID: {this.state.hw.projectid}</h1>

            <h1 style={divStyle}>Set 1 checked out: {this.state.hw.used1} | Set 1 capacity left: {this.state.hw.cap1}</h1>

            <h1 style={divStyle}>Set 2 checked out: {this.state.hw.used2} | Set 2 capacity left: {this.state.hw.cap2}</h1>
```

3) Initialize Hardware_Info DB to hold constants for costs

app.py

```python
@app.route("/api/newproject", methods=["POST"])
def newproject():
    try:
        # pull form submission data.
        # encrypt sensitive information like project id
        projectID = request.json["projectid"]
        projpassword = request.json["password"]
        projectname = request.json["projName"]
        desc = request.json["description"]
        cap = 100
        used = 0
        # confirm the password the user chooses is strong
        #if len(policy.test(projpassword)) > 0:
        #   return jsonify({"error": "Password is not strong enough. Minimums: 8 characters, 1 uppercase, 1 number, 1 special"})

        # check to see if project already exists; else, communicate with user that this already exists
        if Project_Info.find_one({"projectid": projectID}) is not None:
            return jsonify({"error": "Project with that ID already exists"})
        else:
            Project_Info.insert_one({"projName":projectname, "projectid":projectID,"password":pbkdf2_sha256.hash(projpassword), "description":desc})
            Hardware_Info.insert_one({"projectid":projectID, "cap1": cap, "cap2": cap, "used1": used, "used2":used})

        return jsonify({"success": True})
    except:
        # there was an error while processing form submission
        return jsonify({"error": "Invalid form"})
```

cost1 = n cost2 = m

"cost1": cost1, "cost2": cost2

4) Decrement tokens upon checkout

app.py

```python
@app.route("/api/hardware", methods=["POST"])
def hardware():
    try:
        #Get the set info from the request
        set1val = request.json["set1"]
        set2val = request.json['set2']
        check1 = request.json["check1"].lower()
        check2 = request.json["check2"].lower()
        id = request.json["id"]

        if Hardware_Info.find_one({"projectid": id}) is not None:
            result = Hardware_Info.find_one({"projectid": id})
            used1 = result.get("used1")
            cap1 = result.get("cap1")
            used2 = result.get("used2")
            cap2 = result.get("cap2")
```

cost1 = result.get("cost1")
cost2 = result.get("cost2")

Check if tokens are sufficient. Perform error JSON message if not.

user Account_Info.find_one({'email': email of current user's
tokens = user["tokens"]

```python
        if check1 == "out":
            used1 = result.get("used1") + int(set1val)
            cap1 = result.get("cap1") - int(set1val)
        if check1 == "in":
            used1 = result.get("used1") - int(set1val)
            cap1 = result.get("cap1") + int(set1val)

        if check2 == "out":
            used2 = result.get("used2") + int(set2val)
            cap2 = result.get("cap2") - int(set2val)
        if check2 == "in":
            used2 = result.get("used2") - int(set2val)
            cap2 = result.get("cap2") + int(set2val)

        if cap1 > 100 or cap1 < 0:
            return jsonify({"error": "Set 1 value invalid"})
        if cap2 > 100 or cap2 < 0:
            return jsonify({"error": "Set 2 value invalid"})
        Hardware_Info.update_one({"projectid": id}, {"$set": { "used1": int(used1), "used2": int(used2), "cap1": int(cap1), "cap2": int(cap2)}})
        result = Hardware_Info.find_one({"projectid": id})
        del result['_id']
        return result
```

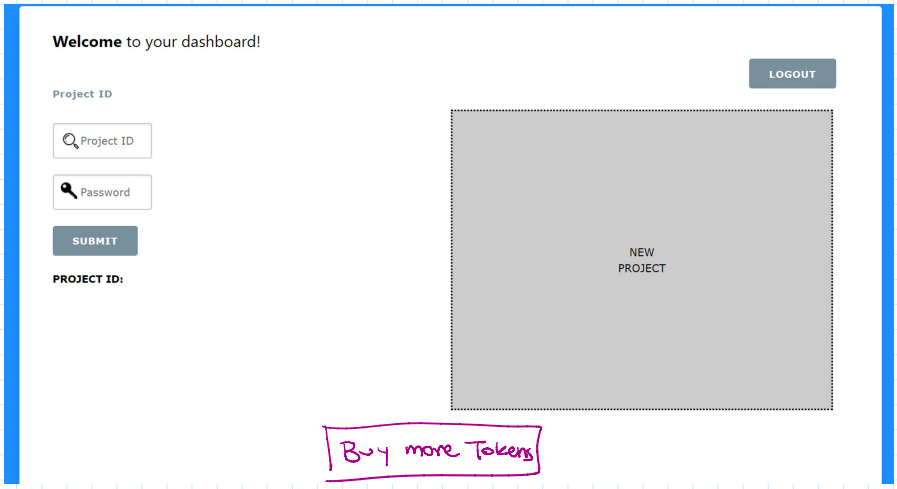
update user token here given cost1

update user token here given cost2

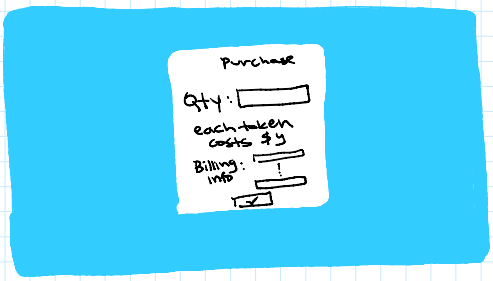
Sends updated user to Account_Info DB w/ update_one()

add a key "tokens" <-> value tokens

5) Buy more Tokens



9) Create a new jsx component for a page to host a purchase form. Call it Purchase.jsx



Register it in App.jsx here as a Protected Route

```jsx
function App() {
    return (
        <React.Fragment>
            {/* route the components to a link that can be referenced with href */}
            <Router>
                <Switch>
                    <Route path="/" exact component={Home} />
                    <Route path="/login" exact component={Login} />
                    <Route path="/register" exact component={Register} />
                    <Route path="/updateuserpassword" exact component={UpdateUserPassword} />
                    <ProtectedRoute path="/hardware" exact component={Hardware}/>
                    <ProtectedRoute path="/newproject" exact component={NewProject}/>
```

b) Create Buy button on Dashboard.jsx

```
{/* Routes to new project or back home after logging out */}
<button onClick={() => this.nextPath('/newproject')} class = "bigbtn" variant = "outline-primary">NEW PROJECT</button>
<button onClick={() => this.nextPath('/')} class = "btnLogout" variant = "outline-primary">Logout</button>
</div>
```

<b. --- ={()=> this.nextPath('/purchase')} -- ---> Buy more tokens </b.-->

c) create new route '/api/purchase' in app.py

```
@app.route("/api/deleteproject", methods=["POST"])
def deleteproject():
```

@app.route("/api/purchase", methods = ["POST"])
def purchase():

// from onlinepayment import OnlinePayment https://pypi.org/project/onlinepayment/

// connect to Paypal

// charge

ex code :

```
# or for paypal, setup auth with user, pass, vendor and product:
auth= { 'username': 'YOUR USERNAME HERE',
        'password': 'YOUR PASSWORD HERE',
        'vendor':   'YOUR VENDOR HERE',
        'product':  'YOUR PRODUCT HERE' }

# connect to PayPal
op = OnlinePayment('paypal', test_mode=True, auth=auth)

# charge a card
try:
    result = op.sale(first_name = 'Joe',
                     last_name  = 'Example',
                     address    = '100 Example Ln.',
                     city       = 'Exampleville',
                     state      = 'NY',
                     zip        = '10001',
                     amount     = '2.00',
                     card_num   = '4007000000027',
                     exp_date   = '0530',
                     card_code  = '1234')

except conn.TransactionDeclined:
   # do something when the transaction fails

except conn.CardExpired:
   # tell the user their card is expired
```

d) In Purchase.jsx, add onClick handler for the button to purchase

```
buy = (e)=>{
  e.preventDefault();

  axios.post("http://.../api/purchase"),{
      // send qty + billing info
      // send logged in user info
            so backend can
            query the DB
  }).
    then((res) =>{

        // error handling if needed

        // success/failure message

  });};
```