Duke Shares Lunch
316 Project Milestone 2
Paul Dellinger, AJ Eckmann, and Josh Romine
https://github.com/pauldellinger/duke-shares-lunch

Section 1. Changes/updates to your original proposal (if any):

We are now using PostgREST instead of creating our own API with PHP/PDO because it is faster and has more functionality. PostgREST has built in resource embedding in the query. This allows us to get info from multiple related tables all in one request. For instance - when querying the Purchase table with GET /purchase we get this JSON in return:


```
[
   {
       "pid": 3001,
       "saleid": 2001,
       "bid": 2,
       "price": 8.95,
       "approve": false,
       "paid": false,
       "p_description": "Hey paul can you get me rigatoni with pomodoro, spinach and extra
chiken"
   }
]
```


This is nice, but we'd like to have info on the buyer to display, without putting extra resources into another whole request on the RegisteredUser table. We also would like to get info on the seller, which would require a query on the ActiveSeller table with saleid and *then* RegisteredUser. We could write functions for the endpoints, which do all the queries, but PostgREST has this functionality built in:
GET /purchase?select=buyer:**registereduser(uid,name,venmo)**,
seller:**activeseller(registereduser(uid,name,venmo))**,pid,
Price,saleid,approve,paid,p_description

Which returns this JSON:


```
[
   {
       "pid": 3001,
       "price": 8.95,
       "saleid": 2001,
       "approve": false,
       "paid": false,
       "p_description": "Hey paul can you get me rigatoni with pomodoro, spinach and extra
chiken",
```

```
      "buyer": {
          "uid": 2,
          "name": "Josh Romine",
          "venmo": "joshielikescash"
      },
      "seller": {
          "registereduser": {
              "uid": 1,
              "name": "Paul Dellinger",
              "venmo": "paul_dellinger"
          }
      }
   }
]
```


So in one endpoint, we've gotten info from the Purchase, ActiveSeller, and RegisteredUser tables.

In addition, we are now using Nginx to serve the web service because it was recommended by the Postgrest documentation.

Section 2. Summary of Progress so far:

The API service is now being hosted with Nginx. We've been accessing it by pointing to the IP address of the course VM with the specified endpoint.

Modified the schema slightly:

- Changed keys from integer to serial type - this way they are assigned automatically at insert
- Several different roles, with different access to tables

Since the first milestone reports we have created our REST API with Postgrest. Additionally we have created many of the endpoints including endpoints that:

- add/edit/delete user info (including an authentication process for users)
- add/edit/delete entries in the sellpreferences endpoint
- add/edit/delete entries in the activesellers endpoint
- built in functionality to modify 'paid' columns in purchase endpoint - approval of price and allowing the seller to mark the transaction as paid

We've gotten started on authentication - interacting with the API requires a JWT token in the header.

In addition, we've begun to connect our data with iOS. On Xcode, we implemented a TableView and created functions to retrieve JSON data from a URL.

Section 3. A list of tasks to be completed before the final due date

Complete the following endpoints in the REST API:

- Complete Purchase (Trigger) - delete purchase row, and add optionality for seller to remain active or terminate their current status as an active seller
- Adding a description column into ActiveSeller where sellers can say where they are - to help them connect with the buyer in person
- Add password into RegisteredUser schema
- Figure out exactly which roles should have what type of access to particular tables
- Create an authentication protocol to create new users from the app

Swift Tasks

- Build function to parse JSON data and populate the table for ActiveSellers
- Write code to allow for buy requests, approval, and purchase confirmation
- Design meal selection once seller is chosen
- Design final interface/aesthetic