```python
from app import models, db
from sqlalchemy import *
from datetime import datetime

def generate_random_postings():
    result = models.Posting.query.join(models.User).with_entities(
        models.Posting.postid, models.Posting.userid, models.User.username,
        models.Posting.title, models.Posting.price, models.Posting.description,
        models.User.rating)
    return result

def add_new_post(form_input, current_user):
    db.session.add(models.Posting(
        userid = current_user.userid,
        date = datetime.now(),
        title = form_input['title'],
        description = form_input['description'],
        price = form_input['price'],
        category = form_input['category'],
        contactmethod = form_input['contactmethod'],
        tags = form_input['tags']))
    db.session.commit()
    return True

def add_user(new_user_info):
    try:
        newUser = models.User(
            username        = new_user_info['username'],
            email           = new_user_info['email'],
            password        = new_user_info['password'],
            phonenumber     = new_user_info['phone'],
            personalemail   = new_user_info['personalemail'],
            bio             = new_user_info['bio'],
            rating          = float(new_user_info['rating']),
            numRatings      = int(new_user_info['numRatings']),
        )
        db.session.add(newUser)
        db.session.commit()
        db.session.refresh(newUser)
        if newUser.userid is None:
            return False
        return True
    except Exception as e:
        return False
    return False

def get_post_id(postid):
    try:
        if postid is None and not int(postid) > 0:
            return None
    except Exception as e:
        return None
    current_post_info = models.Posting.query.filter_by(postid=postid).join(
        models.User).with_entities(
        models.Posting.postid, models.Posting.title, models.Posting.category,
        models.Posting.price, models.Posting.description,
        models.Posting.contactmethod, models.Posting.tags,
        models.User.phonenumber, models.User.email, models.User.userid,
        models.User.username, models.User.rating, models.User.personalemail
    ).first()
    return current_post_info

def modify_post_by_id(results, postid):
    models.Posting.query.filter_by(postid=postid).update(dict(results))
    db.session.commit()

def update_current_user(current_user, result):
```

```
68      if 'password' in result:
69          current_user.password = result['password']
70      current_user.phonenumber = result['phonenumber']
71      current_user.personalemail = result['personalemail']
72      current_user.bio = result['bio']
73      db.session.commit()
74      return True
75
76  def get_posting_by_id(postid):
77      posting_info = models.Posting.query.filter_by(postid=postid).first()
78      if posting_info is None:
79          return None, None
80      if posting_info.contactmethod == 'personalemail':
81          poster_info =
            models.User.query.filter_by(userid=posting_info.userid).with_entities(
82              models.User.personalemail, models.User.rating, models.User.userid,
                models.User.username
83          ).first()
84      elif posting_info.contactmethod == "phonenumber":
85          poster_info =
            models.User.query.filter_by(userid=posting_info.userid).with_entities(
86              models.User.phonenumber, models.User.rating, models.User.userid,
                models.User.username
87          ).first()
88      else:
89          poster_info =
            models.User.query.filter_by(userid=posting_info.userid).with_entities(
90              models.User.email, models.User.rating, models.User.userid,
                models.User.username
91          ).first()
92      return posting_info, poster_info
93
94  def get_user_by_email(email):
95      return models.User.query.filter_by(email=email).first()
96
97  def add_claim(form, postid, user):
98      try:
99          post_info, poster_info = get_posting_by_id(postid)
100         if user.userid == post_info.userid and not form['buyeremail'] == False:
101             buyer_info = get_user_by_email(form['buyeremail'])
102             newClaim = models.Claim(
103                 postid          = postid,
104                 sellerid        = user.userid,
105                 buyerid         = buyer_info.userid,
106                 usersubmitted   = user.userid,
107                 date            = datetime.now(),
108                 Rating          = form['rating']
109             )
110             db.session.add(newClaim)
111             return True, newClaim
112         elif not user.userid == post_info.userid:
113             newClaim = models.Claim(
114                 postid          = postid,
115                 sellerid        = post_info.userid,
116                 buyerid         = user.userid,
117                 usersubmitted   = user.userid,
118                 date            = datetime.now(),
119                 Rating          = form['rating']
120             )
121             db.session.add(newClaim)
122             return True, newClaim
123     except Exception as e:
124         db.session.rollback()
125         return False, False
126     return False, False
127
128 def check_for_transaction(claim):
```

```
        try:
            other_claim = models.Claim.query.filter(
                models.Claim.postid==claim.postid,
                models.Claim.sellerid==claim.sellerid,
                models.Claim.buyerid==claim.buyerid,
                models.Claim.usersubmitted!=claim.usersubmitted
            ).first()
            if other_claim is not None:
                newTransaction = models.Transaction(
                    date            = datetime.now(),
                    claimidseller   = claim.sellerid,
                    claimidbuyer    = claim.buyerid
                )
                db.session.add(newTransaction)
                print("Try to Alter Ratings!")
                alter_ratings(claim, other_claim)
                print("Try to Archive!")
                if archive_posting(claim.postid, newTransaction):
                    print("Archived! Now Delete The Claims:")
                    delete_claim(claim.claimid)
                    delete_claim(other_claim.claimid)
                    print("Finished")
                    return True
                else:
                    raise ValueError
        except Exception as e:
            print("Rollback in check_for_transaction")
            db.session.rollback()
            return None
        return False

    def delete_claim(claimid):
        try:
            models.Claim.query.filter_by(claimid=claimid).delete()
        except Exception as e:
            pass

    def delete_user(userid):
        try:
            someUser = User.query.filter_by(userid=someUserID).first()
            db.session.delete(someUser)
            db.session.commit()
        except Exception as e:
            pass


    def archive_posting(postid, transaction=None):
        try:
            if postid is not None:
                post = models.Posting.query.filter_by(postid=postid).first()
                print("got Post")
                archivedPost = models.ArchivedPosting(
                    transactionid   = transaction.transactionid,
                    postid      = post.postid,
                    buyerid         = transaction.claimidbuyer,
                    sellerid        = transaction.claimidseller,
                    date      = datetime.now(),
                    title       = post.title,
                    description   = post.description,
                    price       = post.price,
                    category    = post.category,
                    contactmethod   = post.contactmethod,
                    tags      = post.tags
                )
            else:
                archivedPost = models.ArchivedPosting(
                    postid      = post.postid,
```

```
196                 date      = datetime.now(),
197                 title       = post.title,
198                 description   = post.description,
199                 price       = post.price,
200                 category    = post.category,
201                 contactmethod   = post.contactmethod,
202                 tags      = post.tags
203             )
204 
205         db.session.add(archivedPost)
206         db.session.delete(post)
207         return True
208     except Exception as e:
209         db.session.rollback()
210     return False
211 
212 def get_new_rating(current_rating, current_number, new_rating):
213     current_number += 1
214     current_rating = current_rating * (current_number-1)/current_number + new_rating *
    1/current_number
215     return [current_rating, current_number]
216 
217 def alter_ratings(claim, other_claim):
218     print("Getting Ratings")
219     user1 = models.User.query.filter_by(userid=claim.usersubmitted).first()
220     user2 = models.User.query.filter_by(userid=other_claim.usersubmitted).first()
221     print("Update the Ratings Manually")
222     [user1.rating, user1.numRatings] = get_new_rating(user1.rating, user1.numRatings,
    other_claim.Rating)
223     [user2.rating, user2.numRatings] = get_new_rating(user2.rating, user2.numRatings,
    claim.Rating)
224     print("FINISHED alter ratings")
225 
226 def get_postings(userid):
227     try:
228         postings = models.Posting.query.filter_by(userid=userid).all()
229         print("Got postings for user")
230         return postings
231     except Exception as e:
232         pass
233     return None
234 
235 def get_claims(userid):
236     try:
237         claims =
        models.Claim.query.filter_by(usersubmitted=userid).join(models.Posting).with_enti
        ties(
238             models.Claim.date, models.Posting.title, models.Posting.postid
239         ).all()
240         print("Got Claims")
241         return claims
242     except Exception as e:
243         pass
244     return None
245 
246 def get_sales(userid):
247     try:
248         sales = models.ArchivedPosting.query.filter_by(sellerid=userid).with_entities(
249             models.ArchivedPosting.title, models.ArchivedPosting.buyerid,
            models.ArchivedPosting.price,
250             ).join(
251             models.Transaction).join(models.User, models.User.userid ==
            models.ArchivedPosting.buyerid).with_entities(
252                     models.Transaction.date, models.ArchivedPosting.title,
                    models.ArchivedPosting.price,
253                     models.User.username, models.User.userid).all()
254         print("Got sales")
```

```
255             return sales
256         except Exception as e:
257             pass
258         return None
259 
260     def get_purchases(userid):
261         try:
262             purchases = models.ArchivedPosting.query.filter_by(buyerid=userid).with_entities(
263                 models.ArchivedPosting.title, models.ArchivedPosting.sellerid,
                    models.ArchivedPosting.price,
264                 ).join(
265                 models.Transaction).join(models.User, models.User.userid ==
                    models.ArchivedPosting.sellerid).with_entities(
266                         models.Transaction.date, models.ArchivedPosting.title,
                            models.ArchivedPosting.price,
267                         models.User.username, models.User.userid).all()
268             print("Got purchases")
269             return purchases
270         except Exception as e:
271             pass
272         return None
273 
```