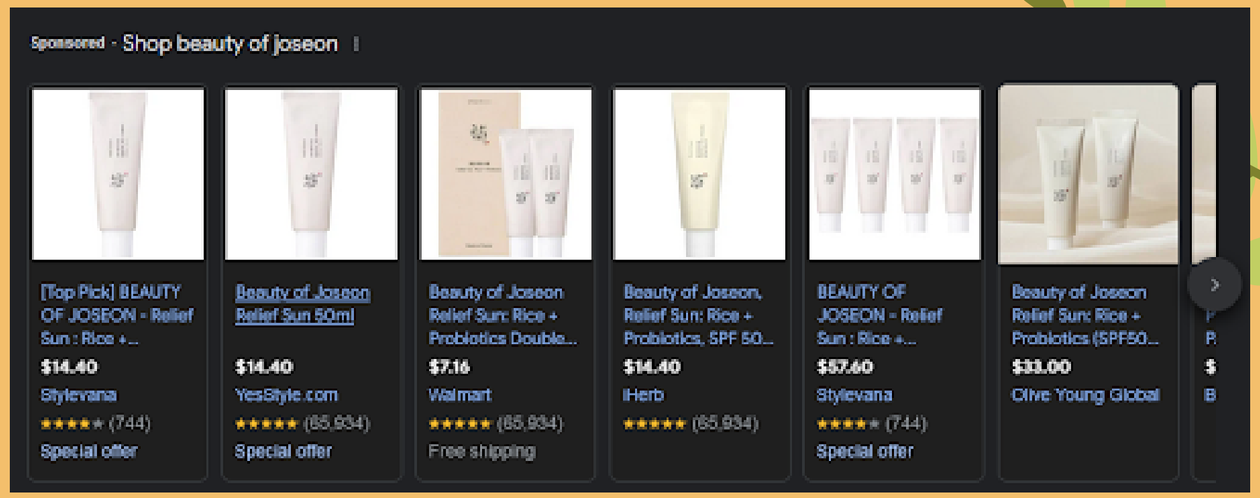
BeautyHive

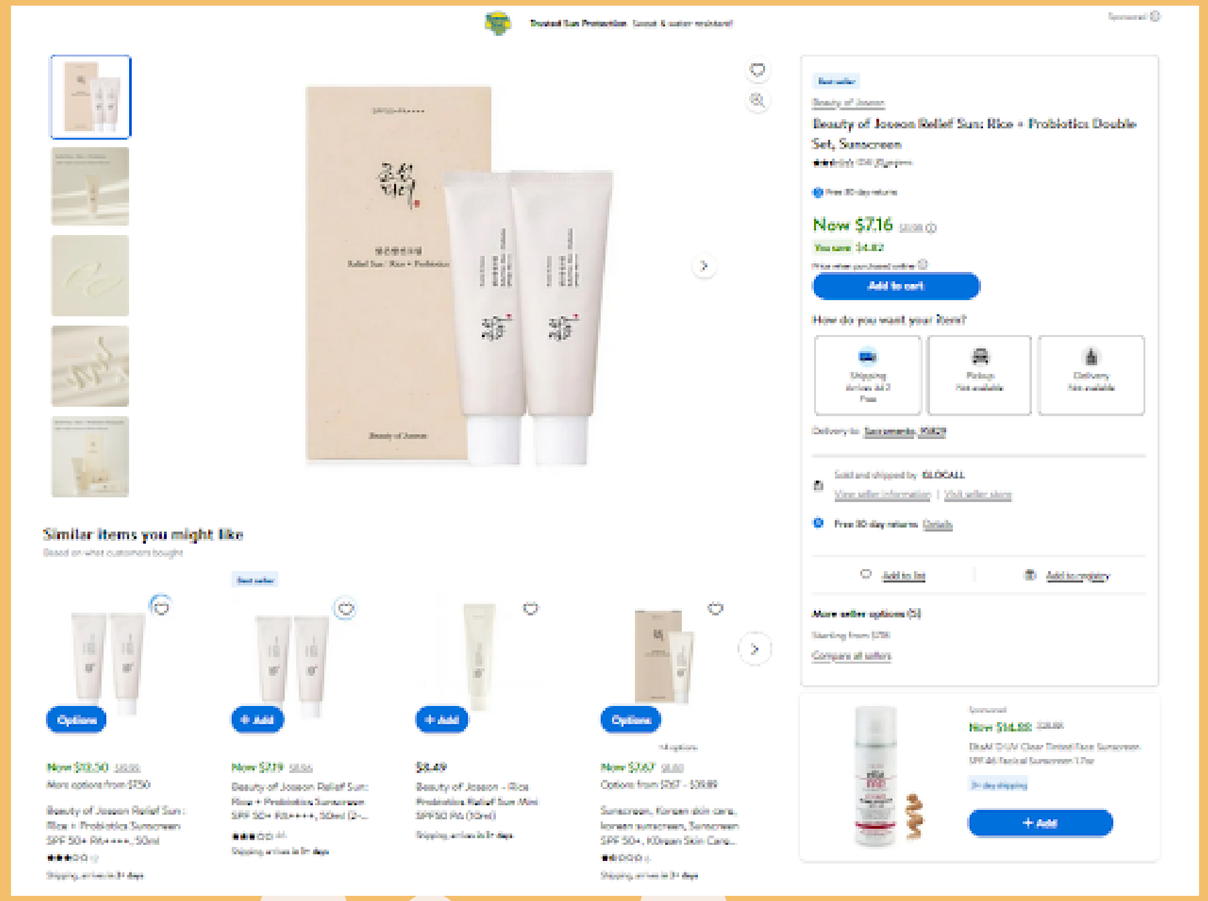
PROJECT DELIVERABLE #3

By Julia Cardillo // CS673

The Issue

- Finding credible reviews for makeup and skincare products can be difficult.
- Reviews are either misleading or are for fake products sold by 3rd party entities.





My Solution

- Introducing... BeautyHive!
- Hub for all beauty product reviews.
- Mobile application that centralizes and aggregates reviews, made for beauty enthusiasts by beauty enthusiasts.

Deliverable #1 - Design and Data Requirements

DATA GENERATION

Data is generated when users register for the platform and create product reviews.

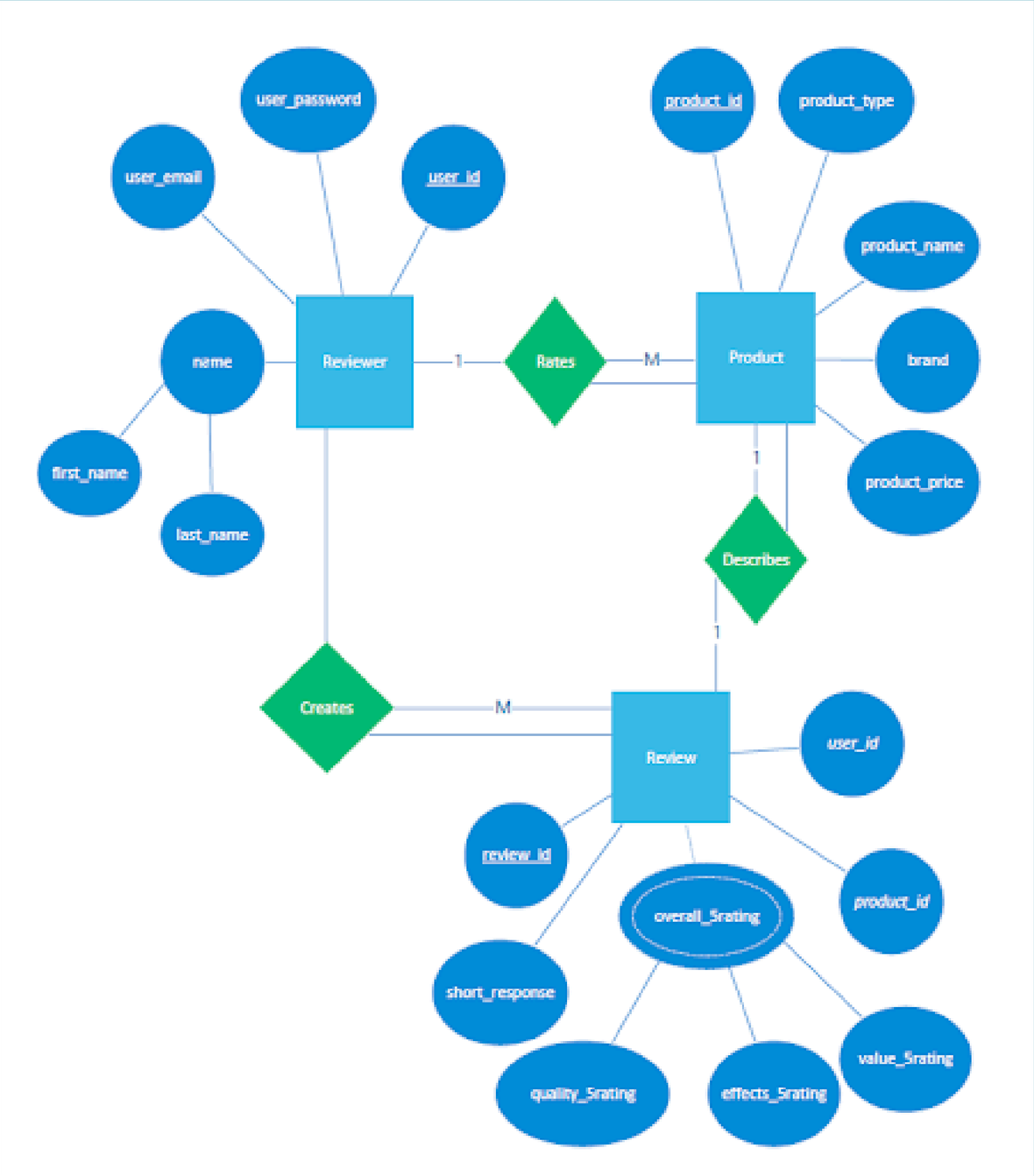
DATA TYPES

- Quantitative (product ratings)
- Qualitative (short responses)
- Categorical data (product type)

RELATIONAL DATABASE

- Data is structured
- Data is regularly queried
- Data can be easily manipulated
- Has constraints
- ACID properties

ER Diagram and Relational Schema



Relational Schema

Reviewer (user_id, user_password, user_email, first_name, last_name)

Review (review_id, *user_id*, *product_id*, overall_5rating, quality_5rating, effects_5rating, value_5rating)

Product (product_id, product_type, product_name, brand, product_price)

Deliverable #2 - Definitions of Terms Used

USER

An individual who registers for this platform, with the purpose of either viewing beauty product reviews or creating and posting their product reviews.

PRODUCT

A beauty (makeup or skincare) product that is available for sale online and in-person via various retailers. These products are reviewed on BeautyHive.

REVIEW

A User's comprehensive rating of a product, including numerical and short responses. A user can only create one review per product.

Changes Made to Deliverable #1

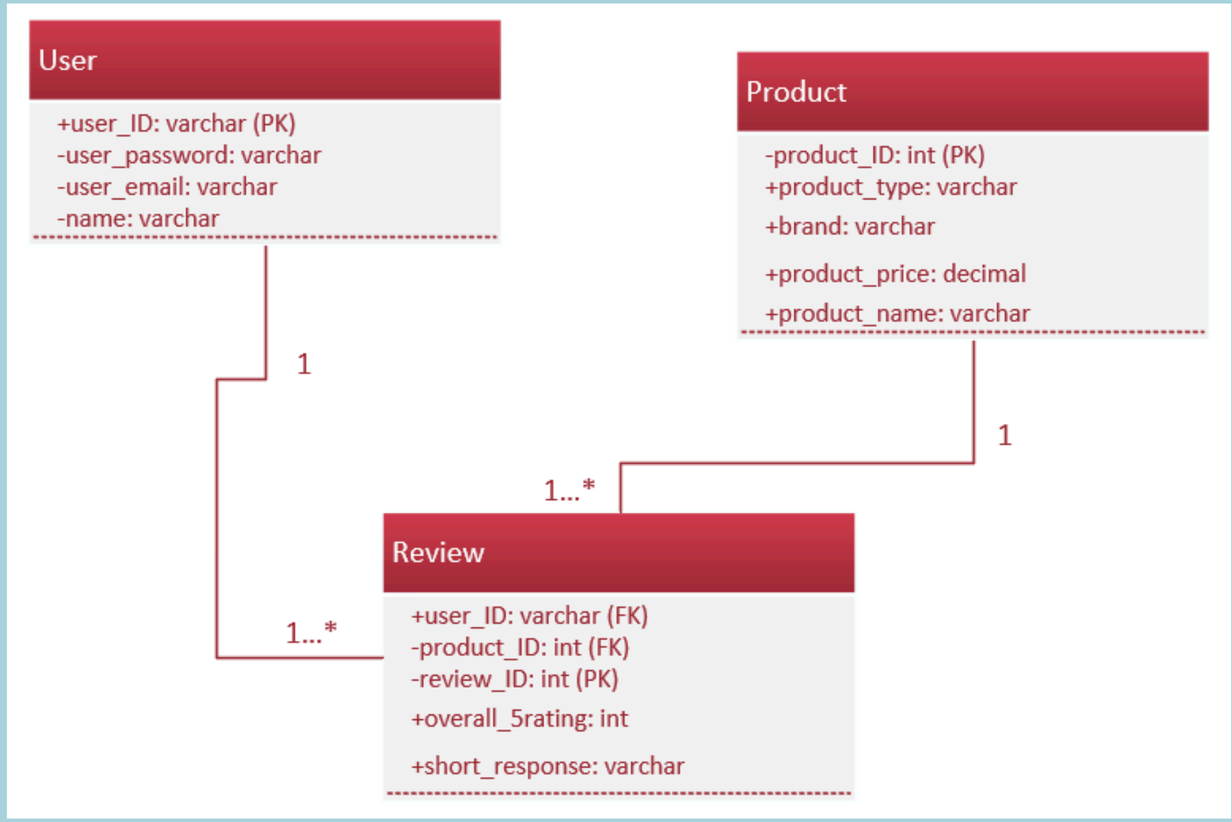
Product and Review cardinality made to be one-to-many, removed relationship from user to product

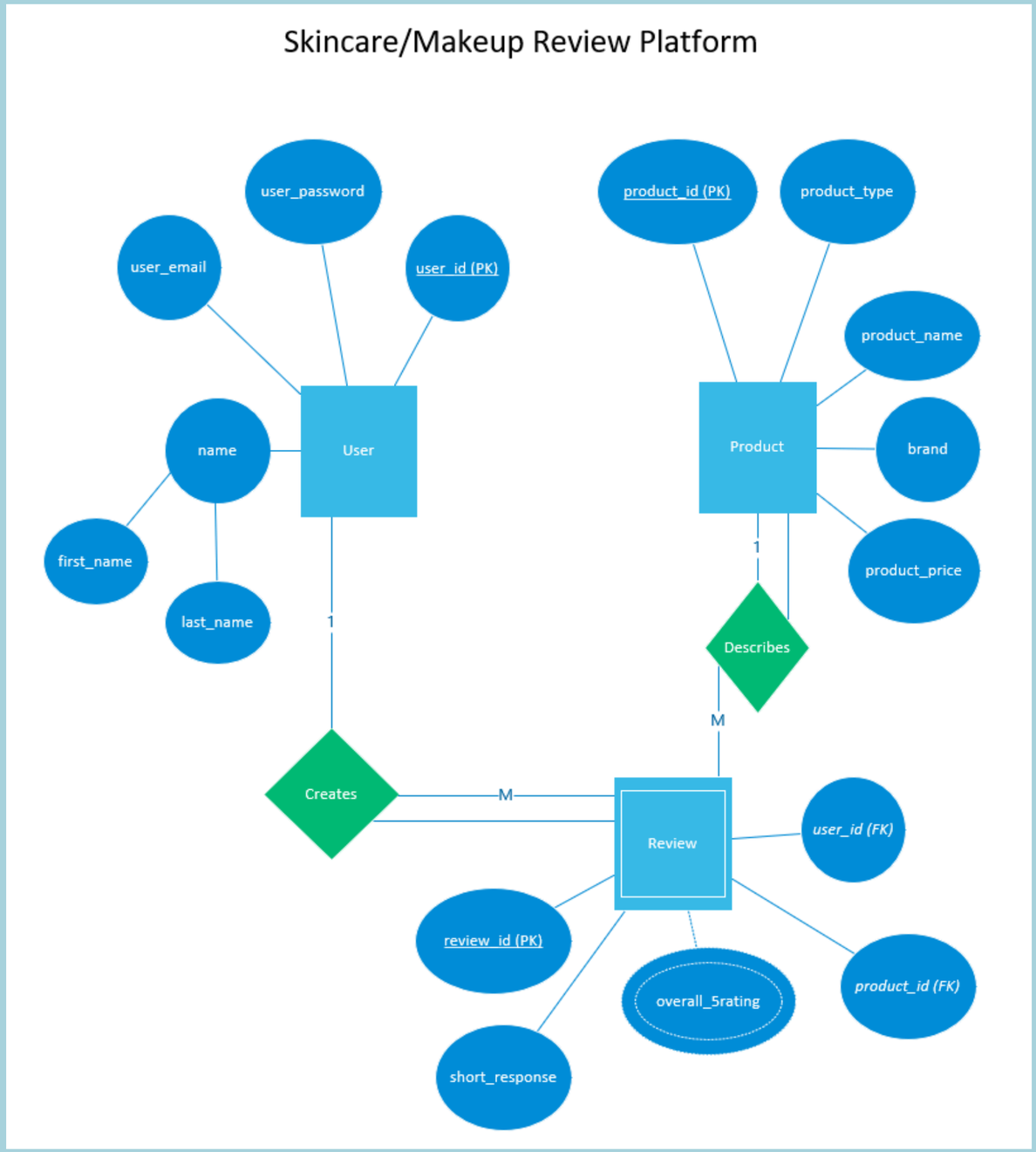
Product and User are strong entities, while Review is a weak entity

Changed Reviewer table to User for clarity

Clarified Primary and Foreign Keys

Updated ER Diagram and UML Diagram





Technical Implementation

1. Used MySql and Python
2. Built a script in Sublime text editor, to build the connection between MySql and Python
3. Replicated all tables and data
4. Demonstrated CRUD operations

C:\Users\julia\Documents\ConnectionScript.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

ConnectionScript.py | test.py

```python
mycursor.execute("INSERT INTO Product (product_type, product_name, brand, product_price) VALUES (%s, %s, %s, %s)",('Makeup', 'Precisely, My Brow Wax', 'Benefit Cosmetics', 27.00))
mycursor.execute("INSERT INTO Product (product_type, product_name, brand, product_price) VALUES (%s, %s, %s, %s)",('Makeup', 'Major Glow Balm', 'Patrick Ta Beauty', 50.00))
mycursor.execute("INSERT INTO Product (product_type, product_name, brand, product_price) VALUES (%s, %s, %s, %s)",('Skincare', 'Vanilla Lip Care Duo', 'Laneige', 34.00))
mycursor.execute("INSERT INTO Product (product_type, product_name, brand, product_price) VALUES (%s, %s, %s, %s)",('Skincare', 'The Silk Serum', 'Tatcha', 98.00))
db.commit()

mycursor.execute("SELECT * FROM Product")
print('Here are all the products:\n')
for x in mycursor:
    print(x)
```

```
Here are all the products:

(1, 'Makeup', 'Precisely, My Brow Wax', 'Benefit Cosmetics', Decimal('27.00'))
(2, 'Makeup', 'Major Glow Balm', 'Patrick Ta Beauty', Decimal('50.00'))
(3, 'Skincare', 'Vanilla Lip Care Duo', 'Laneige', Decimal('34.00'))
(4, 'Skincare', 'The Silk Serum', 'Tatcha', Decimal('98.00'))
[Finished in 784ms]
```

C:\Users\julia\Documents\ConnectionScript.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

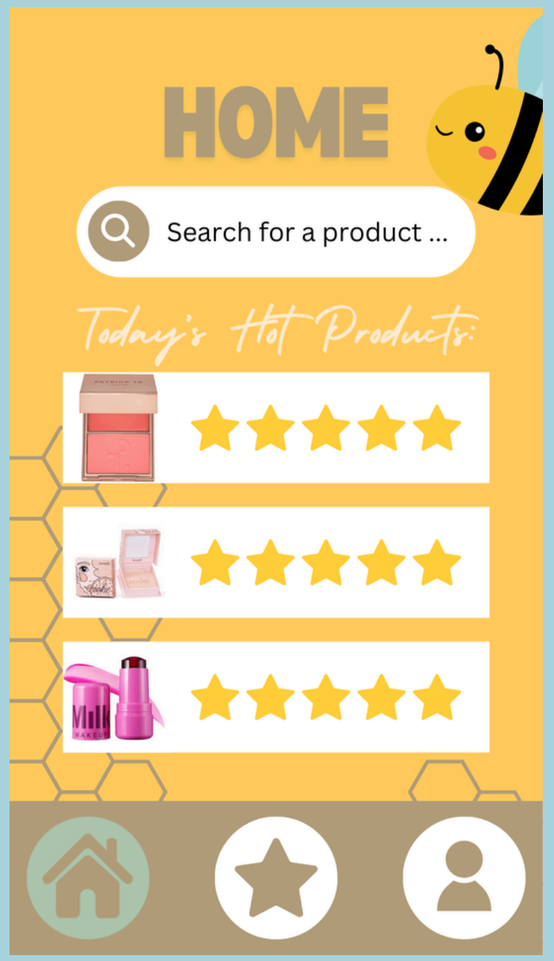
ConnectionScript.py | test.py

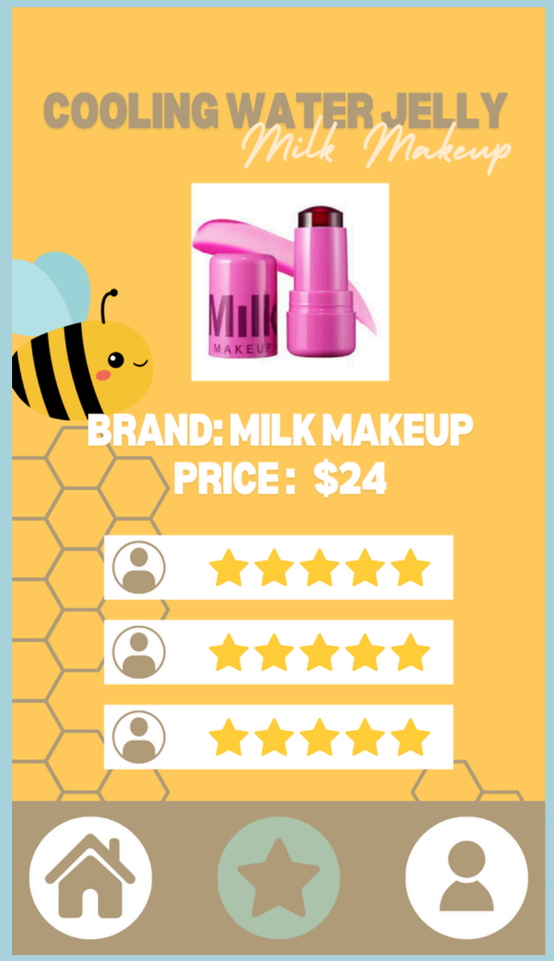
```python
mycursor.execute("SELECT * FROM User")
print('Here are all the users:\n')
for x in mycursor:
    print(x)
```

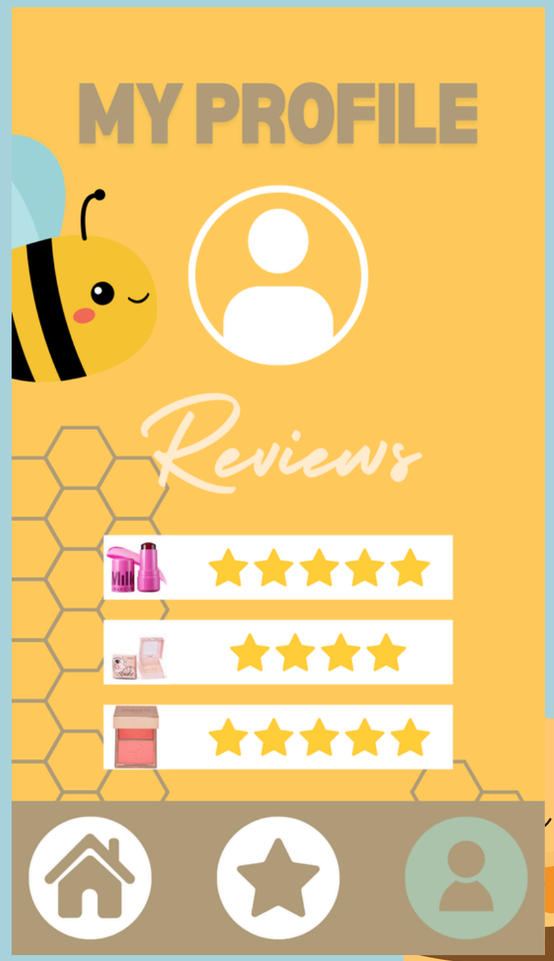
```
Here are all the users:

('AstuteWalrus', 'Homer', 'Pace', 'hpace@pace.edu', '19061906')
('Fluffybunny123', [redacted] 'fluffy!!')
('FunnyCapybara', [redacted] vecapybara1234')
[Finished in 511ms]
```

Deliverable #3 - User Interface







Security Measures

CAPTCHA required during login

Encryption of passwords and emails

We will not sell your data or track your data to be sold!

More Complex Queries

1. Select all reviews for all products and sort in descending order by Overall_5rating.
2. Select and group products by product name, and show the average Overall_5rating for each product.

ConnectionScript.py | test.py

```python
'''

#selects For instance, get all reviews for the product _ and sort in descending order.
try:
    mycursor.execute("SELECT p.product_name, p.brand, p.product_price, r.short_response, r.overall_5rating FROM Product as p, Review as r WHERE p.product_ID = r.product_ID ORDER BY r.overall_5rating DESC")

    print('Here are all the reviews:\n')

    for x in mycursor:
        print(x)
except:
    print("Could not select data, please check your syntax.")

```

```
Here are all the reviews:

('Precisely, My Brow Wax', 'Benefit Cosmetics', Decimal('27.00'), 'Product keeps my eyebrows in place all day!', Decimal('5'))
('Precisely, My Brow Wax', 'Benefit Cosmetics', Decimal('27.00'), 'Product keeps my eyebrows in place all day!', Decimal('5'))
('The Silk Serum', 'Tatcha', Decimal('98.00'), 'This product is my holy grail', Decimal('5'))
('Vanilla Lip Care Duo', 'Laneige', Decimal('34.00'), 'Keeps my lips super smooth all day long.', Decimal('5'))
('Precisely, My Brow Wax', 'Benefit Cosmetics', Decimal('27.00'), 'I like this product a lot, just wish the packaging was less ugly.', Decimal('4'))
('Major Glow Balm', 'Patrick Ta Beauty', Decimal('50.00'), 'This product is a little out of my budget, but works very well!', Decimal('4'))
('Vanilla Lip Care Duo', 'Laneige', Decimal('34.00'), 'I hate the taste and smell of this product. Never buying it again', Decimal('2'))
('Major Glow Balm', 'Patrick Ta Beauty', Decimal('50.00'), 'This product is way too overpriced, did not work for my skin.', Decimal('1'))
[Finished in 588ms]
```

ConnectionScript.py | test.py

```python
        print(x)
except:
    print("Could not select data, please check your syntax.")
'''

try:
    mycursor.execute("SELECT p.product_name, AVG(r.overall_5rating) FROM Product as p, Review as r WHERE p.product_ID = r.product_ID GROUP BY p.product_name")

    print('Here are all of the product names, each with the average 5-star rating for each product:\n')

    for x in mycursor:
        print(x)
except:
    print("Could not select data, please check your syntax.")
```

```
Here are all of the product names, each with the average 5-star rating for each product:

('Precisely, My Brow Wax', Decimal('4.6667'))
('Major Glow Balm', Decimal('2.5000'))
('Vanilla Lip Care Duo', Decimal('3.5000'))
('The Silk Serum', Decimal('5.0000'))
[Finished in 546ms]
```

Enforcing a Unique Constraint

ConnectionScript.py | test.py

```python
mycursor.execute("DELETE FROM Review WHERE review_ID = 3")
db.commit()

mycursor.execute("SELECT * FROM Review")
for x in mycursor:
        print(x)

#ensures that there is only one review, per person, per product
mycursor.execute("ALTER TABLE Review ADD CONSTRAINT OneReview_PerPerson_PerProduct UNIQUE (user_ID, product_ID)")

try:
    mycursor.execute("INSERT INTO Review(user_ID, product_ID, short_response, effectiveness, value, phys_properties) VALUES (%s, %s, %s, %s, %s, %s)", ('Fluffybunny123',1,'Product
        keeps my eyebrows in place all day!', 5, 5, 4))

except:
    print("You have already made a review for this product! Please consider updating your review instead.")
```

```
(1, 'Fluffybunny123', 1, 'Product keeps my eyebrows in place all day!', 5, 5, 4, Decimal('5'))
(2, 'AstuteWalrus', 1, 'I like this product a lot, just wish the packaging was less ugly.', 5, 5, 2, Decimal('4'))
(4, 'FunnyCapybara', 2, 'This product is a little out of my budget, but works very well!', 5, 2, 4, Decimal('4'))
(5, 'Fluffybunny123', 4, 'This product is my holy grail', 5, 5, 5, Decimal('5'))
(6, 'AstuteWalrus', 3, 'Keeps my lips super smooth all day long.', 5, 5, 5, Decimal('5'))
(7, 'AstuteWalrus', 2, 'This product is way too overpriced, did not work for my skin.', 1, 1, 1, Decimal('1'))
(8, 'Fluffybunny123', 3, 'I hate the taste and smell of this product. Never buying it again', 1, 3, 1, Decimal('2'))
[Finished in 708ms]
```

ConnectionScript.py | test.py

```python
mycursor.execute("SELECT * FROM Review")
for x in mycursor:
        print(x)
'''
#ensures that there is only one review, per person, per product
#mycursor.execute("ALTER TABLE Review ADD CONSTRAINT OneReview_PerPerson_PerProduct UNIQUE (user_ID, product_ID)")

#code to test unique constraint, tries to add an observation that violates the unique constraint
try:
    mycursor.execute("INSERT INTO Review(user_ID, product_ID, short_response, effectiveness, value, phys_properties) VALUES (%s, %s, %s, %s, %s, %s)", ('Fluffybunny123',1,'Product
        keeps my eyebrows in place all day!', 5, 5, 4))

except:
    print("You have already made a review for this product! Please consider updating your review instead.")
```

```
You have already made a review for this product! Please consider updating your review instead.
[Finished in 591ms]
```

Performance Optimization

1

Indexing on frequently queried columns like user_Id and product_ID

2

Read replicas in a distributed database for a read-heavy platform

3

Possible de-normalization if performance impact occurs from too many joins

THANK YOU!

Any questions?