System Test Report

Project Name: WaveStyled
Team Name: Wave INC: Matthew Daxner, Sanjay Shrikanth, Griffen Shu, Arka Pal, Dhatchi Govindarajan
Release Date: 6/1/2022

Testing Approach:

To test our components, we used a combination of automated and manual testing. For the frontend, our team incorporated manual testing, which involved going through the user-interface as a user to test certain functionalities, response times, and consistency with the backend data. This was done incrementally as product development went along. Using Expo Go, we ran a server on our computers that connected the application to our phones during development which refreshed the user interface as changes were made in the code immediately. This gave immediate feedback on the code output and allowed us to test the aesthetics and functionality of the frontend smoothly. For the backend, we relied on automated testing. To test the wardrobe and recommender, we created scripts that made sure any updates from the API calls happened correctly and also to check that the data transformation for the neural network were done properly.

Scenarios:

Sprint 1:

- As a user I want to be able to add items from my actual wardrobe to my virtual wardrobe so that I can easily visualize everything I own.

- As a user, I want my wardrobe to be saved when I close the app so that I do not need to re-enter it whenever I reboot the app.

Scenario:

1. Start WaveStyled
2. Press “Create Account”
    a. Name = <TestDummy>
    b. Email = <test@gmail.com>
    c. Password = <testing123>
    d. Repeat Password = <testing123>
    e. Press “Create Account” to Log In
3. Press “Add Item” button in top right corner
    a. Press Camera Icon on top
        - Take Image of Item or Select from Camera Roll

   b. Fill out details of Item
   c. Press “Save”
4. Press “Logout” Button on top left corner
5. Close app and reopen or just log back in and User will see the new item loaded on the top of the home screen. Swipe right and tap on update to view/modify saved details.

Sprint 2:

- As a user I want to see my recommended fits so that I can easily choose(or not) my outfit for the specified weather occasion.

Scenario:

1. Start WaveStyled
2. Either Login with existing email or Create a New Account using steps specified in Sprint 1 Scenario
   a. For testing purposes, use the following information for a preloaded wardrobe account meant for app development
      i. Email: <matt@gmail.com>; Password: <matthew>
3. Press the Shirt Icon on the Bottom Tab, or the “Recommend” Tab
4. Select the desired Occasion
5. Select the desired Weather
6. Assuming the user has enough items in the wardrobe to generate fit for chosen settings, the screen will display generated fit including images
7. Either:
   a. Press the Green Check or Swipe **Right** to “Like” the fit but not choose the Outfit of the Day and see the next fit
   b. Press the Red X or Swipe **Left** to “Dislike” the fit but not choose the Outfit of the Day and see the next fit
   c. Press the Blue Hanger to Select the outfit of the day

Sprint 3:

- As a user, I want to be able to calibrate the recommendation model so that it gives me outfits that fit my preference and learn what I like to wear for future recommendations.
- As a user I want to be able to submit a picture of my clothing item so that when I receive a recommendation, I can easily identify which piece of clothing it is referred to and clearly visualize my outfit right on my phone screen.

Scenario:

1. Start WaveStyled

Unit Test Report

2. Either Login with existing email or Create a New Account using steps specified in Sprint 1 Scenario
    a. For testing purposes, use the following information for a preloaded wardrobe account meant for app development
        i. Email: <matt@gmail.com>; Password: <matthew>
3. Press the Heart Icon on the Bottom Tab, or the "Calibrate" Tab
4. Press the "Get Fits" Button
5. Assuming the user has enough items in the wardrobe to generate fits, the screen will display generated fit including the associated Images of the outfit items
6. User can observe the **Weather** and **Occasion** the displayed fit is being shown for
7. Either:
    a. Press the Green Check or Swipe **Right** to "Like" the fit and see the next fit
    b. Press the Red X or Swipe **Left** to "Dislike" the fit see the next fit
8. Like/Dislike (or Swipe Right/Left) 5-6 times
9. Press the "End Calibration Button"
    a. After 4-5 seconds, the Recommendation Model is now tuned with the new data

Sprint 4:

- As a user, I want to be able to log into my account to access my wardrobe and outfits of the day
- As a fashion designer, I want to be able to have multiple tuned wardrobes so that I can have different recommendations for different sets of clothes (Not completed)

Scenario:

1. Start WaveStyled
2. If the User does not have an Account, follow Create Account steps in Scenario 1; otherwise Log In using existing account information
3. User should see item now rendered on the home screen (wardrobe screen)
4. Press the Hanger Icon on the bottom tab to navigate to the Outfit of the Day Tab
5. Either:
    a. User will see "No Outfit of the Day Available"
        i. Navigate to the "Recommend" Tab, or the Shirt Icon
        ii. Select a Weather
        iii. Select an Occasion
        iv. Press the Blue Hanger Button on a fit that the User chooses as the outfit of the day
        v. Navigate back to the Hanger Icon
        vi. The User should be able to see that chosen fit on the screen
    b. User will see the last chosen Outfit of the Day

Unit Test Report

Unit tests

- Module: python_to_postgres.py
  - Test: Insert to wardrobe database:
    - Equivalent classes: list of integers U list of strings U null value
    - Test cases:{ (1,2,3,4,4) U (‘Brih’)}
  - Test: Delete from wardrobe database
    - Equivalent classes: list of integers U list of strings U NULL
    - Test cases: { (1,2,3,4,4) U (‘Brih’)}
  - Test: Get from wardrobe database
    - Equivalent classes: * U
- Module: Recommender_tests.py
  - Test: Ensure that the recommender can be loaded :
    - Equivalent classes: {non existent file U non path string U existing file path equal to number of rows in csv file }
    - Test cases: { (./csv_files/outfits.csv) -> 108 U ((./nonexistant.csv) -> Error}
  - Test: Checks that Recommender handles a failure to load
    - Equivalent classes :{valid csv input file U invalid csv file}
    - Test cases : {(No dataframe) -> Error}
  - Test: Assure clothing data is normalized:
    - Equivalent classes: {dataframe with pieceIDs U empty dataframe}
    - Test cases: { (dataframe) -> ['hat', 'shirt', 'sweater', 'jacket', 'bottom_layer', 'shoes', 'misc']}
  - Test: Ensure that a model can be saved and then loaded again:
    - Equivalent classes: {non existent machine learning model U existing machine learning model}
    - Test cases: { (“wavestyled”) -> wavestyled model is saved}
  - Test: Ensure the model can train with the data
    - Equivalent classes : {Valid encoded dataframe, invalid dataframe}
    - Test cases: { (./csv_files/outfits.csv) -> 108}
  - Test: Evaluate whether the model can generate recommendations:
    - Equivalent classes: {valid data for the model U invalid data U non existent data }
    - Test cases: { (./csv_files/outfits.csv) -> 0/1 labels, valid fit/weather/occasion }
- Module: wardrobe_test.py
  - Test: Ensure a wardrobe can be loaded from csv file:
    - Equivalent classes: {}
    - Test cases: { (./csv_files/outfits.csv).length = 156 }
  - Test: Test the initialization of a wardrobe object:

Unit Test Report

        - Equivalent classes: {Valid wardrobe initialization code}
        - Test cases: { Obj(Wardrobe).length = 0 }
    - Test: Test adding an item to the wardrobe
        - Equivalent Classes: {Valid wardrobe object U Invalid Wardrobe object}
        - Test Cases: {Obj(Wardrobe).length = 157 }
    - Test: Test deleting an item from the wardrobe
        - Equivalent Classes: {Valid pieceid U Invalid pieceid}
        - Test Cases: {Obj(Wardrobe).length = 156 & Obj(Wardrobe).get(deletedID) ==-1}
    - Test: Try deleting an non existent item from the wardrobe
        - Equivalent Classes: {Invalid pieceid U Invalid pieceid}
        - Test Cases: {Obj(Wardrobe).length = Obj(Wardrobe).length & Obj(Wardrobe).get(pieceid) = None}
    - Test: Test updating an item in the wardrobe
        - Equivalent Classes: {Valid Wardrobe Object U Invalid Wardrobe Object}
        - Test Cases: {Obj(Wardrobe).get(item ID) == updated item}
    - Test: Test wardrobe outfit generation
        - Equivalent Classes: {Valid Occasion & Valid Weather U Valid Occasion & Invalid Weather U Valid Weather & Invalid Occasion U Invalid Weather & Invalid Occasion}
        - Test Cases: {Obj(Wardrobe).gen(oc, we).length = 7 (size of fit) & Obj(Wardrobe).gen(oc,we) are all integers and pieceids in the dataframe}
    - Test: Test wardrobe get random fits
        - Equivalent Classes: {Valid Occasion & Valid Weather U Either Weather or Occasion Invalid U Valid Fits U Invalid Fits}
        - Test Cases: {Obj(Wardrobe).genRandom()[1] has valid occasions and weathers, Obj(Wardrobe).genRandom()[0] has valid fits that are all of size 7 and have valid pieceids}
- Module: user_tests.py
    - Test: Test users can be generated
        - Equivalent Classes: {Valid User Wardrobe U Invalid User Wardrobe}
        - Test Cases: {Obj(Wardrobe).length == 156}
    - Test: wardrobe Operations
        - Test: Test add operation
            - Equivalent Classes: {Valid Wardrobe Object U Invalid Wardrobe Object}
            - Test Cases: {Obj(Wardrobe).length == 157}
        - Test: Test update operation
            - Equivalent Classes: {Valid Wardrobe Object U Invalid Wardrobe Object }

Unit Test Report

        - Test Cases: {Obj(Wardrobe).get(item ID) == updated item}
    - Test: Test delete operation
        - Equivalent Classes: {Valid Wardrobe Object U Invalid Wardrobe Object}
        - Test Cases: {Obj(Wardrobe).get(PieceID) == None and Obj(Wardrobe).length == 156}
- Test: User Calibration
    - Equivalent Classes: {Valid Ratings U Invalid Ratings U Valid Attributes U Invalid Attributes U Valid Fits U Invalid Fits U Valid Recommender U Invalid Recommender}
    - Test Cases: {Obj(User).RecommenderNEW.length == Obj(User).RecommenderOLD.length + number of items added & All Ratings are 1s or 0s & All Attributes are Valid mappings & All Fits are integers and valid pieceids }
- Test: Load the UserBase Object
    - Equivalent Classes: {Valid User Base Object U Invalid User Base Object}
    - Test Cases: {add User to UserBase and check that UserBase.length == 1 and UserBase.get(UserID) works}