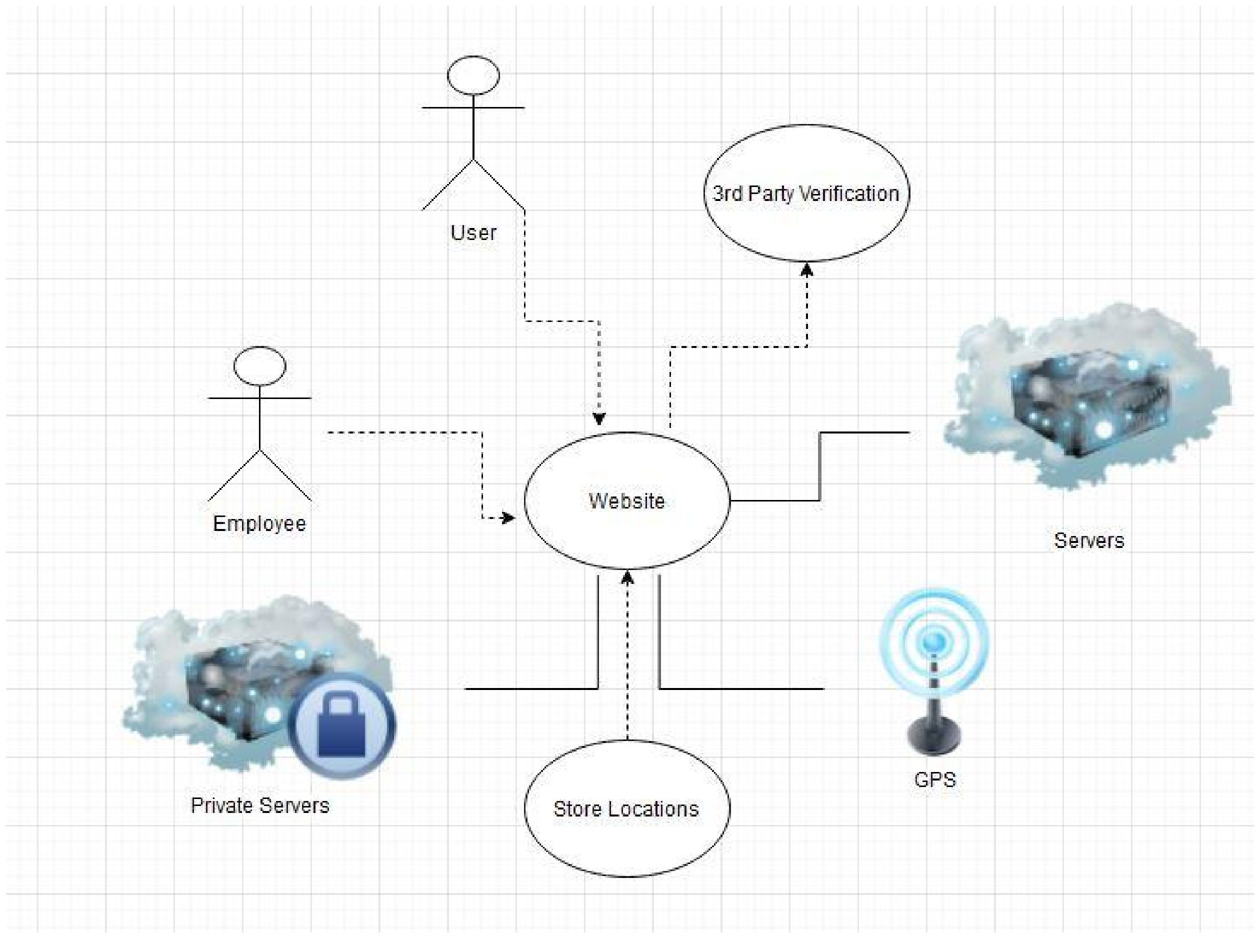
Software Design Specification: Test Plan

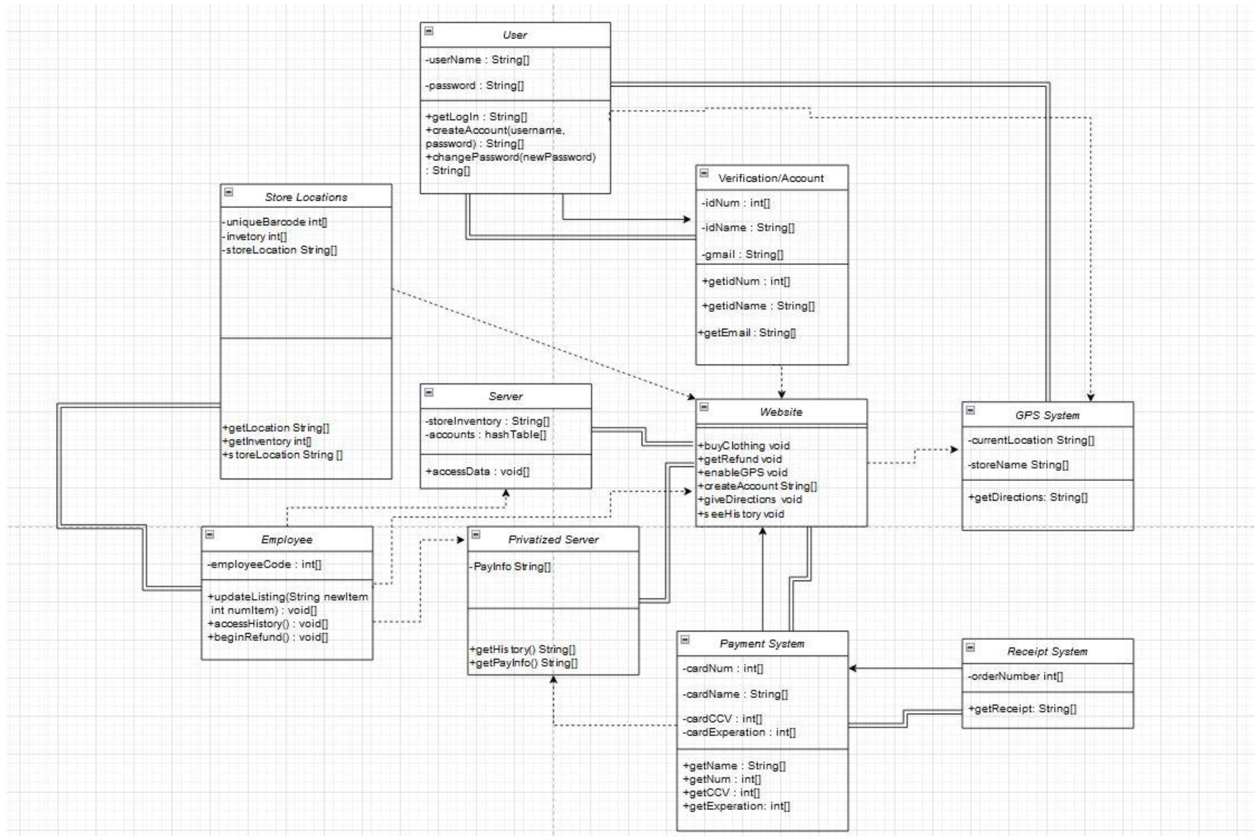
Prepared by Ethan Fox, Jason Park Fabian, and Alan Yi

Architectural Diagram



This is our architectural diagram for our system. At the core of the diagram is the website, which everything interacts with. There are three different interactions with the website, the first one being that someone, or something, is using the website. This interaction is used by the employee, user, and store locations. The user uses the website in order to access their account and purchase things from the store. The employee interacts with the website in a similar way, but they use the website to also update listing and any information on the website. The store location does the same as the employee and uses the website to update information and other things. Once the website receives these updates, is accesses its private server and servers. These are a different interaction because the website does use these servers, but has them at its disposal. These servers primarily keep information that the website is sent, and specifically the private server keeps sensitive information, like credit card numbers and things of that nature. The website also has a GPS system that keeps track of the location of the user to inform them on directions to the store location. The last interaction on this diagram is that the website uses a 3rd party verification. The website doesn't have a 3rd party verification, but uses one because it is a different entity. This 3rd party verification helps confirm user's logging, purchase confirmations, credit card confirmations, and other things that require a confirmation.

UML Diagram



In our updated UML Diagram, the **Employee**, **Server**, and **User** classes got revised. In the **Employee** Class, we added the employeeCode, which is received when an employee tries to log into the website. This code ensures that it is an employee signing in, and not someone else. It also received three new operations, updateListing, accessHistory, and beginRefund. The updateListing used to be in the store location class, but it makes more sense that the employee is updating the listing of inventories because they are making the physical transaction. The accesshistory is also used in order for employees to see if a customer can get a refund. If they can, the employee can begin the refund process with beginRefund, and that will prompt the payment system to begin the refund. The **Server** class got storeInventory and accounts, both of which are new. The storeInventory helps the server store all the inventory and other things that the store keeps track of. The accounts also stores all the user accounts in a hashTable in order so that they can be accessed quickly. It also has accessData, which just goes into the server and accesses the data for whoever is looking for it. Lastly, the **User** Class got userName and password, both of which are strings. userName stores the user's account name, and password stores the password associated with the account. The operation that the User class got are getLogin and createAccount(userName, password), which are also strings. getLogin gets the users login information that they submit, while createAccount is used when a user does not have an account. That needs two inputs, userName and password, which will be used to create the user a new account.

Unit Testing

Unit Test for User

```
User:
    Inputs: userName(String name), password(String password), getLogin()
    Test:
        If (getLogin == name + " " + password)
            printf("Test passed")
        Else
            printf("Test failed")
```

In this example we are testing the user class, and specifically the userName(String name), password(String password), and getLogin() operations. The user enters their username and password for their account, and if the getLogin() function returns the same username and password, the test passes, otherwise it fails.

User:

```
    Inputs: userName(string name), changePassword(String newPassword), getLogin()
    Test:
        changePassword(newPassword);
        If (getLogin() == userName + newPassword)
            print("Test Successful")
        Else
            print("Test Failed")
```

In this unit test we are again in the user class, but this time checking to see if the changePassword operation is successful. For this, the changePassword operation is used to change the password, and after that it checks to see if the login returned from getLogin() is equal to the same username but with the new password. If it is not the same, then the test fails.

Unit Test for Verification/Account

Verification/Account

```
    Inputs: idNum(int IDnum), getidNum()
    Test:
        idNum(idNumber)
        if(getidNum() == idNumber)
            print("Test Successful")
        Else
            print("Test Failed")
```

This unit test is checking whether the idnumber is correctly stored in the verification/account class. For this, it sets the idNum at the beginning and then compares if it is the same as the getidNum() operation, and if it is, then it is successful. If it is not the same, then it has failed.

Verification/Account

```
    Inputs: idName(String name), getidName()
    Test:
        idName(userName)
        if(userName == getidName())
            print("Test Successful")
        Else
            print("Test Failed")
```

During this unit test it is specifically seeing if the idName is storing the idName correctly. In order to do this, it sets the idName to userName, and then sees if userName is equal to getidName(). If they are equal, then it prints out that the tast was successful, if they are not equal then it prints that the test has failed.

Integration Testing

All of these Tests are conducted with the idea that all of our system / unit testings are properly / well tested.

1) Creating an account
   Testing how to create and Validate an account
      Inputs/ requirements: Username, Password, email confirmation
      Returns: User account
   a) **Test 1 Inputs: (Valid Information)**
      i) Have an account already created
      ii) Has the correct passwords and username
      iii) Have the correct email verification
   b) Return:
      i) Being able to login
      ii) Able to see your idNum
   c) Test1: In this test we are testing already made accounts to see how it would work and if it would work properly, this will go between the User and the Verification/Account so that they would get an email confirmation like a two-face security before login and then after login if everything is correct it would show them their idNum and their account.
   d) **Test 2 input: (Missing Information)**
      i) Have correct usernames and passwords
      ii) Does not have email verification
   e) Return:
      i) Not able to login, retry to send another email.
   f) Test 2: In this test if the user have no email verification then the system would not work properly and the user would not be able to login to their account. They have correct password and username but not the two face security code.
   g) **Test 3 input (invalid information)**
      i) Have incorrect username and password
   h) Return:
      i) Unable to login, please create an account
   i) Test 3: in this test we see what would happen if the user have not username nor password, so the system would be empty and it would recommend the user to make a new account.

2) Update Listing
   Test how update listing would work, we assume that they have an inventory.
   Inputs: the items that the Employee wants to add and the amount of it
   Returns: Changes the amount of cloth in inventory
   a) **Test 1 input (Valid information)**
      i) There is an inventory
      ii) There is an item the Employee wants to add or delete
      iii) Uses updateListing to add it or subtract from inventory
   b) Return:
      i) Added or substrates items on inventory that is in store Location
   c) Test 1: We do this test to test if the storeLocation and the Employee works when working together and if they update the listing we can see it in the inventory and this can be used for example in checkouts where they can subtract from the inventory, or when new cloths arrives they can add it to the inventory.
   d) **Test 2 input (Invalid information)**
      i) There is an inventory
      ii) There is an item that employee wants to add
      iii) But when using update Listing they put 0
   e) Return:
      i) Nothing
   f) Test2: there was no item added to the inventory so the program would just accept the 0, there would be no errors but nothing would change.
   g) Test 3 input (Invalid information)
      i) There is an inventory
      ii) There is an item that employee wants twice
      iii) But the inventory only has one left.
   h) Return:
      i) It will return an error telling the employee that the inventory is empty and can not subtract anymore.
   i) Test 3: This is to test what would happen when the employee wants more but there aren't enough items in the inventory. This will check the number of times in the inventory first then tell the employee that there isn't enough to sell.

System Testing

All of these Tests are conducted with the idea that all of our integration / unit testings are properly / well tested.

Furthermore these tests are expected to be conducted in ways that are ethical, for instance creating “mock” payment systems in order to avoid the transaction of real credit cards / identities from being endangered from these testings.

****NOTE**** All of these tests below assume that the user has a valid account, since they wouldn’t be able to access the website.

1) Purchase Clothing
   Testing the Purchasing Clothing System.
   Inputs / Requirements: Credit Card Information, Account
   Returns: Email receipt
   - a) **Test 1 Assumptions/Inputs: (Valid Information)**
     - i) Has an account w/ personal email
     - ii) Has a store location selected
     - iii) Has VALID card: Num, Name, CCV, Expiration
       - (1) Furthermore card used as the proper funds to buy the clothing
     - iv) buyClothing > 0 (cart is not empty)
   - b) **Test 1 Return:**
     - i) Receipt verifying the purchase
   - c) **Test 1 Testing / Expectations**
     - i) Check cart before and after purchase
       - (1) After purchase expected that cart is emptied
     - ii) Validate VALID card information
       - (1) Expected that none of the information is “bounce backed”
     - iii) Process card information
       - (1) Expected that goes through “smoothly”
       - (2) Check bank balance before and after, expecting that change in balance is equivalent to cart total.
     - iv) Receiving email receipt
       - (1) Expected that user will receive receipt in an email connected to their account.
   - d) **Test 2 Assumptions/Inputs: (Invalid Information)**
     - i) Has an account w/ personal email
     - ii) Has a store location selected
     - iii) Has INVALID card: Num, Name, CCV, Expiration
     - iv) buyClothing > 0 (cart is not empty)
   - e) **Test 2 Return:**
     - i) Error message (Prompt new valid info)
   - f) **Test 2 Testing / Expectations**

   i) Check cart before and after purchase
      (1) Expected Cart is still full
   ii) Validate INVALID card
      (1) Expected to throw back card information and throw errors. (END OF TEST)

g) **Test 3 Assumptions/Inputs: (Valid info. No money)**
   i) Has an account w/ personal email
   ii) Has a store location selected
   iii) Has VALID card: Num, Name, CCV, Expiration
      (1) Furthermore card doesn't have enough funds
   iv) buyClothing > 0 (cart's not empty)
h) **Test 3 Return:**
   i) Error message (Prompt to try new card)
i) **Test 3 Testing / Expectations**
   i) Check cart before and after purchase
      (1) After purchase expected that cart is emptied
   ii) Validate VALID card information
      (1) Expected that none of the information is "bounce backed"
   iii) Process card information
      (1) Expected to run into error, throwing a message at the user. (END OF TEST)

2) Request Refund
   Testing the refund system
      Inputs: Account, Purchase Receipt / Order Number, Employee approval
      Outputs: Confirmation email
   a) **Test 1 Assumptions/Inputs: (Valid Information)**
      i) Has an account w/ personal email
      ii) Has valid receipt / order number
         (1) Furthermore has "approved" refund (by employee)
   b) **Test 1 Return:**
      i) Confirmation email for refund
   c) **Test 1 Testing / Expectations:**
      i) Check the receipt / order number
         (1) Since valid, return receipt for refund to customer.

   d) **Test 2 Assumptions/Inputs: (Invalid Information)**
      i) Has an account w/ personal email
      ii) Has invalid receipt / order number
   e) **Test 2 Return:**

        i) Error message prompting invalid order number

f) Test 2 Testing / Expectations:

- i) Check receipt / order number
    - (1) Since invalid, throw an error at the customer.

g) Test 3 Assumptions/Inputs: (Valid info but employee rejects)

- i) Has an account w/ personal email
- ii) Has valid receipt / order
    - (1) But is “rejected” by employee

h) Test 3 Return:

- i) Message to customer that REJECTS their return

i) Test 3 Testing / Expectations

- i) Check receipt / order number
    - (1) Since valid pass onto employee
    - (2) Employee rejects refund

Development plan and timeline

As a group we decided to make some changes to the UML class diagram during the time we had in class on monday. Furthermore we decided the roles between members, Park would take care of the Unit Testing, Alan would take care of integration testing, and Ethan would take care of the System testing. We planned on having at least one of the testings done by Wednesday. And finish the project with peer review on Friday, the day of submission.