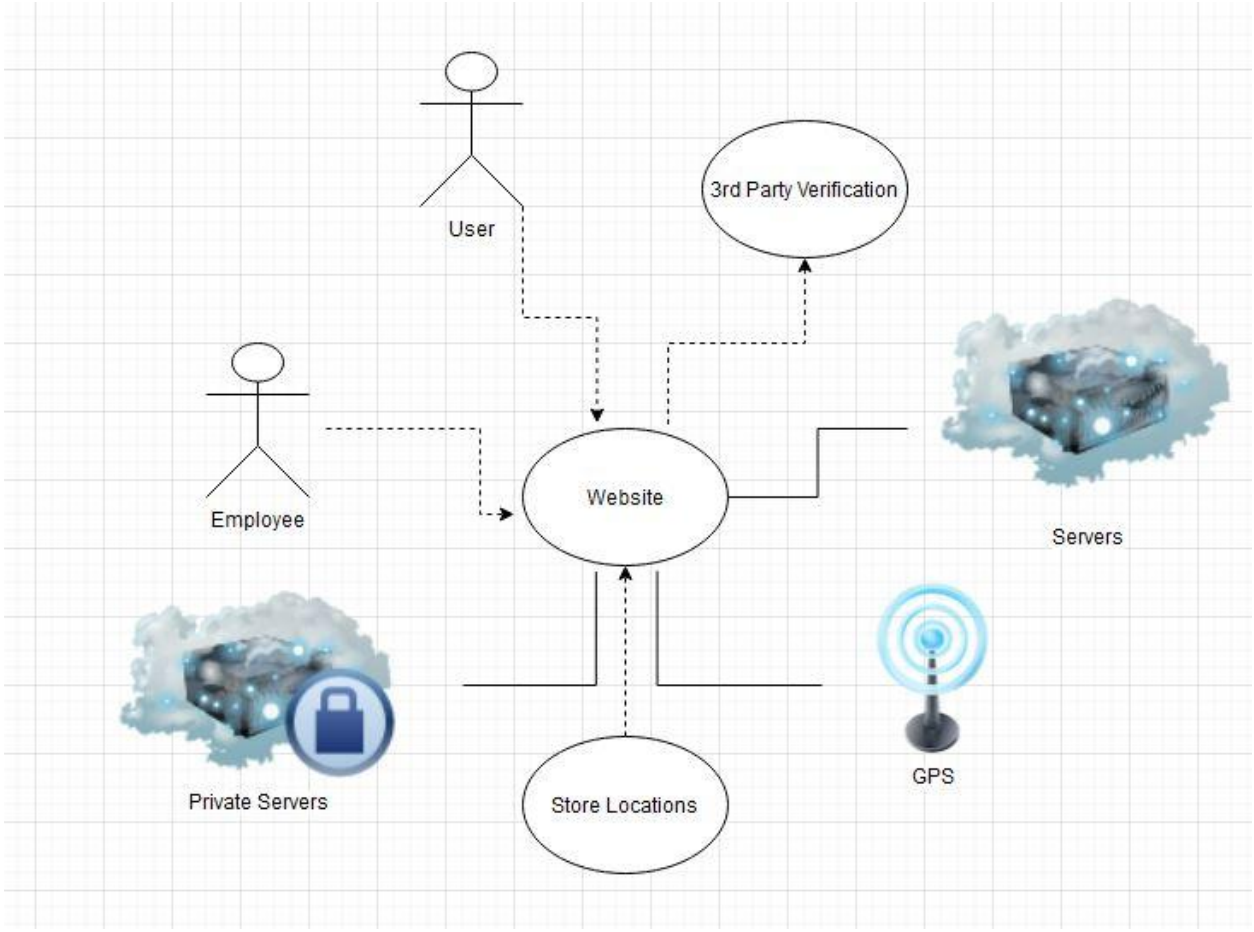
Software Design 2.0

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

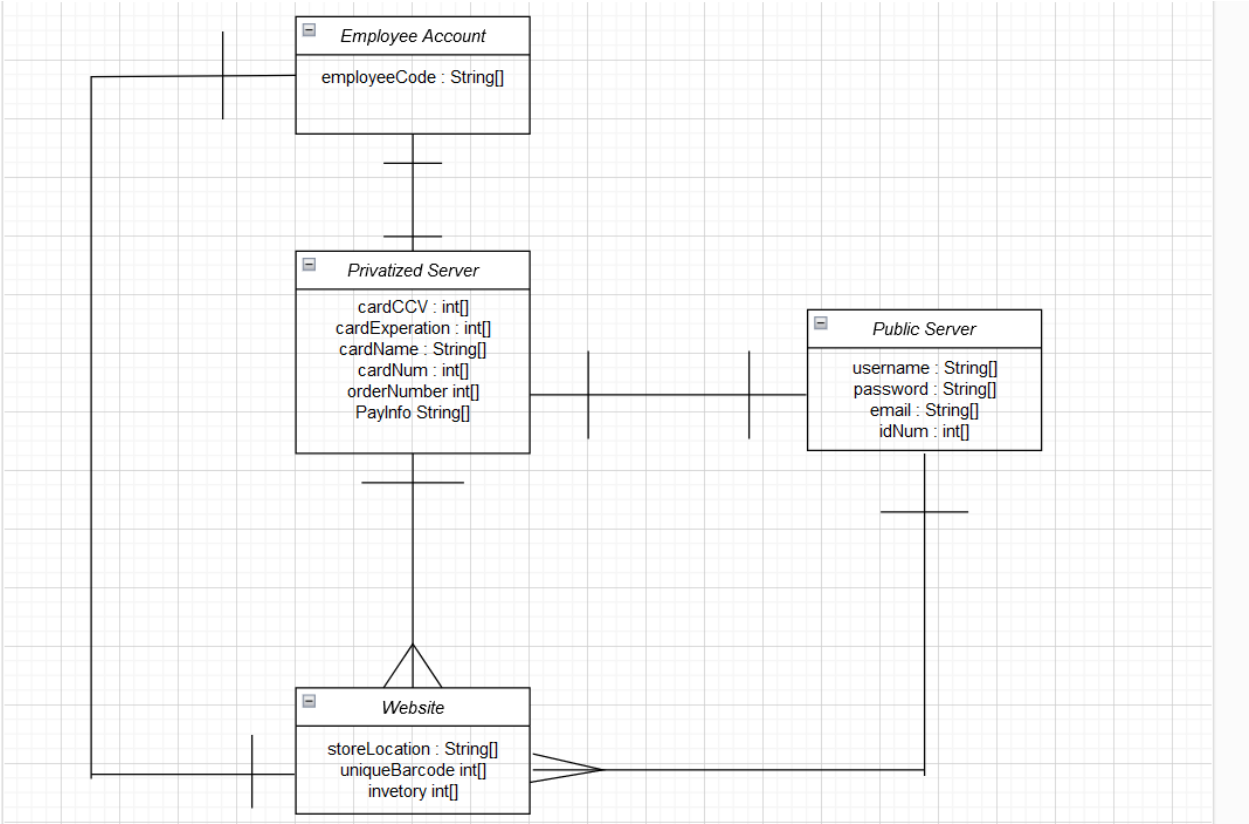
Architectural Diagram



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.

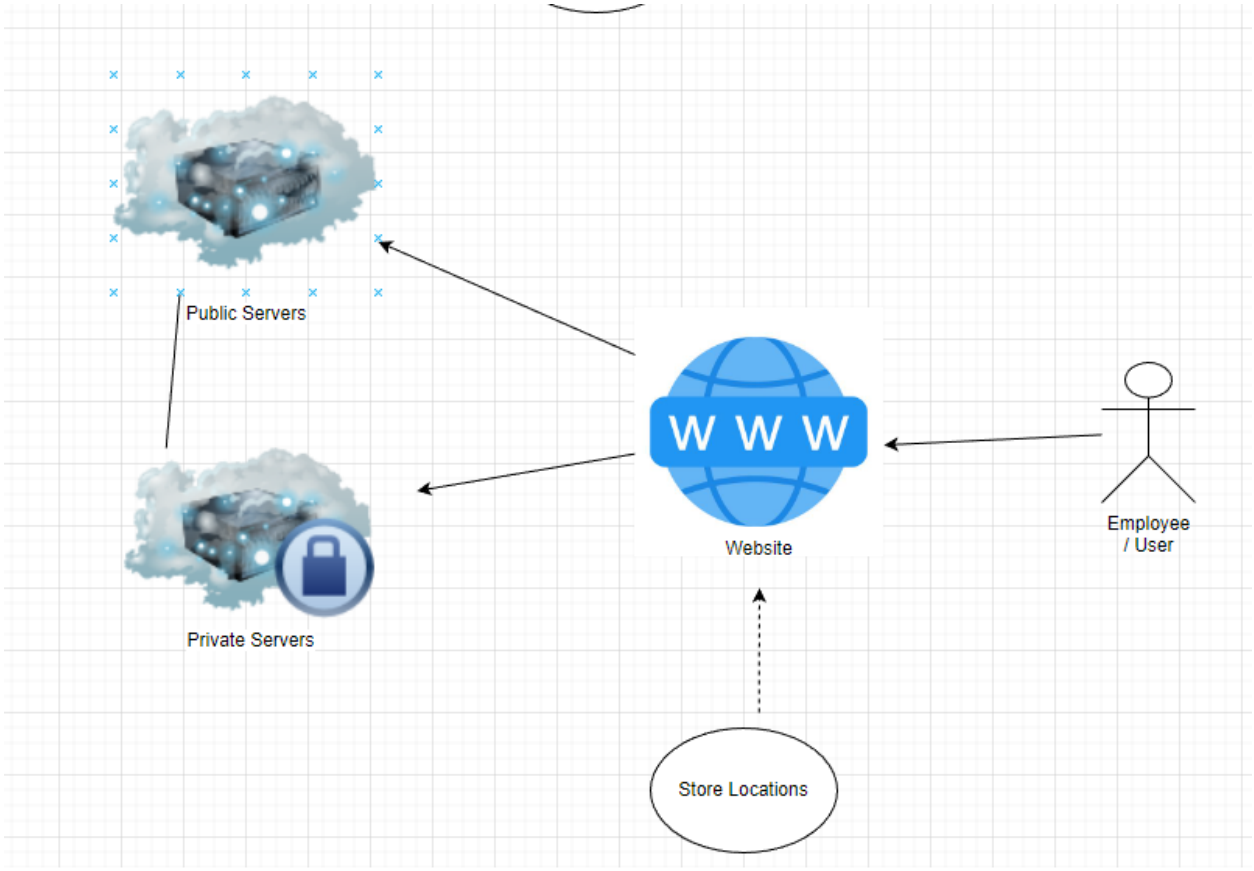
Data Base



The main part of the database diagram is the privatized server. This is where all the sensitive information, like credit card information, payment information, and order numbers. We decided to put all of these in the privatized server because this information is more sensitive than other things in the public server and if compromised could lead to serious issues. It has a lot of different interactions, but the privatized server always is one. For example, the connection between the privatized server and public server is one to one because there is only one privatized server and only one public server. The counter to the privatized server is the public server, which is similar to the private server, but stores less sensitive information. It only stores the username, password, email, and idNum that really has to just deal with account information. All of this is in the public server because it is not as sensitive as the information in the privatized server. That means they only communicate one to one, but the connection between the privatized server and

website is one to many. This is because there are multiple different websites, each store has their own, but they connect to only one privatized server. That relationship is the same between the public server and the website because there is also only one public server. Due to there being multiple websites, that means it stores the storeLocation, the barcodes for the clothes, and the inventory of that specific store. Each store has different values for these variables, but they all function the same. The last database is employee account, which only stores the employeeCode. It only stores one thing because it is essentially a normal user account, which is stored in the public server, but it is able to edit things in the server.

Updated Architectural Design

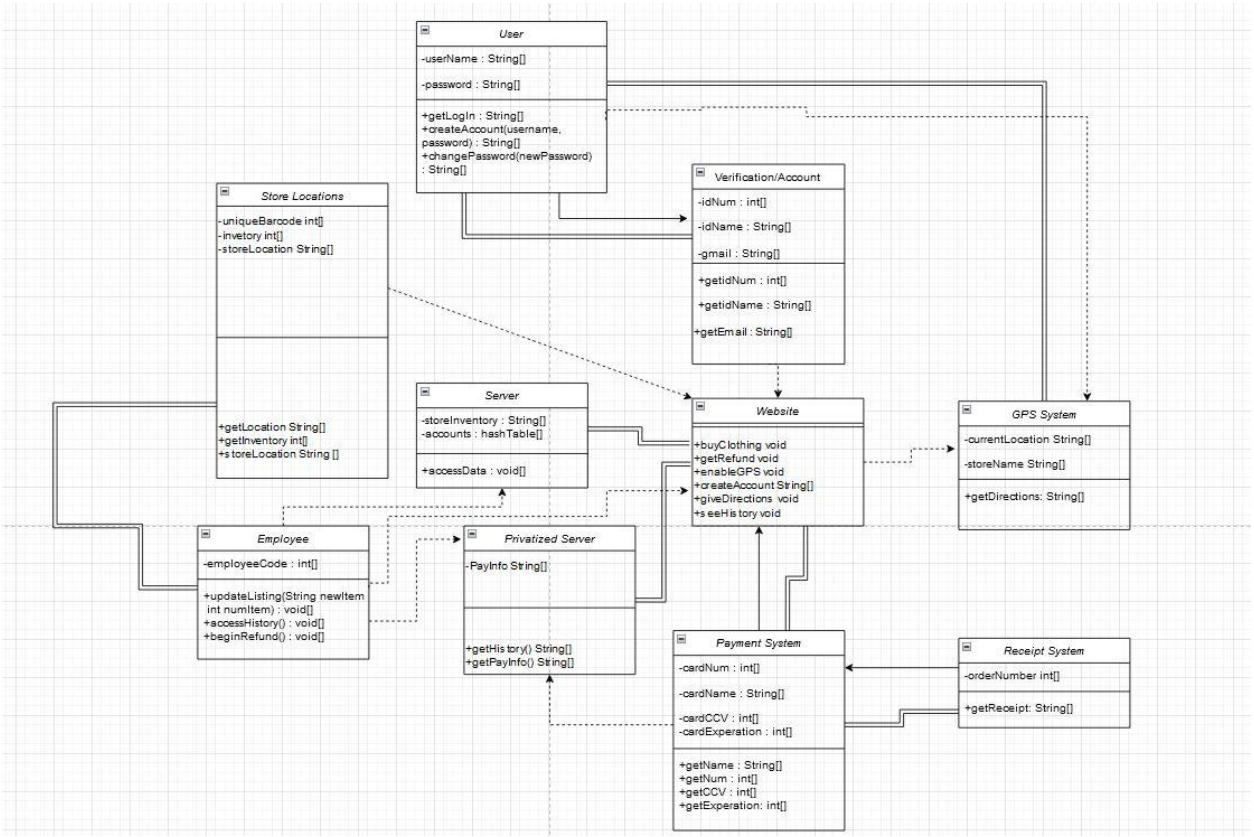


For the most part our first diagram was focused on the nodes that we previously had made, we changed the nodes locations and where it was directed to, since we had to worry about the data managements and the relations between each of them, so it was necessary to erase the 3rd party verification because it was already inside Employee/User, and we also changed the direction of the nodes, where now the public service and the private server associates with each other and connects to each other and it is not through the website but through themselves. The website will be generalized from the servers since the website is part of the server, then we see how the location will be using the website to get the locations. Finally we can see how the Employee/User is also part of the website since they have access to the inside of the website and can use it individually.

SQL vs Non-SQL

We decided to choose an SQL diagram primarily because our data will be relationship based. All nodes within our database will interact with each other, for example looking at our diagram there is a relationship between the private / public servers and the website that will allow users to access the information. Since the SQL strategy is focused on interpreting and manipulating data, we thought that it would be best for the clothing store as the overall "skeleton" (will remain as a clothing store) of the website would be static with values being manipulated within the structure. Furthermore we decided to pick SQL databases since NoSQL cannot guarantee ACID properties, which is not ideal as we will be handling sensitive information such as credit card information. Lastly we decided to avoid non-SQL due to its poor querying functionality, we figured that querying and reporting would be common in our software design as the user / employees would be constantly requesting various information.

UML Diagram


User
-userName : String[]
-password : String[]
+getLogIn : String[]
+createAccount(username, password) : String[]
+changePassword(newPassword) : String[]
Verification/Account
-idNum : int[]
-idName : String[]
-gmail : String[]
+getidNum : int[]
+getidName : String[]
+getEmail : String[]
Store Locations
-uniqueBarcode int[]
-invetory int[]
-storeLocation String[]
+getLocation String[]
+getInventory int[]
+storeLocation String []
Server
-storeInventory : String[]
-accounts : hashTable[]
+accessData : void[]
Website
+buyClothing void
+getRefund void
+enableGPS void
+createAccount String[]
+giveDirections void
+seeHistory void
GPS System
-currentLocation String[]
-storeName String[]
+getDirections: String[]
Employee
-employeeCode : int[]
+updateListing(String newItem int numItem) : void[]
+accessHistory() : void[]
+beginRefund() : void[]
Privatized Server
-PayInfo String[]
+getHistory() String[]
+getPayInfo() String[]
Payment System
-cardNum : int[]
-cardName : String[]
-cardCCV : int[]
-cardExperation : int[]
+getName : String[]
+getNum : int[]
+getCCV : int[]
+getExperation: int[]
Receipt System
-orderNumber int[]
+getReceipt: String[]