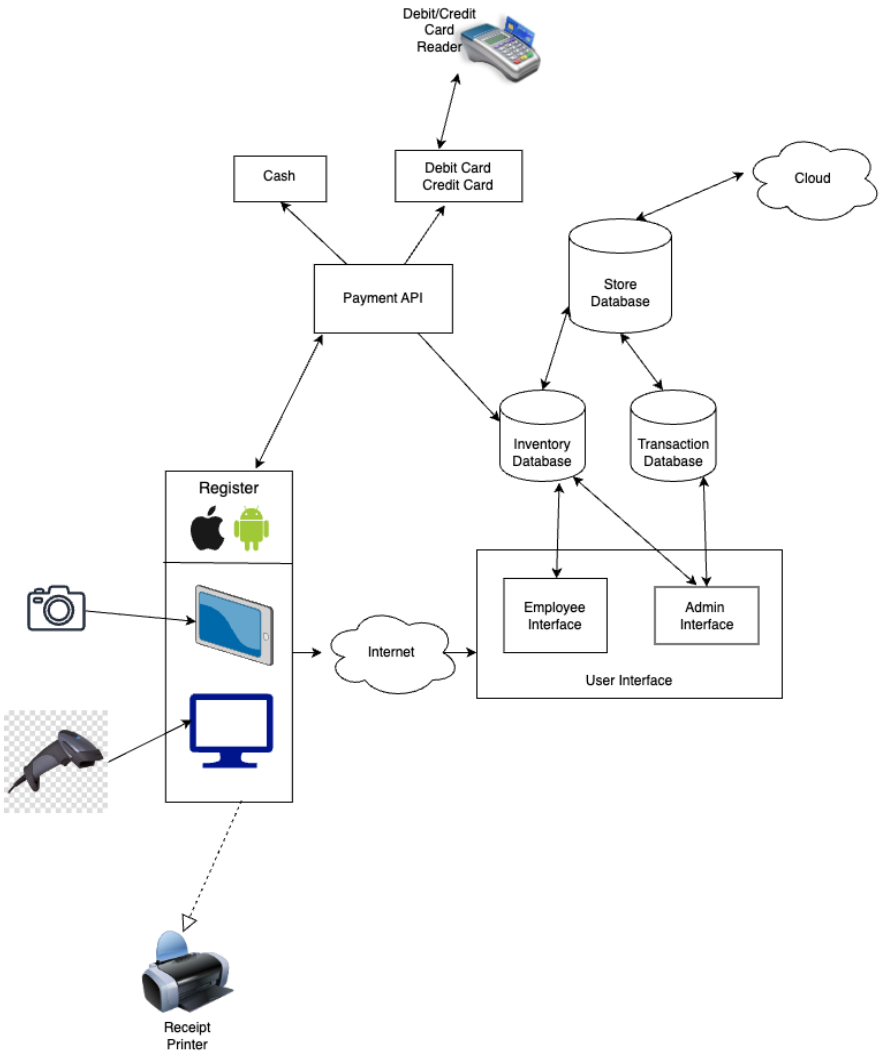
Software Design 2.0

Architecture Diagram

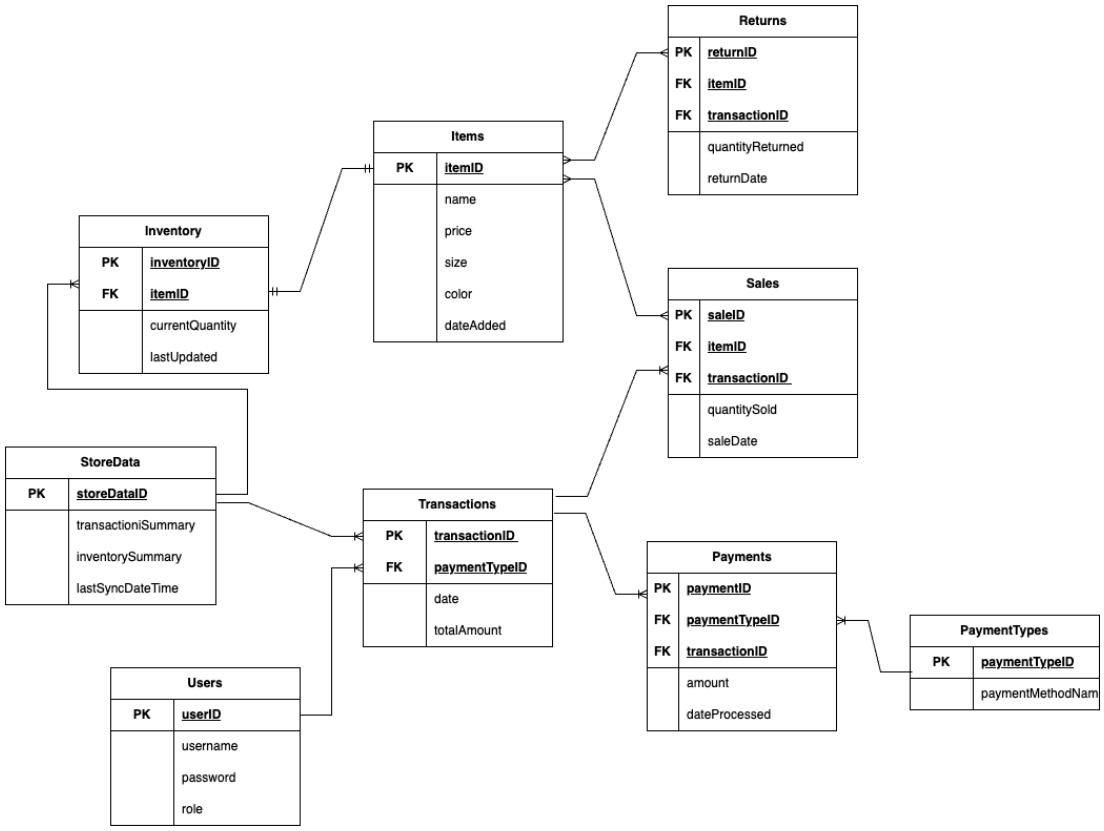


Updates to Architecture Diagram:

We do not plan to make changes our software architecture diagram because we believe that our existing diagram contains all necessary information. In terms of data management, we decided to keep the number of databases the same because we wanted the Transaction Database to only be accessed by the admin and not the employee. There will be one large store database with all the inventory and transaction data that will be stored in the cloud.

Data Management Strategy

Entity Relationship Diagram:



Data Dictionary:

StoreData

Name	Description	Type	Length Max	Range of Values
storeDataID (PK)	Unique identifier for each store data record	Int		Auto-increment, Unique
transactionSummary	Summary details of recent transactions	Varchar	255	Varies
inventorySummary	Summary details of current inventory	Varchar	255	Varies
latestSyncDateTime	The last date and time of synchronization	DateTime		Date and Time format

Inventory

Name	Description	Type	Length Max	Range of Values
inventoryID (PK)	Unique identifier for each inventory record.	Int	-	Auto-increment, Unique
itemID (FK)	Links to the Items table	Int	-	Foreign Key
currentQuantity	Current stock level of the item	Int	-	Varies
lastUpdated	The last date and time the inventory was updated	DateTime	-	Date and Time format

Items

Name	Description	Type	Length Max	Range of Values
itemID (PK)	Unique identifier for each item	Int	-	Unique
name	Name of the item	Varchar	255	
price	Price of the item	Float	-	
size	Size classification of the item	Varchar	255	
color	Color of the item	Varchar	255	
dateAdded	Date item was first added to inventory	DateTime		Date and Time format

Returns

Name	Description	Type	Length Max	Range of Values
returnID (PK)	Unique identifier for a return	Int	-	Auto-increment, Unique
itemID (FK)	Foreign key linking to the	Int	-	Foreign Key

	Items table.			
transactionID (FK)	Foreign key linking to the Transactions table.	Int	-	Foreign Key
quantityReturned	Quantity of the item returned	Int	-	
returnDate	Date item was returned	DateTime		Date and Time format

Sales

Name	Description	Type	Length Max	Range of Values
saleID (PK)	Unique identifier for a sale	Int	-	Auto-increment, Unique
itemID (FK)	Foreign key linking to the Items table.	Int	-	Foreign Key
transactionID (FK)	Foreign key linking to the Transactions table.	String	-	Foreign Key
quantitySold	Quantity of the item sold	Int	-	
saleDate	Date item was sold	DateTime		Date and Time format

Users

Name	Description	Type	Length Max	Range of Values
userID (PK)	Unique identifier for a user	Int		Unique
username	Username of the user	Varchar	255	
password	Password of the user	Varchar	255	
role	The role of the user (e.g., admin, employee)	Varchar	255	

Transactions

Name	Description	Type	Length Max	Range of Values
transactionID (PK)	Unique identifier for a transaction	Int	-	Unique
paymentTypeID (FK)	Foreign key linking to the PaymentTypes table.	Int	-	Foreign Key
date	Date of the transaction	DateTime	-	Date and Time format
totalAmount	Total amount of the transaction	Float	-	

Payments

Name	Description	Type	Length Max	Range of Values
paymentID (PK)	Unique identifier for a payment	Int	-	Auto-increment, Unique
paymentTypeID (FK)	Foreign key linking to the PaymentTypes table.	Int	-	Foreign Key
transactionID (FK)	Foreign key linking to the Transaction table	String	-	Foreign Key
amount	The amount of the payment	Float	-	
dateProcessed	Date the payment was processed	DateTime	-	Date and Time format

Payment Types

Name	Description	Type	Length Max	Range of Values
paymentTypeID (PK)	Unique identifier for the type of payment	Int	-	Auto-increment, Unique
paymentMethodName	The name of the payment method	Varchar	255	

Data Management Description:

In our diagram, we have the main **StoreData** entity which contains the primary key storeDataID. This unique identification number helps differentiate between the different stores in each location. The entity creates the data about a given store which helps the employees track the stores sales, what items are selling and need to be restocked, as well as when the information was last updated. The information is then uploaded to the cloud database connected to other store locations.

The StoreData entity takes information from the **transactions** entity which contains a unique identification number for each different transaction that allows employees and customers to keep track of purchases and helps to set similar purchases apart from one another. The transactionID contains information about the date that the transaction took place on, as well as the total amount of said transaction. All this information is then uploaded and saved into the main server's system to track all sales. The paymentTypeID borrows information from the payments and payment types entities that detail the specific information about the payment.

The **Inventory** entity is also utilized by the StoreData which helps track the items that a store has in stock, and what they are out of. Some primary information that each item has is the inventoryID which allows the store as a whole to track the history of certain items to determine the stock of an item as well as the last time the information was updated. This information is important to track as it allows employees to see which items are out of stock and need to be replaced. Each item is then given an itemID, with specific information regarding each item from the items entity.

For the inventory to be properly tracked, it must borrow information from the **items** entity, which contains a unique itemID and assigns specific properties to each item such as size, color, name price, last time the item was updated, and the quantity of each item given its size and color. The items entity is trivial to the inventory class as it provides important information that allows workers to sort and store information about the items in an easily accessible way as well as being able to provide the information to customers.

The **Users** entity manages system access for employees at various levels. Each employee is assigned a unique userID, along with a username and password, enabling them to log into the system. This setup facilitates the execution of specific tasks aligned with the employee's role—such as transaction processing, inventory updates, and accessing sales reports. Roles are clearly defined (e.g., employee, admin) to ensure that employees can only access features necessary for their duties.

Given that payments are an important part of sales, the **payments** entity contains various information that tracks the payment's total amount and the date the payment was made and processed. This is done by giving each payment a unique numerical value stored as the paymentID. This entity also stores information about the payment type laid out in the paymentTypes entity and links the payment with a specific transactionID, making a connection between each payment and each transaction.

The **paymentTypes** entity contains information about each transaction's payment storing an int value for the payment type as well as a string value containing the method of payment called paymentMethodName. This information is valuable to the store as it helps the store track where payments are made from, whether it was made with cash, debit, or credit card as well as the name of the institution. The name of the institution will be of importance as it allows the store to keep track of payments and allows them information in the case of a payment dispute or their systems not working with a certain institution.

With the many, various transactions processed throughout the day for a store, the transactions draw information from the **sales** entity which contains primary information such as the saleID containing the quantity of sales made, as well as the date of those sales. The sales are tracked by linking them with each item's unique itemID, as well as providing information to the transactionID.

When customers wish to return items, the **returns** entity is utilized which creates a unique returnID that allows the system to track which unique items have been returned and prevents double returns. This entity also tracks how many items are made in a certain return exchange as well as the date the return took place. Each return also contains information about the item's unique ID as well as the transaction ID to determine which item was returned, and the initial transaction that took place.

Trade Off Discussion

In our system, we have chosen a SQL database framework over a noSQL option such as MongoDB, guided by our specific demands for solid transactional integrity and the ability to handle complex queries. SQL databases excel in maintaining data integrity and facilitating relationships between data elements, while providing native support for ACID-compliant transactions and an affinity for structured data. These features are essential for the transaction-heavy operations that our system is designed to handle.

While NoSQL databases offer greater flexibility and are adept at handling large volumes of unstructured data, our system does not anticipate the need for such scalability in the foreseeable future. The structured nature of our data, the necessity for detailed queries, and the absence of large-scale changes in data volume steered us towards SQL.

Our design choice also extends to using multiple databases to logically separate concerns—such as separating transaction processing from inventory management—enhancing maintainability and clarity. The ERD and data dictionary serve as integral tools in this design, enabling a clear visualization of data relationships and providing a detailed description of table contents, which complements the structured nature of SQL databases.