Amazon Go

Software Design Description

Mustafa Ozan ALPAY - 2309615
Birkan ARSLAN - 2110252

Spring 2020

Revision History

1	May 15th, 2020	Initial draft
2	June 25th, 2020	Final version

Table of Contents

List of Figures

List of Tables

1 Introduction

This document provides the Software Design Description (SDD) for the smart store which is called as "Amazon Go".

1.1 Purpose

The purpose of this project is to reduce the time spent during the shopping process and reduce labour-based costs for a regular convenience store.

1.2 Scope

- The system will have a mobile app that allows users to login, generate QR codes to enter the store, update payment and billing information, and show previous purchases.
- The system will have a cloud based storage solution that uses Amazon Web Services, which contains the back-end development.
- The system will have a database server that is being used to store user profiles, payment and billing information, and purchases.
- The system will contain an image processing unit that processes data gathered from the in-store cameras to track users' movements inside the store, and properly compute the virtual shopping basket contents.
- The system will contain an embedded systems manager unit that gathers and process data from weight and proximity sensors at each shelf, cameras that scan barcodes and some special recognition tags, QR code scanners to allow customers enter the store, and RFID antennas to recognize items.
- The system will contain a customer behaviour tracker subsystem that scans for the customer's previous purchases to successfully determine who picked which item at moments where face and motion recognition systems fail.
- The system will have a subsystem for payments that directly deducts the amount from the credit card of the user.
- The system will contain an admin panel that allows the system administrators to change system settings and see reports and logs of the current systems to ensure a seamless store.
- The system will have a customer support panel that allows customers to contact to a support service that helps with refunds, returns, and questions.
- The system will have a store employee panel to allow the in-store employees track inventory and make necessary changes.

1.3 Stakeholders and their Concerns

Within the system, there are several different stakeholders:

1. **Users**

   The users are the customers who do or do not have an account on the system. Their main concern is to be able to shop within the store seamlessly. Their other concerns are having their information kept secure, and the ability to reach to the support staff when they need.

2. **System Admins**

   The System Admins' concern is having specific workflows to follow. Without these documents, tasks may get interrupted and the availability of the system might get affected.

3. **Store Employees**

   Store Employees' main concern is to have an easy-to-use and reliable interface for their mobile devices so they can make updates easily within the store.

4. **Support Staff**

   Support Staff has the concern of having a useful interface to follow up with the support tickets efficiently.

5. **Researchers**

   Researchers' concern is to gather data easily and get results from this data to help understanding shopping behaviour of the customers swiftly. Their research may provide helpful insight on the efficiency of the store, and yield results that can improve the overall shopping experience.

2 References

This document is written with respect to the specifications of the document below:

IEEE standard for information technology–systems design–software design descriptions. (2009). New York, NY: Institute of Electrical and Electronics Engineers.

Other Sources:

Amazon Go Store. (n.d.). Retrieved March 1, 2020, from https://www.amazon.com/b/?node=20931384011

Sommerville, I. (2018). *Software Engineering.* Hallbergmoos/Germany: Pearson.

3 Glossary

Term	Definition
New Customer	A user who has not signed up to the system yet.
Registered Customer	A user who has completed the sign-up process for the system.
System Admin	The administrators of the system who has the highest permission level on the system.
Store Employee	Employees who are in charge of managing the inventory and order of the store.
Support Staff	Employees who are in charge of responding to the support tickets that was generated by the users.
AWS	Amazon Web Services (AWS) is a subsidiary of Amazon that provides on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. The system allows auto-scaling to meet the dynamic demand of the customer.
HTTPS	Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP). It is used for secure communication over a computer network. In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS) or, formerly, its predecessor, Secure Sockets Layer (SSL).
QR Code	QR code (abbreviated from 'Quick Response code') is the trademark for a type of matrix barcode (or two-dimensional barcode) first designed in 1994 for the automotive industry in Japan. Currently, QR codes often contain data for a locator, identifier, or tracker that points to a website or application.
API	Application Programming Interface.
SHA512	SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) and first published in 2001.
Java	Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible.

Python	Python is an interpreted, high-level, general-purpose programming language.
Swift	Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc.
Kotlin	Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference.
PostgreSQL	PostgreSQL, also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and technical standards compliance. It is designed to handle a range of workloads, from single machines to data warehouses or Web services with many concurrent users.
TensorFlow	TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.
CRUD	Create, Read, Update and Delete.

Table 1: Glossary

4 Architectural Views

4.1 Context View

This viewpoint provides a context of the system with all actors in general and detailed viewpoints. The Context Diagram provides a general explanation of the actors and their interactions with the system. In the following sections, Use Case Diagrams and detailed explanations for these diagrams can be found.

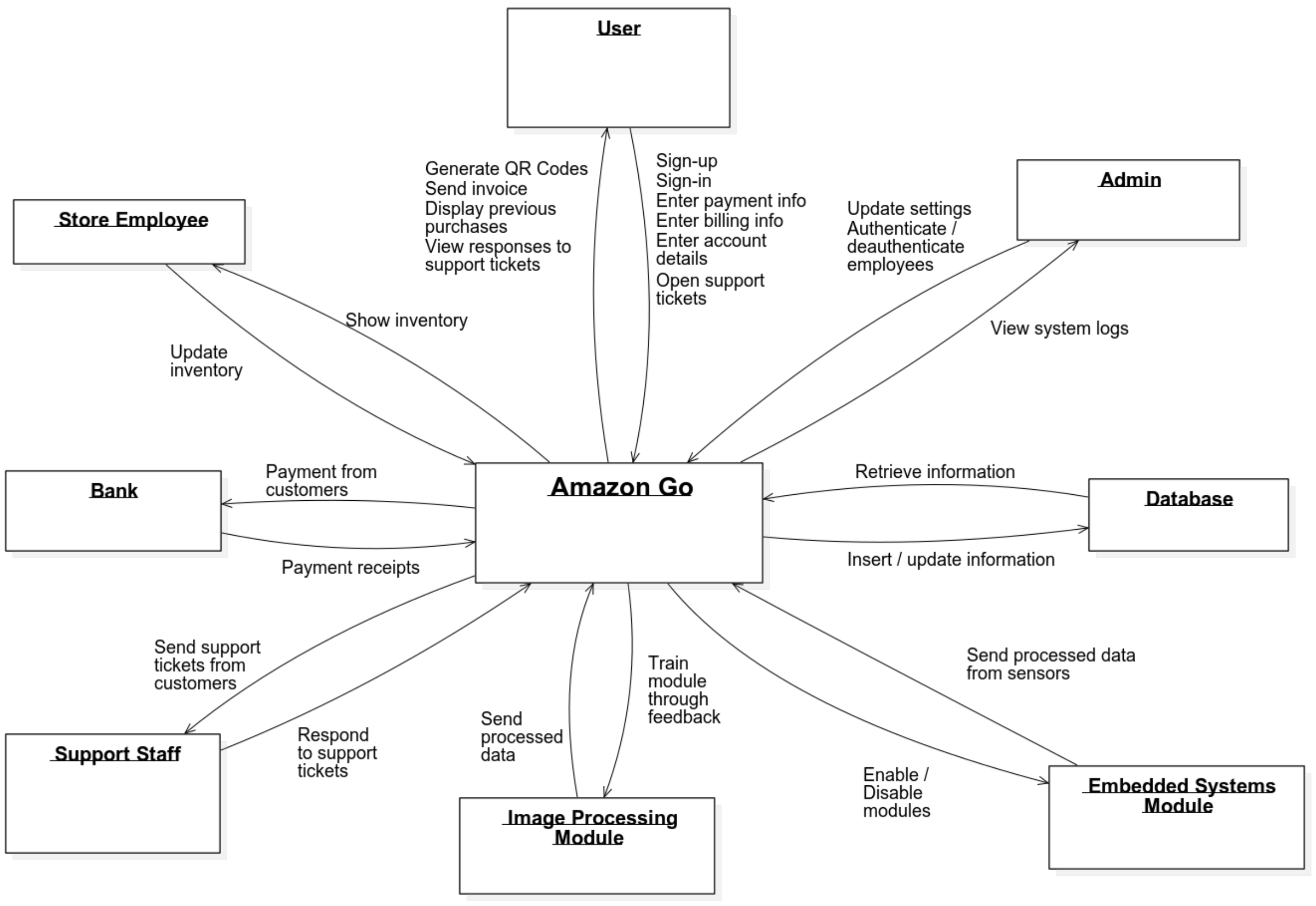


Figure 1: Context Diagram

The use case diagram can be seen at Figure 2.

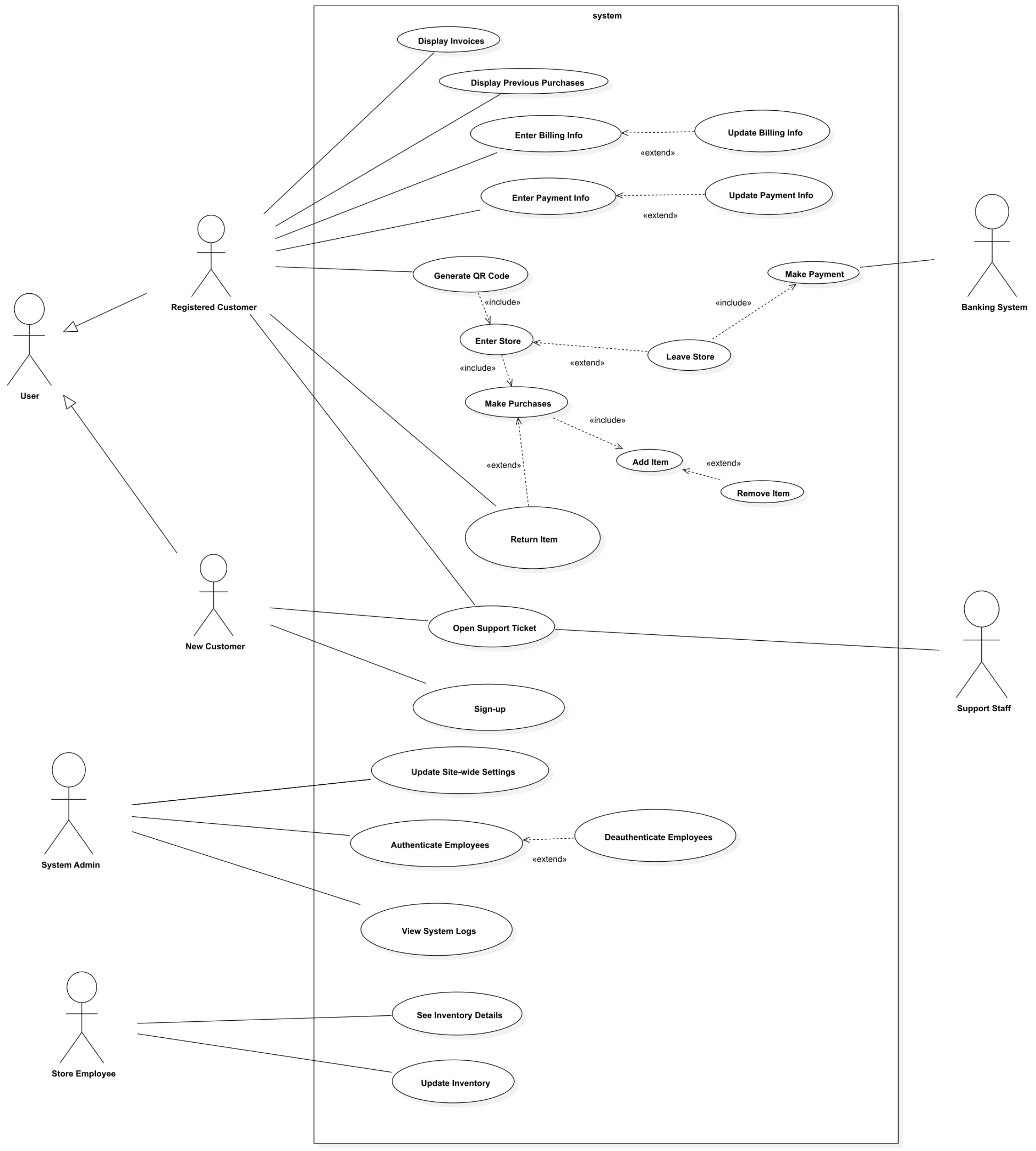


Figure 2: Use Case Diagram

Use Case ID	**1**
Use Case Name	Sign-in
Actors	User
Descriptions	The user signs in to the system in order to use the system.
Preconditions	User exists in the system with valid username, password, and e-mail address.
Postconditions	The user has succesfully logged in.
Normal Flow	1. User clicks on the “sign-in” button. 2. The system prompts the user to enter their username (or e-mail) and password. 3. The user types in their username (or e-mail) and password to the system. 4. System verifies the typed in information. 5. System signs in the verified user.
Exceptions	4. If the entered information is not valid, the system displays “The entered user information is not correct” message. Use case returns on step 2 of normal flow.

Table 2: Use case for Sign-in

Use Case ID	**2**
Use Case Name	Sign-up
Actors	User
Descriptions	The user signs-up to the system in order to use its functionalities.
Preconditions	The user that is not on the system enters their e-mail address, username, password, first name, and last name.
Postconditions	The user has successfully signed up to the system.
Normal Flow	1. User installs the Amazon Go mobile app. 2. User clicks the “Sign-up” Button. 3. User fills in the registration form with their e-mail address, username, password, first name, and last name, and checks the “I agree to the Terms and Conditions” checkbox. 4. System verifies the entered information. 5. User gets validated. 6. User receives an e-mail to activate their account. 7. User clicks the link in their e-mail to activate their account. 8. The account gets activated.
Exceptions	4. If the entered e-mail and/or the username is already in the system, the user will be shown a “The entered e-mail and/or username is already in use” message. Use case returns on step 3 of normal flow. 8. If the user does not click on the activation link, the account never gets activated, therefore the user cannot use their account. Use case returns on step 6 of normal flow.

Table 3: Use case for Sign-up

Use Case ID	**3**
Use Case Name	Enter payment and billing information
Actors	User, Banking System
Descriptions	The user enters their billing and payment information to the system.
Preconditions	The user must be signed up to Amazon Go. The user must be signed in to Amazon Go mobile app. The Amazon Go mobile app is active and running.
Postconditions	The user successfully entered their billing and payment information to the system.
Normal Flow	1. The user clicks on "Payment & Billing Information" button. 2. The user fills in the number, expiry date and CVC2 number of the credit card, as well as the card holders full name. 3. The system verifies the validity of the credit card information using an API call through the Banking System. 4. The credit card gets validated. 5. The user fills in their street address, and tax number. 6. The system validated the address. 7. The billing and payment details gets saved.
Exceptions	4. If the entered credit card information is not valid, the API call from the Banking System returns a `false` value. Use case returns on step 3 of normal flow. 6. If the entered address information is invalid, the user cannot save that information. Use case returns on step 5 of normal flow.

Table 4: Use case for Entering Payment and Billing Information

Use Case ID	**4**
Use Case Name	Update payment and billing information
Actors	User, Banking System
Descriptions	The user updates their billing and payment information in the system.
Preconditions	The user must be signed up to Amazon Go. The user must be signed in to Amazon Go mobile app. The user must be entered their payment and billing information previously. The Amazon Go mobile app is active and running.
Postconditions	The user successfully updated their billing and payment information in the system.
Normal Flow	1. The user clicks on "Payment & Billing Information" button. 2. The user selects the data entry that they want to update. 3. The user fills in the number, expiry date and CVC2 number of the credit card, as well as the card holders full name. 4. The system verifies the validity of the credit card information using an API call through the Banking System. 5. The credit card gets validated. 6. The user fills in their street address, and tax number. 7. The system validated the address. 8. The billing and payment details gets updated.
Exceptions	2. If the user has not entered any information on the system before, no data entry gets displayed. Use case returns on step 1 of normal flow. 5. If the entered credit card information is not valid, the API call from the Banking System returns a `false` value. Use case returns on step 4 of normal flow. 7. If the entered address information is invalid, the user cannot save that information. Use case returns on step 6 of normal flow.

Table 5: Use case for Updating Payment and Billing Information

Use Case ID	**5**
Use Case Name	Enter the store
Actors	User
Descriptions	The user enters an Amazon Go store.
Preconditions	The user must be signed up to Amazon Go. The user must be signed in to Amazon Go mobile app. The user must have their payment and billing info entered to the system. The user must have their mobile phone with Amazon Go mobile app installed on themselves.
Postconditions	The user successfully entered the store.
Normal Flow	1. The user opens the Amazon Go mobile app. 2. The application shows a QR Code. 3. The user shows the QR Code to the turnstile QR Code reader sensor. 4. The turnstile allows the user to enter.
Exceptions	1. The user fails to run the Amazon Go mobile app (due to software or hardware error). Use case returns on step 1 of normal flow. 2. The mobile app fails to generate a valid QR Code. Use case returns on step 2 of normal flow. 3. The turnstile QR Code reader fails to read the QR Code. Use case returns on step 3 of normal flow.

Table 6: Use case for Entering the Store

Use Case ID	**6**
Use Case Name	Buy a product
Actors	User
Descriptions	The user purchases one or more products.
Preconditions	The user must be signed up to Amazon Go. The user must be signed in to Amazon Go mobile app. The user must have their payment and billing info entered to the system. The user must have their mobile phone with Amazon Go mobile app installed on themselves. The user must be entered to the store.
Postconditions	The user has successfully purchased one or more products.
Normal Flow	1. The user picks an item from the shelf. 2. The system adds the item to the users virtual basket. 3. The user walks through the exit turnstile. 4. The system computes the total number of items and automatically deducts the amount from the credit card of the user. 5. An automated invoice gets generated and sent to the user.
Exceptions	1. If the user puts back an item that they have picked earlier (into the related or unrelated shelf), the system removes the item from the users virtual basket. The use case returns on step 0 of normal flow. 4. If the system is unable to deduct the amount from the credit card of the user (in the case of the credit card being maxed out), the user becomes in debt to Amazon and the user will get a warning stating that they should update their payment information as soon as possible. The use case returns on step 3 of normal flow.

Table 7: Use case for Buying a Product

Use Case ID	**7**
Use Case Name	Return a product
Actors	User
Descriptions	The user returns a product to the store that they have purchased earlier.
Preconditions	The user must be signed up to Amazon Go. The user must be signed in to Amazon Go mobile app. The Amazon Go mobile app is active and running. The user must have made a purchase.
Postconditions	The user successfully returned an item that they have purchased earlier to Amazon.
Normal Flow	1. The user views their related purchase through the "Previous Purchases" button. 2. The user clicks the "Return item(s)" button. 3. The user selects the item(s) that they want to return. 4. The user clicks on the "Approve" button. 5. The system generates a QR Code. 6. The user scans the QR Code to the return lockers of the store. 7. The system will unlock one of the available lockers door. 8. The user puts the item(s) that they want to return to the locker and shuts the door. 9. The system validates the action. 10. The user gets a refund.

Exceptions	5. The mobile app fails to generate a valid QR Code. Use case returns on step 4 of normal flow. 6. The QR Code reader fails to read the QR Code. Use case returns on step 5 of normal flow. 7. The system fails to unlock a locker door due to either mechanical error or not having an empty locker. The mobile app displays "The return system is currently unavailable, please try again later." message. Use case returns on step 6 of normal flow. 8. The locker door fails to lock. The mobile app displays "The return system is currently unavailable, please try again later." message. Use case returns on step 7 of normal flow. 9. The system detects that the user did not put the items that they wanted to return, or put irrelevant items to the locker. The user gets a warning. Use case returns on step 8 of normal flow. 10. The banking system fails to make a refund. The user gets notified and selects either getting Amazon Credits instead or using the Banking System later again. Use case returns on step 9 of normal flow.

Table 8: Use case for Returning a Product

Use Case ID	**8**
Use Case Name	Display previous purchases
Actors	User
Descriptions	The user displays their previous purchases through the Amazon Go mobile app.
Preconditions	The user must be signed up to Amazon Go. The user must be signed in to Amazon Go mobile app. The Amazon Go mobile app is active and running.
Postconditions	The user views their previous purchases.
Normal Flow	1. The user clicks on “Previous Purchases” button. 2. The mobile app displays a list of previous purchases as clickable objects that allows the details of the previous purchases to be seen.
Exceptions	2. If the user has no previous purchases, the system displays a message with “You have no previous purchases” text. The system returns on step 1 of normal flow.

Table 9: Use case for Displaying Previous Purchases

Use Case ID	**9**
Use Case Name	Display invoices
Actors	User
Descriptions	The user views their invoices.
Preconditions	The user must be signed up to Amazon Go. The user must be signed in to Amazon Go mobile app. The Amazon Go mobile app is active and running. The user is on the “Previous Purchases” page. The user has made at least one purchase.
Postconditions	The user views their invoices related to their purchases.
Normal Flow	1. The user clicks on the purchase that they want to view the invoice of. 2. The mobile app displays the invoice and allows a PDF version to be downloaded via the button “Download Invoice”.
Exceptions	None

Table 10: Use case for Displaying Invoices

Use Case ID	**10**
Use Case Name	Open a support ticket
Actors	User, Support Staff
Descriptions	The user opens a support ticket in order to communicate with the Support Staff.
Preconditions	The Amazon Go mobile app is active and running.
Postconditions	The user successfully opened a support ticket.
Normal Flow	1. The user clicks on the button "Support". 2. If the user has already signed-in, the user enters the subject and message fields. If the user is a non-signed-up user, the user must fill their full name, and e-mail. 3. The system verifies entered information. 4. The support ticket gets opened with a unique ticket ID. 5. The ticket ID gets shown to the user.
Exceptions	3. If the entered information is not valid, the system displays an error message. Use case returns on step 2 of normal flow. 4. If the system fails to generate a unique ID, the support ticket cannot get generated. The system displays an error message. Use case returns on step 3 of normal flow.

Table 11: Use case for Opening a Support Ticket

Use Case ID	**11**
Use Case Name	Viewing inventory details
Actors	Store Employee
Descriptions	The store employee views inventory details.
Preconditions	The store employee must be authenticated. The store employee must be signed-in to the system.
Postconditions	The inventory details have been viewed successfully by the store employee.
Normal Flow	1. The store employee clicks the "Inventory Details" button. 2. The store employee scans the items barcode, or QR code. 3. The inventory details of the product gets displayed.
Exceptions	2. If the barcode or the QR code is not visible or readable, the system will give a warning and request the store employee to type in the name of the product. Use case returns on step 1 of normal flow. 3. If the entered information is invalid, no inventory detail can be displayed. The store employee will receive an error message. Use case returns on step 2 of normal flow.

Table 12: Use case for Viewing Inventory Details

Use Case ID	**12**
Use Case Name	Update Inventory Details
Actors	Store Employee
Descriptions	The store employee updates inventory details.
Preconditions	The store employee must be authenticated. The store employee must be signed-in to the system. The inventory display screen of the product is active on the mobile device of the employee.
Postconditions	The inventory details have been updated successfully by the store employee.
Normal Flow	1. The store employee clicks on "Update Inventory" button. 2. The enters the new quantity of the product. 3. The store employee gets asked to confirm the action. 4. The inventory of the product gets updated.
Exceptions	2. If the quantity area is left empty, the system will give a warning and request the store employee to fill a valid number in the area. Use case returns on step 1 of normal flow. 3. If the employee does not confirm the new quantity, the system will not update the inventory. Use case returns on step 2 of normal flow.

Table 13: Use case for Updating Inventory Details

Use Case ID	**13**
Use Case Name	View support tickets
Actors	Support Staff
Descriptions	The support staff views open support tickets.
Preconditions	The support staff must be signed-in to the system.
Postconditions	The support tickets have been successfully viewed.
Normal Flow	1. The support staff clicks on the button “Support Tickets”. 2. The system displays support tickets that are currently open.
Exceptions	2. If there are no support tickets that are open, the system will display a message stating that. Use case returns on step 1 of normal flow.

Table 14: Use case for Viewing Support Tickets

Use Case ID	**14**
Use Case Name	Authenticate employees
Actors	System Admin, Store Employee
Descriptions	The System Admin authenticates a new employee to use the system.
Preconditions	The employee must be signed up to the system. The system admin must be signed in to the management panel.
Postconditions	The employee has successfully authenticated.
Normal Flow	1. The system admin clicks on "Authenticate Employees" button. 2. The system admin selects the store that the employee should be authenticated to. 3. The system admin approves the selection. 4. The system checks the entered information. 5. The employee has successfully authenticated for the aforementioned store.
Exceptions	4. If the system admin does not fill all the required information, the system will display an error message. Use case returns on step 3 of normal flow.

Table 15: Use case for Authenticating Employees

Use Case ID	**15**
Use Case Name	Update Site-Wide Settings
Actors	System Admin
Descriptions	The System Admin updates Site-Wide Settings of the system.
Preconditions	The system admin must be signed in to the management panel.
Postconditions	Site-Wide Settings have been updated.
Normal Flow	1. The system admin clicks on "Update Site-Wide Settings" button. 2. The system admin selects the setting that needs to be updated. 3. The system admin makes the necessary change(s) on the previously selected field. 4. The system admin approves the change. 5. Site-Wide Settings have been successfully updated.
Exceptions	2. If the System Admin does not select a setting to be updated, the system will display an error message. Use case returns on step 1 of normal flow. 3. If the system admin does not make any change on the field(s), the system will display an error message. Use case returns on step 3 of normal flow. 4. If the System Admin does not approve the changes, the system will display an error message. Use case returns on step 4 of normal flow.

Table 16: Use case for Updating Site-Wide Settings

4.2 Composition View

This viewpoint provides the components of the system from a broad perspective. Detailed information regarding these systems are available in the following sections.

4.2.1 Component Diagram

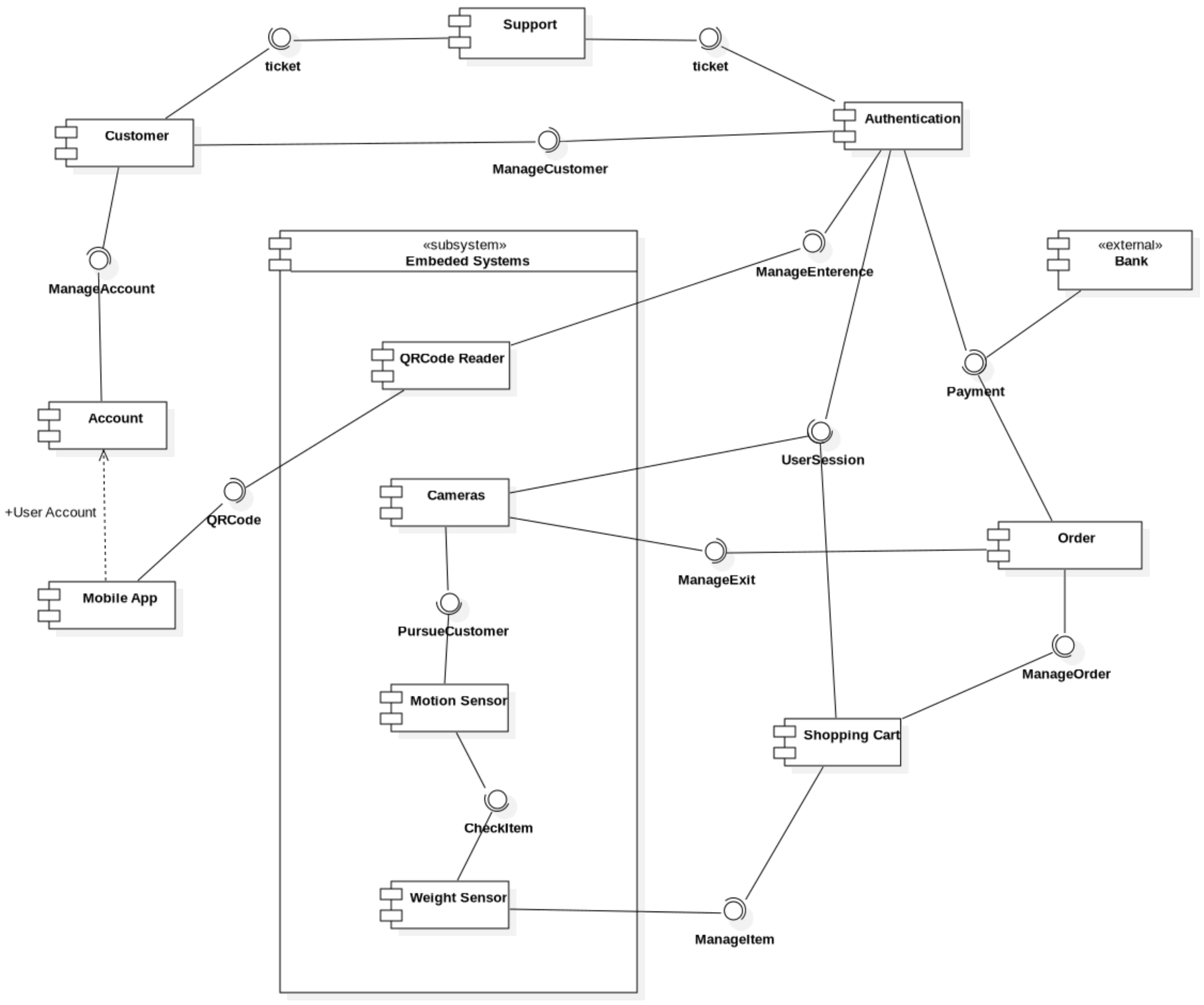


Figure 3: Component Diagram

Design Rationale

1. Creating a subsystem to handle all embedded systems responses was preferred to ensure all embedded systems work in sync and any response from the aforementioned systems gets provided when the data is processed.

2. The embedded systems subsystem is responsible for following a customer inside the store, checking if they have taken an item or put it back, checking the fullness of the shelves to ensure each item in inventory can be delivered to customers, and allowing a customer enter or exit the store using their authentication via the QR Code.

3. Other parts of the system are not grouped into bigger subsystems since they require constant communication with each other and grouping them into a bigger subsystem does not have a clear benefit.

4. The Mobile App component, which depends on the Account component, allows the user to enter the store by providing a valid QR Code to the Embedded Systems subsystem.

5. The Customer component allows a customer to sign in to their account to do any shopping or non-shopping activities.

6. The Authentication component communicates with Customer, Order, Shopping Cart and Embedded Systems subsystem components. By communicating with the Embedded Systems, it allows a customer to enter or leave the store. By communicating with the Order component, it marks an order as complete through the bank.

4.2.2 Deployment Diagram

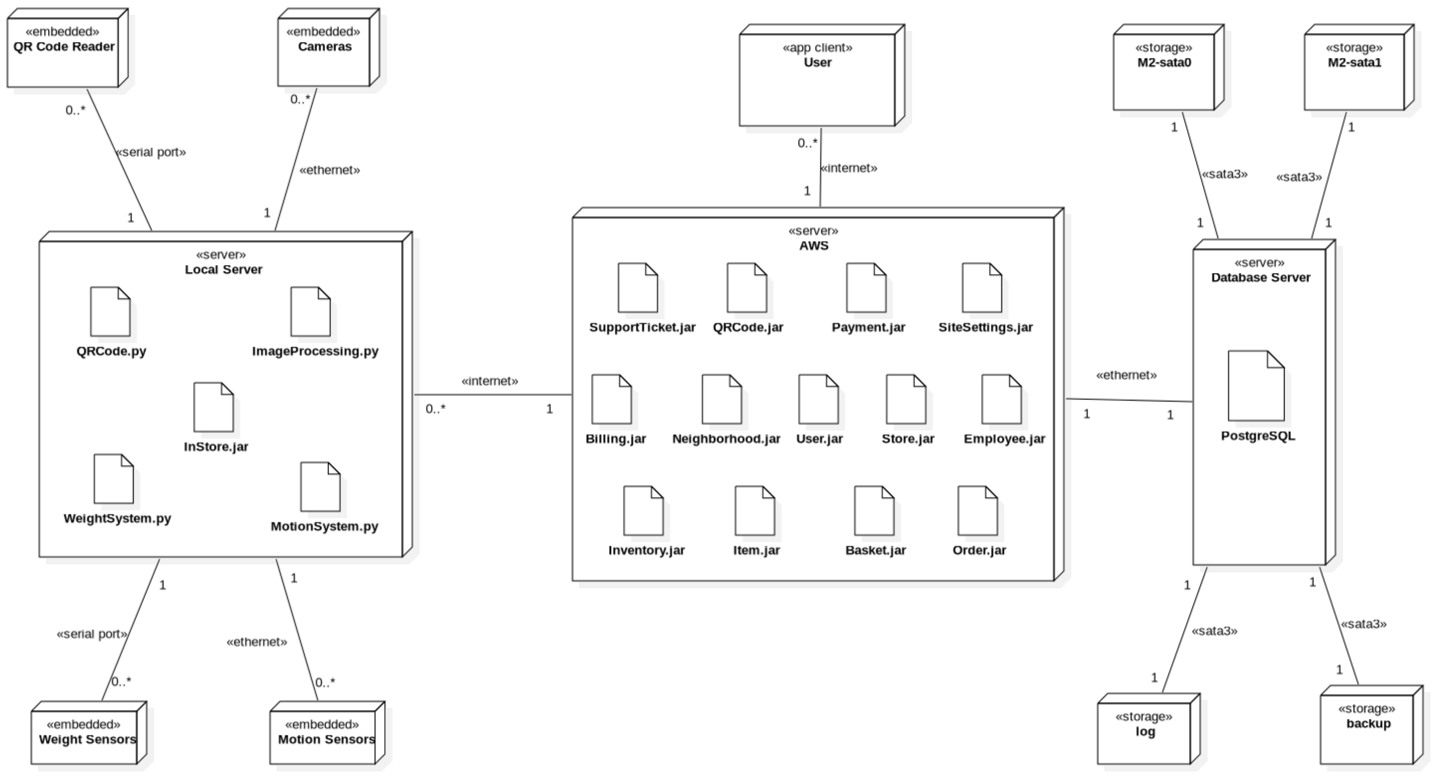


Figure 4: Deployment Diagram

Design Rationale

1. Due to the high data flow from the embedded systems, especially the cameras that track customer movement and activity, an in-store server is needed to process this data and communicate with the main server using this processed data to ensure swiftness.

2. Each store must have their own local server to handle such computing jobs, namely reading the QR Codes, measuring the weight on the shelves, tracking motion around the shelves, and user activity within the store using the cameras. All of these sub-tasks can be done via Python libraries.

3. The result from the aforementioned sub-tasks communicate with the in-store main process, which collects data and processes it before communicating with the global server.

4. The cameras and the motion sensors communicate with the local server using ethernet connection that is installed within the store. The QR Code readers and the weight sensors communicate directly using serial port, since they are embedded devices rather than separate devices that can run on their own.

5. The communication between the local server and the global server is made via internet connection.

6. Customers who use the mobile app is connected to the main server using the specified endpoints by internet.

7. The system uses PostgreSQL as the database solution.

8. The database server uses 2 M2-SSD devices as RAID to ensure redundancy and high speed.

9. The database server uses another M2-SSD for logs, and another M2-SSD for backup purposes.

10. All connection between the storage devices and the database is made using SATA3 connection.

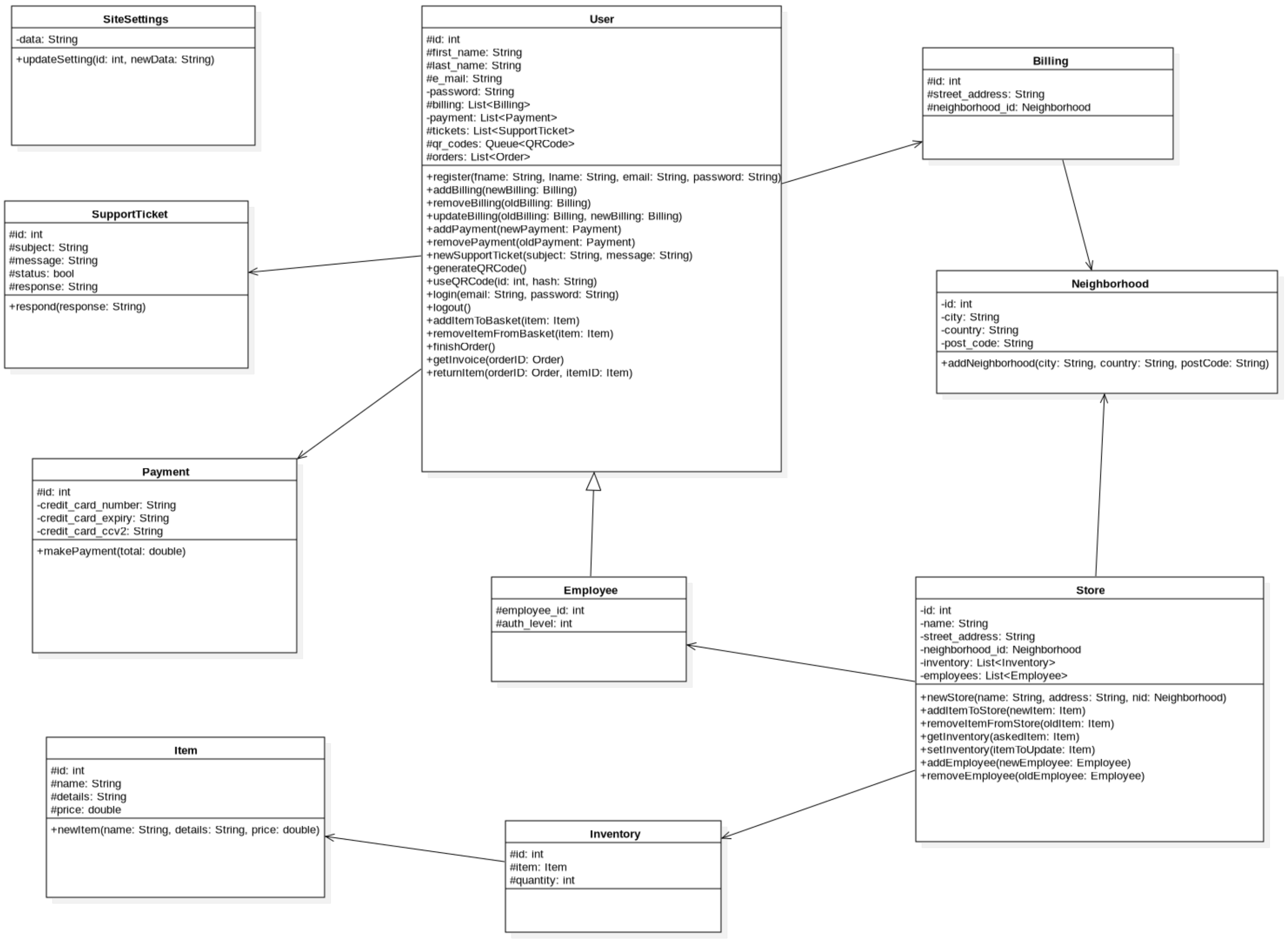
11. Any internet communication is protected via SSL.

4.3 Information View

This view provides an overview of the organization and relation of the data that will be stored in the system while providing the methods related to them. Additionally, any effect of the aforementioned methods will be explored in terms of Create, Read, Update and Delete (CRUD) operations.

4.3.1 Class Diagram

The Interface Class Diagram can be seen at Figure 5. Descriptions of each class method can be seen at Table 17, and design of these methods can be seen at Table 18. To provide a better understanding of the whole system, the full class diagram of the system is also available at Figure 6. To ease readability, most of the generic getter-setter methods and all private methods have been removed from the class diagram while creating the Interface Class Diagram.



SiteSettings
-data: String
+updateSetting(id: int, newData: String)
User
#id: int
#first_name: String
#last_name: String
#e_mail: String
-password: String
#billing: List<Billing>
-payment: List<Payment>
#tickets: List<SupportTicket>
#qr_codes: Queue<QRCode>
#orders: List<Order>
+register(fname: String, lname: String, email: String, password: String)
+addBilling(newBilling: Billing)
+removeBilling(oldBilling: Billing)
+updateBilling(oldBilling: Billing, newBilling: Billing)
+addPayment(newPayment: Payment)
+removePayment(oldPayment: Payment)
+newSupportTicket(subject: String, message: String)
+generateQRCode()
+useQRCode(id: int, hash: String)
+login(email: String, password: String)
+logout()
+addItemToBasket(item: Item)
+removeItemFromBasket(item: Item)
+finishOrder()
+getInvoice(orderID: Order)
+returnItem(orderID: Order, itemID: Item)
Billing
#id: int
#street_address: String
#neighborhood_id: Neighborhood
SupportTicket
#id: int
#subject: String
#message: String
#status: bool
#response: String
+respond(response: String)
Neighborhood
-id: int
-city: String
-country: String
-post_code: String
+addNeighborhood(city: String, country: String, postCode: String)
Payment
#id: int
-credit_card_number: String
-credit_card_expiry: String
-credit_card_ccv2: String
+makePayment(total: double)
Employee
#employee_id: int
#auth_level: int
Store
-id: int
-name: String
-street_address: String
-neighborhood_id: Neighborhood
-inventory: List<Inventory>
-employees: List<Employee>
+newStore(name: String, address: String, nid: Neighborhood)
+addItemToStore(newItem: Item)
+removeItemFromStore(oldItem: Item)
+getInventory(askedItem: Item)
+setInventory(itemToUpdate: Item)
+addEmployee(newEmployee: Employee)
+removeEmployee(oldEmployee: Employee)
Item
#id: int
#name: String
#details: String
#price: double
+newItem(name: String, details: String, price: double)
Inventory
#id: int
#item: Item
#quantity: int

Figure 5: Interface Class Diagram

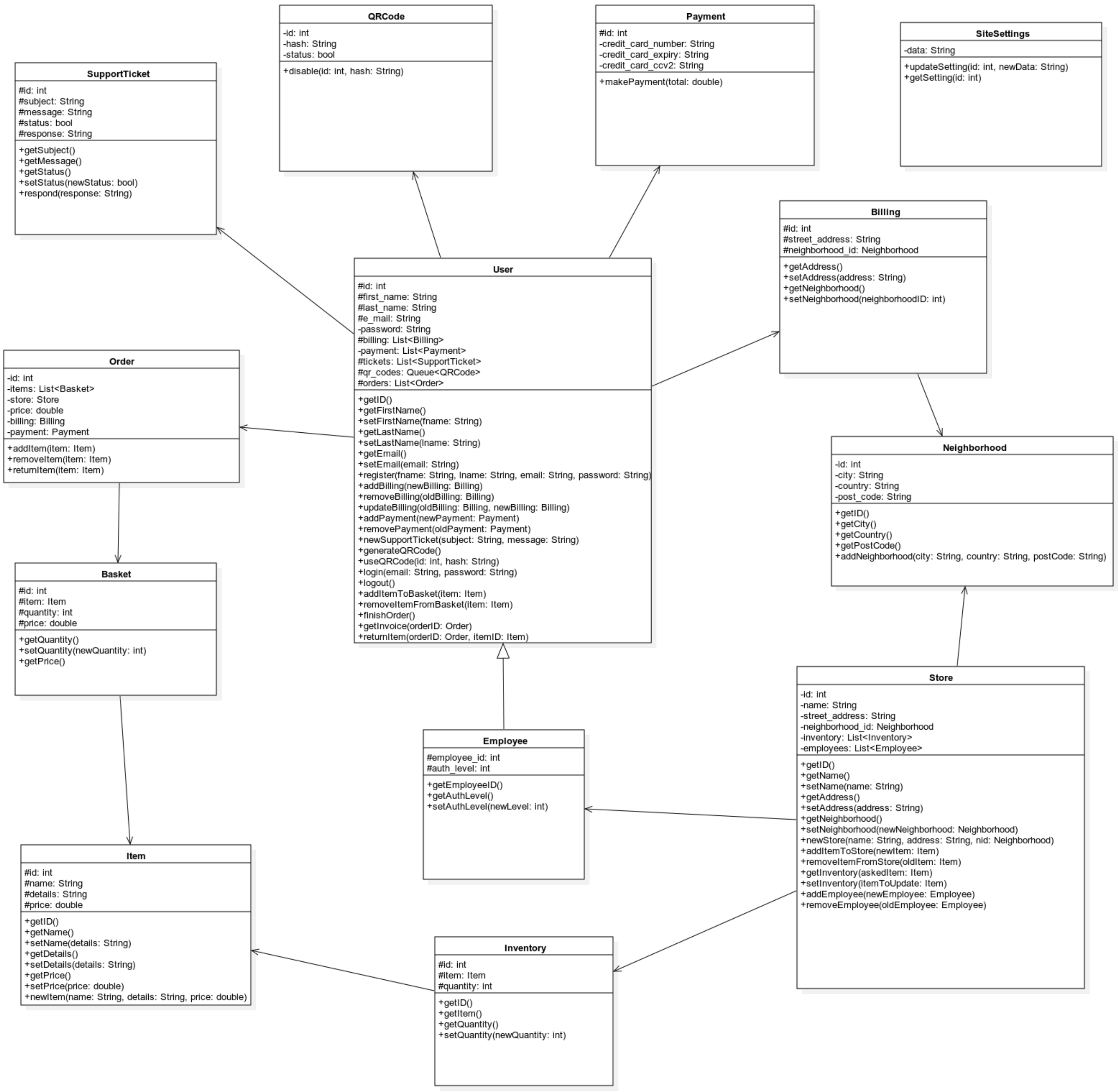

QRCode
-id: int
-hash: String
-status: bool
+disable(id: int, hash: String)
Payment
#id: int
-credit_card_number: String
-credit_card_expiry: String
-credit_card_ccv2: String
+makePayment(total: double)
SiteSettings
-data: String
+updateSetting(id: int, newData: String)
+getSetting(id: int)
SupportTicket
#id: int
#subject: String
#message: String
#status: bool
#response: String
+getSubject()
+getMessage()
+getStatus()
+setStatus(newStatus: bool)
+respond(response: String)
Billing
#id: int
#street_address: String
#neighborhood_id: Neighborhood
+getAddress()
+setAddress(address: String)
+getNeighborhood()
+setNeighborhood(neighborhoodID: int)
User
#id: int
#first_name: String
#last_name: String
#e_mail: String
-password: String
#billing: List<Billing>
-payment: List<Payment>
#tickets: List<SupportTicket>
#qr_codes: Queue<QRCode>
#orders: List<Order>
+getID()
+getFirstName()
+setFirstName(fname: String)
+getLastName()
+setLastName(lname: String)
+getEmail()
+setEmail(email: String)
+register(fname: String, lname: String, email: String, password: String)
+addBilling(newBilling: Billing)
+removeBilling(oldBilling: Billing)
+updateBilling(oldBilling: Billing, newBilling: Billing)
+addPayment(newPayment: Payment)
+removePayment(oldPayment: Payment)
+newSupportTicket(subject: String, message: String)
+generateQRCode()
+useQRCode(id: int, hash: String)
+login(email: String, password: String)
+logout()
+addItemToBasket(item: Item)
+removeItemFromBasket(item: Item)
+finishOrder()
+getInvoice(orderID: Order)
+returnItem(orderID: Order, itemID: Item)
Order
-id: int
-items: List<Basket>
-store: Store
-price: double
-billing: Billing
-payment: Payment
+addItem(item: Item)
+removeItem(item: Item)
+returnItem(item: Item)
Neighborhood
-id: int
-city: String
-country: String
-post_code: String
+getID()
+getCity()
+getCountry()
+getPostCode()
+addNeighborhood(city: String, country: String, postCode: String)
Basket
#id: int
#item: Item
#quantity: int
#price: double
+getQuantity()
+setQuantity(newQuantity: int)
+getPrice()
Store
-id: int
-name: String
-street_address: String
-neighborhood_id: Neighborhood
-inventory: List<Inventory>
-employees: List<Employee>
+getID()
+getName()
+setName(name: String)
+getAddress()
+setAddress(address: String)
+getNeighborhood()
+setNeighborhood(newNeighborhood: Neighborhood)
+newStore(name: String, address: String, nid: Neighborhood)
+addItemToStore(newItem: Item)
+removeItemFromStore(oldItem: Item)
+getInventory(askedItem: Item)
+setInventory(itemToUpdate: Item)
+addEmployee(newEmployee: Employee)
+removeEmployee(oldEmployee: Employee)
Employee
#employee_id: int
#auth_level: int
+getEmployeeID()
+getAuthLevel()
+setAuthLevel(newLevel: int)
Item
#id: int
#name: String
#details: String
#price: double
+getID()
+getName()
+setName(details: String)
+getDetails()
+setDetails(details: String)
+getPrice()
+setPrice(price: double)
+newItem(name: String, details: String, price: double)
Inventory
#id: int
#item: Item
#quantity: int
+getID()
+getItem()
+getQuantity()
+setQuantity(newQuantity: int)

Figure 6: Class Diagram of the System

Operation	**Description**
register	A user is gets registered to the system.
addBilling	The given billing information is added to the user.
removeBilling	The given billing information is removed from the user.
updateBilling	The billing information is updated to reflect the given one.
addPayment	The given payment information is added to the user.
removePayment	The given payment information is removed from the user.
newSupportTicket	A new support ticket is saved on the system.
generateQRCode	A new QR Code gets generated for the user.
useQRCode	The user uses a QR Code and the code gets marked as disabled.
login	The user logs-in to the system.
logout	The user logs-out from the system.
addItemToBasket	The user adds an item to their basket.
removeItemFromBasket	The user removes an item from their basket.
finishOrder	An order has completed.
getInvoice	The user receives the invoice for the order.
returnItem	The user returns an item.
respond	A support staff responds to a support ticket.
makePayment	A payment gets done after an order.
addNeighborhood	A new neighborhood gets added to the system.
newStore	A new store gets added to the system.
addItemToStore	An item gets added to the store.
removeItemFromStore	An item gets removed from the store.
getInventory	The current inventory of the asked item gets fetched.
setInventory	The current inventory of the item gets set.

addEmployee	A new employee gets added to the store.
removeEmployee	An existing employee gets removed from a store.
newItem	A new item gets added to the system.
updateSetting	A site-wide setting gets updated.

Table 17: Operation Descriptions

Operation	**Inputs**	**Outputs**	**Exceptions**
register	- fname - lname - email - password	`UserID` if operation is successful, `0` otherwise	- A user with the provided e-mail already exists - Database server is not available
addBilling	- newBilling	`BillingID` if operation is successful, `0` otherwise	- Database server is not available
removeBilling	- oldBilling	`True` if operation is successful, `False` otherwise	- The billing information to remove is not valid - Database server is not available
updateBilling	- oldBilling - newBilling	`True` if operation is successful, `False` otherwise	- The billing information to update does not exist - Database server is not available
addPayment	- newPayment	`PaymentID` if operation is successful, `0` otherwise	- The payment information is not valid - Database server is not available

removePayment	- oldPayment	`True` if operation is successful, `False` otherwise	- The payment information to remove does not exist - Database server is not available
newSupportTicket	- subject - message	`TicketID` if operation is successful, `0` otherwise	- Subject or field is empty - Database server is not available
generateQRCode		`QRCodeID` if operation is successful, `0` otherwise	- Database server is not available
useQRCode	- id - hash	`True` if operation is successful, `False` otherwise	- The QR Code is not valid - Database server is not available
login	- email - password	`UserID` if operation is successful, `0` otherwise	- E-mail and password is not valid - Database server is not available
logout		`True` if operation is successful, `False` otherwise	- User has not logged in - Database server is not available
addItemToBasket	- item	`True` if operation is successful, `False` otherwise	- Item is not valid - Database server is not available
removeItemFromBasket	- item	`True` if operation is successful, `False` otherwise	- The item is not in the basket of the user - Database server is not available

finishOrder		`OrderID` if operation is successful, `0` otherwise	- An active order has not found - Database server is not available
getInvoice	- orderID	`True` if operation is successful, `False` otherwise	- The order ID is not valid - Database server is not available
returnItem	- orderID - itemID	`True` if operation is successful, `False` otherwise	- The order ID is not valid - The item with the given item ID is not in the order - Database server is not available
respond	- response	`True` if operation is successful, `False` otherwise	- The response field is empty - Database server is not available
makePayment	- total	`True` if operation is successful, `False` otherwise	- The payment amount is not valid - Database server is not available
addNeighborhood	- city - country - postCode	`NeighborhoodID` if operation is successful, `0` otherwise	- The neighborhood information is not valid - Database server is not available
newStore	- name - address - nid	`StoreID` if operation is successful, `0` otherwise	- Name and/or address field(s) are empty - Neighborhood ID is not valid - Database server is not available

addItemToStore	- newItem	`True` if operation is successful, `False` otherwise	- Item is not valid - Database server is not available
removeItemFromStore	- oldItem	`True` if operation is successful, `False` otherwise	- Item is not valid - Item does not exist in the store - Database server is not available
getInventory	- askedItem	`Inventory` if operation is successful, `NULL` otherwise	- Item is not valid - Database server is not available
setInventory	- itemToUpdate	`True` if operation is successful, `False` otherwise	- Item is not valid - Entered inventory is not valid - Database server is not available
addEmployee	- newEmployee	`EmployeeID` if operation is successful, `0` otherwise	- Employee data is not valid - Database server is not available
removeEmployee	- oldEmployee	`True` if operation is successful, `False` otherwise	- Employee data is not valid - Employee is not assigned to the store previously - Database server is not available
newItem	- name - details - price	`ItemID` if operation is successful, `0` otherwise	- Name, details and/or price is not valid - Database server is not available

updateSetting	- id - newData	`True` if operation is successful, `False` otherwise	- The ID of the setting and/or the new data is not valid - Database server is not available

Table 18: Operation Design

Design Rationale

1. The `User` class is the main class that handles all the data integrity of the system. This class references other classes to store data as needed.

2. All the actions that can be made by a `User` make necessary calls to other methods from the `User` class.

3. To communicate with the banking system, payments happen inside another class, namely `Payment`, which has the duty to process the payment and get response from the bank.

4. The embedded systems inside the store that track the inventory works with the `Inventory` class only.

5. In-store employees can take actions that affect `Item`, `Inventory` classes. However they do not have direct access to these interfaces, they can only communicate with the `Store` class with limited authentication level.

6. Support Staff can only communicate with `SupportTicket` class.

7. The System Admins can communicate with every interface if needed.

4.3.2 Database Operations

The Entity-Relationship Model of the system can be seen at Figure 7. Detailed CRUD operations information can be found at Table 19.

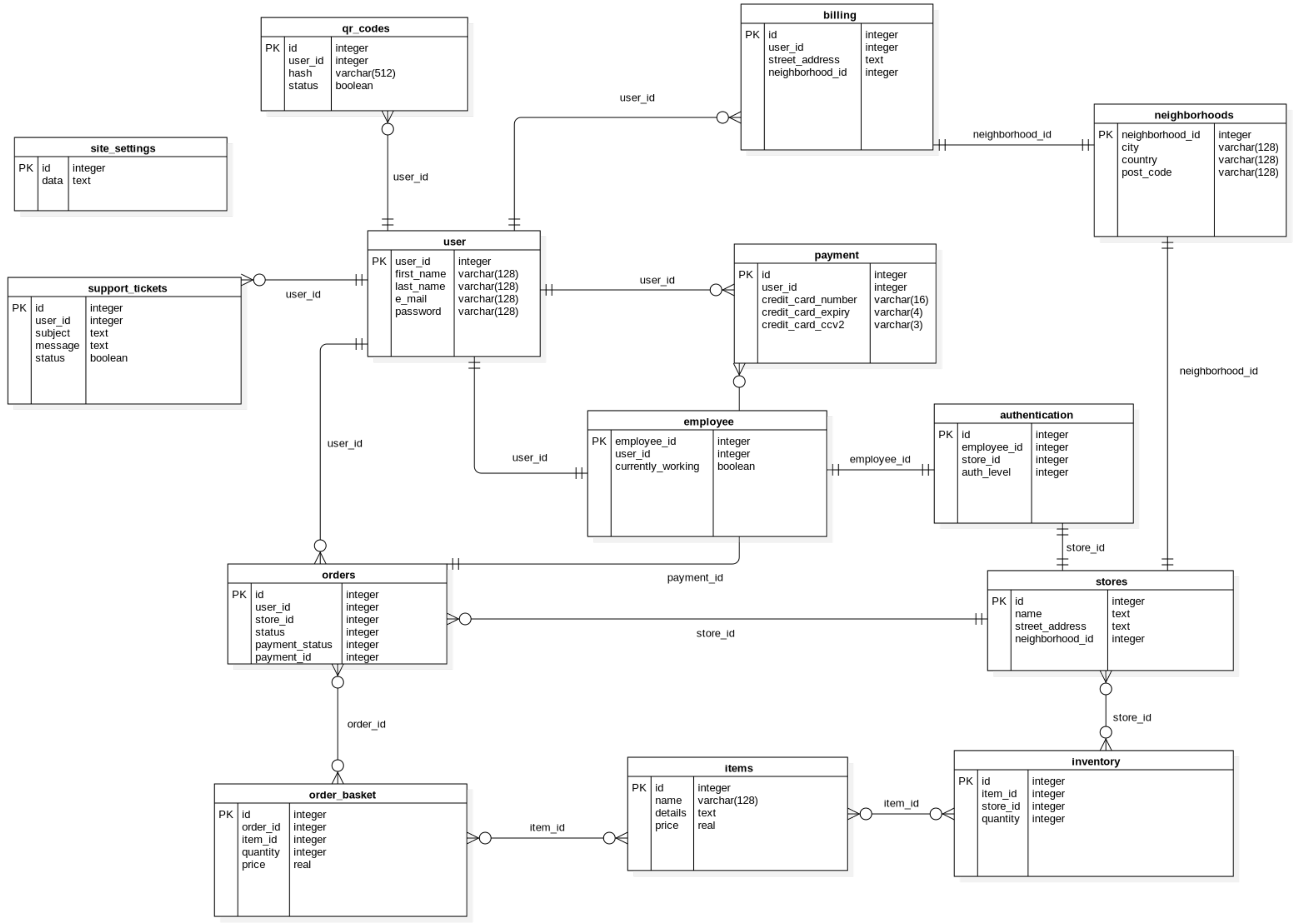


Figure 7: The Entity-Relationship Model for the Database of the System

Operation	**CRUD Operations**

register	CREATE: `user`
	READ:
	UPDATE:
	DELETE:

addBilling	CREATE: `billing`
	READ: `neighborhoods`
	UPDATE:
	DELETE:

removeBilling	CREATE:
	READ:
	UPDATE:
	DELETE: `billing`

updateBilling	CREATE:
	READ: `neighborhood`
	UPDATE: `billing`
	DELETE:

addPayment	CREATE: `payment`
	READ:
	UPDATE:
	DELETE:

removePayment	CREATE:
	READ:
	UPDATE:
	DELETE: `payment`

newSupportTicket	CREATE: `support_tickets`
	READ:
	UPDATE:
	DELETE:

generateQRCode	CREATE: `qr_codes`
	READ:
	UPDATE:
	DELETE:

useQRCode	CREATE:
	READ:
	UPDATE: `qr_codes`
	DELETE:

login	CREATE:
	READ: `user`
	UPDATE:
	DELETE:

logout	CREATE:
	READ: `user`
	UPDATE:
	DELETE:

addItemToBasket	CREATE: `order_basket`
	READ: `items`
	UPDATE: `inventory`
	DELETE:

removeItemFromBasket	CREATE: `order_basket`
	READ: `items`
	UPDATE: `inventory`
	DELETE:

finishOrder	CREATE:
	READ:
	UPDATE: `orders`
	DELETE:

getInvoice	CREATE:
	READ: `orders`
	UPDATE:
	DELETE:

returnItem	CREATE:
	READ: `orders`, `payment`
	UPDATE: `order_basket`, `inventory`
	DELETE:

respond	CREATE: `support_tickets`
	READ:
	UPDATE: `support_tickets`
	DELETE:

makePayment	CREATE:
	READ: `payment`
	UPDATE: `orders`
	DELETE:

addNeighborhood	CREATE: `neighborhoods`
	READ:
	UPDATE:
	DELETE:

newStore	CREATE: `stores`
	READ: `neighborhoods`
	UPDATE:
	DELETE:

addItemToStore	CREATE: `inventory`
	READ: `items`, `store`
	UPDATE:
	DELETE:

removeItemFromStore	CREATE:
	READ: `items`, `store`
	UPDATE:
	DELETE: `inventory`

getInventory	CREATE:
	READ: `inventory`
	UPDATE:
	DELETE:

setInventory	CREATE:
	READ:
	UPDATE: `inventory`
	DELETE:

addEmployee	CREATE: `employee`, `authentication`
	READ: `user`, `stores`
	UPDATE:
	DELETE:

removeEmployee	CREATE:
	READ: `user`, `stores`
	UPDATE:
	DELETE: `authentication`, `employee`

newItem	CREATE: `items`
	READ:
	UPDATE:
	DELETE:

updateSetting	CREATE:
	READ:
	UPDATE: `site_settings`
	DELETE:

Table 19: CRUD Operations

Design Rationale

1. The integrity of the database is provided by the usage of Foreign Keys when necessary.
2. Repetition of data is avoided by using referencing frequently used data, such as user ID's, neighbourhood, et cetera.
3. PostgreSQL will be used for database management.
4. Each database query will be run with a `try - catch` block to allow exception handling.

4.4 Interface View

This view provides an overview of communication between different internal interfaces and external interfaces of Amazon Go.

4.4.1 Internal Interfaces

The Interface Between the Embedded Systems and the Database Server

The embedded systems constantly provide data regarding the items and the customers in the store. When a customer takes an item from the shelf, or puts it back, the inventory of this item must be updated. This is achieved by the interface between the embedded systems and the database server. The interaction between these two interfaces can be seen at Figure 8.

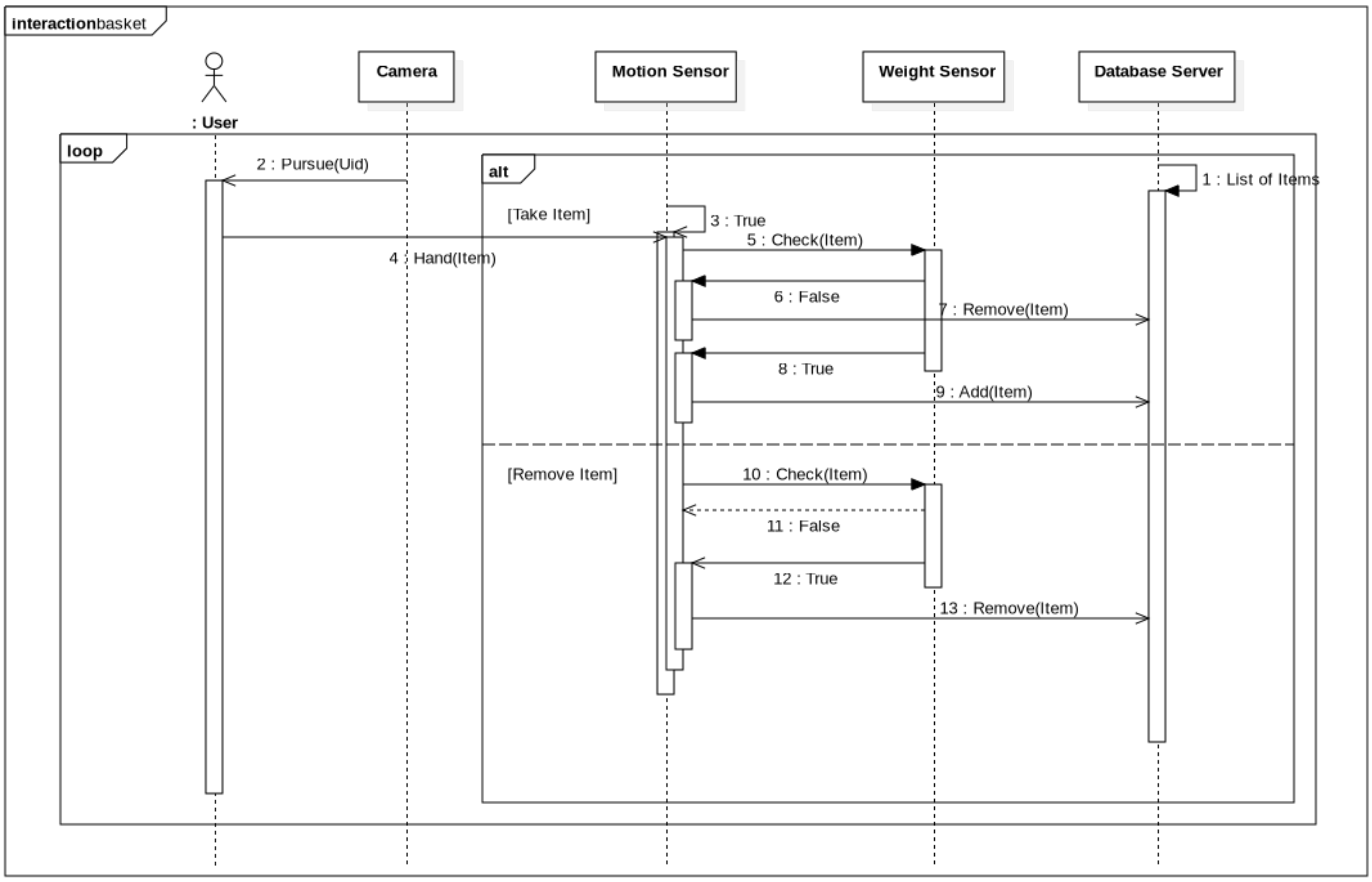


Figure 8: Sequence Diagram showing the interface between the Embedded Systems and the Database Server while adding and/or removing an item to/from the basket

Design Rationale

1. The embedded systems works as a subsystem with different components. Each member of the subsystem communicate with each other to ensure reliability. In case of data mismatch, the majority decision taken as the final decision. This provides an N-version programming aspect to ensure the customer is not charged wrongfully.

2. With constant updating of the inventory details through the database, the in-store inventory check system gets notified immediately if something runs out of stock. This provides fast response to out-of-stock situations.

The Interface Between the Mobile App and the Database Server

The Mobile App needs to communicate with the Database Server to ensure several key functionalities, such as sign-up, log-in, generate a QR Code, display previous purchases, et cetera. In case of any communication error with the database, the mobile app cannot run, and it displays error messages.

Design Rationale

1. The Mobile App has a local NoSQL database to store personal data locally, however this database is not connected to the main Database Server. It is used as a medium to store user data and let the application run smoothly.
2. When the Mobile App gets run for the first time, after log-in, the local NoSQL database gets filled with non-sensitive personal data to ensure the app can run even when the user does not have any internet connection.
3. Whenever the user accesses the Mobile App's regions that require up-to-date information, or any part that require sensitive information, an API call gets made to the database server and required information gets fetched to get displayed to the user.

The Interface Between the Order System and the Database Server

The Order system allows the orders to get processed and saved inside the system. This system has connections with the Inventory system due to the nature of the shopping experience. Whenever a customer places an order, this requires a set of actions in an order, namely: entering the store, adding items to the basket, leaving the store and making payment, and receiving an invoice. All these actions require communication with the database to allow the actions getting recorded within the system. Sequence diagram at Figure 9 shows an example run of the Order System.

Design Rationale

1. The Embedded Systems provide necessary data and makes the updates on the Database regularly. This has been explained prior. With these updates, other parts of the Update system gets triggered.
2. When the customer proceeds through the exit turnstiles, several actions gets triggered, namely: calculation of the basket, getting an authentication for the payment, and generating an invoice.
3. Calculation of the basket requires the current list of items that the customer has for the active session. This data then used to calculate the total sum of the active session.

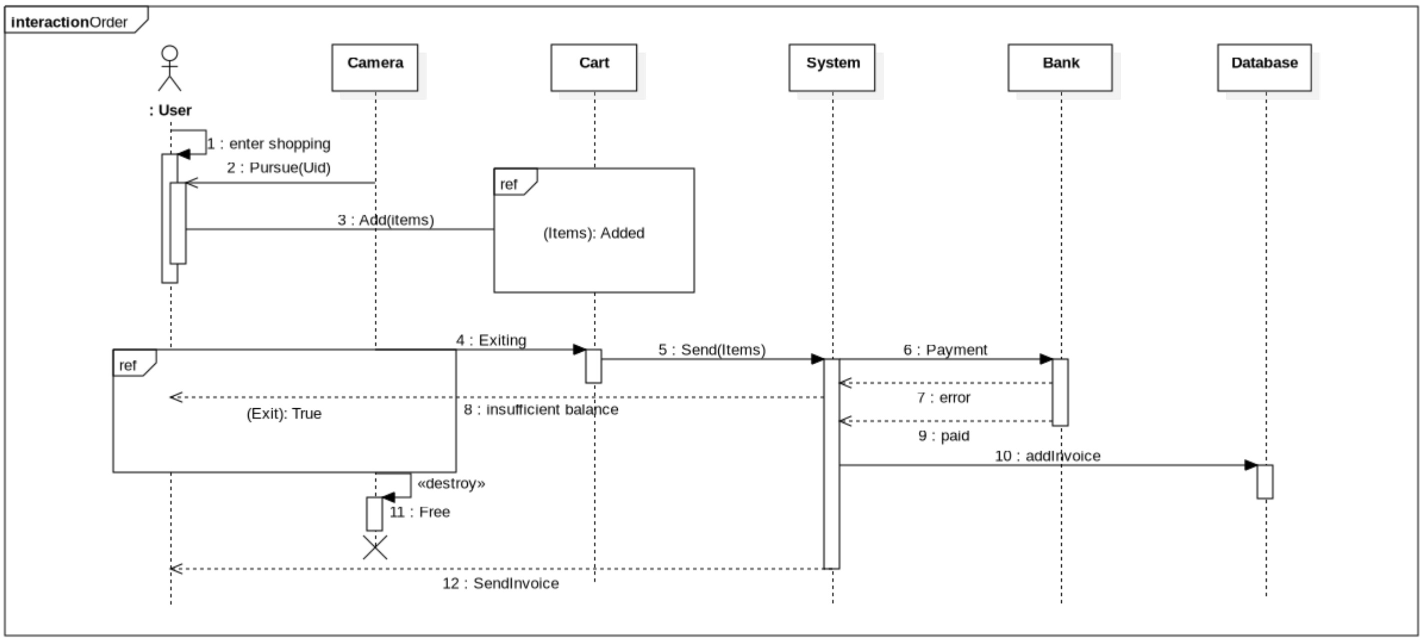


Figure 9: Sequence Diagram showing the interface between the Order System and the Database Server while making an order

4. The total sum of the Order gets sent to the Banking system using the stored payment information in the database. After receiving the authentication from the bank, the invoice gets generated by the items in the order.

5. The invoice is saved in the database and gets sent to the user.

The Interface Between the Payment System and the Database Server

This interface allows the authentication of the payments by using the stored payment information within the database. This is the most sensitive part of the database and the system, since any mistake within this system may cause colossal problems.

Design Rationale

1. The Payment System must be the only system within the whole Amazon Go system that can access to the payment information. This is required not only to allow security, but also to track any monetary actions.

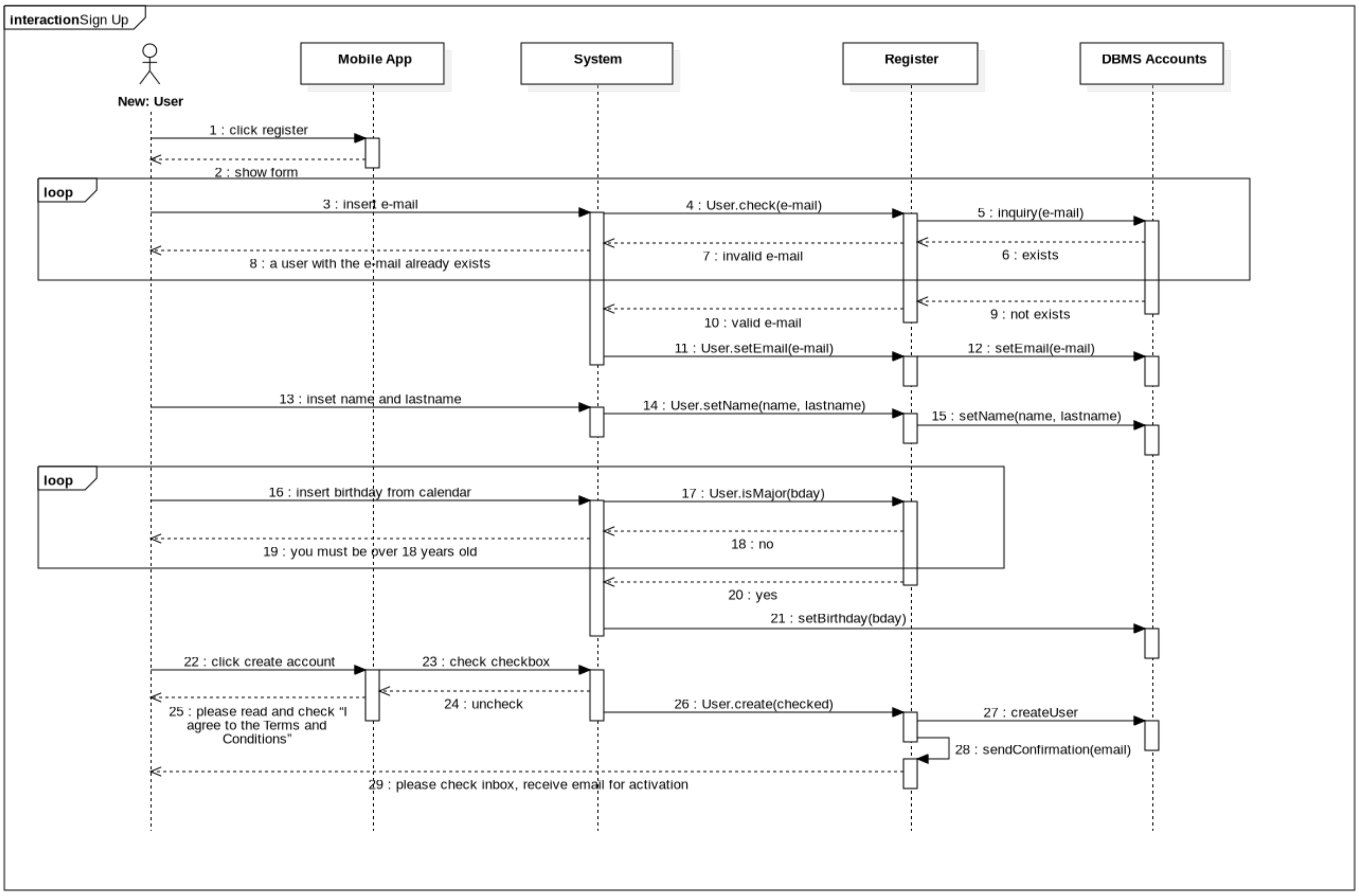
2. The Payment System can be triggered by only a limited set of methods, and these are: making a payment while exiting the store, and making a return.

4.4.2 External Interfaces

4.4.2.1 User Interfaces

Customer Interface

This interface is the main interface that the users of the Mobile App see. It allows registration, log-in, entering the store, saving and displaying user data, displaying previous purchases and many other functionalities. An excerpt from the sign-in screen can be seen at Figure 11a. A sample QR Code to enter the store can be seen at Figure 11b. A sequence diagram showing the sign-up process of a new customer can be seen at Figure 10.



interactionSign Up
New: User
Mobile App
System
Register
DBMS Accounts
1 : click register
2 : show form
loop
3 : insert e-mail
4 : User.check(e-mail)
5 : inquiry(e-mail)
6 : exists
7 : invalid e-mail
8 : a user with the e-mail already exists
9 : not exists
10 : valid e-mail
11 : User.setEmail(e-mail)
12 : setEmail(e-mail)
13 : inset name and lastname
14 : User.setName(name, lastname)
15 : setName(name, lastname)
loop
16 : insert birthday from calendar
17 : User.isMajor(bday)
18 : no
19 : you must be over 18 years old
20 : yes
21 : setBirthday(bday)
22 : click create account
23 : check checkbox
24 : uncheck
25 : please read and check "I agree to the Terms and Conditions"
26 : User.create(checked)
27 : createUser
28 : sendConfirmation(email)
29 : please check inbox, receive email for activation

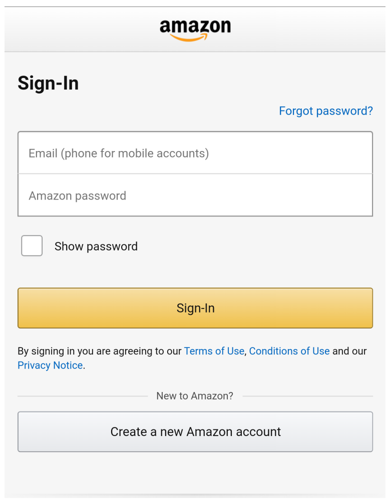
Figure 10: Sequence Diagram showing the Sign-Up process of the User Interface

Design Rationale

1. The Mobile App is designed in a manner to allow the users to use it without getting confused and lost while trying to do any tasks related to the Amazon Go experience.

2. The general design of this interface is kept as simple as possible to achieve a pleasant user experience while not undermining effectiveness.

3. By using native development environments of Android and iOS, many functionalities regarding the operating systems of the mobile devices kept intact and the user experiences a smooth,


amazon

Sign-In

Forgot password?

Email (phone for mobile accounts)

Amazon password

Show password

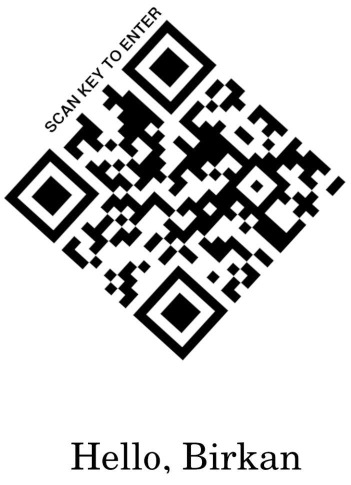
Sign-In

By signing in you are agreeing to our Terms of Use, Conditions of Use and our Privacy Notice.

New to Amazon?

Create a new Amazon account

(a) Sign-in screen of the application



(b) Sample QR Code to enter the store

Figure 11: Graphical User Interface of the Mobile App

natural-feeling user experience.

Store Interface

The Store Interface is used to add or modify (if authentication level is adequate) Amazon Go stores. It also provides necessary end points to display current employees, inventory details, address information.

Design Rationale

1. This interface may be used by in-store employees and higher management. Therefore it is designed as a simple interface that does not require any technical knowledge to navigate through.

2. If the user has modification permissions, their login has a secondary level authentication via a USB dongle to ensure security.

Neighborhood Interface

This interface is used to add a new neighborhood to the system. It is not used frequently but it provides a crucial section to mark address information correctly within the system. This interface can be accessed by any user who can access any interface that has an address field.

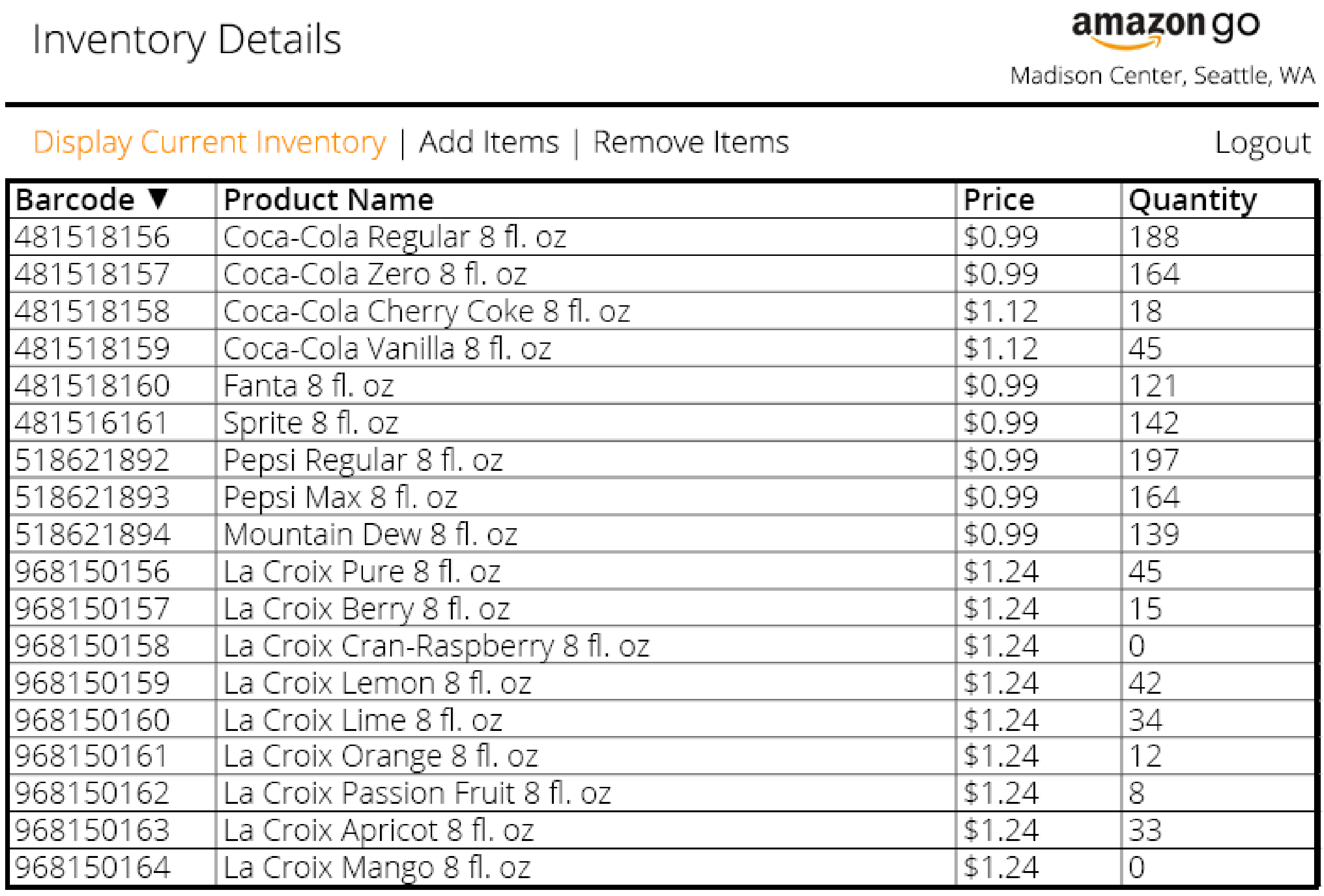
Design Rationale

1. It is designed as a very simple interface to allow any user without any technical skill to add a new neighborhood to the system.

2. It also supports location gathering via devices that has in-built location tracking.

Inventory Interface

This interface allows the in-store employees to track and update inventory details of their assigned store. Since it is used on mobile devices, it is built in a responsive manner using HTML5 to allow flexibility. A mock-up of the interface can be seen at Figure 12. A sequence diagram to update inventory can be seen at Figure 13.



Inventory Details

amazon go

Madison Center, Seattle, WA

Display Current Inventory | Add Items | Remove Items

Logout

Barcode ▼	Product Name	Price	Quantity
481518156	Coca-Cola Regular 8 fl. oz	$0.99	188
481518157	Coca-Cola Zero 8 fl. oz	$0.99	164
481518158	Coca-Cola Cherry Coke 8 fl. oz	$1.12	18
481518159	Coca-Cola Vanilla 8 fl. oz	$1.12	45
481518160	Fanta 8 fl. oz	$0.99	121
481516161	Sprite 8 fl. oz	$0.99	142
518621892	Pepsi Regular 8 fl. oz	$0.99	197
518621893	Pepsi Max 8 fl. oz	$0.99	164
518621894	Mountain Dew 8 fl. oz	$0.99	139
968150156	La Croix Pure 8 fl. oz	$1.24	45
968150157	La Croix Berry 8 fl. oz	$1.24	15
968150158	La Croix Cran-Raspberry 8 fl. oz	$1.24	0
968150159	La Croix Lemon 8 fl. oz	$1.24	42
968150160	La Croix Lime 8 fl. oz	$1.24	34
968150161	La Croix Orange 8 fl. oz	$1.24	12
968150162	La Croix Passion Fruit 8 fl. oz	$1.24	8
968150163	La Croix Apricot 8 fl. oz	$1.24	33
968150164	La Croix Mango 8 fl. oz	$1.24	0

Figure 12: In-store Employee Inventory Interface Mock-up

Design Rationale

1. Since it is intended to be used on mobile devices, this interface must be accessible via browsers at any resolution and device. Thus, a responsive design is required.

2. The interface is designed using HTML5 technology to ensure availability on any mobile device while reducing production cost.

3. This interface limits access to in-store network to improve security, while requiring a valid username and password for an employee as the first security measure.

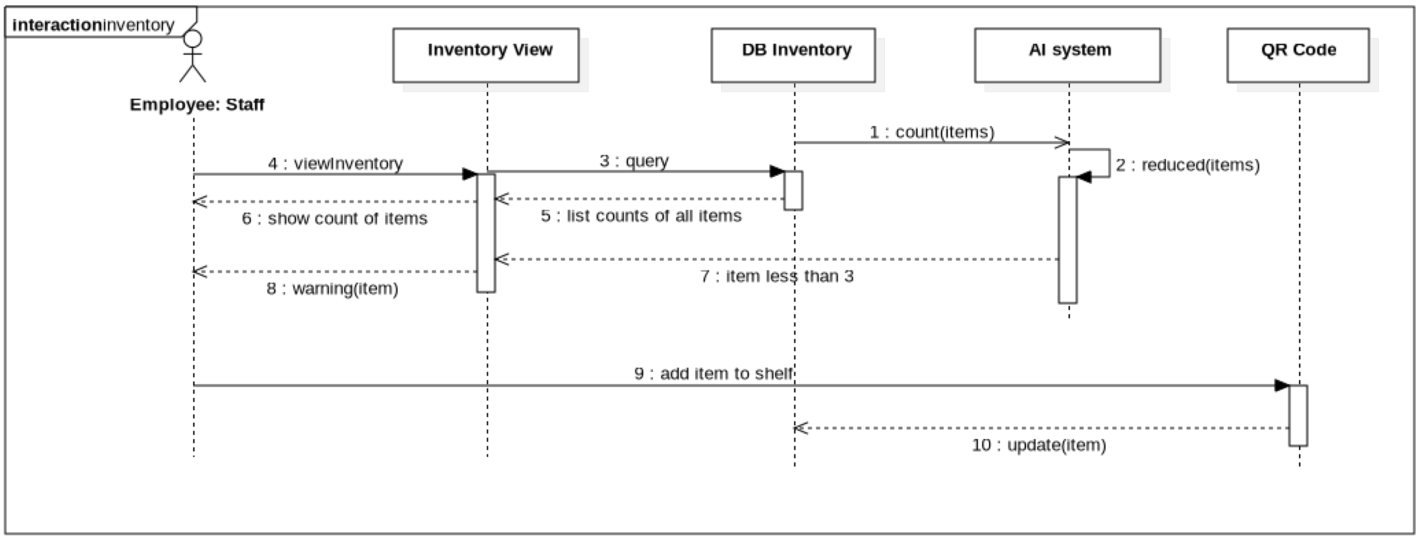


Figure 13: Sequence Diagram showing the inventory update process of the Inventory Interface

Researcher Interface

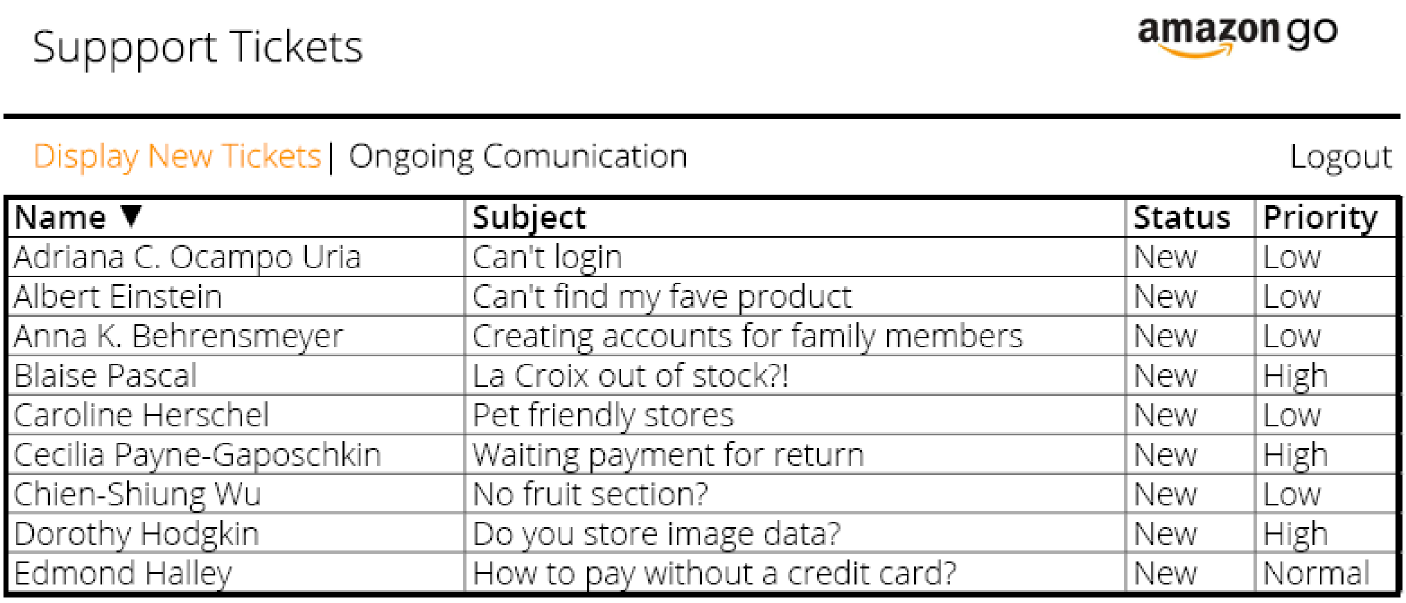
This interface allows the researchers to investigate user activity to determine human behaviour in a humanless store. Since the store is one-of-a-kind, any research in the field of behavioral science is missing, and the researchers employed by Amazon Go need access to the customer activity data within the store to conclude their research.

Design Rationale

1. The researchers only need to access data, therefore this interface only allows read access from the database for very specific fields.
2. This interface allows anonymized data to be downloaded so it can be imported into other statistical tools to analyse further.
3. The raw data from embedded systems and the computed data from the ML library can be viewed and downloaded, depending on the requirements.
4. Researchers cannot access to any personal information regarding the customers to ensure protection of personal data with respect to legal regulations.

Support Staff Interface

This interface allows the Support Staff to review and respond to support tickets. Since it is used by people who do not require any technical skills, it is made as a very simple web page using HTML5 technology. A mock-up design for this interface at the Support Staff end can be seen at Figure 14. A sequence diagram to responding and finalizing a support ticket can be seen at Figure 15.



Name ▼	Subject	Status	Priority
Adriana C. Ocampo Uria	Can't login	New	Low
Albert Einstein	Can't find my fave product	New	Low
Anna K. Behrensmeyer	Creating accounts for family members	New	Low
Blaise Pascal	La Croix out of stock?!	New	High
Caroline Herschel	Pet friendly stores	New	Low
Cecilia Payne-Gaposchkin	Waiting payment for return	New	High
Chien-Shiung Wu	No fruit section?	New	Low
Dorothy Hodgkin	Do you store image data?	New	High
Edmond Halley	How to pay without a credit card?	New	Normal

Figure 14: Support Staff Interface Mock-up

Design Rationale

1. This interface is made using HTML5 with a responsive design to allow the support staff use it easily on their computers, tablets and other devices.
2. It provides an option to follow up on ongoing communication to allow support staff to resolve any stuck support request by sending reminders to the customers.
3. The support staff has very limited information that they can view, namely the order that the ticket is related to, and basic personal information of the customer.

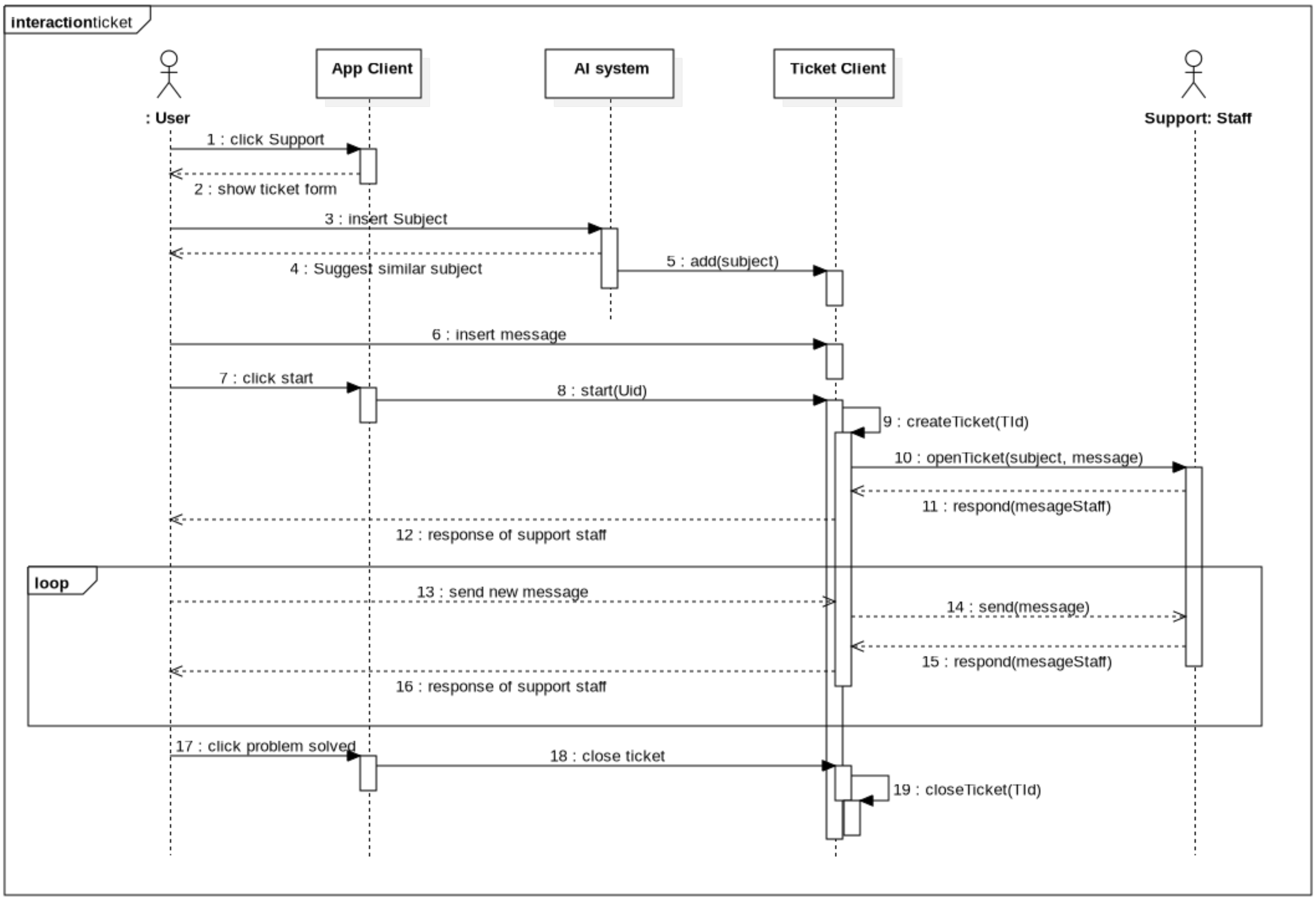


Figure 15: Sequence Diagram showing the support ticket response process of the Support Staff Interface

Site Admin Interface

The Site Admin Interface is the main interface that is used by the system admins to manage the system effectively. In this interface, the admins can update site settings, authenticate or deauthenticate employees, view system logs, and view system status. This interface extends the functionalities of the Store, Neighborhood, Inventory, Researcher and Support Staff interfaces. A sample mock-up design of authenticate and deauthenticate employees interface can be seen at Figure 16.

Design Rationale

1. The Site Admin interface allows sensitive operations to be performed on the system. In other words, this interface has authentication to perform CRUD operations fully.

2. Due to the sensitive nature of this interface, accessing to the Site Admin Interface requires a second authentication step following the username and password based login, which is achieved by a USB dongle. This is necessary to block any unauthorized access in case of password or username leaks.

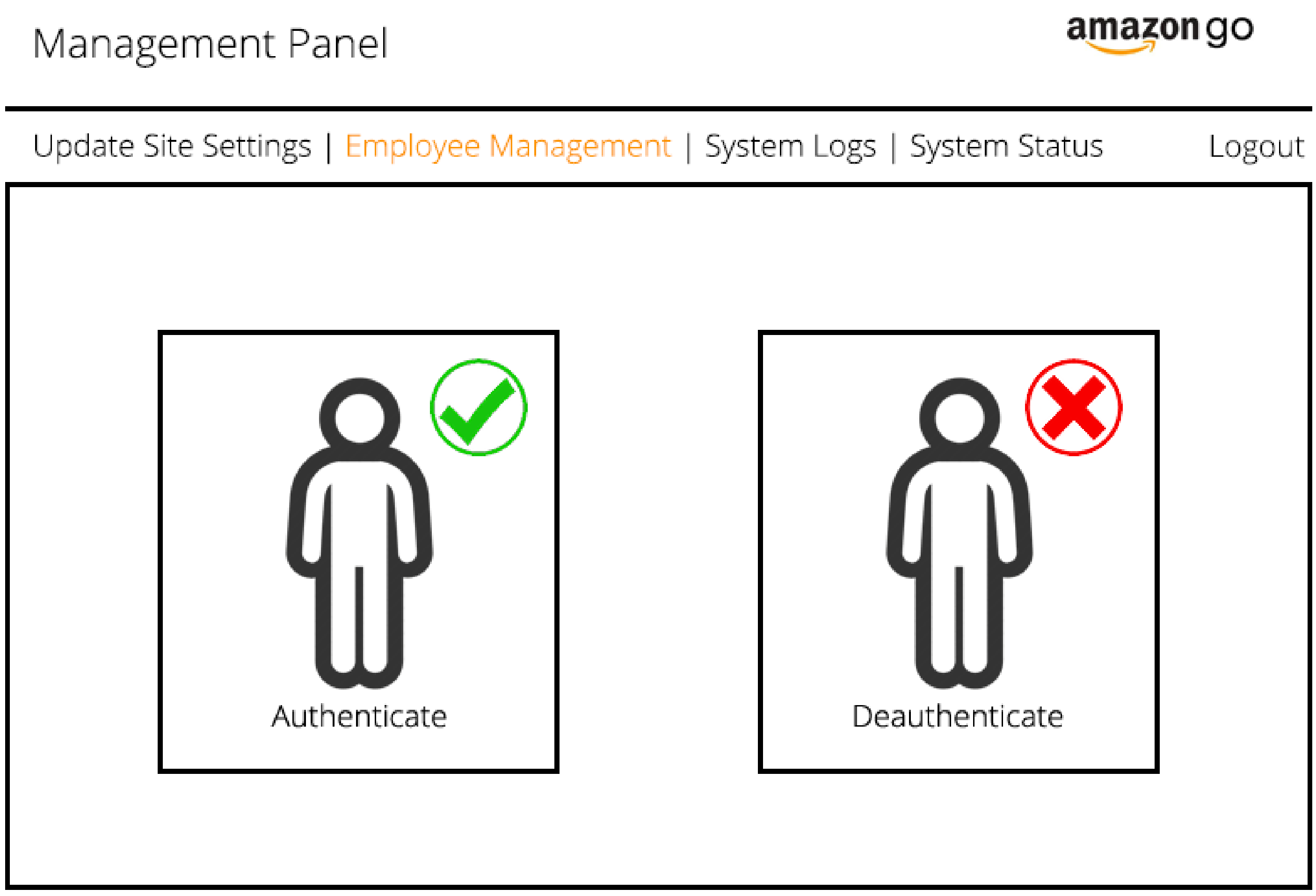

Management Panel
amazon go
Update Site Settings | Employee Management | System Logs | System Status
Logout
Authenticate
Deauthenticate

Figure 16: System Admin Interface Mock-up

4.4.2.2 System Interfaces

The Interface Between the Payment System and the Bank

This interface provides a secure communication layer initiated by the Payment System through the Banking System. It allows the payment transaction to conclude and the amount of the order to be deducted from the credit card of the user.

Design Rationale

1. The communication between the Payment System and the Bank is concluded using secure, encrypted, state-of-the-art communication channels using TCP with SSL protocols.

2. To ensure secure communication and avoid man-in-the-middle attacks, the data that is transferred between two channels are encrypted by using PGP (Pretty Good Privacy) keys.

The Interface Between the Admin System and the Cloud Hosting Server

The communication between the Admin System and the Cloud Hosting Server allows the system to scale according to the needs and demand. It is also used when an admin changes a site-wide setting.

Design Rationale

1. Since the servers hosted using Amazon Web Services cloud, there is not a local server that the system admins can go and update if needed. This lead to the need of creating an admin interface that communicates with the servers to reflect any changes.

2. Any communication between these two interfaces is protected by PGP and TCP with SSL.

The Interface Between the Order System and the E-mail Service

The Order System uses the E-mail Service for notification based tasks, namely: when a customer concludes an order, when a return has been processed, when a support ticket has been responded to, et cetera. The communication between these two interfaces are maintained by this system interface.

Design Rationale

1. The system is hosted on a cloud based hosting, in order not to overwhelm the servers, external e-mail services are being used. This allows less maintenance, more server power to process the store-related tasks, and easier configuration.

2. Any communication for outgoing e-mail happens via TCP with SSL protocol. The order system provides an e-mail object, which gets translated into an outgoing e-mail and then gets sent to the user.

3. If the e-mail service does not respond or responds as `False`, the e-mail waits in the queue at the Order System and gets re-sent every 30 minutes until the process succeeds. This allows ensuring the delivery of the notifications via e-mail.