2022 Spring CS6310 Assignment 5

<u>Drone Delivery System</u>

Group 061

Anqi Bao - abao32

Guochiuan Wang - gwang383

Jingjing Yao - jyao304

Minjia Li - mli726

Yan Zhang - yzhang688

Georgia Institute of Technology

Table of Contents

Overview

The Drone Delivery application is a system to monitor deliveries of grocery items to customers.
It serves as a third-party grocery service which allows customers to place order with various
stores using this system. The orders will be delivered by drones and stores can hire drone pilot to
fly drones and make deliveries.

At this phase of project, we implemented three topics that improves the application
architecturally – Authorization/Authentication, Configurability, and Auditability.
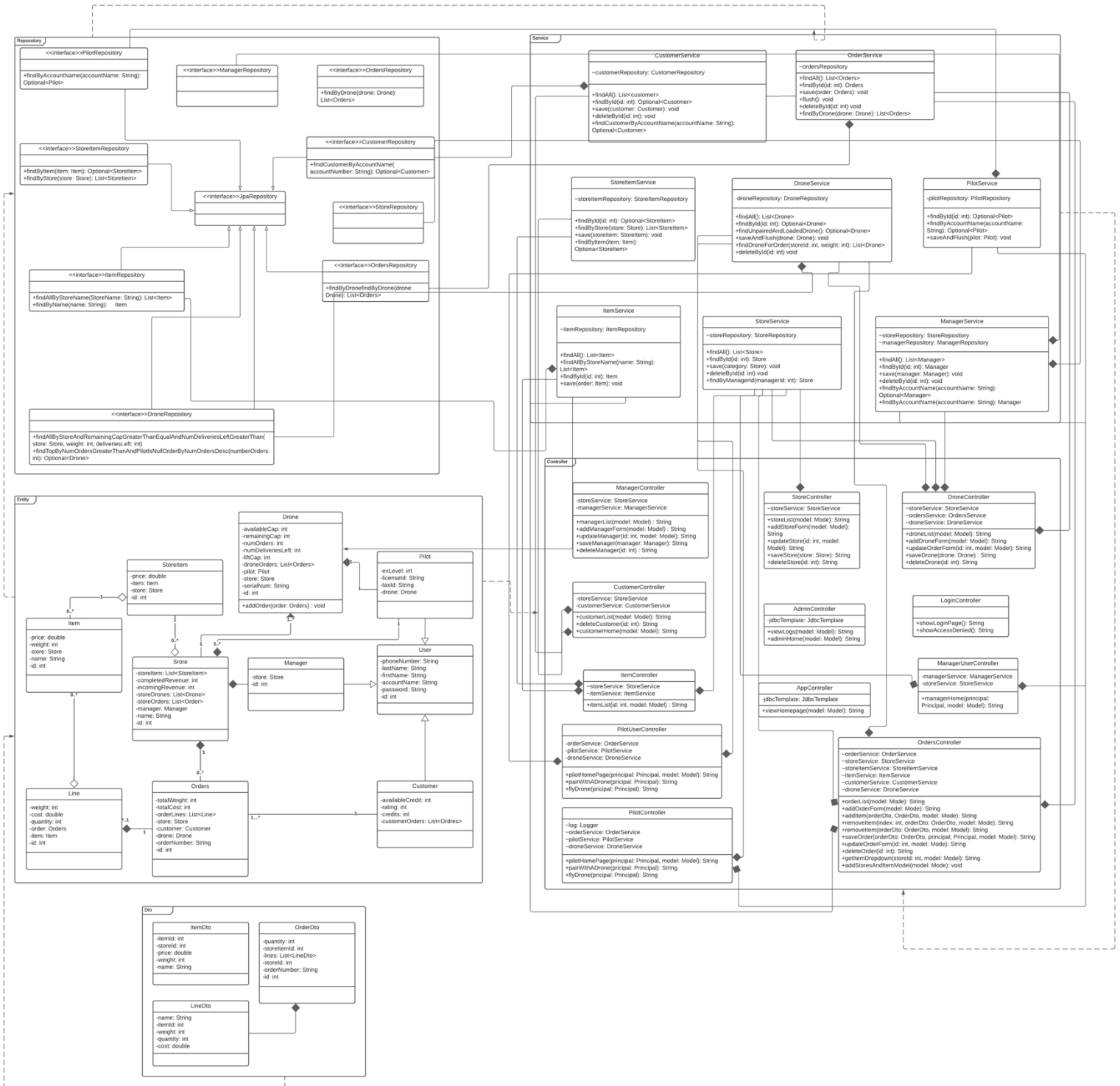
We chose Java Sprint Boot for the backend implementation, Thymeleaf for frontend, and
MySQL server for our database.

In addition, we utilized the following frameworks in our application:
- Spring Security

- Spring DTO

- Spring AOP

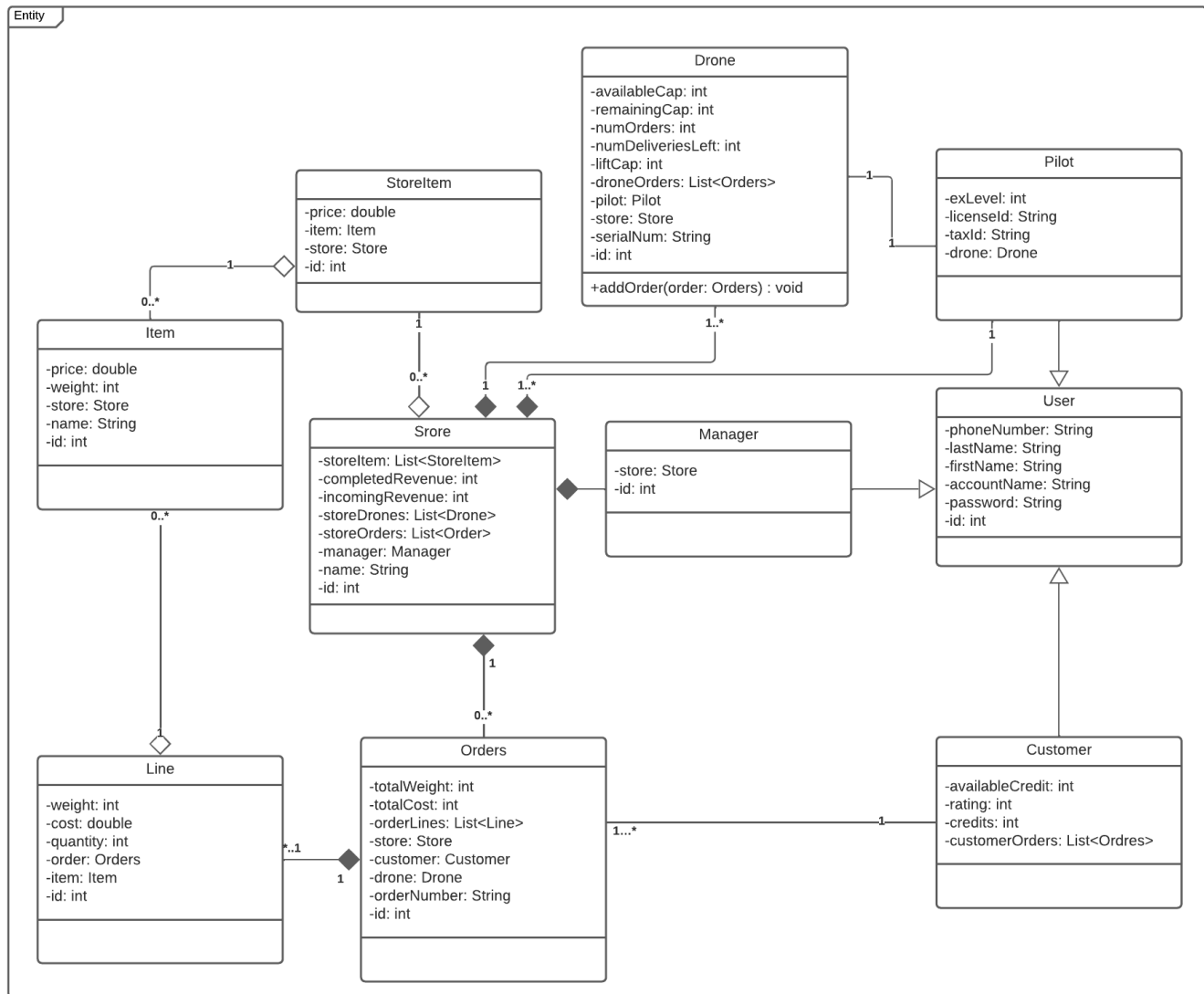- Spring Data JDBC

- Lombok

- Bootstrap

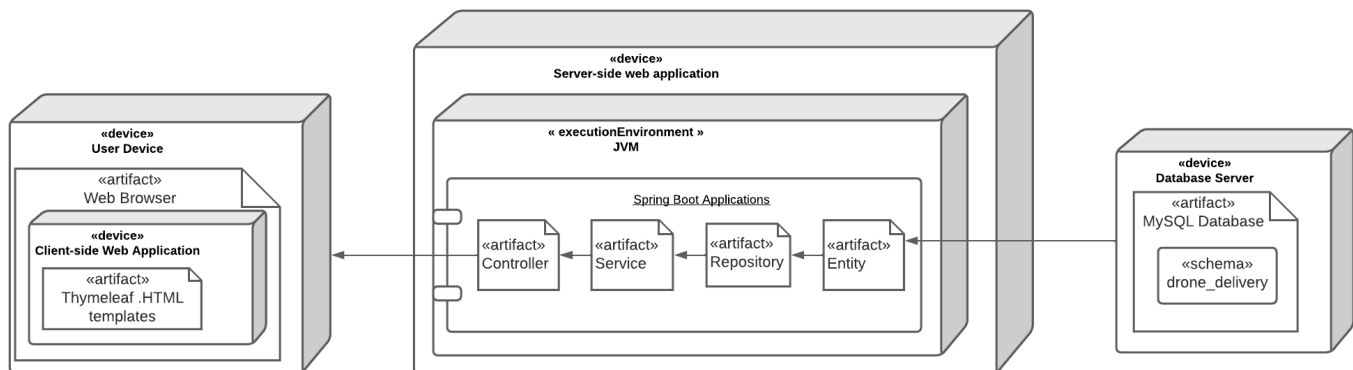# 1. UML Structural Diagrams in UML 1.4 version

1.1. Class Diagram

Repository

<<interface>>PilotRepository
+findByAccountName(accountName: String): Optional<Pilot>

<<interface>>ManagerRepository

<<interface>>OrdersRepository
+findByDrone(drone: Drone): List<Orders>

<<interface>>StoreItemRepository
+findByItem(item: Item): Optional<StoreItem>
+findByStore(store: Store): List<StoreItem>

<<interface>>CustomerRepository
+findCustomerByAccountName(accountNumber: String): Optional<Customer>

<<interface>>JpaRepository

<<interface>>StoreRepository

<<interface>>ItemRepository
+findAllByStoreName(StoreName: String): List<Item>
+findByName(name: String): Item

<<interface>>OrdersRepository
+findByDronefindByDrone(drone: Drone): List<Orders>

<<interface>>DroneRepository
+findAllByStoreAndRemainingCapGreaterThanEqualAndNumDeliveriesLeftGreaterThan(store: Store, weight: int, deliveriesLeft: int)
+findTopByNumOrdersGreaterThanAndPilotIsNullOrderByNumOrdersDesc(numberOrders: int): Optional<Drone>

Service

CustomerService
~customerRepository: CustomerRepository
+findAll(): List<customer>
+findById(id: int): Optional<Cusotmer>
+save(customer: Customer): void
+deleteById(id: int): void
+findCustomerByAccountName(accountName: String): Optional<Customer>

OrderService
~ordersRepository: OrderService
+findAll(): List<Orders>
+findById(id: int): Orders
+save(order: Orders): void
+flush(): void
+deleteById(id: int) void
+findByDrone(drone: Drone): List<Orders>

StoreItemService
~storeItemRepository: StoreItemRepository
+findById(id: int): Optional<StoreItem>
+findByStore(store: Store): List<StoreItem>
+save(storeItem: StoreItem): void
+findByItem(item: Item): Optiona<StoreItem>

DroneService
-droneRepository: DroneRepository
+findAll(): List<Drone>
+findById(id: int): Optional<Drone>
+findUnpairedAndLoadedDrone(): Optional<Drone>
+saveAndFlush(drone: Drone): void
+findDroneForOrder(storeId: int, weight: int): List<Drone>
+deleteById(id: int) void

PilotService
-pilotRepository: PilotRepository
+findById(id: int): Optional<Pilot>
+findByAccountName(accountName: String): Optional<Pilot>
+saveAndFlush(pilot: Pilot): void

ItemService
~itemRepository: ItemRepository
+findAll(): List<Item>
+findAllByStoreName(name: String): List<Item>
+findById(id: int): Item
+save(order: Item): void

StoreService
~storeRepository: StoreRepository
+findAll(): List<Store>
+findById(id: int): Store
+save(category: Store): void
+deleteById(id: int) void
+findByManagerId(managerId: int): Store

ManagerService
~storeRepository: StoreRepository
~managerRepository: ManagerRepository
+findAll(): List<Manager>
+findById(id: int): Manager
+save(manager: Manager): void
+deleteById(id: int): void
+findByAccountName(accountName: String): Optional<Manager>
+findByAccountName(accountName: String): Manager

Controller

ManagerController
-storeService: StoreService
-managerService: ManagerService
+managerList(model: Model): String
+addManagerForm(model: Model): String
+updateManager(id: int, model: Model): String
+saveManager(manager: Manager): String
+deleteManager(id: int): String

StoreController
~storeService: StoreService
+storeList(model: Mode): String
+addStoreForm(model: Model): String
+updateStore(id: int, model: Model): String
+saveStore(store: Store): String
+deleteStore(id: int): String

DroneController
~storeService: StoreService
~ordersService: OrdersService
~droneService: DroneService
+droneList(model: Model): String
+addDroneForm(model: Model): String
+updateOrderForm(id: int, model: Model): String
+saveDrone(drone: Drone) : String
+deleteDrone(id: int): String

CustomerController
-storeService: StoreService
-customerService: CustomerService
+customerList(model: Model): String
+deleteCustomer(id: int): void
+customerHome(model: Model): String

AdminController
-jdbcTemplate: JdbcTemplate
+viewLogs(model: Model): String
+adminHome(model: Model): String

LoginController
+showLoginPage(): String
+showAccessDenied(): String

ItemController
~storeService: StoreService
~itemService: ItemService
+itemList(id: int, model: Model): String

ManagerUserController
-managerService: ManagerService
-storeService: StoreService
+managerHome(principal: Principal, model: Model): String

AppController
-jdbcTemplate: JdbcTemplate
+viewHomepage(model: Model): String

PilotUserController
-orderService: OrderService
-pilotService: PilotService
-droneService: DroneService
+pilotHomePage(principal: Principal, model: Model): String
+pairWithADrone(principal: Principal): String
+flyDrone(principal: Principal): String

PilotController
~log: Logger
-orderService: OrderService
-pilotService: PilotService
-droneService: DroneService
+pilotHomePage(principal: Principal, model: Model): String
+pairWithADrone(principal: Principal): String
+flyDrone(principal: Principal): String

OrdersController
~orderService: OrderService
~storeService: StoreService
~storeItemService: StoreItemService
~itemService: ItemService
~customerService: CustomerService
~droneService: DroneService
+orderList(model: Model): String
+addOrderForm(model: Mode): String
+addItem(orderDto: OrderDto, model: Mode): String
+removeItem(index: int, orderDto: OrderDto, model: Model): String
+removeItem(orderDto: OrderDto, model: Model): String
+saveOrder(orderDto: OrderDto, principal, Principal, model: Model): String
+updateOrderForm(id: int, model: Model): String
+deleteOrder(id: int): String
+getItemDropdown(storeId: int, model: Model): String
+addStoresAndItemModel(model: Mode): void

Entity

Drone
-availableCap: int
-remainingCap: int
-numOrders: int
-numDeliveriesLeft: int
-liftCap: int
-droneOrders: List<Orders>
-pilot: Pilot
-store: Store
-serialNum: String
-id: int
+addOrder(order: Orders) : void

StoreItem
-price: double
-item: Item
-store: Store
-id: int

Pilot
-exLevel: int
-licenseId: String
-taxId: String
-drone: Drone

Item
-price: double
-weight: int
-store: Store
-name: String
-id: int

Srore
-storeItem: List<StoreItem>
-completedRevenue: int
-incomingRevenue: int
-storeDrones: List<Drone>
-storeOrders: List<Order>
-manager: Manager
-name: String
-id: int

Manager
-store: Store
-id: int

User
-phoneNumber: String
-lastName: String
-firstName: String
-accountName: String
-password: String
-id: int

Line
-price: double
-cost: double
-quantity: int
-order: Orders
-item: Item
-id: int

Orders
-totalWeight: int
-totalCost: int
-orderLines: List<Line>
-store: Store
-customer: Customer
-drone: Drone
-orderNumber: String
-id: int

Customer
-availableCredit: int
-rating: int
-credits: int
-customerOrders: List<Ordres>

Dto

ItemDto
-itemId: int
-storeId: int
-price: double
-weight: int
-name: String

OrderDto
-quantity: int
-storeItemId: int
-lines: List<LineDto>
-storeId: int
-orderNumber: String
-id: int

LineDto
-name: String
-itemId: int
-weight: int
-quantity: int
-cost: double

1.2 Entity Class Diagram

To better demonstrate the relationship between different entities, please see below a zoom-in for
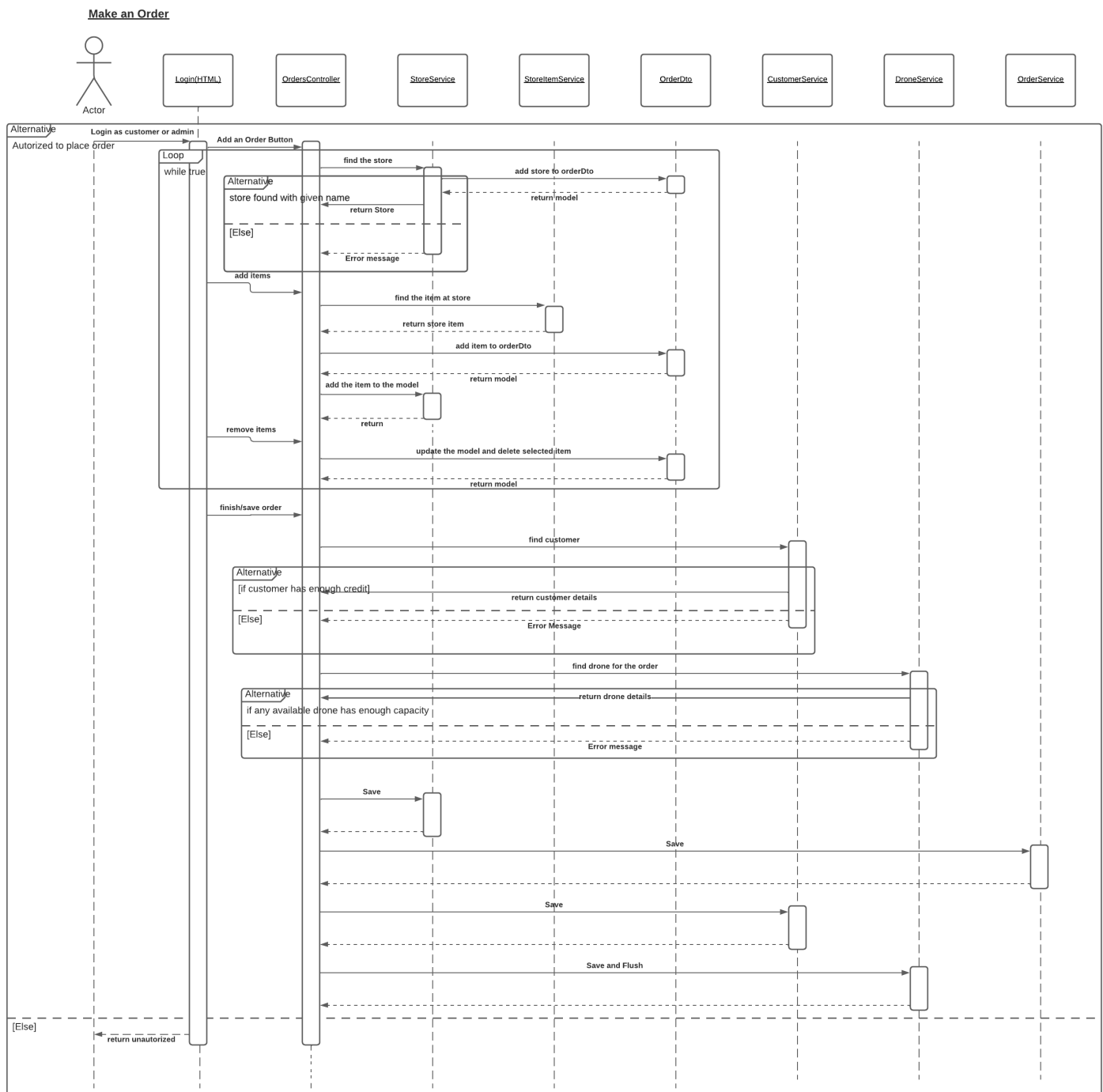
entity classes only.

1.3 Deployment Diagram
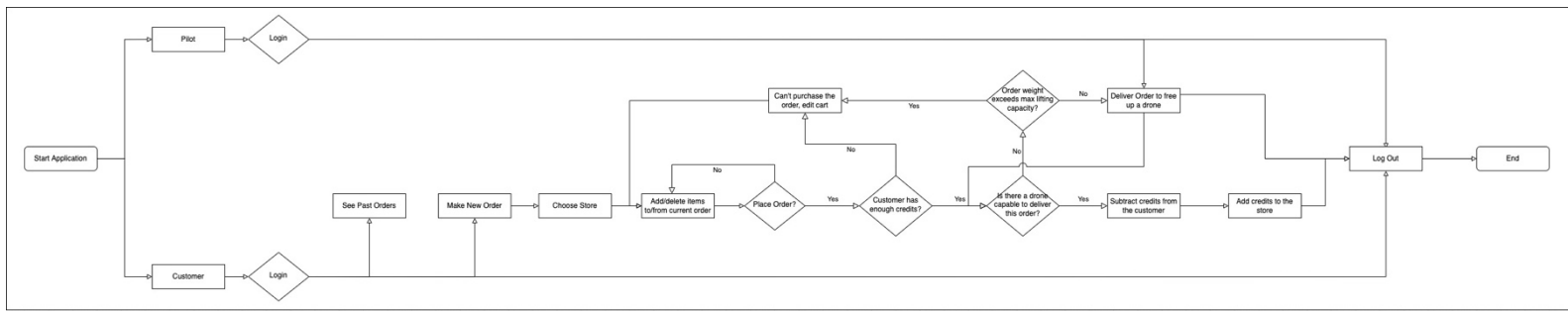
2. UML Behavioral Diagrams in UML 1.4 version

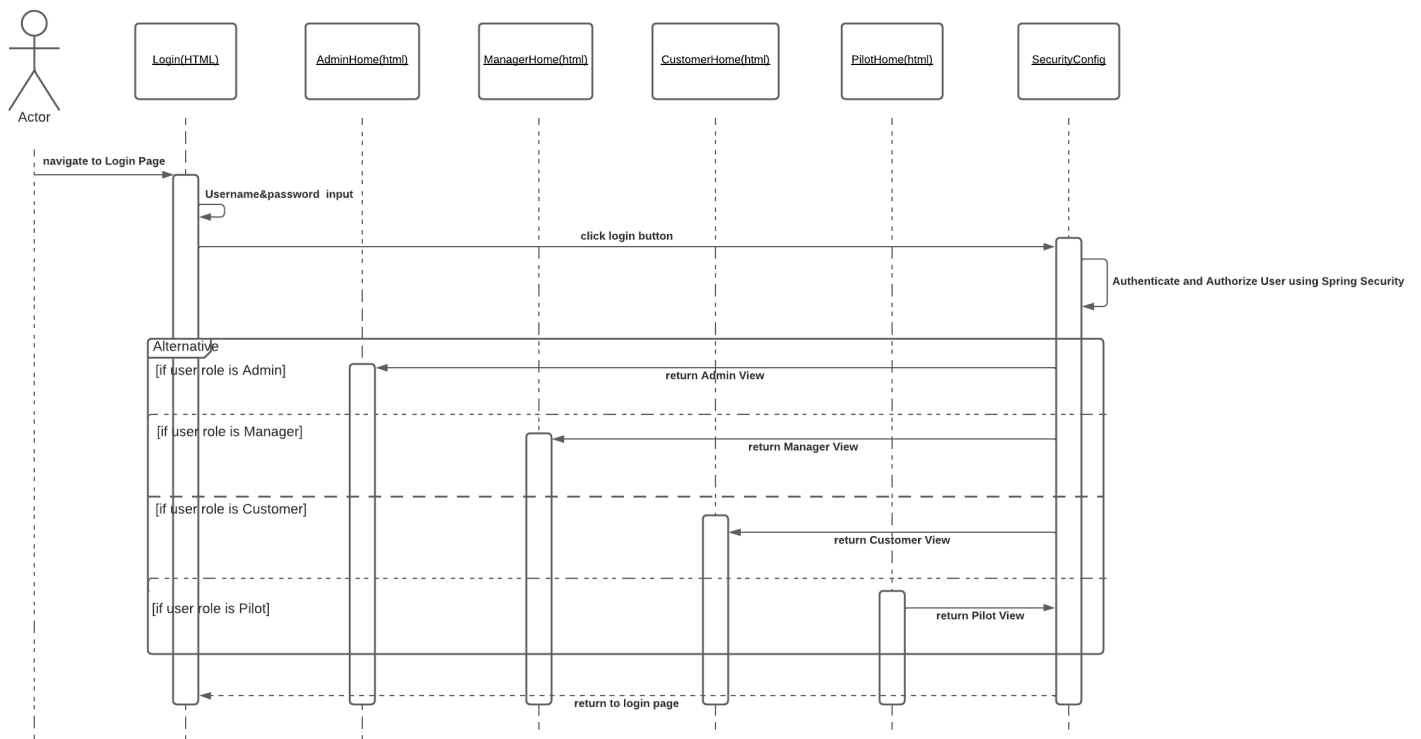## 2.1 Sequence Diagram

2.1.1 Make an Order as Customer

2.1.1a Sequence Diagram

2.1.1b Flow Chart

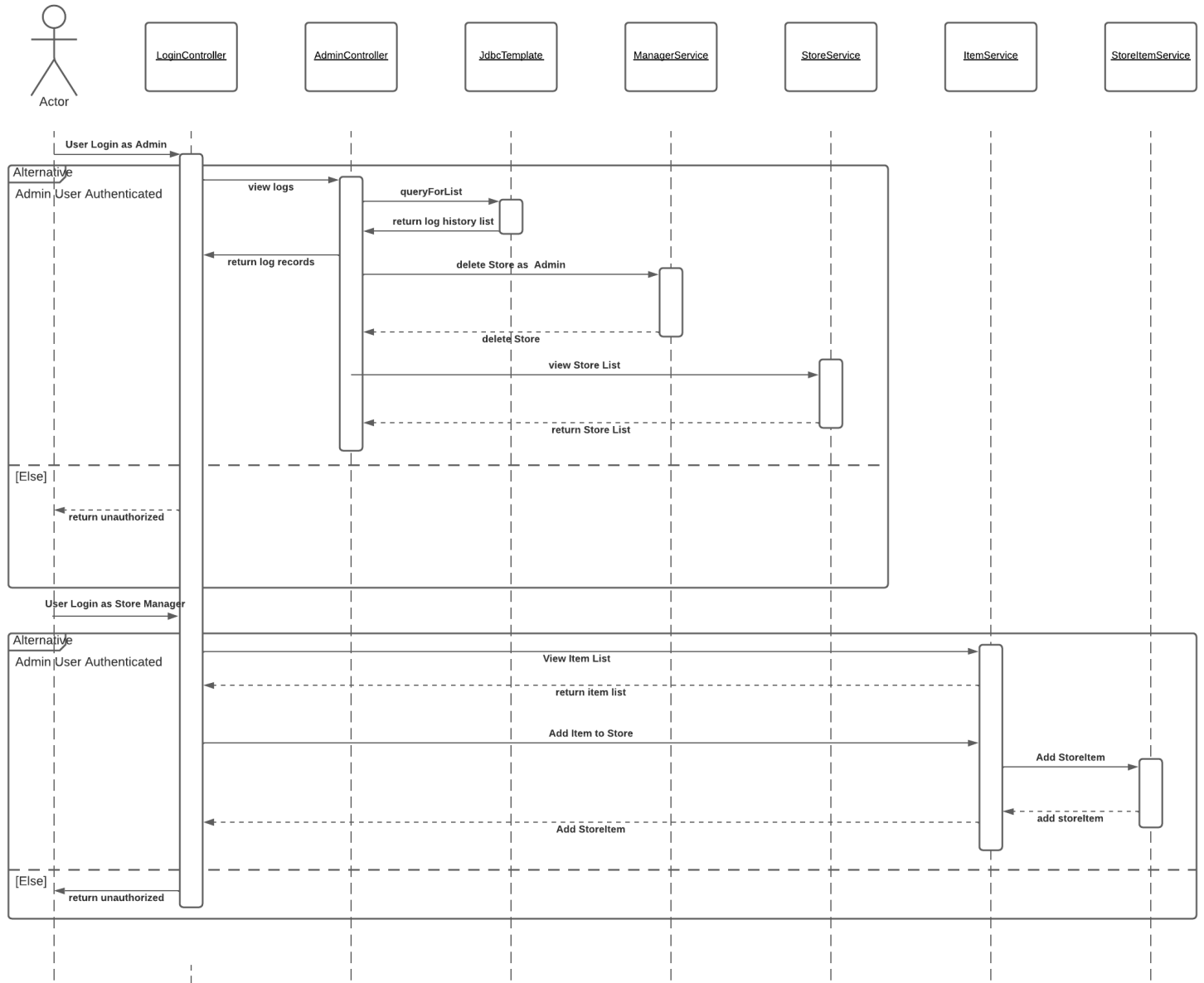## 2.1.2 Authorization/ Authentication - Login to the system in different roles

2.1.3 Configurability/ Auditability –

2.1.3a Different operations based on user roles

2.1.3b Administrator access log files

2.2 Use Case Diagram

User Case Diagram

References

https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5116986#overview