Software Engineering
Software Requirements Specification (SRS) Document

Project Name: “Restaurant Ready”
Date: 9/25/2023
Version: 1.0.0
By: Eric Hall, Sam Buttonow, Rizik Haddad

Table of Contents

1. Introduction

1.1 Purpose

The primary objective of the Point of Sale (PoS) system is to provide an easy-to-use order submission and tracking system for restaurants.This system intends to replace manual order calls by front-of-house (FoH) or waitstaff by providing a graphical user interface (GUI) to select and submit orders and facilitate transactions. The back-of-house (BoH) departments receive these orders on their own GUI terminals and mark them as waiting, in-progress, ready, or completed. It includes an administrative mode with simple tools to manage various business-related data, including the ability to perform detailed financial analysis.

1.2 Documentation Conventions & Purpose

The purpose of this Software Requirements Document (SRD) is to provide an overview of this PoS system's design, use case scenarios, and feature set. It contains descriptive information to avoid any unnecessary assumptions one may have about how this system works, from both a developer perspective and a non-technical user perspective.

1.3 Definitions, Acronyms, and Abbreviations

— FoH, BoH, SDR, GUI, API, MVC, Java, IDE, IntelliJ, SpringBoot, Figma, Insomnia JavaFX, MySQL

1.4 Intended Audience

This document is targeted to a few primary audiences and users:

- Front-of-House (FoH) personnel will gain insights into how their graphical user interface interacts with the central ticket management system.
- Back-of-House (BoH) will find details regarding the ticket's appearance on their terminals. This includes department categories like Bar, Sautee, Grill, or Bake, where specific menu items are dispatched when an order is submitted.
- Managerial personnel (admin): this document outlines administrative powers and commands available to managers of the restaurant PoS system.
- Developers will use this document to review descriptive information about how subsystems work and information about relevant technologies.

1.5 Project Scope

This system intends to provide an easy-to-use and intuitive interface for restaurant workers to coordinate as a team and satisfy incoming orders. Its presence will be at the heart of the restaurant, up to and including the responsibility of facilitating most (if not all) transactions. System owners have the ability to configure cashier profiles and define menu items, prices, and department categories.

1.6 Technology Challenges

As learning software developers, we will face many unexpected challenges when developing this software. Some of us have never implemented a GUI (graphics user interface) before, for example.

1.7 References

—

2. General Product Description

2.1 Product Perspective

Restaurant Ready is a group project for our Software Engineering course (CSC-340) at UNCG. We decided to make a PoS system based on our experience using them at jobs we've worked in the past. PoS systems have a lot of different features and as per the assignment, ours has three clearly-defined users: cashier, service department, and administrator.

2.2 Product Features

This is a multi-purpose software system designed to run on multiple machines simultaneously. Instances of the application will communicate via server-client network model. On launch, the system can be configured as a “host” or “client”. The host application runs the central ticket management system, the menu repository (database), and runs the server to which client instances connect. The client instances of the application run on terminals distributed throughout the restaurant, for example. Depending on the user (FoH or BoH), orders are sent or received. In a typical configuration, FoH workers share access to the same physical terminals and submit orders to BoH workers who receive and fulfill the orders on their terminals, in the kitchen.

2.3 User Class & Characteristics

- Host mode (restaurant owner, administrator, manager, etc.)
    - This is the main instance of the application responsible for receiving orders from the cashier terminals and dispatching them to the appropriate department terminals. It tracks incoming and active tickets to ensure all outstanding balances are satisfied or voided. This mode gives a birds-eye view of the entire system, including access to all administrative features and privileges. Here are some of the features available to hosts:
        - Add, remove, and edit cashier profiles.
        - Define the menu items, prices, department targets, and other metadata.
        - View status of currently connected clients.
        - Query the ticket management system and sales database for sales reports, etc.
- Client mode (FoH or BoH worker)
    - After connecting to a host, this mode is used by FoH to open new tickets, input orders, and facilitate transactions. Because multiple cashiers may share the same physical terminal, the "main" screen will be a profile selector to give each cashier or department access to their own interface with the system. When a ticket is opened, orders can be added to the ticket at any time and dispatched to the server’s central ticket manager.

    - This mode is also used by BoH to receive incoming, itemized orders from the host server and view the active order queue. These orders contain other, simple metadata, such as the name of the cashier that "rang it in" and time stamps. BoH can update order status as waiting, in-progress, ready, or completed.

2.4 Operating Environment

This application is intended to run on multiple devices simultaneously throughout a restaurant. Restaurant Ready is deployed as a standalone desktop application. Because it is primarily Java, we believe it may be able to run on any platform supported by JavaFX and SpringBoot, including Windows, Linux, and iOS.

2.5 Constraints

We are on a very short timeline to complete this project duly. Even with a simple feature set, the overall structure of our project is difficult to conceive and implement fully. We are completely new to developing large projects, let alone anything with a GUI or using public APIs, on top of the requirement to satisfy the needs of three different users.

2.6 Assumptions & Dependencies

Our project is dependent on SpringBoot and JavaFX and we will develop most of it in IntelliJ, a Java IDE. We will be implementing a third-party payment processing API (to be decided) to handle transactions.

3. Functional Requirements

3.1 Primary

- FR0: Order Management: The host system shall provide a user-friendly interface for restaurant staff to efficiently manage customer orders. This includes the ability to create, modify, and track orders throughout their lifecycle.
- FR1: Menu Management: The host system shall enable restaurant administrators to define and manage the restaurant's menu items. This includes specifying item names, descriptions, prices, and categorization into department categories (e.g., Bar, Sautee, Grill, Bake, Pantry). It will also include some basic functionality to manage inventory, enabling administrators to mark items as limited or "out of stock".
- FR2: User Management: The host system shall support user profiles and permissions management, allowing different staff members to have role-based access to specific features and functionalities. Time allowing, user log-in activity will be tracked for payroll.

- FR4: Payment Processing: The client system shall facilitate secure and seamless payment processing for customer orders. This includes various payment methods such as cash, credit card, and mobile payment options.
- FR6: Point-of-Sale Terminal: The client system shall provide an easy-to-use graphical interface for opening new tickets and submitting new orders. Users will have the ability to “split” tickets and individual items, review order status from the department responsible, and group items for payment processing.
- FR7: Ticket Queue Terminal: The client system shall provide department workers with an easy-to-use graphical interface for viewing incoming orders. It will also enable them to update the status of orders by the commands of the waitstaff and mark them as waiting, in-progress, ready, or completed.

3.2 Secondary

- SR0: Customizable GUI: The host system may offer customization options for the user interface, enabling restaurant owners or managers to tailor the appearance and layout to suit their brand and preferences.
- SR1: Reporting and Analytics: The host system shall offer comprehensive reporting and analytics tools to provide administrators with insights into restaurant performance. This includes the ability to generate reports on sales, revenue, and inventory levels.

4. Technical Requirements

4.1 Operating System & Compatibility

See 2.4.

4.2 Interface Requirements

4.2.1 User Interfaces

Upon launching the application, the user will be prompted to create or load a host environment or connect to a host as a client. Host environments are password-protected and loaded from disk, containing all data of the POS system. Upon logging in to the system as a host, the admin may configure the menu, user profiles, and review other information. Most importantly, they can launch the listen server responsible for communicating with the clients. If the user chose to connect to a host as a client, they will be prompted to enter the host server URL and password, which will initiate a handshake procedure. Upon connecting to the server, the host will send information for profile selection (cashier or department) and a copy of the menu repository. At this point, cashiers will have the ability to create new orders which will be dispatched to the server and relayed to the department terminals on the network.

4.2.2 Hardware Interfaces

Our standalone application will be deployed on multiple computers simultaneously. They will communicate with the host server via Transmission Control Protocol (TCP). The client-server model will function using a custom “heartbeat” protocol with buffered commands. Depending on JavaFX’s capabilities, it may be trivial in the future to implement the GUI as a touchscreen application which may run on a wall-mounted touchscreen monitor or iPad.

4.2.3 Communications Interfaces

Restaurant Ready will implement its heartbeat and command-based protocol on TCP through Java’s built-in java.net and java.nio packages. The host system will communicate with its persistent storage layer, MySQL, using SQL. The server itself will run an instance of SpringBoot which will primarily be responsible for communicating with any third-party APIs.

4.2.4 Software Interfaces

We will use JavaFX to handle any and all GUI features for Restaurant Ready. We will design our GUI online using the Figma tool. The JavaFX “front-end” will communicate with the server “back-end”, on the host application, via TCP.

5. Non-Functional Requirements

5.1 Performance Requirements

NFR1: The system must provide a response time of less than 2 seconds for order submission.
NFR2: The system should be capable of handling orders from multiple cashier terminals simultaneously.
NFR3: The system should be scalable to accommodate additional cashier terminals as the restaurant grows.

5.2 Safety Requirements

NFR4: The system must ensure that all transactions are secure and comply with industry-standard encryption practices.
NFR5: User authentication is required for access to administrative functions.
NFR6: Access to sensitive financial data should be restricted to authorized personnel only.

5.3 Security Requirements

NFR7: User passwords must be stored securely using encryption techniques.
NFR8: The system should provide role-based access control, distinguishing between administrators, cashiers, and department workers.

NFR9: Data backups must be performed regularly to prevent data loss in case of system failures.

5.4 Software Quality Attributes

5.4.1 Availability

NFR10: The system should be available 24/7, with scheduled maintenance windows communicated in advance.

5.4.2 Correctness

NFR11: The system must accurately calculate and display order totals, taxes, and discounts.

5.4.3 Maintainability

NFR12: The codebase should be well-documented, and updates or changes should be easy to implement.

5.4.4 Reusability

NFR13: Components of the system should be designed for reusability to reduce development time for future enhancements.

5.4.5 Portability

NFR14: The system should be designed to run on multiple platforms, including Windows, Linux, and iOS.

5.5 Process Requirements

5.5.1 Development Process Used

NFR15: The project will follow an Agile development methodology, with regular sprint reviews and client feedback.

5.5.2 Time Constraints

NFR16: The project must be completed within the allotted time frame of the CSC-340 course.

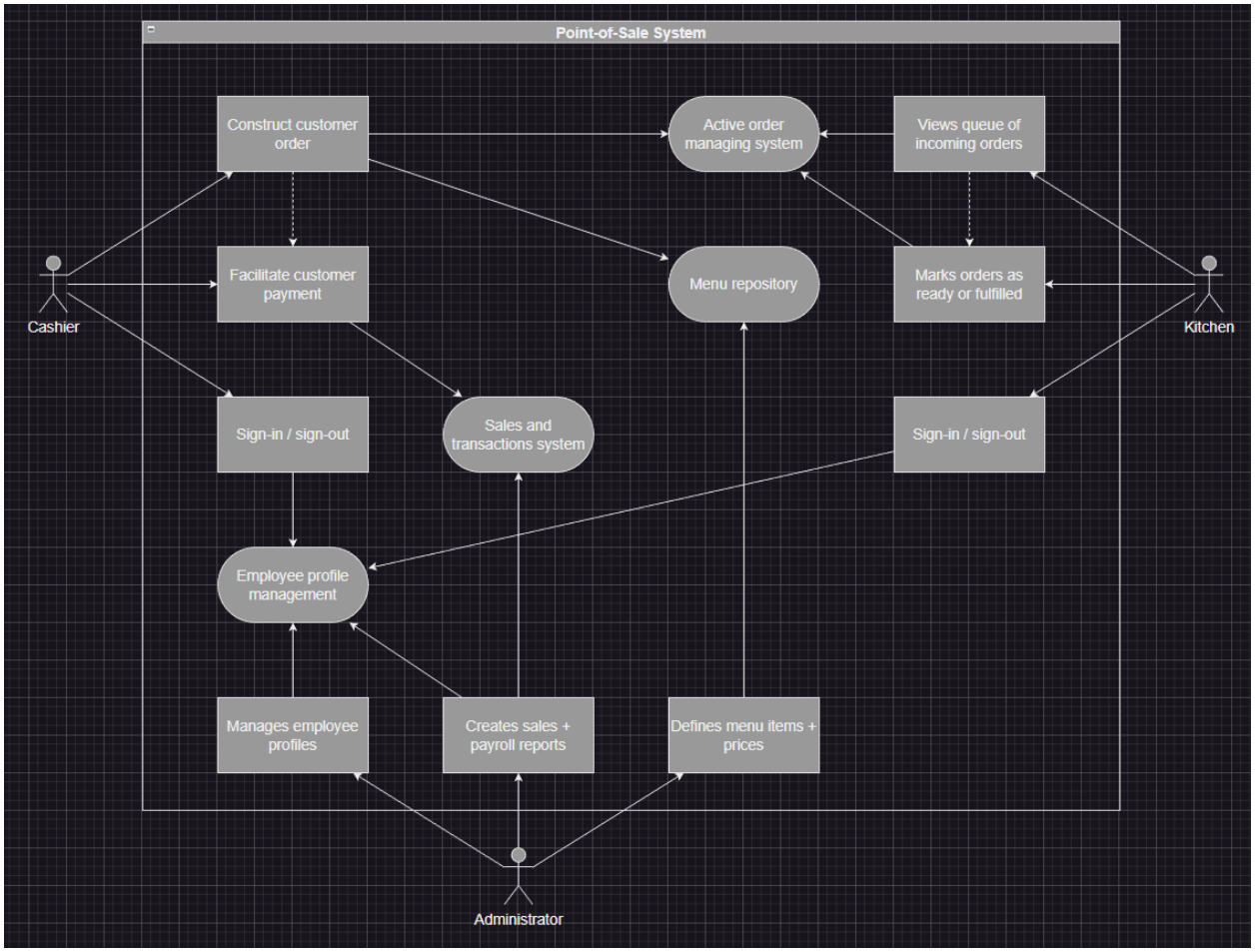
5.5.3 Cost & Delivery Date

NFR17: The development costs should be kept within the budget, and the final system should be delivered by the project deadline.

5.6 Other Requirements

NFR18: The system should be compatible with third-party payment processing APIs (to be decided).

5.7 Use-Case Model Diagram



5.8 Use-Case Model Descriptions

5.8.1 Actor: Administrator (Eric Hall)

- Manage cashier profiles. Create, update, delete.
- Define menu items, prices, inventory restrictions, department categories, etc. Read-write privileges over menu repository.
- Start/stop network service for client terminals.
- Optional: create sales/payroll reports (time permitting).

5.8.2 Actor: Front-of-House Cashier (Rizik Haddad)

- Construct customer orders via authorized client terminals.
- Facilitate transactions to finalize orders.
- Optional: “fire” an order: signal to the target department(s) to add the order to the active queue.

5.8.3 Actor: Back-of-House Department (Sam Buttonow)

- View and manage the incoming order queue.
- Collapse, expand, and otherwise rearrange order items for convenience.
- Mark orders as “waiting”, “in-progress”, “ready”, or “completed.
- Optional: signal to the host about low inventory.
- Optional: send basic messages to cashiers, ask questions, etc.

5.9 Use-Case Model Scenarios

5.9.1 Actor: Administrator (Eric Hall)

- Manage cashier profiles
    - Normal operation: Read-write privileges over cashier profiles and data.
    - What can go wrong: Persistent storage layer failures. Also, synchronization errors with client terminals.
    - System state on completion: Cashier profiles loaded and profile selection information sent to client terminals.
- Define menu repository
    - Normal operation: Add new menu items to the repository. Group items by categories, types (drink, appetizer, etc.), and departments.
    - What can go wrong: Synchronization errors with client terminals.
    - System state on completion: Menu database is well-defined and easily loaded. Client terminals receive a copy and any updates.
- Start/stop network server.
    - Normal operation: Listen server handles client connections and I/O.
    - What can go wrong: Network socket errors. Shutting down service during business operations requires client re-sync.
    - System state on completion: Client connections are stable and authorized securely.

5.9.2 Actor: Front-of-House Cashier (Rizik Haddad)

- Construct customer orders.
    - Normal operation: Browse menu and select appropriate modifiers per ticket.
    - What can go wrong: Expected menu items missing, prices not matching, etc.
    - System state on completion: Tickets updates are relayed successfully to the host application.

- Complete customer transactions.
    - Normal operation: Charges accumulated by orders are paid for by customer's cash, credit card, or mobile payment options.
    - What can go wrong: Third-party payment processing systems may fail for any number of reasons.
    - System state on completion: All outstanding balances are satisfied: paid or voided by an administrator.

5.9.3 Actor: Back-of-House Department (Sam Buttonow)

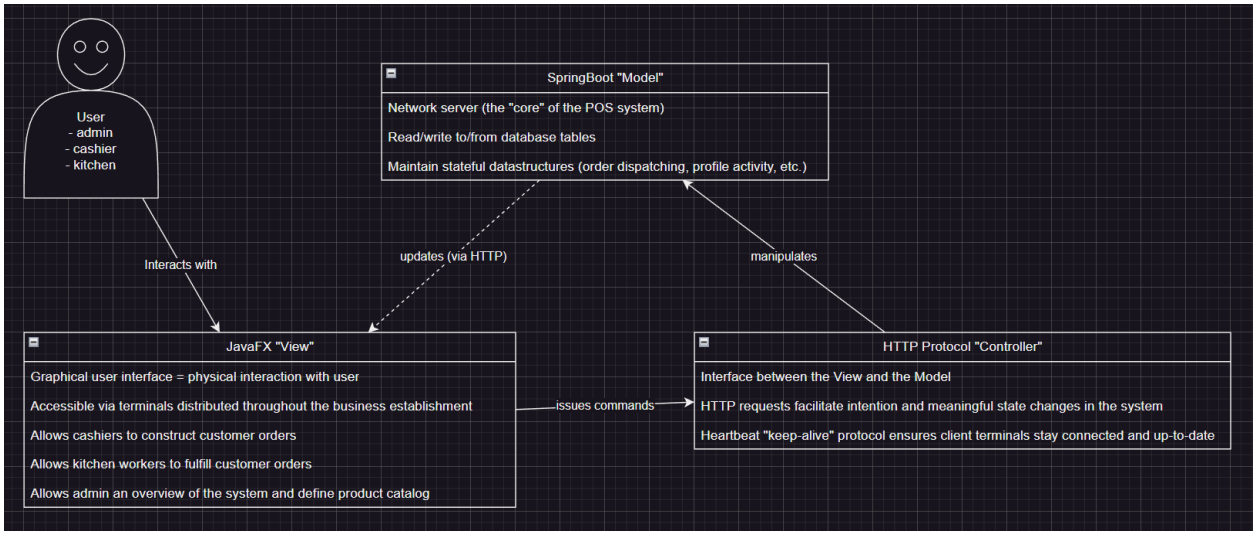
- Manage customer order queue.
    - Normal operation: Department workers can navigate the queue of active or completed orders and update status.
    - What can go wrong: Synchronization errors with the host system.
    - System state on completion: Orders are displayed correctly and all controls function as expected.

6. Implementation Overview

6.1 Software architecture

We have chosen to implement a simple Model-View-Controller design structure for our application. This is suitable because there is a well-established separation between the model (the SpringBoot server) and the view (the JavaFX client). Communication between the view and the model is handled by the controller (the HTTP protocol).

A conceptual model of this architecture is described by the diagram below:



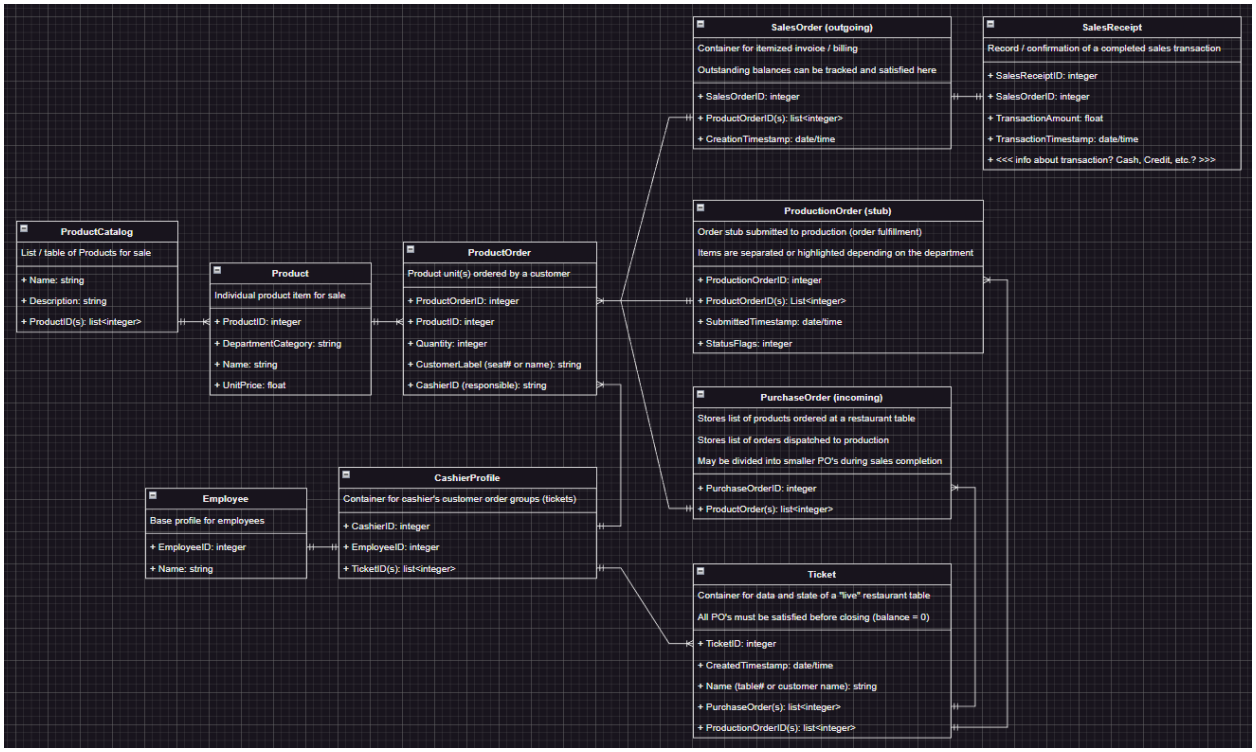
6.2 High-Level Schema

Our system needs to handle and manipulate a lot of data. This data exists in the form of objects encapsulating some stateful representation of the system. Sometimes, this data is persistent (in the case of sales and profile configurations) and other times it is transient or temporary (in the case of the GUI lifecycle). As itemized orders come in, they need to be defined, tracked, fulfilled, and payment must be satisfied or voided manually. This data must be accessible in multiple locations, in whole or in part, and so we have modeled a general "schema" to map the relationship between different entities in the system.

In summary, the admin or owner of the POS system is responsible for defining the "product catalog". This can be thought of as the menu handed to you when you sit down at a service restaurant, or an illuminated menu displayed above and behind the service counter at a fast food restaurant. The cashier is responsible for opening a "ticket" and adding product options chosen by the customer from the product catalog. Each ticket is an itemized list of products ordered by the customer that need to be paid for. The term "order" has a different meaning depending on the context in which it is used.

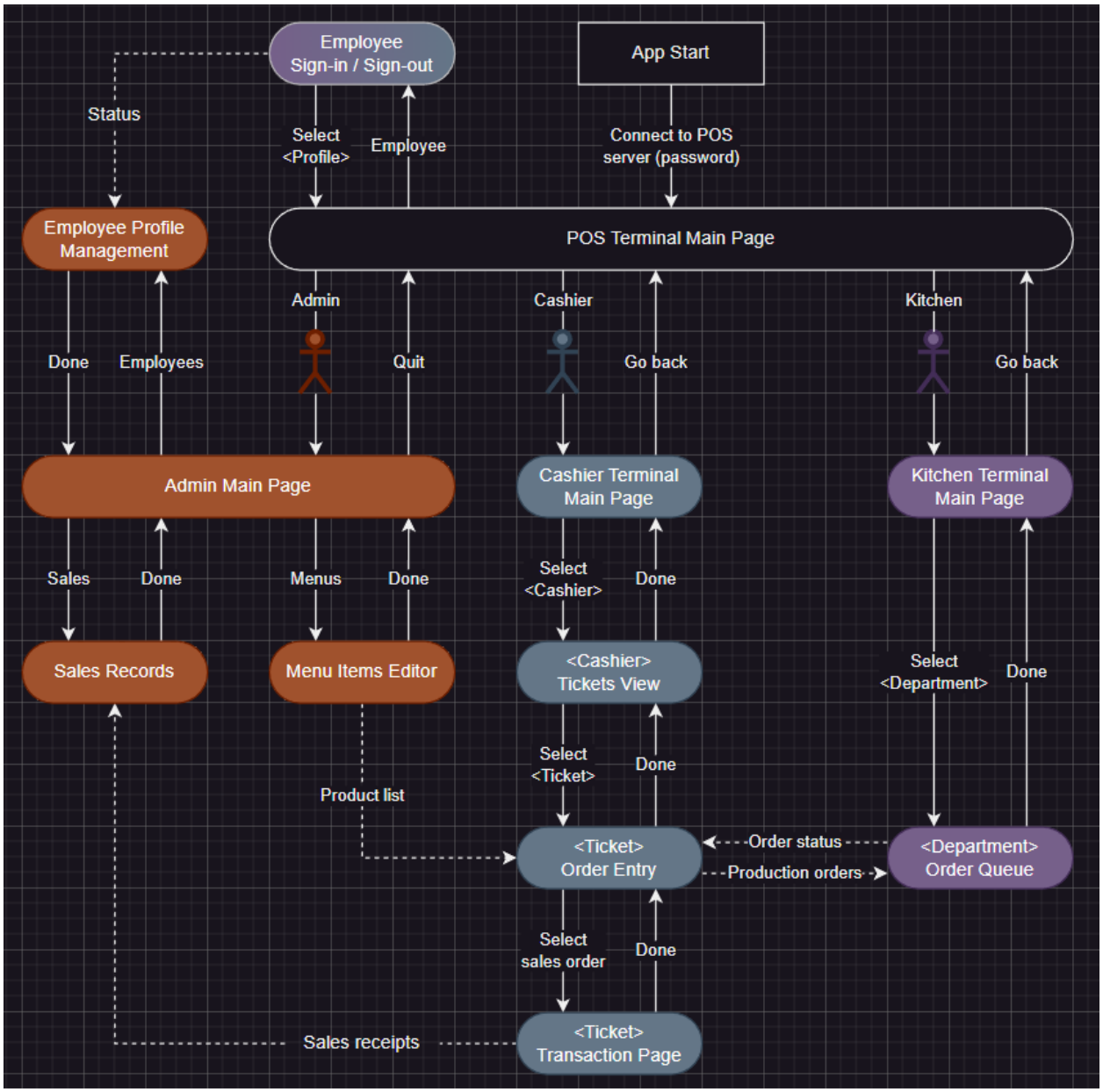
1. **Production Order:** This is a group of items to be produced *together* by the production staff (i.e. the kitchen). A production order may contain course breaks and the POS system should reflect back to the cashier the production status of the items ordered: holding, in-progress, ready, complete, etc. A production order is simply a "stub" or a stand-in for a purchase order.
2. **Purchase Order:** This contains a full list of items ordered by customers associated with a ticket, typically organized by a seating position number. These represent "incoming" orders issued by a customer. Purchase orders can be "split" into multiple purchase orders to allow cashiers to issue select subsets of the total items ordered as a single charge.
3. **Sales Order:** This is an itemized invoice or bill issued to the customer for payment. Sales orders are paid for by the customer and converted into sales receipts. These represent "outgoing" orders to be paid for by the customer.

Below is a relational diagram representing a prototype model of the data that we will be working with:



6.3 Software Interface - GUI State Machine

Below is a diagram showing an overview of the entire JavaFX front-end application. It is worth noting that each node is a distinct JavaFX page (loaded by an .fxml document) with a corresponding controller and will be capable of manipulating or viewing a subset of the data defined above. Only the admin part of the application will be protected by password. The admin may also “spin up” the SpringBoot server to set the computer running it as the host network application to which the terminals distributed throughout the restaurant can connect.



6.4 UML Class Diagram

- Did not finish.