**Supermarket Handling Application**

**Yash Nigam**

**A2305219003**

**INDEX**

Abstract

Chapter 1: Introduction

Chapter 2: Literature

Chapter 3: Methodology

Chapter 4: Design

Chapter 5: Input / Output

Chapter 6: Conclusion

Chapter 7: Future Scope

**Abstract**

Supermarket has become a very important part of our life. In this difficult time of pandemic, it has become a very crucial as everyone wants to get the necessities and want to buy them is safe environment as well as do not have to spend any more time than they want to in the Supermarket in order to protect themselves. Even before this the problem of standing in long checkout queues was there which had to be addressed. Hence a need for much better system than the one already in place. Therefore, I thought to design a system which would address the following concerns:

* Maintaining social distance while shopping
* Zero contact policy

The system I designed includes all the features required in the Handling of a Supermarket as well as a self-checkout system which could be either implemented at the end stage like ordinary checkout system but the customers have to scan the products themselves or could be implemented as when the customer picks and puts the product in the cart it would be automatically be added to the cart, this would save a lot of time and effort, would also streamline the shopping process and would address these immediate concerns.

That is all about the project, I have used JavaFX version 11 for the development of the project, JavaFX provides many features like designing UI without writing codes with the help of Scene Builder which creates a FXML file for the same, it also provides CSS (Cascading Style Sheets) to style the already made UI. At the same the back-end logic could be written in Java. I chose JavaFX because I want to expand my knowledge of Java and wanted to make full fledge application of java with a working database.

**Chapter 1: Introduction:**

Supermarket as is a place where a customer could get a wide variety of products at a single place. There are many different sizes of stores, some are small shops whereas others are as big as two adjoined warehouses and in all these something are common irrespective of the size of the store. For all these stores to function smoothly and properly a lot of systems are deployed to maintain the balance of working in these stores. Apart from systems, manpower is also needed to do so, as things need to be configured, checked, and corrected by a person.

Hence a lot of systems are deployed in supermarket to cover every aspect of the store like checkout system, inventory system, management system, etc, this in sync with the employees is what forms a supermarket which functions smoothly.

The objective of this project was to create a unified system which would include various independent systems like inventory management system, checkout system, employee management system, etc. This would make the management streamline and easy. It would be easy for the owner to manage all the details and for the employees to mark their attendance and perform checkouts, also it makes easier for the customers to perform self-checkout and submit feedbacks which help in improve the overall customer service experience and also the management system. Deployment of a unified system also helps in the accounting as all the database a centrally managed and hence is easier to perform operation related to it.

**Chapter 2: Literature:**

**2.1 Java FX**

JavaFX library is written as a Java API, JavaFX application code can reference APIs from any Java library. Like, JavaFX applications are able to use Java API libraries to access native system capabilities and connect to server-based middleware applications. JavaFX applications UI and design can be modified by the use of CSS. Cascading Style Sheets (CSS) separate appearance and style from implementation so that developers can concentrate on coding. To separate the user interface (UI) and the back-end logic, then you can develop the presentation aspects of the UI in the FXML scripting language and use Java code for the application logic. To design UIs without writing code, JavaFX Scene Builder can be used. As the UI has been designed, Scene Builder creates FXML markup that can be ported to an Integrated Development Environment (IDE) so that the business logic can be added to it later.

The JavaFX APIs are available as a fully integrated feature of the Java SE Runtime Environment (JRE) and the Java Development Kit (JDK). Because the JDK is available for all major desktop platforms (Windows, Mac OS X, and Linux), JavaFX applications compiled to JDK 7 and later also run on all the major desktop platforms. Support for ARM platforms has also been made available with JavaFX 8. JDK for ARM includes the base, graphics, and controls components of JavaFX. The cross-platform compatibility enables a consistent runtime experience for JavaFX applications developers and users. Oracle ensures synchronized releases and updates on all platforms and offers an extensive support program for companies that run mission-critical applications.

Key features included in JavaFX 8 and in later releases are Java APIs, FXML and Scene Builder, Web View, Swing interoperability, Built-in UI controls and CSS, Moderna themes, 3D Graphics Features, Canvas API, Printing API, Rich Text Support, Multitouch Support, Hi-DPI support, Hardware-accelerated graph pipeline, High-Performance media engine, Self-contained application deployment model.

The features that have been Used in this project:

* **FXML and Scene Builder:** FXML is an XML-based declarative markup language for constructing a JavaFX application user interface. Code can be written in FXML or JavaFX Scene Builder could be used to interactively design the graphical user interface (GUI). Scene Builder generates FXML markup that can be ported to an IDE where a developer can add the business logic.
* **Built-in UI and controls and CSS:** JavaFX provides all the major UI controls that are required to develop a full-featured application. Components can be skinned with standard Web technologies such as CSS. The DatePicker and TreeTableView UI

controls are now available with the JavaFX 8 release.

* + **TableView:** it is designed to visualize an unlimited number of rows of data, broken out into columns. A TableView is therefore very similar to the ListView control, with the addition of support for columns.
  + **GridPane Layout:** it enables to create a flexible grid of rows and columns in which the controls could be laid out. Cells in this layout can expand or contract as per the need.
  + **ImageView:** The ImageView is a Node used for painting images loaded with [Image](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html) class. This class allows resizing the displayed image (with or without preserving the original aspect ratio) and specifying a viewport into the source image for restricting the pixels displayed by this ImageView.
  + **-fx-background** styling option is used to set the background colour of the Pane, buttons, text fields etc
  + **-fx-text-fill** styling is used to change the colour of the text in the fields
  + **-fx-border-radius** styling is used to make the buttons rounds
* **Self-contained application deployment model:** Self-contained application

packages have all the application resources and a private copy of the Java and

JavaFX runtimes. They are distributed as native installable packages and provide

the same installation and launch experience as native applications for that

operating system.

**2.2 Database**

Database is a well-structured collection of information, or data, typically stored electronically in a computer system. It is controlled by a database management system (DBMS). A system is called a database system when it has application associated with database.

The most common way to sort data in the database is in the form of rows and columns is a series of table which makes processing and data query efficient. In this form data can be easily and quickly accessed, modified, controlled, organized, updated, and managed. Most of the database use Structured Query Language (SQL) for writing and query data.

SQL is a programming language used by nearly all relational database to query, manipulate, and define data, and to provide access control. It was first developed in the 1970s with Oracle as a major contributor, which led to implementation of the SQL ANSI standard, SQL has spurred many extensions from companies such as IBM, Oracle, and Microsoft. Although SQL is still widely used today, new programming language are beginning to appear.

There are any different types of databases:

* **Relational Database:** it became dominant in the 1980s. Items in a relational database are organized as a set of tables with columns and rows. Relational database technology provides the most efficient and flexible way to access structed information.
* **Objected-Oriented Database:** information in an object-oriented database is represented in the form of objects, as in objects-oriented programming.
* **Distributed Databases:** a distributed database consists of two or more files located in different sites. The database may be stored on multiple computers, located in the same physical location, or scattered over different networks.
* **Data Warehouses:** A central repository for data, a data warehouse is a type of database specifically designed for fast query and analysis.
* **NoSQL Database:** A [NoSQL](https://www.oracle.com/in/database/nosql-cloud.html), or nonrelational database, allows unstructured and semi-structured data to be stored and manipulated (in contrast to a relational database, which defines how all data inserted into the database must be composed). NoSQL databases grew popular as web applications became more common and more complex.
* **Open-Source Database:** An open-source database system is one whose source code is open source; such databases could be SQL or NoSQL databases.
* **Cloud Database:** A [cloud database](https://www.oracle.com/in/database/what-is-a-cloud-database/) is a collection of data, either structured or unstructured, that resides on a private, public, or hybrid cloud computing platform. There are two types of cloud database models: traditional and database as a service (DBaaS). With DBaaS, administrative tasks and maintenance are performed by a service provider.
* **Multi-model Database:** Multi-model databases combine different types of database models into a single, integrated back end. This means they can accommodate various data types.
* **Document/JSON Database:** Designed for storing, retrieving, and managing document-oriented information, [document databases](https://www.oracle.com/in/autonomous-database/autonomous-json-database/) are a modern way to store data in JSON format rather than rows and columns.
* **Self-Driving Database:** The newest and most ground-breaking type of database, self-driving databases (also known as autonomous databases) are cloud-based and use machine learning to automate database tuning, security, backups, updates, and other routine management tasks traditionally performed by database administrators.

MySQL is an open-source relation database management system based on SQL which was designed to run on any platform and has been optimized for web applications. It has become the top choice for web-based application as new and different requirements are surfacing with the internet. It has been designed to process tens of thousands of queries and transactions and hence has become a popular choice for the ecommerce businesses. On-demand flexibility is the primary feature of MySQL. It is in use for many popular websites and web-based application like Airbnb, Uber, LinkedIn, Facebook, Twitter, and YouTube.

**Chapter 3: Methodology**

**3.1 Planning of the Project**

When I started my project, I laid out a plan which comprised of the following details:

* Name of the pages which would be there in the application
* Name and function of the components to be used in those pages.
* Designing the data flow logic
* Designing the overall application flow logic
* Studying various documents in order to find the best possible component.
* Study the concepts of JavaFX
* Study the concepts of database and its integration in Java

Through extensive study from the oracle documentation, blogs regarding the concerned topics and YouTube videos I was able to gather enough know-how to develop the application and provided a better understanding of the technology.

**3.2 Resources**

* Researched and found various JavaFX components which would best suit my application
* Studied JavaFX documentation which would give the know how of how to initialize a JavaFX application, how to configure JavaFX and how to run the application.
* Explored JavaFX libraries that would be required to develop the application
* Studied Database Management, how to create schemas, tables and how to integrate them into the application
* Learnt MySQL queries required to create code that would communicate with the database through the application.

**3.3 Technology and Software Used**

There has been use of several technologies in this project like FXML and CSS for frontend and Java and Database for backend. JavaFX preferred over Java Swing as it provides more tools and styling options. It also has many features that helps in making a much better production ready application.

The Software used in the development of the application are:

* **IntelliJ**

IntelliJ IDEA is an integrated development environment written in Java for developing computer software. It is developed by JetBrains and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition.

Background pattern

Description automatically generated

* **Scene Builder**

It is interactive GUI designing software developed by Oracle that helps in designing of the user interface quickly and with less need of writing the code in JavaFX.

Icon

Description automatically generatedGraphical user interface, application

Description automatically generated

* **MySQL Workbench**

It is a visual database design tool that integrates SQL development, administration, database design, creation, and maintenance into a single integrated development environment for the MySQL database system. It is the successor to DBDesginer 4.

Graphical user interface, text, application, email

Description automatically generated

**Chapter 4: Design**

**4.1 Components**

Application consists of various components as listed below:

* Homepage
* Login page
* Admin View
* Employee View
* Employee Attendance System
* Inventory Management System
* Checkout System
* Vendor Management and Placing Order System
* Customer Feedback System

**4.2 Components**

The working of the application is as follows:

* The Login System is used by both the owner and the employees, in the case of employees it is used to open the Employee View and mark their attendance and in the case of owner it is used to open the Owners View where vendors, employee and inventory can be managed. It also shows Today’s Sale.

Diagram

Description automatically generated

**Fig 4.2.1 Login Page Data Flow Diagram**

* In the owner view there are many buttons performing various functionality, like managing inventory, managing employee details, checking if inventory of short of any product, checking the attendance of the employee, managing vendors, and seeing today’s sale. In the vendors section there is a functionality to place order for specific products with the vendors of the choosing, in the attendance section there the owner could check the attendance of employees of any day.

Diagram

Description automatically generated

**Fig 4.2.2 Vendor Page Data Flow Diagram**

Diagram

Description automatically generated

**Fig 4.2.3 Inventory Page Data Flow Diagram**

Diagram

Description automatically generated

**Fig 4.2.4 Manage Employee Detail Data Flow Diagram**

* In the employee view, the employee marks their attendance of the day and if they are present on the day, they proceed to cashier checkout or if they are on leave, they mark on leave and give the reason.

Diagram

Description automatically generated

**Fig 4.2.5 Attendance System Data Flow Diagram**

* Self-Checkout is the application screen that is displayed on the customers machines, it is done so because they do not need the functionality of login hence no need to direct them through that page as it would create a vulnerability in the system. Furthermore, in the self-checkout page the customer can check out the item themselves without waiting in long cashier ques, the only requirement for this is that the payment method is card payment or when implemented as web application could be integrated with a payment gateway which would them turn into digital money. In the self-checkout the customer has to enter the product id mentioned on the label and enter the quantity or scan through the scanner, after that the customer can add more products and even remove the already added products. The similar procedures followed by the cashier on the cashier checkout page.

Diagram

Description automatically generated

**Fig 4.2.6 Checkout System Data Flow Diagram**

* After the this as the Pay button takes them to the purchase confirmation page. There are two types of confirmation page in the application, Cash Purchase Confirmation page and Card Purchase confirmation page.
  + Cash Purchase Confirmation Page: this page in integrated with the cashier checkout page as if they are going through cashier checkout the customer would be paying cash. There the page takes the amount paid by the customer and returns the change left.

Diagram

Description automatically generated

**Fig 4.2.7 Cashier Payment Confirmation Data Flow Diagram**

* + Card Purchase Confirmation Page: this page is integrated with the self-checkout page as the customers are required to pay with a card if they are going through self-checkout page. The total amount is displayed on the screen and customer is supposed to pay the amount with either tapping or swiping the card on the card machine, in response to this the machine would generate a unique code which is associated with the payment. The code is displayed to the customer, and they have to enter it in the system to continue further. On the payment end when the payment is received for the specified amount the unique code as well as the amount is automatically storing in the database. When the customer enters the unique code and confirms the payment the system checks the amount against that unique code, the amount matches they can process, if it does not, they must talk to the cashier to complete the transaction.

Diagram

Description automatically generated

**Fig 4.2.8 Card Payment Confirmation Data Flow Diagram**

* Both the checkout pages have a membership discount option, if the customer holds an active membership of the store, they avail a discount of 5% on the total amount. If not so they can enrol for a new membership via cashier of through the self-checkout system itself. Customer can also check their membership status in the self-checkout by clicking on the membership button.

Diagram

Description automatically generated

**Fig 4.2.9 Membership Details Data Flow Diagram**

* In the feedback page the customer can leave a feedback for any improvement. It is saved in the database and can only be viewed by owner in the feedback section the owner view.

Diagram

Description automatically generated

**Fig 4.2.10 Feedback Data Flow Diagram**

**Chapter 5: Input / Output**

**Homepage:**

Diagram

Description automatically generated with medium confidence

**Fig 5.1 Homepage**

**Owner View:**

Graphical user interface, application

Description automatically generated

**Fig 5.2 Owner View**

**Login System:**

Graphical user interface, application

Description automatically generated

**5.3 Admin Login**

Graphical user interface, application

Description automatically generated

**Fig 5.4 Employee Login**

**Employ View:**

A screenshot of a cell phone

Description automatically generated with medium confidence

A screenshot of a cell phone

Description automatically generated

**Self-Checkout:**

Graphical user interface, table

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

**Cashier Checkout:**

Graphical user interface, table

Description automatically generated

Graphical user interface

Description automatically generated

**Cash Payment Authentication System:**

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

**Card Payment Authentication System:**

Graphical user interface, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**Membership View System:**

Graphical user interface, website

Description automatically generated

**Feedback System:**

**Graphical user interface, website

Description automatically generated**

**Table

Description automatically generated**

**Chapter 5: Conclusion**

The application is working for the intended purpose. The self-checkout System is working efficiently. The GUI is working smoothly. All the other systems are also working properly, there are no bugs in the code and the project gave me the working of JavaFX, FXML, CSS and MySQL and how to use them all together. The libraries required for them to work together. All the systems are interlinked and working as intended.

**Chapter 6: Future Work**

The application made by me has a lot of future scope and can be expanded as per the requirements. More importantly this application can be deployed as a standalone application, could be run on a single system, and can also be deployed as web-application.

Hence when deployed as web-application a payment gateway can also be included to provide more payment options like Net Banking, UPI, and other Wallets like Paytm, PhonePe and many more.

**Code:**

**References:**

* <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html>
* <https://www.oracle.com/in/database/what-is-database>