Takeaway Management System

CS4125 - Systems Analysis and Design

Team members

- **Conor Moloney - 15144127**
- **Brian Dooley - 15123529**
- **Artem Semenov - 15164748**
- **Maihanjan Abdul Qayoom - 14018276**

Narrative Description

Takeout restaurants are a huge industry that do hundreds of millions of euro worth of business around the world each year. However businesses in this industry must tackle a number of challenges in order to be successful. These include allowing users to set up accounts, make orders, allowing their employees to process and track the orders and managing their stock levels.

Our aim for this project was to develop a system capable of addressing these challenges. As this system has the possibility of being implemented by a wide variety of businesses around the world, many of which may deal with extremely large amounts of transactions, we aimed to develop it with key quality attributes such as extensibility and scalability in mind.

The key pieces of functionality that our system should provide to the user are as follows: The ability to create an account, the ability to log in using a previously created account, the ability for a customer to view the menu, the ability for a customer to place an order, the ability for a customer to view their previous orders, the ability for an employee to process orders, the ability for an employee to order stock.

Software Development Life-Cycle

Software development life cycle (SDLC) is series of phases, it provides the common understanding in software building process. (SDLC) describe stages and tasks that are involved in each steps of a project to write and deploy software, software engineer must have enough knowledge on how to choose the right SDLC based on the project context. To ensure the success of the project its very important to choose the right SDLC based on project context and requirements.
The above diagram Figure1 designed in Adobe Photoshop CC 2018.

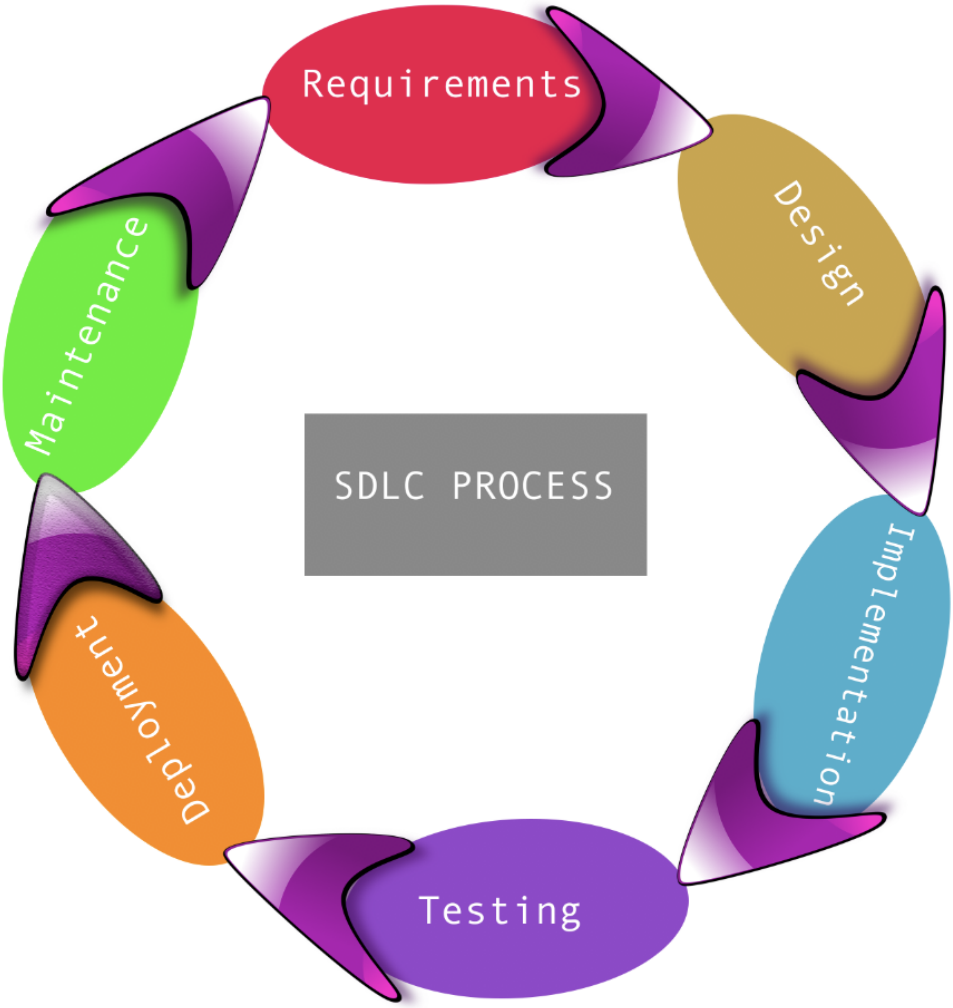


Figure 1.

Waterfall Model

Waterfall model is the earliest SDLC approach that was used for software development. The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

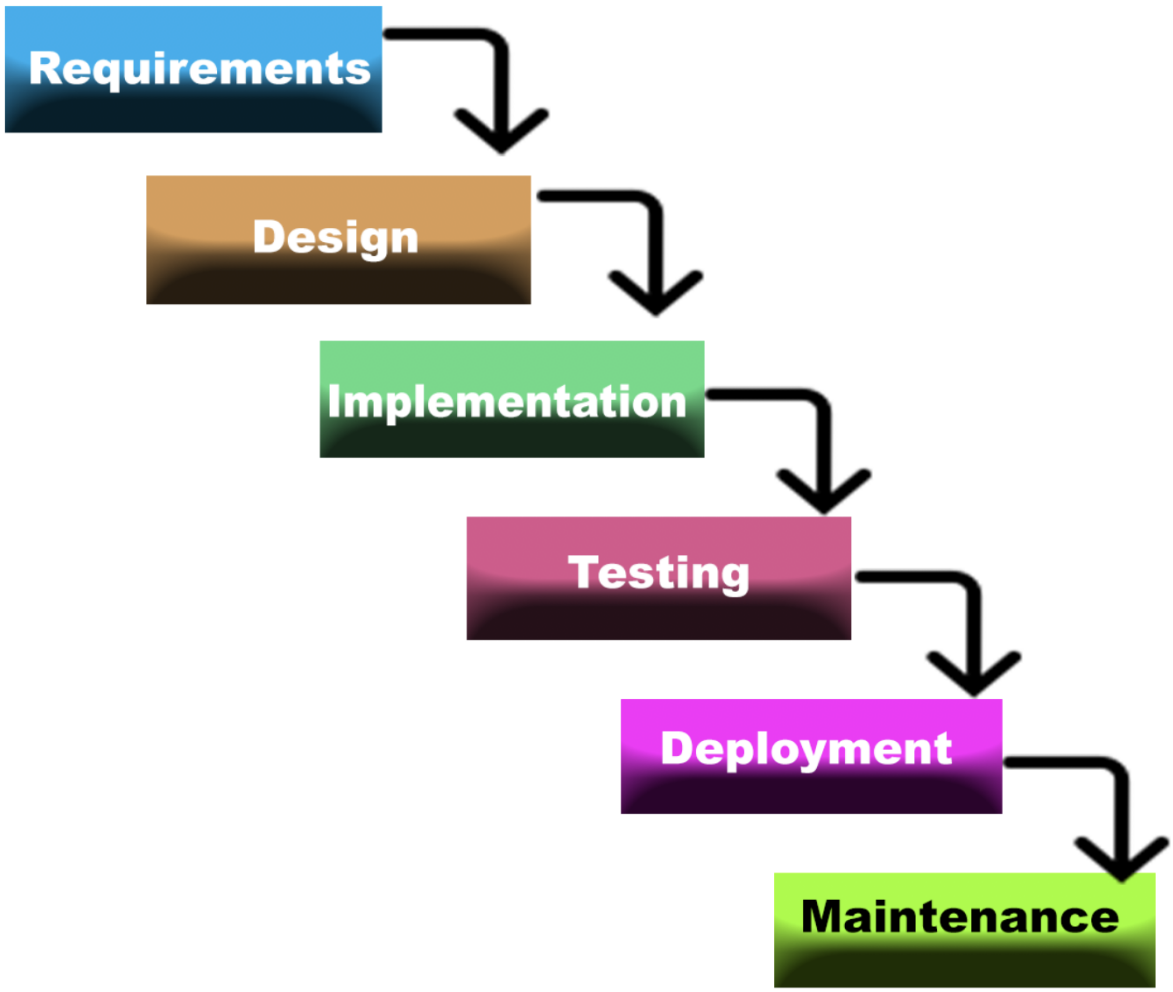


Figure 2.

The above diagram Figure 2 designed in Adobe Photoshop CC 2018.

The sequential phases in Waterfall model are −

- **Requirement:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **Design**: The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in

specifying hardware and system requirements and helps in defining the overall system architecture.

- **Implementation**: With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Testing**: All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment:** Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model Advantages

1. Simple and easy to understand.
2. Helps to plan and schedule the project.
3. Each phase has specific deliverable and review process.
4. Waterfall model works well for smaller projects where requirements are well defined.
5. Each phase completed one at a time.
6. Easy to arrange tasks process and results are well documented.
7. Verification at each stage ensures early detection of errors/misunderstanding.

Waterfall Model Disadvantages

1. Once the application is in testing its difficult to go back and change something that was not well-thought out the concepts stage.
2. Not a good model for complex and object oriented projects.
3. Poor model for long going projects
4. Not suitable for the projects where requirements are at a moderate to high risk of changin.
5. Required more time, in addition to the detailed plan.

After weighing the advantages and disadvantages of the waterfall model we decided it was too inflexible to use on the project and decided to continue looking for a more suitable development model.

V-Model

V-model is an SDLC model, the execution of processes take place in sequential manner in v-shape.
V-Model much like waterfall model The process of V-model specifically specifies a series of linear stages that should occur across the life cycle, one each at a time till the system is build. Understanding the model can be more challenging for everyone on a team. V-Model represents several stages that will be passed through during the software development life cycle. The stage begging is at the top-left and working, over time toward the top-right shown on figure 3.
The stages represent a linear progression of development similar to the waterfall model.

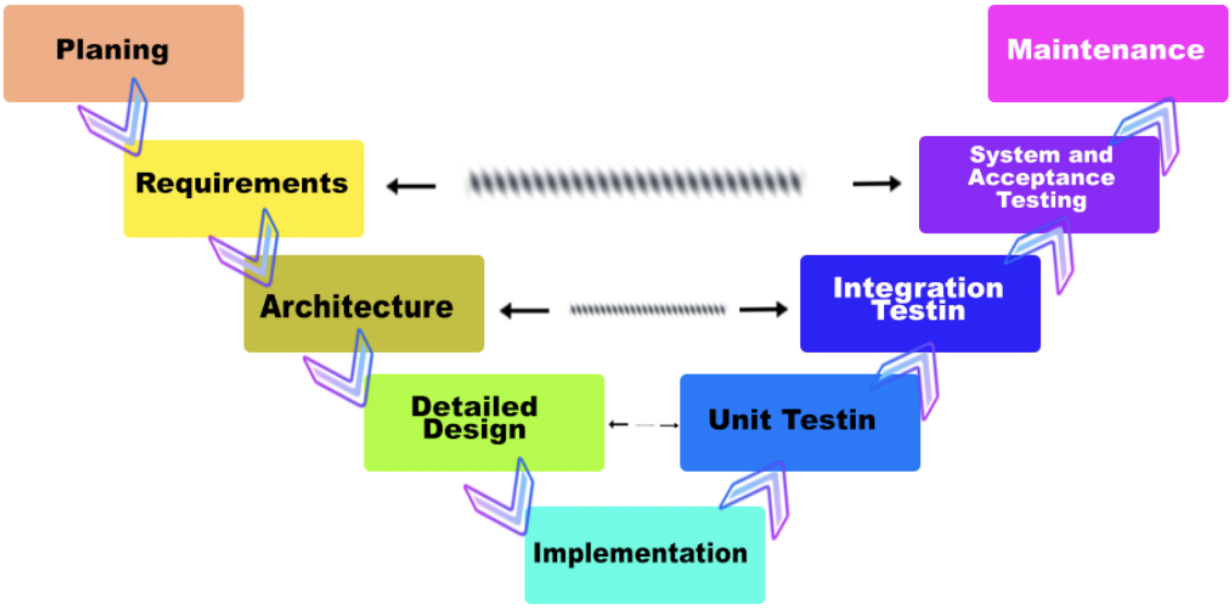


Figure 3.
The above diagram Figure 3 designed in Adobe Photoshop CC 2018.

V-Model Advantages

1. Simple and easy to read
2. Each phase has specific deliverables
3. Better Model over waterfall model, since it has higher chances of success in compilation of a system/project.
4. Works well where the requirements are easily understood.

V-Model Disadvantages

1. The most problematic aspect to the V-Model is its inability to adapt to any necessary changes during the development life cycle. For example, an overlooked issue within some fundamental system design, that is then only discovered during the implementation phase, can present a severe setback in terms of lost man-hours as well as increased costs.

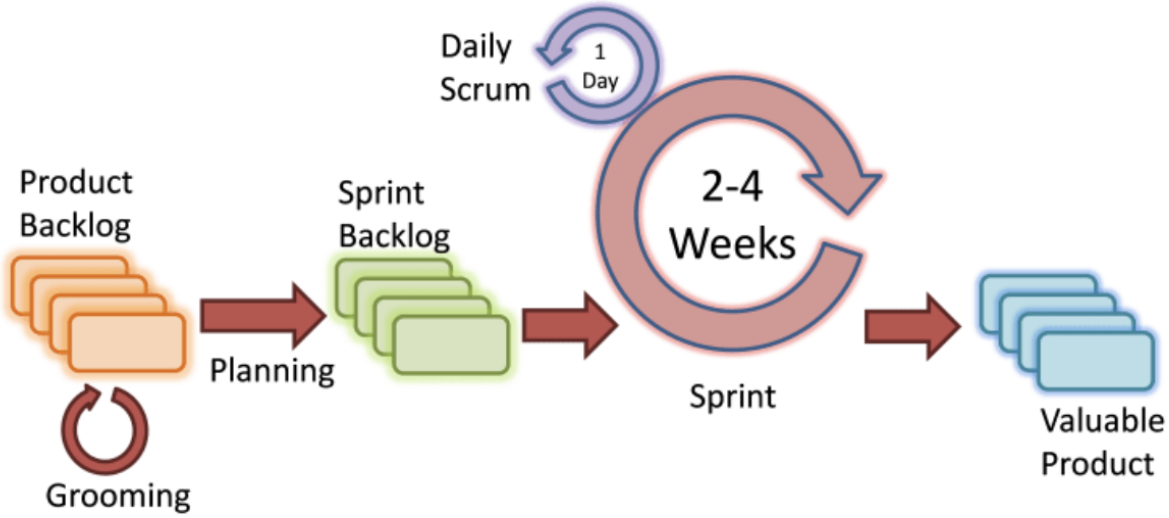
2. Required more time and costly in addition to detailed plan
3. Adjusting scope is difficult and expensive
4. During testing the model will not provide a clear path for problems that's been found.

While it was an improvement over the waterfall model we ultimately decided not to use the V-model and instead kept looking for a model tha better suited the projects needs.

Agile Model

The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability. Agile model is based on iterative and incremental development, where requirements and solutions develop through collaboration between cross-functional teams.
Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements, the tasks are divided into small time frames to deliver specific features for release.

Here is the graphical illustration of the agile model.



Scrum Agile Model

Figure 4.

Agile uses an adaptive approach, where there is no detailed planning and there are clarity on future tasks only in respect of what features need to be developed and delivered. Agile have feature driven development and the team adapts to the changing products requirements dynamically. They product is tested frequently due to minimize the risk of any major failures in future. The testing process is through the release iterations.
Agile method may not always be suitable for all products

Agile Advantages

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Gives flexibility to developers.
- Face to face communication and continuous input in the system from consumer.

Agile Disadvantages

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- Documentation at late stages.

After weighing the pros and cons of agile and comparing it to the other models we ultimately decided that it was the most suitable option for our project due to its flexibility, ability to respond to changing requirements and focus on team collaboration.

Project Plan

For the first few weeks of the project our plan was to focus on determining the high-level functionality of the project. This included the Use Case Descriptions, Use Case Diagrams and prototype GUIs.

In order to accomplish these tasks we agreed to meet at least once a week to discuss and work on the project. We also used the labs every 2 weeks share our work and obtain feedback and advice from JJ.

Short Term Goals

- Set up version control system (Github)
- Set up system for sharing report (Google Docs Shared folder)
- Agree on means of communication
- Research structure of previous projects
- Research projects with similar functionality
- Outline core functionality and stretch goal requirements
- Identify sources of added value outside project description

Project Roles

1.	Project Manager	Maihanjan Abdul Qayoom
2.	Documentation Manager	Maihanjan Abdul Qayoom
3.	Business Analyst/Requirements Engineer	Conor Moloney
4.	Architect	Artem Semenov
5.	Systems Analysis	Artem Semenov
6.	Designer	Brian Dooley
7.	Technical Lead	Artem Semenov
8.	Programmers	All
9.	Tester	Conor Moloney
10.	Dev Ops	Brian Dooley

Use Cases

Use Case 1	Register
Actor Action	System Response
1 – Client clicks register	2 – User details required
Alternative Route	
1(a) – Staff registers client 1(b) – Details already on the system	2(a) – 2(b) – If credentials already associated with an account ask for different details

Use Cases

User

Non-functional Requirement: Security

Password must be hashed, and all private information stored securely in a database.

Non-functional Requirement: Scalability

The system should be able to handle large amounts of accounts being created

Use Case 2	View account
Actor Action	System Response
1 – Client views account details	1 – Clients account details displayed
Alternative Route	

Use Case 3	Login
Actor Action	System Response
1 – Client enters Username and Password	2 – Credentials are checked. Main screen is shown
Alternative Route	
1(a) – Client enters incorrect details	2(a) – Client credentials fail, error message displayed

Non-functional Requirement: Security

Clients password needs to be encrypted i.e. hashed passwords.

Use Case 4	Logout
Actor Action	System Response
1 – Client selects logout option	2 – Client returned to homepage

Alternative Route	

Non-functional Requirement: Security

System remembers user data, allowing for easier login in the future.

Use Case 5	**Create Order**
Actor Action	System Response
1 – Client creates an order 3 – Client adds different options 5 – Client enters payment details	2 – System displays food options 4 – System adds up options and displays total price 6 – System validates payment
Alternative Route	
1(a) – Client orders over phone 3(a) – Client adds options 5(a) – Client chooses payment options	2(a) – Staff offers food options 4(a) – Staff offers payment options 6(a) – Staff finishes payment

Non-functional Requirement: Usability

Creating an order should be straight-forward and easy to use to not confuse the user.

Use Case 6	**Create review**
Actor Action	System Response
1 – Client creates a review	2 – System adds review to list of reviews
Alternative Route	

Use Case 7	**View reviews**
Actor Action	System Response
1 – Client selects view reviews option	2 – System displays a list of all reviews
Alternative Route	

Staff

Use Case 8	**Check order history**
Actor Action	System Response
1 – Staff selects order history	2 – System displays order history
Alternative Route	

Use Case 9	View reviews
Actor Action	System Response
1 – Staff selects view reviews	2 – System displays list of reviews
Alternative Route	

Use Case 10	Check order status
Actor Action	System Response
1 – Staff selects check order status	2 – System displays the order status
Alternative Route	

Non-functional Requirement: Performance

The order status should appear fast and be responsive.

Use Case 11	Change order status
Actor Action	System Response
1 – Staff selects change order status	2 – System changes state of order
Alternative Route	

Use Case 12	Place order for customer
Actor Action	System Response
1 – Staff enters order for client	2 – System enters order of client
Alternative Route	

Use Case 13	Check stock
Actor Action	System Response
1 – Staff selects check stock	2 – System retrieves list of stock
Alternative Route	

Non-functional requirement: Security

Only a Manager account can check stock

Manager

Use Case 14	Manager can do staff use cases
Actor Action	System Response
1 – All staff options are Manager options	2 – Manager can carry out all Staff options
Alternative Route	

Use Case 15	Award points to user
Actor Action	System Response
1 – Manager adds points to user	2 – System adds points to client account
Alternative Route	

Use Case 16	Create staff account
Actor Action	System Response
1 – Manager creates a staff account	2 – System creates a new staff account
Alternative Route	

Non-functional Requirement: Scalability

The system should be able to handle large amounts of accounts being created

Use Case 17	View financial data
Actor Action	System Response
1 – Manager selects financial data	2 – System retrieves financial data
Alternative Route	

Non-functional requirement: Performance

Manager should be able to set parameters so that certain stock is automatically ordered when it falls below certain levels

Use Case 18	Order stock
Actor Action	System Response
1 – Manager orders additional stock	2 – System notifies that the order has been made
Alternative Route	

Automatic

Use Case 18	**Stock check and order**
Actor Action	System Response
1 – User carries out order	2 – System returns information on stock
Alternative Route	

Use Case 19	**Add loyalty points to user account**
Actor Action	System Response
1 – User fulfils an order	2 – System adds loyalty points to User
Alternative Route	

Use Case Detailed Description

Use Case 3		Create an Order
Goal in Context		Food ordered, customer order added to account & payment validated
Scope & Level		Website
Preconditions		1. Customer must be logged in. 2. Order must consist of 1 or more menu items 3. Restaurant must have sufficient stock to fill order
Postconditions		1. Price of order transferred from Customer account to restaurant. 2. Stock is decremented from database 3. New stock ordered if stock amount fell below limit
Success End Condition		Customer receives order, we have order accounted for and payment received
Failed End Condition		We have not sent out the order, client has not spent money
Primary, Secondary, Actors		Client, staff on behalf of customer, payment system, bank
Trigger		Customer initiates an order online
DESCRIPTION	**Step**	**Action**
	1	Client uses website
	2	Client selects food type
	3	Client selects toppings
	4	Client selects sides

	5	Client selects drinks
	6	Client enters payment details
	7	Food ordered
EXTENSIONS	**Step**	**Branching Action**
	2a	Food type unavailable, offer different type of food
	6a	Payment action fails, cannot order without valid payment
VARIATIONS		**Branching Action**
	1	Client may contact by phone
	6	Payment may be on collection

RELATED INFORMATION	3. Create an Order
Priority	Top
Performance	5 minutes for customer details/interaction Less than 1 minute for credit card validation
Frequency	200/day
Channel to Actors	Website GUI
OPEN ISSUES	Credit card is stolen Order terminated during transaction
Due Date	Release 1.0
…any other management information	
Superordinates	User logs in (use case X)
Subordinates	Cancel Order (use case X)

Tactics for Handling Quality Attributes

We decided to use java for this project as it is a modern, object-oriented language which would facilitate modular development using design patterns. Java also provides support for developing intuitive UIs and connecting to a wide variety of databases. In addition to this Java supports portability as Java byte code can be run on any hardware with a Java Virtual Machine (JVM).

We used a MySQL database to store the data. This provides security through MySQL's in-built security features and supports performance by allowing for caching and indexing of queries and results. MySQL also supports scalability and has been used for storing vast amounts of data by large international companies such as Facebook and Twitter. We accessed the MySQL database in our project using the Java Database Connector (JDBC) which provided a number of powerful and useful options for accessing and manipulating the database.

While we initially used Java Swing to develop the GUI we decided to instead use JavaFX later in the project due to its ease of use and powerful features. Using FX we were able to create an intuitive user-friendly UI that provides the user with all the information they need to make use of the system.

Functional Use Case Diagram

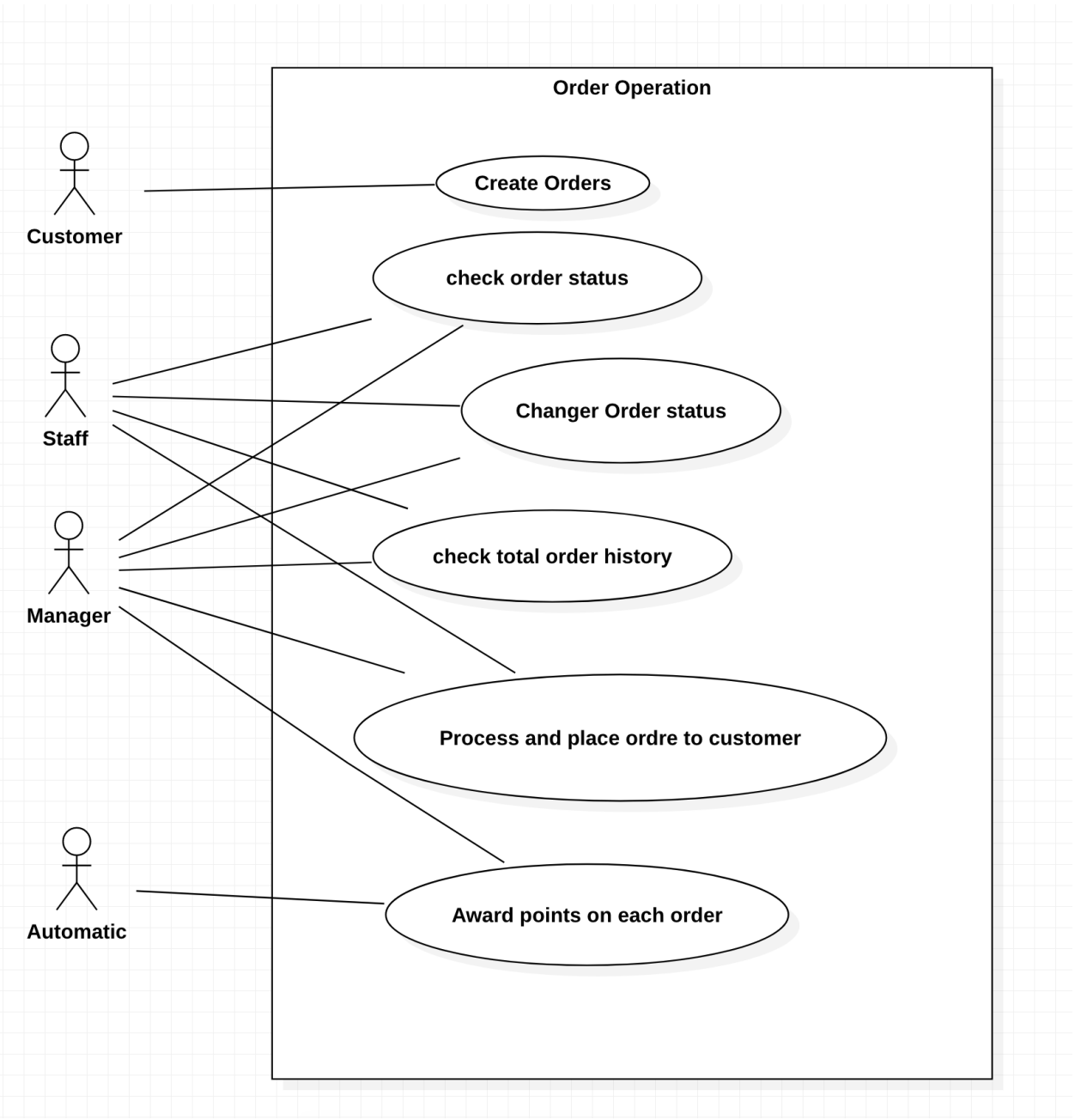

Order Operation
Customer
Staff
Manager
Automatic
Create Orders
check order status
Changer Order status
check total order history
Process and place ordre to customer
Award points on each order

Figure 5.


System operation
Account Register
View Account Details
«incldues»
login
Logout
Create Order
Create reviews
View previous reviews
check or change order status
check Stock
Check total ordrer hostory
Award points to users order
Can order stock
USER
Staff
Manager

Figure 6.

GUI Prototypes

Login System

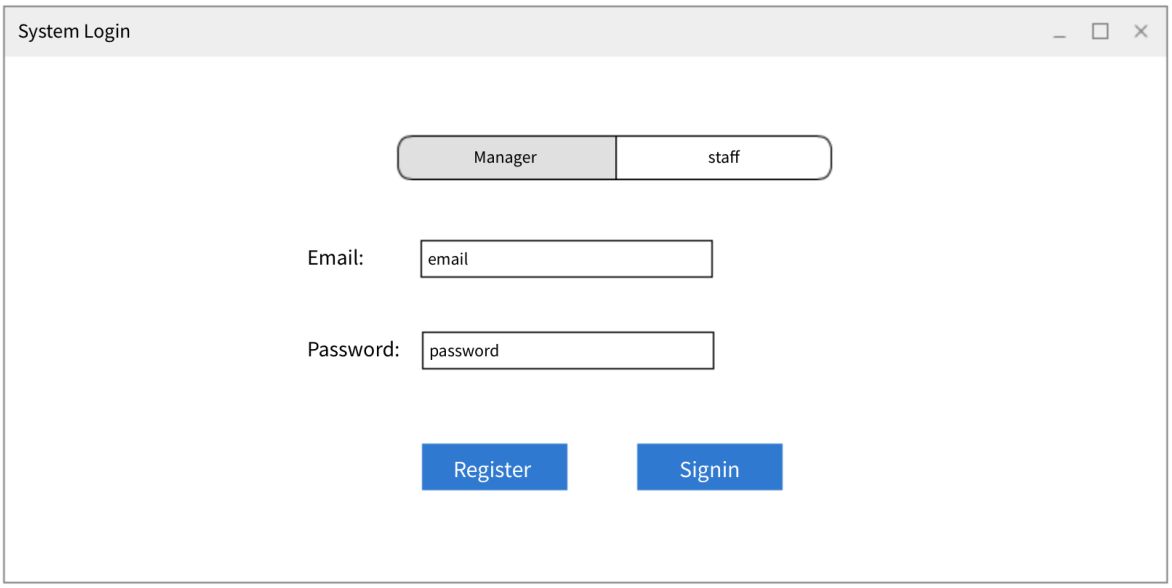


Figure 7.

Main Window

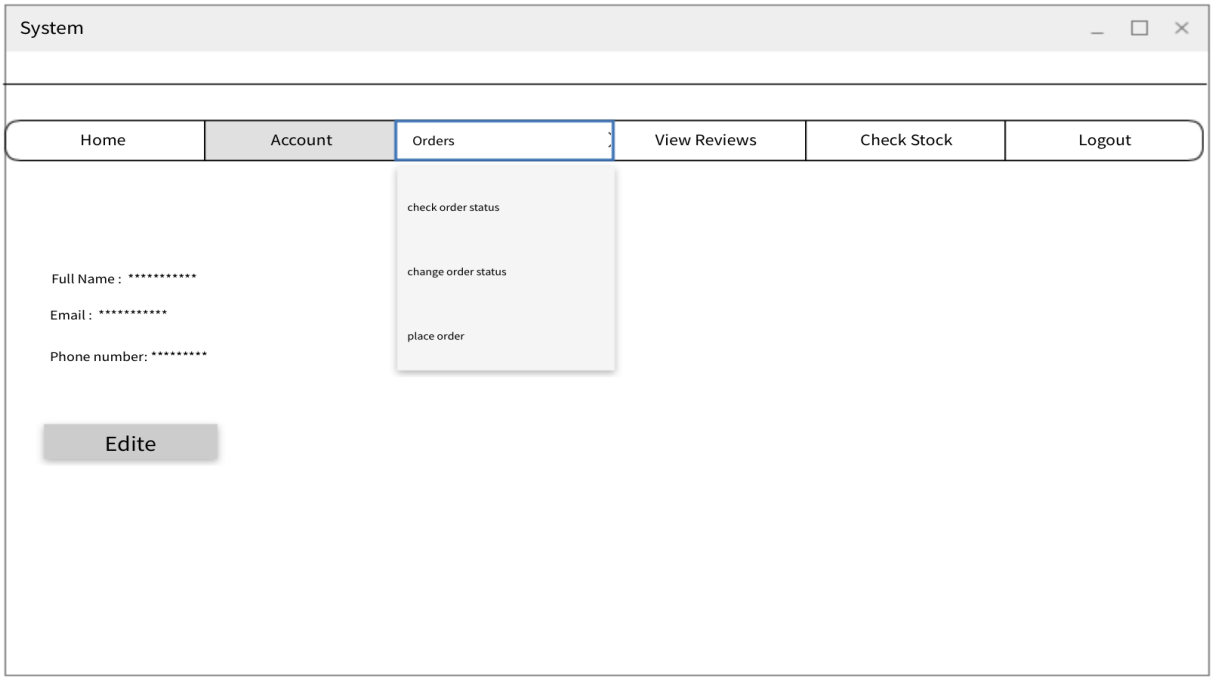


Figure 8.

System Architecture

We planned to use the model-view-controller (MVC) design pattern when implementing our project. This involves splitting the architecture of our system into 3 layers: the UI Layer, Business Layer and Database/Data Persistence Layer. The MVC pattern decouples these major components allowing for easier and more efficient code reuse and parallel development. The Database layer is the Model, the UI layer is the View and the Business Logic layer is the controller.

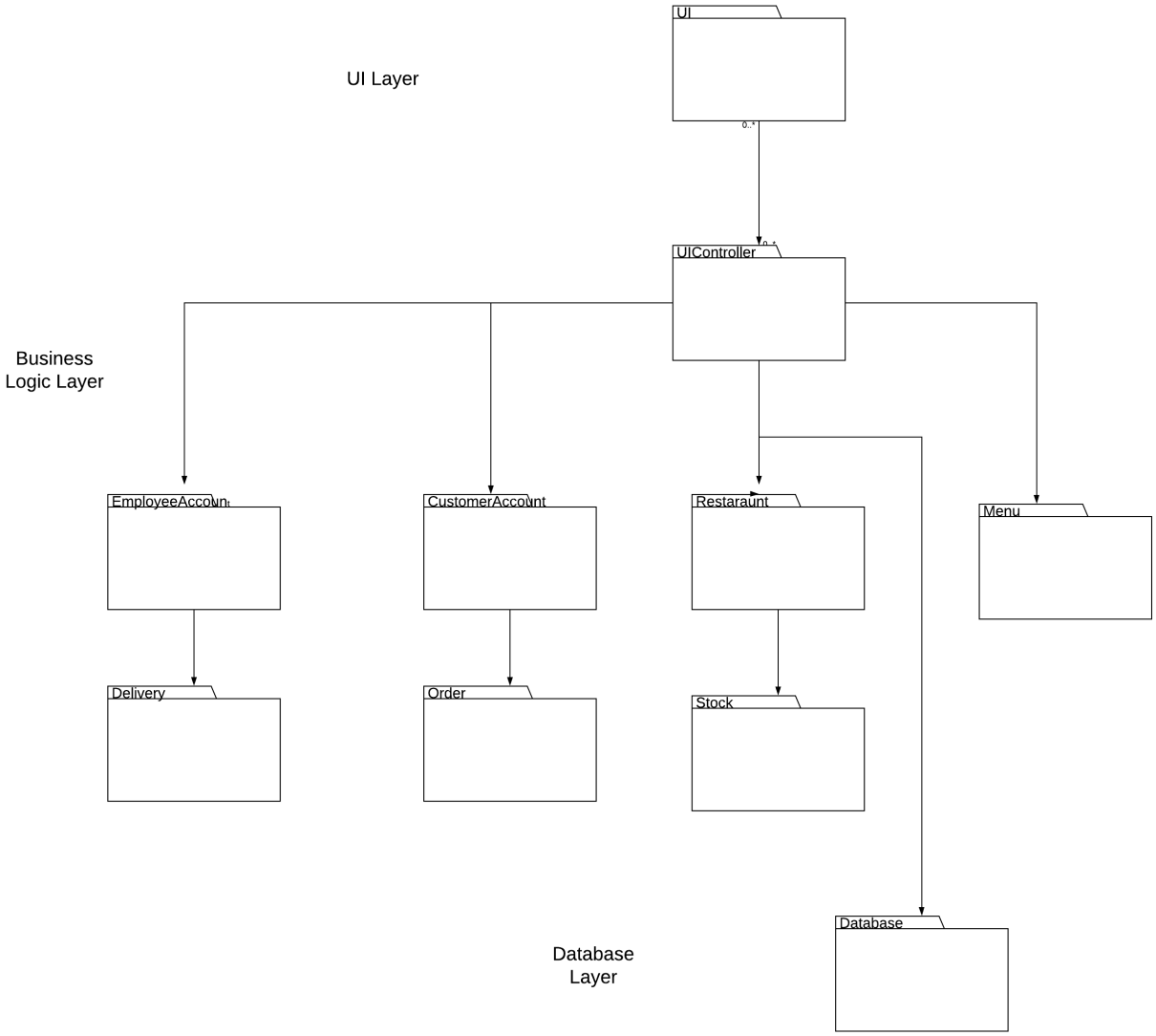


Figure 9.

Analysis

During analysis we identify the software elements necessary to satisfy the functional requirements. Analysis is also sometimes referred to as Requirements Analysis or Conceptual Class Modelling. A good class model consists of classes of enduring domain objects and is not dependent on currently available technologies. One technique for creating the Analysis Class Model is to model the dynamics of each use case. The dynamics (behaviour) of the use case is determined by the set of classes taking part in that use case.

The technique we used to identify candidate classes from Use Case Descriptions is Data Driven Design (DDD), which performs key domain abstraction using the noun identification technique.

Candidate Objects
1. User
2. EmployeeAccount
3. CustomerAccount
4. StockItem
5. MenuItem
6. Order
7. Review
~~8. Username~~ Attribute
~~9. Password~~ Attribute
10. Delivery
~~11. Security~~ Irrelevant
~~12. Performance~~ Irrelevant
13. Stock
14. StockOrder
15. Restaurant

User: Interface to be implemented by Employee, Customer and Manager. Each user must have a name and ID.
StockItem: One item of stock in the restaurants inventory. The stock is used to make orders for customers.
MenuItem: Item appearing on the menu which can be ordered by a customer. Contains price.
Order: Ordered placed by a customer. Should contain 1 or more MenuItems.
Review: Review left by a customer.
Delivery: Delivery of an order to a customer.
Stock: Total stock in the restaurants inventory. Made up of a list of StockItems.
StockOrder: Order placed by restaurant for more stock. Should consist of a list of StockItems.
Restaurant: Represents the takeaway restaurant. Contains information about the current stock and receives money when a customer places an order.

Interaction Diagrams

Communications Diagram - Customer Order

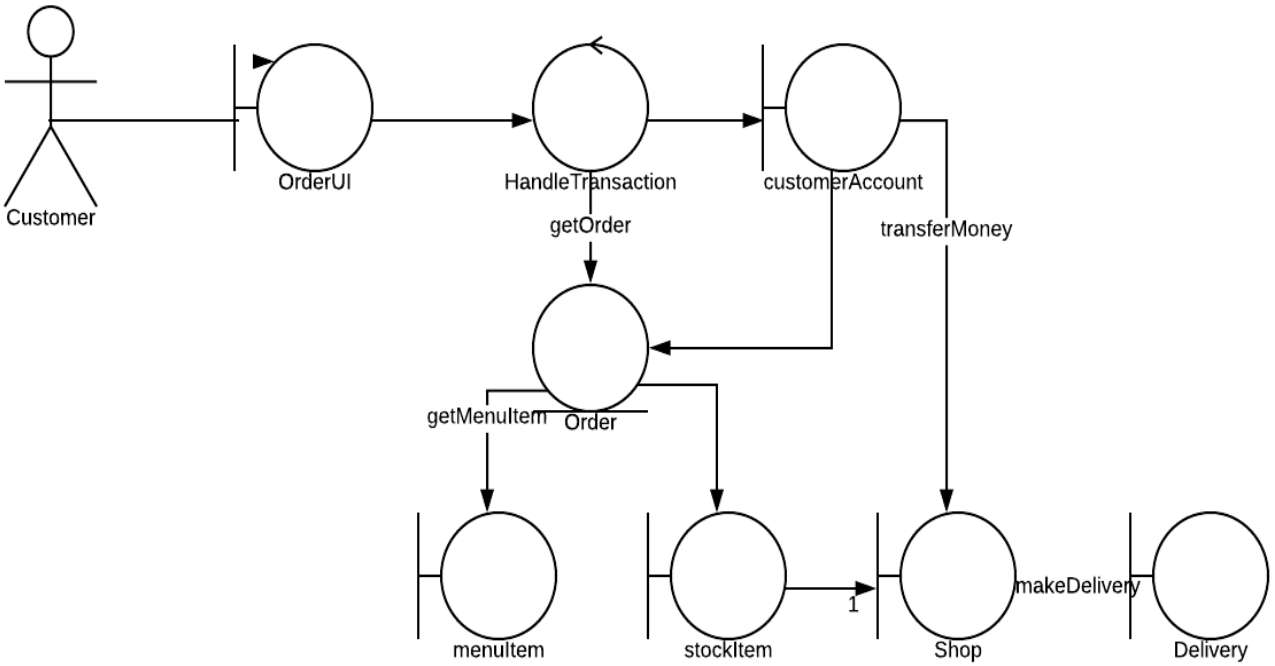


Figure 10.

Collaboration Diagram

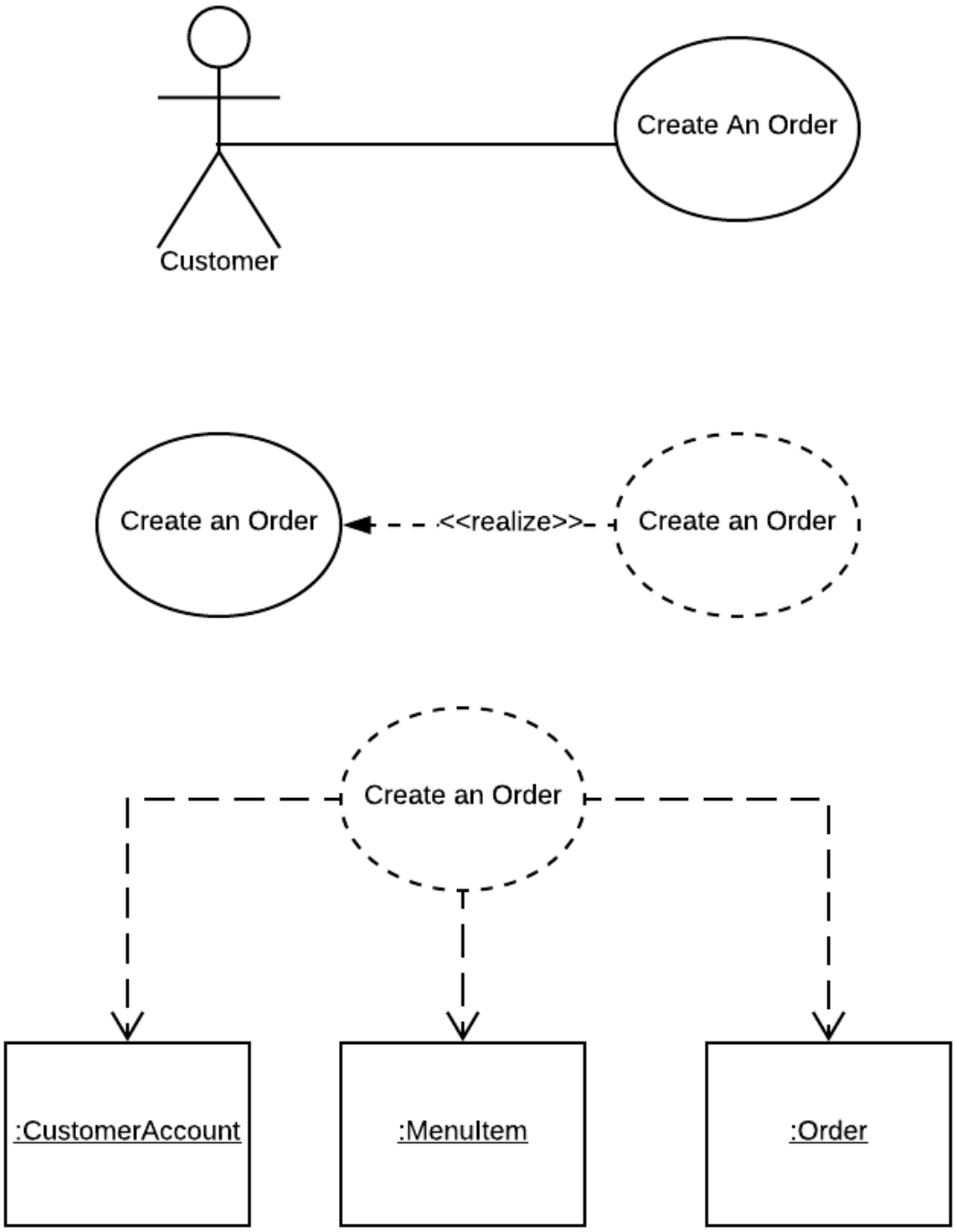


Figure 11.

Entity-Relationship Diagram

An entity-relationship diagram is a type of flow-chart which illustrates how “entities” such as people, objects and concepts relate to each other. They mirror grammatical structure with an entity as a noun and a relationship as a verb.

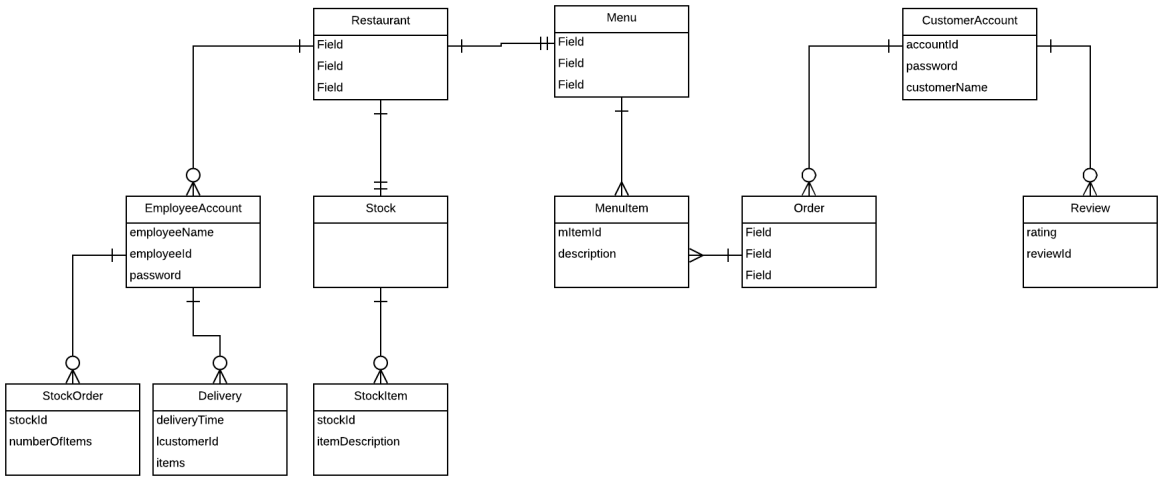


Figure 12.

Database Diagram:

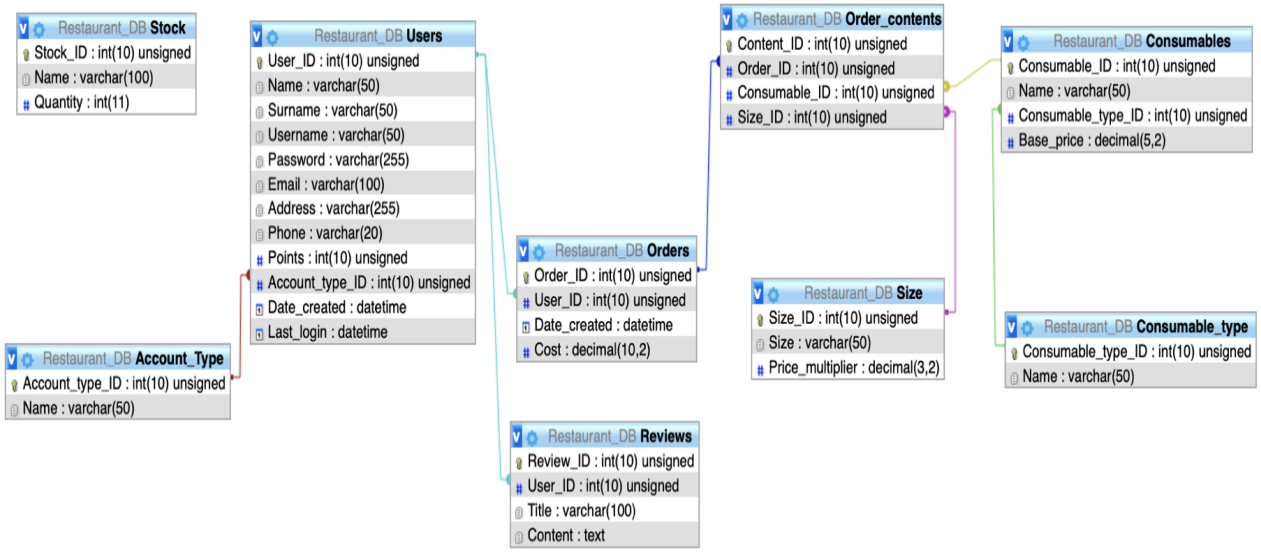


Figure 13.

Code

Use Cases Implemented

While time constraints prevented us from being able to implement all the use cases we initially planned we were able to implement a number of key use cases.

We implemented use cases which allow the user to register a new account and log in using that account. When a user attempts to register a new account a number of checks are performed ensuring it's validity. These include checking that the user has provided all the necessary information, that the user information provided is unique, that the password given matches the confirmed password. If these checks succeed a new account for the user is created in the database.

From there a user can log in to their account using the information they provided to register. The log-in process will check the information provided by the user against the database and if it is correct allow the user to log-in.

After logging in a user is able to view the menu and select items to order from a number of categories (food, sides, drinks). When the user is finished selecting items for their order they can checkout which will calculate the total cost of their order and apply any relevant discounts.

Lines of Code per Team Member

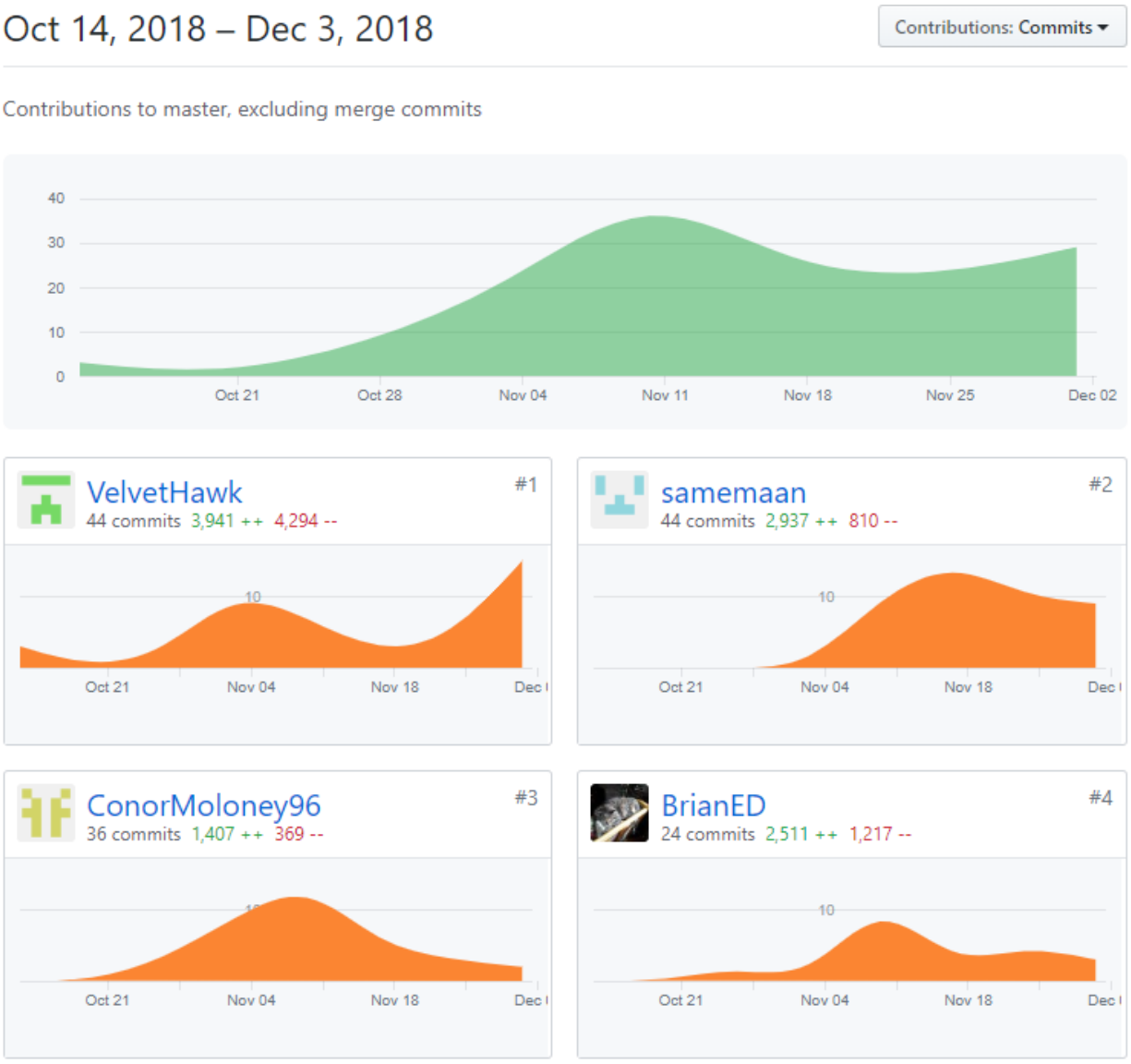


Figure 14.

GUI Screenshots

Start Window

Figure 15.

Registration Window

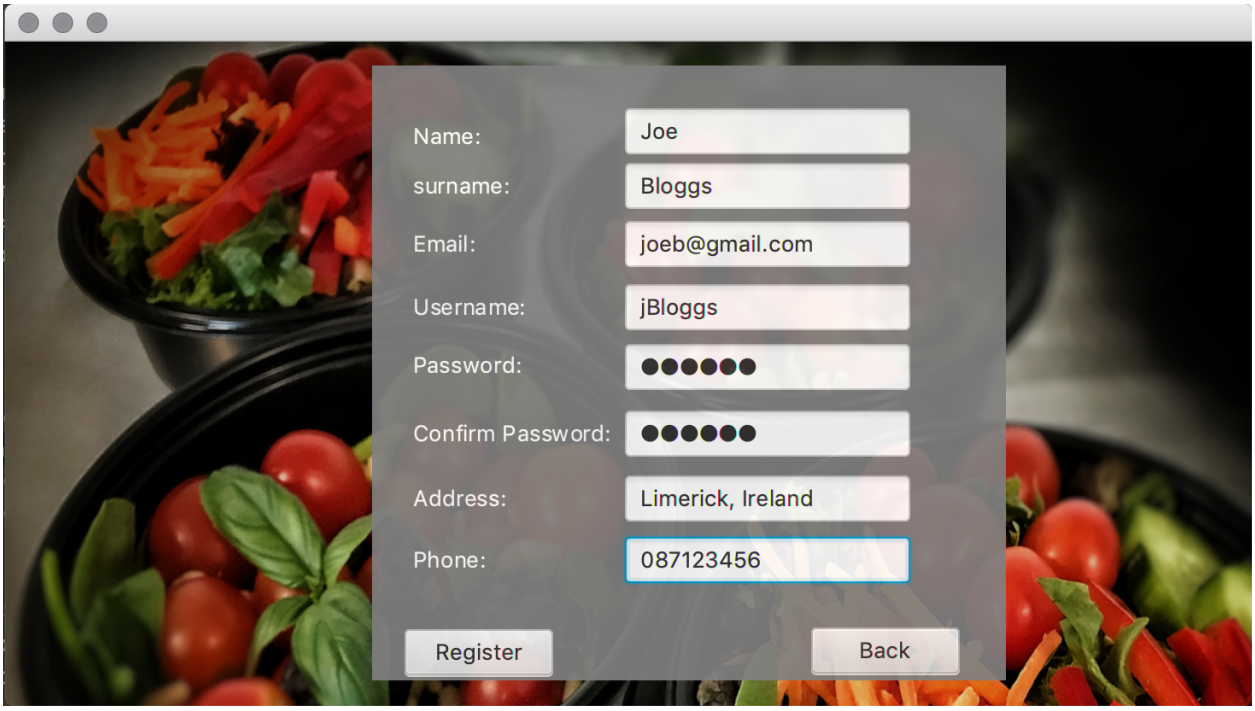


Figure 16.

Registration Window successful

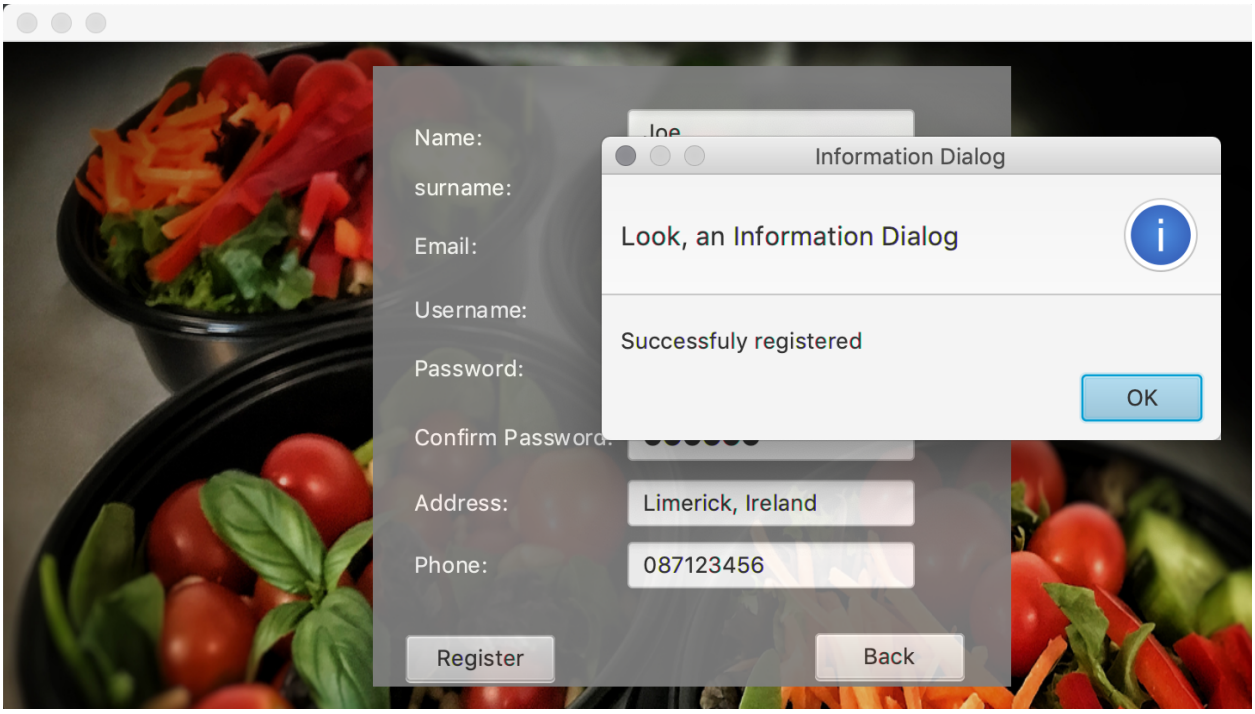


Figure 17.

Registration Window fail

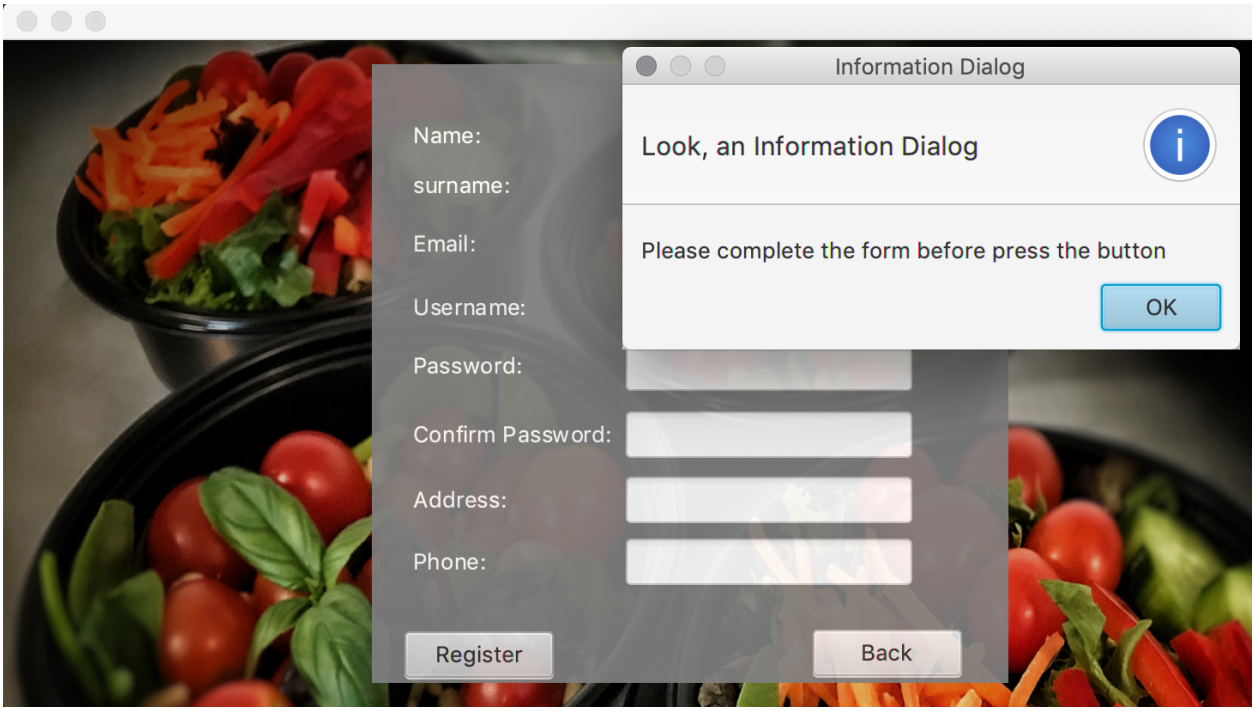


Figure18.

Login window Fail

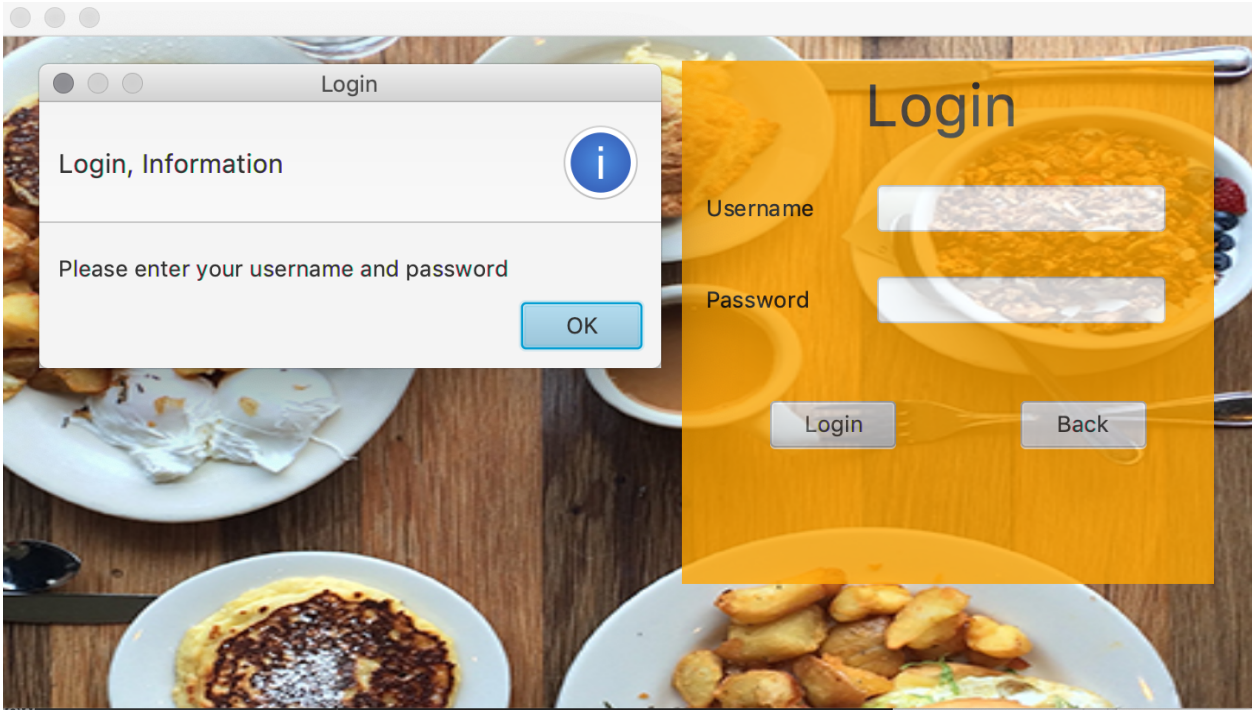


Figure 19.

Login Window

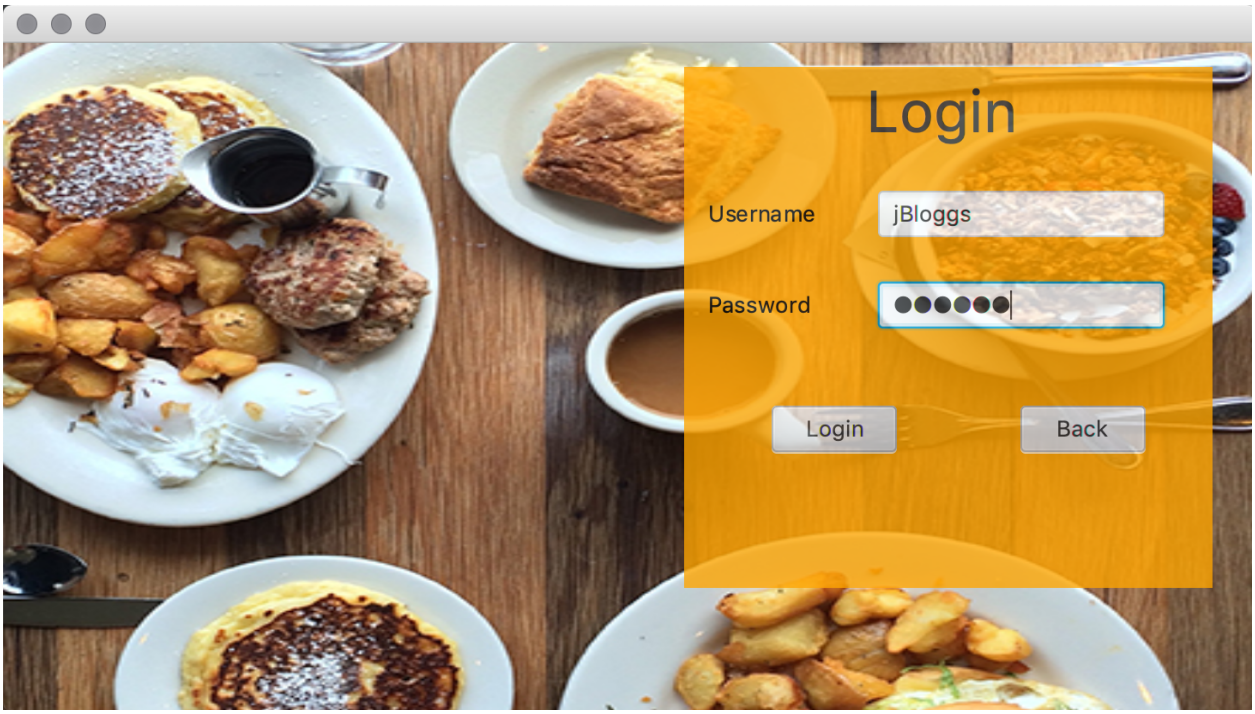


Figure 20.

Menu

Figure 21.

Make an order window:

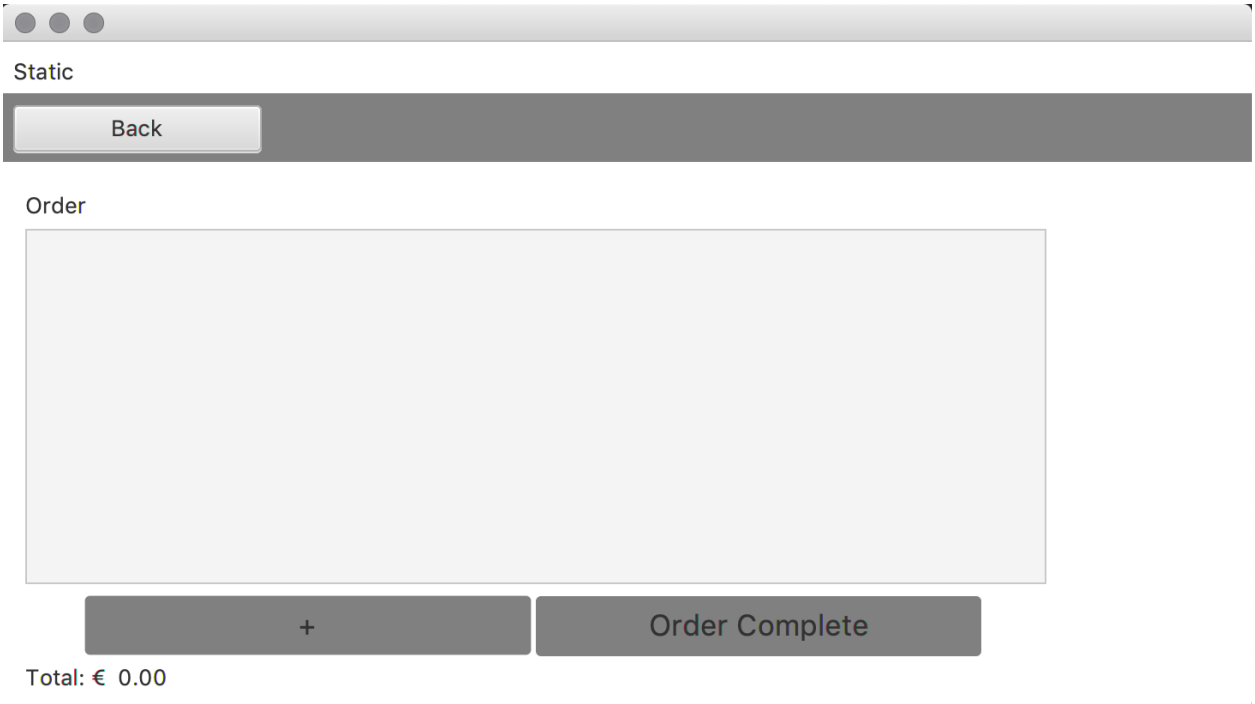


Figure 22.

Choice Selection window

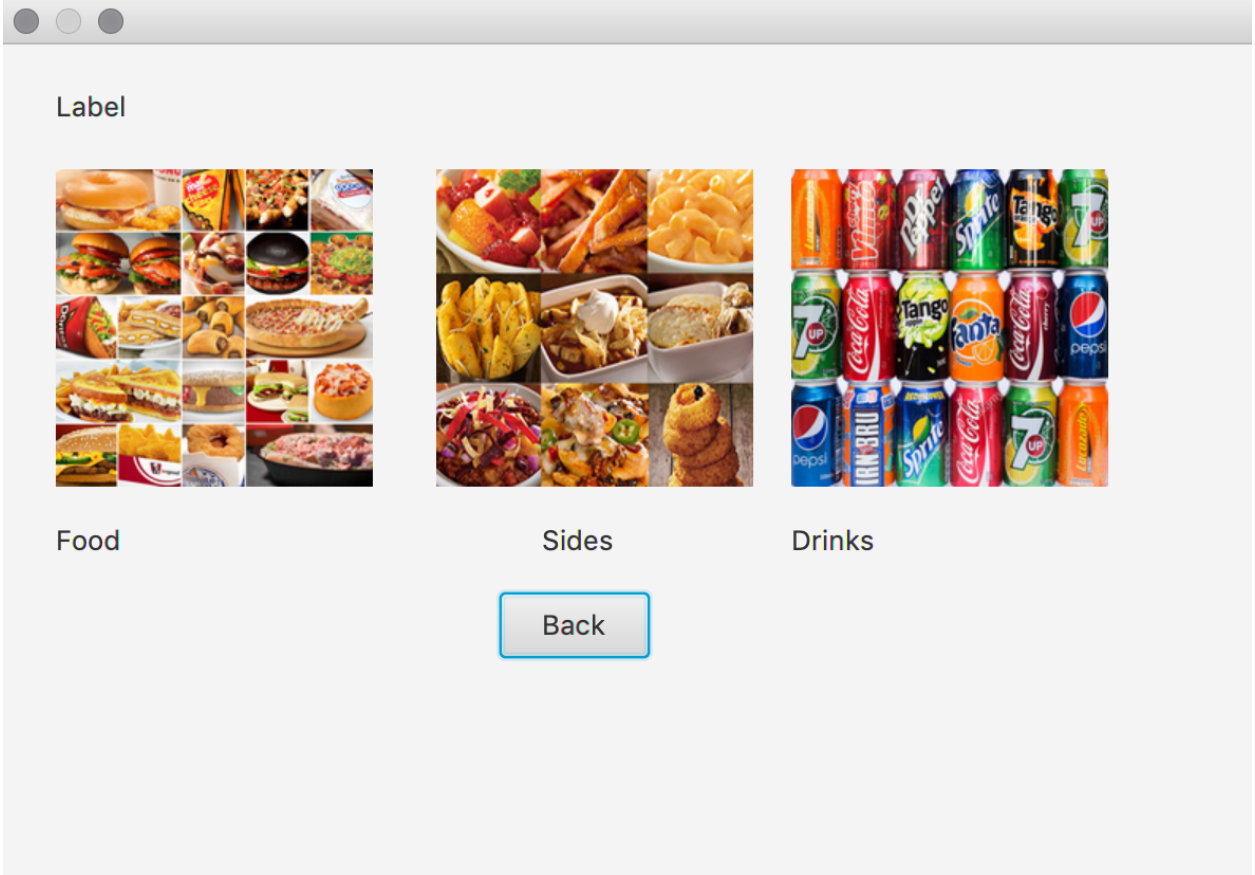


Figure 23.

Food window

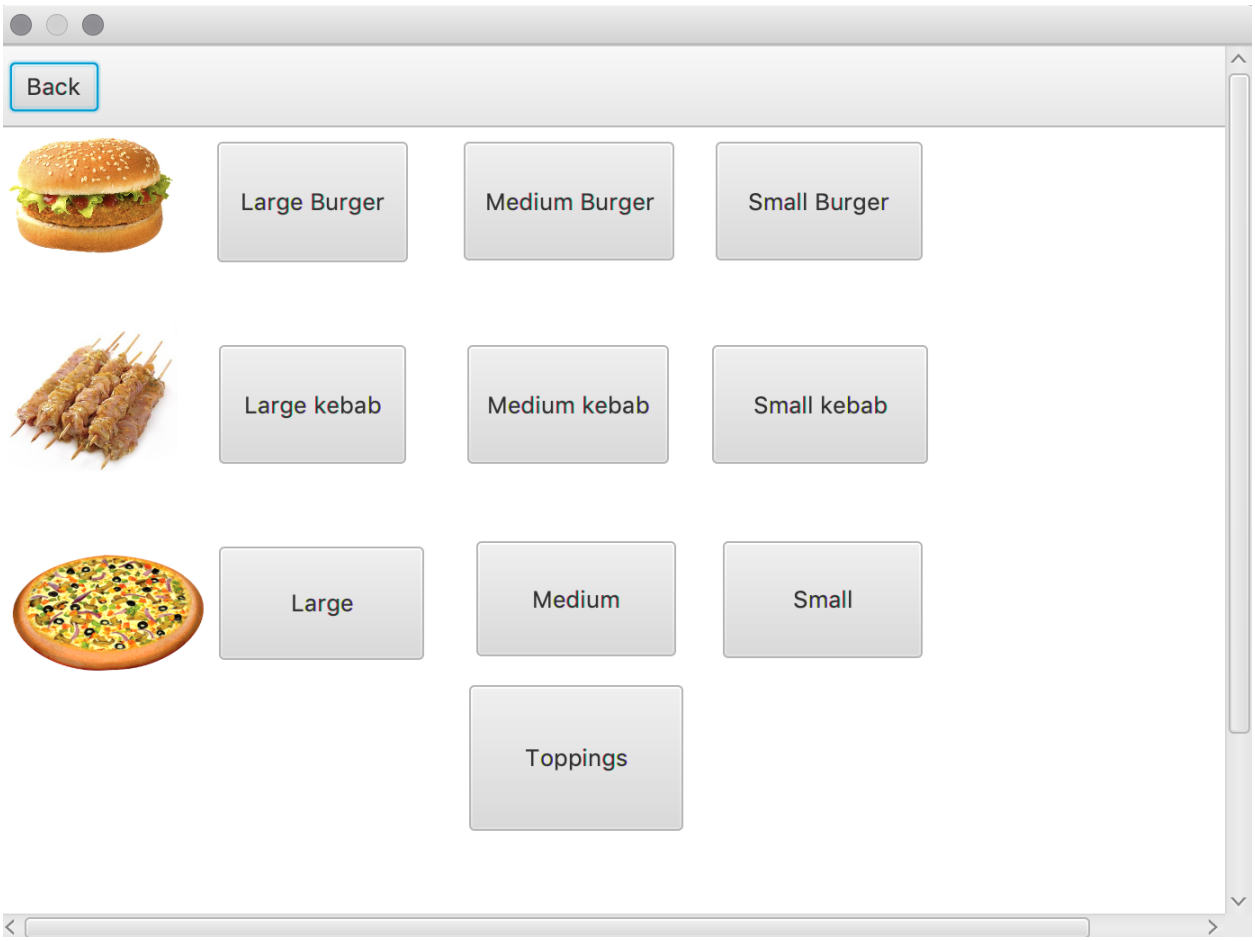


Figure 24.

Topping Window

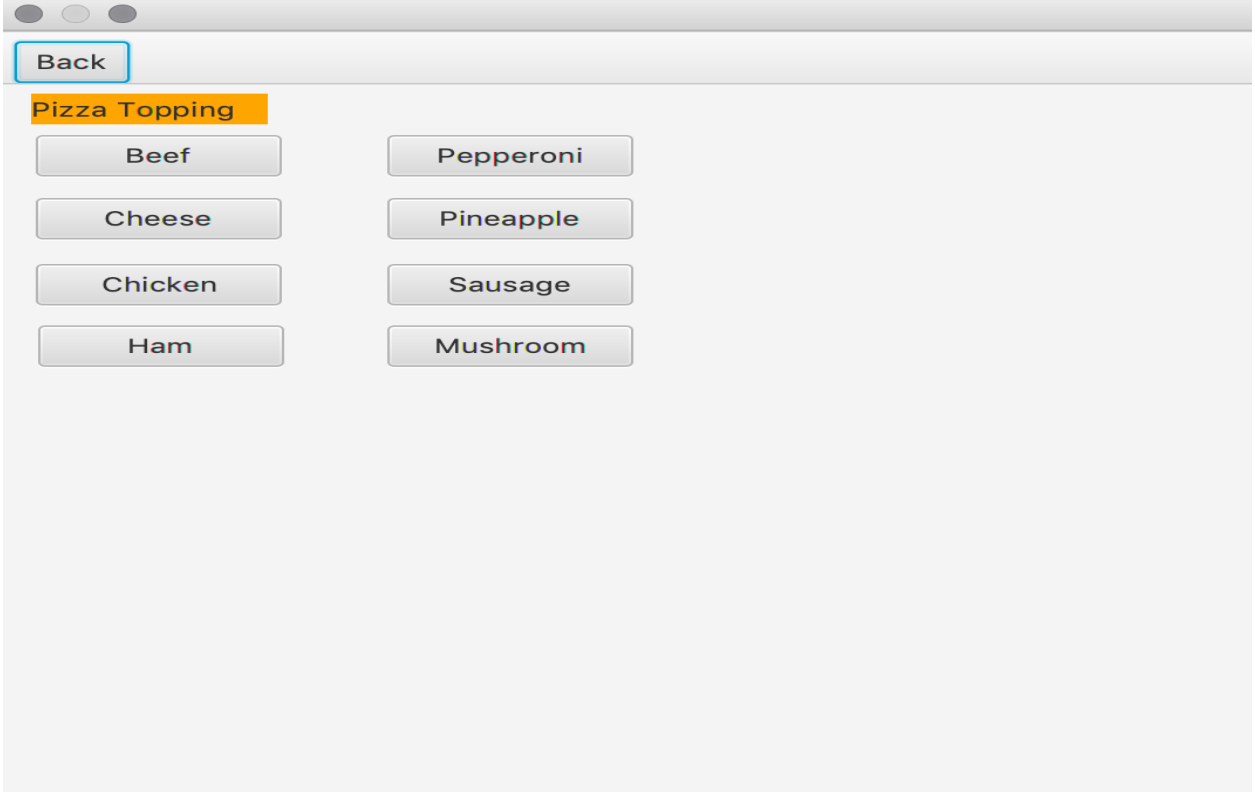


Figure 25.

Side Food Window

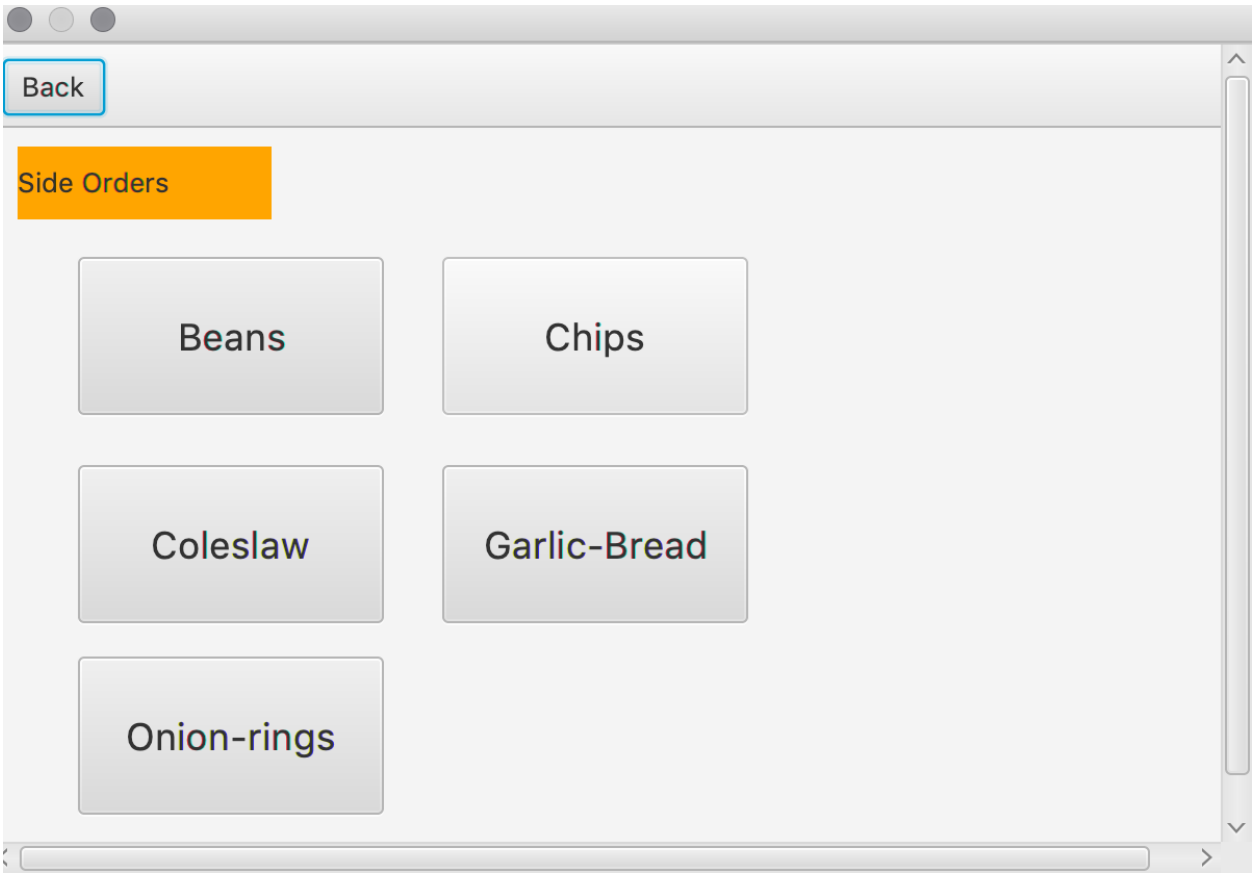


Figure 26.

Drinks window

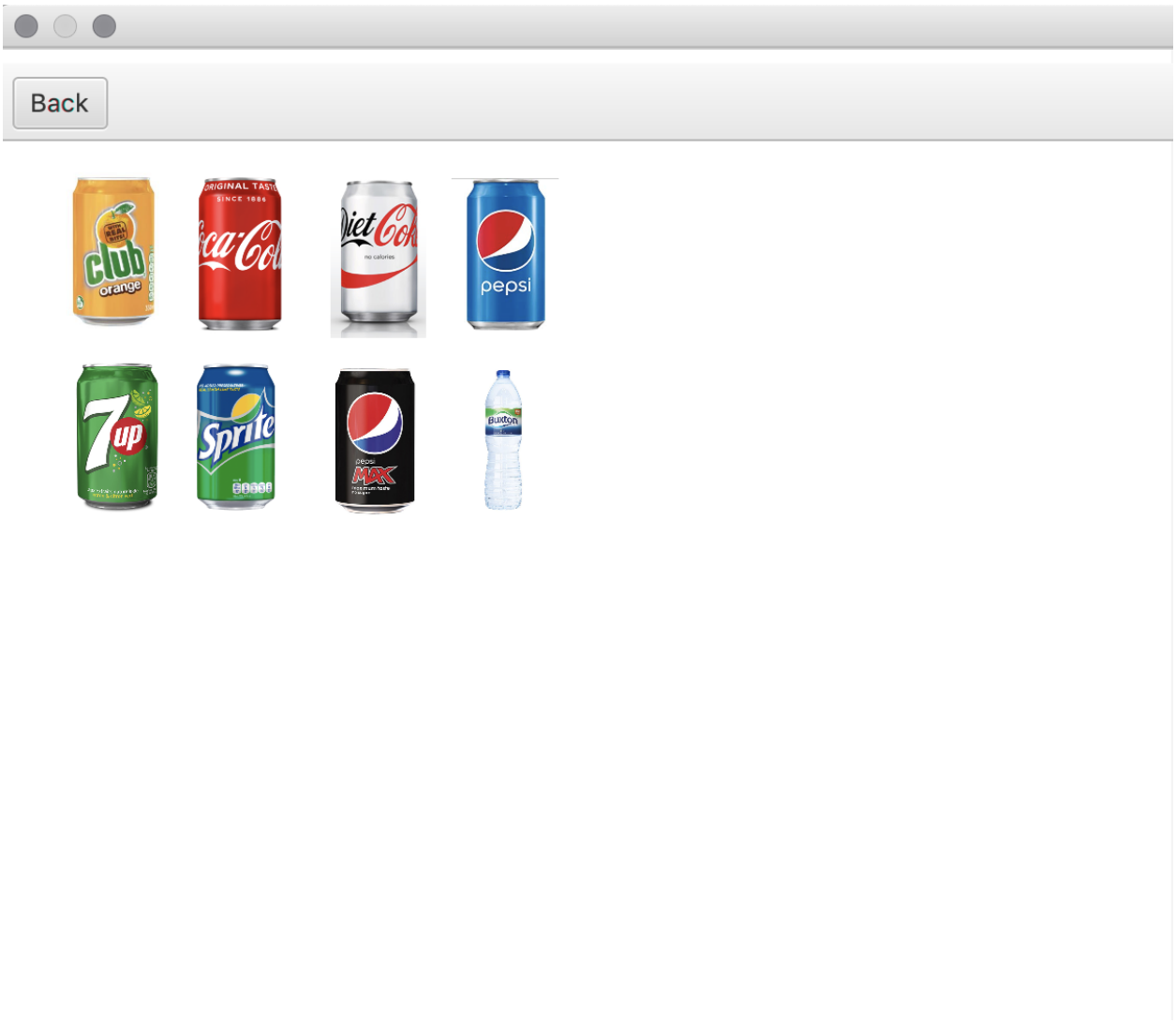


Figure 27.

Make an order with Updated invoice window:

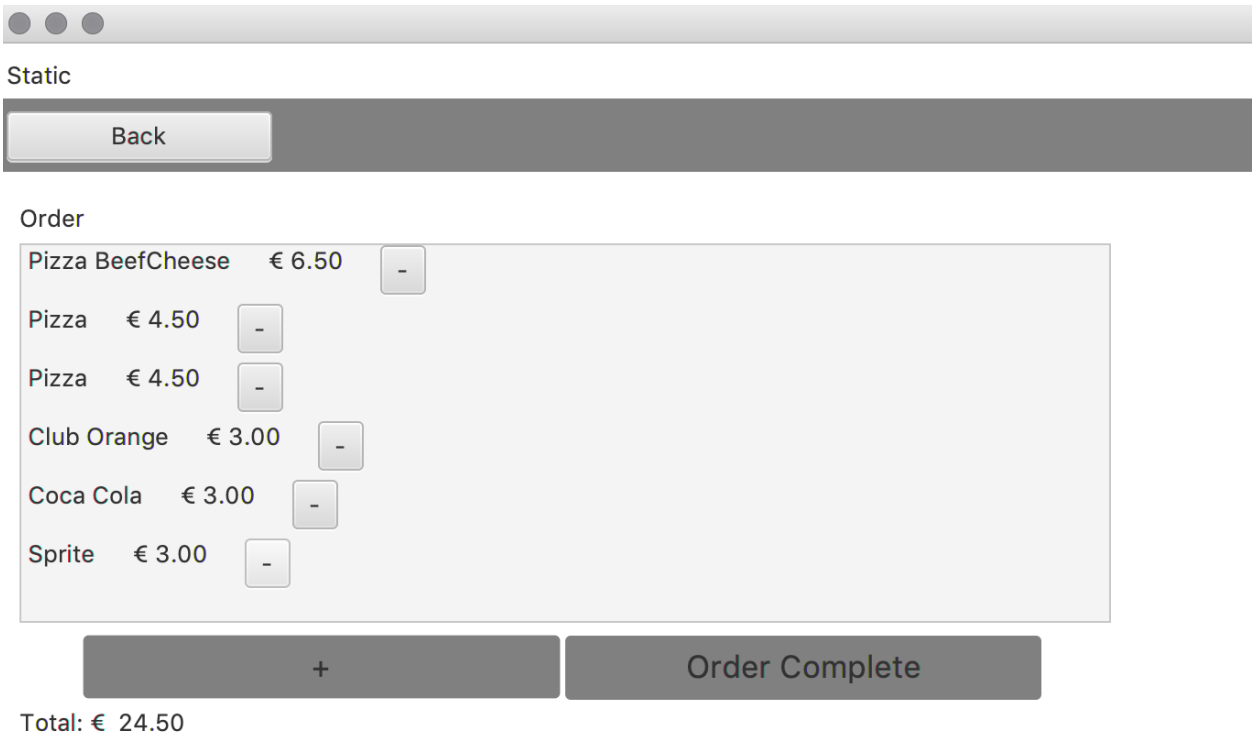


Figure 28.

Design Patterns

For this project we used a number of design patterns to support the development of re-usable, extensible code. Our use of design patterns was heavily based on “Design Patterns: Elements of Reusable Object Oriented Software” by Gamma, Helm, Johnson et. al (i.e. the “Gang of Four”).

We also made use of architectural patterns to structure our project at a higher level. The 2 main architectural patterns we used were Model View Controller (MVC) and Database Access Object (DAO).

Abstract Factory Pattern

The abstract factory pattern works around a super factory which creates other factories. This factory is also called a factory of factories. This is a creational design pattern as it facilitates the creation of objects. It provides a way to encapsulate a group of individual factories that have a common theme with specifying their concrete classes. This was used in the creation of drink, food, side and toppings factories which called new classes for each type of food, drink, side and topping.

Decorator Pattern

The decorator pattern allows new functionality to be added to an existing object without changing its structure. This pattern creates a decorator class which wraps around the original class allowing it's signature to remain intact while the decorator provides additional functionality. This is an example of a structural design pattern as the decorator is used to obtain new functionality from an existing class.

State Pattern

State pattern allow an object to alter its behavior when it's internal state changes. The object will appear to change class. State pattern is also well known pattern for solution to a problem how to make behavior depend on state. Create an object which represent multiple states and a define Context object whose behavior changes as its state object changes.

Observer Pattern

We can not talk about Object Oriented Programming without considering the state of the objects. Since object oriented programming is about objects and their interaction.
To have a good design means to decouple (differentiate) as much as possible and to reduce the dependencies. The Observer Design Pattern can be used whenever a subject has to be observed by one or more observers. Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category. Observer pattern uses three actor classes. Subject, Observer and Client. Subject is an object having methods to attach and detach observers to a client object. We have created an abstract class *Observer* and a concrete class *Subject* that is extending class *Observer*.

Database Access Object

Through our SQLConnector class we implemented a Database Access Object pattern which we used to separate high-level business logic from the low-level data accessing API. By mapping application calls to the persistence layer (the database), the DAO provides access to data operations without exposing details of the database. This architectural pattern also improved the security of the system as it placed another layer of protection around the database which could store sensitive information about the user and business.

Sample Class Diagram for DAO design pattern

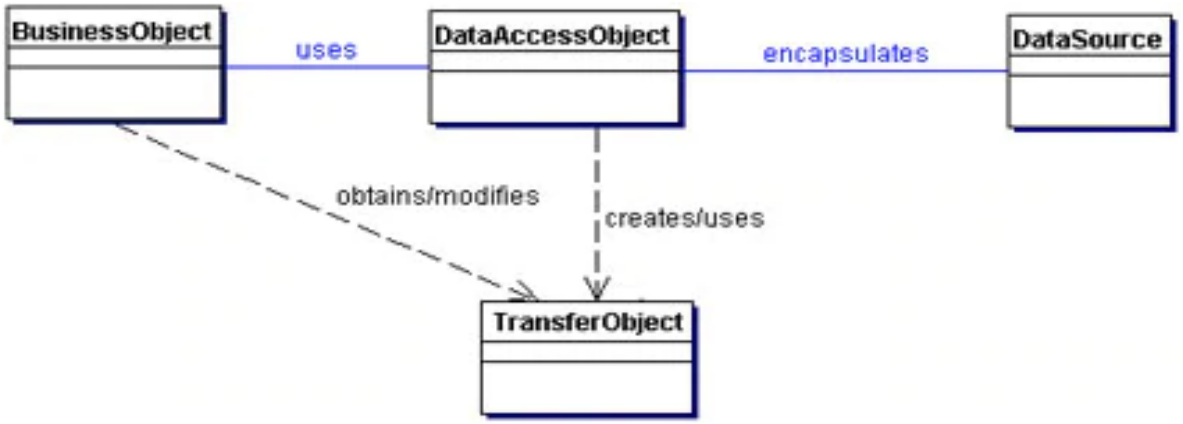
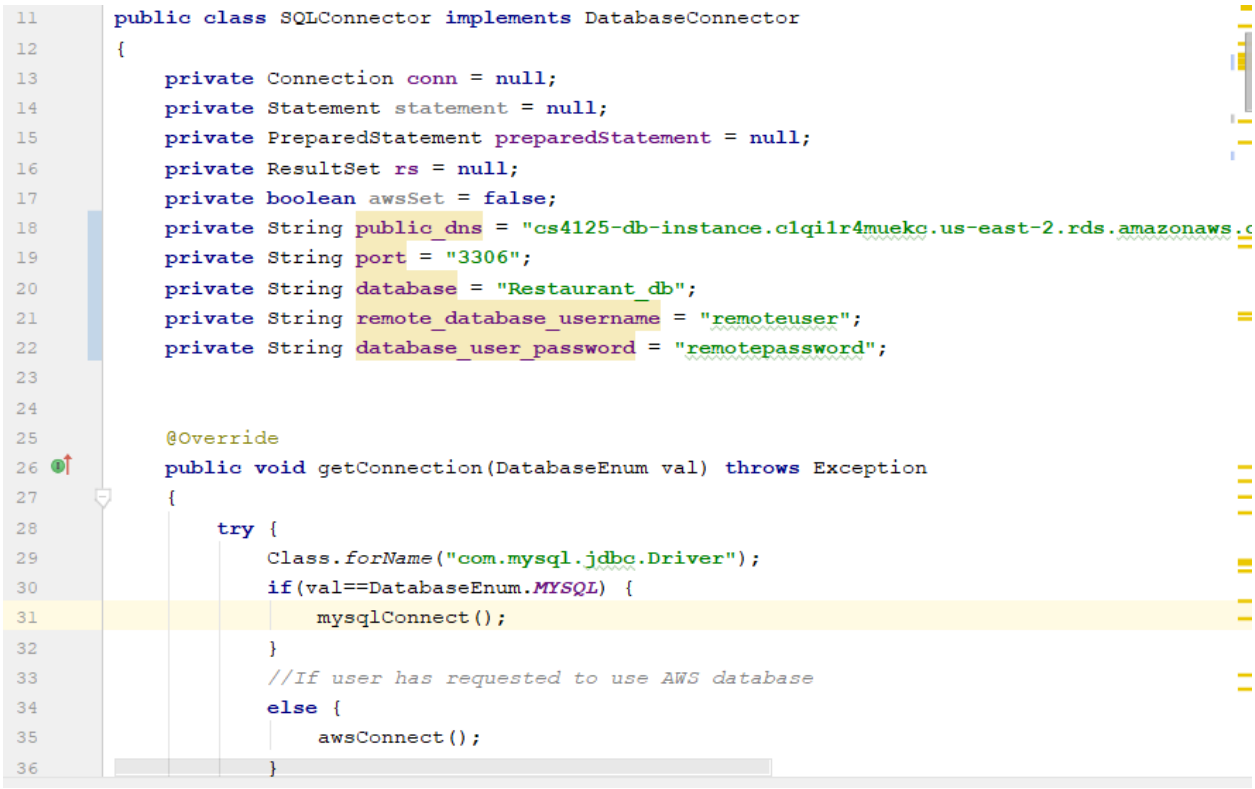


Figure 29.
Source: https://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Code snippet from SQLConnector.java



```
11  public class SQLConnector implements DatabaseConnector
12  {
13      private Connection conn = null;
14      private Statement statement = null;
15      private PreparedStatement preparedStatement = null;
16      private ResultSet rs = null;
17      private boolean awsSet = false;
18      private String public_dns = "cs4125-db-instance.c1qi1r4muekc.us-east-2.rds.amazonaws.c
19      private String port = "3306";
20      private String database = "Restaurant_db";
21      private String remote_database_username = "remoteuser";
22      private String database_user_password = "remotepassword";
23
24
25      @Override
26      public void getConnection(DatabaseEnum val) throws Exception
27      {
28          try {
29              Class.forName("com.mysql.jdbc.Driver");
30              if(val==DatabaseEnum.MYSQL) {
31                  mysqlConnect();
32              }
33              //If user has requested to use AWS database
34              else {
35                  awsConnect();
36              }
```

Figure 30.

MVC Pattern

We made use of the MVC architectural pattern to separate the application into 3 separate types of component. The model components are responsible for accessing the database and the main functionality, the views are responsible for the UI and the Components are responsible for managing updates to the UI. In order to further decouple the views from other components we used JavaFX for UI. Replacing the initial Swing UI with JavaFX helped us to enforce the principle that core functionality should be independent of the interface to enable multiple interface styles to co-exist. Using the MVC pattern facilitated a greater degree of portability and eased maintenance by making the code more modular. We also implemented a closed system architecture that prevented view components from directly accessing model components in order to further decouple the layers.

Added Value

MySQL Database

We stored our data in a MySQL database which we accessed through Java's JDBC Driver. MySQL is a free, open-source database whose scalability and security have seen it used by small startups to giant companies such as Facebook and Twitter. Although the database took longer to set up than a CSV file would have we felt the advantages it offered in terms of security, scalability and performance (through features such as indexing) justified this decision, especially when considering the support MySQL would provide in the future if the project were to be expanded.

AWS

In addition to a local database we also decided to give the user the option of using Amazon Web Services (AWS) to store their data. We used Amazon's RDS (Relational Database Service) running MySQL to implement this. AWS cloud computing provides a number of advantages to business users. These include being able to avoid having to make large up-front investments in data centers and servers and instead make use of a more flexible payment model where users pay only as they consume.

Lambdas

In Java we can use Lambdas to simplify classes that implement interfaces that have just one method i.e. interfaces which qualify as functional interfaces. A popular example of a functional interface is ActionListener, which has just 1 method: ActionPerformed. The key advantages of lambda are enabling more concise code and greater readability.

Use of Lambda in LoginView class

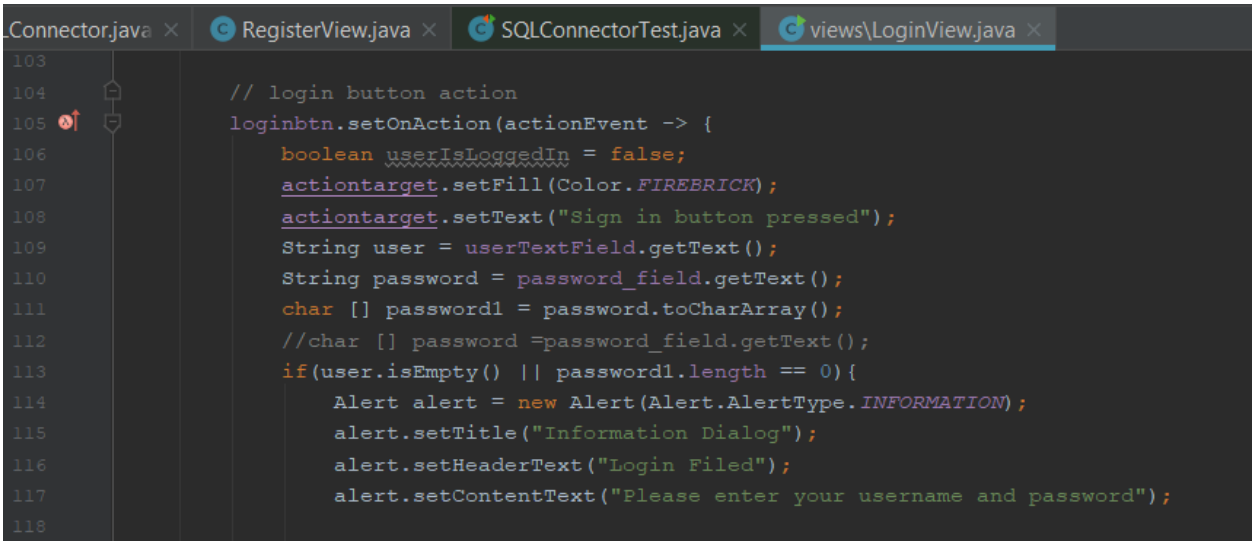


Figure 31.

JUnit Testing

We used JUnit to unit test the code during development. Developing a unit test framework helped to identify bugs and possible edge cases as well as to make the development process more agile by making refactoring easier (as having a suite of unit tests in place allowed us to quickly and easily see any problems caused by refactoring).

JUnit code from SQLConnectorTest.java

```
102     //Tests ability to connect to AWS
103     @Test
104     public void testAWSConnection(){
105         try {
106             sqlconnect.getConnection(DatabaseEnum.AWS);
107         }
108         catch(Exception e){
109
110         }
111     }
112     //Test update operation
113     @Test
114     public void testUpdate(){
115         try {
116             sqlconnect.getConnection(DatabaseEnum.MYSQL);
117         }
118         catch(Exception e){
119
120         }
121         String [] columns = {"username", "password"};
122         String [] values = {"Mikeyo", "pass123"};
123         boolean updated = sqlconnect.update( table: "users", columns, values,  whereClause: "WHE
124         System.out.println(updated);
125         assertEquals(updated,  actual: true);
126         assertTrue(updated);
127     }
```

Figure 32.

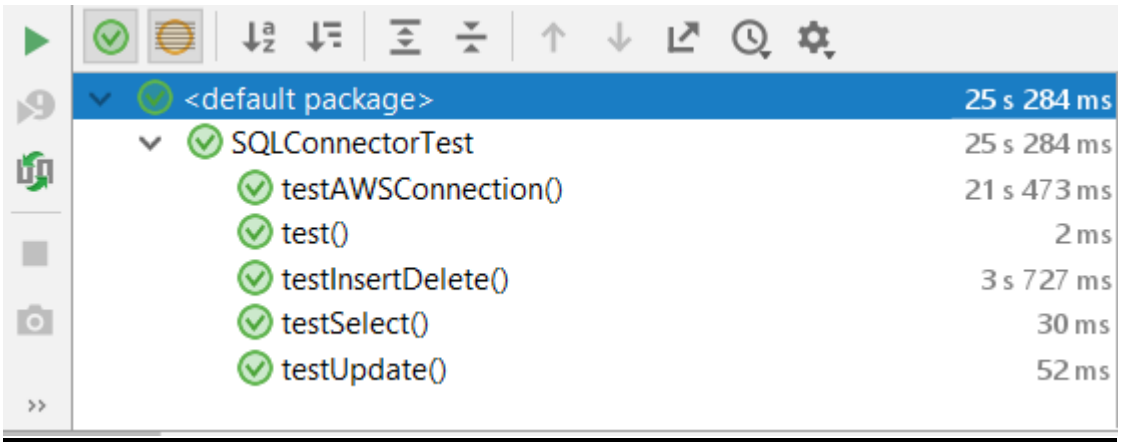


Figure 33.

Functional Interfaces

In this project we also made use of functional interfaces. A functional interface is an interface with only one method. In Java, this can be enforced at compile time using the @FunctionalInterface annotation, which also makes it clear to other programmers that

this should be a functional interface. Functional interfaces work especially well with Method References.

```
1  package controllers;
2
3  @FunctionalInterface
4  public interface ControlledScreen
5  {
6      //This method will allow the injection of the Parent ScreenPane
7      public void setScreenParent(ScreensController screenPage);
8  }
```

Figure 34.

Method Reference

Coupled with the functional interfaces, we were also able to make use of the Method Reference functionality in Java, to simplify code and make it more readable. They are often used when a lambda does nothing except call another method.

```
@Override
public void notifyAllObservers()
{
    observers.forEach(Observer::update);
}
```

Figure 35.

Adobe Photoshop CC 2018

we have used photoshop to design the software development life cycle diagram(SDLC). The diagrams are designed for,

1. Software Development Life-Cycle (SDLC) process
2. Waterfall Model
3. V-Model

GitHub

We used Github for version control in our project. Github is a web-based hosting service for version control using Git. It offers all of the Distributed Version Control (DVC) and Source Code Management (SCM) of Git with added features. This allowed us to track and compare changes as well as recover states if necessary. We also attempted to use the principle of Continuous Integration (as recommended by Grady Booch). While we were not able to push and merge all work several times a day (due mostly to having a large amount of other projects to work on) we attempted to push as often as possible in order to reduce integration problems.

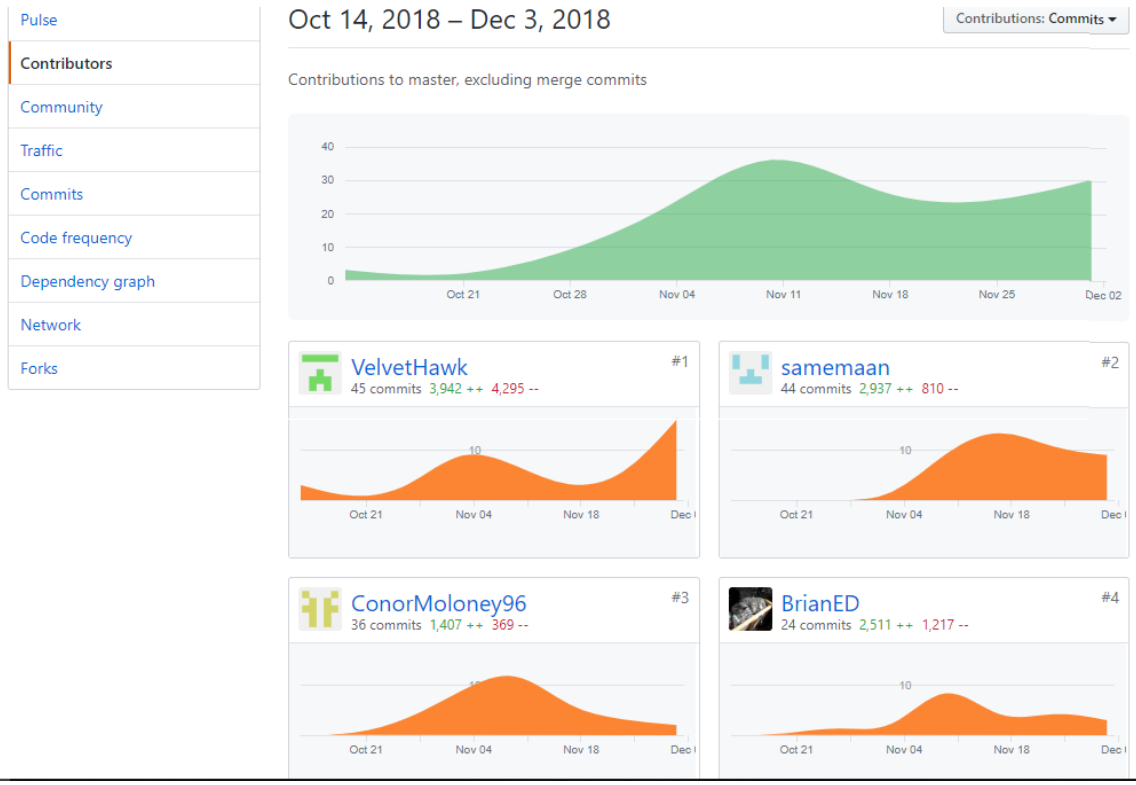


Figure 36.

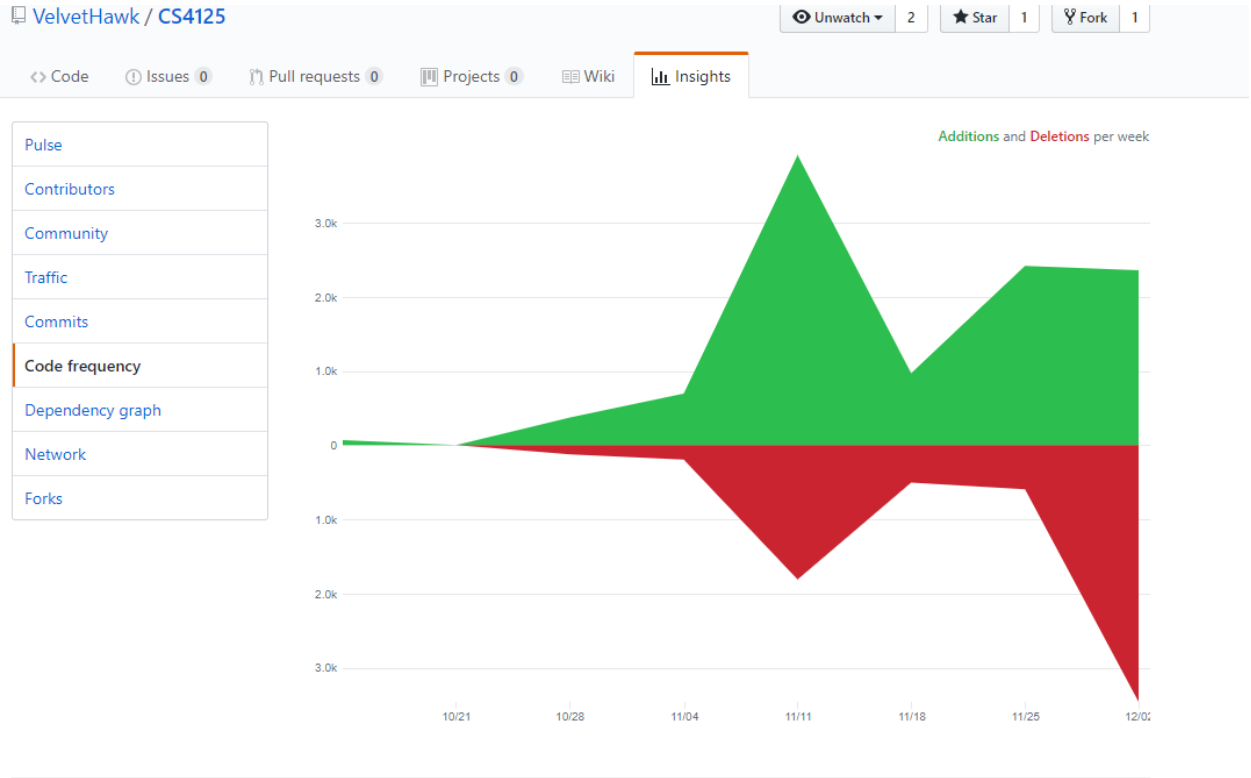


Figure 37.

SonarQube

We used SonarQube to give us statistics and analysis of the code base. It detects bugs, code smells and security vulnerabilities in multiple languages. It also reports duplicate code, coding standards, unit tests and code coverage. It is a useful tool used for continuous integration and continuous inspection of code quality. Two versions of this analysis tool could be used for the project. SonarLite which can be integrated into Intellij IDEA and SonarQube itself, which gathers information and displays in on a web GUI. The program was ran on the command line using SonarQube and SonarScanner applications to gather metrics. The inputs for this were the source files of the project as well as the binaries.

The images below show the SonarQube dashboard which shows the relevant information on the project and gives in depth information on the bugs, vulnerabilities, code smells and duplications

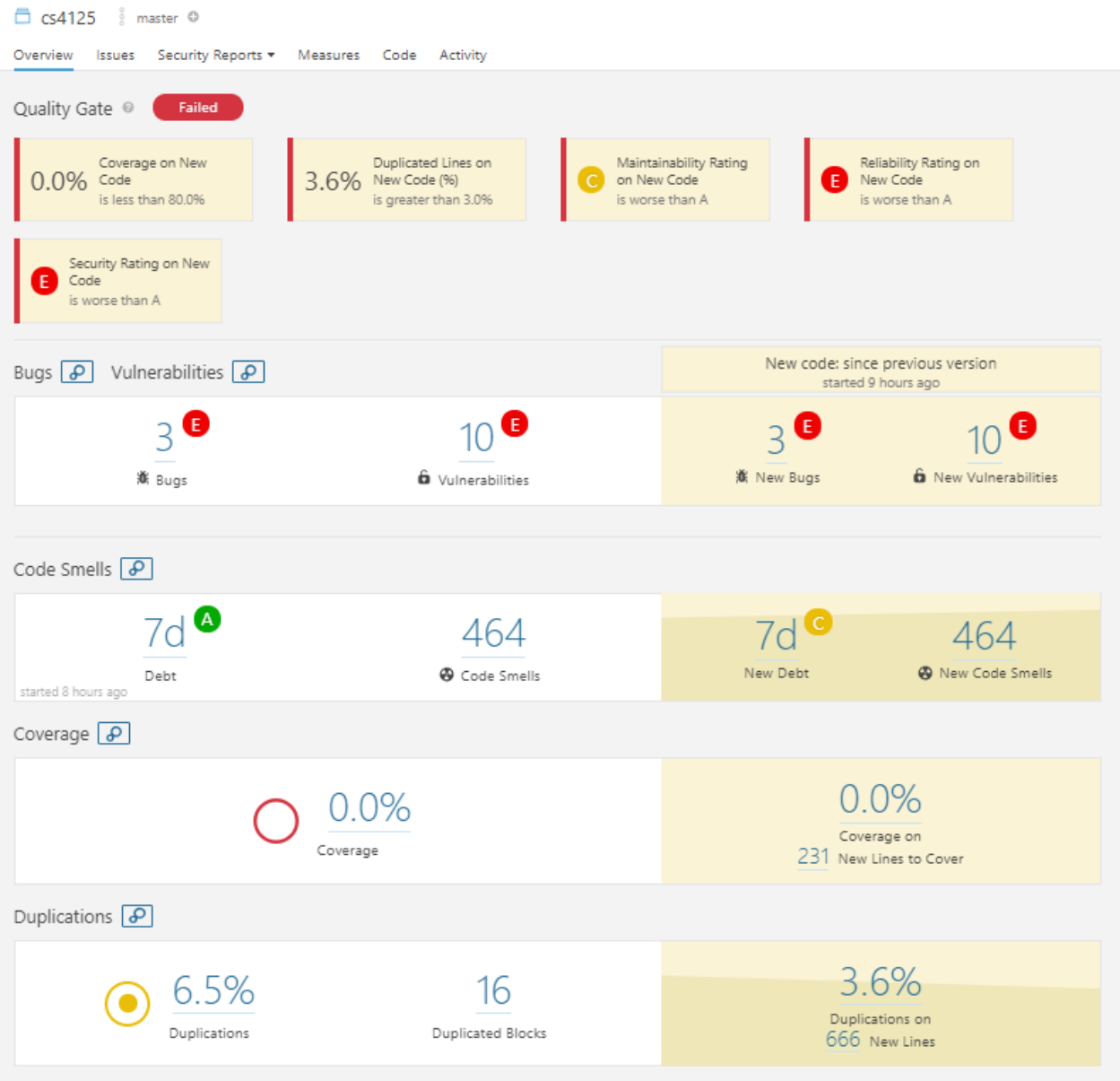


Figure 38.

Figure 38 showing the SonarQube dashboard prior to refactoring

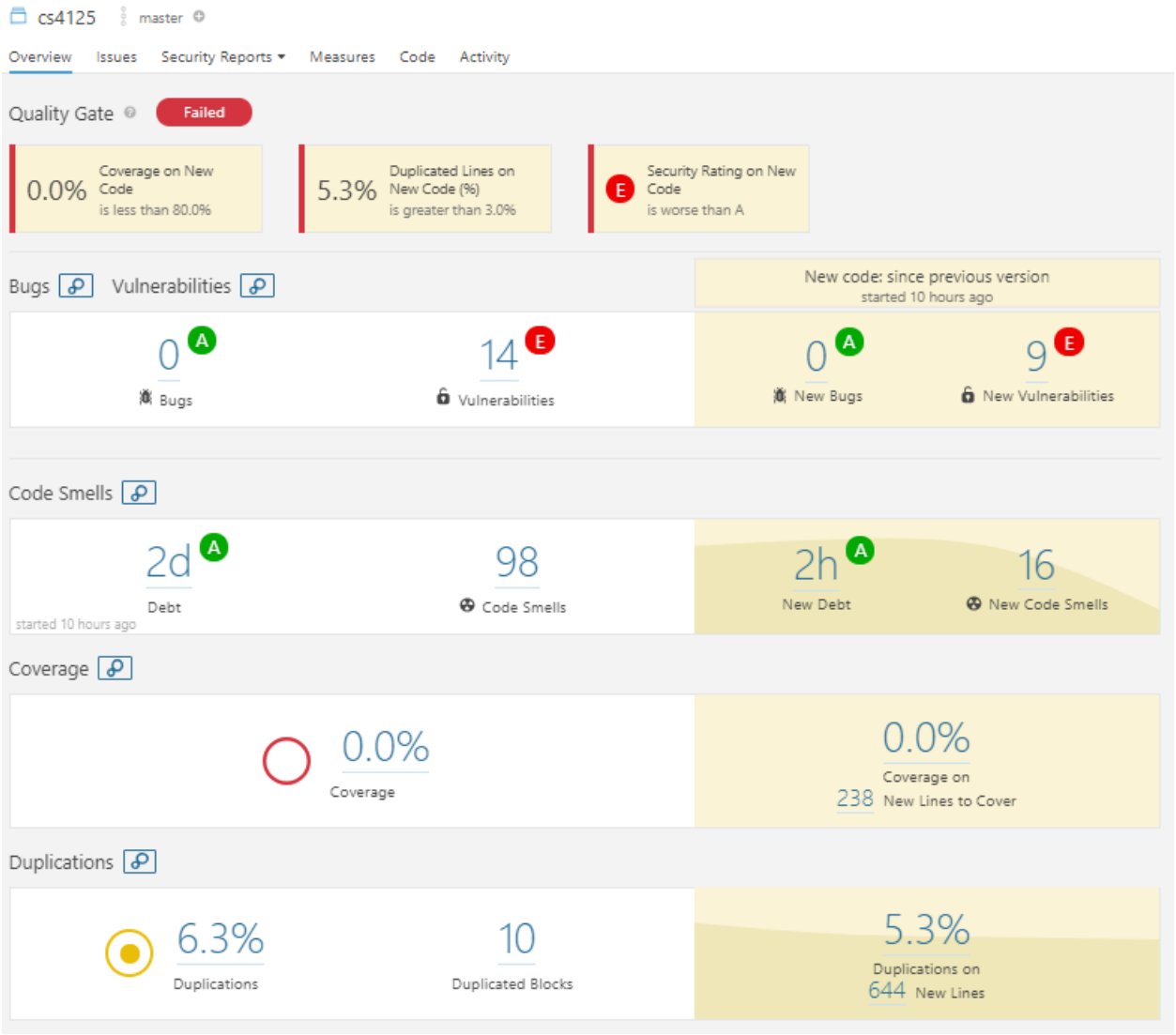


Figure 39.

Figure 39 showing the SonarQube dashboard after some refactoring (Code smells greatly reduced)

Figure 40.

Figure 40 showing the bugs SonarQube has detected

Figure 41.

Figure 41 showing code smells

Trello:

We used Trello to help us organize our tasks objectives for the project and clearly visualize which team member was responsible for which assignment.

A Trello board has four key components, but comes with unlimited possibility.
Cards - The fundamental unit of a board is a card. Cards are used to represent tasks and ideas. A card can be something that needs to get done.
Cards can be customized to hold a wide variety of useful information by clicking on them. Drag and drop cards across lists to show progress. There's no limit to the number of cards you can add to a board.

For our project, we have used the following 4 statuses for a task:

1. To-Do
2. In Progress
3. In Review
4. Completed

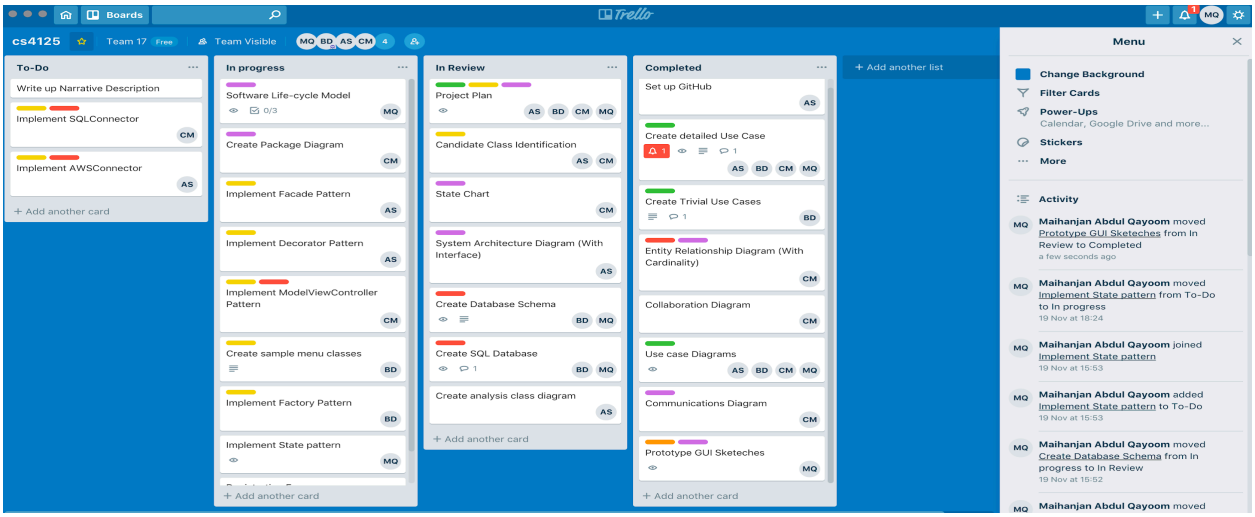


Figure 42.

Recovered Blueprints

Order Status State Chart Based on this project

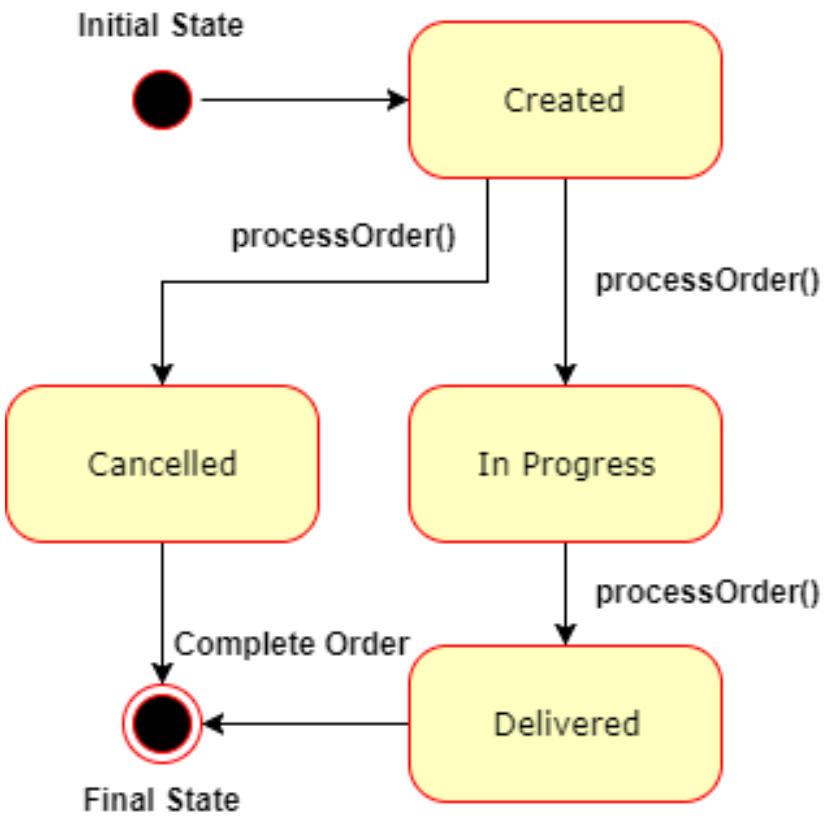

Initial State
Created
processOrder()
processOrder()
Cancelled
In Progress
processOrder()
Complete Order
Delivered
Final State

Figure 43.

Sequence Diagram

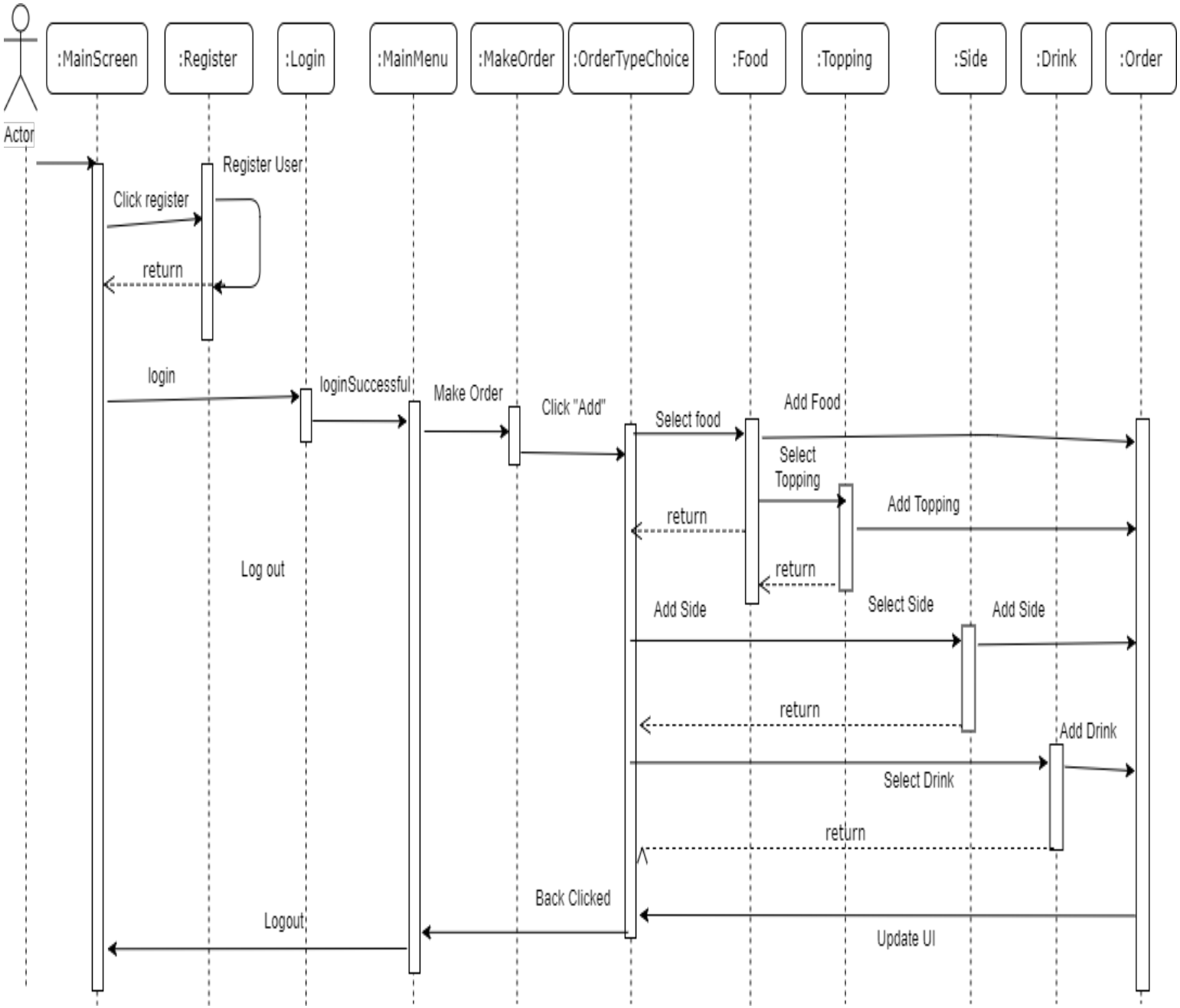

Actor
:MainScreen
:Register
:Login
:MainMenu
:MakeOrder
:OrderTypeChoice
:Food
:Topping
:Side
:Drink
:Order
Register User
Click register
return
login
loginSuccessful
Make Order
Click "Add"
Select food
Add Food
Select Topping
Add Topping
return
return
Log out
Add Side
Select Side
Add Side
return
Add Drink
Select Drink
return
Back Clicked
Logout
Update UI

Figure 44.

Critique and Analysis of Design Artifacts

Overall we are happy with how we have implemented our project. While we didn't have time to implement all the use cases we outlined in the planning stage we were able to implement a number of key use cases and we were also able to design the system in such a way that adding additional functionality in the future should be simplified.

While our final implementation differed in several ways from the uml diagrams we developed during the Analysis phase the development of those diagrams forced us to think about the structure of the project in the early planning stages and this greatly influenced our thinking during the implementation.

During the lifecycle of the project continuous analysis of the source code using SonarQube could have improved code quality and kept the project bug, vulnerability and code smell free throughout its implementation. Running code analysis software towards the end of the project life cycle showed up many code smells which would need time to sift through and fix.

In the future additional functionality such as the ability for employees to order stock as well as security improvements such as hashing and salting the passwords stored in the database could be implemented. Fortunately the design of the system (especially it's modularity due to the implementation of the MVC and DAO architectural patterns) facilitates efficient extensibility.

References

Software Development Life Cycle (SDLC):
https://www.tutorialspoint.com/sdlc/index.htm
https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/

Some of the GUI images are taken from the following link, free download.
https://www.pexels.com/search/food/

JavaFx scene builder
https://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html

Database Access Object Architectural Pattern (diagram also taken from link):
https://www.oracle.com/technetwork/java/dataaccessobject-138824.html

How to Develop JUnit tests in IntelliJ: https://www.jetbrains.com/help/idea/create-tests.html

Setting up a MySQL database:
https://dev.mysql.com/doc/mysql-getting-started/en/

Connecting to AWS via JDBC:
https://medium.com/modernnerd-code/connecting-to-mysql-db-on-aws-ec2-with-jdbc-for-java-91dba3003abb

Lambdas in Java:
https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

Appendix

Final Class Analysis Diagram