

Software Engineering Specification

Eat N' Beat

Revision 3.5

Table of Contents

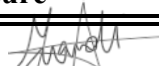
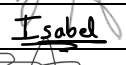

1. Introduction.....	4
1.1 Purpose of the System	4
1.2 Product Scope	4
1.3 Assumptions and Dependencies.....	4
1.4 Conventions.....	4
2. Software Requirements Specifications	5
2.1 Administrator Application (High Privilege Interface)	5
2.1.1 Description and Priority	5
2.1.2 Stimulus / Response Sequences	5
2.1.3 Functional Requirements.....	5
2.2 Smart Cash Register (Medium Privilege Interface)	6
2.2.1 Description and Priority	6
2.2.2 Stimulus / Response Sequences.....	6
2.2.3 Functional Requirements.....	6
2.3 Customer Interfaces (Low Privilege Interface).....	6
2.3.1 Description and Priority	6
2.3.2 Stimulus / Response Sequences.....	6
2.3.3 Functional Requirements.....	6
2.4 Other Nonfunctional Requirements	1
2.4.1 Performance Requirements	1
2.4.2 Security Requirements	1
2.4.3 Software Quality Attributes.....	1
2.5 Requirements Summary	1
3. Software Design Specification.....	2
3.1 Structure, dependencies and design decisions	2
3.2 Database Structure	3
3.3 Administrator Application Specification.....	4
3.4 Smart Cash Register Specification.....	3
3.5 Customer Interfaces	6
4. Software Implementation	8
4.1 Tools and Technology Used	8
4.1.1 AAP	8
4.1.2 SCR.....	8
4.1.3 CI	8
4.1.4 Links to GitHub	8
4.2 Screenshots of the Application	9
4.2.1 AAP	9
4.2.2 SCR.....	12
4.2.3 CI	15
4.3 Changes during Implementation	19
4.3.1 Database	19
4.3.2 AAP	20
4.3.3 SCR.....	22
4.3.4 CI	22
4.4 APIs and Libraries	23
4.4.1 AAP	23
4.4.2 SCR.....	23
4.4.3 CI	23
5. Software Testing.....	24
5.1 Automated Testing.....	24
5.2 Usability Testing.....	24
5.2.1 AAP	25
5.2.2 SCR.....	26
5.2.3 IC	27

Revision History

Version	Name	Reason For Changes	Date
1.0	Juan	Software Requirements Specification	04/10/22
1.1	Juan	Software Requirements Specification	14/10/22
2.0	Juan	Software Design Specification	25/11/22
2.1	Isabel	Software Design Specification	25/11/22
2.2	Burak	Software Design Specification	30/11/22
2.3	Juan	Software Design Specification	02/12/22
3.0	Juan	Software Implementation & Software Testing	21/03/23
3.1	Juan	Software Implementation & Software Testing	22/03/23
3.2	Juan	Software Implementation & Software Testing	23/03/23
3.3	Isabel	Software Implementation & Software Testing	23/03/23
3.4	Burak	Software Implementation & Software Testing	23/03/23
3.5	Juan	Putting together all our contributions.	24/03/23

Approved By

Approvals should be obtained for project manager, and all developers working on the project.

Name	Signature	Department	Date
Juan		High Privilege Interface	24/03/2023
Isabel		Medium Privilege Interface	24/03/2023
Burak		Low Privilege Interface	24/03/2023

1. Introduction

1.1 Purpose of the System

The purpose of this system is providing an environment through which a restaurant can become more efficient, better organized and more profitable. The system allows restaurants to do so by providing tools to gain insight, erase the need of repetitive labor, and provide exposure to the platform's users.

1.2 Product Scope

The software consists of an environment through which the administration, employees and customers of a restaurant can dynamically interact. For each of these actors there will be their own features and privileges. The goal of the software is to improve everyone's experience by granting a more personalized and high-quality experience. This high-quality experience will be achieved by making data about the restaurant's countability, inventory and human resources more accessible to the administration of the restaurant, making its flaws easier to spot. For this reason, this product is designed for franchises and chain-restaurants, in which the responsible figures of the restaurant may not have the possibility of spending the needed time to spot certain weak points.

1.3 Assumptions and Dependencies

For this system to provide accurate information, it is assumed that the restaurant implementing it must have very clear knowledge of its costs and therefore profits per product. Another assumption is that the people that will interact with the system have a minimum prior knowledge about interactive systems and the internet. This system depends highly in the quality of the data provided by the administrators when setting up the system and how constant the working habits remain over time (this is, using roughly the same amount of a certain ingredient to make the same plate, food waste, ...). If any of these two is not good enough, it may result in the system providing data that leads to incorrect conclusions. Also, this system is dependent to a reliable internet connection.

1.4 Conventions

A few acronyms are used throughout the specification report, although they are not many, they are important for the correct understanding of the report.

AAP	Administrator Application
SCR	Smart Cash Register
CI	Customer Interfaces (Web and App)
LP	Loyalty Points
DB	Database

2. Software Requirements Specifications

The system features and their functional requirements and non-functional requirements are all ordered based on their priority in descendent order. Priority is defined as 1/High, 2/Medium or 3/Low.

The requirements are highlighted based on whether they were partially, fully or not at all accomplished. The follow the following rubric:

- Fully accomplished
- Partially accomplished
- Not accomplished
- Unviable to implement

2.1 Administrator Application (High Privilege Interface)

2.1.1 Description and Priority

It is the interface through which the administrator of a restaurant can set up and gather the information from a system. Here all the information about a restaurant's performance will be accessible in the easiest manner as well as concise reports about the restaurant's performance. This feature is of highest-priority as it is the feature which makes the system desirable to use by its owners, and mandatory for the rest of the features to function.

2.1.2 Stimulus / Response Sequences

- Set-up/modify menu.
- Set-up/modify prices.
- Set-up/modify ingredients.
- Set-up/modify providers.
- Set-up/modify employees/shifts.
- Get sales report.
- Get expenses report.
- Notify providers.
- Stablish discounts based on LP.
- Check for late entries.

2.1.3 Functional Requirements

- REQ-2.1.3.1: The interface shall allow to set-up/modify menus. [1]
- REQ-2.1.3.2: The interface shall allow to set-up/modify products. [1]
- REQ-2.1.3.3: The interface shall allow to set-up/modify ingredients. [1]
- REQ-2.1.3.4: The interface shall allow to set-up/modify providers. [1]
- REQ-2.1.3.5: The interface shall allow to set-up/modify employees/shifts. [1]
- REQ-2.1.3.6: The interface shall provide sales reports. [1]
- REQ-2.1.3.7: The interface shall provide expenses reports. [1]
- REQ-2.1.3.8: The interface shall allow to modify menus' descriptions / images. [2]
- REQ-2.1.3.9: The interface shall allow to modify products' descriptions / images. [2]
- REQ-2.1.3.10: The interface shall allow to stablish discounts based on LP. [2]
- REQ-2.1.3.11: The interface shall allow to contact provider based on sales. [2]
- REQ-2.1.3.12: The interface shall provide a tool to check for late entries. [2]

2.2 Smart Cash Register (Medium Privilege Interface)

2.2.1 Description and Priority

It is the interface through which waiters will have access to the online-orders as well as keep track of the restaurant's availability and current orders. Also, this interface will register the check-in and check-out times of each employee as well as allow waiters to register on the system "manually ordered" orders. This is a high-priority feature as this is the feature which will provide the system the information necessary to produce quality data for the administration.

2.2.2 Stimulus / Response Sequences

- *Check-in/-out.*
- *Enter order.*
- *Modify order.*
- *Check table availability.*
- *View orders.*
- *Charge.*
- *Generate receipt.*
- *Modify availability of ingredient.*

2.2.3 Functional Requirements

- REQ-2.2.3.1: The interface shall allow to take orders. [1]
- REQ-2.2.3.2: The interface shall provide live information on table availability. [1]
- REQ-2.2.3.3: The interface shall calculate subtotal of an order. [1]
- REQ-2.2.3.4: The interface shall generate the receipt of an order. [1]
- REQ-2.2.3.5: The interface shall provide live information on orders. [1]
- REQ-2.2.3.6: The interface shall allow to modify orders. [2]
- REQ-2.2.3.7: The interface shall allow employees to check-in/out. [2]
- REQ-2.2.3.8: The interface shall allow to modify the availability of an ingredient. [3]

2.3 Customer Interfaces (Low Privilege Interface)

2.3.1 Description and Priority

It is the interface through which the customer could directly order; saving both, the waiter's and his own time. This is a low priority feature, as it is not completely required to fulfill the main goal, but enriches the user experience.

2.3.2 Stimulus / Response Sequences

- *Log-in/-out.*
- *Sign-in/-out.*
- *Order.*
- *Modify order.*
- *Gain LP.*
- *Redeem LP in exchange of discount.*

2.3.3 Functional Requirements

- REQ-2.3.3.1: The website shall provide the menu. [1]
- REQ-2.3.3.2: The phone app shall provide the menu. [1]
- REQ-2.3.3.3: The phone app shall allow customers to order from their phone. [1]
- REQ-2.3.3.4: The phone app shall allow customer to modify their order. [2]
- REQ-2.3.3.5: The phone app shall allow customers to create an account/log in. [2]
- REQ-2.3.3.6: The phone app shall allow registered cust. to earn LP. [2]
- REQ-2.3.3.7: The phone app shall allow registered cust. to redeem LP. [2]

2.4 Other Nonfunctional Requirements

2.4.1 Performance Requirements

REQ-2.4.1.1: The website shall not require user validation unless loyalty points are to be redeemed. Extends on REQ-2.3.3.1 [1]

REQ-2.4.1.2: All the information provided by interacting with the system that may interfere with another actor's decisions should be available in any interface of the system within a 10 seconds timeframe. [1]

REQ-2.4.1.3: The website's email verification should not take longer than 15 seconds for the user to receive the verification code from the moment of the request. Extends on REQ-2.3.3.5 [2]

2.4.2 Security Requirements

REQ-2.4.2.1: The system shall validate a user by requesting to provide a code sent to their email. Extends on REQ-2.3.3.5 [1]

REQ-2.4.2.2: The system shall allow customers to access their personal data and remove it from the database, according to the GDPR. Extends on REQ-2.3.3.5 [1]

2.4.3 Software Quality Attributes

REQ-2.4.3.1: The system shall allow for admins to receive an email when a certain product has run out. Extends on REQ-2.2.3.8 [1]

REQ-2.4.3.2: The phone app shall provide a basket/shopping cart from where the current products of the order can be viewed. Extends on REQ-2.3.3.4 [1]

REQ-2.4.3.3: The CIs shall allow to filter the search based on allergies and popularity. Extends on REQ-2.3.3.1 [2]

REQ-2.4.3.4: The system shall allow for admins to specify the timelines for the creation of reports. Extends on REQ-2.1.3.6 and REQ-2.1.3.7 [2]

REQ-2.4.3.5: The system shall allow for customers to decide whether they want to receive a copy of their receipt by email. Extends on REQ-2.2.3.4 [3]

REQ-2.4.3.6: The system shall allow for admins to receive a copy of the report to their emails. Extends on REQ-2.1.3.6 and REQ-2.1.3.7 [3]

2.5 Requirements Summary

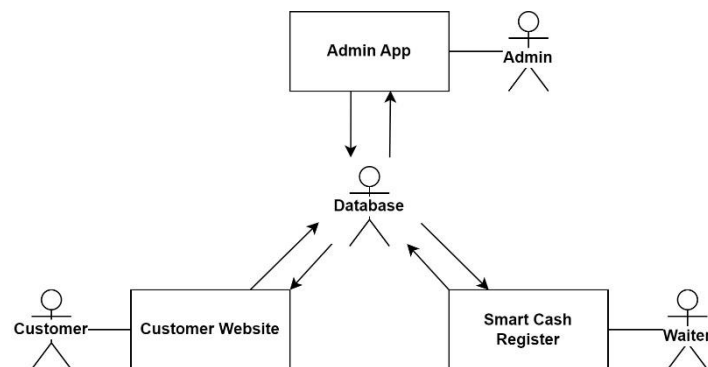
Functional Requirements	Priority
REQ-2.1.3.1 - REQ-2.1.3.7 :	1
REQ-2.1.3.8 - REQ-2.1.3.12 :	2
REQ-2.2.3.1 - REQ-2.2.3.5 :	1
REQ-2.2.3.6 - REQ-2.2.3.7 :	2
REQ-2.2.3.8 :	3
REQ-2.3.3.1 - REQ-2.3.3.3 :	1
REQ-2.3.3.4 - REQ-2.3.3.7 :	2

Nonfunctional Requirements	Priority
REQ-2.4.1.1 - REQ-2.4.1.2 :	1
REQ-2.4.1.3:	2
REQ-2.4.2.*	1
REQ-2.4.3.1 - REQ-2.4.3.2 :	1
REQ-2.4.3.3 - REQ-2.4.3.4 :	2
REQ-2.4.3.5 - REQ-2.4.3.6 :	3

3. Software Design Specification

3.1 Structure, dependencies and design decisions

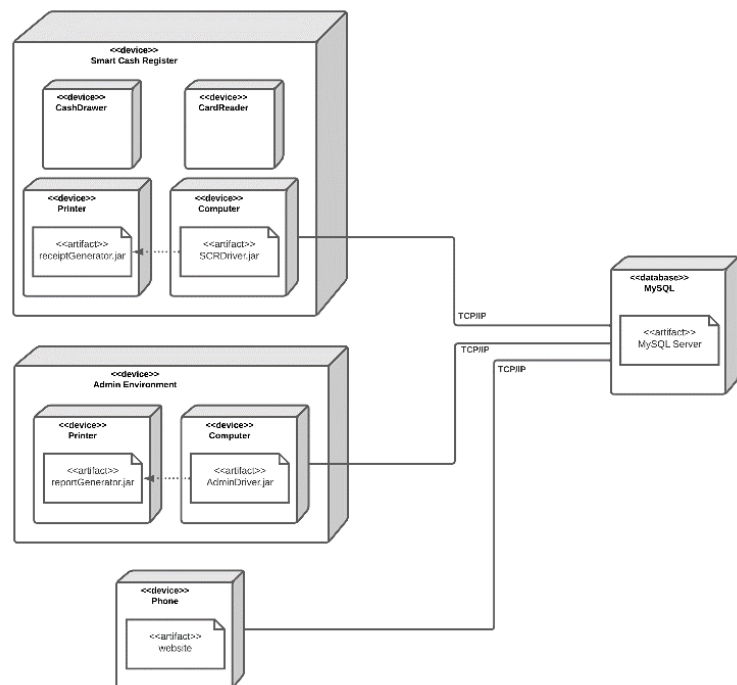
The system provides each user a completely different interface designed to accomplish certain tasks. These interfaces make sense of each other by connecting to a database. This can be easily appreciated in the following UML Use Case diagram, which represents an overview of the system:



*Each component of this use case is later broken down into further use cases

The system has been decided to consist of a phone app (Flutter) and two apps developed in Java. We have decided to develop the desktop apps on Java as it is the programming languages everyone in the team is most familiar with, the system does not require any special library that is not present in java; and, this would allow to re-use code if an android phone app wants to be created, as java is widely used in android phone apps.

Also, as it has been decided to store all the information in the database and not locally, the system is not functional if there is no internet connection, so this one being reliable is a must. This can be seen in the following deployment diagram:

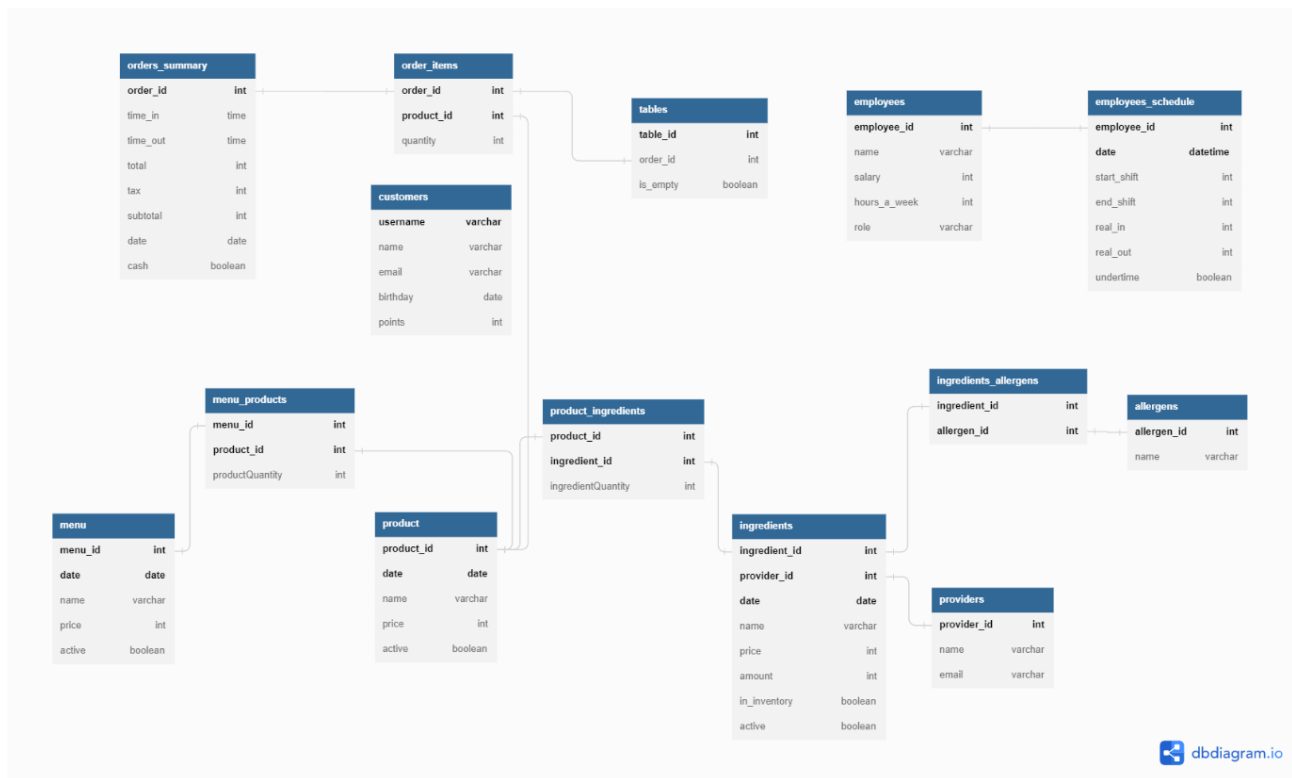


3.2 Database Structure

The database acts as the bridge between the components of the system. The database stores all the information that may be needed/used at any point, so its architecture is a crucial point of the system. This will be later seen in the deployment diagram.

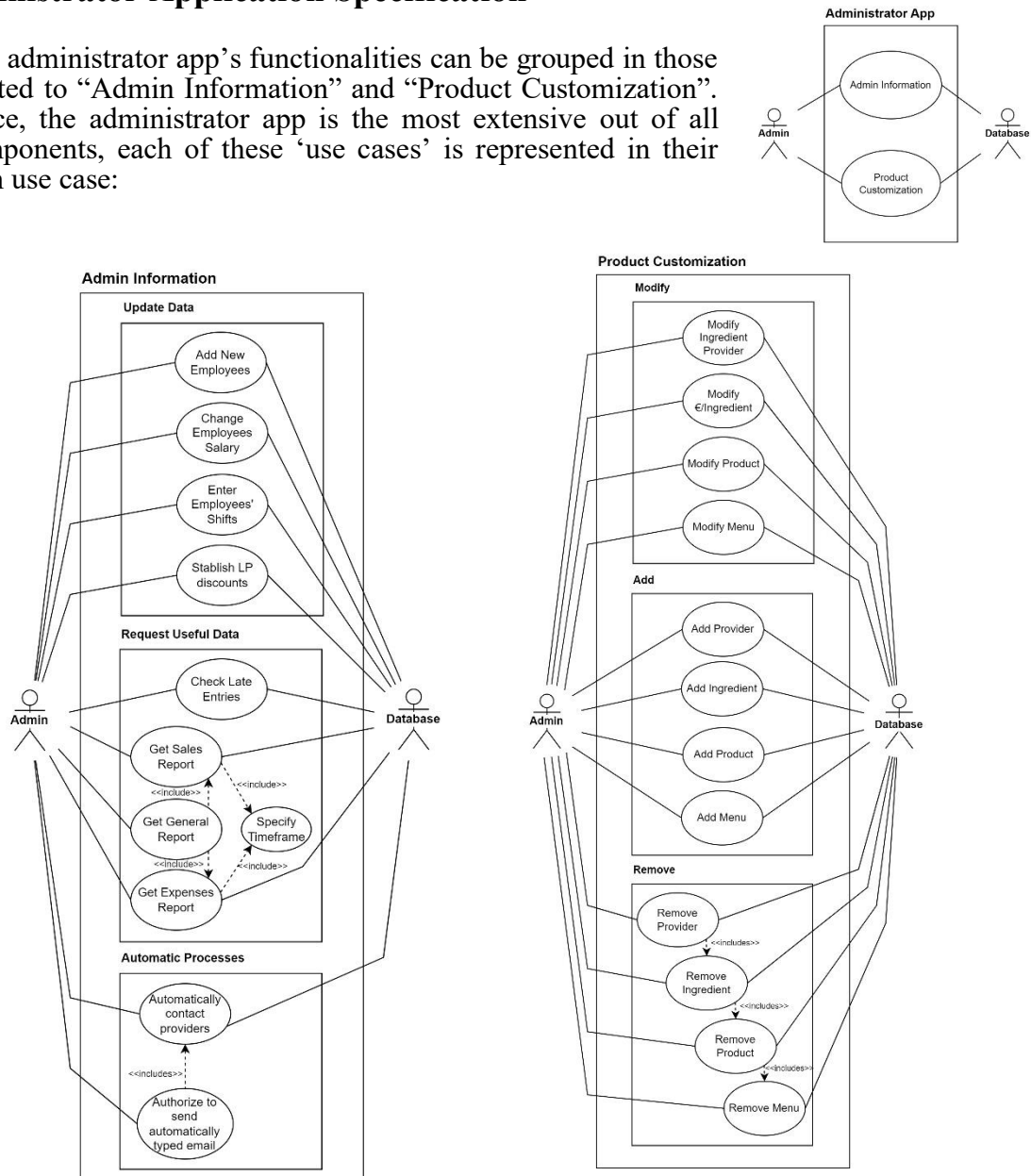
The database follows the design principles and best practices for this type of database. Every table keeps data integrity by having as a primary key either an ID, or the combination of various IDs, and, in the cases where the prices of something may change, also the date when the product price was last established. This, plus the correct referencing to foreign keys should preserve the integrity of the database.

The following entity-relationship diagram is an accurate representation of our database architecture.

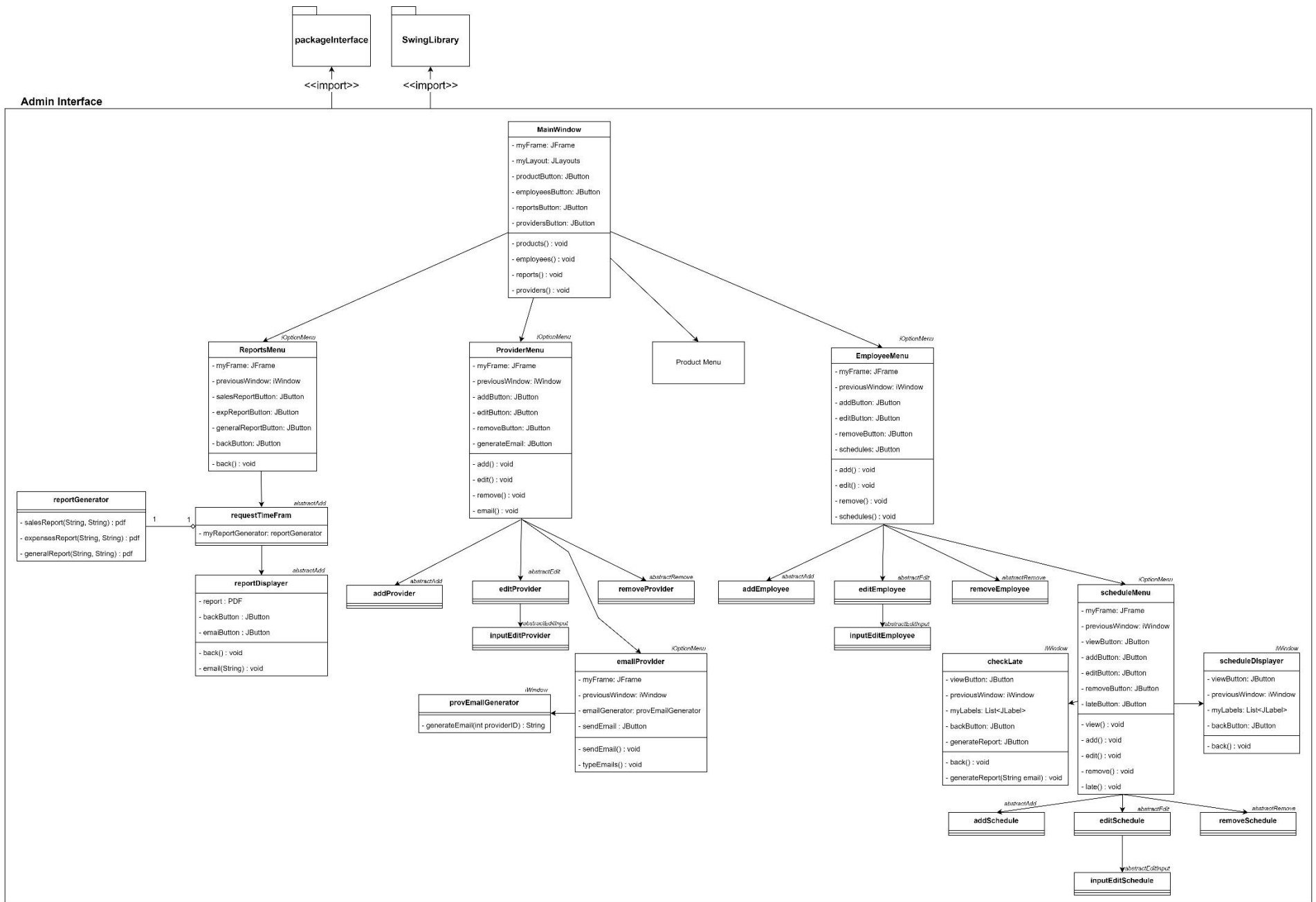


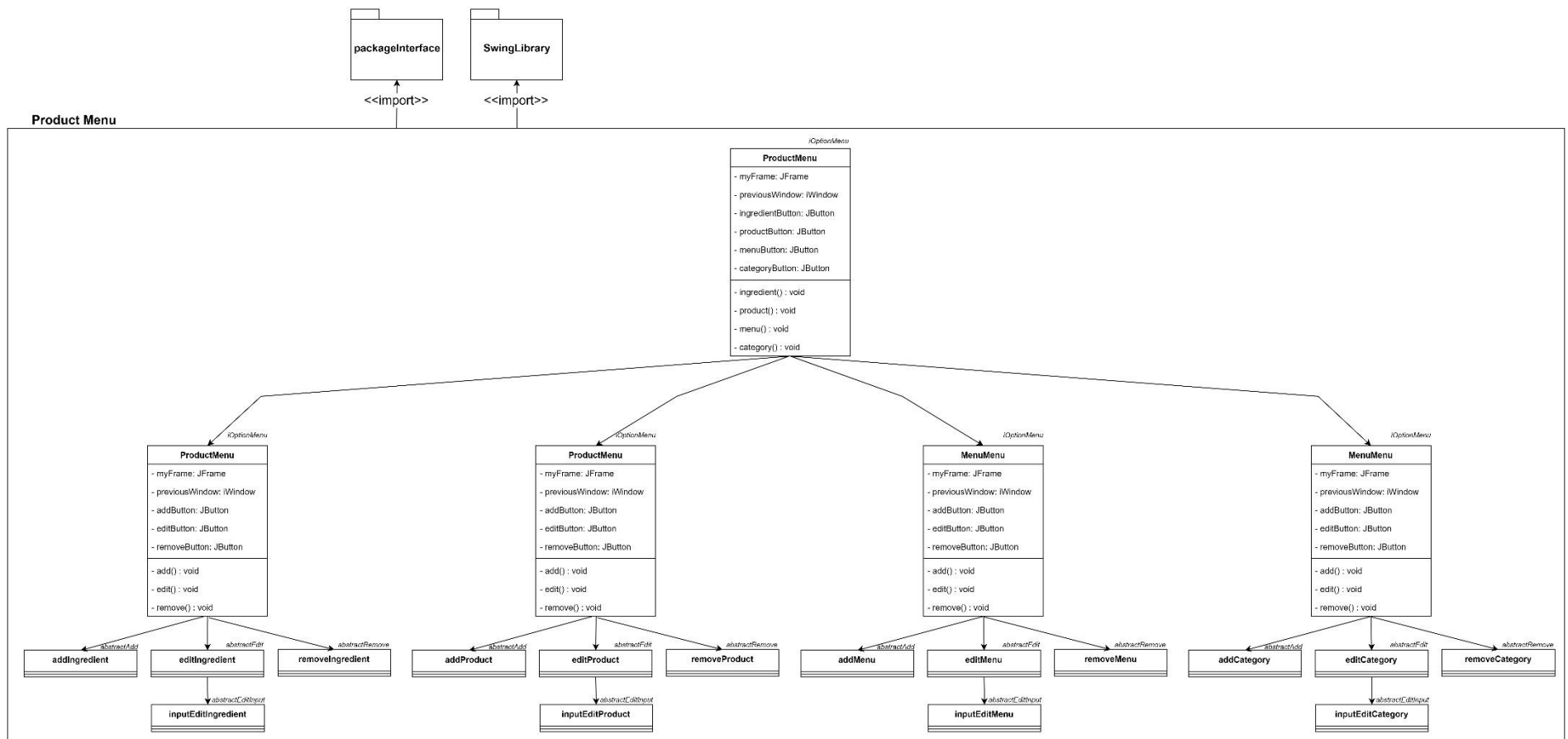
3.3 Administrator Application Specification

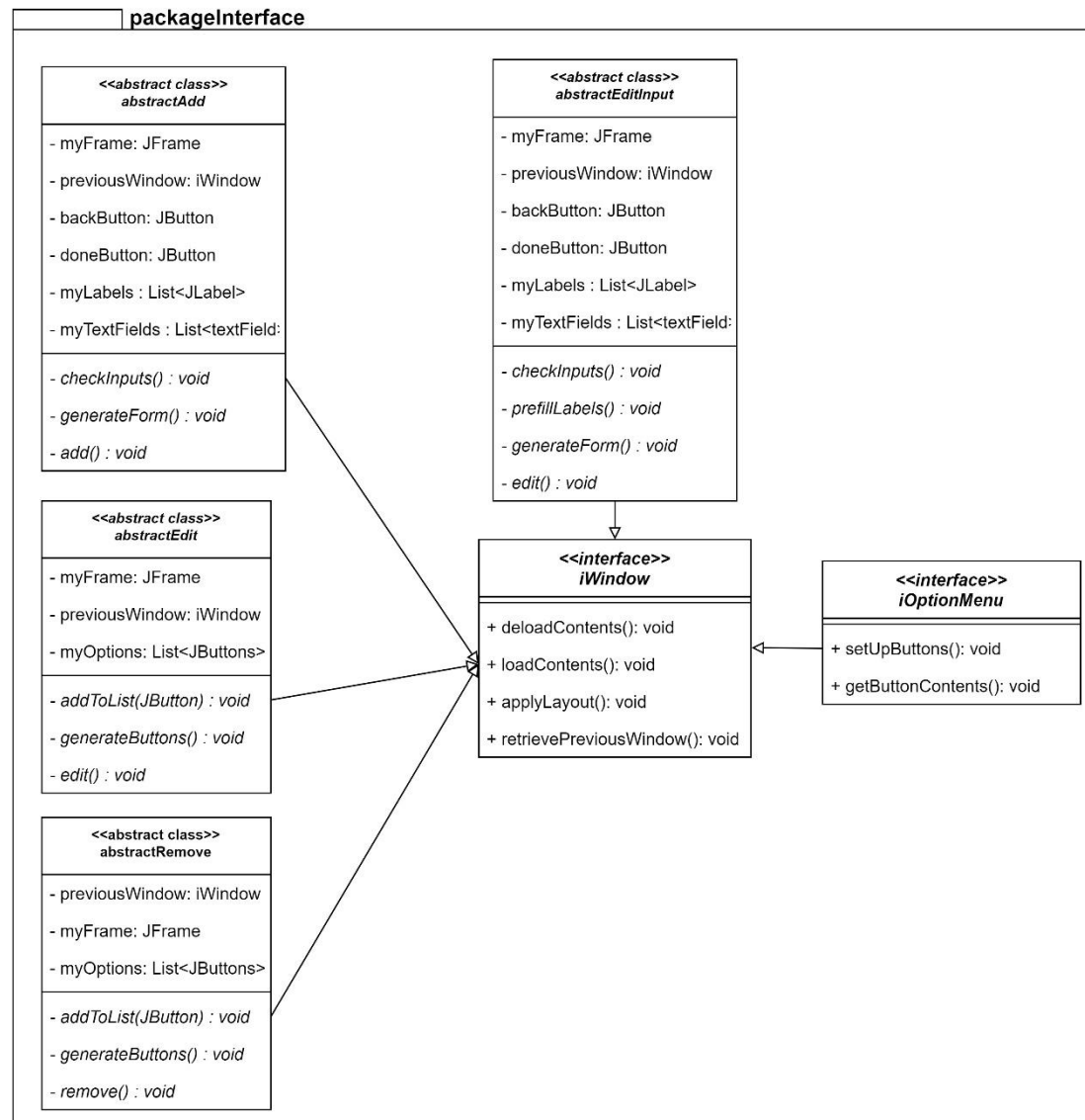
The administrator app's functionalities can be grouped in those related to "Admin Information" and "Product Customization". Since, the administrator app is the most extensive out of all components, each of these 'use cases' is represented in their own use case:



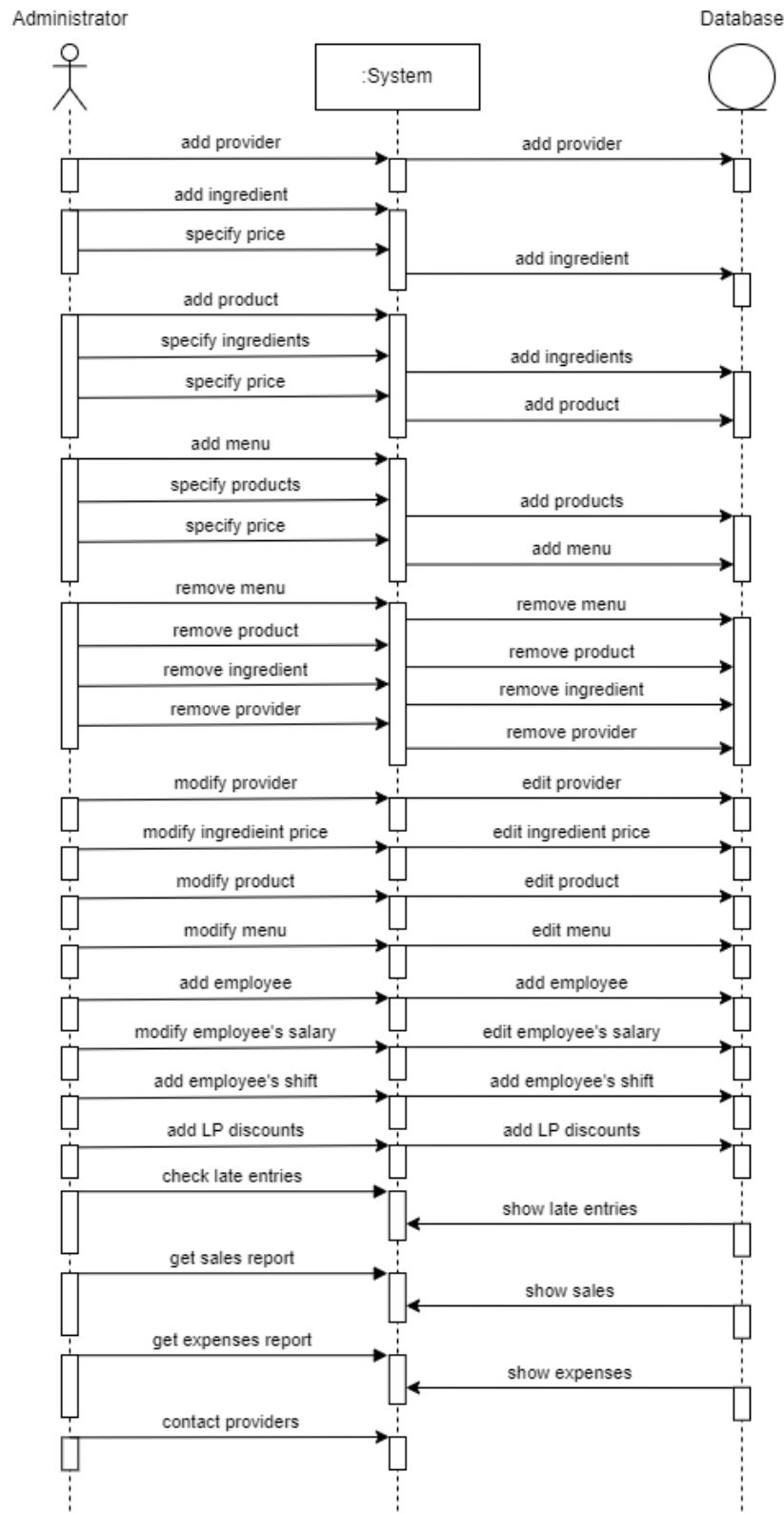
This component provides the administrator of the system all the needed functionalities to easily carry out their job. The administrator app is thought to be made out of a single window, which changes its contents based on the clicked option, allowing for at most, one pop-up window at a time. This is accomplished by implementing an interface which is common to all the classes that may alter the main window, this is "iWindow". This interface is part of a package of interfaces and abstract classes which will be imported to every class which makes up the administrator app. This can be seen in the following class diagrams:



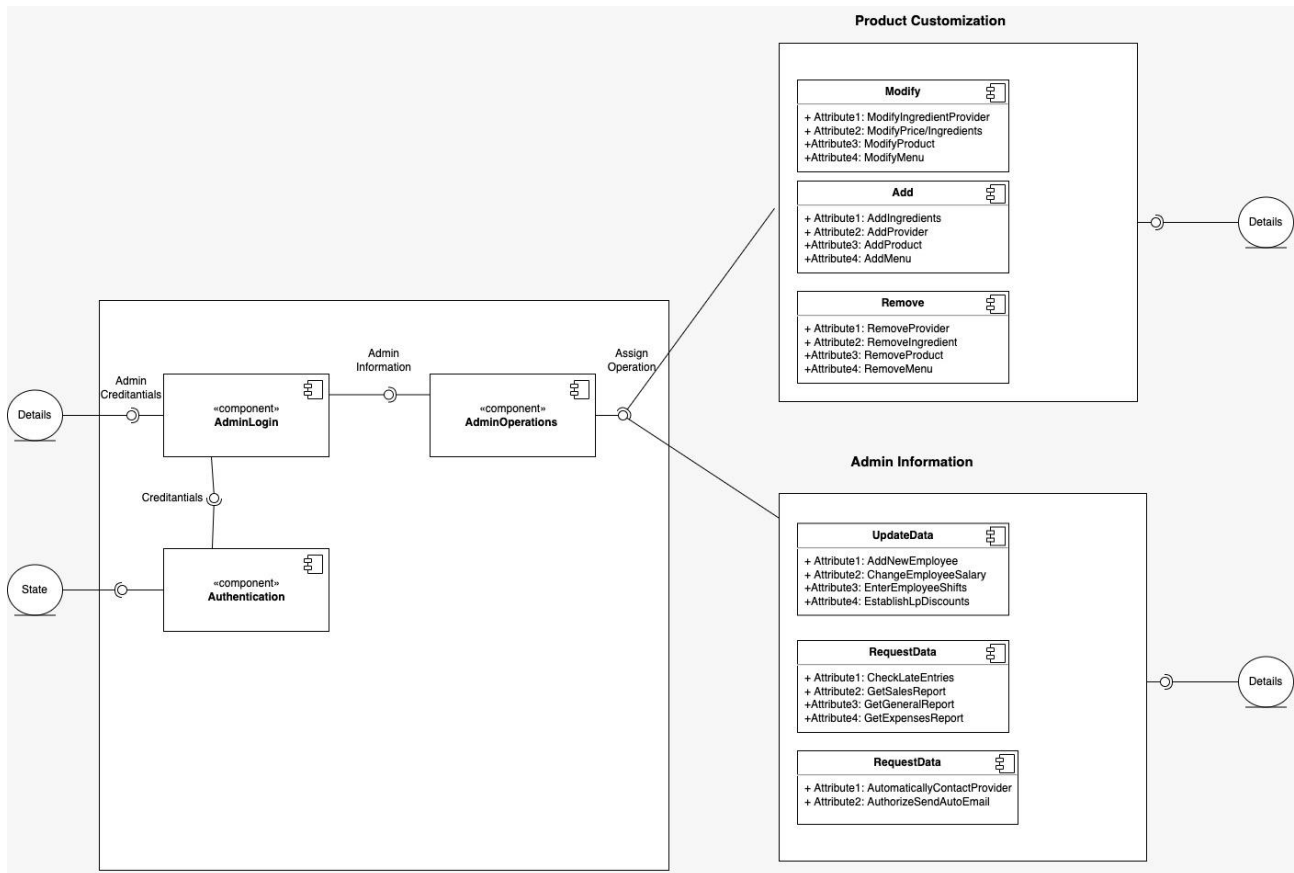




Most of the administrator's tasks are usually one way, as, the administrator mostly provides data to the database. The admin only requires data to be requested from the database when a report is to be generated or a check-in time to be checked.

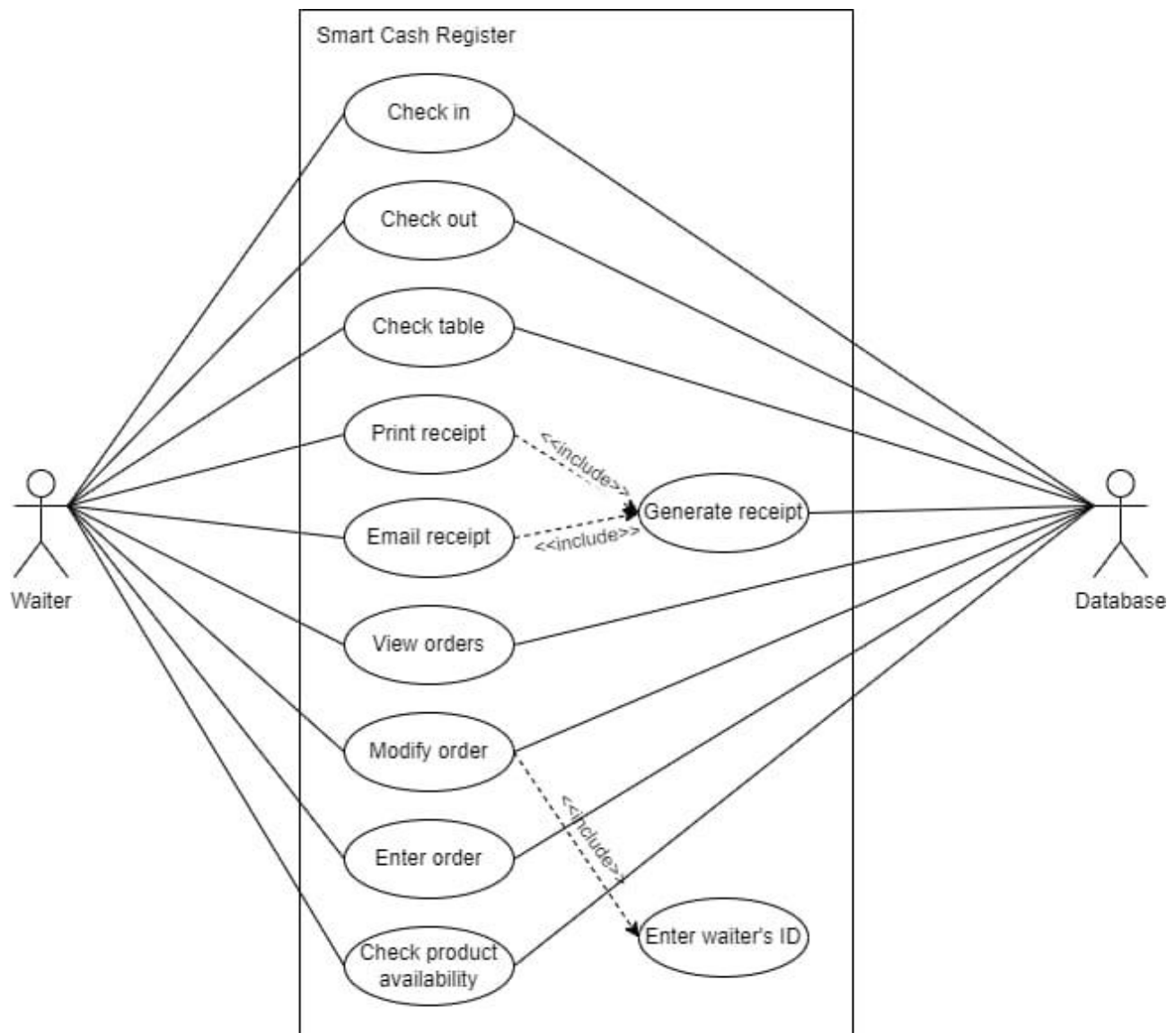


And, to sum up, the administrator's component diagram specifies how this component will be developed:

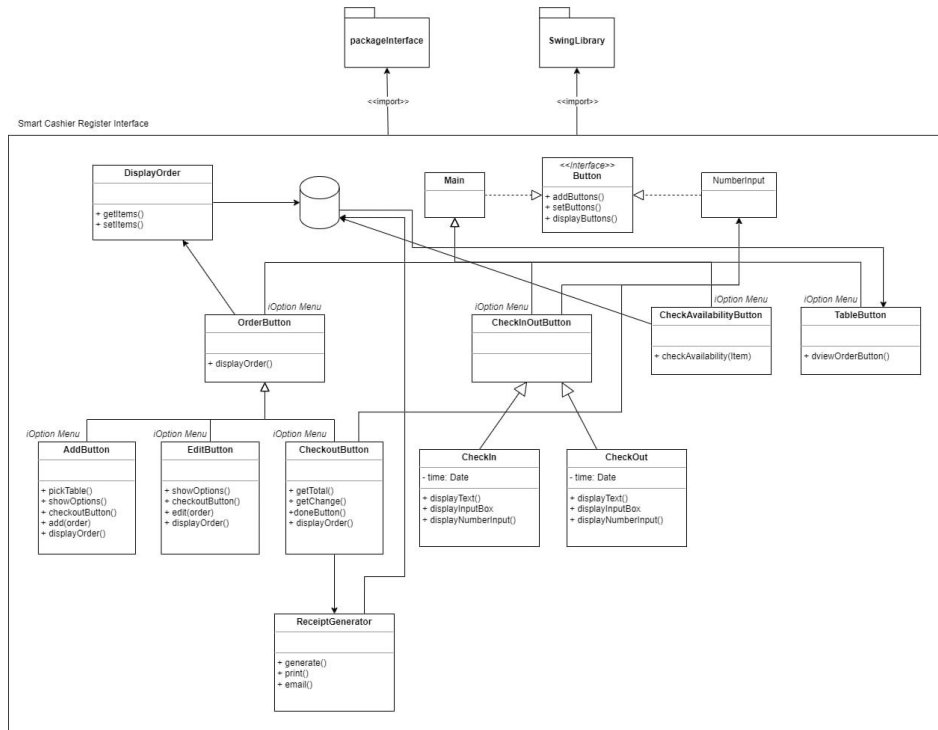


3.4 Smart Cash Register Specification

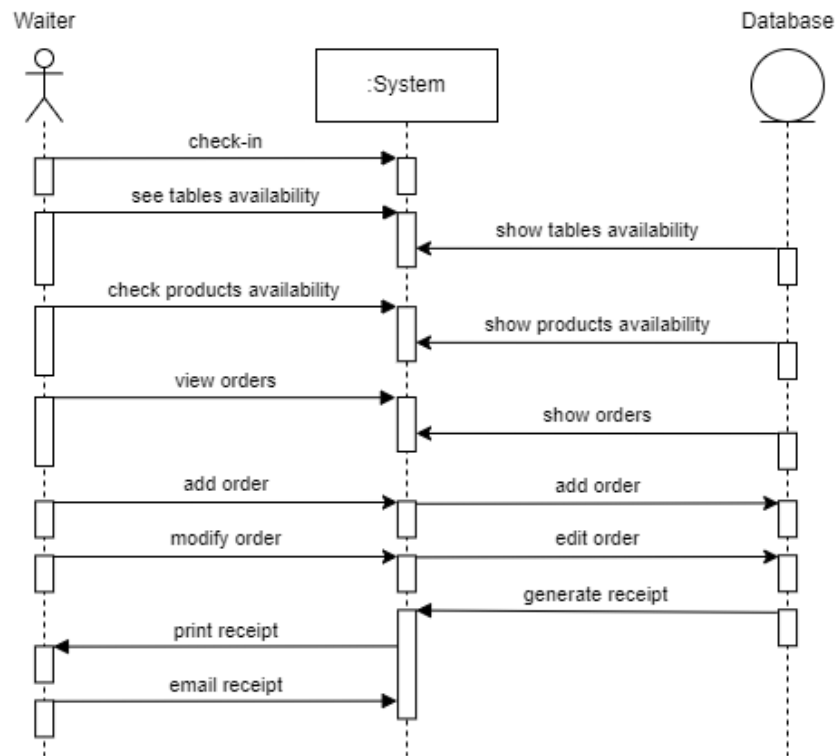
The SCR functionalities are very straight forward. It is simply an interface offering buttons which map to a functionality each. The goal of this component is to provide the waiters at a restaurant a user-friendly tool which makes them more efficient, and is also likeable to use. It must be noted that the check-in system is not meant to accomplish anything similar to a log-in system, but it is rather acting as a work clock.



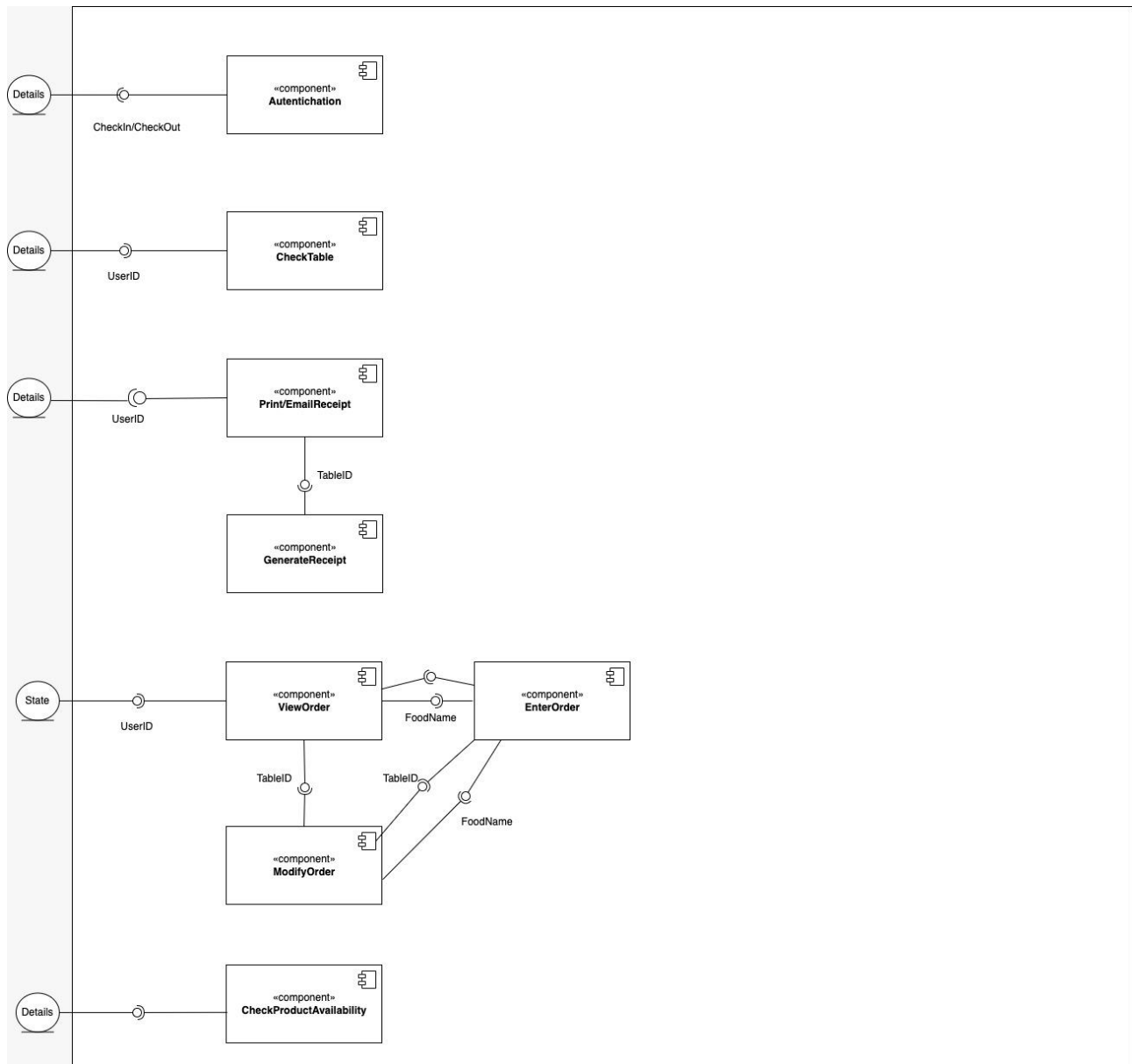
This component, similarly to the administrator app, would be implemented in such a way where there is a single window which is refreshed after every action and through the use of interfaces:



Contrary to the administrator's sequence diagram, in this case we can see how the interaction between the waiter and the database are more of a two-way communication, where the customer requests data in each task, even if it is not explicitly requested.

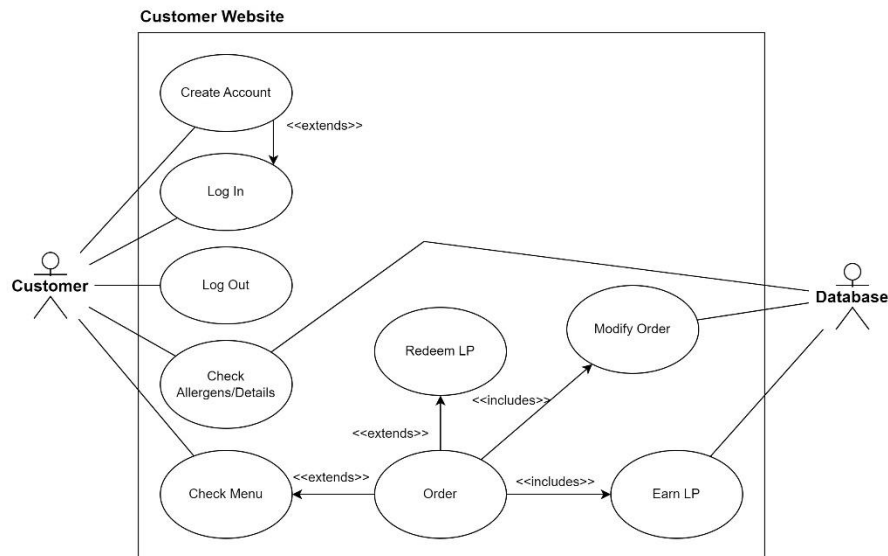


The SCR's component diagram also shows a noticeable difference in respect to the one for the administrator app. The SCR's diagram is much simpler and can be easily summarized into one component:

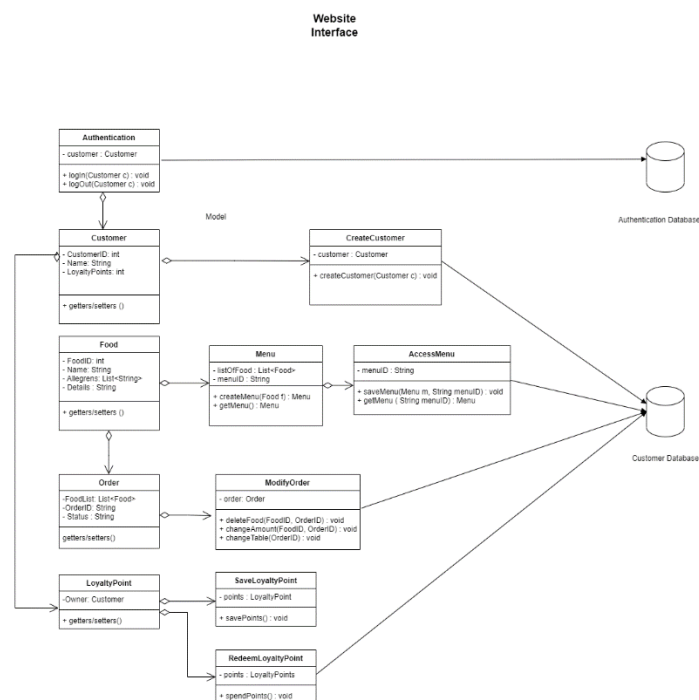


3.5 Customer Interfaces

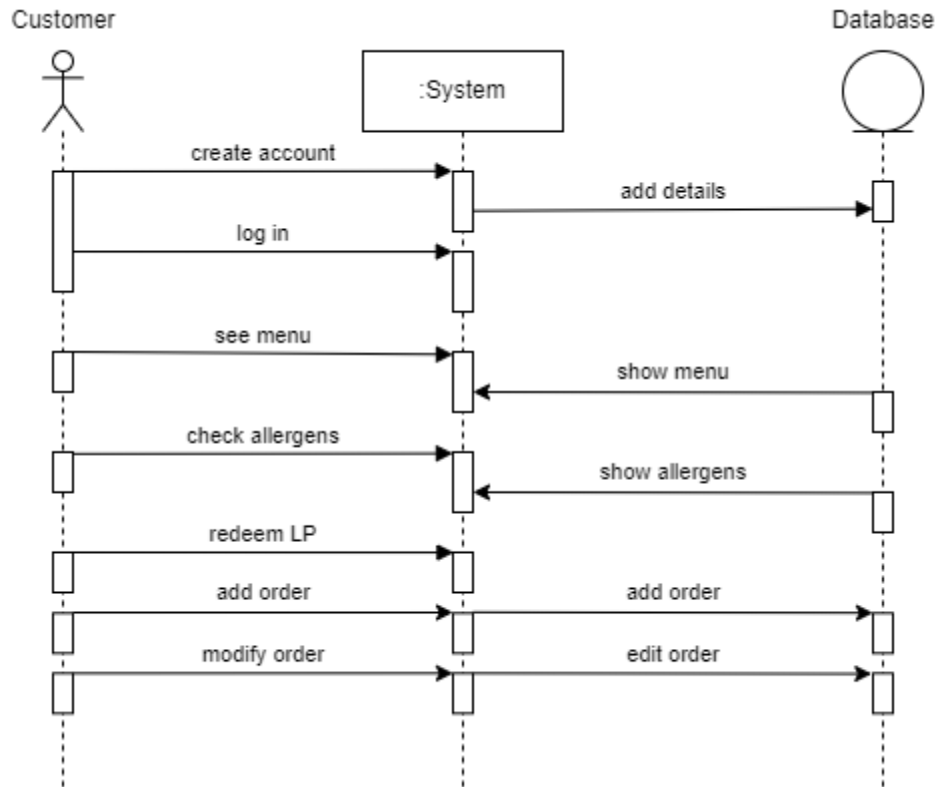
The CI's functionalities are few, but are of large importance. The website allows the customers to look at the menu and order. The website also connects to the database every time it is being accessed. The information shown to the user should take into consideration real-time setbacks, such as running out of a product.



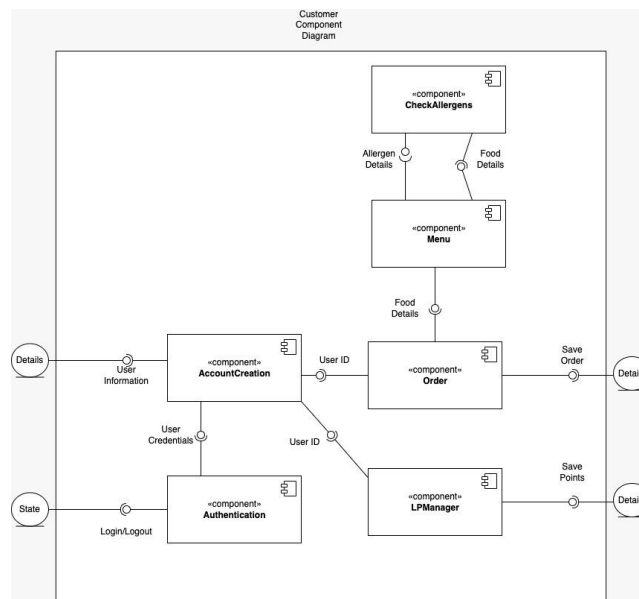
This component, contrary to the previous two, does not rely on interface and abstract elements as it would not be implemented in Java, but in HTML and PHP. But, similarly to the previous two, we can still appreciate a large dependency on the database:



Like in the administrator's sequence diagram, for this component there is mostly one-way communications with the database. The only times the component requests data from the database is when it needs to load the contents of the menu, but other than that, it is mostly about the customer writing data on the database(their order) or modify contents(their personal information / order).



And, to completely sum up, the Customer Website's component diagram:



4. Software Implementation

4.1 Tools and Technology Used

4.1.1 AAP

The AAP was developed using Java for back- and front-end. The design and coding of the front end was assisted using Apache NetBeans, which provides an IDE to design a GUI based on the Java Swing library components as well as auto-generating the code for a created design. The auto-generated code still needed some further tweaks, but it can be found on the functions under the name “initComp#”.

4.1.2 SCR

The SCR, like the APP was developed using Java for both, back- and front-end. It's front-end development was also assisted using Apache NetBeans.

4.1.3 CI

The phone app was developed using Flutter for back- and front-end. The design was first made in Figma then coded in Flutter. Xcode's emulator was used for simulating the phone application. The coding was made in Android Studio IDE, and some additional coding added to Xcode, so it works both in Android and IOS smart devices including tablets. The logo was made in Canva.

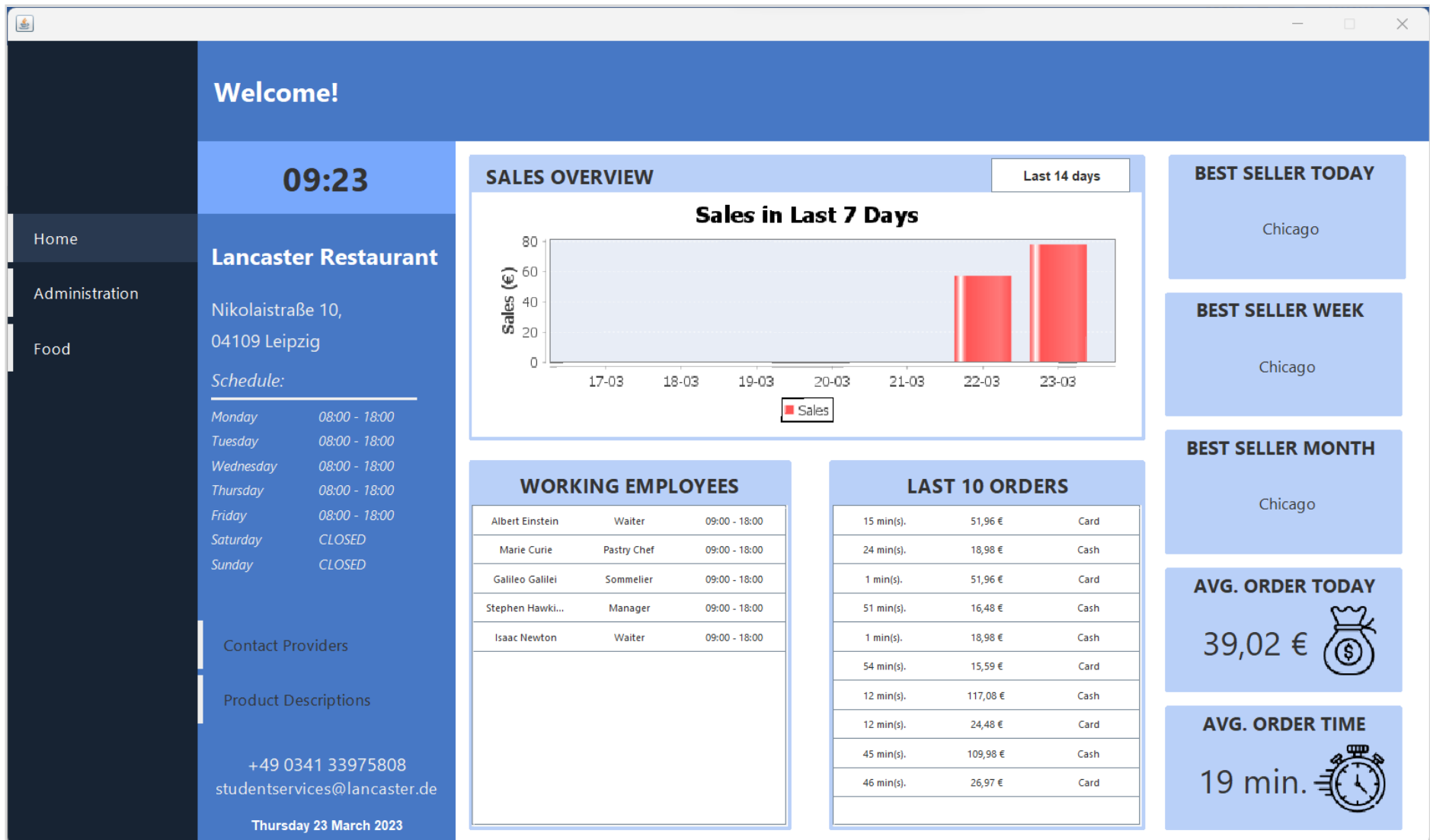
The website was created using Javascript, CSS and HTML for the front-end and PHP for the back-end. The website's used code corresponding to the front-end was taken from W3Layouts, and it is clearly noticed at the bottom of the page. Since each restaurant's website is dynamically updated through PHP code, and this code needs to have access to the database. In order to achieve this, I needed to use XAMPP, an open-source cross-platform web server solution. XAMPP allows to easily create a web server as well as software to administrate a database through phpMyAdmin.

4.1.4 Links to GitHub

- AAP : <https://github.com/juandiseg/adminApp>
- SCR : <https://github.com/juandiseg/smartCashReg>
- Phone App : <https://github.com/juandiseg/phoneApp>
- Website : <https://github.com/juandiseg/websiteEatNBeat>

4.2 Screenshots of the Application

4.2.1 AAP



Welcome!

09:18

Home

Administration

Food

Lancaster Restaurant

Nikolaistraße 10,
04109 Leipzig

Schedule:

Monday	08:00 - 18:00
Tuesday	08:00 - 18:00
Wednesday	08:00 - 18:00
Thursday	08:00 - 18:00
Friday	08:00 - 18:00
Saturday	CLOSED
Sunday	CLOSED

Contact Providers

Product Descriptions

+49 0341 33975808
studentservices@lancaster.de

Thursday 23 March 2023

Generate Provider Emails

Provider

REWE

From

01-01-2023

Until

04-04-2023

Generate Email

provider@rewe.de

Copy to Clipboard

Dear [name],

I am contacting from the Restaurant [name], based in [address].

We would like to request the following products to be provided to us:

- Nestea 0.33 Can : 2,00.
- Fanta 0.33 Can : 3,00.
- Coke 0.33 Can : 2,00.
- Beer Bottle : 2,00.
- Potatoes : 2,35.
- Flour : 9,80.
- Pepperoni : 3,15.
- Mozzarella : 0,00.
- Eggs : 16,00.

Copy to Clipboard

Home

Administration

Food

Welcome!

09:24

Reports

Shifts

Employees

Roles

Thursday 23 March 2023

Double-Click on Shift to edit it

From01-06-2022To01-06-2023

Apply

Employee	Role	Shift Date	Start	End	Check-In	Check-Out
Alan Turing	Chef	09-03-20...	12:00	18:00	11:59	18:03
Charles D...	Chef	09-03-20...	09:00	15:00	08:55	15:01
Isaac New...	Waiter	09-03-20...	12:00	18:00	11:59	18:03
Albert Ein...	Waiter	09-03-20...	11:00	18:00	10:59	18:01
Stephen ...	Manager	09-03-20...	09:00	16:00	09:01	15:45
Archimed...	Manager	09-03-20...	12:00	18:00	11:59	18:03
Nicolaus ...	Pastry Chef	09-03-20...	09:00	15:00	09:05	14:46
Marie Curie	Pastry Chef	09-03-20...	09:00	16:00	09:01	15:45
Nikola Tes...	Sommelier	09-03-20...	09:00	15:00	08:55	15:01
Galileo Ga...	Sommelier	09-03-20...	11:00	18:00	11:02	18:01
Charles D...	Chef	10-03-20...	09:00	15:00	08:55	15:01
Alan Turing	Chef	10-03-20...	12:00	18:00	12:10	17:55
Albert Ein...	Waiter	10-03-20...	11:00	18:00	11:00	18:01

Check Undertime

Sort by Employee

Add Shifts

4.2.2 SCR

The image displays a Software Requirements Canvas (SCR) for a system named "Beat n' Eat". The canvas is organized into several functional areas:

- Top Left:** A header area with two columns labeled "Product" and "Quantity". Below this is a large, empty rectangular area, likely intended for a list or table of products.
- Bottom Left:** A numeric keypad layout consisting of a 4x3 grid of buttons. The buttons are labeled: 1, 2, 3 (top row); 4, 5, 6 (second row); 7, 8, 9 (third row); and ., 0, <- (bottom row).
- Center:** A vertical toolbar containing five buttons: "Add", "Delete", "Pay", "Tables", and "Orders". Below these are two more buttons: "Availability" and "Check".
- Top Right:** A 3x3 grid of nine buttons labeled "Table 1" through "Table 9".
- Bottom Right:** Two large, empty rectangular areas labeled "Products" and "Menus", positioned side-by-side.

Beat n' Eat

Order: 13

Product	Quantity
Neapolitan	3

Add

Delete

Pay

Tables

Orders

Availability

Check

Order 13

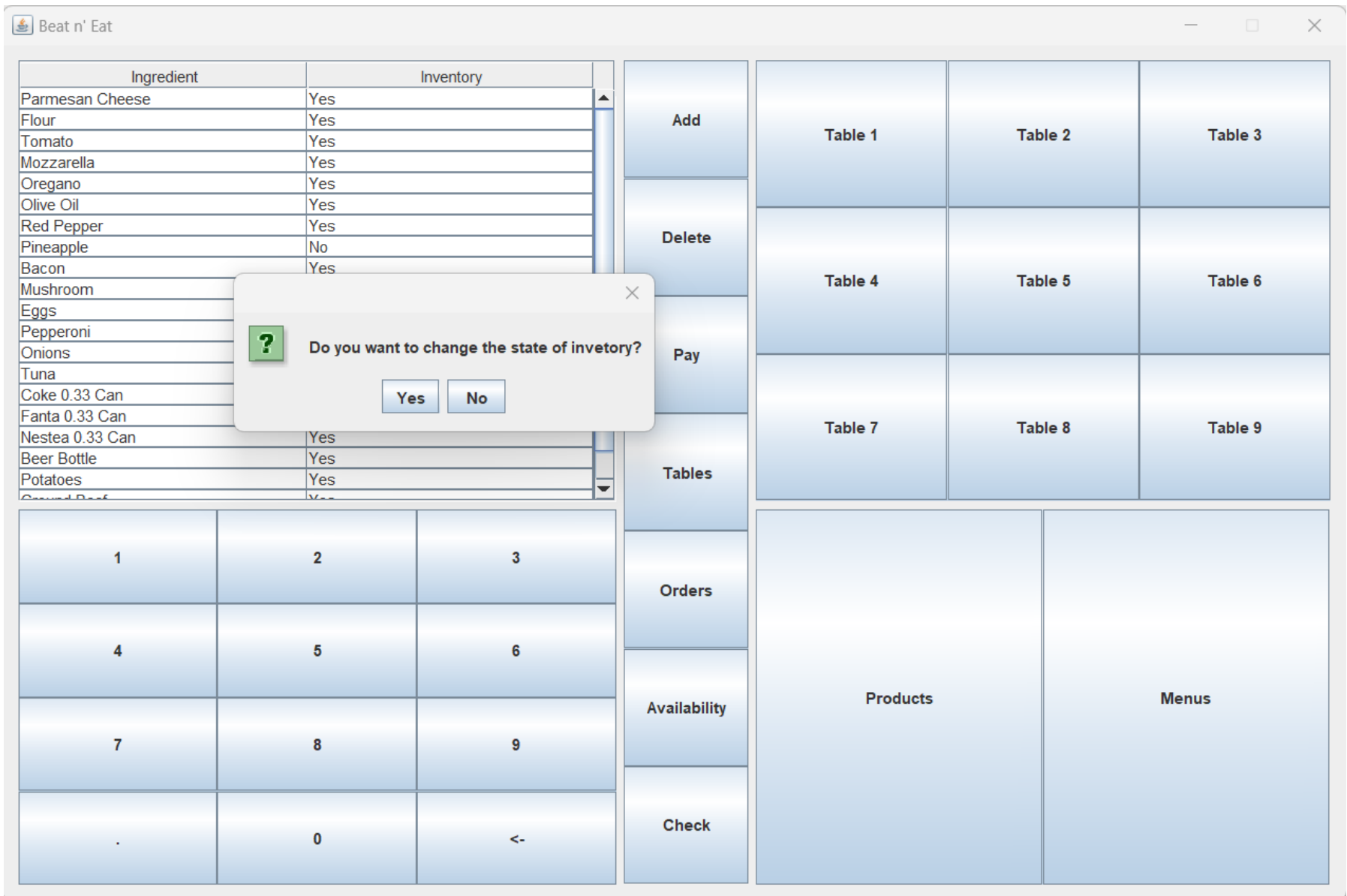
Menu 2 Seasons Deluxe

Healthy Squad

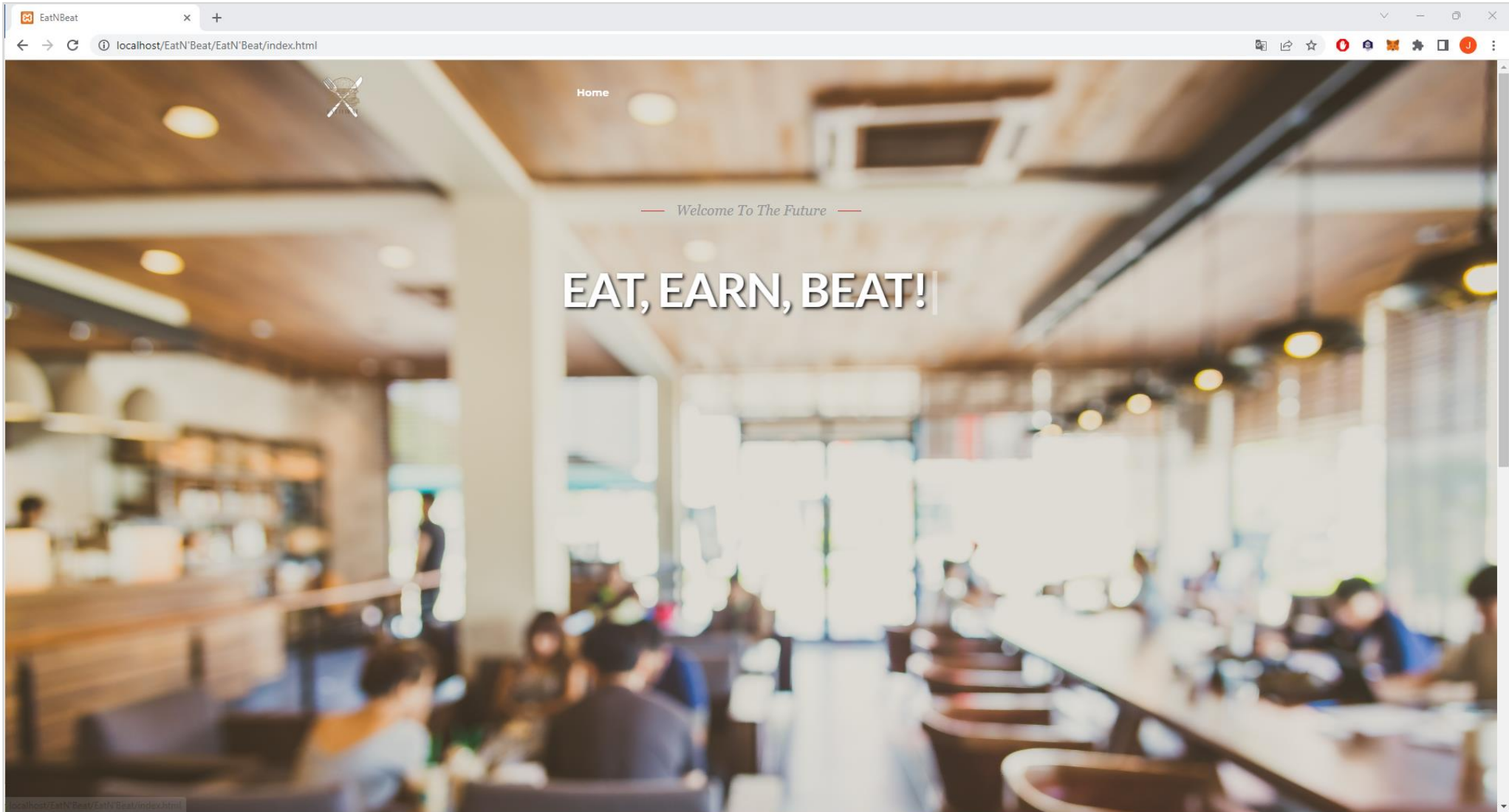
Carnivorous Family

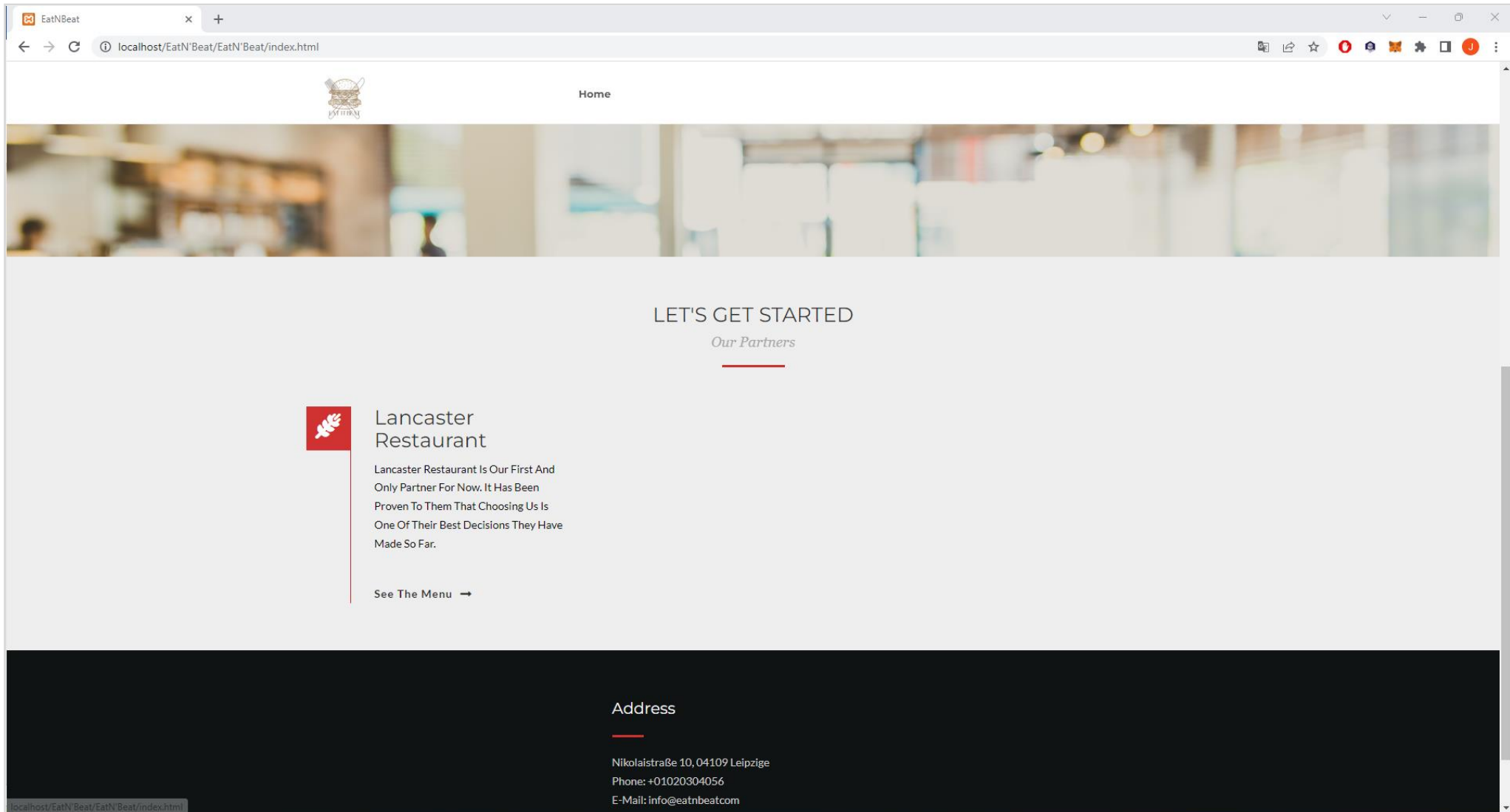
<-

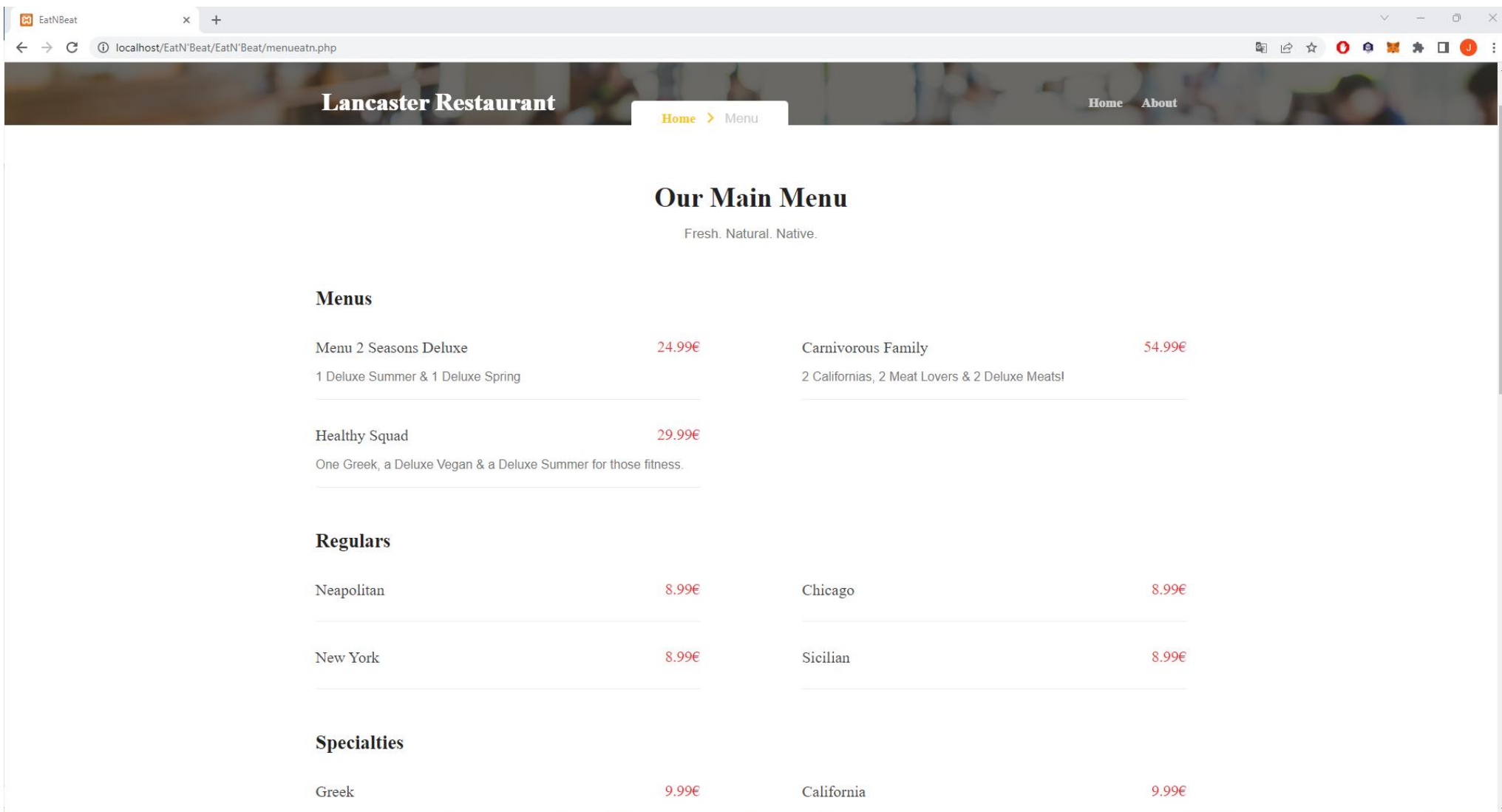
1	2	3
4	5	6
7	8	9
.	0	<-



4.2.3 CI








11:27

< Sign In



Email


Password

Log In

Don't have an account? [Sign Up](#)

11:33

< Sign Up



Welcome To Eat n Beat

Name

Email

Password

Confirm Password

Sign up

11:31

< Menus

Welcome Burak Alp !

5 - Pizza Menu

- 0 Menu 2 Seasons Deluxe
Price: \$19.99
- 1 Carnivorous Family
- 2
- 3
- 4

Select a table

Table 1

Table 3

Table 4

Table 5

Table 6

Table 7

Table 8

Order Summary:

Turkish Pizza Menu
Price: \$19.99

Test Menu
Price: \$17.99

Order

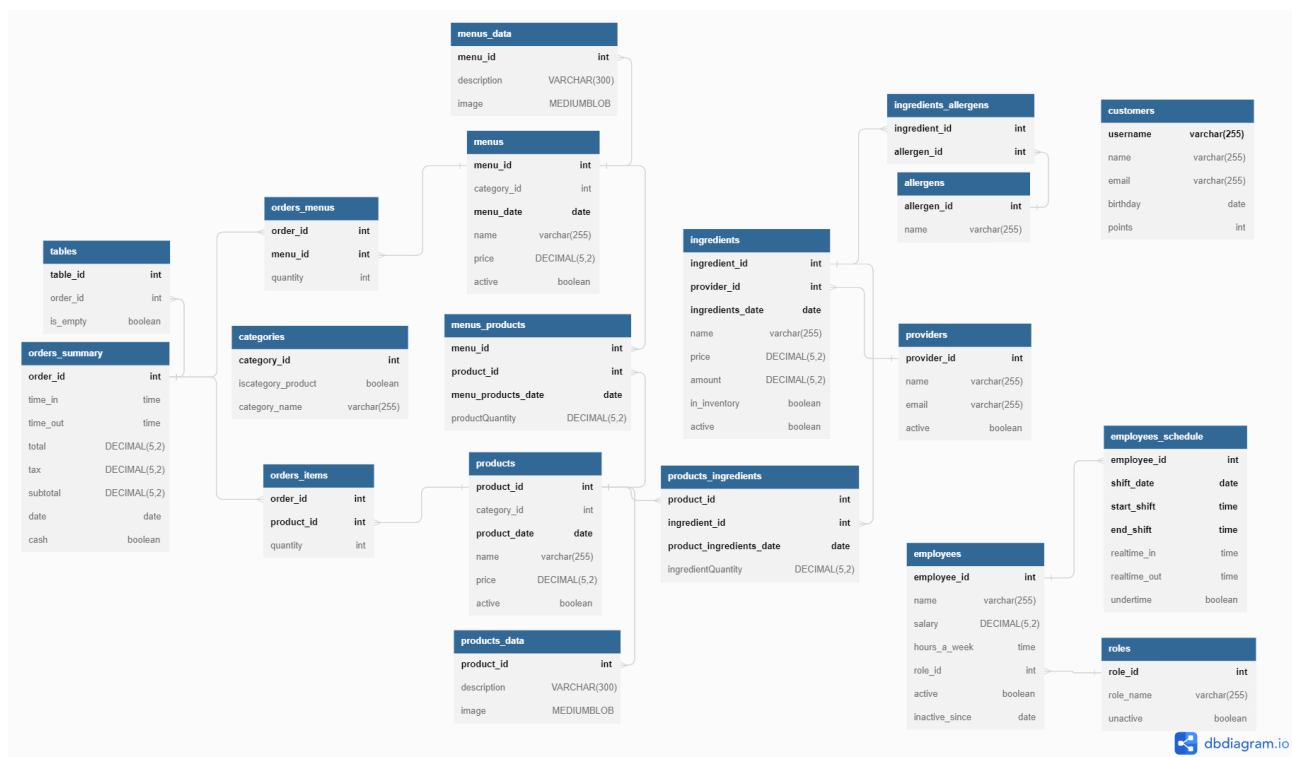
4.3 Changes during Implementation

4.3.1 Database

During the implementation of the system, we realized that although the DB seemed take everything into account, there were things left out. Among the most significant changes are:

- The addition of the “categories” table, so products and menus could be categorized in a sensible way.
- Similarly, the table “roles” was added to give employees a role which has data integrity.
- The “products_data” and “menus_data” tables were also added as there was no way to store either a description or an image of a product/menu. By adding this table, it also allowed to dynamically update a product’s / menu’s description on the CI, instead of having to manually change the source code every time a change is to be made.
- The table “orders_menus” was added as well, as there was no table tracking if a menu had been ordered in a certain order.
- All columns that were regarding prices had to be changed to support two decimal numbers. Before, their functioning was irregular and also not precise.

This is what the current architecture of the database looks like:

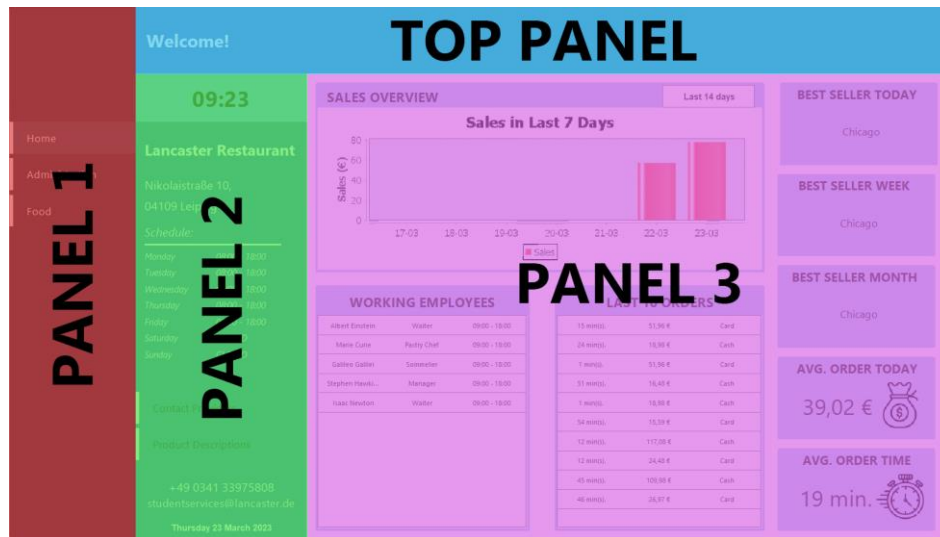


The commands to create the database can be found in the text file: “commandsDB.txt”. And the database can be populated running on the DB manager the commands found in the text file “populateDB.py”.

4.3.2 AAP

The AAP was at first implemented just like specified in the Class Diagram specified at [index 3.4](#). With that class structure all the back-end functionalities were implemented, but once the GUI was changed to its current state the class diagram needed refactoring.

In the original class diagram, the whole JFrame would be updated with new components, but given the new design, now this would not be necessary, so the “packageInterface” was dropped entirely. Instead, the JFrame is divided into 4 JPanels distributed as following:

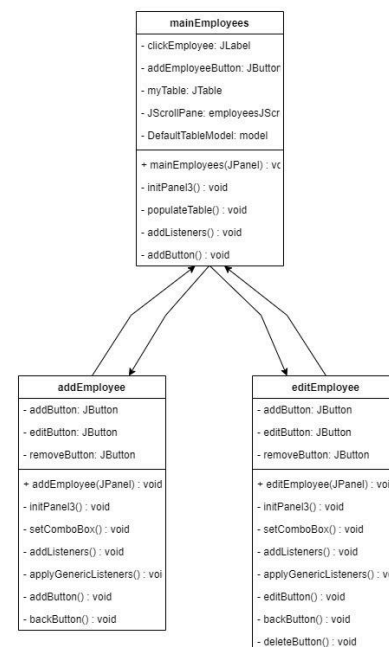


PANEL 1 and the TOP PANEL always remains the same, so it is only initiated when the app is booted. When an option in PANEL 1 is clicked, it removes the contents of PANEL 2 and 3, and initiates them again with the corresponding components. PANEL 3's contents are also reinitiated when an option from PANEL 2 is selected.

With this new distribution, the classes for each menu would only ask as parameters the needed panels, leading to not having to refresh everything entirely and needing to apply some changes to the class diagram.

The distribution of the functionalities was also modified during implementation. In the class diagram at [index 3.4](#) there was a menu, and therefore a class, for checking, editing, adding, and deleting. This was changed as some of these could be combined. Instead, the new distribution of the functionalities consists of main (which displays a table with all contents, so it behaves like checking), adding, and editing (which also implements the delete option). This way, the number of classes is far lower and clearer to understand what each has to do.

This is an example of what all GUI classes related to employee do now.



[illegible]

Also, as things going it was soon found the need to create some classes behaving as APIs, as we could retrieve information about products, ingredients, menus... from the database, but we needed them to be their own objects for code readability as well as reusability. This food components can be found in package called “componentsFood”. These only hold values and allow to set them to new values and retrieve them.

Finally, some patterns were used to ease possible extension of the code in the future, such as: implementing the factory pattern when a report is to be generated, facilitating easily extending the number of reports that can be provided. This can be found at **Appendix A**.

Another design choice to mention is that the AAP doesn't allow window resizing. The reason for this is that Java Swing, as it is a very old library, it does not adapt too well to modern user interfaces. So, in order to create an interface catching to the eye, it had to be done using the `AbsoluteLayout`, which functions with absolute coordinates and does not allow resizing.

4.3.3 SCR

The SCR actual class diagram is similar to what was described in the first SDS (notice, that in the SDS most of the classes had Button at the end, we deleted that for simplicity), but while the code was being done and because we changed the GUI design, some changes had to be made. The changes that were made in the diagram are:

- There are not add and edit button, they both do basically the same, so it wasn't necessary to create two different classes.
- The same goes for the check-in and check-out, we put both in the same class.
- Button interface wasn't required, instead we have another class called AbstractPanel, that works with a panel, creating a grid layout so you can add the buttons you want, and they will be put in an organised way, plus you can go back and forward in the panels. This class is basically only for the panel with the product's options to order.
- Adding to this AbstractPanel class, the classes that will make the panels with the options to order will extend this class.
- Last but not least, we also have a class called ManagerDB, this class connects to the database and make all the necessary queries. This facilitates accessing the needed data and is more organized.

In a folder called objects are some more classes that were really useful for the program, representing some things we needed, like categories, products, orders, among others.

Regarding the requirements specified in the section "Software Requirements Specification", all the functional requirements have been successfully achieved. But there was one nonfunctional requirement which had to be dropped. This is "REQ-2.4.3.1", this one consisted of sending an email notifying the administrator when an ingredient has ran out. To send an email we had to count many things into consideration that we didn't, for example the restaurant needs to have a real email with a password to be able to send emails, and then because the cash register in real life wouldn't be in a laptop but just one screen, we would have to create a keyboard with all the character we need to only write the email direction. They were requirements that weren't really feasible.

4.3.4 CI

The customers interfaces where in the beginning meant to both carry all functionalities of the system, and to be up to the user to decide which to use. But as the project was coming together, we realized it would be better if each component had its own purpose, instead of both fulfilling the same. In order to do this, we decided to have the website act as an information provider.

The website would only dynamically display the menu according to what's in inventory. This way, any customer could still access the menu from their phones without the need of downloading the app, but loyal users, which would probably not mind to download the app, would need to use the app in order to earn LPs which could later be exchanged for rewards.

Then, the website fulfilled all its intentioned requirements, as they were limited to displaying the menu, but the app had to face some shortage on its functionalities. One of the ideas that had to dropped due to technical issues was having a QR code which could be read from a phone with the app installed to automatically detect which restaurant the customer is at. Other than that, the phone app can display the menus, order them, and select which table one is occupying; as well as allowing for logging-in/-out and signing-in/-out.

4.4 APIs and Libraries

Every component of the system, regardless of the technology they were built on, made use of just one external API in order to connect to the MySQL database. The APIs used for this purpose were:

- [JDBC](#) for the components built on Java.
- The native PHP driver [MySQLi](#) to dynamically update the website.
- [MYSQLI](#) flutter package used for dynamically updating the contents of the phone app.

4.4.1 AAP

- [JDBC](#), to connect and interact with the database.
- [Apache POI](#) to create Excel files containing the data of the reports.
- [JFreeChart](#) to create the charts displayed in the main menus.
- [Swing-jnafilechooser](#) to display to the user a file explorer to specify which folder to save the reports at. There is a Swing component which has the same functionality but looks outdated.

4.4.2 SCR

- [JDBC](#), to connect and interact with the database.
- [iText](#), to create the PDFs of the receipts once an order is paid for.

4.4.3 CI

- [MYSQLI](#) flutter package used for connecting the MySQL database.
- [Firebase](#) used for keeping data of the customers (e-mail and password), and also verifying the e-mail.
- [Flutter Material](#) package used for creating the widgets for the mobile app.
- [jQuery](#) is used for animations in the Website.

5. Software Testing

5.1 Automated Testing

Since we could not deploy the system on a database ran in a server, but rather having the database on a local machine, we could not do the testing regarding the nonfunctional performance requirements related to delay (REQ-2.3.1.2). Instead, the automated testing focused on verifying that the entries in the database made sense logically, regardless of their integrity. The scenarios we checked for are the followings:

- Check that the table matching a menu with its products has entries for every menu in the “menus” table. This assures that there are no menus made up of nothing.
- Check that the table matching a product with its ingredients has entries for every product in the “products” table. This assures that there are no products made up of nothing.
- Check that the total in the table “orders_summary” is the right addition of subtotal and taxes.
- Check that product categories are only being referenced from “products”.
- Check that menu categories are only being referenced from “menus”.

This is tested through a Python script named “automaticTestDB.py”.

5.2 Usability Testing

For each of the data collection techniques, we gave five tasks to our testing users who were new to the system in order to evaluate the quality of the system. Our testing user pool was made up of 6 international University Students aged 18-26.

Through these testing, we aimed to determine whether our developed GUI were usable and determine what’s improvable for our design based on user feedback.

We recorded the time to complete a given task, the number of errors, comments, as well as later user feedback based on the difficulty of performing each task (Single Ease Questions).

5.2.1 AAP

5.2.1.1 Usability Testing

User Name/Age/Gender:	Enrique de Llano, 21, Male.		
Task Description	Task Time	No. errors	Problems faced / Comments
Add an ingredient.	00:25	0	-
Add the added ingredient to a product.	01:13	2	Went in ingredients section first. Wasn't clear where to specify qty used
Change the amount of a product in a menu.	00:32	0	A bug was encountered where after modifying the amount used of a product it did not display properly.
Create a new employee.	00:43	1	Salary didn't specify "pay/hour", the user thought it to be monthly salary.
Change the description of a menu.	01:17	1	User thought this functionality would be under the food menu.

User Name/Age/Gender:	Tom Becker, 19, Male.		
Task Description	Task Time	No. errors	Problems faced / Comments
Add an ingredient.	00:40	0	-
Add the added ingredient to a product.	01:24	1	It is not clear where to specify quantity. Names could be larger.
Change the amount of a product in a menu.	00:29	0	-
Create a new employee.	00:29	0	-
Change the description of a menu.	00:46	0	User thought this functionality would be under the food menu.

5.2.1.2 Single Ease Question

Users 1 and 2 rated the tasks on a scale of 7, according to the question 0 - Failed to perform 1 - Very Difficult 7 - Very Easy			
		1	2
1	Add an ingredient.	7	7
2	Add the added ingredient to a product.	5	5
3	Change the amount of a product in a menu.	7	6
4	Create a new employee.	6	7
5	Change the description of a menu.	4	4
User Average		5.80	5.80
Total Average		5.80	

5.2.2 SCR

5.2.2.1 Usability Testing

User Name/Age/Gender:	Junior Sarr, 19, Male.		
Task Description	Task Time	No. errors	Problems faced / Comments
Check-in, assuming your ID is 0.	00:05	0	-
Add an order in table 7 consisting of medium fries and a Coca-Cola	01:30	5	Confused on whether to click the table name or order first
Change the quantity of the fried to 3.	00:10	0	-
Change inventory status of tuna.	00:06	0	-
Pay for the order you just created with cash.	00:23	1	-

User Name/Age/Gender:	Nikoleta Demosthenous, 20, Female.		
Task Description	Task Time	No. errors	Problems faced / Comments
Check-in, assuming your ID is 0.	00:09	0	-
Add an order in table 7 consisting of medium fries and a Coca-Cola	02:30	15	Confused on whether to click the table name or order first
Change the quantity of the fried to 3.	00:41	4	-
Change inventory status of tuna.	00:07	1	-
Pay for the order you just created with cash.	00:17	0	-

5.2.2.2 Single Ease Question

Users 1 and 2 rated the tasks on a scale of 7, according to the question			
0 - Failed to perform			
1 - Very Difficult			
7 - Very Easy			
		1	2
1	Add an ingredient.	7	7
2	Add the added ingredient to a product.	2	2
3	Change the amount of a product in a menu.	5	3
4	Create a new employee.	7	6
5	Change the description of a menu.	4	6
User Average		5.00	4.80
Total Average		5.92	

5.2.3 IC

5.2.3.1 Usability Testing

User Name/Age/Gender:	Tolga Ziya, 21, Male.		
Task Description	Task Time	No. errors	Problems faced / Comments
Sign-up.	00:40	0	The app could automatically detect when the email has been verified.
Order from menu	00:10	1	Order summary should be saved somewhere
Editing customer details	00:26	0	-
Checking LPs.	00:09	0	Loyalty points are static.
Log-in with signed account.	00:15	2	Could add adding an “I forgot my password” option

User Name/Age/Gender:	Ahmet Düşer, 23, Male.		
Task Description	Task Time	No. errors	Problems faced / Comments
Sign-up.	00:29	0	-
Order from menu	00:18	2	User accidently clicked for food while hovering down.
Editing customer details	00:20	0	-
Checking LPs.	00:05	0	-
Log-in with signed account.	00:13	1	The user like the “show password” option

5.2.3.2 Single Ease Question

Users 1 and 2 rated the tasks on a scale of 7, according to the question 0 - Failed to perform 1 - Very Difficult 7 - Very Easy			
		1	2
1	Add an ingredient.	5	6
2	Add the added ingredient to a product.	7	6
3	Change the amount of a product in a menu.	7	7
4	Create a new employee.	7	7
5	Change the description of a menu.	5	6
User Average		6.20	6.40
Total Average		6.30	

Overall, the conclusions from the Usability Testing are that the apps were in general very clear to follow (as they yield an average SUS score of 6), and only some minor tweaks were needed to be implemented in order to improve the usability.

Appendix A: Screenshots of Interfaces and Code Snippets

AAP. Factory pattern applied to generate reports:

```
3 public class reportGeneratorFactory {
4
5     // A factory is implemented for the report generator, as it is a functionality
6     // which may need to be extended in the future. The GUI for generating excel
7     // reports could also be used for generating any other type of report as well.
8     // It is only necessary to modify this class in order for "mainReports" to have
9     // access to newly implemented reports.
10
11     public static iReportable createReportGenerator(String request) {
12         iReportable reportGenerator = null;
13         if ("Sales Report".equals(request)) {
14             return new salesReportGenerator();
15         } else if ("Expenses Report".equals(request)) {
16             return new expensesReportGenerator();
17         }
18         return reportGenerator;
19     }
20
21     public static String[] getReportTypes() {
22         return new String[] { "Sales Report", "Expenses Report" };
23     }
24 }
25
```

AAP. Example of object created to handle the data provided by the DB APIs:

```
3 public class provider {
4
5     private final int id;
6     private String name;
7     private String email;
8
9 > public provider(int id, String name, String email) { ...
14
15 > public int getId() { ...
18
19 > public String getName() { ...
22
23 > public String getEmail() { ...
26
27 > public String toString() { ...
30
31 > public boolean equals(Object obj) { ...
40 }
41
```

Website. Example of how the information on the website is dynamically retrieved from the DB:

```
<div class="menu-section">
  <h3 class="menu-section-title">Menus</h3>
  <!-- list starts -->
  <div class="appetizers">
    <!-- Item starts -->
    <?php
    require "connect.php";
    $conditionalQuery = 'SELECT menu_id, name, price, description FROM beatneat.menus NATURAL JOIN menus_data WHERE active = true;';
    $conditionalResult = $conn->query($conditionalQuery);
    while ($row = $conditionalResult->fetch_assoc()) {
      $checkQuery1 = "SELECT 1 FROM products.ingredients NATURAL JOIN ingredients WHERE product_ingredients_date = (SELECT MAX(product_ingredients_date) FROM products.ingredients)";
      $checkResult1 = $conn->query($checkQuery1);
      if (mysqli_num_rows($checkResult1) == 0) {
        echo '
          <div class="menu-item">
            <div class="row border-dot no-gutters">
              <div class="col-8 menu-item-name">
                <h6>' . $row['name'] . '</h6>
              </div>
              <div class="col-4 menu-item-price text-right">
                <h6>' . $row['price'] . '</h6>
              </div>
            </div>
            <div class="menu-item-description">
              <p class="para">' . $row['description'] . '</p>
            </div>
          </div>';
      }
    }
  </div>
  <!-- list end-->
</div>
```

SCR. Showing how the class AbstractPanel abstract class contemplated in the SDS was adapted and implemented in the final product.

```
iomullony, 2 days ago | 1 author (iomullony)
5 // Abstract class to work with panels where you can go backwards
6 public abstract class AbstractPanel {
7
8     public static JPanel thePanel;
9     public static ManagerDB theManagerDB;
10    private AbstractPanel previousPanel;
11
12    public AbstractPanel() { }
13
14    > /** ...
19    > final public void setPanel(JPanel _thePanel) { ...
22
23    > /** ...
28    > final public void setManagerDB(ManagerDB _theManagerDB) { ...
31
32    > /** ...
35    > public void updateToThisPanel() { ...
41
42    > /** ...
45    > private void deleteContents() { ...
48
49    > /** ...
52    > public void updateToPreviousPanel() { ...
57
58    > /** ...
61    > private void refreshPanel() { ...
65
66    > /** ...
69    > public abstract void addComponents();
70
71    > /** ...
74    > public abstract void addActionListeners();
75 }
```