

Dawid Madej

Nr Albumu 124138

**Tytuł:** Aplikacja kawiarni z perspektywy  
administratora i użytkownika

**Nazwa Przedmiotu:**

Programowanie obiektowe

**Grupa laboratoryjna: 2**

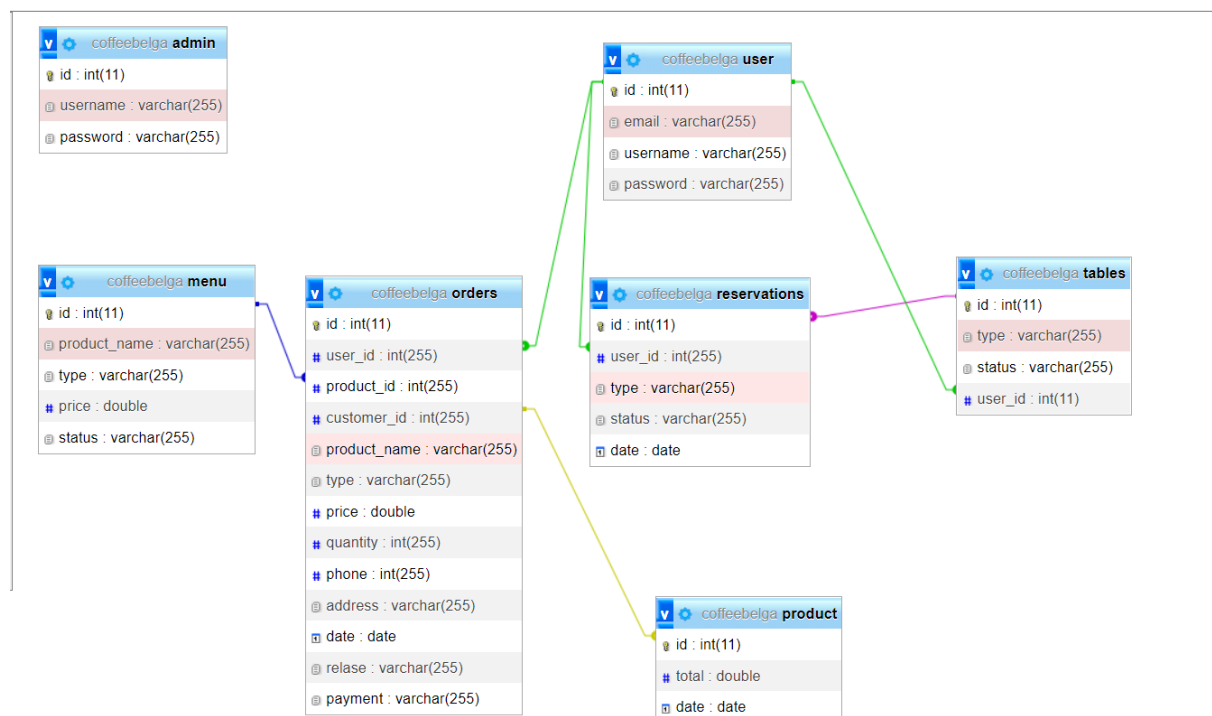
**Data oddania projektu:**

# 1. Wprowadzenie

Aplikacja ma na celu zasymulowanie działania kawiarni z widoku administratora oraz użytkownika. Użytkownik po założeniu konta, którego dane znajdują się w bazie danych może wykonywać takie operacje jak wyszukiwanie elementu z pozycji menu, rezerwacje stolików, a także złożyć zamówienie, które może następnie edytować, a następnie złożyć czy też sprawdzić jego stan. Aplikacja ze strony administratora przedstawia możliwość edycji elementów tabel z których właśnie korzysta użytkownik. Administrator posiada także wgląd na wykresy zarobków czy dostępność stolików. Aplikacja przeznaczona jest także na złożenie zamówień w samym lokalu, które przyjmuje administrator. Aplikacja opiera się na bazie danych w której wykonywane są różne operacje na tablicach powiązanych między sobą. Zawarte są także odpowiednie metody i walidacje, które zabezpieczają użytkownika przed odpowiednią rejestracją czy korzystaniem z aplikacji.

## 2. Wymagania systemowe

## 3. Baza danych



## Opis Tabel:

Aplikacja korzysta z 7 tabel:

- admin:

\*id \ Integer – klucz główny posiadający typ danych z jedną stałą wartością, gdzie przechowuje id jednego administratora z jednym przypisanym indeksem. Nie jest przypisany do żadnej tablicy, ponieważ tablica służy wyłącznie w celu posiadania danych potrzebnych do zalogowania administratora.

\*username \ Varchar – kolumna ta służy do przechowywania nazwy administratora w celu logowania za pomocą znaków z alfabetu, dlatego został wykorzystany tutaj typ zmiennych varchar.

\*password \ Varchar - kolumna ta służy do przechowywania hasła administratora w celu logowania za pomocą znaków z alfabetu, dlatego został wykorzystany tutaj typ zmiennych varchar.

- user:

\*id \ Integer – klucz główny posiadający typ danych zmiennej został przypisany automatycznie. Przechowuje wartości ID rejestrowanych użytkowników.

\*email \ Varchar – typ zmiennej Varchar został przypisany do kolumny email, ponieważ służy ona do przechowywania emaila użytkownika. Typ zmiennej wymaga wykorzystania ciągu liter.

\*username \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania nazwy użytkownika. Typ zmiennej wymaga wykorzystania ciągu liter.

\*password \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania hasła użytkownika. Typ zmiennej wymaga wykorzystania ciągu liter.

- menu:

\*id \ Integer - klucz główny posiadający typ danych zmiennej został przypisany automatycznie. Przechowuje wartości ID produktów zawartych w menu.

\*product\_name \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania nazwy produktu znajdującego się w menu. Typ zmiennej wymaga wykorzystania ciągu liter.

\*type \ Varchar – typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania jakiego typu jest produkt znajdujący się w menu. Typ zmiennej wymaga wykorzystania ciągu liter.

\*price \ Double - typ zmiennej Double został przypisany do danej kolumny, ponieważ służy ona do przechowywania wartości przecinkowej ceny. Typ zmiennej wymaga wykorzystania ciągu znaków liczb z dokładnością przecinkową.

\*status \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania statusu dostępności produktu znajdującego się w menu. Typ zmiennej wymaga wykorzystania ciągu liter.

- order:

\*id \ Integer – klucz główny posiadający typ danych zmiennej został przypisany automatycznie. Przechowuje wartości ID składanych zamówień.

\*user\_id \ Integer – został przypisany tutaj typ zmiennej Integer ponieważ kolumna przyjmuje wartości klucza głównego z tabeli user, gdzie połączona jest relacją ManyToOne. Kolumna przechowuje wartości id użytkownika, który złożył zamówienie.

\*customer\_id \ Integer - został przypisany tutaj typ zmiennej Integer ponieważ kolumna przyjmuje wartości klucza głównego z tabeli product, gdzie połączona jest relacją OneToOne. Kolumna przechowuje wartości id w pełni złożonego zamówienia.

\*product\_name \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania nazwy produktu znajdującego się w zamówieniu. Typ zmiennej wymaga wykorzystania ciągu liter.

\*type \ Varchar – typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania jakiego typu jest produkt znajdujący się w zamówieniu. Typ zmiennej wymaga wykorzystania ciągu liter.

\*price \ Double - typ zmiennej Double został przypisany do danej kolumny, ponieważ służy ona do przechowywania wartości ceny. Typ zmiennej wymaga wykorzystania ciągu znaków liczb z dokładnością przecinkową.

\*quantity \ Integer – typ zmiennej Integer został przypisany do danej kolumny, ponieważ służy ona do przechowywania ilości danych produktów w zamówieniu, która wymaga ciągu znaków liczb.

\*phone \ Integer - typ zmiennej Integer został przypisany do danej kolumny, ponieważ służy ona do przechowywania numeru telefonu na który składane jest zamówienie, która wymaga ciągu znaków liczb.

\*address \ Varchar – typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania adresu pod który ma zostać złożone zamówienie. Typ zmiennej wymaga wykorzystania ciągu liter.

\*date \ Date – typ zmiennej Date został przypisany do danej kolumny, ponieważ służy ona do przechowywania daty złożenia zamówienia. Typ zmiennej wymaga wykorzystania do tego aktualnej daty.

\*relase \ Varchar – typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania statusu realizacji zamówienia. Typ zmiennej wymaga wykorzystania ciągu liter.

\*payment \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania statusu dokonania płatności za zamówienie. Typ zmiennej wymaga wykorzystania ciągu liter.

- product:

\*id \ Integer – klucz główny posiadający typ danych zmiennej został przypisany przez wartość z tabeli order z której przyjęty jest indeks złożonego zamówienia. Przechowuje wartości ID przyjętych zamówień.

\*total \ Double - typ zmiennej Double został przypisany do danej kolumny, ponieważ służy ona do przechowywania całości zapłaty za zamówienie. Typ zmiennej wymaga wykorzystania ciągu znaków liczb z dokładnością przecinkową.

\*date \ Date – typ zmiennej Date został przypisany do danej kolumny, ponieważ służy ona do przechowywania daty złożonego zamówienia. Typ zmiennej wymaga wykorzystania do tego aktualnej daty.

- tables:

\*id \ Integer – klucz główny posiadający typ danych zmiennej został przypisany automatycznie. Przechowuje wartości ID oraz numer wprowadzanych stolików.

\*type \ Varchar – typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania jakiej wielkości jest stół. Typ zmiennej wymaga wykorzystania ciągu liter.

\*status \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania statusu dostępności stołu. Typ zmiennej wymaga wykorzystania ciągu liter.

\*user\_id \ Integer – został przypisany tutaj typ zmiennej Integer ponieważ kolumna przyjmuje wartości klucza głównego z tabeli user, gdzie połączona jest relacją ManyToOne. Kolumna przechowuje wartości id użytkownika, który posiada rezerwacje na dany stół.

-reservations:

\*id \ Integer – klucz główny posiadający typ danych zmiennej został przypisany za pomocą relacji OneToOne z tabeli tables z której przyjmuje wartości indeksu stołu. Przechowuje wartości ID stołów.

\*user\_id \ Integer – został przypisany tutaj typ zmiennej Integer ponieważ kolumna przyjmuje wartości klucza głównego z tabeli user, gdzie połączona jest relacją OneToOne. Kolumna przechowuje wartości id użytkownika, który posiada rezerwacje na dany stół.

\*type \ Varchar – typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania jakiej wielkości jest stół. Typ zmiennej wymaga wykorzystania ciągu liter.

\*status \ Varchar - typ zmiennej Varchar został przypisany do danej kolumny, ponieważ służy ona do przechowywania statusu czy stół został zarezerwowany. Typ zmiennej wymaga wykorzystania ciągu liter.

\*date \ Date – typ zmiennej Date został przypisany do danej kolumny, ponieważ służy ona do przechowywania daty rezerwacji. Typ zmiennej wymaga wykorzystania do tego aktualnej daty.

## 4. Instalacja

a) INSTRUKCJA INSTALACJI:

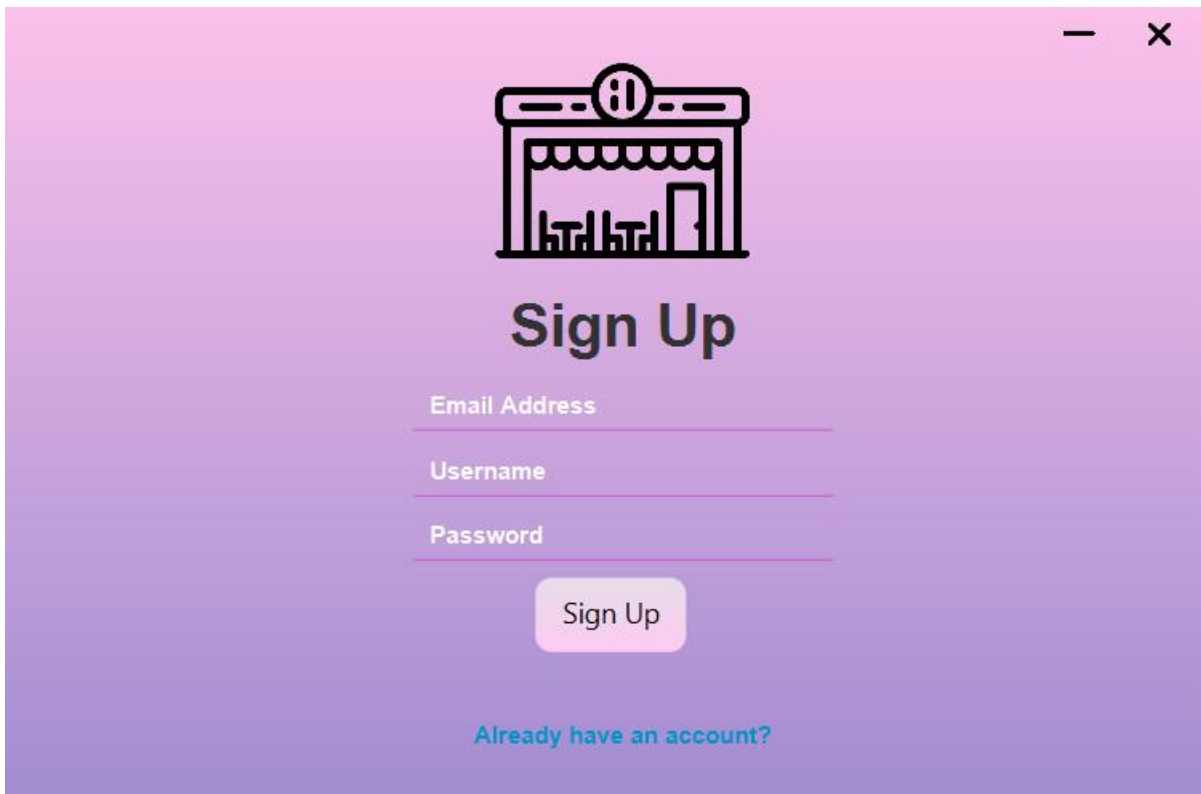
b) KONFIGURACJA SYSTEMU:

## 5. Interfejs użytkownika oraz funkcjonalność aplikacji.

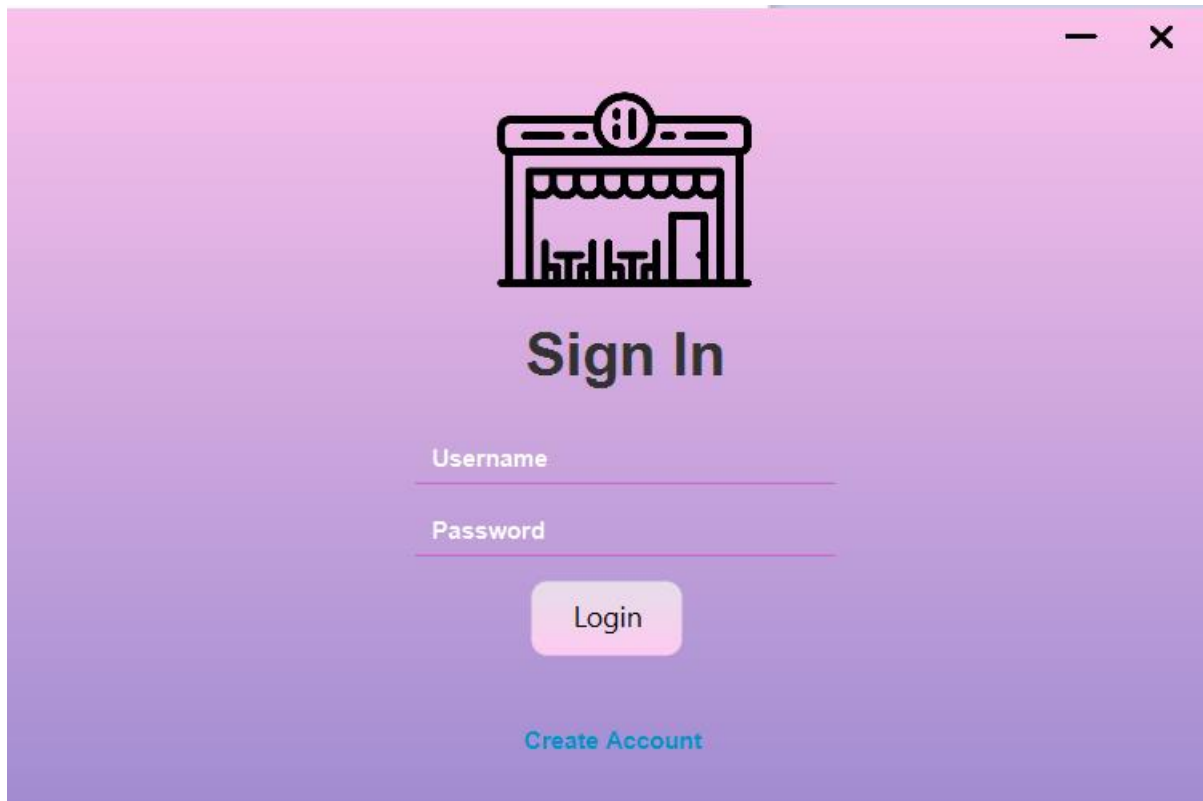
Lista funkcjonalności:

- Dodawanie elementów do tabel
- Usuwanie elementów z tabeli
- Edytowanie elementów z tabeli
- Wyszukiwanie elementów w tabeli
- Poruszanie się między oknami
- Transportowanie elementów między tabelami

Przebieg aplikacji:

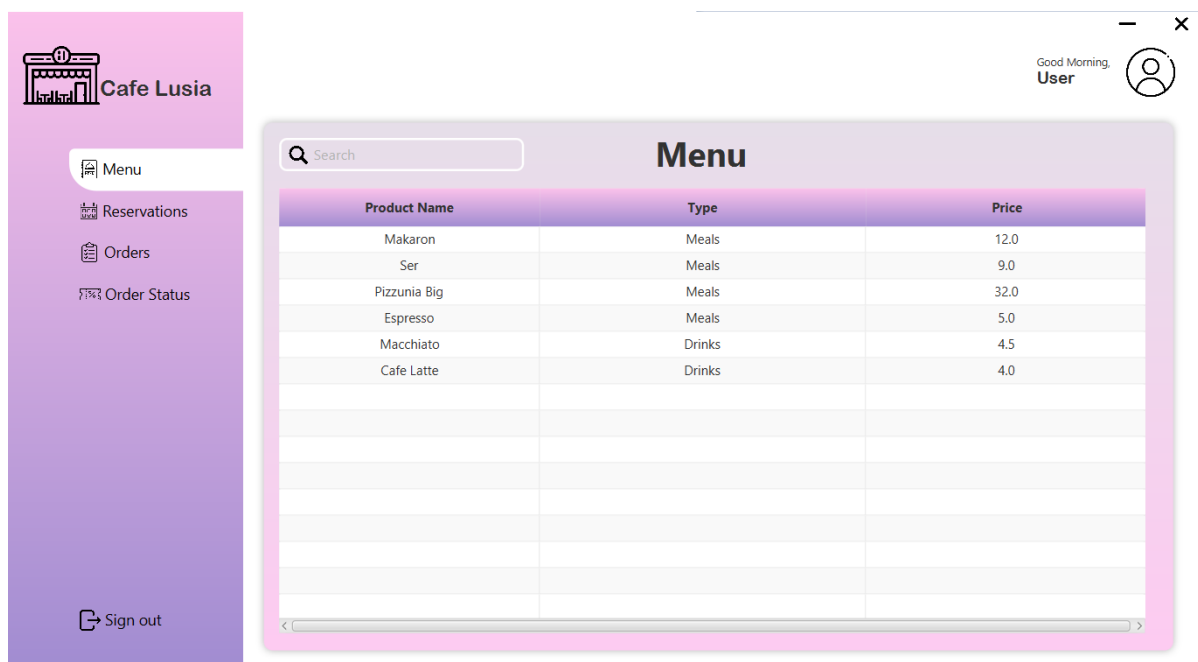


Pierwszym oknem jest ekran rejestracji w której użytkownik może utworzyć konto potrzebne do korzystania z aplikacji. Musi podać odpowiedni email, niepowtarzającą się nazwę oraz hasło przynajmniej 8 znakowe.

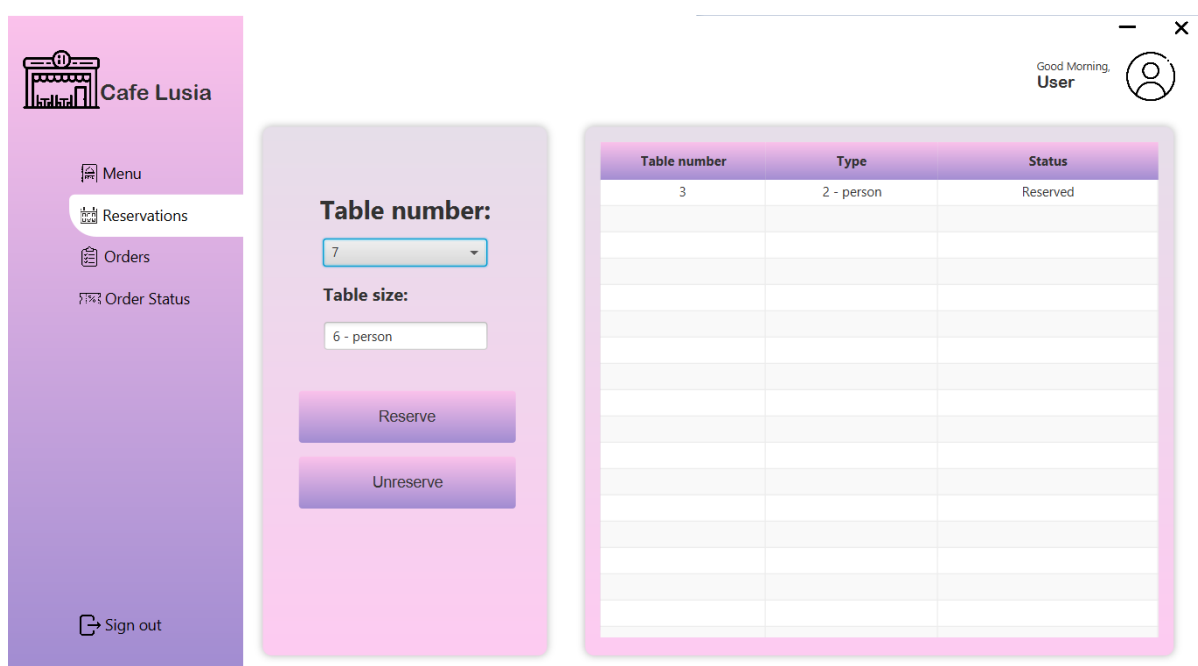


Następnie można przejść do okna logowania, w którym podaje się dane z pomyślnej rejestracji. W wypadku błędnej nazwy lub hasła program pokaże odpowiedni komunikat.





Po zalogowaniu program wyświetli pierwszą stronę, czyli Menu. Można sprawdzić na niej dostępne dania oraz napoje z ceną. Można także wyszukiwać dania jakie nas interesuje za pomocą jego nazwy, typu czy ceny.



Po wciśnięciu po lewej stronie przycisku 'Reservations' przejdziemy do zakładki z rejestracją, gdzie możemy zarezerwować lub odrezerwować jeden z wybranych przez stolików. Stoliki posiadają dany rozmiar, który po wybraniu numeru pokaże się w poniżej.

**Cafe Lusia**

- Menu
- Reservations
- Orders**
- Order Status

Sign out

Product Name	Type	Price	Quantity
Macchiato	Drinks	4.5	1

Product Name: Macchiato

Quantity: 1

Phone: 123123123

Address: Fake123

Add

Total: **4.5zł**

Pay Remove

W zakładce 'Orders' możemy złożyć zamówienie wybierając nazwę produktu, a następnie zaznaczając ilość i wypełniając pola numeru telefonu oraz adresu. Po dodaniu dania czy napoju za pomocą przycisku 'Add', możemy za nie zapłacić lub usunąć błędny element po zaznaczeniu go na tablicy.

**Cafe Lusia**

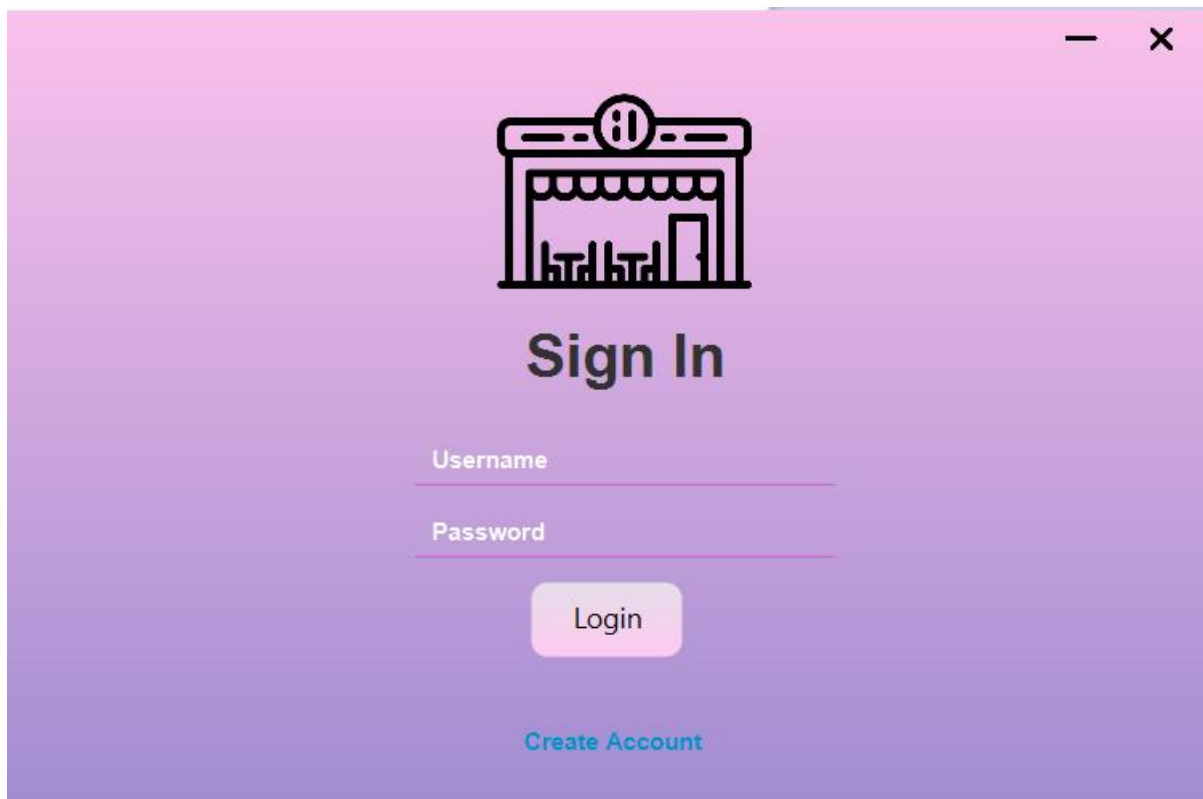
- Menu
- Reservations
- Orders
- Order Status**

Sign out

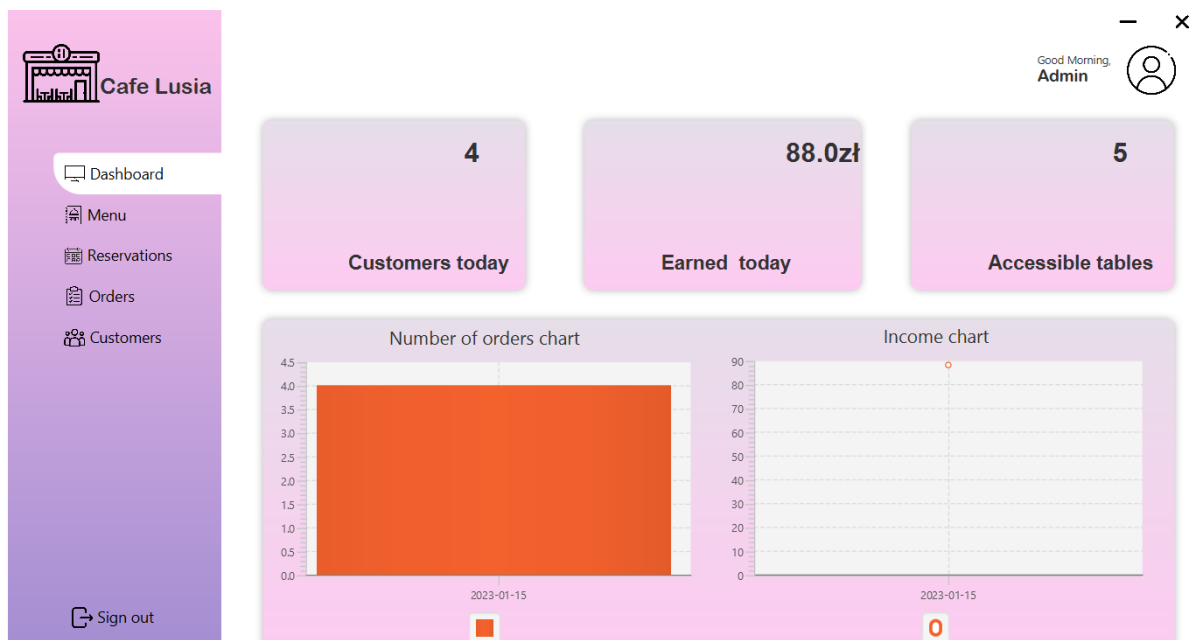
Product Name	Type	Price	Status	Payment
Ser	Meals	18.0	Dispatched	done
Macchiato	Drinks	4.5	In process	Waiting for payment

Przechodząc do zakładki 'Order Status', możemy sprawdzić tam nasze zamówienie, nazwe produktów, ich typ, cenę, status w jakim aktualnie się znajdują oraz czy została za nie dokonana płatność.

Aby wylogować się z aplikacji wciskamy przysick w lewym dolnym rogu 'Sign out' i zatwierdzamy nasz wybór.



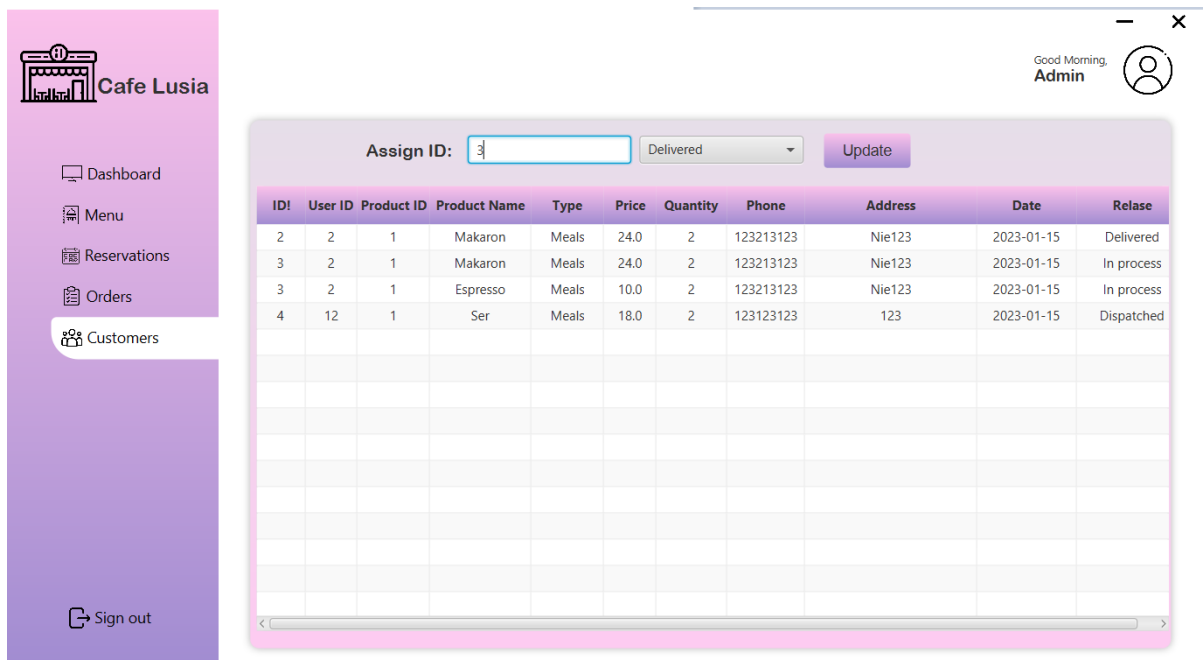
Znajdujemy się ponownie przy oknie logowania. Aby zalogować się do konta admina wpisujemy nazwę 'admin', hasło 'admin'.



Wyświetli się ekran na którym jest możliwość sprawdzenia ilości dzisiejszych klientów, zarobionej wartości czy dostępnych stolików. Pokazują się także wykresy ilości zamówień oraz zarobionych pieniędzy na określony dzień.



[illegible]



Ostatnim oknem w aplikacji jest 'Customers', gdzie jest wgląd na wszystkie opłacone zamówienia. Zapisane są tu wszystkie informacje na temat zamówień z aktualnego dnia. Można zmieniać tutaj status zamówienia, wprowadzając w pole 'Assign ID' indeks zamówienia oznaczony 'ID!', a następnie wybrać jego status i zaktualizować go za pomocą przycisku obok.

Można wylogować się z aplikacji dokładnie tak samo jak w wersji użytkownika.

## 6. Bezpieczeństwo

Do bazy danych wystarczy jedynie login, przez co nie robi jej to dobrze zabezpieczonej. W Aplikacji zabezpieczenie jest znacznie większe, gdzie użytkownik jak i Administrator posiadają nazwy oraz hasła.

Zalogowany użytkownik nie może przeprowadzać interakcji pod innym indeksem jak nie swoim. Przez co nie może wystąpić błąd polegający, że któryś z użytkowników zamówi coś na inne konto. Tak samo nie może kontrolować konta administratora, na które ten jedynie ma wgląd.

Pola w aplikacji posiadają walidacje, która nie pozwala być im pusta w trakcie wystawiania lub edycji danych do bazy danych lub pola liczbowe mają posiadać jedynie liczby.

## 7. Dokumentacja API

### 1) Klasa Database

Jest to klasa która ma na celu połączyć się z bazą danych, przy użyciu interfejsu oprogramowania JDBC. Klasa ta zawiera jedną metodę `connectDb()` która służy właśnie do połączenia z bazą danych.

### 2) Klasa DashboradController

Służy do posługiwania się metodami dla ekranu administratora. Posiada metody, które pozwalają na

- tworzenie rekordów
- usuwanie rekordów
- aktualizacje rekordów
- pobieranie rekordów
- przełączanie między ekranami

Metody:

\*`dashboardCT()`: Przyjmuje wartość 'id' z tabeli 'product' w danym dniu, które następnie podlicza. Służy do wyświetlenia ilości zamówień na dzień.

\*`dashboardET()`: sumuje wartości 'total' z tabeli 'product' z aktualną datą, a następnie wyświetla je jako tekst w aplikacji.

\*`dashboardAT()`: sumuje wartości 'id' z tabeli 'tables', gdzie status wynosi 'Not reserved', a następnie wyświetla w aplikacji jako dostępne stoliki.

\*`dashboardNOOC()`: metoda pobiera podliczane wartości 'id' z tabeli product, które grupowane według daty, a następnie wyświetla jak chart. Metoda ustawia limit pobrania danych z 5 dni.

\*`dashboardIC()`: sumuje pobrane dane kolumny 'total' z tabeli product, grupowanych przez date, a następnie wyświetla je z limitem do 5 ostatnich dni.

\*menuAdd(): wysyła do tabeli 'menu' wartości wprowadzone przez użytkownika. Metoda w trakcie sprawdza czy każde pole zostało wypełnione, aby do bazy nie wysłano żadnego null. Sprawdza także czy istnieje już wartość o tym samym nazewnictwie.

\*menuUpdate(): sprawdza czy każde pole zostało odpowiednio wypełnione przez użytkownika, a następnie aktualizuje wybrane wartości w bazie danych w tabeli 'menu'.

\*menuClear(): czyści wszystkie TextFieldy i ComboBoxy.

\*menuDelete(): sprawdza, czy każde pole zostało odpowiednio uzupełnione, a następnie wysyła zapytanie o usunięciu rekordów z tabeli 'menu'

\*menuShowList(): zapisuje wszystkie rekordy z tabeli 'menu' do ObservableList za pomocą get'ów z klasy 'Menus'.

\*menuShow(): wyświetla ObservableList z metody menuShowList() w wybranym TableView.

\*menuSelect(): metoda umożliwia zaznaczenie elementu z TableView i wyświetlenie jej danych w TextField i ComboBox.

\*menuTypes(): zamieszcza wartości z tablicy 'types' do Listy 'listTypes', a następnie wyświetla jej wartości w ComboBox.

\*menuStatus(): zamieszcza wartości z tablicy 'status' do Listy 'listStatus', a następnie wyświetla jej wartości w ComboBox.

\*menuSearch(): służy do wyszukiwania elementów w sposób przypisania wyszukiwanych wartości do FiltredList<Menus>, a następnie zmianie wartości TableView na te z listy

\*reservationAdd(): wysyła do tabeli 'tables' wartości wprowadzone przez użytkownika. Metoda w trakcie sprawdza czy każde pole zostało wypełnione, aby do bazy nie wysłano żadnego null. Sprawdza także czy istnieje już wartość o tym samym numerowaniu.

\*reservationUpdate(): sprawdza czy każde pole zostało odpowiednio wypełnione przez użytkownika, a następnie aktualizuje wybrane wartości w bazie danych w tabeli 'tables'.



\*reservationClear(): aktualizuje tabele 'tables' na status = 'Not reserved' oraz user\_id = '1'(wartość domyślna). Usuwa także rekordy z tabeli 'reservations', gdzie status = 'Reserved'. Przed dokonaniem zmian w bazie danych, metoda sprawdza również czy potrzebne pola zostały powypełniane.

\*reservationShowList(): zapisuje wszystkie rekordy z tabeli 'tables' do ObservableList za pomocą get'ów z klasy 'Tables'.

\*reservationShow(): wyświetla ObservableList z metody reservationShowList() w wybranym TableView.

\*reservationSelect(): metoda umożliwia zaznaczenie elementu z TableView i wyświetlenie jej danych w TextField i ComboBox.

\*reservationTypes(): zamieszcza wartości z tablicy 'typesRe' do Listy 'listTypesRe', a następnie wyświetla jej wartości w ComboBox.

\*reservationStatus(): zamieszcza wartości z tablicy 'statusRe' do Listy 'listStatusRe', a następnie wyświetla jej wartości w ComboBox.

\*reservationsSearch(): służy do wyszukiwania elementów w sposób przypisania wyszukiwanych wartości do FiltredList<Tables>, a następnie zmianie wartości TableView na te z listy

\*ordersProductName(): pobiera rekordy 'product\_name' z tabeli menu, gdzie status='Available' do Combobox, który służy następnie do wyboru produktu z menu do zamówienia.

\*ordersSpinner(): pozwala na zmianę wartości Spinner

\*ordersQuantity(): pobiera wartość z metody ordersSpinner, gdzie później używana jest na przykład przy podliczaniu ceny

\*ordersAdd(): wysyła do tabeli 'orders' wartości wprowadzone przez użytkownika, niektóre są wypełniane domyślnie przez metode. Metoda w trakcie sprawdza czy każde pole zostało wypełnione, aby do bazy nie wysłano żadnego null. Pobiera też wartości z tabeli 'menu', rekordy z której podliczana jest wartość 'orderPrice'.

\*showProductId(): służy do pobierania id z tabeli 'menu' i przypisuje ją do zmiennej, która jest później używana w metodzie ordersAdd().

\*orderTotal(): sumuje wartość rekordów 'price' z tabeli 'orders', a następnie przypisuje do zmiennej która jest używana w poszczególnych metodach

\*orderAmount(): używa zmiennej z metody orderTotal() do obliczenia 'balance', czyli reszty z wartości jaka została wprowadzona

\*orderPay(): sprawdza czy każde pole zostało odpowiednio wypełnione przez użytkownika, a następnie aktualizuje wybrane wartości w bazie danych w tabeli 'product' wraz z aktualną datą.

\*orderDisplayTotal(): wyświetla jako tekst całą wartość zmiennej 'totalP' odpowiadającej za całkowitą cenę

\*ordersList(): zapisuje wszystkie rekordy z tabeli 'orders' do ObservableList za pomocą get'ów z klasy 'OrdersUser'.

\*orderRemove(): sprawdza, czy każde pole zostało odpowiednio uzupełnione, a następnie wysyła zapytanie o usunięciu rekordów z tabeli 'orders'.

\*orderSelectData(): metoda umożliwia zaznaczenie elementu z TableView przypisując do zmiennej i wyświetlenie jej danych w TextField i ComboBox.

\*orderDisplayData(): wyświetla ObservableList z metody ordersList() w wybranym TableView.

\*ordersCustomerID(): przypisuje wartość 'customer\_id' z tabeli 'orders' do zmiennej, która służy do zapamiętania indeksu zamówień

\*customerUserDisplayData(): wyświetla ObservableList z metody customersUserList() w wybranym TableView.

\*customersUserList(): zapisuje wszystkie rekordy z tabeli 'orders' z aktualną datą do ObservableList za pomocą get'ów z klasy 'OrdersUser'.

\*statusUpdate(): sprawdza czy każde pole zostało odpowiednio wypełnione przez użytkownika, a następnie aktualizuje wybrane wartości w bazie danych w tabeli 'orders'. Aktualizuje ona tylko rekordy 'relase', jeśli rekord będzie wynosił 'Closed', usuwa się.

\*statusStatus(): zamieszcza wartości z tablicy 'statusRelease' do Listy 'listStatusRelace', a następnie wyświetla jej wartości w ComboBox.

\*switchForm(): metoda służąca do przełączania między oknami aplikacji w sposób włączania i wyłączania widoczności reszty. Metoda uruchamia przy przełączeniu okna niektóre metody.

\*logout(): metoda przełącza się do okna logowania 'hello-view.fxml' usatwiając parametry x i y, które służą do przeciągania okna

\*displayUsername(): zmienia text nazwy aktualnego użytkownika, pobierając dane z klasy 'Data'

\*close(): służy do zamykania okna

\*minimize(): służy do minimalizacji okna

\*initialize(URL url, ResourceBundle resourceBundle): aktywuje niektóre metody przy starcie aplikacji

### 3) Klasa UserController

Służy do posługiwania się metodami dla ekranu użytkownika. Posiada metody, które pozwalają na

- tworzenie rekordów
- usuwanie rekordów
- aktualizacje rekordów
- pobieranie rekordów
- przełączanie między ekranami

Metody:

\*menuShowList(): zapisuje wszystkie rekordy z tabeli 'menu' do ObservableList za pomocą get'ów z klasy 'Menus'.

\*menuShow(): wyświetla ObservableList z metody menuShowList() w wybranym TableView.

\*menuSearch(): służy do wyszukiwania elementów w sposób przypisania wyszukiwanych wartości do FiltredList<Menus>, a następnie zmianie wartości TableView na te z listy

\*reservationAdd(): wysyła do tabeli 'reservations' wartości wprowadzone przez użytkownika. Metoda w trakcie sprawdza czy każde pole zostało wypełnione, aby do bazy nie wysłano żadnego null. Sprawdza także czy dla danego 'user\_id' przypisana jest już rezerwacja przez zapytanie o tabelę 'tables' gdzie indeks jest równy wybranemu numerowi stolika. Następnie wysyła zapytanie o aktualizację tabeli 'tables', która zmienia 'status' oraz 'user\_id'

\*reservationSize(): wypisuje z tabeli 'tables' typ stolika, czyli jego rozmiar i wyświetla go w TextField

\*reservationClear(): sprawdza czy każde pole zostało uzupełnione po czym aktualizuje rezerwowany stół do wartości wcześniejszych przy pomocy zapytania dla tabeli 'tables' zmieniając 'status' i 'user\_id'. Następnie usuwa z tabeli 'reservations' 'user\_id' danego użytkownika.

\*reservationShowList(): zapisuje wszystkie rekordy z tabeli 'reservations' do ObservableList za pomocą get'ów z klasy 'Reservations'.

\*reservationShow(): wyświetla ObservableList z metody reservationShowList() w wybranym TableView.

\*reservationTableNumber(): pobiera rekordy 'id' z tabeli 'tables', tych stolików które są wolne i wyświetla w ComboBox.

\*ordersProductID(): przypisuje wartość 'product\_name' z tabeli 'menu' do zmiennej, która służy do zapamiętania nazwy produktu

\*ordersSpinner(): pozwala na zmianę wartości Spinner

\*ordersQuantity(): pobiera wartość z metody ordersSpinner, gdzie później używana jest na przykład przy podliczaniu ceny

\*ordersAdd(): wysyła do tabeli 'orders' wartości wprowadzone przez użytkownika. Metoda w trakcie sprawdza czy każde pole zostało wypełnione, aby do bazy nie wysłano żadnego null. Pobiera też wartości z tabeli 'menu', rekordy z której podliczana jest wartość 'orderPrice'.

\*showProductId(): służy do pobierania id z tabeli 'menu' i przypisuje ją do zmiennej, która jest później używana w metodzie ordersAdd().

\*orderTotal(): sumuje wartość rekordów 'price' z tabeli 'orders' wyszukiwanych przez 'user\_id' aktualnego użytkownika przy pomocy metody showProductId(), a następnie przypisuje do zmiennej która jest używana w poszczególnych metodach

\*orderPay(): sprawdza czy każde pole zostało odpowiednio wypełnione przez użytkownika, a następnie aktualizuje wybrane wartości w bazie danych w tabeli 'product' wraz z aktualną datą, gdzie indeks zamówienia jest taki sam do końca dokonania tej metody.

\*orderDisplayTotal(): wyświetla jako tekst całą wartość zmiennej 'totalP' odpowiadającej za całkowitą cenę

\*orderRemove(): sprawdza, czy każde pole zostało odpowiednio uzupełnione, a następnie wysyła zapytanie o usunięciu rekordów z tabeli 'orders'.

\*orderSelectData(): metoda umożliwia zaznaczenie elementu z TableView przypisując do zmiennej i wyświetlenie jej danych w TextField i ComboBox.

\*orderDisplayData(): wyświetla ObservableList z metody ordersList() w wybranym TableView.

\*ordersUserList(): zapisuje wszystkie rekordy z tabeli 'orders' do ObservableList za pomocą get'ów z klasy 'OrdersUser'.

\*methodUserId(): służy do pobierania id z tabeli 'user' i przypisuje ją do zmiennej, która jest później używana na przykład w metodzie ordersAdd().

\*statusDisplayData(): wyświetla ObservableList z metody statusUserList() w wybranym TableView.

\*statusUserList: zapisuje wszystkie rekordy z tabeli 'orders' z aktualną datą, gdzie 'user\_id' jest określony id aktualnego użytkownika za pomocą metody methodUserId() do ObservableList za pomocą get'ów z klasy 'OrdersUser'.

\*switchForm(): metoda służąca do przełączania między oknami aplikacji w sposób włączania i wyłączania widoczności reszty. Metoda uruchamia przy przełączeniu okna niektóre metody.

\*logout(): metoda przełącza się do okna logowania 'hello-view.fxml' usatwiając parametry x i y, które służą do przeciągania okna

\*displayUsername(): zmienia text nazwy aktualnego użytkownika, pobierając dane z klasy 'Data'

\*close(): służy do zamykania okna

\*minimize(): służy do minimalizacji okna

\*initialize(URL url, ResourceBundle resourceBundle): aktywuje niektóre metody przy starcie aplikacji

#### 4) Klasa Data

Klasa ta służy do inicjalizowania stanu obiektu do konstruktora klasy, które następnie są pobierane za pomocą metod get w poszczególnych metodach klasy DashboardController lub UserController. Zawarte zmienne:

- String username

#### 5) Klasa Menus

Klasa ta służy do inicjalizowania stanu obiektu do konstruktora klasy, które następnie są pobierane za pomocą metod get w poszczególnych metodach klasy DashboardController lub UserController. Zawarte zmienne:

- String productId

- String productName

- String type

- Double price

- String status

#### 6)Klasa OrdersUser

Klasa ta służy do inicjalizowania stanu obiektu do konstruktora klasy, które następnie są pobierane za pomocą metod get w poszczególnych metodach klasy DashboardController lub UserController. Zawarte zmienne:

- Integer id
- Integer user\_id
- Integer customer\_id
- Integer product\_id
- String product

## 7) Klasa Reservations

Klasa ta służy do inicjalizowania stanu obiektu do konstruktora klasy, które następnie są pobierane za pomocą metod get w poszczególnych metodach klasy DashboardController lub UserController. Zawarte zmienne:

- Integer id
- Integer userId
- Integer tableNumber
- String type
- String status
- Date date

## 8) Klasa Tables

Klasa ta służy do inicjalizowania stanu obiektu do konstruktora klasy, które następnie są pobierane za pomocą metod get w poszczególnych metodach klasy DashboardController lub UserController. Zawarte zmienne:

- Integer tableNumber
- String type
- String status
- Integer userId

## 9) Klasa User

Klasa ta służy do inicjalizowania stanu obiektu do konstruktora klasy, które następnie są pobierane za pomocą metod get w poszczególnych metodach klasy DashboardController lub UserController. Zawarte zmienne:

- Integer userId
- String email
- String username
- String password

## 10) Klasa HelloController

Jest to klasa która zawiera metody potrzebne do utworzenia konta wraz z walidacjami oraz metodami potrzebnymi do zalogowania w aplikacji

Metody:

\*checkEmail(): zawiera walidacje do sprawdzenia czy email został wpisany z odpowiednimi znakami

\*signup(): najpierw sprawdza czy pola zostały wypełnione prawidłowo, czyli czy hasło zawiera odpowiednią ilość znaków oraz czy nazwa została powtórzona (za pomocą zapytania do bazy danych liczy czy taka nazwa już się pojawiła, jeśli nie przepuszcza dalej), następnie wysyła dane podane przez użytkownika do tabeli 'user'

\*login(): sprawdza najpierw czy pola zostały odpowiednio wypełnione, następnie pobiera wszystkie rekordy z tabeli 'admin' i sprawdza czy wartości podane przez użytkownika odpowiadają tym w bazie danych. Następnie jeśli rekordy się nie zgadzają przechodzi do metody loginuser(). Na koniec po zatwierdzeniu przenosi do okna aplikacji 'adminpanel.fxml'

\*loginuser(): sprawdza najpierw czy pola zostały odpowiednio wypełnione, następnie pobiera wszystkie rekordy z tabeli 'user' i sprawdza czy wartości podane przez użytkownika odpowiadają tym w bazie danych. Na koniec po zatwierdzeniu przenosi do okna aplikacji 'userpanel.fxml'

\*userLoginID(): metoda przypisuje odpowiedni indeks z tabeli 'user' dla logującego się użytkownika



\*switchForm(): przełącza widoczność okna logowania oraz rejestracji naprzemiennie

\*close(): służy do zamykania okna

\*minimize(): służy do minimalizacji okna

#### 11) Klasa HelloAplication

Klasa odpowiadająca za uruchomienie aplikacji i wyświetlenie pierwszego okna logowania 'hello-view.fxml', ustawia także parametry x i y, opowiadające za przeciąganie oknem logowania

#### 12) Klasa Main

Służy do uruchomienia całej aplikacji, zawiera jedynie w metodzie main klasę 'HelloAplication'

### 8. Ograniczenia aplikacji

Aplikacja nie pozwala na rezerwacje stolików na określony dzień, przez co do tej symulacji należy jedynie brać, że rezerwacja jest na dzień aktualny. Po tym dniu administrator musiałby wyczyścić zarezerwowane stoliki.

Przy zamówieniach trzeba wyszukiwać nazwy produktów po nazwach, zamiast wybierać je na przykład z TableView.

Zamówienia ze strony administratora posiadają opcję wprowadzania ceny do zapłaty, nie może ona jednak być równa podanej sumie, musi być większa przynajmniej o 1.

TableView w customers lekko się nie mieści z prawej strony. ID może być mylące z User ID.

### 9. Zalety aplikacji

Do zalet można zaliczyć sporo rozbudowany system zamówień, który może być wykonywany z każdego konta, zaś administrator może zmieniać status zamówienia, które wyświetla się dla użytkowników.

Administrator posiada także dobry wgląd poprzez wykresy na pierwszym oknie aplikacji.

Można wyszukiwać płynnie dania oraz napoje w przestronnym menu, przez obojętny parametr.

Logowanie jak i wylogowywanie z konta jest przejrzyste i proste w obsłudze.