

МИНИСТЕРСТВО ОБРАЗОВАНИЯ НАУКИ И МОЛОДЕЖНОЙ ПОЛИТИКИ  
НИЖЕГОРОДСКОЙ ОБЛАСТИ  
Государственное бюджетное профессиональное образовательное учреждение  
НИЖЕГОРОДСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Специальность 09.02.07 Информационные системы и программирование

Квалификация Программист

КУРСОВАЯ РАБОТА

МДК 11.01 Технология разработки и защиты баз данных  
Тема: Разработка приложения для предметной области «Пиццерия»

Выполнил  
обучающийся группы  
2ИСиП-22-3с  
Потапова А.С.

Проверил преподаватель  
Дворянинова М.С.

Проект защищен с  
оценкой

\_\_\_\_\_  
Дата защиты \_\_\_\_\_  
Подпись \_\_\_\_\_

Нижний Новгород  
2023 г.

## Оглавление

Введение .....	3
1. Теоретические основы разрабатываемой темы.....	4
1.1 Анализ предметной области .....	4
1.2. Выбор комплекса задач автоматизации и характеристика существующих бизнес-процессов.....	5
2. Разработка технического задания .....	6
2.1 Назначение и цели создания системы .....	6
2.1.1 Назначение системы .....	6
2.1.2 Цели создания системы. ....	6
2.2 Требования к функциям, выполняемым системой .....	6
2.3 Требования к видам обеспечения .....	6
2.3.1 Требования к программному обеспечению .....	6
2.3.2 Доступные программные средства: .....	6
3. Проектирование .....	7
3.1. Информационно-логическая модель базы данных .....	7
3.1.1 Словарь данных.....	8
3.1.2 Описание структуры базы данных. ....	8
4. Разработка.....	10
4.1. Создание и заполнение таблиц .....	10
4.1.1 Создание таблиц из диаграммы.....	10
4.1.2 Заполнение таблицы созданной базы данных. ....	16
4.2. Запросы, хранимые процедуры и триггеры. ....	20
4.3 Разработка приложения .....	22
4.3.1 Разработка графического интерфейса .....	22
4.3.2 Разработка логической части приложения .....	29
6. Руководство пользователя. ....	46
Заключение.....	49
Список литературы. ....	50

## **Введение**

Современные технологии играют ключевую роль в совершенствовании бизнес-процессов и обеспечении удовлетворения потребностей клиентов. Один из таких сегментов, где информационные системы приносят значительные улучшения – сфера общественного питания. Рестораны и кафе постоянно стремятся оптимизировать свои процессы, предоставляя клиентам удобные способы заказа и доставки блюд.

Целью данной курсовой работы является создание базы данных и визуального интерфейса для пиццерии, позволяющего клиентам авторизоваться, просматривать меню, оформлять заказы на доставку или самовывоз, а также отслеживать статус своего заказа. Это позволит пиццерии значительно улучшить качество обслуживания клиентов, сократить время ожидания заказов и повысить общую удовлетворенность клиентов.

В соответствие с поставленной целью необходимо решить следующие задачи:

- анализ предметной области;
- использование методов проектирование баз данных;
- использование методов разработки баз данных;
- интеграция функциональных компонентов;
- сделать выводы, спроектировать базу данных и графический(визуальный) для пиццерии.

Создание базы данных и визуального интерфейса для пиццерии станет важным шагом в повышении эффективности бизнеса и совершенствовании обслуживания клиентов.

## **1. Теоретические основы разрабатываемой темы.**

### **1.1 Анализ предметной области**

Пиццерия — это специализированное заведение общественного питания, которое специализируется на приготовлении и продаже пиццы. Анализ предметной области пиццерии включает в себя несколько ключевых аспектов:

1. Меню и ассортимент продукции:
  - Пиццерии предлагают разнообразные виды пиццы, включая классические, оригинальные и специальные варианты.
  - Также могут предлагаться закуски, салаты, напитки и десерты.
2. Ценообразование:
  - Пиццерии имеют разнообразные ценовые категории, начиная с бюджетных заведений и заканчивая ресторанами с более высокими ценами.
  - Цены могут зависеть от размера и состава пиццы.
3. Местоположение:
  - Выбор местоположения пиццерии играет важную роль. Она может располагаться в торговом центре, на улице, вблизи офисных комплексов или в других местах с высокой проходимостью.
4. Обслуживание и формат заведения:
  - Пиццерии могут быть ресторанными, фаст-фудами или иметь различные форматы обслуживания.
  - Наличие доставки и самовывоза также важные аспекты.
5. Конкурентная среда:
  - Конкуренция среди пиццерий может быть высокой, и владельцам необходимо учитывать конкурентные преимущества и стратегии.
6. Ингредиенты и качество продукции:
  - Качество ингредиентов и приготовления пиццы играют ключевую роль в привлечении и удержании клиентов.
7. Маркетинг и реклама:
  - Рекламные кампании, программа лояльности клиентов и акции могут помочь привлечь больше клиентов.
8. Лицензии и регулирование:
  - Пиццерии должны соответствовать местным законам и нормативам, а также обеспечивать безопасность пищи.
9. Управление персоналом:
  - Эффективное управление персоналом, обучение и соблюдение стандартов обслуживания играют важную роль в успехе пиццерии.
10. Технологические аспекты:
  - Многие пиццерии используют POS-системы, онлайн-заказы и мобильные приложения для облегчения работы и увеличения удобства для клиентов.
11. Тенденции в индустрии:

- Индустрия пиццерий постоянно развивается. Тенденции могут включать в себя новые виды пиццы, использование органических продуктов, развитие онлайн-заказов и другие инновации.

## **1.2. Выбор комплекса задач автоматизации и характеристика существующих бизнес-процессов.**

1. Разработка приложения для заказа пиццы: создание интерфейса для выбора пиццы, добавления в корзину и оформления заказа.
2. Интеграция с базой данных пиццерии: наличие меню с ценами, актуальное расписание работы и информация о специальных предложениях.

Бизнес-процессы:

- Заказ пиццы: вручную через звонок или личное посещение заведения.
- Доставка или самовывоз: клиент самостоятельно оформляет доставку или забирает заказ из пиццерии.

## **2. Разработка технического задания**

### **2.1 Назначение и цели создания системы**

#### **2.1.1 Назначение системы**

Приложение для персонального компьютера создается с целью обеспечения:

1. Удобства при выборе заказа и его совершении;
2. Получения информации о клиентах и оформленных заказах;
3. Приложение разработано для покупателей, при помощи которого они смогут собрать заказ и сэкономить своё время на любые другие свои дела.

#### **2.1.2 Цели создания системы.**

Разработка информационной преследует следующие цели:

1. Отслеживание заказов;
2. Автоматизация процессов заказов со стороны клиента;
3. Хранение информации о клиентах и их заказах.

### **2.2 Требования к функциям, выполняемым системой**

К функциям предъявляются следующие требования:

1. Хранение и добавление информации о клиентах;
2. Внесение изменений в базу данных.

### **2.3 Требования к видам обеспечения**

#### **2.3.1 Требования к программному обеспечению**

Доступные аппаратные средства:

– персональный компьютер с доступом в интернет.

#### **2.3.2 Доступные программные средства:**

– операционная система;

### 3. Проектирование

#### 3.1. Информационно-логическая модель базы данных

ERD (Entity-Relationship Diagram) представляет собой графическую модель, которая описывает сущности и связи между ними в базе данных. На изображении ERD изначально должны находиться блоки, представляющие сущности, связанные линиями, обозначающими типы отношений между этими сущностями.

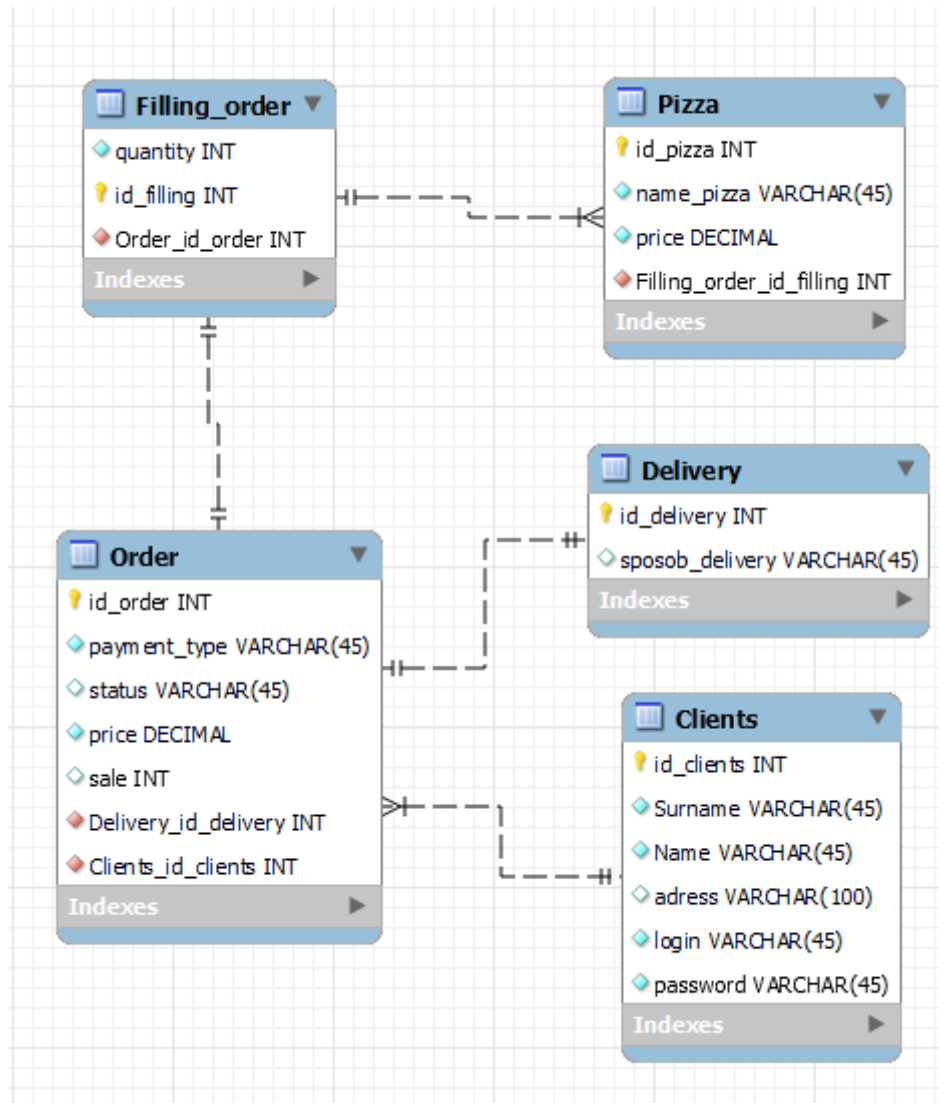


Рис 1. – ERD-модель.

На данном изображении представлены следующие сущности:

1. Filling\_order: Информация о наполнении заказа
2. Order: Информация о заказах
3. Pizza: Информация о пиццах
4. Delivery: Информация о доставке
5. Clients: Информация о клиентах

Также видно, что сущности связаны между собой через внешние ключи, такие как filling\_order\_id\_filling и delivery\_id\_delivery. Это указывает на существование связей между различными таблицами в базе данных.

Для каждой сущности указаны её атрибуты, например, для сущности Pizza это id\_pizza, name\_pizza и price. Эти атрибуты представляют информацию, хранящуюся в базе данных для каждой сущности.

Таким образом, данный ERD представляет собой модель базы данных, отражающую структуру информации о заказах пиццы, клиентах и доставке.

### 3.1.1 Словарь данных.

Для разработки базы данных мы создаём ERD-модель, в которой существует 5 сущностей:

1. Клиенты (clients):
  - a. Код клиента (id\_clients),
  - b. Фамилия (Surname),
  - c. Имя (Name),
  - d. Логин (login),
  - e. Пароль (password);
2. Пицца (Pizza):
  - a. Код пиццы (id\_pizza),
  - b. Название (name),
  - c. Цена (price);
3. Доставка (Delivery):
  - a. Код доставки (id\_delivery),
  - b. Способ доставки (sposob\_delivery);
4. Заказ (Order):
  - a. Код заказа (id\_order),
  - b. Тип оплаты (payment\_type),
  - c. Статус заказа (satus),
  - d. Цена (price),
  - e. Скидка (sale);
5. Наполнение заказа (Filling\_order):
  - a. Код наполнения (id\_filling),
  - b. Количество (quantity).

### 3.1.2 Описание структуры базы данных.

Таблица Filling\_order

Key	Field Name	Data Type	Required?	Примечания
PK	Id_filling	INT	Y	
	quantity	INT	Y	
FK	Order_id_order	INT	Y	Ссылка на таблицу Order



Таблица Pizza

Key	Field Name	Data Type	Required?	Примечания
PK	Id_pizza	INT	Y	
	Name_pizza	VARCHAR(45)	Y	
	price	DECIMAL	Y	
FK	Filling_order_id_filling	INT	Y	Ссылка на таблицу Filling_order

Таблица Clients

Key	Field Name	Data Type	Required?	Примечания
PK	Id_clients	INT	Y	
	Surname	VARCHAR(45)	Y	
	Name	VARCHAR(45)	Y	
	login	VARCHAR(45)	Y	
	password	VARCHAR(45)	Y	

Таблица Order

Key	Field Name	Data Type	Required?	Примечания
PK	Id_order	INT	Y	
	Payment_type	VARCHAR(45)	Y	
	status	VARCHAR(45)	Y	
	price	DECIMAL	Y	
	sale	INT	Y	
FK	Delivery_id_delivery	INT	Y	Ссылка на таблицу Delivery
FK	Clients_id_clients	INT	Y	Ссылка на таблицу Clients

Таблица Delivery

Key	Field Name	Data Type	Required?	Примечания
PK	Id_delivery	INT	Y	
	Sposob_delivery	VARCHAR(45)	Y	

## 4. Разработка

Существует достаточно много с систем управления базами данных такие как PostgreSQL, Oracle Database, Firebird, Interbase, IBM DB2, Informix, MS SQL Server и т.п.

База данных будет разрабатывается с помощью системы управления базами данных MySQL, а также инструмента для визуального проектирования баз данных MySQL Workbench, так как это очень удобный инструменты для проектирования и редактирования баз данных и они подходят для малых и средних приложений.

### 4.1. Создание и заполнение таблиц

Создание таблиц будет проходить через экспорт из ER-Diagramm, а заполнение будет осуществляться с помощью запросов.

#### 4.1.1 Создание таблиц из диаграммы.

Открываем нашу диаграмму и переходим на вкладку Database.

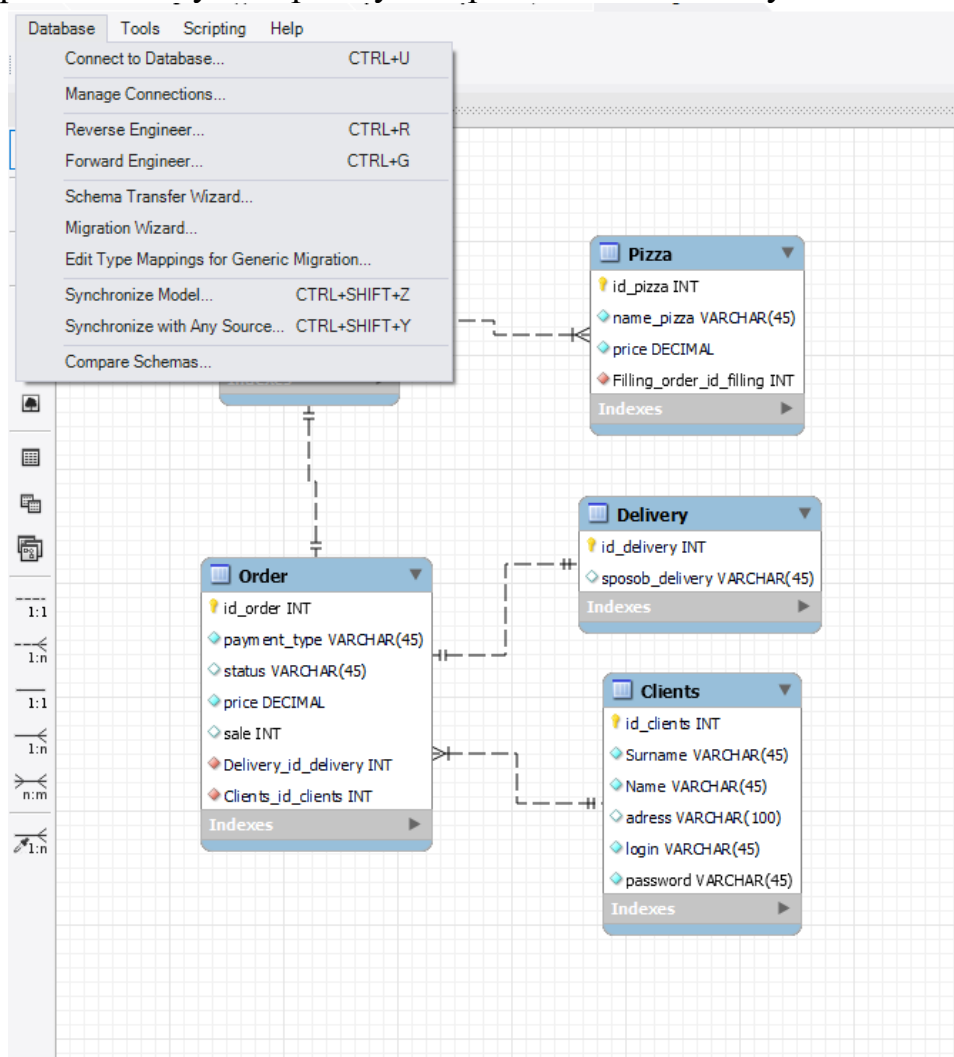


Рис. 2.

Далее выбираем пункт Forward Engineer...

Forward Engineer to Database

**Connection Options**

Options

Select Objects

Review SQL Script

Commit Progress

**Set Parameters for Connecting to a DBMS**

Stored Connection: PIZZA Select from saved connection settings

Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname: 127.0.0.1 Port: 3306 Name or IP address of the server host - and TCP/IP port.

Username: root Name of the user to connect with.

Password: Store in Vault ... Clear The user's password. Will be requested later if it's not set.

Default Schema: The schema to use as default schema. Leave blank to select it later.

Back Next Cancel

Рис. 3 – Connection Options.

У нас появляется данное окно (ис.3). Проверяем что все данные сходятся и нажимаем кнопку Next.

Connection Options

**Options**

Select Objects

Review SQL Script

Commit Progress

**Set Options for Database to be Created**

**Tables**

☐ Skip creation of FOREIGN KEYS

☐ Skip creation of FK Indexes as well

☐ Generate separate CREATE INDEX statements

☐ Generate INSERT statements for tables

☐ Disable FK checks for INSERTs

**Other Objects**

☐ Don't create view placeholder tables

☐ Do not create users. Only create privileges (GRANTS)

**Code Generation**

☐ DROP objects before each CREATE object

☐ Generate DROP SCHEMA

☐ Omit schema qualifier in object names

☐ Generate USE statements

☐ Add SHOW WARNINGS after every DDL statement

☒ Include model attached scripts

Back Next Cancel

Рис. 4 – Options.

На данной вкладке Options ничего не меняем и переходим по кнопке дальше.

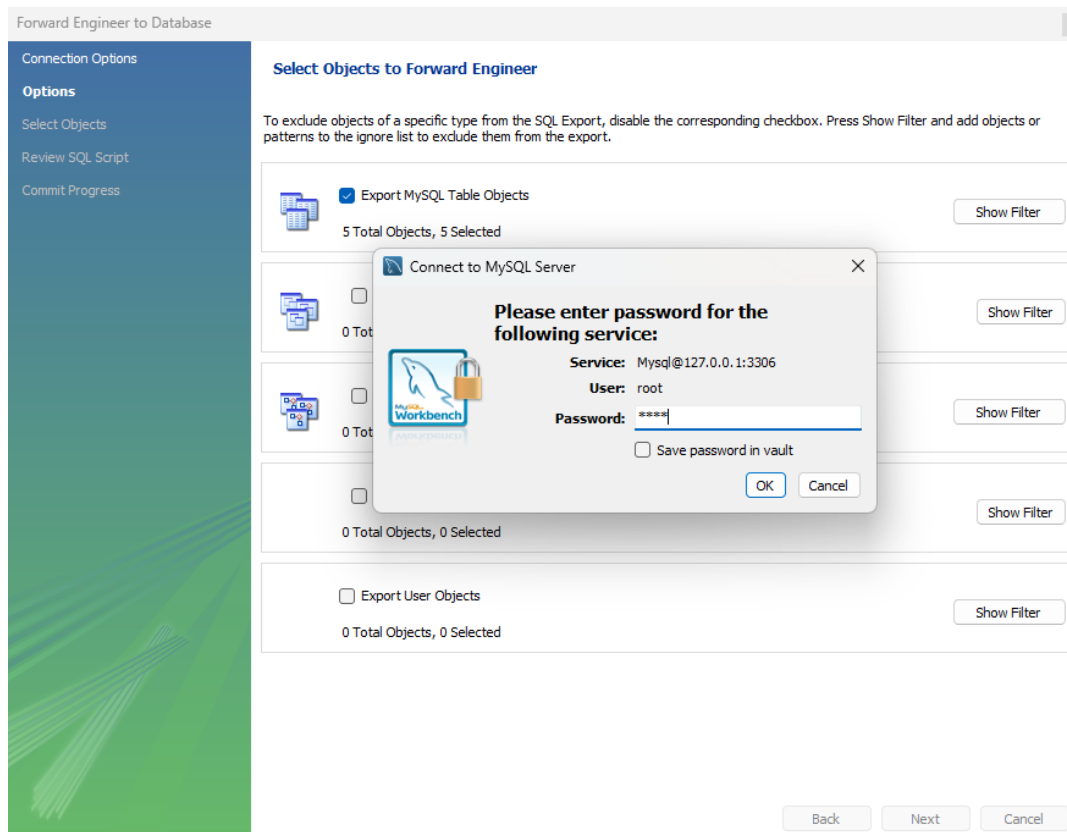


Рис. 5 – password.

Переходя далее, видим, что помимо основного окна у нас появилось окно с заполнением пароля. Данное окно появляется тогда, когда приложение требует разрешения на экспорт таблицы в сервер, который мы создали для нашей базы данных. После введения пароля можем увидеть, что у нас есть выбор одного из нескольких параметров (рис.6).

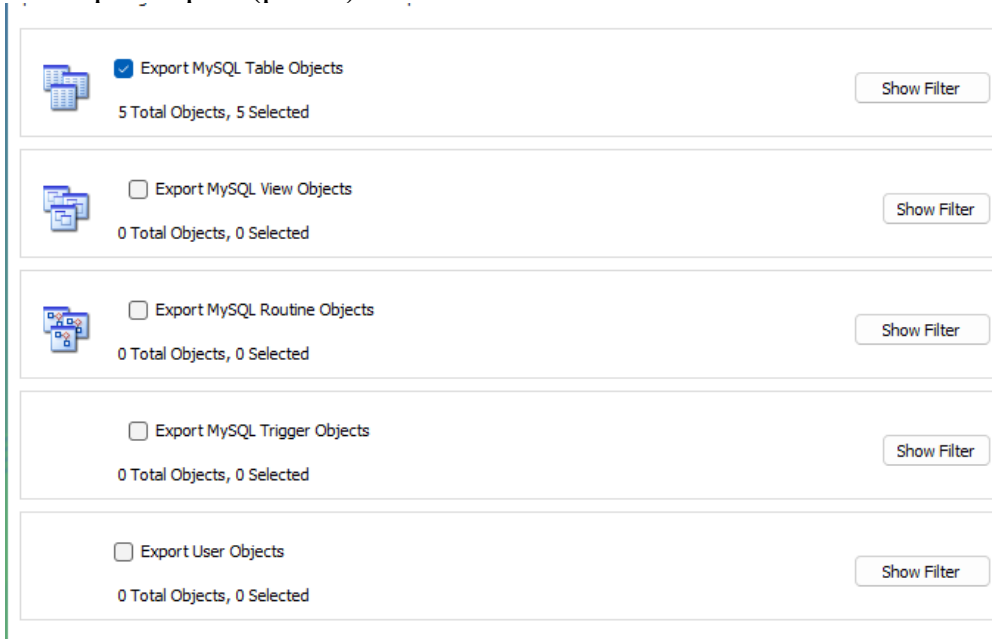


Рис. 6 – Export Table.

Из всех имеющихся параметров нам нужно выбрать только тот, который отвечает за экспорт таблиц. Выбираем его и переходим далее.

У нас открывается вкладка Review SQL Script, на которой мы можем увидеть листинг экспорта, вот как он выглядит:

```
-- MySQL Workbench Forward Engineering

SET                @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0;
SET                @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET                @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-----
-- Schema mydb
-----
-- Schema mydb
-----
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-----
-- Table `mydb`.`Clients`
-----
CREATE TABLE IF NOT EXISTS `mydb`.`Clients` (
  `id_clients` INT NOT NULL AUTO_INCREMENT,
  `Surname` VARCHAR(45) NOT NULL,
  `Name` VARCHAR(45) NOT NULL,
  `adress` VARCHAR(100) NULL,
  `login` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_clients`))
ENGINE = InnoDB;

-----
-- Table `mydb`.`Delivery`
-----
CREATE TABLE IF NOT EXISTS `mydb`.`Delivery` (
  `id_delivery` INT NOT NULL,
  `sposob_delivery` VARCHAR(45) NULL,
```

```

PRIMARY KEY (`id_delivery`))
ENGINE = InnoDB;

-----
-- Table `mydb`.`Order`
-----

CREATE TABLE IF NOT EXISTS `mydb`.`Order` (
  `id_order` INT NOT NULL,
  `payment_type` VARCHAR(45) NOT NULL,
  `status` VARCHAR(45) NULL,
  `price` DECIMAL NOT NULL,
  `sale` INT NULL,
  `Delivery_id_delivery` INT NOT NULL,
  `Clients_id_clients` INT NOT NULL,
  PRIMARY KEY (`id_order`),
  INDEX `fk_Order_Delivery1_idx` (`Delivery_id_delivery` ASC) VISIBLE,
  INDEX `fk_Order_Clients1_idx` (`Clients_id_clients` ASC) VISIBLE,
  CONSTRAINT `fk_Order_Delivery1`
    FOREIGN KEY (`Delivery_id_delivery`)
      REFERENCES `mydb`.`Delivery` (`id_delivery`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_Order_Clients1`
    FOREIGN KEY (`Clients_id_clients`)
      REFERENCES `mydb`.`Clients` (`id_clients`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `mydb`.`Filling_order`
-----

CREATE TABLE IF NOT EXISTS `mydb`.`Filling_order` (
  `quantity` INT NOT NULL,
  `id_filling` INT NOT NULL,
  `Order_id_order` INT NOT NULL,
  PRIMARY KEY (`id_filling`),
  INDEX `fk_Filling_order_Order1_idx` (`Order_id_order` ASC) VISIBLE,
  CONSTRAINT `fk_Filling_order_Order1`
    FOREIGN KEY (`Order_id_order`)
      REFERENCES `mydb`.`Order` (`id_order`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```
-- Table `mydb`.`Pizza`
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Pizza` (  
  `id_pizza` INT NOT NULL,  
  `name_pizza` VARCHAR(45) NOT NULL,  
  `price` DECIMAL NOT NULL,  
  `Filling_order_id_filling` INT NOT NULL,  
  PRIMARY KEY (`id_pizza`),  
  INDEX `fk_Pizza_Pizza_has_Order1_idx` (`Filling_order_id_filling` ASC)  
  VISIBLE,  
  CONSTRAINT `fk_Pizza_Pizza_has_Order1`  
    FOREIGN KEY (`Filling_order_id_filling`)  
    REFERENCES `mydb`.`Filling_order` (`id_filling`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;  
  
SET SQL_MODE=@OLD_SQL_MODE;  
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;  
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Листинг 1.

После проверки того, что у нас импортируются все таблицы мы нажимаем далее и переходим на следующую страницу. На новой странице Commit Progress, которая снова запрашивает у нас пароль для импорта (рис.7).

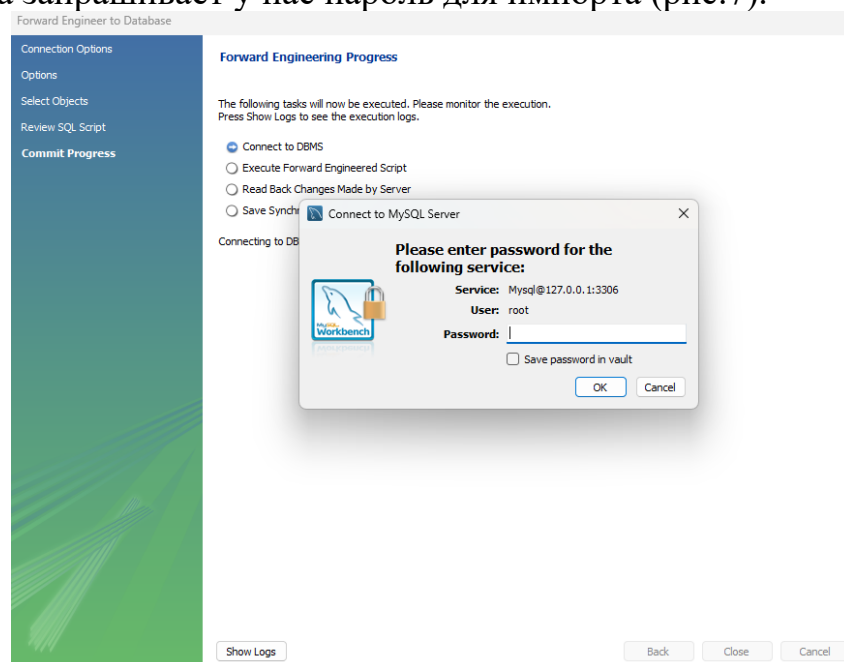


Рис. 7 – commit progress.

После ввода пароля у нас идет проверка всех атрибутов и по успешному окончании (рис.8) мы можем нажать закрыть, так как наша База данных успешно импортировалась.



Рис. 8. – Проверка.

#### 4.1.2 Заполнение таблицы созданной базы данных.

Заполнение таблицы Clients (листинг 2).

```
insert into mydb.clients (id_clients, Surname, Name, login, password)
values (1, 'Erofeev', 'Alexander', 'sashasuper228', '123456789F'),
(2, 'Sorokin', 'Andrew', 'andrushenka-sorokin-03', '111111'),
(3, 'Moiseev', 'Alexey', 'alex09', '56789GTN'),
(4, 'Vlasova', 'Kseniya', 'kliploker777', 'ledybug'),
(5, 'Stepanova', 'Nadezhda', 'greetings18', 'hello19'),
(6, 'Golodyaev', 'Boris', 'borya14', 'penis'),
(7, 'Potapova', 'Anastasia', 'anpotapova960', '987654321'),
(8, 'Levin', 'Dmitriy', 'dimansentvega', 'vega');
```

Листинг 2.

Проверяем заполненность наших данных с помощью запроса (листинг 3).

```
SELECT * FROM mydb.clients;
```

Листинг 3.

Вот как заполнилась наша таблица (рис.9):

Исходя из увиденного можно сделать вывод, что наша таблица заполняется корректно и можно заполнять следующие таблицы по этому же принципу.



Result Grid					
Filter Rows:					
	id_clients	Surname	Name	login	password
▶	1	Erofeev	Alexander	sashasuper228	123456789F
	2	Sorokin	Andrew	andrushenka-sorokin...	111111
	3	Moiseev	Alexey	alex09	56789GTN
	4	Vlasova	Kseniya	kliploker777	ledybug
	5	Stepanova	Nadezhda	greetings18	hello19
	6	Golodyaev	Boris	borya14	penis
	7	Potapova	Anastasia	anpotapova960	987654321
	8	Levin	Dmitriy	dimansentvega	vega
*	NULL	NULL	NULL	NULL	NULL

Рис. 9 – таблица клиенты.

Заполнение таблицы Delivery (листинг 4, листинг 5, рисунок 10):

```
insert into mydb.delivery (id_delivery, sposob_delivery)
values (1, 'pickup'),
(2, 'delivery'),
(3, 'delivery'),
(4, 'pickup'),
(5, 'pickup'),
(6, 'pickup'),
(7, 'delivery'),
(8, 'delivery'),
(9, 'pickup'),
(10, 'pickup'),
(11, 'pickup'),
(12, 'pickup');
```

Листинг 4.

```
SELECT * FROM mydb.delivery;
```

Листинг 5.

	id_delivery	sposob_delivery
▶	1	pickup
	2	delivery
	3	delivery
	4	pickup
	5	pickup
	6	pickup
	7	delivery
	8	delivery
	9	pickup
	10	pickup
	11	pickup
	12	pickup
*	NULL	NULL

Рис. 10 – таблица доставка.

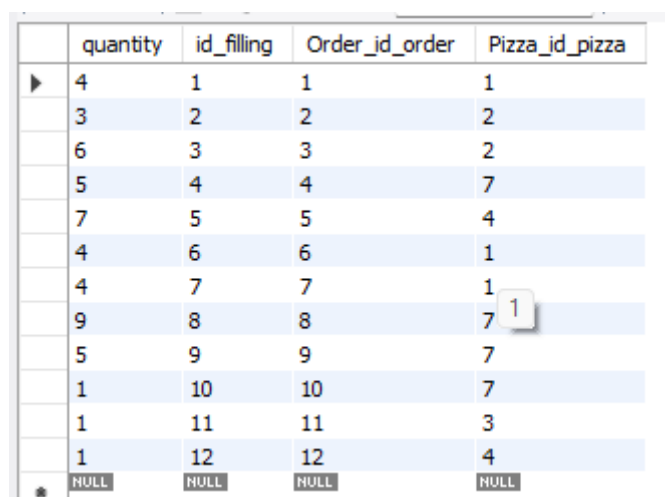
Заполнение таблицы Filling\_order (листинг 6, листинг 7, рисунок 11):

```
insert into mydb.filling_order (quantity, id_filling, Order_id_order, Pizza_id_pizza)
values (4, 1, 1, 1),
(3, 2, 2, 2),
(6, 3, 3, 2),
(5, 4, 4, 7),
(7, 5, 5, 4),
(4, 6, 6, 1),
(4, 7, 7, 1),
(9, 8, 8, 7),
(5, 9, 9, 7),
(1, 10, 10, 7),
(1, 11, 11, 3),
(1, 12, 12, 4);
```

Листинг 6.

```
SELECT * FROM mydb.filling_order;
```

Листинг 7.



	quantity	id_filling	Order_id_order	Pizza_id_pizza
▶	4	1	1	1
	3	2	2	2
	6	3	3	2
	5	4	4	7
	7	5	5	4
	4	6	6	1
	4	7	7	1
	9	8	8	7
	5	9	9	7
	1	10	10	7
	1	11	11	3
	1	12	12	4
*	NULL	NULL	NULL	NULL

Рис. 11 – таблица наполнения заказа.

Заполнение таблицы Order (листинг 8, листинг 9, рисунок 12):

```
insert into mydb.order (id_order, payment_type, status, price, sale,
Delivery_id_delivery, Clients_id_clients)
values (1, 'money', 'ready', 20.0, 0, 1, 7),
(2, 'money', 'ready', 18.0, 0, 2, 8),
(3, 'money', 'in process', 35.6, 0, 3, 6),
(4, 'card', 'in process', 35.0, 0, 4, 5),
(5, 'card', 'ready', 63.0, 0, 5, 4),
(6, 'money', 'in process', 20.4, 0, 6, 6),
(7, 'card', 'in process', 20.4, 0, 7, 3),
(8, 'card', 'ready', 63.0, 0, 8, 1),
(9, 'card', 'ready', 35.0, 0, 9, 1),
```

```
(10, 'money', 'ready', 7.0, 0, 10, 5),
(11, 'money', 'ready', 7.5, 0, 11, 4),
(12, 'card', 'in process', 9.0, 0, 12, 1);
```

Листинг 8.

```
SELECT * FROM mydb.order;
```

Листинг 9.

	id_order	payment_type	status	price	sale	Delivery_id_delivery	Clients_id_clients
▶	1	money	ready	20	0	1	7
	2	money	ready	18	0	2	8
	3	money	in process	36	0	3	6
	4	card	in process	35	0	4	5
	5	card	ready	63	0	5	4
	6	money	in process	20	0	6	6
	7	card	in process	20	0	7	3
	8	card	ready	63	0	8	1
	9	card	ready	35	0	9	1
	10	money	ready	7	0	10	5
	11	money	ready	8	0	11	4
	12	card	in process	9	0	12	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 12 – таблица заказа.

Заполнение таблицы Pizza (листинг 10, листинг 11, рисунок 13):

```
insert into mydb.pizza (id_pizza, name_pizza, price)
values (1, 'pepperoni', 5),
(2, 'mushroom', 6),
(3, 'margarita', 7.9),
(4, 'cheese chicken', 9),
(5, 'chicken', 7.5),
(6, 'four cheese', 6.8),
(7, 'ham and cheese', 7),
(8, 'venice', 8.5),
(9, 'creamy', 8.9);
```

Листинг 10.

```
SELECT * FROM mydb.pizza;
```

Листинг 11.

	id_pizza	name_pizza	price
▶	1	pepperoni	5
	2	mushroom	6
	3	margarita	8
	4	cheese chicken	9
	5	chicken	8
	6	four cheese	7
	7	ham and cheese	7
	8	venice	9
	9	creamy	9
*	NULL	NULL	NULL

Рис. 13 – таблица пицца.

#### 4.2. Запросы, хранимые процедуры и триггеры.

Теперь все наши таблицы заполнены, и мы можем работать с нашей базой данных дальше.

Теперь создадим ещё несколько запросов на выборку, одну хранимую процедуру и два триггера.

Первый запрос выводит список названия пицц с ценой в заказе свыше 20:

```
select pizza.id_pizza, pizza.name, pizza.price, Max (pizza.price) AS pr
FROM pizza INNER JOIN filling_order ON filling_order.id_pizza = price.id_pizza
where ((filling_order.price)>20)
group by filling_order.id_pizza, filling_price;
```

Листинг 12 – первый запрос.

Второй запрос выводит группировку пицц с одинаковой ценой.

```
SELECT id_pizza, price_pizza
FROM pizza
Where price_pizza in(5)
Order by id_pizza;
```

Листинг 13 – второй запрос.

Запрос на вывод клиентов, которые выбрали пиццу пипперони.

```
SELECT pizza.id_pizza, pizza.name
Where pizza name = 'pipperoni';
```

Листинг 14 – третий запрос.

Запрос на вывод имени клиента и его заказа

```
SELECT clients.name, order.id_order
FROM clients;
```

Листинг 15 – четвертый запрос;

Хранимая процедура выводит цену пиццы, название пиццы и количество с помощью ORDER BY и JOIN.

```
CREATE PROCEDURE GetPizzaName
AS
BEGIN
    SELECT
        pizza.name,
```

```

    pizza.price,
    filling_order.quantity
FROM
    pizza
JOIN
    filling_order ON pizza.id_pizza = filling_order.id_pizza
ORDER BY
    pizza.name,
    pizza.price;
END

```

Листинг 16 – Хранимая процедура.

```
EXEC GetPizzaName
```

Листинг 17 – вызов хранимой процедуры.

Напишем 2 триггера:

Первый триггер написан для обновления цены пиццы.

```

CREATE TRIGGER UpdatePricePizza
ON pizza
AFTER UPDATE
AS
BEGIN
    IF UPDATE(price)
    BEGIN
        UPDATE pizza
        SET price = CASE
            WHEN price < 0 THEN 0
            WHEN price > 100 THEN 100
            ELSE price
        END
        WHERE price IN (SELECT id_pizza FROM inserted)
    END
END

```

Листинг 18 – первый триггер.

Второй триггер написан для вывода имени и фамилии с большого регистра.

```

CREATE TRIGGER FormatClientNames
ON client
FOR INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE s

```

```
SET c.surname = UPPER(LEFT(c.surname, 1)) +  
LOWER(SUBSTRING(c.surname, 2, LEN(c.surname))),  
    c.name = UPPER(LEFT(c.name, 1)) + LOWER(SUBSTRING(c.name, 2,  
LEN(c.name)))  
FROM inserted i  
INNER JOIN clients s ON i.id_clients = c.id_client;  
EN
```

Листинг 19 – второй триггер.

## 4.3 Разработка приложения

### 4.3.1 Разработка графического интерфейса

Для того, чтобы начать разработку приложения нам нужно проанализировать какие окна у нас будут задействованы и активны. Так как мы разрабатываем приложение для пиццерии нам необходимо понимать что для приложения необходимы такие окна как:

1. Окно авторизации,
2. Окно регистрации,
3. Главное окно, которое содержит в себе меню с картинками,
4. Окно информации,
5. Окно корзины,
6. Окно профиля;

Теперь, когда мы определились с активными окнами, можем начать заниматься разработкой каждого окна с помощью SceneBuilder'a.

Первым создадим окно авторизации, оно будет начальным при запуске приложения.

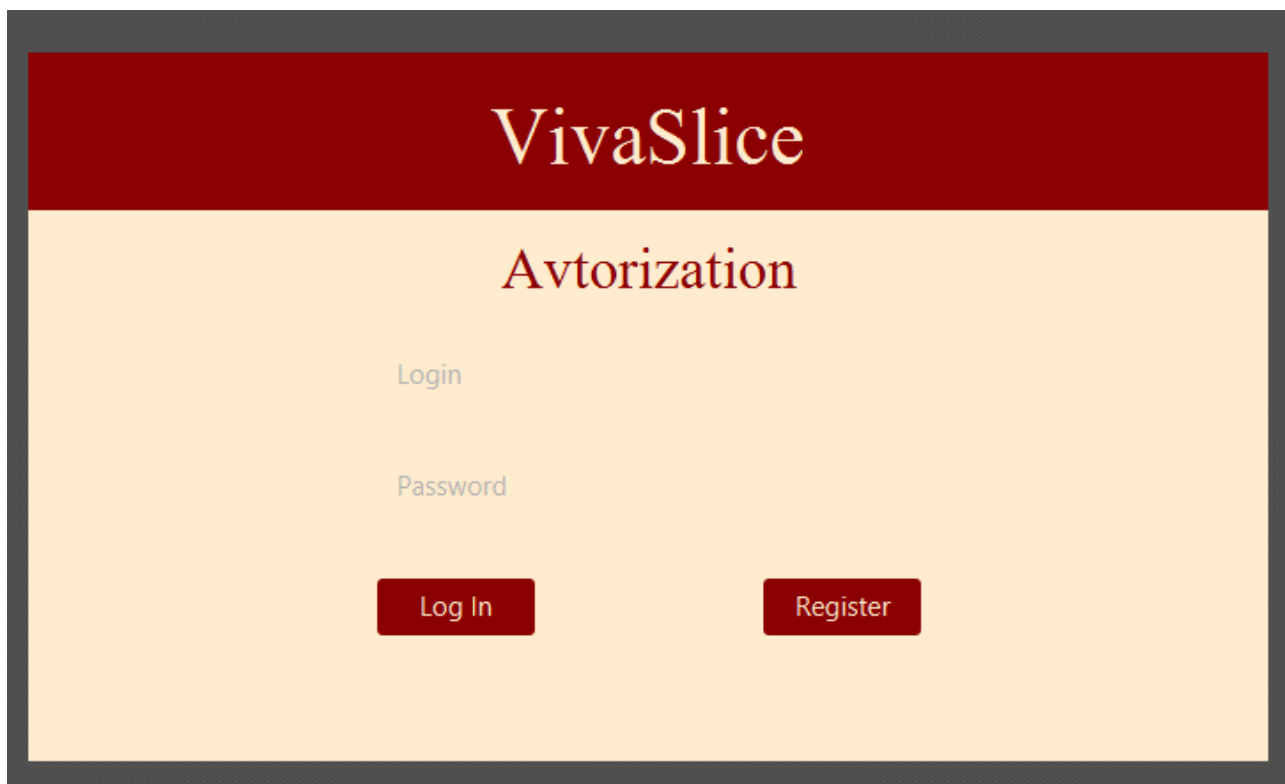


Рис. 14 – окно авторизации.

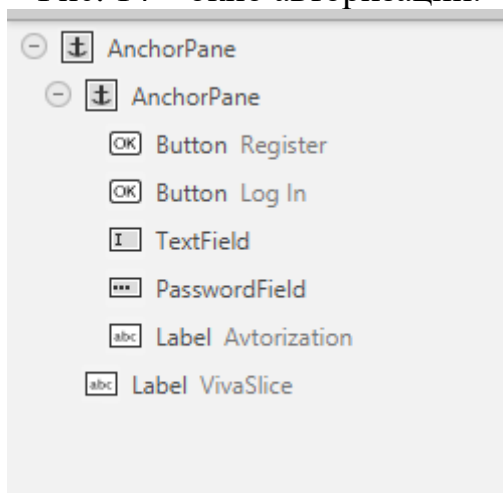


Рис. 15 – дерево.

Для создания данного окна нам потребовалось использовать такие элементы, как 2 кнопки для регистрации и входа, кнопка регистрации переносит нас на второе окно, где новый пользователь сможет создать аккаунт для удобного использования созданного приложения, а кнопка авторизации при правильно введенных данных пересылает на окно с ассортиментом.

Также для ввода данных мы использовали элементы textfield и passwordfield (Рисунок 14) для ввода логина и пароля соответственно.

Далее рассмотрим окно регистрации(Рисунок 15)



Рис. 16 – окно регистрации.

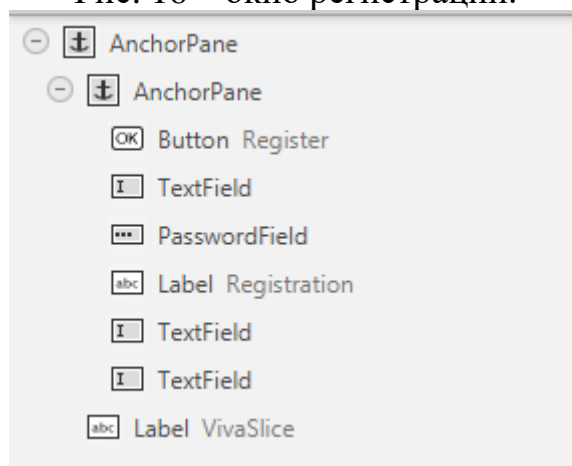


Рис. 17 – дерево.

При создании этого окна были использованы такие элементы:

1. Кнопка для сохранения новых данных пользователя и регистрации, при нажатии на нее также происходит действие перехода обратно на страницу с авторизацией, чтобы можно было войти в приложение (Рисунок 16).
2. Для того чтобы заполнить данные о себе, пользователю предоставляются поля для ввода текста и пароля (Рисунок 17).

Следующим нашим шагом идет создание окна с ассортиментом. Посмотрим что оно содержит.





Рис. 19 – окно с ассортиментом.

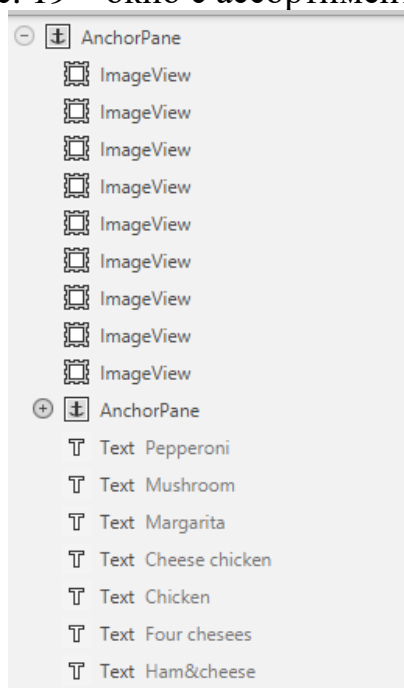


Рис. 20 – дерево.

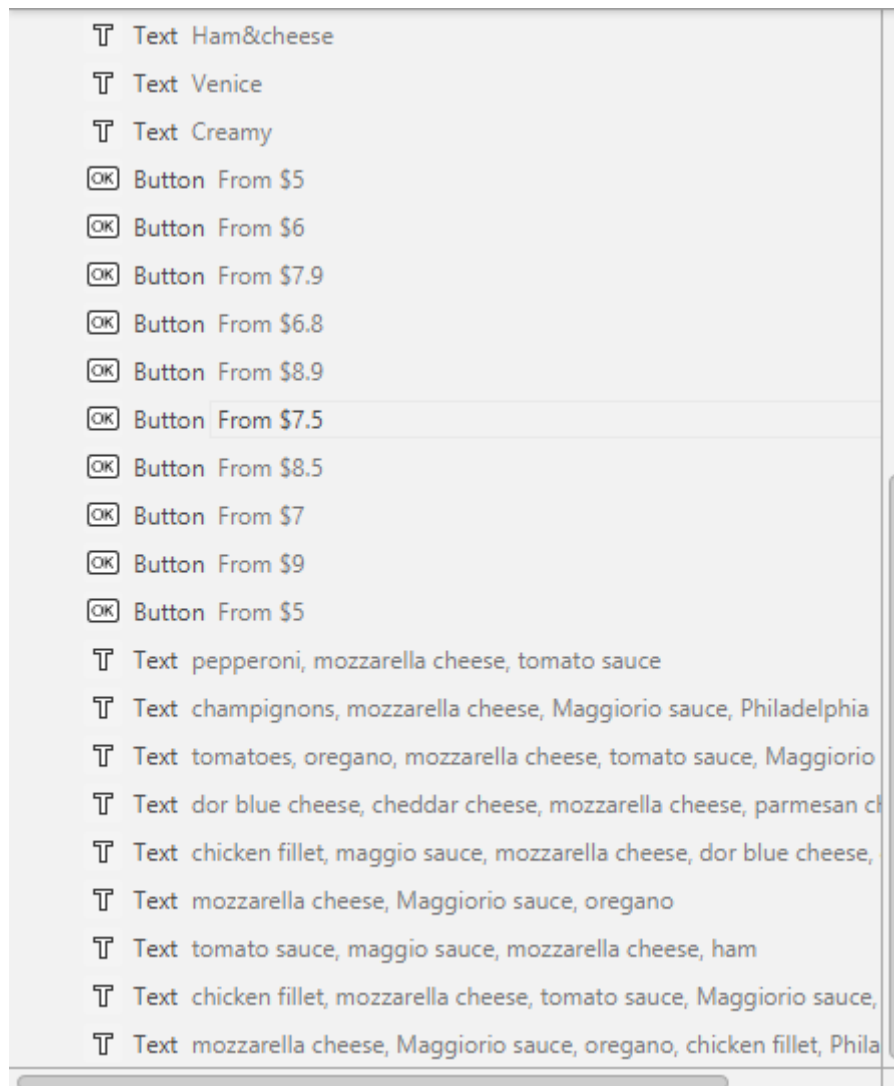


Рис. 21 – дерево.

Опишем составляющие данного окна:

На окне ассортимента у нас присутствуют элементы для вывода изображения `ImageView`, с помощью таких элементов мы можем вставлять картинки для нашей формы, но чтобы они отображались корректно нужно создать папку в нашем проекте и загрузить их туда (Рисунок 22).

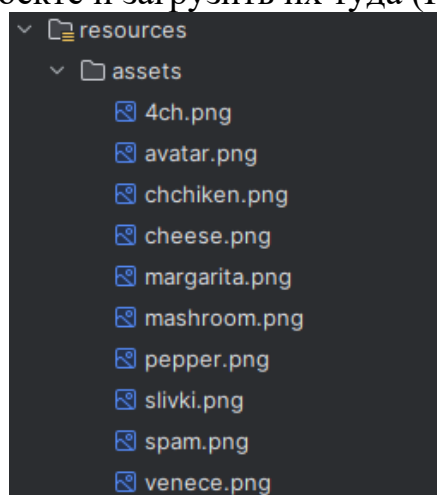


Рис. 22 – папка с изображениями.

Помимо этого элемента на форме присутствуют кнопки, пересылающие нас к (Рисунок 19):

1. Профилю клиента,
2. Информации,
3. Корзине;

Также на форме присутствуют и другие кнопки, они выполняют функцию добавления той или иной пищи в корзину (Рисунок 19).

Еще на форме присутствуют и обычные текстовые поля для описания и названий (Рисунок 19).

Следующим нашим этапом было создание окна с информацией, на котором были использованы текстовые элементы, и кнопка с помощью которой осуществляется переход на изначальную страницу (Рисунок 23).

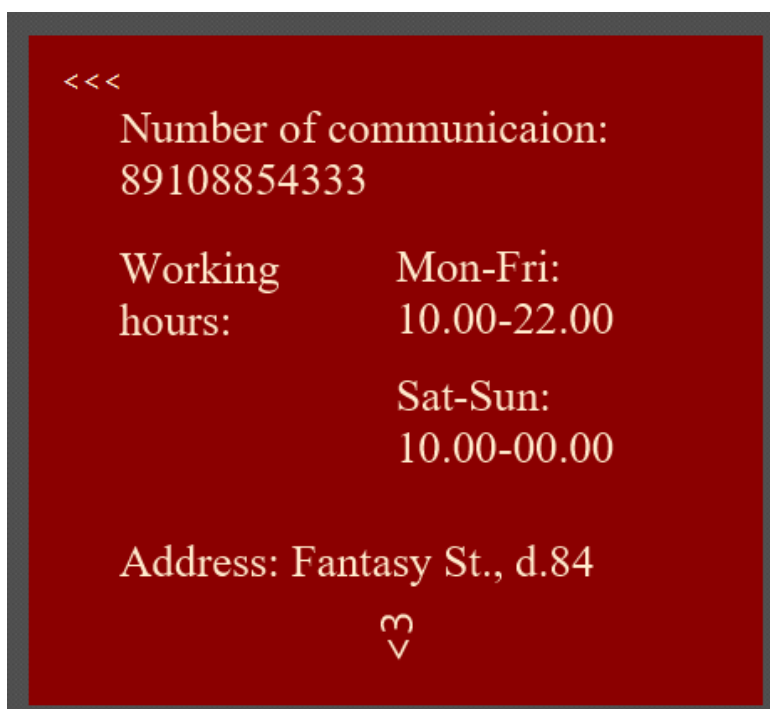


Рис. 23 – окно информации.

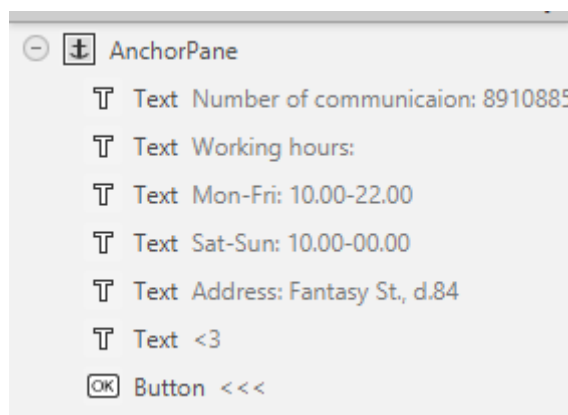


Рис. 24 – дерево.

Также для удобства использования приложения пользователем создадим окно, на котором будет выводиться информация о нем, то есть окно профиля (Рисунок 25).



Рис. 25 – окно профиля.

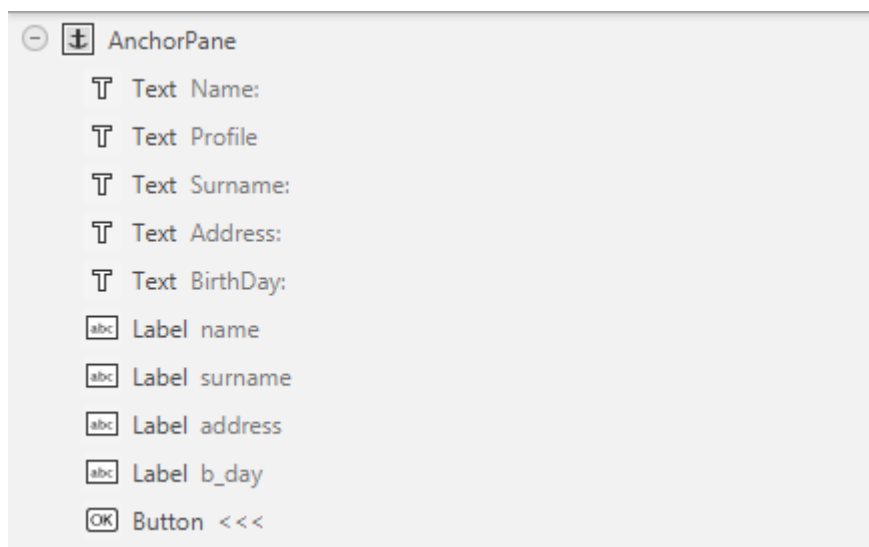


Рис. 26 – дерево.

При создании данного окна использовались не только текстовые элементы, но и также элементы вывода текста и возвращающая назад кнопка (Рисунок 26).

Следующим нашим шагом будет создание заключительного окна – окна корзины (Рисунок 27).



Рис. 27 – окно профиля.

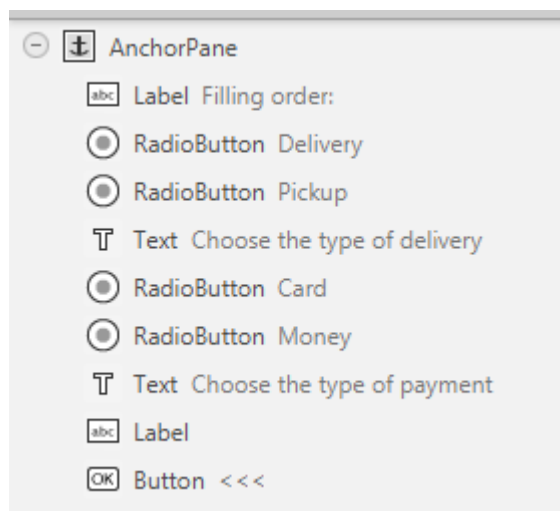


Рис. 28 – дерево.

На данном окне мы видим новые элементы, которые не использовались ранее на наших окнах – радиокнопки. Они отвечают за выборку того или иного критерия доставки или оплаты (Рисунок 27).

Но можем также и заметить уже знакомые для нас элементы, такие как кнопки и текстовые поля.

#### 4.3.2 Разработка логической части приложения

Для запуска нашего приложения используем класс `HelloApplication` созданный при помощи `JavaFX`.

```

package com.example.pizza;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 700, 400);
        stage.setScene(scene);
        stage.show();
        stage.setTitle("VivaSlice");
        stage.getIcons().add(new Image("file:images/pepperoni.png"));
    }

    public static void main(String[] args) {
        launch();
    }
}

```

Листинг 20 – класс HelloApplication.

Для подключения базы данных мы используем такой класс как DBConnection, который позволяет нам работать с нашей базой внутри приложения.

```

package com.example.pizza;

import java.sql.DriverManager;
import java.sql.SQLException;
public class DBConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/mydb";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "1111";

    public static java.sql.Connection getConnection() throws SQLException{
        return DriverManager.getConnection(URL, USERNAME, PASSWORD);
    }
}

```

Листинг 21 – класс DBConnection.

Еще у нас есть класс для подключения базы данных для ввода и логина и пароля и для внесения данных в окно регистрации.

```
package com.example.pizzza;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class LoginConnection {
    public boolean checkLogin(String login, String password){
        String query = "SELECT * FROM clients WHERE login = ? AND password = ?";
        try (Connection connection = DBConnection.getConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(query)) {
            preparedStatement.setString(1, login);
            preparedStatement.setString(2, password);
            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                return resultSet.next(); // возвращает true, если совпадение найдено,
            }
            иначе - false
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    public void createUser(String name, String surname, String login, String password){
        String query = "INSERT INTO clients(surname, name, login, password) VALUES (?, ?, ?, ?)";
        try (Connection connection = DBConnection.getConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(query)){
            preparedStatement.setString(1, surname);
            preparedStatement.setString(2, name);
            preparedStatement.setString(3, login);
            preparedStatement.setString(4, password);

            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

Листинг 22 – класс LoginConnection.

Класс, с помощью которого мы можем прописывать логику окна авторизации:

```
package com.example.pizza;  
  
import java.io.IOException;  
import java.net.URL;  
import java.util.ResourceBundle;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
import javafx.scene.Node;  
import javafx.event.ActionEvent;  
  
public class HelloController {  
  
    @FXML  
    private ResourceBundle resources;  
  
    @FXML  
    private URL location;  
  
    @FXML  
    private Button loginbut;  
  
    @FXML  
    private TextField loginfield;  
  
    @FXML  
    private PasswordField passwordfield;  
  
    @FXML  
    private Button register;  
  
    @FXML  
    //void initialize() {  
        //register.setOnAction(event ->{
```



```

// register.getScene().getWindow().hide();

// FXMLLoader loader = new FXMLLoader();
// loader.setLocation(getClass().getResource("register.fxml"));

// try {
//     loader.load();
// } catch (IOException e) {
//     throw new RuntimeException(e);
// }

// Parent root = loader.getRoot();
// Stage stage = new Stage();
// stage.setScene(new Scene(root));
// stage.showAndWait();
//});

//login.setOnAction(event ->{
//login.getScene().getWindow().hide();

//FXMLLoader loader = new FXMLLoader();
//loader.setLocation(getClass().getResource("firstmenu.fxml"));

//try {
//    loader.load();
//} catch (IOException e) {
//    throw new RuntimeException(e);
//}

//Parent root = loader.getRoot();
//Stage stage = new Stage();
//stage.setScene(new Scene(root));
//stage.showAndWait();

//});

private void loginButClicked(ActionEvent event) {
    String login = loginfield.getText();
    String password = passwordfield.getText();

```

```

LoginConnection loginConnection = new LoginConnection();

if (loginConnection.checkLogin(login, password)) {
    try {
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("firstmenu.fxml"));
        Parent root = loader.load();
        Stage stage = new Stage();
        stage.setScene(new Scene(root));

        Node source = (Node) event.getSource();
        Stage currentStage = (Stage) source.getScene().getWindow();
        currentStage.close();

        stage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
    } else {
        System.out.println("Error");
    }
}

@FXML
private void goToRegister(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("registration.fxml"));
    Stage stage = (Stage) register.getScene().getWindow();
    stage.setScene(new Scene(root));
}
}

```

Листинг 23 – класс HelloController

Также у нас есть классы, которые по аналогии с этим прописывают логику для других страниц (Листинг 24-28).

```

package com.example.pizzza;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;

```

```

import javafx.stage.Stage;

public class Basket {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private Button back;

    @FXML
    void initialize() {
        back.setOnAction(event ->{
            back.getScene().getWindow().hide();

            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(getClass().getResource("firstmenu.fxml"));

            try {
                loader.load();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }

            Parent root = loader.getRoot();
            Stage stage = new Stage();
            stage.setScene(new Scene(root));
            stage.show();

        });
    }
}

```

Листинг 24 - класс Basket.

```

package com.example.pizza;

```

```
import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;
```

```
public class FirstMenu {
```

```
    @FXML
    private ResourceBundle resources;
```

```
    @FXML
    private URL location;
```

```
    @FXML
    private ImageView ChChicken;
```

```
    @FXML
    private ImageView Chicken;
```

```
    @FXML
    private ImageView Fouch;
```

```
    @FXML
    private ImageView Hamch;
```

```
    @FXML
    private ImageView Margarita;
```

```
    @FXML
    private ImageView Mushroom;
```

```
    @FXML
    private ImageView Pepperoni;
```

```
    @FXML
    private Button basket;
```

```
@FXML
private Button ch;

@FXML
private Button chch;

@FXML
private Button cream;

@FXML
private ImageView creamy;

@FXML
private Button fourchick;

@FXML
private Button hum;

@FXML
private Button info;

@FXML
private Button marg;

@FXML
private Button mush;

@FXML
private Button pepper;

@FXML
private Button profile;

@FXML
private Button ven;

@FXML
private ImageView venice;

@FXML
void initialize(MouseEvent event) {

}
```

@FXML

void initialize() {

```
    profile.setOnAction(event ->{  
        profile.getScene().getWindow().hide();
```

```
  
        FXMLLoader loader = new FXMLLoader();  
        loader.setLocation(getClass().getResource("profile.fxml"));
```

```
  
        try {  
            loader.load();  
        } catch (IOException e) {  
            throw new RuntimeException(e);  
        }
```

```
  
        Parent root = loader.getRoot();  
        Stage stage = new Stage();  
        stage.setScene(new Scene(root));  
        stage.show();
```

```
    });
```

```
    info.setOnAction(event ->{  
        info.getScene().getWindow().hide();
```

```
  
        FXMLLoader loader = new FXMLLoader();  
        loader.setLocation(getClass().getResource("info.fxml"));
```

```
  
        try {  
            loader.load();  
        } catch (IOException e) {  
            throw new RuntimeException(e);  
        }
```

```
  
        Parent root = loader.getRoot();  
        Stage stage = new Stage();  
        stage.setScene(new Scene(root));  
        stage.show();
```

```

    });

    basket.setOnAction(event ->{
        basket.getScene().getWindow().hide();

        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("basket.fxml"));

        try {
            loader.load();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        Parent root = loader.getRoot();
        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.showAndWait();

    });

}

}

```

ЛИСТИНГ 25 – FirstMenu

```

package com.example.pizza;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class Info {

    @FXML

```

```

private ResourceBundle resources;

@FXML
private URL location;

@FXML
private Button back;

@FXML
void initialize() {

    back.setOnAction(event ->{
        back.getScene().getWindow().hide();

        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("firstmenu.fxml"));

        try {
            loader.load();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        Parent root = loader.getRoot();
        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.show();

    });
}
}

```

Листинг 26 – класс Info.

```

package com.example.pizzza;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;

```



```

import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class Profile {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private Button back;

    @FXML
    void initialize() {
        back.setOnAction(event ->{
            back.getScene().getWindow().hide();

            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(getClass().getResource("firstmenu.fxml"));

            try {
                loader.load();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }

            Parent root = loader.getRoot();
            Stage stage = new Stage();
            stage.setScene(new Scene(root));
            stage.showAndWait();

        });
    }
}

```

Листинг 27 – класс Profile;

```

package com.example.pizzza;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import javafx.event.ActionEvent;

public class Registration {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private TextField LoginReg;

    @FXML
    private TextField NameField;

    @FXML
    private PasswordField PasswordReg;

    @FXML
    private TextField SurnameField;

    @FXML
    private Button registration;

    @FXML
    public void register(ActionEvent event) {
        String Name = NameField.getText();
        String Surname = SurnameField.getText();
        String login = LoginReg.getText();
    }
}

```

```

String password = PasswordReg.getText();

LoginConnection loginConnection = new LoginConnection();
loginConnection.createUser(Name, Surname, login, password);

Node node = (Node) event.getSource();
Stage stage = (Stage) node.getScene().getWindow();
stage.close();

try{
    FXMLLoader loader = new FXMLLoader(getClass().getResource("hello-
view.fxml"));
    Parent root = loader.load();
    Stage primaryStage = new Stage();
    primaryStage.setScene(new Scene(root));
    primaryStage.show();
}catch (IOException e){
    e.printStackTrace();
}

//registration.setOnAction(event -> {
//    registration.getScene().getWindow().hide();

//    FXMLLoader loader = new FXMLLoader();
//    loader.setLocation(getClass().getResource("hello-view.fxml"));

//    try {
//        loader.load();
//    } catch (IOException e) {
//        throw new RuntimeException(e);
//    }

//    Parent root = loader.getRoot();
//    Stage stage = new Stage();
//    stage.setScene(new Scene(root));
//    stage.show();

//});

```

```
}  
  
}
```

Листинг 28 – класс Registration.

## 5. Тестирование

При тестировании приложения ошибок почти не было выявлено, но они все равно были. Так как ошибки были очень мелкими их было легко исправить.

Одной из таких ошибок является ошибка синтаксиса. Такую ошибку можно допустить при неправильном написании слова или при использовании нижнего регистра, а не верхнего или наоборот.

Также была допущена незначительная ошибка в логике открывания окна. Для некоторых окон, которые открываются не на долгое время и на них не происходит весомых действий нужно прописать `show` вместо `showAndWait`, как это было прописано изначально. Потому что, если не исправить эту ошибку наши окна будут становиться белыми при переходе.

После исправления всех вышеуказанных ошибок код работает корректно.

## 6. Руководство пользователя.

Когда пользователь запускает приложение он должен ввести свой логин и пароль (Рис.29).

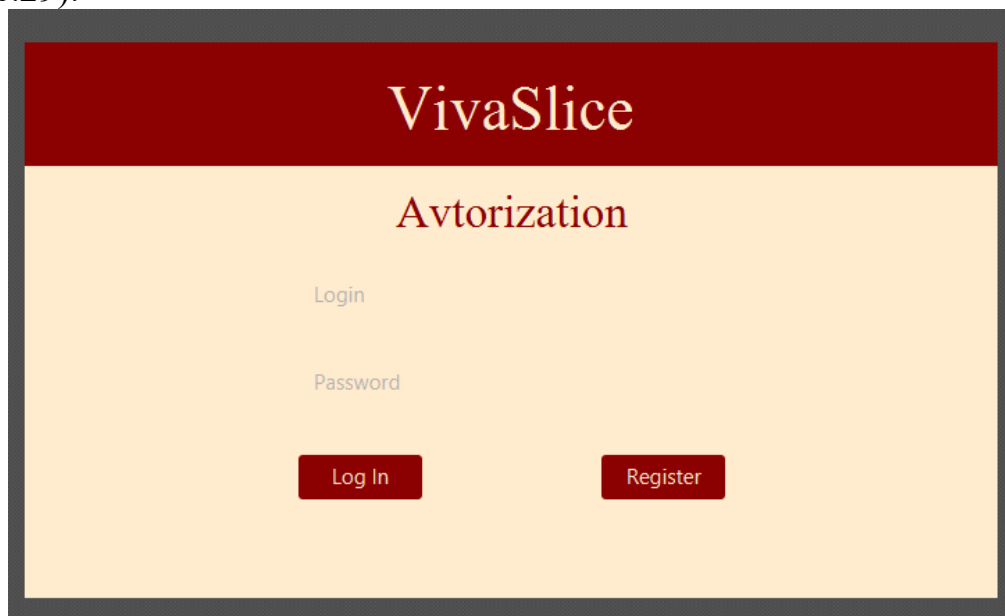
The image shows a web application window titled 'VivaSlice' with a dark red header. Below the header, the word 'Avtorization' is centered in a dark red font. There are two input fields: 'Login' and 'Password', both with light gray placeholder text. Below these fields are two dark red buttons: 'Log In' and 'Register'.

Рис. 29 – окно авторизации.

Также есть окно регистрации с помощью которого пользователь может создать аккаунт, если такого нет (Рис. 30).

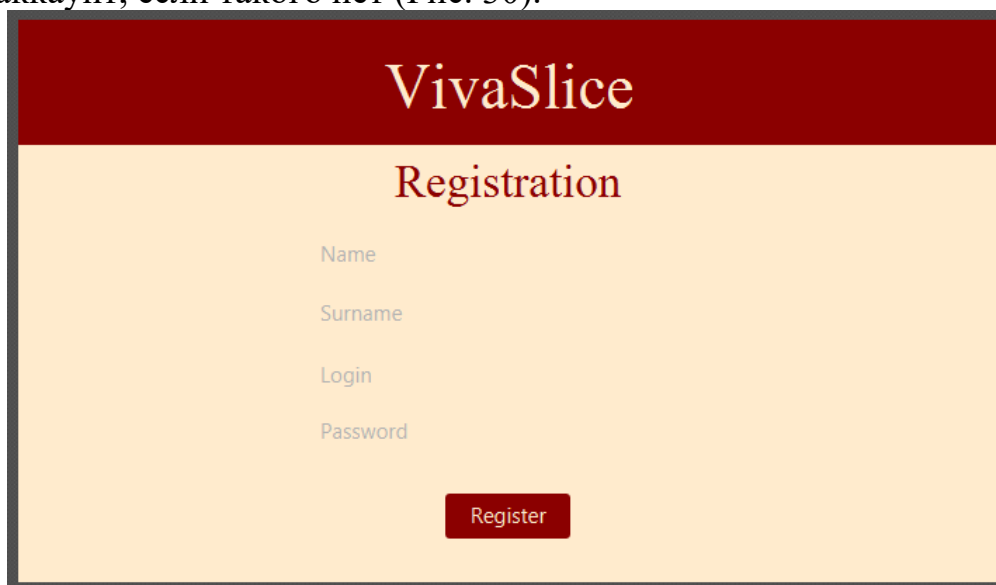
The image shows a web application window titled 'VivaSlice' with a dark red header. Below the header, the word 'Registration' is centered in a dark red font. There are four input fields: 'Name', 'Surname', 'Login', and 'Password', all with light gray placeholder text. Below these fields is a single dark red button labeled 'Register'.

Рис. 30 – окно регистрации.

На первом окне после прохождения регистрации клиент должен выбрать пиццу, которая добавиться в корзину. С окна корзина пользователь сможет выбрать нужные критерии для заказа и сделать его (Рис. 31-32).



Рис. 31 – окно меню.

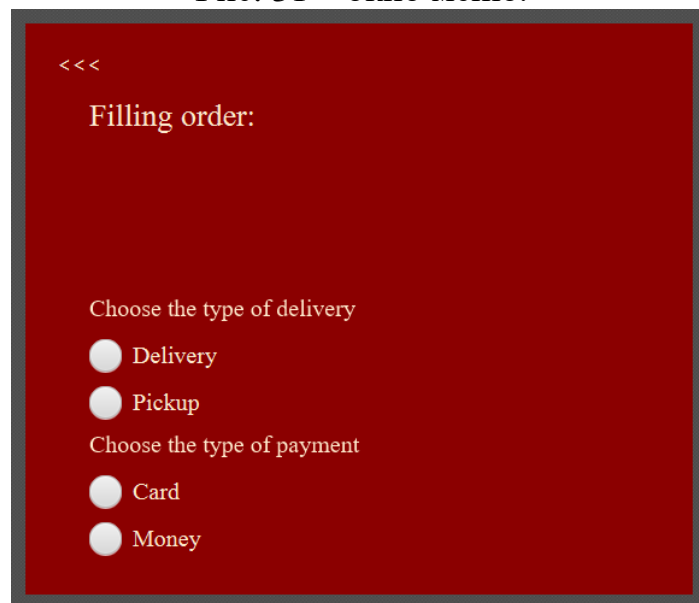


Рис. 32 – окно корзины.

Помимо этого, клиент сможет посмотреть информацию о своем профиле и о работе пиццерии (Рисунок 33-34).

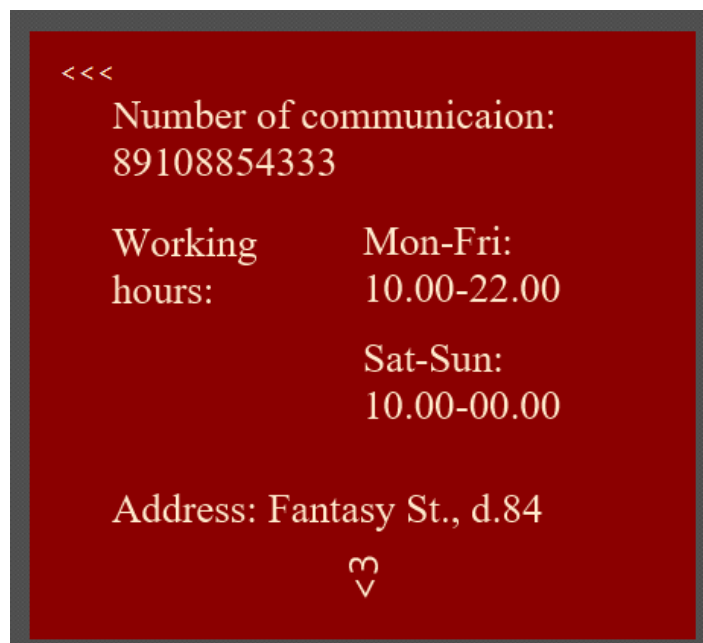


Рис. 33 – окно информации.

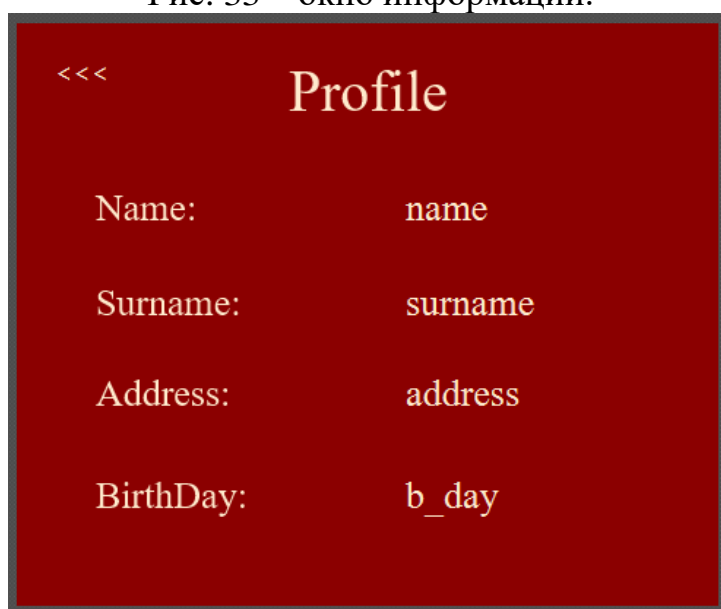


Рис. 34 – окно профиля.



### **Заключение**

В данной курсовой работе была создана база данных по предметной области «Пиццерия» с помощью инструмента для визуального проектирования баз данных My SQL и MySQL Workbench, спроектирована ERD-модель, создан интерфейс приложения с помощью JavaFX и база данных соединена с интерфейсом.

Из полученных результатов в ходе работы над курсовой работой можно сделать вывод, что цель достигнута и все поставленные задачи выполнены.

### **Список литературы.**

1. [How to connect JavaFX with SQLite – Eden Coding](#)
2. [Руководство по JavaFX \(metanit.com\)](#)
3. [tashirpizza.ru](#)