Изображение выглядит как графическая вставка, Графика, мультфильм

Автоматически созданное описание

**КУРСОВОЙ ПРОЕКТ**

**По дисциплине: МДК 01.01 Разработка программных модулей**

**Тема: «Разработка системы классов для музыкального магазина»**

**Специальность 09.02.07 «Информационные системы и программирование»**

|  |  |  |
| --- | --- | --- |
| **Выполнил студент(ка) группы 301ИС-22** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Е. Е Драбатулин** |
| **Руководитель** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Л. Б. Гусятинер** |

**Москва 2024**

Изображение выглядит как графическая вставка, Графика, мультфильм

Автоматически созданное описание

**УТВЕРЖДАЮ**

**Заместитель директора КМПО**

**С. Ф. Гасанов**

**«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.**

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

**По дисциплине: МДК 01.01 Разработка программных модулей**

**Специальность 09.02.07 «Информационные системы и программирование»**

**Студент 301ИС-22 Драбатулин Егор**

**ТЕМА: «Разработка системы классов для музыкального магазина»**

Дата выдачи задания «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г.

Срок сдачи проекта «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г.

**Москва 2024**

Оглавление

[**1. Введение.** 4](#_Toc184966772)

[**2. Основная часть.** 7](#_Toc184966773)

[**2.1. Архитектура приложения.** 7](#_Toc184966774)

[**2.2. Описание классов и их функциональности.** 8](#_Toc184966775)

[**2.3 Реализация консольного интерфейса и работа с данными.** 14](#_Toc184966776)

[**2.4. Примеры использования, тестирования, рекомендации по внедрению.** 16](#_Toc184966777)

[**2.4.1 Примеры использования.** 16](#_Toc184966778)

[**2.4.2. Тестирование отладка приложения.** 19](#_Toc184966779)

[**2.4.3 Рекомендации по внедрению.** 20](#_Toc184966780)

[**3. Заключение.** 20](#_Toc184966781)

[**4. Список использованных источников.** 24](#_Toc184966782)

# **1. Введение.**

В последние годы музыкальная индустрия в России претерпевает значительные изменения, вызванные как внутренними, так и внешними факторами. Одним из наиболее заметных аспектов является рост интереса к музыке и музыкальному обучению среди населения. С увеличением числа людей, желающих заниматься музыкой как хобби или профессией, малые и средние музыкальные магазины сталкиваются с рядом серьёзных проблем.

Проблемы малого и среднего музыкального бизнеса.

Одной из ключевых проблем является нехватка качественных бюджетных инструментов на российском рынке. Многие начинающие музыканты, студенты и любители ищут доступные по цене инструменты, которые соответствовали бы их требованиям, но часто сталкиваются с ограниченным выбором или высокими ценами на продукцию. Эта ситуация особенно актуальна для молодежи и студентов, которые не могут позволить себе дорогие инструменты от известных брендов. В результате этого многие потенциальные покупатели вынуждены отказываться от покупки или искать альтернативные варианты за пределами страны. Ситуация усугубляется экономическими санкциями, наложенными на Россию, которые значительно повлияли на импорт музыкальных инструментов и оборудования. Эти санкции привели к резкому увеличению цен на товары, что делает их недоступными для большинства покупателей. Малые и средние магазины, которые зависят от импорта, сталкиваются с проблемами в поддержании конкурентоспособности на рынке. В условиях высокой инфляции и нестабильной экономики многие магазины вынуждены повышать цены на свою продукцию, что еще больше ограничивает доступность товаров для конечного потребителя. Кроме того, многие производители музыкальных инструментов находятся за пределами России. В условиях международной политической напряженности возникают задержки в поставках или полное прекращение их. Это приводит к тому, что магазины не могут своевременно обновлять свои запасы, что негативно сказывается на их способности удовлетворять потребности клиентов.

Необходимость эффективного управления.

В таких условиях важность эффективного управления бизнесом становится критически актуальной. Владельцы музыкальных магазинов нуждаются в инструментах, которые помогут им оптимизировать процессы управления запасами, взаимодействия с клиентами и оформления заказов. Без современных решений эти магазины рискуют потерять клиентов и не справиться с растущими требованиями рынка. Код консольного приложения, представленный в данной работе, направлен на решение этих проблем. Он предоставляет простой и интуитивно понятный интерфейс для управления товарами, клиентами и заказами. Основные задачи, которые решает данное приложение:

1. Упрощение учета товаров. Автоматизация процессов добавления и удаления товаров позволяет избежать ошибок при ручном вводе данных. Например, приложение позволяет быстро добавлять новые товары в систему с указанием их названия, цены и количества на складе. Это особенно важно для небольших магазинов, где сотрудники могут выполнять множество задач одновременно.
2. Управление клиентской базой. Возможность быстро добавлять новых клиентов и отслеживать их заказы помогает улучшить качество обслуживания. Продавцы смогут быстрее реагировать на запросы клиентов и предлагать им персонализированные решения. Например, если клиент возвращается в магазин для повторной покупки, продавец может быстро найти информацию о предыдущих покупках клиента.
3. Обработка заказов. Упрощение процесса оформления заказов позволяет продавцам быстрее обслуживать клиентов, что повышает уровень удовлетворенности. Быстрое оформление заказов может стать конкурентным преимуществом в условиях высокой конкуренции.
4. Сохранение данных. Использование формата JSON для хранения данных о товарах и клиентах обеспечивает долговременное хранение информации и легкость ее обработки. Это позволяет владельцам магазина быстро получать доступ к необходимым данным для анализа продаж и планирования закупок.
5. Генерация чеков. Приложение автоматически генерирует чеки для каждого заказа в текстовом формате, что упрощает процесс учета продаж и позволяет клиентам получать подтверждение своих покупок.

Таким образом, данное решение поможет малым и средним музыкальным магазинам в России оптимизировать свои бизнес-процессы, повысить эффективность работы и улучшить взаимодействие с клиентами. Приложение не только упрощает управление магазином, но также способствует повышению уровня обслуживания клиентов и укреплению позиций на рынке в условиях экономической нестабильности.

Кроме того, приложение может стать основой для дальнейшего развития бизнеса: его функциональность можно расширять в зависимости от потребностей пользователей и изменений на рынке. Например, можно добавить возможность интеграции с онлайн-магазинами или платформами для проведения акций и распродаж. В заключение можно отметить, что создание эффективной системы управления музыкальным магазином является важным шагом к успешному ведению бизнеса в условиях современных экономических реалий России. Консольное приложение на Python представляет собой доступное решение для владельцев музыкальных магазинов, позволяющее им адаптироваться к изменяющимся условиям рынка и удовлетворять потребности своих клиентов.

# **2. Основная часть.**

# **2.1. Архитектура приложения.**

Изображение выглядит как диаграмма, текст, Технический чертеж, План

Автоматически созданное описание

Рисунок 1. Блок-схема кода

Архитектура приложения для управления музыкальным магазином построена на принципах объектно-ориентированного программирования (ООП), что позволяет организовать код в виде классов и объектов, обеспечивая модульность, гибкость и простоту в поддержке и расширении системы.

Приложение состоит из нескольких ключевых компонентов, каждый из которых выполняет свою уникальную роль:

1. Классы. В приложении выделены четыре основных класса:

* «Product»: отвечает за представление товаров в магазине;
* «Customer»: описывает клиентов магазина;
* «Order»: управляет заказами клиентов;
* «Store»: основной класс, который управляет всеми товарами, клиентами и заказами.

1. Методы. Каждый класс содержит методы, которые позволяют выполнять операции над его данными. Например, класс «Product» имеет методы для продажи товара и обновления его количества на складе, а класс «Store» включает методы для добавления товаров, создания заказов и сохранения данных.
2. Консольный интерфейс. Приложение взаимодействует с пользователем через текстовый интерфейс, который предлагает различные команды для выполнения операций. Это упрощает использование приложения и делает его доступным для пользователей без необходимости в графическом интерфейсе.
3. Хранение данных. Данные о товарах, клиентах и заказах хранятся в формате JSON, что обеспечивает долговременное хранение информации и легкость ее обработки.

Вывод. Эта архитектура позволяет эффективно управлять бизнес-процессами музыкального магазина, обеспечивая удобство использования и простоту в обслуживании.

# **2.2. Описание классов и их функциональности.**

Каждый из классов приложения выполняет свою уникальную роль:

Класс «Product».

Класс «Product» отвечает за представление товаров в магазине. Он включает следующие атрибуты:

− «product\_id»: уникальный идентификатор товара;

− «name»: название товара;

− «price»: цена товара;

− «stock»: количество товара на складе.

*Методы класса*:

− «sell(quantity)»: уменьшает количество товара на складе при продаже и возвращает True, если продажа успешна; иначе возвращает False. Этот метод проверяет наличие достаточного количества товара перед его продажей, что помогает избежать ситуации с нехваткой запасов.

− «to\_dict()»: преобразует объект в словарь для удобства сохранения данных при записи в файл JSON.

class Product:  
 def \_\_init\_\_(self, product\_id, name, price, stock):  
 self.product\_id = product\_id  
 self.name = name  
 self.price = price  
 self.stock = stock  
  
 def sell(self, quantity):  
 if quantity <= self.stock:  
 self.stock -= quantity  
 return True  
 return False  
  
 def to\_dict(self):  
 return {  
 'product\_id': self.product\_id,  
 'name': self.name,  
 'price': self.price,  
 'stock': self.stock  
 }

Класс «Customer».

Класс «Customer» предназначен для хранения информации о клиентах магазина. Он включает следующие атрибуты:

− «customer\_id»: уникальный идентификатор клиента;

− «name»: имя клиента;

− «email»: электронная почта клиента.

*Методы класса*:

− «to\_dict()»: преобразует объект в словарь для сохранения данных о клиенте. Это позволяет легко сериализовать информацию о клиентах при сохранении в файл.

class Customer:  
 def \_\_init\_\_(self, customer\_id, name, email):  
 self.customer\_id = customer\_id  
 self.name = name  
 self.email = email  
  
 def to\_dict(self):  
 return {  
 'customer\_id': self.customer\_id,  
 'name': self.name,  
 'email': self.email  
 }

Класс «Order».

Класс «Order» управляет заказами клиентов и включает следующие атрибуты:

− «order\_id»: уникальный идентификатор заказа;

− «customer»: объект класса «Customer», представляющий клиента;

− «products»: список товаров в заказе.

*Методы класса*:

* «add\_product(product, quantity)»: добавляет товар в заказ; проверяет наличие на складе перед добавлением. Если товар успешно добавлен, метод возвращает True; иначе – False.
* «total\_price()»: вычисляет общую стоимость заказа путем суммирования цен всех добавленных товаров с учётом их количества.
* «to\_dict()»: преобразует объект заказа в словарь для сохранения данных о заказе.
* «save\_receipt()»: сохраняет чек заказа в текстовом файле. Этот метод генерирует текстовый файл с информацией о заказе, включая данные о клиенте и списке купленных товаров.

class Order:  
 def \_\_init\_\_(self, order\_id, customer):  
 self.order\_id = order\_id  
 self.customer = customer  
 self.products = []  
  
 def add\_product(self, product, quantity):  
 if product.sell(quantity):  
 self.products.append((product, quantity))  
 return True  
 return False  
  
 def total\_price(self):  
 return sum(product.price \* quantity for product, quantity in self.products)  
  
 def to\_dict(self):  
 return {  
 'order\_id': self.order\_id,  
 'customer': {'customer\_id': self.customer.customer\_id},  
 'products': [(product.product\_id, quantity) for product, quantity in self.products]  
 }  
  
 def save\_receipt(self):  
 receipt\_filename = f"receipt\_{self.order\_id}.txt"  
 with open(receipt\_filename, 'w', encoding='utf-8') as f:  
 f.write(f"Чек для заказа ID: {self.order\_id}\n")  
 f.write(f"Клиент: {self.customer.name}\n")  
 f.write(f"Электронная почта: {self.customer.email}\n")  
 f.write("Товары:\n")  
 for product, quantity in self.products:  
 f.write(f"- {product.name}: {quantity} шт. по {product.price} руб.\n")  
 f.write(f"Общая сумма: {self.total\_price()} руб.\n")  
  
 print(f"Чек сохранен в файл: {receipt\_filename}")

Класс «Store».

Класс «Store» является основным классом приложения и управляет всеми товарами, клиентами и заказами. Он включает методы для добавления и удаления товаров и клиентов (методы «add\_product», «remove\_product», «add\_customer»), создания заказов (метод «create\_order») и сохранения данных в файл (метод «save\_to\_file»).

*Методы класса*:

* «add\_product(name, price, stock)»: создает новый объект класса «Product» и добавляет его в список товаров магазина;
* «remove\_product()»: удаляет товар из магазина по ID с помощью метода поиска по списку товаров;
* «add\_customer(name, email)»: создает новый объект класса «Customer» и добавляет его в список клиентов магазина;
* «create\_order()»: создает новый заказ для существующего клиента по его ID. Этот метод позволяет продавцам легко оформлять заказы на основе выбранных товаров;
* «save\_to\_file(filename)»: сохраняет данные о товарах, клиентах и заказах в файл JSON.
* «load\_from\_file(filename)»: загружает данные из файла JSON при запуске приложения;
* методы отображения списков («display\_products()», «display\_customers()», «display\_orders()») выводят информацию о текущих запасах товаров клиентов и заказах на экран.

import json  
from product import Product  
from customer import Customer  
from order import Order  
  
  
class Store:  
 def \_\_init\_\_(self):  
 self.products = []  
 self.customers = []  
 self.orders = []  
  
 def add\_product(self, name, price, stock):  
 product\_id = len(self.products) + 1 # Генерация уникального ID товара  
 new\_product = Product(product\_id, name, price, stock)  
 self.products.append(new\_product)  
 print(f"Товар '{name}' добавлен.")  
  
 def remove\_product(self):  
 self.display\_products()  
 product\_id = int(input("Введите ID товара для удаления: "))  
  
 for i, product in enumerate(self.products):  
 if product.product\_id == product\_id:  
 del self.products[i]  
 print(f"Товар с ID {product\_id} удален.")  
 return  
  
 print(f"Товар с ID {product\_id} не найден.")  
  
 def add\_customer(self, name, email):  
 customer\_id = len(self.customers) + 1 # Генерация уникального ID клиента  
 new\_customer = Customer(customer\_id, name, email)  
 self.customers.append(new\_customer)  
 print(f"Клиент '{name}' добавлен.")  
  
 def create\_order(self):  
 self.display\_customers()  
  
 customer\_id = int(input("Введите ID клиента для заказа: "))  
  
 customer = next((c for c in self.customers if c.customer\_id == customer\_id), None)  
  
 if not customer:  
 print(f"Клиент с ID {customer\_id} не найден.")  
 return  
  
 order = Order(len(self.orders) + 1, customer)  
  
 while True:  
 self.display\_products()  
  
 product\_id = int(input("Введите ID товара для заказа (или 0 для завершения): "))  
  
 if product\_id == 0:  
 break  
  
 product = next((p for p in self.products if p.product\_id == product\_id), None)  
  
 if not product:  
 print(f"Товар с ID {product\_id} не найден.")  
 continue  
  
 quantity = int(input(f"Введите количество товара '{product.name}': "))  
  
 if order.add\_product(product, quantity):  
 print(f"Добавлено {quantity}x '{product.name}' в заказ.")  
 else:  
 print(f"Недостаточно товара '{product.name}' на складе.")  
  
 self.orders.append(order)  
 print(f"Заказ ID {order.order\_id} создан для клиента {customer.name}.")  
  
 # Сохранение чека в текстовый файл  
 order.save\_receipt()  
  
 def save\_to\_file(self, filename):  
 data = {  
 'products': [product.to\_dict() for product in self.products],  
 'customers': [customer.to\_dict() for customer in self.customers],  
 'orders': [order.to\_dict() for order in self.orders]  
 }  
  
 with open(filename, 'w', encoding='utf-8') as f:  
 json.dump(data, f, ensure\_ascii=False, indent=4)  
  
 def load\_from\_file(self, filename):  
 with open(filename, 'r', encoding='utf-8') as f:  
 data = json.load(f)  
  
 for product\_data in data['products']:  
 self.add\_product(product\_data['name'], product\_data['price'], product\_data['stock'])  
  
 for customer\_data in data['customers']:  
 self.add\_customer(customer\_data['name'], customer\_data['email'])  
  
 for order\_data in data['orders']:  
 customer = next(c for c in self.customers if c.customer\_id == order\_data['customer']['customer\_id'])  
 order = Order(order\_data['order\_id'], customer)  
  
 for product\_info in order\_data['products']:  
 product = next(p for p in self.products if p.product\_id == product\_info[0])  
 order.add\_product(product, product\_info[1])  
  
 self.orders.append(order)  
  
 def display\_products(self):  
 print("\nСписок товаров:")  
 for product in self.products:  
 print(f"{product.product\_id}. {product.name} - {product.price} руб., Остаток: {product.stock}")  
  
 def display\_customers(self):  
 print("\nСписок клиентов:")  
 for customer in self.customers:  
 print(f"{customer.customer\_id}. {customer.name} - {customer.email}")  
  
 def display\_orders(self):  
 # Логика просмотра заказов по ID  
 if not self.orders:  
 print("Нет созданных заказов.")  
 return  
  
 for order in self.orders:  
 products\_list = ', '.join([f"{quantity}x {product.name}" for product, quantity in order.products])  
 total\_price = order.total\_price()  
 print(  
 f"Заказ ID {order.order\_id}: Клиент - {order.customer.name}, Товары - [{products\_list}], Общая сумма - {total\_price} руб.")

# **2.3 Реализация консольного интерфейса и работа с данными.**

Консольный интерфейс приложения позволяет пользователям взаимодействовать с системой через текстовые команды.

Приложение предлагает меню с различными опциями:

1. Добавить товар;

2. Удалить товар;

3. Добавить клиента;

4. Создать заказ;

5. Сохранить данные;

6. Вывести список товаров;

7. Вывести список клиентов;

8. Вывести список заказов;

9. Выход.

Каждая опция обрабатывается соответствующим блоком кода:

from store import Store  
  
def main():  
 store = Store()  
  
 # Загружаем данные из файла при запуске приложения (если файл существует)  
 try:  
 store.load\_from\_file('store\_data.json')  
 except FileNotFoundError:  
 print("Файл данных не найден. Создан новый магазин.")  
  
 while True:  
 print("\n--- Музыкальный Магазин | F | C | Db | Eb | ---")  
 print("1. Добавить товар")  
 print("2. Удалить товар")  
 print("3. Добавить клиента")  
 print("4. Создать заказ")  
 print("5. Сохранить данные")  
 print("6. Вывести список товаров")  
 print("7. Вывести список клиентов")  
 print("8. Вывести список заказов")  
 print("9. Выход")  
  
 choice = input("Выберите действие: ")  
  
 # Обработка выбора пользователя  
 if choice == "1":  
 name = input("Введите название товара: ")  
 price = float(input("Введите цену товара: "))  
 stock = int(input("Введите количество на складе: "))  
 store.add\_product(name, price, stock)  
  
 elif choice == "2":  
 store.remove\_product()  
  
 elif choice == "3":  
 name = input("Введите имя клиента: ")  
 email = input("Введите электронную почту клиента: ")  
 store.add\_customer(name, email)  
  
 elif choice == "4":  
 store.create\_order()  
  
 elif choice == "5":  
 store.save\_to\_file('store\_data.json')  
 print("Данные сохранены в файл.")  
  
 elif choice == "6":  
 store.display\_products()  
  
 elif choice == "7":  
 store.display\_customers()  
  
 elif choice == "8":  
 store.display\_orders()  
  
 elif choice == "9":  
 break  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Для обеспечения долговременного хранения данных приложение использует формат JSON для сохранения информации о товарах, клиентах и заказах в файл на диске.

Метод «save\_to\_file» класса «Store» записывает данные в файл JSON с использованием параметров для улучшения читаемости:

def save\_to\_file(self, filename):  
 data = {  
 'products': [product.to\_dict() for product in self.products],  
 'customers': [customer.to\_dict() for customer in self.customers],  
 'orders': [order.to\_dict() for order in self.orders]  
 }  
  
 with open(filename, 'w', encoding='utf-8') as f:  
 json.dump(data, f, ensure\_ascii=False, indent=4)

Данные сохраняются в формате JSON с отступами для удобства чтения человеком.

# **2.4. Примеры использования, тестирования, рекомендации по внедрению.**

# **2.4.1 Примеры использования.**

Приложение позволяет пользователям выполнять различные операции через консольный интерфейс:

1. Добавление товара. Пользователь может добавить новый товар в магазин путем ввода названия товара его цены и количества на складе.

* Введите название товара: Электрогитара FENDER VINTERA '60s Stratocaster Ice Blue Metallic;
* Введите цену товара: 195000;
* Введите количество на складе: 5;

1. Удаление товара. Пользователь может удалить товар из магазина по его ID.

Список товаров:

1. Электрогитара S by Solar SB4.6FRFBR – 45 990,0 руб., Остаток: 15;

2. Электрогитара S by Solar EB4.6W – 36 990,0 руб., Остаток: 15;

3. Электрогитара FENDER VINTERA '60s Stratocaster Ice Blue Metallic – 195 000,0 руб., Остаток: 5;

4. Электрогитара FENDER SQUIER 40th Anniversary Telecaster LRL Sherwood Green – 88 000,0 руб., Остаток: 15;

5. Электрогитара Schecter BANSHEE-6 SGR SBK – 30 096,0 руб., Остаток: 12.

Введите ID товара для удаления: 1

Товар с ID 1 удален.

1. Добавление клиента. Пользователь может добавить нового клиента путем ввода его имени электронной почты.

* Введите имя клиента: Мэттью Беллами;
* Введите электронную почту клиента: PlugInBaby@gmail.com;
* Клиент 'Мэттью Беллами' добавлен.

1. Создание заказа. Пользователь может создать новый заказ для существующего клиента по его ID добавить товары по их ID.

Список клиентов:

1. Мэттью Беллами - PlugInBaby@gmail.com;
2. Крис Уолстенхолм - BassKing@gmail.com;
3. Джеймс Хэтфилд - DownpickKing@gmail.com;
4. Синистер Гейтс - SweepLord@gmail.com;
5. Джонни Гринвуд - JonnyGreen@gmail.com;
6. Кирк Хэмметт - WahWahMan@gmail.com.

Введите ID клиента для заказа: 4

Список товаров:

1. Электрогитара S by Solar SB4.6FRFBR – 45 990,0 руб., Остаток: 15;
2. Электрогитара S by Solar EB4.6W – 36 990,0 руб., Остаток: 15;
3. Электрогитара FENDER VINTERA '60s Stratocaster Ice Blue Metallic – 195 000,0 руб., Остаток: 5;
4. Электрогитара FENDER SQUIER 40th Anniversary Telecaster LRL Sherwood Green – 88 000,0 руб., Остаток: 15;
5. Электрогитара Schecter BANSHEE-6 SGR SBK – 30 096,0 руб., Остаток: 12;

Введите ID товара для заказа (или 0 для завершения): 1

Введите количество товара ' Электрогитара S by Solar SB4.6FRFBR ': 1

Добавлено 1x ' Электрогитара S by Solar SB4.6FRFBR ' в заказ.

Заказ ID 4: Клиент – Синистер Гейтс...

1. Сохранение данных. Все изменения могут быть сохранены в файл JSON для долговременного хранения информации о магазине.

# **2.4.2. Тестирование отладка приложения.**

Тестирование является важной частью разработки программного обеспечения; оно помогает выявить ошибки до того момента как приложение будет запущено в эксплуатацию или предоставлено конечному пользователю.

В процессе тестирования данного консольного приложения были проведены следующие этапы:

1. Юнит-тестирование. Каждый класс тестировался отдельно с использованием различных входных данных для проверки корректности работы методов классов («Product», «Customer», «Order», «Store»). Это позволило убедиться в том, что каждый компонент работает правильно независимо от других частей системы.
2. Интеграционное тестирование. После юнит-тестирования проводилось интеграционное тестирование всех компонентов вместе – проверялось взаимодействие классов между собой при выполнении операций добавления товаров или создания заказов.
3. Тестирование пользовательского интерфейса. Проверялась интуитивность интерфейса – насколько легко пользователям выполнять операции через консольный интерфейс без дополнительных инструкций или помощи разработчика.
4. Тестирование производительности. Оценивалась скорость выполнения операций при работе с большим количеством данных – например при добавлении большого количества товаров или создании множества заказов одновременно.

Тестирование проводилось как вручную, так и автоматически с использованием библиотек Python, таких как – unittest или pytest, которые позволяют создавать тесты для проверки различных аспектов работы программы.

# **2.4.3 Рекомендации по внедрению.**

Для успешного внедрения разработанного консольного приложения рекомендуется следующее:

1. Обучение персонала. Проведение тренингов для сотрудников, чтобы они могли эффективно использовать новую систему управления магазином.
2. Постепенное внедрение системы. Начать использование приложения параллельно с существующими процессами чтобы минимизировать риски, связанные с переходом на новую систему.
3. Сбор обратной связи от пользователей. Регулярно собирать отзывы от сотрудников о работе системы чтобы выявить возможные проблемы улучшить функциональность приложения согласно потребностям пользователей.
4. Планирование обновлений системы. регулярно обновлять приложение чтобы исправлять ошибки добавлять новые функции согласно изменяющимся требованиям бизнеса.

# **3. Заключение.**

Разработанное консольное приложение для управления музыкальным магазином представляет собой важный инструмент, который отвечает на актуальные потребности малого и среднего музыкального бизнеса в России. В условиях экономической нестабильности, вызванной как внутренними, так и внешними факторами, включая санкции и рост цен на импортные товары, данное решение становится особенно актуальным.

Ключевые достижения приложения.

1. Автоматизация управления. Приложение значительно упрощает процессы учета товаров и управления запасами. Владельцы магазинов могут быстро добавлять новые товары, удалять устаревшие позиции и отслеживать наличие на складе. Это не только снижает вероятность ошибок, связанных с ручным вводом данных, но и позволяет минимизировать риски потери продаж из-за отсутствия товара. В условиях высокой конкуренции, где каждая продажа имеет значение, автоматизация процессов становится критически важной для поддержания прибыльности бизнеса.
2. Улучшение обслуживания клиентов. Возможность регистрации клиентов и отслеживания их заказов помогает повысить качество обслуживания. Продавцы могут быстро находить информацию о предыдущих покупках клиентов, что способствует созданию более персонализированного подхода к каждому покупателю. Это укрепляет доверие клиентов и повышает вероятность повторных покупок. Кроме того, приложение может помочь в выявлении предпочтений клиентов, что позволит магазинам предлагать более целевые маркетинговые акции.
3. Эффективное оформление заказов. Упрощенный процесс создания заказов позволяет продавцам быстрее обслуживать клиентов. Это особенно важно в условиях высокой конкуренции, когда скорость обслуживания может стать решающим фактором для привлечения и удержания клиентов. Приложение позволяет легко добавлять товары в заказ, что сокращает время ожидания для клиента.
4. Генерация чеков. Автоматическая генерация чеков в текстовом формате улучшает клиентский опыт и упрощает процесс учета продаж. Чеки служат подтверждением покупки и могут использоваться для учета финансовых операций. Возможность быстро распечатать или отправить чек по электронной почте также улучшает клиентский опыт, так как клиенты получают документальное подтверждение своих покупок.
5. Долговременное хранение данных. Использование формата JSON для хранения информации о товарах, клиентах и заказах обеспечивает долговременное хранение данных и легкость их обработки. Это позволяет владельцам магазина быстро получать доступ к необходимым данным для анализа продаж и планирования закупок. Более того, возможность загружать данные из файла при запуске приложения делает его использование более удобным.

Перспективы развития.

Приложение не только решает текущие проблемы малого и среднего музыкального бизнеса, но также открывает новые горизонты возможностей для дальнейшего развития. В будущем можно рассмотреть следующие направления:

1. Расширение функциональности. Постоянное улучшение приложения путем добавления новых функций позволит адаптироваться к меняющимся условиям рынка и требованиям клиентов. Например, можно интегрировать систему с онлайн-магазинами или платформами для проведения акций, что может значительно увеличить охват аудитории.
2. Мобильная версия приложения. Разработка мобильной версии позволит пользователям управлять своим бизнесом из любой точки мира, что особенно актуально для владельцев магазинов, которые часто находятся вне офиса. Мобильное приложение может включать функции уведомлений о новых поступлениях или акциях, что поможет поддерживать интерес клиентов.
3. Аналитика и прогнозирование. Внедрение аналитических инструментов поможет владельцам магазинов лучше понимать поведение своих клиентов, предсказывать спрос на определенные товары и оптимизировать запасы. Использование данных о продажах для анализа тенденций может помочь в принятии более обоснованных решений о закупках.
4. Интеграция с CRM-системами. Связывание приложения с системами управления взаимоотношениями с клиентами (CRM) позволит более эффективно управлять взаимодействием с клиентами и улучшить маркетинговые стратегии. Это может включать автоматизацию рассылок новостей о новых товарах или акциях.
5. Обратная связь от пользователей. Регулярный сбор отзывов от пользователей поможет выявить слабые места программы и определить направления её дальнейшего развития. Создание платформы для обсуждения идей по улучшению приложения может помочь в выявлении потребностей пользователей.
6. Обучающие материалы. Разработка обучающих материалов и видеоруководств по использованию приложения позволит новым пользователям быстрее освоиться с функционалом программы и повысит общую удовлетворённость от использования системы.
7. Интеграция новых технологий. Внедрение технологий машинного обучения может позволить приложению анализировать данные о продажах и поведении клиентов для создания прогнозов по спросу на товары или предложения персонализированных рекомендаций.

В условиях растущей конкуренции и экономической нестабильности малые и средние музыкальные магазины должны активно искать способы оптимизации своих бизнес-процессов. Разработанное консольное приложение на Python предоставляет доступное решение для автоматизации управления запасами, клиентами и заказами, что позволяет повысить эффективность работы магазина и улучшить качество обслуживания клиентов. Таким образом, данное приложение не только отвечает на текущие вызовы музыкального бизнеса в России, но также создает основу для будущего роста и развития в условиях меняющегося рынка. Владельцы музыкальных магазинов могут быть уверены в том, что с помощью этого инструмента они смогут успешно адаптироваться к новым условиям и удовлетворять потребности своих клиентов. В конечном итоге это приложение представляет собой не просто инструмент управления – это стратегический актив, который поможет музыкальным магазинам не только выжить в условиях кризиса, но и процветать благодаря эффективному управлению ресурсами и повышению уровня обслуживания клиентов. Это приложение становится надежным помощником владельцев музыкальных магазинов в их стремлении к успеху на конкурентном рынке, предоставляя им все необходимые инструменты для эффективного управления бизнесом в условиях современных вызовов. С его помощью владельцы смогут не только справляться с текущими трудностями, но также находить новые возможности для роста и расширения своего бизнеса в будущем. Кроме того, успешная реализация данного проекта может вдохновить других разработчиков на создание аналогичных решений для других сегментов розничной торговли или услуг в России. Это подчеркивает важность инноваций в управлении бизнесом и необходимость адаптации к новым экономическим условиям. Таким образом, консольное приложение становится не просто инструментом – это комплексное решение для управления бизнесом в условиях современных вызовов рынка музыкальных инструментов в России. Оно предоставляет владельцам возможность не только эффективно управлять текущими операциями магазина, но также стратегически планировать будущее своего бизнеса с учетом изменений на рынке и потребностей клиентов. В заключение можно сказать, что данное приложение является важным шагом к цифровизации музыкального бизнеса в России. Оно открывает новые горизонты возможностей как для владельцев магазинов, так и для их клиентов – обеспечивая удобство покупок, доступ к качественным товарам по разумным ценам и высокий уровень обслуживания.

# **4. Список использованных источников.**

1.”Добрый, добрый Python с Сергеем Балакиревым”

<https://stepik.org/course/100707/syllabus>;

2. «Object-Oriented Programming in Python»

https://www.freecodecamp.org/news/object-oriented-programming-in-python/;

3 . "JSON: Полное руководство"

https://www.json.org/json-en.html.

Эта работа подчеркивает важность разработки эффективных инструментов управления бизнесом в условиях современного рынка торговли музыкальными инструментами; она показывает возможности языка Python как мощного инструмента для создания таких приложений благодаря своей простоте использования богатству библиотек для работы с данными создания пользовательских интерфейсов.

Кроме того, данный проект служит отличным примером применения принципов ООП, а также методов разработки программного обеспечения, которые могут быть полезны не только начинающим программистам, но также опытным разработчикам, стремящимся улучшить свои навыки программирования.