МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Кафедра математического обеспечения и суперкомпьютерных технологий

Направление подготовки: «Программная инженерия»

ОТЧЕТ

по лабораторной работе дисциплины
«Стандарты и средства управления качеством процесса»

на тему:

«Разработка концепции информационной системы и её тестирование»

Выполнил: студент группы 3823М1ПР2
_____Точилашвили М.Г.
Подпись

Преподаватель:

_____Волков К.В.
Подпись

Нижний Новгород

2024

Введение

Разработана информационная система для автоматизации процессов управления заказами, учёта запасов, аналитики продаж, программы лояльности и бронирования столиков в кафе/ресторане. Целью данной системы является повышение эффективности работы заведения, улучшение обслуживания клиентов и снижение рутинных операций для сотрудников.

Концептуальная модель

Информационная система состоит из нескольких взаимосвязанных модулей:

1. **Модуль управления заказами**
    - Создание и редактирование заказов
    - Отправка заказов на кухню и бар
    - Просмотр статусов заказов
    - Закрытие заказов
2. **Модуль учёта запасов**
    - Управление продуктами на складе
    - Уведомления о минимальных остатках
    - История изменений запасов
3. **Модуль аналитики продаж**
    - Анализ продаж за разные периоды
    - Определение популярных блюд и напитков
    - Прогнозирование потребностей в продуктах
4. **Модуль программы лояльности**
    - Регистрация клиентов
    - Начисление и использование бонусов
    - Специальные предложения и акции
5. **Модуль бронирования столиков**
    - Онлайн-бронирование через сайт и мобильное приложение
    - Управление бронями со стороны администратора
    - Уведомления клиентов о подтверждении и отмене брони
6. **Модуль обратной связи**
    - Оставление отзывов клиентами
    - Просмотр и ответ на отзывы администрацией

Функциональные требования

1. Модуль управления заказами

Основные функции:

- **Создание нового заказа**: Официанты должны иметь возможность создавать новые заказы, добавляя в них блюда и напитки из меню.
- **Редактирование заказа**: Возможность изменять состав заказа до момента отправки на кухню (добавить/удалить позиции, изменить количество).
- **Отправка заказа на кухню**: После завершения оформления заказа он должен автоматически отправляться на кухню и бар для начала приготовления.
- **Просмотр статусов заказов**: Официанты и администраторы должны иметь доступ к списку всех активных заказов с возможностью просмотра текущего статуса каждого заказа («в процессе приготовления», «готово к выдаче» и т.п.).
- **Закрытие заказа**: По завершении обслуживания клиента заказ закрывается, и данные передаются в систему учета.

2. Модуль учета запасов

Основные функции:

- **Добавление новых позиций**: Администратор должен иметь возможность добавлять новые продукты на склад.
- **Учет количества**: Система должна вести учет количества каждого продукта на складе и обновлять эти данные при поступлении новых партий и использовании продуктов в приготовлении блюд.
- **Автоматическое уведомление**: При достижении минимального уровня запасов система должна отправлять уведомления ответственным лицам (администраторам, закупщикам).
- **История изменений**: Ведение журнала всех операций с запасами (поступления, списания, перемещения).

3. Модуль аналитики продаж

Основные функции:

- **Анализ продаж**: Предоставление отчетов по продажам за различные временные интервалы (дневной, недельный, месячный).
- **Популярные блюда**: Выявление наиболее популярных блюд и напитков на основании статистики продаж.
- **Пиковые часы загрузки**: Анализ загруженности заведения в разное время суток и дни недели.
- **Доходность**: Расчеты общей доходности заведения с учетом затрат на продукты и услуги.

4. Модуль лояльности

Основные функции:

- **Регистрация клиентов**: Клиенты должны иметь возможность зарегистрироваться в программе лояльности через мобильное приложение или веб-сайт.
- **Начисление бонусов**: За каждую покупку клиент получает определенное количество бонусных баллов, которые затем можно использовать для получения скидок или бесплатных блюд.
- **Использование бонусов**: Клиент должен иметь возможность использовать накопленные бонусы при оплате следующего заказа.
- **История транзакций**: Ведение истории всех транзакций по каждому клиенту, включая начисление и использование бонусов.

5. Модуль бронирования столиков

Основные функции:

- **Онлайн-бронирование**: Клиенты должны иметь возможность забронировать столик через веб-сайт или мобильное приложение, выбрав дату, время и количество гостей.
- **Управление бронями**: Администратор должен иметь возможность просматривать и редактировать список бронирований, подтверждать или отменять их.
- **Уведомления клиентам**: Автоматическая отправка уведомлений клиентам о подтверждении или отмене бронирования.

6. Модуль обратной связи

Основные функции:

- **Оставление отзывов**: Клиенты должны иметь возможность оставлять отзывы о своем визите через QR-код на чеке или через мобильное приложение.
- **Просмотр отзывов**: Администратор должен иметь доступ ко всем оставленным отзывам с возможностью фильтрации по различным критериям (оценка, дата, блюдо и т.п.).
- **Ответ на отзывы**: Возможность отвечать на отзывы клиентов, чтобы показать внимание к их мнению.

Техническая архитектура

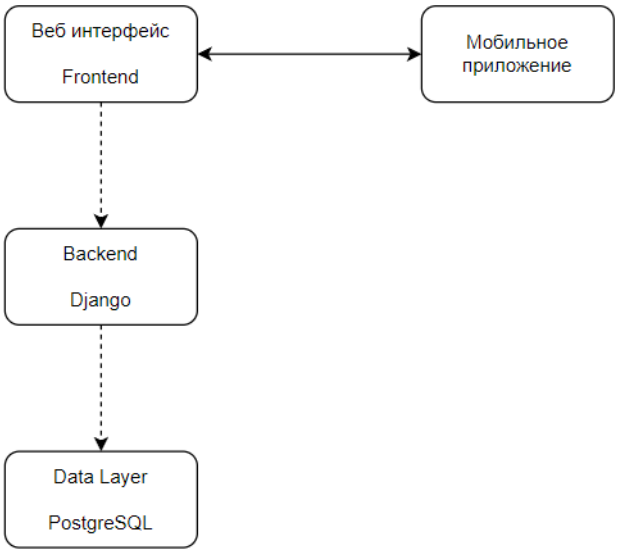

Рис. 1. Пример схемы архитектуры

Система разработана по принципам многослойной архитектуры, состоящей из трёх уровней:

1. **Представительный уровень (Frontend)**
   - Веб-интерфейсы для административной панели, сайта для клиентов и мобильных приложений для официантов и клиентов.
   - Используемые технологии: React.js, Vue.js, React Native, Flutter.
2. **Бизнес-логика (Backend)**
   - RESTful API на основе Django REST Framework или FastAPI.

- Сервисы для управления заказами, учёт запасов, аналитики продаж, программы лояльности, бронирования столиков и обратной связи.

3. **Уровень данных (Data Layer)**
    - Базы данных на основе PostgreSQL для хранения информации о заказах, запасах, клиентах, бонусах, бронированиях и отзывах.
    - Кеширование с использованием Redis для ускорения доступа к часто запрашиваемым данным.

Представительный уровень (Frontend)

Веб-интерфейсы:

- **Административная панель**: Реализована на React.js или Vue.js. Позволяет администраторам управлять заказами, запасами, аналитикой продаж, программой лояльности и бронированием столиков.
- **Интерфейс для официантов**: Мобильное приложение на React Native или Flutter, позволяющее официантам принимать и обрабатывать заказы.
- **Сайт для клиентов**: Разработка на React.js или Vue.js. Включает функционал бронирования столиков, регистрации в программе лояльности и оставления отзывов.

Мобильные приложения:

- **Приложение для клиентов**: Разработано на React Native или Flutter. Позволяет клиентам бронировать столики, участвовать в программе лояльности, оставлять отзывы и получать уведомления о специальных предложениях.
- **Приложение для официантов**: Аналогично мобильному интерфейсу для официантов, но с дополнительной функциональностью, такой как сканирование QR-кодов для подтверждения оплаты.

Бизнес-логика (Backend)

Фреймворк:

- **Django REST Framework** или **FastAPI** для построения RESTful API, которое будет использоваться для взаимодействия между фронтендом и базой данных.

Сервисы:

- **Сервис управления заказами**: Обрабатывает создание, изменение и закрытие заказов, отправку заказов на кухню и бар, а также отслеживание статусов заказов.
- **Сервис учета запасов**: Управляет запасами продуктов, уведомляет об остатках и ведет историю изменений.
- **Сервис аналитики продаж**: Производит расчеты и предоставляет отчеты по продажам, популярным блюдам и пикам нагрузки.
- **Сервис программы лояльности**: Управляет регистрацией клиентов, начислением и использованием бонусов, историей транзакций.
- **Сервис бронирования столиков**: Обеспечивает онлайн-бронирование, управление бронями и отправку уведомлений клиентам.
- **Сервис обратной связи**: Прием и обработка отзывов от клиентов, предоставление доступа администраторам для ответа на отзывы.

Уровень данных (Data Layer)

База данных:

- **PostgreSQL** или другая реляционная база данных для хранения информации о заказах, запасах, клиентах, бонусах, бронированиях и отзывах.

Кеширование:

- **Redis** для ускорения доступа к часто запрашиваемым данным, таким как текущие статусы заказов и остатки продуктов.

Интеграционные компоненты

Асинхронные задачи:

- **Celery** для выполнения длительных задач, таких как отправка уведомлений, генерация отчетов и обновление статусов заказов.

Уведомления:

- **RabbitMQ** или **Kafka** для обмена сообщениями между сервисами и доставки уведомлений пользователям.

Инфраструктура

Контейнеризация:

- **Docker** для упаковки и развертывания приложений и сервисов в контейнерах.

- **Kubernetes** или **Docker Swarm** для оркестрации контейнеров и управления кластером.

CI/CD:

- **GitLab CI/CD** или **Jenkins** для автоматизации сборки, тестирования и развертывания кода.

Безопасность

Аутентификация и авторизация:

- **JWT (JSON Web Tokens)** для аутентификации пользователей и защиты API.
- **OAuth 2.0** для интеграции с внешними сервисами (например, социальными сетями для входа в систему).

Шифрование:

- **SSL/TLS** для защищенного соединения между клиентом и сервером.
- **Хранение паролей в зашифрованном виде** с использованием хэш-функции (например, bcrypt).

Структура базы данных

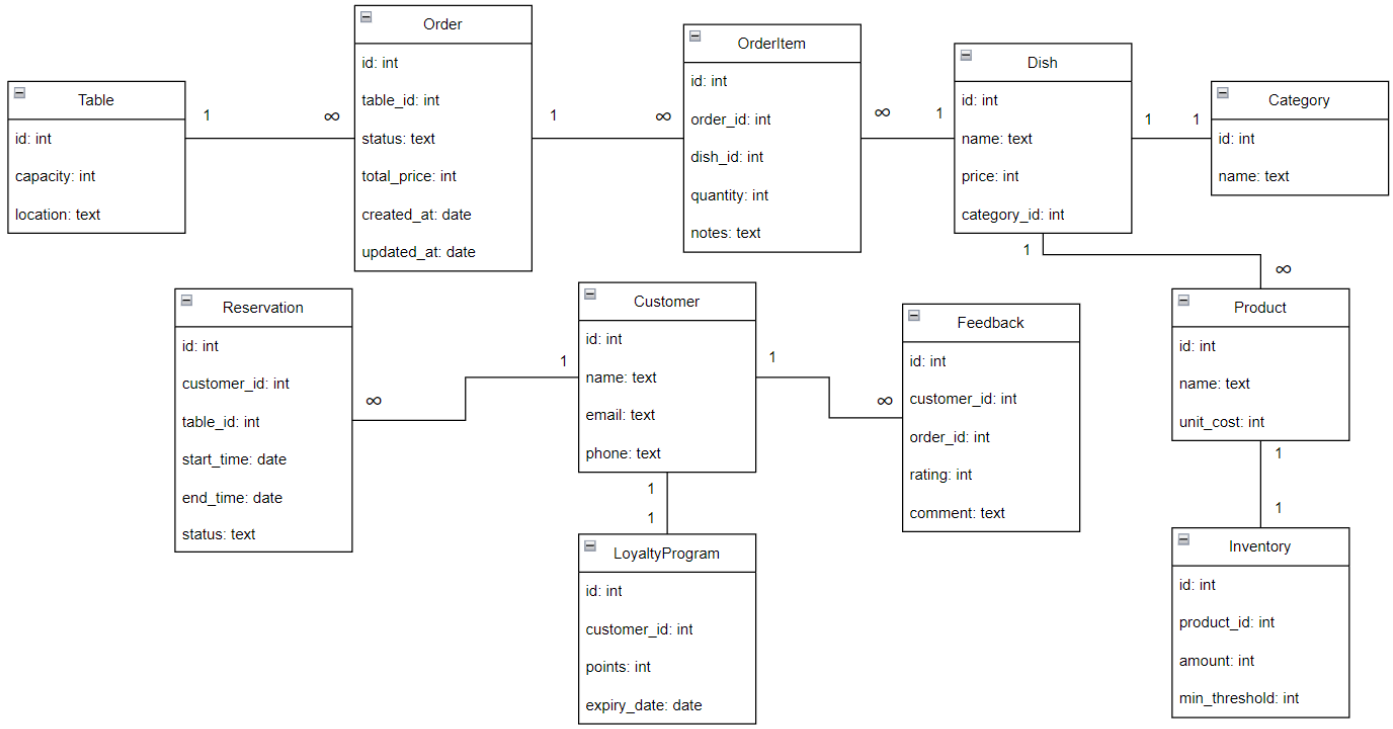

Рис. 2. Структура базы данных

Таблицы и связи между ними:

1. Table (Стол)

- **id**: уникальный идентификатор стола.
- **capacity**: максимальная вместимость стола.
- **location**: местоположение стола в зале.

Связи:

- **Order (Заказ)**: Один стол может содержать множество заказов.

2. Order (Заказ)

- **id**: уникальный идентификатор заказа.
- **table_id**: внешний ключ, ссылающийся на Table.
- **status**: текущий статус заказа (pending, in_progress, ready, delivered).
- **total_price**: общая стоимость заказа.

- **created_at**: время создания заказа.
- **updated_at**: время последнего обновления заказа.

Связи:

- **OrderItem (Позиция в заказе)**: Один заказ может содержать множество позиций.
- **Table**: Один заказ всегда относится к одному столу.

3. OrderItem (Позиция в заказе)

- **id**: уникальный идентификатор позиции в заказе.
- **order_id**: внешний ключ, ссылающийся на Order.
- **dish_id**: внешний ключ, ссылающийся на Dish.
- **quantity**: количество данного блюда в заказе.
- **notes**: любые специальные инструкции или комментарии к позиции.

Связи:

- **Dish (Блюдо)**: Одна позиция в заказе всегда связана с одним блюдом.
- **Order**: Одна позиция в заказе всегда принадлежит одному заказу.

4. Dish (Блюдо)

- **id**: уникальный идентификатор блюда.
- **name**: название блюда.
- **price**: цена блюда.
- **category_id**: внешний ключ, ссылающийся на Category (например, закуски, основные блюда, десерты).

Связи:

- **OrderItem**: Одно блюдо может быть включено во множество позиций в заказах.
- **Category**: Одно блюдо всегда принадлежит одной категории.

5. Category (Категория блюд)

- **id**: уникальный идентификатор категории.
- **name**: название категории.

Связи:

- **Dish**: Одно блюдо всегда принадлежит одной категории.

6. Customer (Клиент)

- **id**: уникальный идентификатор клиента.
- **name**: имя клиента.
- **email**: электронная почта клиента.
- **phone**: контактный телефон клиента.

Связи:

- **Reservation (Бронь)**: Один клиент может сделать множество броней.
- **Feedback (Отзыв)**: Один клиент может оставить множество отзывов.
- **LoyaltyProgram (Программа лояльности)**: Один клиент может участвовать в одной программе лояльности.

7. Reservation (Бронь)

- **id**: уникальный идентификатор брони.
- **customer_id**: внешний ключ, ссылающийся на Customer.
- **table_id**: внешний ключ, ссылающийся на Table.
- **start_time**: запланированное время начала брони.
- **end_time**: запланированное время окончания брони.
- **status**: статус брони (confirmed, cancelled).

Связи:

- **Customer**: Одна бронь всегда относится к одному клиенту.
- **Table**: Одна бронь всегда относится к одному столу.

8. Inventory (Запасы)

- **id**: уникальный идентификатор записи о запасах.
- **product_id**: внешний ключ, ссылающийся на продукт.
- **amount**: текущее количество продукта на складе.
- **min_threshold**: минимальный порог запаса, при достижении которого будет отправлено уведомление.

Связи:

- **Product**: Одна запись о запасах всегда относится к одному продукту.

9. Feedback (Отзыв)

- **id**: уникальный идентификатор отзыва.
- **customer_id**: внешний ключ, ссылающийся на Customer.
- **order_id**: внешний ключ, ссылающийся на Order.
- **rating**: рейтинг, данному клиентом.
- **comment**: текст отзыва.

Связи:

- **Customer**: Один отзыв всегда относится к одному клиенту.
- **Order**: Один отзыв всегда относится к одному заказу.

10. LoyaltyProgram (Программа лояльности)

- **id**: уникальный идентификатор участника программы.
- **customer_id**: внешний ключ, ссылающийся на Customer.
- **points**: накопленные баллы.
- **expiry_date**: дата истечения срока действия баллов.

Связи:

- **Customer**: Один участник программы лояльности всегда относится к одному клиенту.

Функциональные возможности модуля управления заказами

1. **Создание нового заказа**:
    - Официант вводит номер стола и выбирает блюда и напитки из меню.
    - Добавляются комментарии к заказу (например, особые пожелания клиента).
2. **Редактирование заказа**:
    - Изменение состава заказа до момента отправки на кухню (добавление/удаление позиций, изменение количества).
3. **Отправка заказа на кухню**:

- После завершения оформления заказа он автоматически отправляется на кухню и бар для начала приготовления.

4. **Просмотр статусов заказов**:
   - Официанты и администраторы имеют доступ к списку всех активных заказов с возможностью просмотра текущего статуса каждого заказа («в процессе приготовления», «готово к выдаче» и т.п.).
5. **Закрытие заказа**:
   - По завершении обслуживания клиента заказ закрывается, и данные передаются в систему учета.

Внутреннее устройство модуля

Структура данных

Модуль управления заказами использует несколько моделей для хранения информации о заказах, позициях в заказе и статусе заказа.

Модель Order

```
class Order(models.Model):
    STATUS_CHOICES = (
        ('pending', 'Ожидает обработки'),
        ('in_progress', 'Готовится'),
        ('ready', 'Готово'),
        ('delivered', 'Отдан клиенту'),
    )

    table_number = models.IntegerField()
    total_price = models.DecimalField(max_digits=10, decimal_places=2)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Модель OrderItem

```
class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE, related_name='items')
```

```
    dish = models.ForeignKey(Dish, on_delete=models.PROTECT)
    quantity = models.PositiveIntegerField(default=1)
    notes = models.TextField(blank=True)
```

Модель Dish (в качестве примера)

```
class Dish(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    category = models.ForeignKey(Category, on_delete=models.PROTECT)
```

Логика работы

1. **Создание заказа**:
    - Официант создает новый объект Order, указывая номер стола.
    - Затем добавляет объекты OrderItem, выбирая блюда из меню и указывая количество.
2. **Редактирование заказа**:
    - Официант может изменять состав заказа, добавляя или удаляя объекты OrderItem.
    - Также возможно изменение количества уже добавленных позиций.
3. **Отправка заказа на кухню**:
    - Когда заказ завершен, официант нажимает кнопку "Отправить". Статус заказа меняется на 'in_progress'.
    - Информация о заказе передается на кухню через API или внутренний мессенджер (например, RabbitMQ).
4. **Обновление статуса заказа**:
    - Повар или бармен изменяют статус заказа на 'ready', когда блюдо готово.
    - Официант видит изменения статуса в своей панели и может забрать готовое блюдо.
5. **Закрытие заказа**:
    - После того как клиент оплатил заказ, официант меняет статус заказа на 'delivered'.

- Данные о закрытом заказе сохраняются в базе данных для дальнейшего анализа и отчетности.

Примеры методов и бизнес-логики

Метод создания заказа

```
def create_order(table_number, items):
    # Создаем новый заказ
    order = Order.objects.create(table_number=table_number)

    # Добавляем позиции в заказ
    for item in items:
        OrderItem.objects.create(order=order,        dish=item['dish'],
quantity=item['quantity'])

    return order
```

Метод обновления статуса заказа

```
def update_order_status(order_id, new_status):
    try:
        order = Order.objects.get(id=order_id)
        order.status = new_status
        order.save()
        return True
    except Order.DoesNotExist:
        return False
```

Прототипирование и тестирование

Был создан скелет системы, включающий базовые модели и методы для работы с заказами, запасами, клиентами и другими сущностями. Написаны юнит-тесты для проверки корректности работы этих методов, а также проведены интеграционные тесты для проверки взаимодействия различных компонентов системы.

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Установите последнюю версию PowerShell для новых функций и улучшения! https://aka.ms/PSWindows

PS C:\Programs\UnitApp\restaurant_system> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 09, 2025 - 13:42:53
Django version 5.1.4, using settings 'restaurant_system.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[09/Jan/2025 13:43:00] "GET /api/ HTTP/1.1" 200 10460
[09/Jan/2025 13:43:00] "GET /static/rest_framework/css/bootstrap.min.css HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/css/bootstrap-tweaks.css HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/css/prettify.css HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/css/default.css HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/js/jquery-3.7.1.min.js HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/js/ajax-form.js HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/js/csrf.js HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/js/bootstrap.min.js HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/js/prettify-min.js HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/js/default.js HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/img/grid.png HTTP/1.1" 304 0
[09/Jan/2025 13:43:00] "GET /static/rest_framework/js/load-ajax-form.js HTTP/1.1" 304 0
```

Рис. 2. Запуск сервера

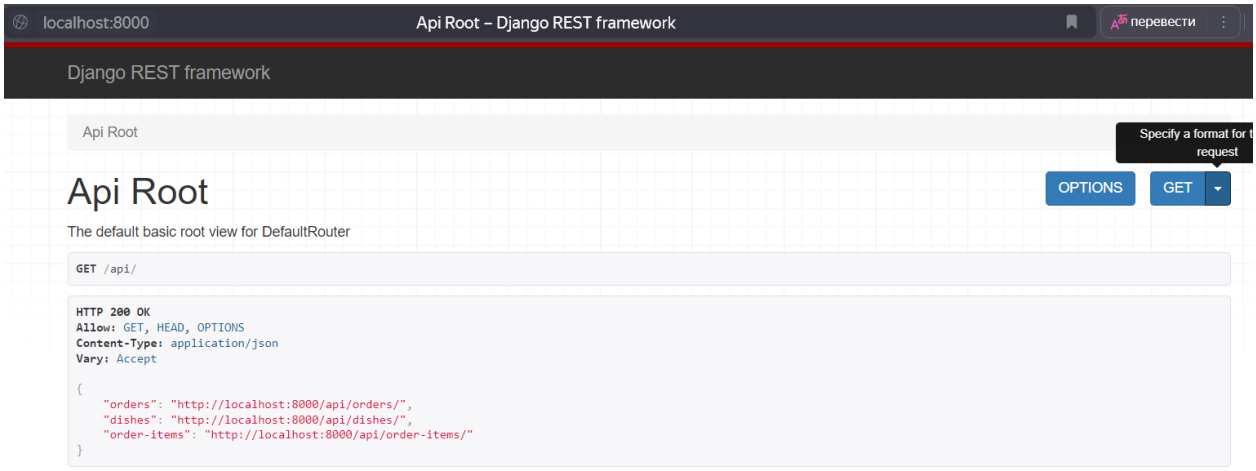

Рис. 3. Пример работы сервера на Django

Тестирование

1. Тест создания заказа (test_order_creation):

Этот тест проверяет базовый сценарий создания нового заказа. После создания заказа и добавления к нему двух блюд, тест вычисляет общую стоимость заказа и проверяет, что она рассчитана правильно. Также проверяется корректное сохранение статуса заказа по умолчанию и общее количество позиций в заказе.

2. Тест изменения статуса заказа (test_order_status_change):

Тест оценивает возможность изменения статуса заказа. Создается новый заказ, после чего его статус изменяется на "в процессе". Тест проверяет, что статус был успешно обновлен и сохранен в базе данных.

3. Тест расчета общей стоимости (test_total_price_calculation):

Этот тест проверяет корректность вычисления общей стоимости заказа. Создается заказ с несколькими позициями, после чего вычисляется общая стоимость. Тест подтверждает, что итоговая сумма соответствует ожидаемому значению, основанному на ценах и количествах блюд.

4. Тест создания блюда (test_dish_creation):

Тест фокусируется на создании нового блюда в системе. Проверяется, что имя и цена блюда сохраняются правильно и соответствуют введенным значениям. Это гарантирует, что модель блюда работает корректно.

5. Тест обновления количества позиции в заказе (test_order_item_quantity_update):

Этот тест оценивает возможность изменения количества конкретного блюда в заказе. После обновления количества тест проверяет, что новое количество было правильно сохранено. Это важно для проверки корректности обновлений в базе данных.

6. Тест удаления позиции из заказа и пересчета общей стоимости (test_order_item_deletion):

Тест проверяет, что удаление позиции из заказа приводит к корректному пересчету общей стоимости. Создаются несколько позиций, одна из них удаляется, и тест подтверждает, что общая стоимость заказа обновляется соответственно.

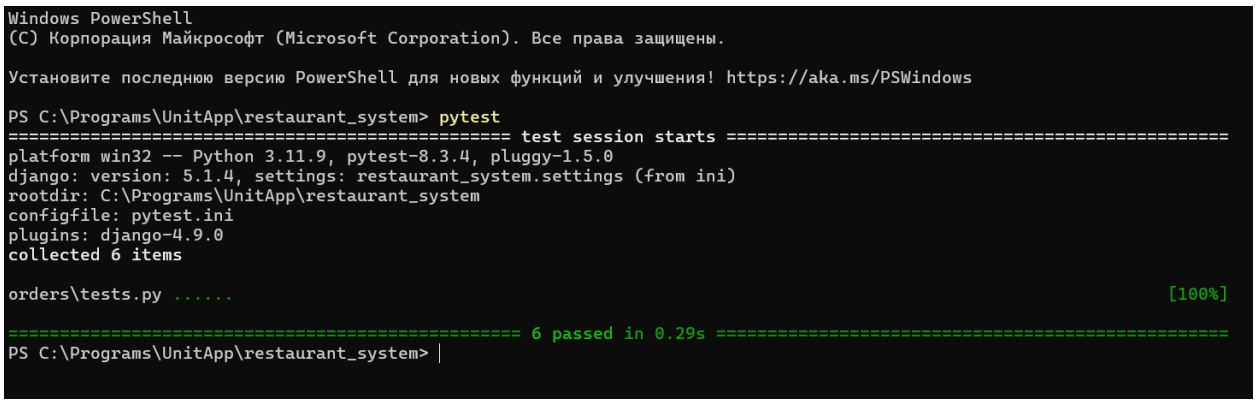

Рис. 4. Результат работы тестов

Заключение

Разработанная информационная система удовлетворяет требованиям к автоматизации процессов управления заказами, учёта запасов, аналитики продаж, программы лояльности и бронирования столиков в кафе/ресторане. Проведённые тесты подтвердили работоспособность и надёжность системы. Дальнейшее развитие системы может включать внедрение дополнительных функций, таких как интеграция с системами бухгалтерского учёта и управления персоналом.

Приложения

Ссылка на GitHub с проектом ИС

https://github.com/mTochilashvili/New-IS.git

Исходный код модуля и тестов.

Файл models.py:

```
from django.db import models


class Dish(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.name


class Order(models.Model):
    STATUS_CHOICES = (
        ('pending', 'Ожидает обработки'),
        ('in_progress', 'Готовится'),
        ('ready', 'Готово'),
        ('delivered', 'Отдан клиенту'),
    )

    table_number = models.IntegerField()
    total_price = models.DecimalField(max_digits=10, decimal_places=2,
blank=True, null=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default='pending')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def save(self, *args, **kwargs):
        if not self.pk:
            self.calculate_total_price()
        super().save(*args, **kwargs)

    def calculate_total_price(self):
       if self.pk:  # Проверяем, что объект уже сохранен
           total = sum(item.quantity * item.dish.price for item in
self.items.all())
           self.total_price = total



    def __str__(self):
        return f'Заказ №{self.id}'
```

```
class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE,
related_name='items')
    dish = models.ForeignKey(Dish, on_delete=models.PROTECT)
    quantity = models.PositiveIntegerField(default=1)
    notes = models.TextField(blank=True)

    def __str__(self):
        return f'{self.quantity}x {self.dish.name}'
```

Файл views.py:

```
from django.shortcuts import render

from rest_framework import viewsets
from rest_framework.response import Response
from rest_framework.decorators import action
from .models import Order, OrderItem, Dish
from .serializers import OrderSerializer, OrderItemSerializer, DishSerializer


class DishViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Dish.objects.all()
    serializer_class = DishSerializer


class OrderViewSet(viewsets.ModelViewSet):
    queryset = Order.objects.all()
    serializer_class = OrderSerializer

    @action(detail=True, methods=['patch'])
    def change_status(self, request, pk=None):
        order = self.get_object()
        new_status = request.data.get('new_status')
        if new_status in dict(Order.STATUS_CHOICES).keys():
            order.status = new_status
            order.save()
            return Response({'message': 'Статус заказа успешно изменен'})
        else:
            return Response({'error': 'Неверный статус'}, status=400)


class OrderItemViewSet(viewsets.ModelViewSet):
    queryset = OrderItem.objects.all()
    serializer_class = OrderItemSerializer
```

Файл serializers.py:

```
from rest_framework import serializers
from .models import Order, OrderItem, Dish


class DishSerializer(serializers.ModelSerializer):
    class Meta:
        model = Dish
        fields = ['id', 'name', 'price']


class OrderItemSerializer(serializers.ModelSerializer):
    class Meta:
        model = OrderItem
        fields = ['id', 'dish', 'quantity', 'notes']


class OrderSerializer(serializers.ModelSerializer):
    items = OrderItemSerializer(many=True)

    class Meta:
        model = Order
        fields = ['id', 'table_number', 'total_price', 'status', 'created_at',
'updated_at', 'items']

    def create(self, validated_data):
        items_data = validated_data.pop('items')
        order = Order.objects.create(**validated_data)
        for item_data in items_data:
            OrderItem.objects.create(order=order, **item_data)
        return order

    def update(self, instance, validated_data):
        items_data = validated_data.pop('items')
        instance.table_number = validated_data.get('table_number',
instance.table_number)
        instance.status = validated_data.get('status', instance.status)
        instance.save()

        keep_items = []
        for item_data in items_data:
            if "id" in item_data:
                item = OrderItem.objects.get(id=item_data["id"])
                item.dish = item_data.get("dish", item.dish)
                item.quantity = item_data.get("quantity", item.quantity)
                item.notes = item_data.get("notes", item.notes)
                item.save()
                keep_items.append(item.id)
            else:
                OrderItem.objects.create(order=instance, **item_data)
```

```
        for item in instance.items.all():
            if item.id not in keep_items:
                item.delete()

        return instance
```

Файл tests.py:

```
from django.test import TestCase
from .models import Order, OrderItem, Dish

class OrderModelTests(TestCase):

    def setUp(self):
        self.dish1 = Dish.objects.create(name='Dish 1', price=10.00)
        self.dish2 = Dish.objects.create(name='Dish 2', price=15.00)

    def test_order_creation(self):
        order = Order.objects.create(table_number=1)
        OrderItem.objects.create(order=order, dish=self.dish1, quantity=2)
        OrderItem.objects.create(order=order, dish=self.dish2, quantity=1)
        order.calculate_total_price()
        order.save()

        self.assertEqual(order.total_price, 35.00)
        self.assertEqual(order.status, 'pending')
        self.assertEqual(order.items.count(), 2)

    def test_order_status_change(self):
        order = Order.objects.create(table_number=1)
        order.status = 'in_progress'
        order.save()

        self.assertEqual(order.status, 'in_progress')

    def test_total_price_calculation(self):
        order = Order.objects.create(table_number=1)
        OrderItem.objects.create(order=order, dish=self.dish1, quantity=3)
        OrderItem.objects.create(order=order, dish=self.dish2, quantity=2)
        order.calculate_total_price()
        order.save()

        self.assertEqual(order.total_price, 60.00)

    def test_dish_creation(self):
       dish = Dish.objects.create(name='Dish 3', price=20.00)
       self.assertEqual(dish.name, 'Dish 3')
       self.assertEqual(dish.price, 20.00)
```

```
    def test_order_item_quantity_update(self):
        order = Order.objects.create(table_number=1)
        item = OrderItem.objects.create(order=order, dish=self.dish1, quantity=1)
        item.quantity = 5
        item.save()
        self.assertEqual(item.quantity, 5)

    def test_order_item_deletion(self):
        order = Order.objects.create(table_number=1)
        item1 = OrderItem.objects.create(order=order, dish=self.dish1, quantity=2)
        item2 = OrderItem.objects.create(order=order, dish=self.dish2, quantity=1)
        order.calculate_total_price()
        self.assertEqual(order.total_price, 35.00)

        item1.delete()
        order.calculate_total_price()
        order.save()
        self.assertEqual(order.total_price, 15.00)
```