МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №42

КУРСОВАЯ РАБОТА
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

старший преподаватель		С.Ю. Гуков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

Магазин товаров или услуг

по дисциплине: ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4329		Д.С. Шаповалова
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В эпоху бурного развития цифровых технологий, когда каждый день приносит новые вызовы, перед нами встаёт важнейшая задача — создание мощного, надёжного и удобного программного обеспечения.

Цель нашей работы ясна и решительна: разработать программный продукт среднего уровня сложности, который не просто будет выполнять рутинные функции, а станет настоящим оружием в борьбе с неэффективностью, хаосом устаревших систем и отсталыми методами управления.

Наш проект должен быть построен на качественной, продуманной архитектуре — как крепкий фундамент будущего информационных технологий. Мы применим комплексную обработку исключений, чтобы ни одна ошибка не смогла подорвать стабильность системы. Мы обеспечим расширяемость, чтобы наш продукт мог расти и развиваться вместе с потребностями пользователей. И, конечно, мы используем современные методы проектирования, потому что только передовые технологии достойны нового, прогрессивного общества.

Эта работа — не просто учебное задание. Это шаг вперёд в закреплении наших знаний, в овладении передовыми технологиями программирования, в подготовке кадров, способных строить цифровое будущее. Наше программное обеспечение должно стать надёжным инструментом для пользователей, реализуя процессы купли-продажи, устраняя бюрократические проволочки и давая клиентам мощные инструменты для сбыта товара.

Курсовая работа позволяет:

1. систематизировать теоретические знания и закрепить полученные практические умения по дисциплине «Технологии программирования» в соответствии с требованиями к уровню подготовки по направлению 09.03.02 «Информационные системы и технологии»;
2. сформировать умения работы с учебной литературой и иными информационными источниками;
3. развить профессиональную письменную и устную речь;
4. развить системное мышление, творческую инициативу, самостоятельность, организованность и ответственность за принимаемые решения;
5. сформировать навыки планомерной регулярной работы над решением поставленных задач.

1. Индивидуальное задание

Описание задания: необходимо разработать программу, которая имитирует покупку товаров или услуг в магазине.

Покупатель, у которого есть бонусная карта, кошелек и список покупок, приходит в магазин, в котором он может приобрести какие-то товары или услуги. Покупатель может положить товар в корзину, выложить товар, совершить покупку. Также покупатель перед тем, как положить в корзину некоторые товары, сначала должен их взвесить. В случае магазина услуг, а не товаров, придумать похожую аналогию. Покупатель может оплатить покупку 1 наличными средствами, дебетовой картой либо бонусами. Он может оплатить как полностью весь чек одним из методов, так и по частям разными способами. Необходимо также предусмотреть ситуации, при которых у покупателя не хватает средств для совершения покупки либо некоторые товары не взвешены. Если средств не хватает для совершения покупки, то покупатель может выкладывать товары из корзины, пока денег либо бонусов не хватит для совершения покупки. То есть успешный сценарий, при котором покупатель должен купить хоть что-то, должен обязательно быть – он не может прийти в магазин полностью без средств. Сохранение сессии (информации о данных покупателя и оставшемся наличии товаров/услуг) при перезапуске программы не обязательно, но приветствуется.

Также необходимо провести модульное тестирование нескольких функций. Выбрать достаточное тестовое покрытие.

Проект может быть выполнен либо в качестве консольного приложения (тогда обязателен командно-текстовый интерфейс), либо иметь графический пользовательский интерфейс (User Interface, UI), а также может быть написан на любом языке программирования.

Требования к структуре проекта:

1) Применение принципов ООП (наследования, инкапсуляции, полиморфизма, абстракции) и SOLID
2) Дружелюбный командно-текстовый либо графический пользовательский интерфейс
3) Использование шаблонов архитектуры системы (любой)
4) Использование паттернов проектирования (минимум два)

5) Проведение модульного тестирования (минимум три функции) с достаточным тестовым покрытием
6) Товары и услуги должны подгружаться из внешнего хранилища (файл, база данных и т.д.)
7) Объем отчета не менее 17 страниц без учета приложения (в приложении весь исходный код)

2. Краткое описание хода разработки и назначение используемых технологий

1. Постановка задачи: создать модель магазина с логикой оплаты.
2. Выбор архитектуры: применён шаблон проектирования MVC.
3. Разработка модели: реализация товаров, корзины, способов оплаты.
4. Реализация логики оплаты: учёт переплат, недоплат, возврат сдачи.
5. Создание пользовательского интерфейса: приятный одностраничный веб-интерфейс.
6. Проведение модульного тестирования: с использованием unittest.
7. Отладка и исправление ошибок по результатам тестов.
8. Подготовка документации и итоговой версии проекта.

Разработка программного обеспечения началась с анализа задачи и постановки цели — создать интерактивную модель магазина, в которой пользователь мог бы выбрать товар, оплатить его различными способами и получить подтверждение о совершённой покупке. Эта цель подразумевала реализацию базовой бизнес-логики, близкой к повседневным операциям в сфере розничной торговли.

Ключевым техническим решением стало применение архитектурного шаблона MVC (Model-View-Controller), который позволил структурировать код и разграничить зоны ответственности:

- **Модель** (Model) реализует внутреннюю логику — товары, корзину, историю покупок.
- **Представление** (View) отвечает за отображение информации пользователю.
- **Контроллер** (Controller) управляет логикой взаимодействия между моделью и представлением.

В качестве языка программирования выбран Python, поскольку он является современным, мощным и при этом простым для создания прототипов.

В качестве фреймворка для пользовательского интерфейса был использован Flask – удобный в использовании, по требованию расширяемый плагинами, веб-фреймворк. Он предоставляет базовые инструменты для создания веб-приложений, таких как маршрутизация URL, обработка запросов и рендеринг HTML-страниц, но не включает в себя готовые решения для таких задач, как проверка форм, работа с базами данных или

аутентификация. Вместо этого Flask позволяет разработчикам использовать расширения и библиотеки для расширения функциональности.

Язык разметки HTML, используемый для создания и структурирования веб-страниц и контента. Он определяет структуру веб-страницы, указывая браузеру, какие элементы отображать и как они должны быть размещены.

А чтобы всё предстало пред пользователем в прилежном виде – использовался CSS – язык для описания внешнего вида веб-страниц, в основном для оформления HTML-документов. Он отвечает за все визуальные аспекты страницы, такие как цвета, шрифты, расположение элементов и многое другое.

После завершения основной разработки была проведена серия модульных тестов, с использованием модуля - unittest, входящего в стандартную библиотеку Python.

3. Пользовательская документация

Первое и последнее, что предстаёт пред нашими глазами – главная и единственная веб-страница онлайн-магазина.

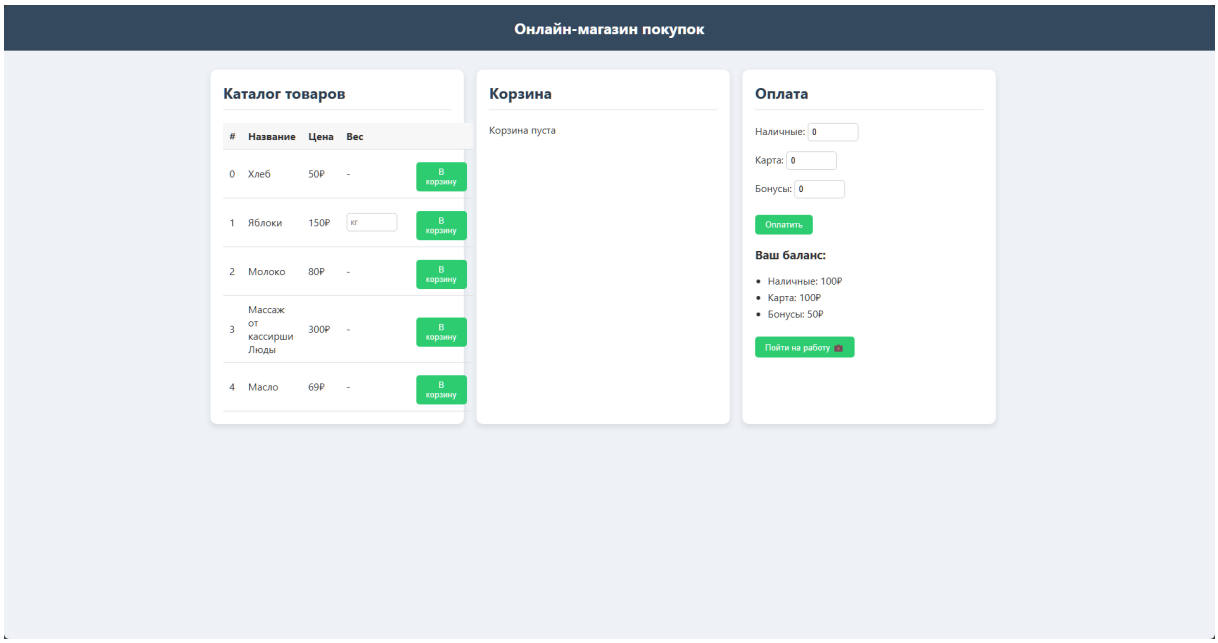

Рисунок 1.1 – главная страница

Страница разделена на 4, но на первый взгляд 3 блока: каталог товаров, корзина, денежные операции. 4 блок представляет собой историю покупок, внизу страницы, изначально не виден.

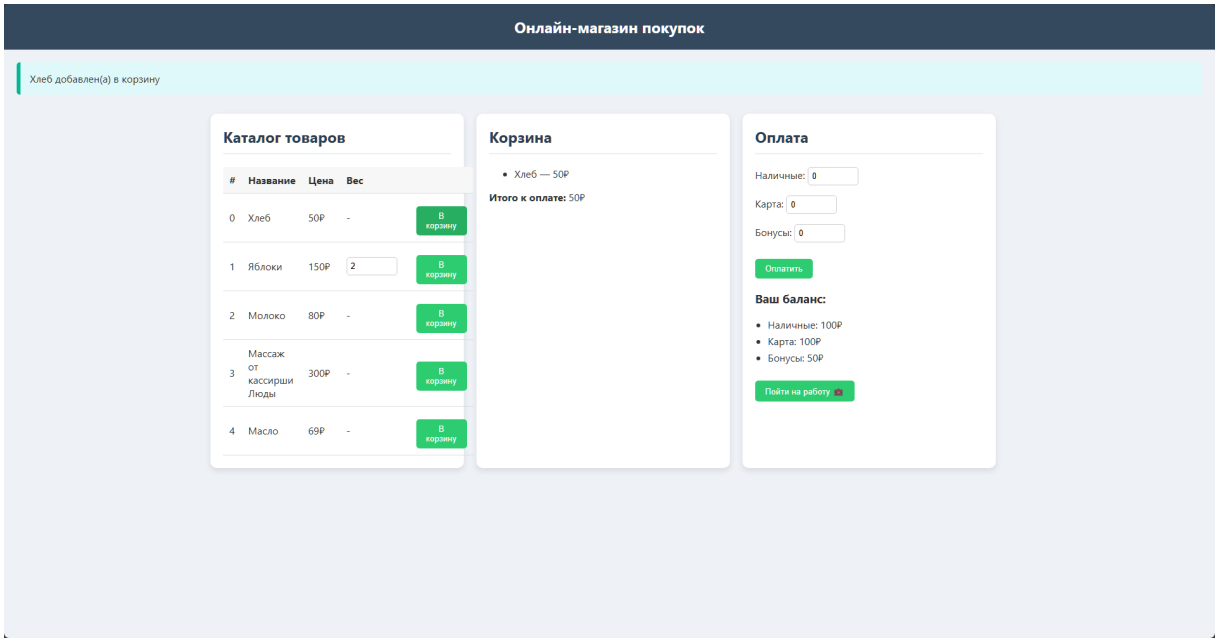

Рисунок 1.2 – Добавление товара в корзину

На штучных товарах поле «вес» недоступно, а на продуктах, которые продаются на развес, необходимо указать вес в килограммах.

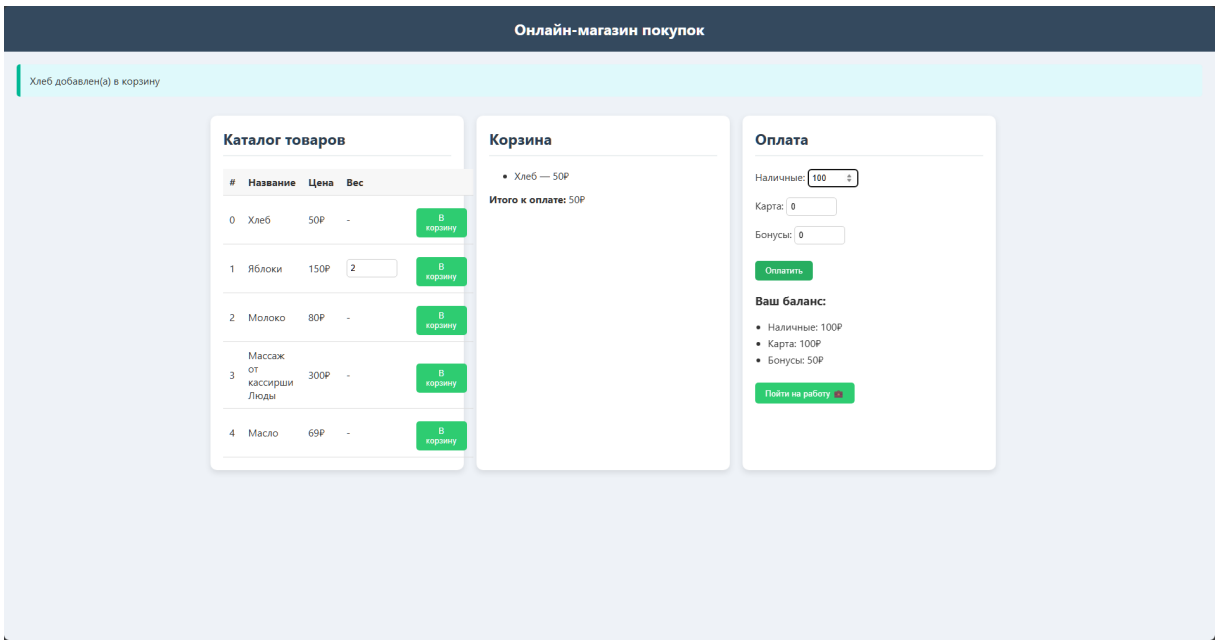

Рисунок 1.3 – Оплата товара

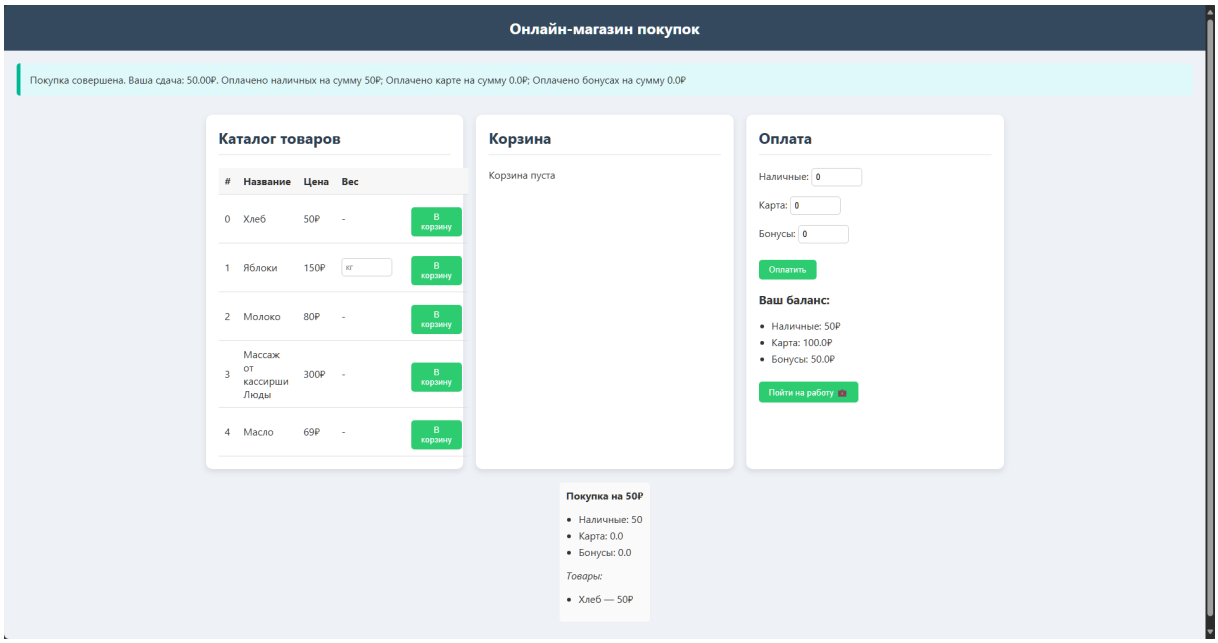

Рисунок 1.4 – Товар оплачен, дали сдачу

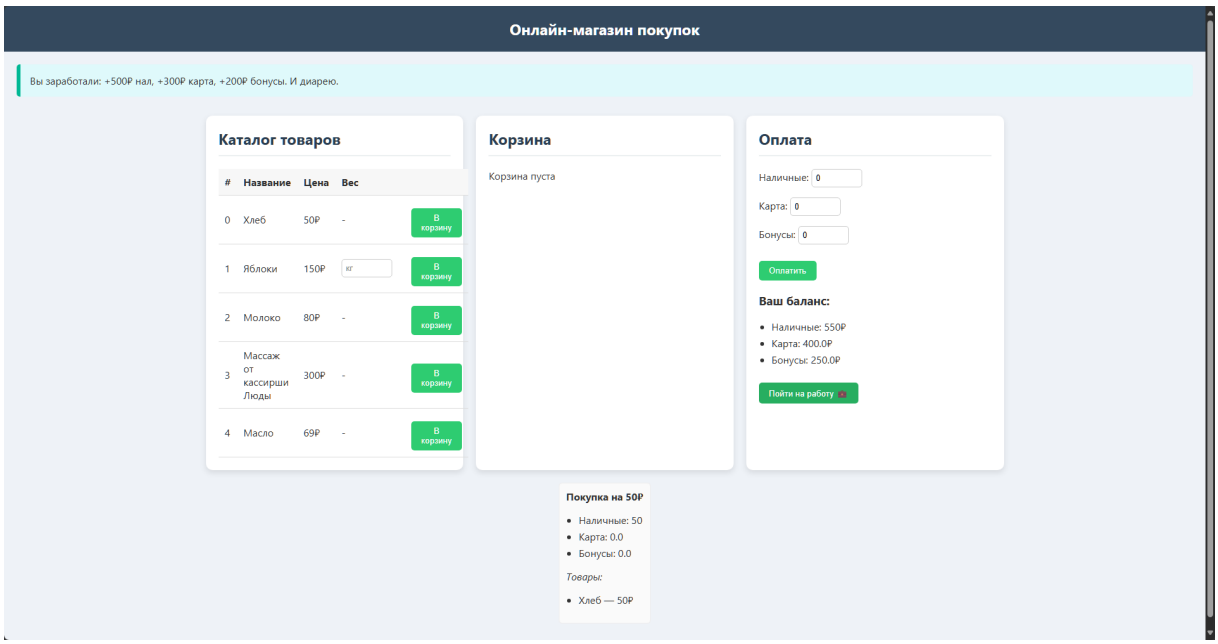

Рисунок 1.5 – Зарабатываем деньги

Есть возможность ходить на работу, чтобы было на что ходить в онлайн-магазин. Также учтена система чеков.

4. Техническая документация

В **models.py** реализуются два основных класса: Product и Customer. Это у нас файл, отвечающий за главную бизнес-логику. Классы играют ключевую роль в моделировании предметной области системы, связанной с электронной коммерцией или торговлей. Класс Product представляет отдельный товар, который можно приобрести, а класс Customer моделирует покупателя с его состоянием, корзиной и возможностями оплаты.

Класс Product предназначен для описания товара, который может быть либо простым, с фиксированной ценой, либо весовым, цена которого рассчитывается в зависимости от веса.

Конструктор класса принимает четыре параметра:

- id — уникальный идентификатор товара, используется для однозначного различия товаров в системе;
- name — наименование товара, строка, описывающая название продукта;
- price — базовая цена товара за единицу, представлена числом (чаще всего целым или с плавающей точкой);
- requires_weight — булевский флаг, указывающий, является ли товар весовым. Если True, цена будет рассчитываться исходя из веса.

Внутри объекта товара сохраняется исходная цена как base_price и значение веса, изначально равное None. Это значит, что если товар весовой, то до установки веса его цена не определена.

Для весовых товаров предусмотрен метод set_weight(weight), который устанавливает значение веса, преобразуя переданный аргумент в число с плавающей точкой. Это позволяет более гибко учитывать массу при расчете стоимости.

Для получения итоговой стоимости товара реализовано свойство price. В зависимости от того, весовой товар или нет, оно возвращает либо базовую цену (base_price), либо произведение базовой цены на установленный вес. Если вес не задан у весового товара, возвращается базовая цена, что можно считать ограничением на корректность использования.

Метод is_ready() проверяет, можно ли считать товар готовым к покупке. Для обычных товаров он всегда возвращает True, так как вес не требуется. Для весовых — проверяет, что вес установлен (не None), и возвращает результат проверки. Это обеспечивает контроль над полнотой данных перед продажей.

Класс Customer моделирует покупателя, который взаимодействует с системой: формирует корзину, оплачивает товары, хранит историю покупок и управляет собственными средствами.

При инициализации объекта создаются несколько полей:

- cash — сумма наличных средств покупателя;
- card — баланс, доступный на банковской карте покупателя;
- bonus — количество бонусных средств, которые могут быть использованы для оплаты;
- cart — список товаров, которые покупатель положил в корзину, но ещё не оплатил;
- purchase_history — список совершённых покупок, каждая запись которого содержит информацию о товарах, сумме и способах оплаты.

Метод total_cart() служит для подсчёта общей стоимости всех товаров, находящихся в корзине. Для этого он суммирует итоговые цены каждого товара в списке cart. Используется встроенная функция sum с генератором цен.

Оплата товаров реализована методом pay(amount, cash, card, bonus). Входными параметрами выступают:

- amount — общая сумма оплаты;
- cash, card, bonus — разбивка оплаты по источникам (наличные, карта, бонусы).

Сначала метод проверяет, совпадает ли сумма оплат с общей суммой, переданной параметром amount. Если сумма средств из трёх источников не равна amount, то возвращается ошибка о несовпадении сумм.

Далее проверяется достаточность средств в каждом источнике. Если пользователь пытается оплатить наличными больше, чем есть, или аналогично по карте и бонусам — возвращается ошибка о недостатке средств.

Если все проверки проходят, происходит списание соответствующих сумм с каждого баланса.

В историю покупок добавляется запись с деталями: список купленных товаров (имя и цена), общая сумма и распределение оплаты по источникам.

После успешной оплаты корзина очищается.

Метод go_to_work() симулирует игровой или служебный механизм, при котором покупатель пополняет свои балансы. Балансы наличных, карты и бонусов увеличиваются на фиксированные величины: 500, 300 и 200 соответственно. Метод возвращает положительный результат и сообщение о пополнении.

Файл models.py реализует фундаментальную бизнес-логику, связанную с товарами и покупателями. Класс Product обеспечивает хранение и вычисление стоимости товаров, учитывая особенности весовых единиц. Класс Customer управляет финансовыми ресурсами покупателя, корзиной и процессом оплаты с возможностью частичной оплаты разными способами.

Такое разделение обеспечивает чистоту архитектуры, позволяя в дальнейшем масштабировать систему, добавлять новые типы товаров, способы оплаты, а также интегрировать пользовательский интерфейс и контроллеры без вмешательства в логику моделей.

Controllers.py – реализует **контроллер** — центральную часть паттерна MVC. Контроллер управляет взаимодействием между пользователем и моделью: загрузкой товаров, корзиной, оплатой и пополнением баланса.

Загрузка товаров: Функция load_products_from_json() читает JSON-файл и создаёт список объектов Product, каждый из которых может быть обычным или весовым товаром.

Класс Controller

Конструктор __init__:

- Загружает товары.
- Создаёт покупателя (Customer).
- Создаёт три прокси для оплаты: наличные, карта, бонусы (через PaymentProxy), чтобы контролировать списание средств.

Методы:

- get_products() — возвращает все товары.
- get_cart() — возвращает содержимое корзины.
- add_to_cart(index, weight) — добавляет товар в корзину. Если товар весовой, требует указания веса. Создаёт копию товара с заданным весом.
- pay_with_proxies(amounts) — производит оплату:
    - Сравнивает сумму с общей стоимостью корзины.
    - Списывает средства через прокси по приоритету.
    - Сохраняет покупку в истории, очищает корзину.
    - Возвращает сообщение и сдачу, если есть.
- go_to_work() — игровая механика: прибавляет к балансу 500₽ нал, 300₽ карта, 200₽ бонусов.

Используемые паттерны:

Proxy — для контроля доступа к источникам оплаты. Паттерн предоставляет объект-заместитель, который контролирует доступ к какому-либо другому объекту.

В нашем случае прокси применяется для управления оплатой с разных источников: наличные, карта и бонусы. Вместо того чтобы напрямую изменять значения полей cash, card и bonus в классе Customer, оплата производится через экземпляры класса PaymentProxy. Реализован отдельным файлом payment_proxy.py

Каждому источнику оплаты соответствует свой PaymentProxy. Он:

- получает текущий баланс (через переданную функцию),

- проверяет, достаточно ли средств для оплаты,
- списывает нужную сумму при успехе.

Таким образом, Controller не зависит от конкретной реализации логики работы с деньгами: она инкапсулирована внутри прокси. Это упрощает масштабирование: например, добавление нового источника оплаты (например, "QR-код") не потребует переписывания всей логики оплаты.

Command – отдельные команды вызывают методы контроллера. Инкапсулирует запрос как объект, позволяя параметризовать методы объектами команд, откладывать выполнение и поддерживать отмену операций. Часто используется в системах с GUI, меню, кнопками и т.п.

В нашем случае команды реализуются в виде отдельных классов, производных от абстрактного базового класса Command. Каждый такой класс реализует метод execute(), в котором выполняется конкретное действие с контроллером. Реализован отдельным файлом commands.py

Разработанные команды:

1. AddToCartCommand – добавляет товар в корзину.
2. PayCommand – выполняет оплату через контроллер.
3. WorkCommand – вызывает игровую механику "пойти на работу".

Каждая команда получает нужные данные через конструктор, а вся логика выполняется при вызове execute().

Команды можно передавать в другие компоненты интерфейса (например, кнопки), не зная, что именно они делают. Это упрощает структуру программы и делает действия пользователя взаимозаменяемыми объектами.

Согласно шаблону проектирования MVC – контроллер полностью управляет пользовательскими действиями, абстрагируя модель от представления.

Компонент шаблона MVC – **View** – реализован с помощью фреймворка **Flask** – за это отвечает файл run.py, index.html, style.css.

Более подробно о всех методах в веб-документации, разработанной с помощью docstring и pdoc3.

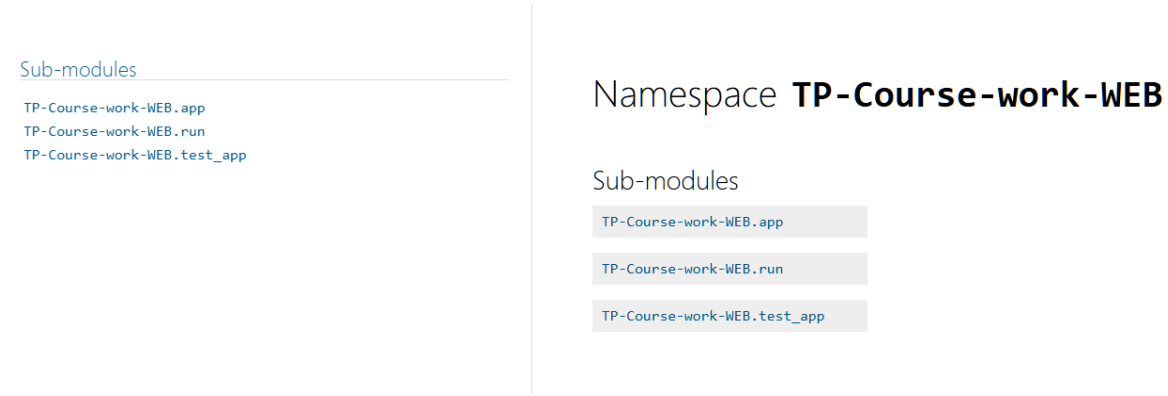

Рисунок 2.1 – Главная страница документации

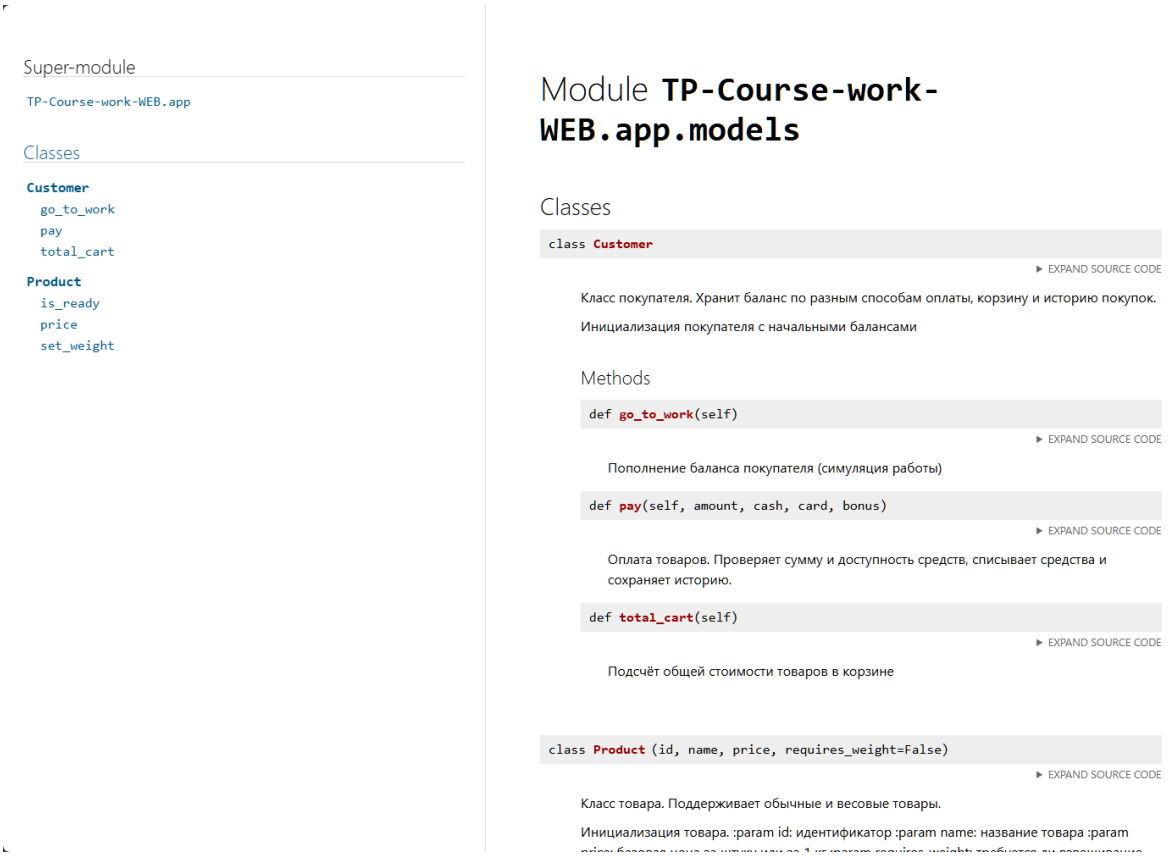

Рисунок 2.2 – Подробно разобраны классы и функции модели

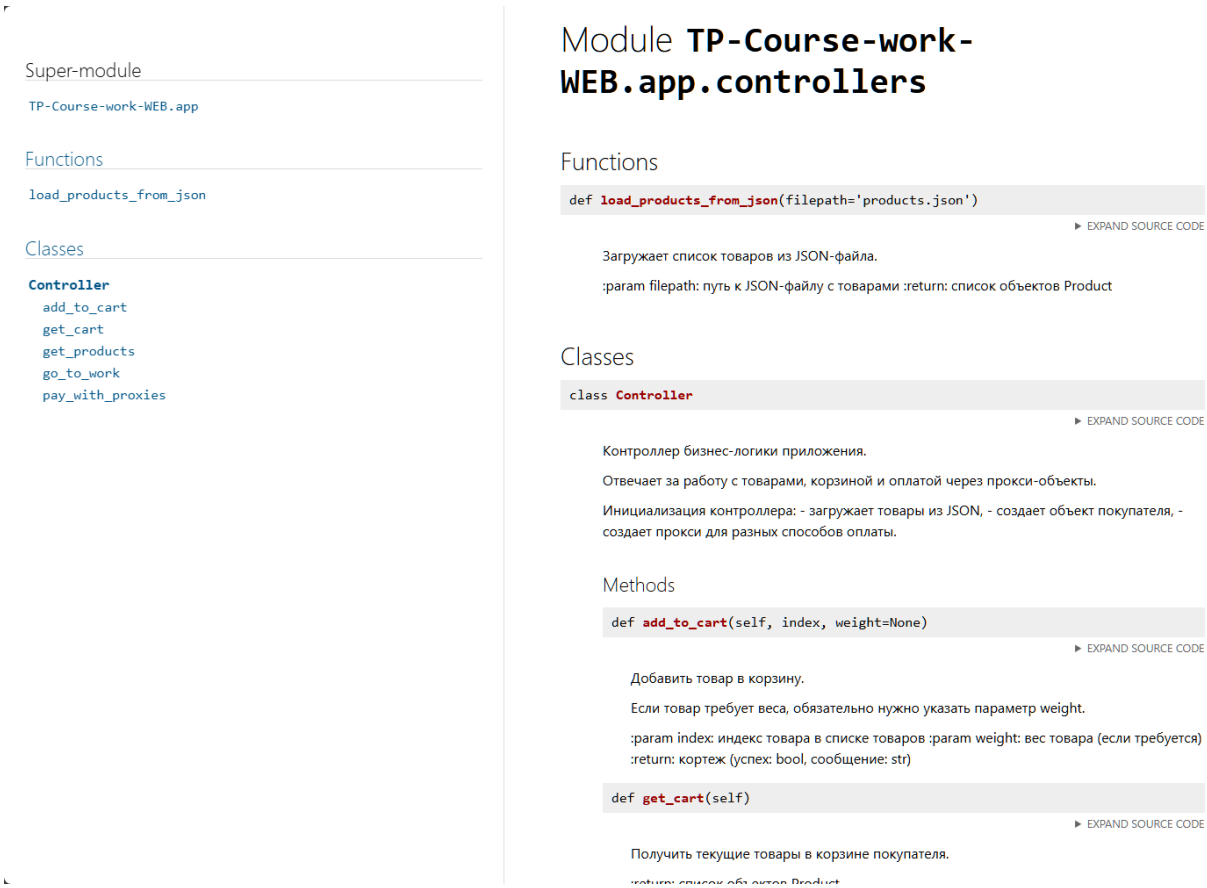

Рисунок 2.3 – Пример документации по части контроллера

5. Описание проведения модульного тестирования

Модульное тестирование = unit test. Для тестирования функциональности программы, был разработан файл test_app.py. Тестируем:

1) Модель покупателя (Customer)
2) Контроллер (Controller)
3) Сценарии оплаты через прокси (PaymentProxy)

Каждая группа тестов оформлена в виде отдельного класса, унаследованного от unittest.TestCase.

1. TestCustomer

Проверяет:

- Успешную оплату товара с точной суммой;
- Ошибку при недостатке средств;
- Ошибку при некорректном распределении оплаты по типам средств.

2. TestController

Проверяет:

- Добавление товаров в корзину;
- Обработку весовых товаров;
- Ошибки при отсутствии веса.

3. TestPayWithProxies

Проверяет оплату через паттерн Proxy:

- Недостаток средств;
- Точная оплата;
- Переплата (возврат сдачи).

Модульное тестирование проверяет:

- Изоляция: каждый тест работает независимо.
- Повторяемость: при одинаковом входе ожидается одинаковый результат.
- Проверка бизнес-логики: оплаты, корзины, прокси и др.
- Обратная связь при изменениях: если в коде появится ошибка — тест это покажет.

6. Результаты работы программы с примерами разных сценариев

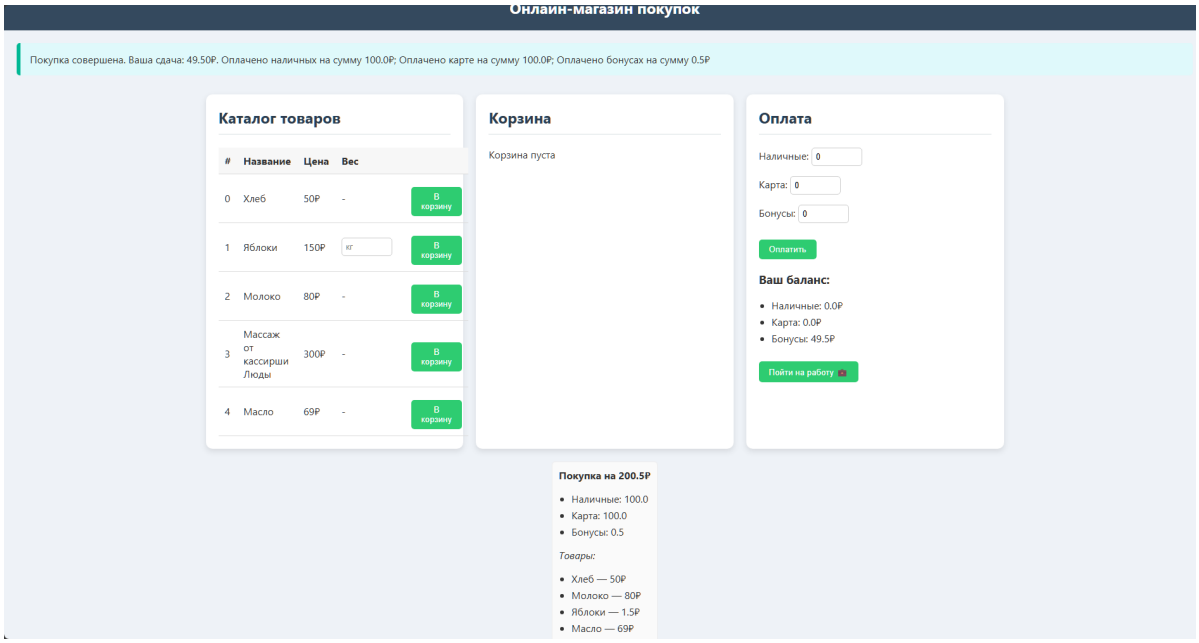

Рисунок 3.1 – Пользователь успешно отоваривается

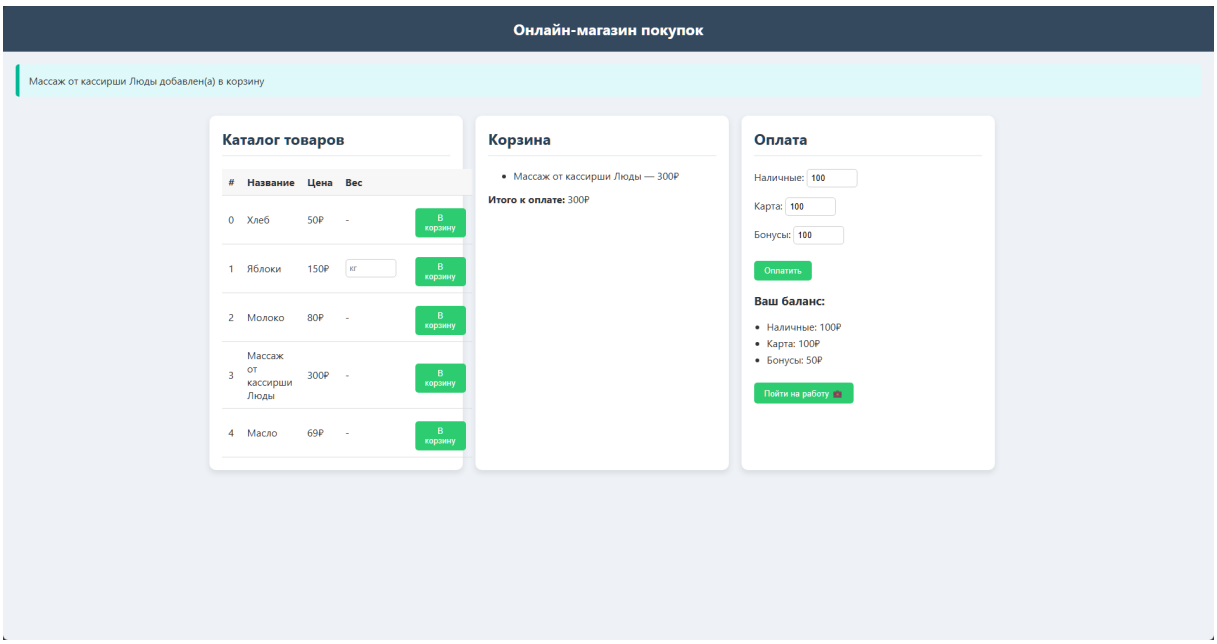

Рисунок 3.2 – Пользователь пытается обмануть

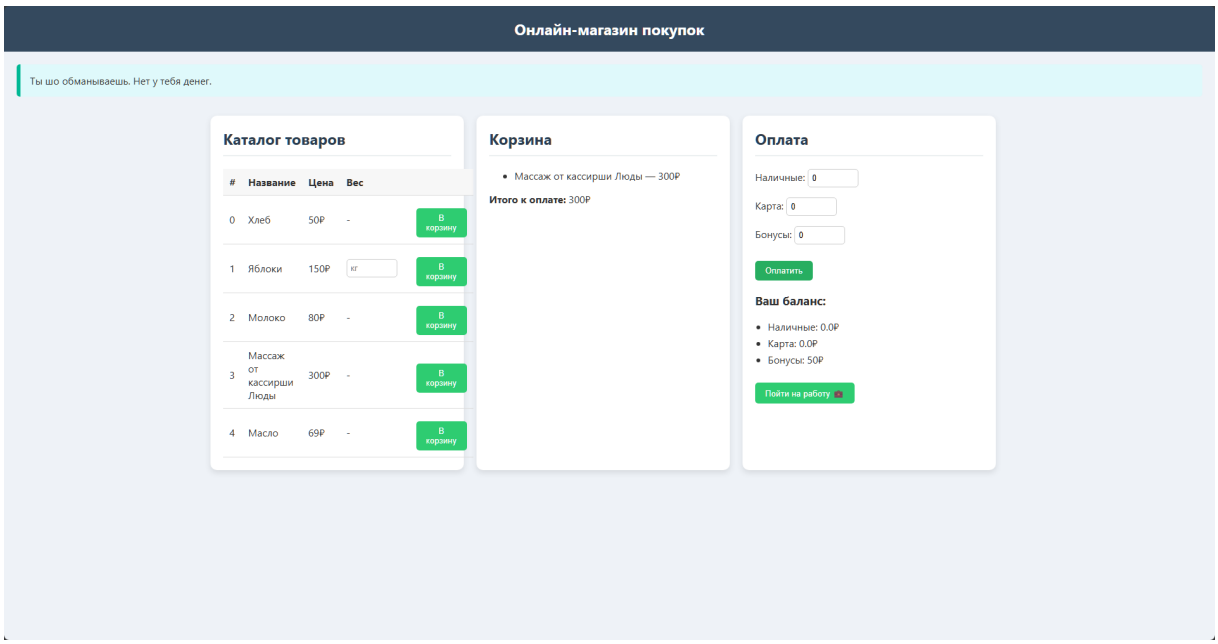

Рисунок 3.3 – Пользователю не хватило денег (и он заплатил штраф с того, что было)

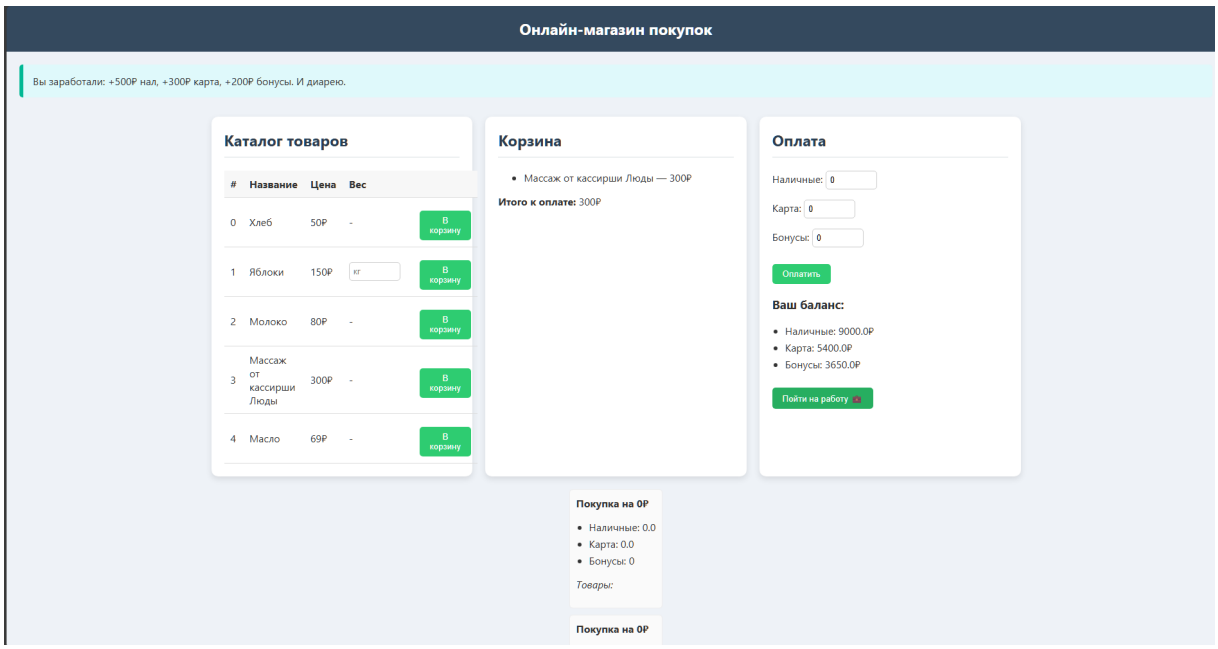

Рисунок 3.4 – Пользователь пошёл на работу и заработал много денег

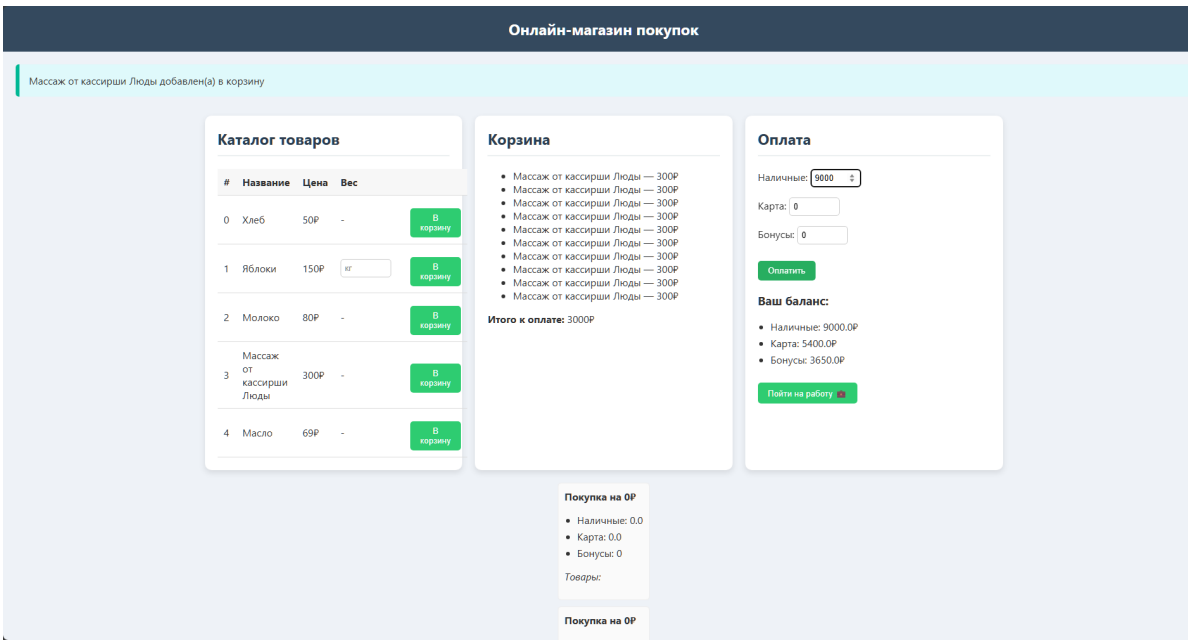

Рисунок 3.5 – Пользователь шикует

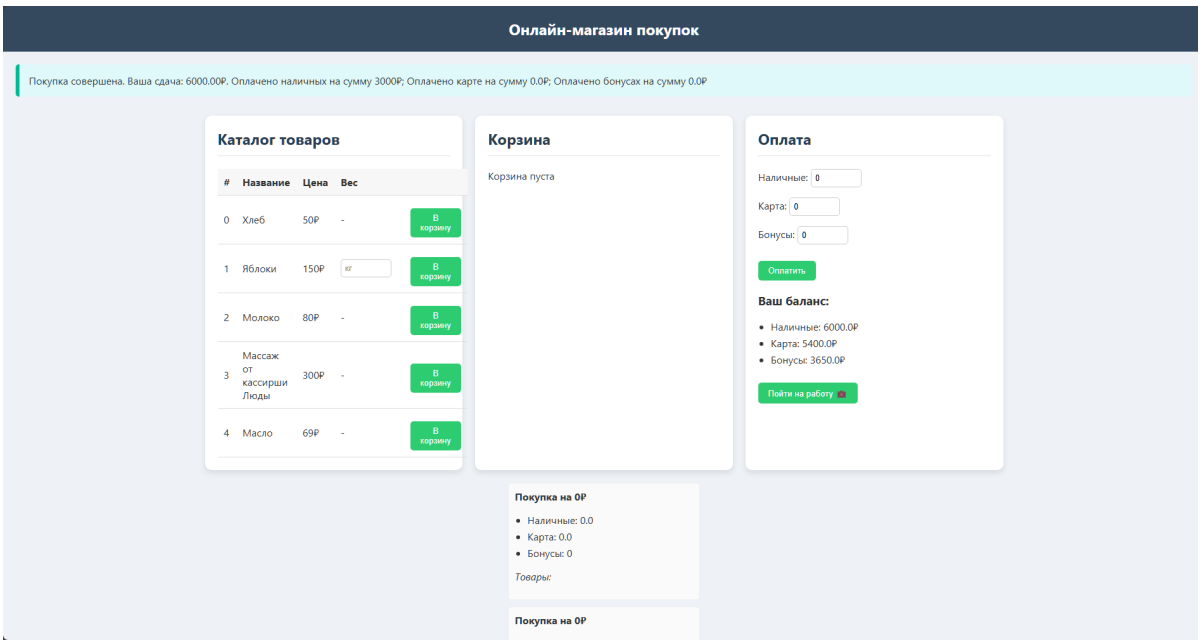

Рисунок 3.6 – Но веб-страница не принимает чаевые, а даёт сдачи.

ЗАКЛЮЧЕНИЕ

Код программы выложен и доступен на удаленном репозитории на GitHub`e: MyataEtoki/TP-Course-work-WEB

В результате выполнения курсового проекта была разработана программа по шаблону проектирования MVC, имитирующая веб-страницу магазина. Были использованы такие паттерны программирования как: Proxy (Заместитель) и Command (Команда). Они позволили сделать программу более расширяемой и понятной. Файлы проекта содержат все необходимые классы и методы для корректной работы программы. Также было выполнено модульное тестирование, используя unittest, что говорит о работоспособности проекта.

В данной работе реализованы основные принципы объектно-ориентированного программирования: полиморфизм, инкапсуляция, наследование, абстракция; а также принципы SOLID.

Функционал программы был подробно описан в пользовательской и технической документации. Все поставленные задачи реализованы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справка по поведенческому паттерну «Команда» (дата посещения 06.06.25): https://refactoring.guru/ru/design-patterns/command
2. Справка по структурному паттерну «Заместитель» (дата посещения 06.06.25): https://refactoring.guru/ru/design-patterns/proxy
3. Model-View-Controller — Википедия (дата посещения 06.06.2025): https://ru.wikipedia.org/wiki/Model-View-Controller
4. «Разработка веб-приложений с использованием Flask на языке Python» автор: Гринберг М., изд. «ДМК Пресс», 2014г.
5. Юнит-тестирование для чайников / Хабр (дата посещения 06.06.2025) https://habr.com/ru/articles/169381/
6. Паттерны для новичков: MVC vs MVP vs MVVM (дата посещения 06.06.2025) https://habr.com/ru/post/215605/
7. Современная MVI-архитектура на базе Kotlin. Часть 1 (дата посещения 06.06.2025) https://www.pvsm.ru/programmirovanie/298986
8. Шаблоны (паттерны) проектирования для людей (дата посещения 06.06.2025) https://github.com/design-patterns-for-humans/Russian
9. Паттерны проектирования: какие бывают и как выбрать нужный (дата посещения 06.06.2025) https://gb.ru/blog/patterny-proektirovaniya/
10. Паттерны проектирования: твоя настольная статья (дата посещения 06.06.2025) https://proglib.io/p/patterny-proektirovaniya-tvoya-nastolnaya-statya-2019-10-27

ПРИЛОЖЕНИЕ

run.py:

```
from flask import render_template, request, redirect, url_for, session, flash
from app import create_app
from app.controllers import Controller
from app.commands import PayCommand

# Инициализация Flask-приложения и основного контроллера
app = create_app()
controller = Controller()

@app.route('/')
def index():
    """
    Главная страница.
    Отображает список товаров, корзину, баланс и историю покупок.
    """
    return render_template(
        "index.html",
        products=controller.get_products(),
        cart=controller.get_cart(),
        history=controller.customer.purchase_history,
        balance={
            "cash": controller.customer.cash,
            "card": controller.customer.card,
            "bonus": controller.customer.bonus
        },
        total=controller.customer.total_cart()
    )

@app.route('/add', methods=["POST"])
def add():
    """
    Обработка добавления товара в корзину.
    Учитывает вес, если он требуется.
    """
    index = int(request.form['product_index'])
    weight = request.form.get(f'weight_{index}', None)
    success, msg = controller.add_to_cart(index, weight)
    flash(msg)
    return redirect(url_for('index'))

@app.route('/remove/<int:index>')
def remove(index):
    """
    Удаляет товар из корзины по его индексу.
    """
    controller.remove_from_cart(index)
    return redirect(url_for('index'))

@app.route('/buy', methods=['POST'])
```

```
def buy():
    """
    Обрабатывает оплату покупки.
    Использует паттерн 'Команда'.
    """
    amounts = [
        request.form.get('cash', '0'),
        request.form.get('card', '0'),
        request.form.get('bonus', '0')
    ]
    cmd = PayCommand(controller, amounts)
    success, msg = cmd.execute()
    flash(msg)
    return redirect(url_for('index'))

@app.route('/work')
def work():
    """
    Добавляет пользователю деньги и бонусы.
    Симулирует 'поход на работу'.
    """
    success, msg = controller.customer.go_to_work()
    flash(msg)
    return redirect(url_for('index'))

if __name__ == "__main__":
    """
    Запуск сервера Flask в режиме отладки.
    """
    app.run(debug=True)
```

index.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Магазин</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>

<header>
    Онлайн-магазин покупок
</header>

{% with messages = get_flashed_messages() %}
  {% if messages %}
    {% for message in messages %}
      <div class="message">{{ message }}</div>
    {% endfor %}
  {% endif %}
{% endwith %}
```

```
<main>

  <!-- Товары -->
  <div class="block">
    <h2>Каталог товаров</h2>
    <form action="/add" method="post">
<table>
  <tr><th>#</th><th>Название</th><th>Цена</th><th>Вес</th><th></th></tr>
  {% for p in products %}
  <tr>
    <td>{{ loop.index0 }}</td>
    <td>{{ p.name }}</td>
    <td>{{ p.price }}₽</td>
    <td>
      {% if p.requires_weight %}
        <input type="text" name="weight_{{ loop.index0 }}" placeholder="кг">
      {% else %}
        -
      {% endif %}
    </td>
    <td>
      <button type="submit" name="product_index" value="{{ loop.index0 }}">В
корзину</button>
    </td>
  </tr>
  {% endfor %}
</table>
</form>

  </div>

  <!-- Корзина -->
  <div class="block">
    <h2>Корзина</h2>
    {% if cart %}
      <ul>
        {% for item in cart %}
          <li>{{ item.name }} — {{ item.price }}₽</li>
        {% endfor %}
      </ul>
      <p><strong>Итого к оплате:</strong> {{ total }}₽</p>
    {% else %}
      <p>Корзина пуста</p>
    {% endif %}
  </div>

  <!-- Оплата -->
  <div class="block">
    <h2>Оплата</h2>
    <form action="/buy" method="post">
      <p><label>Наличные: <input type="number" name="cash" value="0"
min="0"></label></p>
```

```
        <p><label>Карта: <input type="number" name="card" value="0"
min="0"></label></p>
        <p><label>Бонусы: <input type="number" name="bonus" value="0"
min="0"></label></p>
        <input type="submit" value="Оплатить">
      </form>

      <div class="balance">
        <h3>Ваш баланс:</h3>
        <ul>
          <li>Наличные: {{ balance.cash }}₽</li>
          <li>Карта: {{ balance.card }}₽</li>
          <li>Бонусы: {{ balance.bonus }}₽</li>
        </ul>
        <form action="/work">
          <button type="submit">Пойти на работу 💼</button>
        </form>
      </div>
    </div>

    <!-- История -->
    <div class="history">
    {% for record in history %}
      <div class="history-item">
        <strong>Покупка на {{ record.total }}₽</strong>
        <ul>
          <li>Наличные: {{ record.method.cash }}</li>
          <li>Карта: {{ record.method.card }}</li>
          <li>Бонусы: {{ record.method.bonus }}</li>
        </ul>
        <em>Товары:</em>
        <ul>
          {% for product in record["items"] %}
            <li>{{ product[0] }} — {{ product[1] }}₽</li>
          {% endfor %}
        </ul>
      </div>
    {% endfor %}
</div>


</main>
</body>
</html>
Model.py:
class Product:
    """
    Класс товара. Поддерживает обычные и весовые товары.
    """

    def __init__(self, id, name, price, requires_weight=False):
        """
```

```
        Инициализация товара.
        :param id: идентификатор
        :param name: название товара
        :param price: базовая цена за штуку или за 1 кг
        :param requires_weight: требуется ли взвешивание
        """
        self.id = id
        self.name = name
        self.base_price = price
        self.requires_weight = requires_weight
        self.weight = None

    def set_weight(self, weight):
        """Установка веса товара (если требуется)"""
        self.weight = float(weight)

    @property
    def price(self):
        """Расчёт итоговой цены с учётом веса, если нужно"""
        return self.base_price * self.weight if self.requires_weight and self.weight else
self.base_price

    def is_ready(self):
        """Проверка, готов ли товар к покупке (взвешен ли он, если требуется)"""
        return not self.requires_weight or self.weight is not None


class Customer:
    """
    Класс покупателя. Хранит баланс по разным способам оплаты,
    корзину и историю покупок.
    """

    def __init__(self):
        """Инициализация покупателя с начальными балансами"""
        self.cash = 100
        self.card = 100
        self.bonus = 50
        self.cart = []
        self.purchase_history = []

    def total_cart(self):
        """Подсчёт общей стоимости товаров в корзине"""
        return sum(item.price for item in self.cart)

    def pay(self, amount, cash, card, bonus):
        """
        Оплата товаров.
        Проверяет сумму и доступность средств,
        списывает средства и сохраняет историю.
        """
        if cash + card + bonus != amount:
```

```
            return False, "Недостаточно средств, Сумма не совпадает с общей стоимостью"
        if cash > self.cash or card > self.card or bonus > self.bonus:
            return False, "Недостаточно средств"

        self.cash -= cash
        self.card -= card
        self.bonus -= bonus

        self.purchase_history.append({
            "items": [(p.name, p.price) for p in self.cart],
            "total": amount,
            "method": {"cash": cash, "card": card, "bonus": bonus}
        })

        self.cart = []
        return True, "Оплата прошла успешно"

    def go_to_work(self):
        """
        Пополнение баланса покупателя (симуляция работы)
        """
        self.cash += 500
        self.card += 300
        self.bonus += 200
        return True, "Вы заработали: +500₽ нал, +300₽ карта, +200₽ бонусы. И диарею."
```

Commands.py:

```
from abc import ABC, abstractmethod

class Command(ABC):
    """Абстрактный класс команды с методом execute."""

    @abstractmethod
    def execute(self):
        """Выполнить команду."""
        pass


class AddToCartCommand(Command):
    """Команда для добавления товара в корзину."""

    def __init__(self, controller, product_index, weight=None):
        """
        :param controller: контроллер приложения
        :param product_index: индекс товара в списке
        :param weight: вес товара (если требуется)
        """
        self.controller = controller
        self.product_index = product_index
        self.weight = weight

    def execute(self):
        """Выполнить добавление товара в корзину."""
```

```
        return self.controller.add_to_cart(self.product_index, self.weight)


class PayCommand(Command):
    """Команда для оплаты товаров."""

    def __init__(self, controller, amounts):
        """
        :param controller: контроллер приложения
        :param amounts: список сумм для оплаты [наличные, карта, бонусы]
        """
        self.controller = controller
        self.amounts = amounts

    def execute(self):
        """Выполнить оплату с проверкой формата сумм."""
        try:
            cash = float(self.amounts[0])
            card = float(self.amounts[1])
            bonus = float(self.amounts[2])
        except (ValueError, TypeError, IndexError):
            return False, "Неверный формат введённых сумм."

        return self.controller.pay_with_proxies([cash, card, bonus])


class WorkCommand(Command):
    """Команда для пополнения баланса (симуляция работы)."""

    def __init__(self, controller):
        """
        :param controller: контроллер приложения
        """
        self.controller = controller

    def execute(self):
        """Выполнить пополнение баланса."""
        return self.controller.go_to_work()
Payment_proxy.py:
class PaymentProxy:
    """
    Прокси для управления оплатой с разных источников баланса.

    :param get_balance_func: функция получения текущего баланса
    :param set_balance_func: функция установки нового баланса
    :param name: название источника (наличные, карта, бонусы)
    """

    def __init__(self, get_balance_func, set_balance_func, name):
        self._get = get_balance_func
        self._set = set_balance_func
        self.name = name
```

```
    def pay(self, amount):
        """
        Попытка списания указанной суммы с баланса.

        :param amount: сумма для списания
        :return: кортеж (успех: bool, сообщение: str)
        """
        balance = self._get()
        if balance >= amount:
            self._set(balance - amount)
            return True, f"Оплачено {self.name} на сумму {amount}₽"
        return False, f"Недостаточно средств на {self.name} (требуется {amount}₽, есть
{balance}₽)"
```

Controllers.py:

```
from .models import Product, Customer
from .payment_proxy import PaymentProxy  # Используем паттерн Прокси для оплаты
import json

def load_products_from_json(filepath='products.json'):
    """
    Загружает список товаров из JSON-файла.

    :param filepath: путь к JSON-файлу с товарами
    :return: список объектов Product
    """
    with open(filepath, encoding='utf-8') as f:
        data = json.load(f)
    products = []
    for p in data:
        products.append(Product(p['id'], p['name'], p['price'], p.get('requires_weight', False)))
    return products


class Controller:
    """
    Контроллер бизнес-логики приложения.

    Отвечает за работу с товарами, корзиной и оплатой через прокси-объекты.
    """

    def __init__(self):
        """
        Инициализация контроллера:
        - загружает товары из JSON,
        - создает объект покупателя,
        - создает прокси для разных способов оплаты.
        """
        self.products = load_products_from_json(filepath='products.json')
        self.customer = Customer()
        self.proxies = [
            PaymentProxy(lambda: self.customer.cash, lambda v: setattr(self.customer, 'cash', v),
```

```
"наличных"),
        PaymentProxy(lambda: self.customer.card, lambda v: setattr(self.customer, 'card', v),
"карте"),
        PaymentProxy(lambda: self.customer.bonus, lambda v: setattr(self.customer, 'bonus',
v), "бонусах")
    ]

  def get_products(self):
    """
    Получить список всех доступных товаров.

    :return: список объектов Product
    """
    return self.products

  def get_cart(self):
    """
    Получить текущие товары в корзине покупателя.

    :return: список объектов Product
    """
    return self.customer.cart

  def add_to_cart(self, index, weight=None):
    """
    Добавить товар в корзину.

    Если товар требует веса, обязательно нужно указать параметр weight.

    :param index: индекс товара в списке товаров
    :param weight: вес товара (если требуется)
    :return: кортеж (успех: bool, сообщение: str)
    """
    try:
      product = self.products[int(index)]
      if product.requires_weight:
        if not weight:
          return False, "Этот товар нужно взвесить"
        product = Product(product.id, product.name, product.base_price, True)
        product.set_weight(weight)
      self.customer.cart.append(product)
      return True, f"{product.name} добавлен(а) в корзину"
    except Exception as e:
      return False, f"Ошибка: {str(e)}"

  def pay_with_proxies(self, amounts):
    """
    Оплатить товары из корзины с использованием нескольких способов оплаты.

    :param amounts: список сумм для списания [наличные, карта, бонусы]
    :return: кортеж (успех: bool, сообщение: str)
    """
```

```
        total = self.customer.total_cart()
        total_payment = sum(float(a or 0) for a in amounts)

        if total_payment < total:
            return False, f"Недостаточно средств для оплаты. Нужно {total}₽, а вы указали
{total_payment}₽."

        paid = 0
        messages = []
        change = total_payment - total
        remaining = total

        for proxy, amount in zip(self.proxies, amounts):
            amount = float(amount or 0)
            to_pay = min(amount, remaining)
            success, msg = proxy.pay(to_pay)
            messages.append(msg)
            if success:
                paid += to_pay
                remaining -= to_pay
            else:
                return False, "Ты шо обманываешь. Нет у тебя денег. Плоти штраф."

        # Сохраняем историю покупок и очищаем корзину
        items = [(p.name, p.price) for p in self.customer.cart]
        self.customer.purchase_history.append({
            "items": items,
            "total": total,
            "method": {
                "cash": min(float(amounts[0] or 0), total),
                "card": min(float(amounts[1] or 0), max(0, total - float(amounts[0] or 0))),
                "bonus": min(float(amounts[2] or 0), max(0, total - float(amounts[0] or 0) -
float(amounts[1] or 0)))
            }
        })
        self.customer.cart = []

        change_msg = f" Ваша сдача: {change:.2f}₽." if change > 0 else ""
        return True, "Покупка совершена." + change_msg + " " + "; ".join(messages)

    def go_to_work(self):
        """
        Игровая механика: пополнение баланса покупателя.

        Увеличивает наличные, баланс карты и бонусы.

        :return: кортеж (успех: bool, сообщение: str)
        """
        self.customer.cash += 500
        self.customer.card += 300
        self.customer.bonus += 200
        return True, "Вы заработали: +500₽ нал, +300₽ карта, +200₽ бонусы. И невроз."
```

Test_app.py:

```
import unittest
from app.models import Product, Customer
from app.controllers import Controller

# запуск - в терминал: python -m unittest test_app.py

class TestCustomer(unittest.TestCase):
    """Тесты для класса Customer: проверка оплаты и работы с корзиной"""

    def test_successful_payment(self):
        """Проверка успешной оплаты при точном соответствии суммы"""
        customer = Customer()
        customer.cart = [Product(0, "Хлеб", 50)]
        success, msg = customer.pay(50, 50, 0, 0)
        self.assertTrue(success)
        self.assertEqual(customer.cash, 50)          # Баланс наличных обновился
        self.assertEqual(len(customer.cart), 0)      # Корзина очищена после покупки
        self.assertEqual(len(customer.purchase_history), 1)  # История покупок обновлена

    def test_insufficient_funds(self):
        """Проверка ошибки при несоответствии суммы оплаты и стоимости"""
        customer = Customer()
        customer.cart = [Product(0, "Хлеб", 100)]
        success, msg = customer.pay(100, 50, 30, 10)  # Сумма меньше цены товара
        self.assertFalse(success)
        self.assertIn("Сумма не совпадает с общей стоимостью", msg)

    def test_wrong_split(self):
        """Проверка ошибки при превышении средств на оплату отдельными способами"""
        customer = Customer()
        customer.cart = [Product(0, "Хлеб", 100)]
        success, msg = customer.pay(90, 60, 30, 10)
        self.assertFalse(success)
        self.assertIn("Недостаточно средств", msg)

class TestController(unittest.TestCase):
    """Тесты для контроллера: добавление товаров в корзину с разными условиями"""

    def setUp(self):
        """Подготовка контроллера с товарами для тестов"""
        self.controller = Controller()
        self.controller.products = [
            Product(0, "Хлеб", 50),
            Product(1, "Яблоки", 150, requires_weight=True)
        ]

    def test_add_to_cart_simple(self):
        """Добавление простого товара без веса в корзину"""
        success, msg = self.controller.add_to_cart(0)
        self.assertTrue(success)
        self.assertEqual(len(self.controller.customer.cart), 1)
```

```
    def test_add_to_cart_requires_weight_missing(self):
        """Попытка добавить товар с весом без указания веса — ошибка"""
        success, msg = self.controller.add_to_cart(1)
        self.assertFalse(success)
        self.assertIn("нужно взвесить", msg)

    def test_add_to_cart_with_weight(self):
        """Добавление товара с весом и проверка правильного расчёта цены"""
        success, msg = self.controller.add_to_cart(1, weight="2")
        self.assertTrue(success)
        product = self.controller.customer.cart[0]
        self.assertEqual(product.weight, 2.0)
        self.assertAlmostEqual(product.price, 300.0)

class TestPayWithProxies(unittest.TestCase):
    """Тесты оплаты через прокси с разными сценариями"""

    def setUp(self):
        """Подготовка контроллера с одним товаром в корзине"""
        self.controller = Controller()
        self.controller.products = [Product(0, "Молоко", 80)]
        self.controller.customer.cart = [self.controller.products[0]]

    def test_underpayment(self):
        """Оплата с недостаточной суммой — отказ"""
        result, msg = self.controller.pay_with_proxies([20, 20, 20])  # 60 < 80
        self.assertFalse(result)
        self.assertIn("Недостаточно средств", msg)

    def test_exact_payment(self):
        """Оплата точной суммой — успешный платёж"""
        result, msg = self.controller.pay_with_proxies([30, 30, 20])  # ровно 80
        self.assertTrue(result)
        self.assertIn("Покупка совершена", msg)

    def test_overpayment(self):
        """Оплата с переплатой — проверка наличия сдачи"""
        result, msg = self.controller.pay_with_proxies([100, 0, 0])  # переплата
        self.assertTrue(result)
        self.assertIn("сдача", msg.lower())
products.json:
[
  {"id": 0, "name": "Хлеб", "price": 50, "requires_weight": false},
  {"id": 1, "name": "Яблоки", "price": 150, "requires_weight": true},
  {"id": 2, "name": "Молоко", "price": 80, "requires_weight": false},
  {"id": 3, "name":  "Массаж от кассирши Люды", "price":  300, "requires_weight": false},
  {"id": 4, "name": "Масло", "price": 69, "requires_weight":  false}
]
```