ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

ОТЧЕТ

по учебной практике
по профессиональному модулю

ПМ.01. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ

специальности 09.02.07 Информационные системы и программирование
Квалификация Программист

Студент гр. ИС(ПРО)-41 _____ Мальцева А.А.
подпись *Фамилия И.О.*

Руководитель практики от ГБПОУ РХ ХПК _____ _____ _____ Черкашин Д.С.
оценка *дата* *подпись* *Фамилия И.О.*

Абакан 2026 г.

СОДЕРЖАНИЕ

1 Тестирование программных модулей

Тестирование программного обеспечения — это комплексный процесс, включающий в себя разнообразные методы и стратегии для всесторонней проверки работоспособности продукта. Его главная цель заключается не только в поиске технических ошибок и сбоев, но и в тщательной верификации соответствия системы заявленным требованиям и реальным потребностям пользователей.

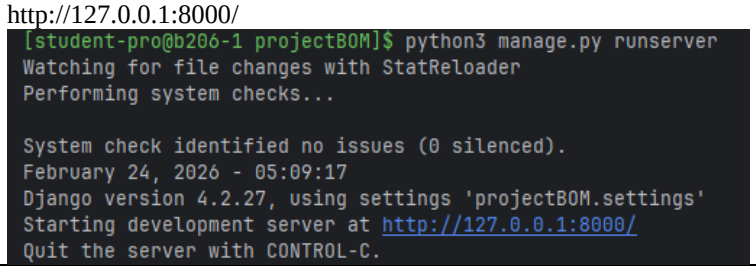
Этот этап разработки позволяет инженерам оценить, насколько стабильно и эффективно приложение функционирует, как в контролируемой тестовой среде, так и в условиях реальной эксплуатации конечными пользователями. Благодаря тестированию команда разработки получает объективную аналитику о качестве созданного продукта: выявляются скрытые уязвимости, узкие места в архитектуре и потенциальные риски, которые могли быть упущены на ранних стадиях.

1.1 Разработка тест-кейсов

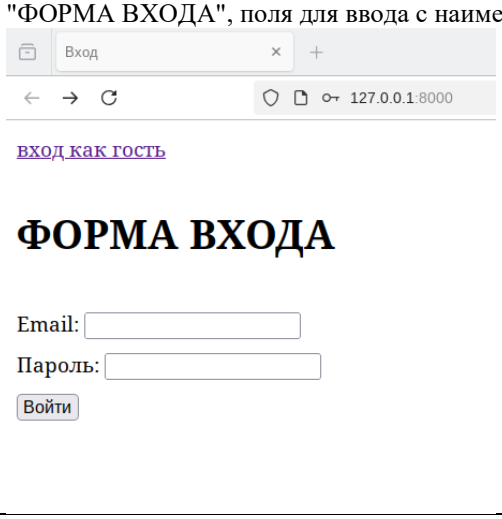
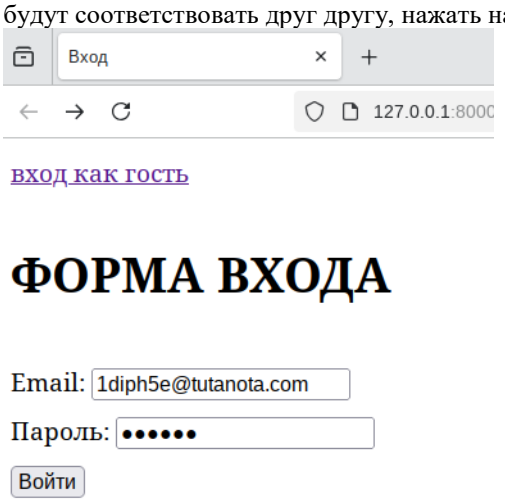
Тест-кейсом называют документ с набором шагов и ожидаемым результатом, который описывает проверку каких-либо определенных функций программного продукта. Тест-кейс включает в себя уникальный идентификатор, заголовок, предусловия, шаги, ожидаемый результат и статус.

Идентификатор: уникальный номер или код, используемый для поиска, ссылок и отчетов. Заголовок: краткое и понятное описание того, что именно проверяется. Предусловия: описание состояния системы или необходимых действий до начала теста. Шаги: нумерованная последовательность действий, необходимых для проверки. Ожидаемый результат: Точное описание того, как система должна отреагировать, если она работает правильно.

Таблица 1.1 - Тест-кейс №1

Номер	1
Название	Тестирование входа в аккаунт с ролью “Менеджер”
Предусловие	--
Шаг №1	Запустить программу через python3 manage.py runserver и перейти на на страницу http://127.0.0.1:8000/ 

Продолжение таблицы 1.1

Результат шага №1	Отображение окна с наименованием "Вход", ссылкой на "вход как гость", заголовок "ФОРМА ВХОДА", поля для ввода с наименованиями "Email" и "Пароль", кнопка войти. 
Шаг №2	Ввести email и пароль с существующими данными в таблице, где пароль и email будут соответствовать друг другу, нажать на кнопку "Войти" 

Продолжение таблицы 1.1

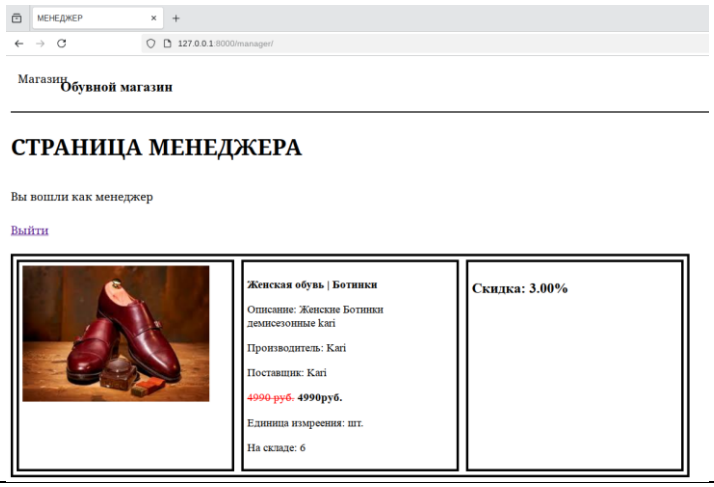
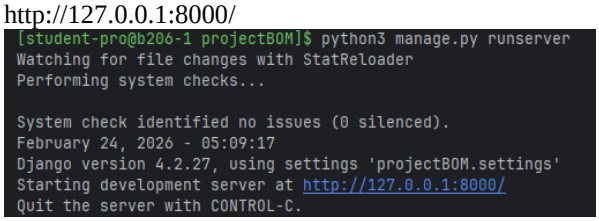
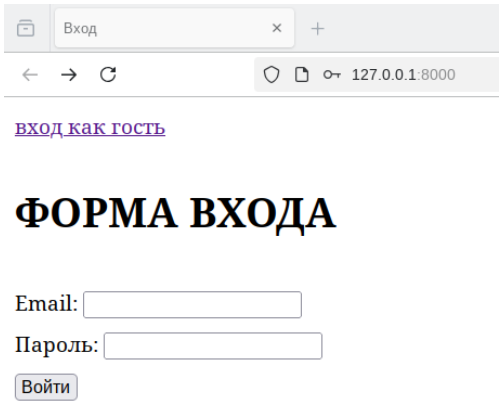
Результат шага №2	Переход по ссылке на соответствующую этому email и паролю страницу в зависимости от пары выбранных данных и присвоенной ей ролью "Менеджер". Переход должен произойти на существующую страницу "МЕНЕДЖЕР", с информацией о товарах и кнопкой "Выйти" 
Поведение	Позитивное
Статус	Пройден

Таблица 1.2 - Тест-кейс №2

Номер	2
Название	Тестирование защиты входа и переадресации на страницу входа при ложном email
Предусловие	---
Шаг №1	Запустить программу через python3 manage.py runserver и перейти на на страницу http://127.0.0.1:8000/ 
Результат шага №1	Отображение окна с наименованием "Вход", ссылкой на "вход как гость", заголовок "ФОРМА ВХОДА", поля для ввода с наименованиями "Email" и "Пароль", кнопка войти. 

Продолжение таблицы 1.2

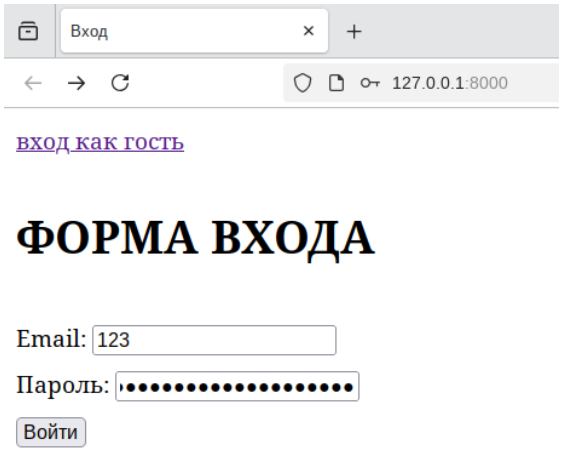
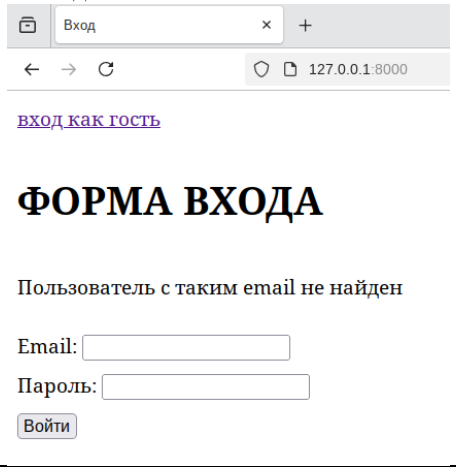
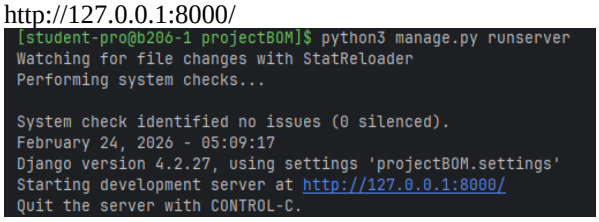
Шаг №2	Ввод ложных данных в форму входа не соответствующие данным, нажать на кнопку “Войти” 
Результат шага №2	Пользователь остался на экране входа с сообщением об ошибке входа в следствии ненайденного логина 
Поведение	Позитивное
Статус	Пройден

Таблица 1.3 - Тест-кейс №3

Номер	3
Название	Тестирование защиты входа через изменение ссылки
Предусловие	---
Шаг №1	Запустить программу через python3 manage.py runserver и перейти на на страницу http://127.0.0.1:8000/ 

Продолжение таблицы 1.3

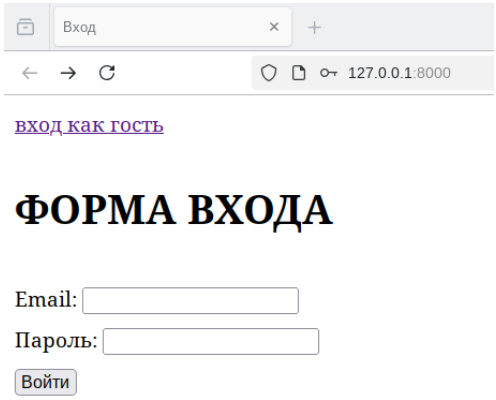
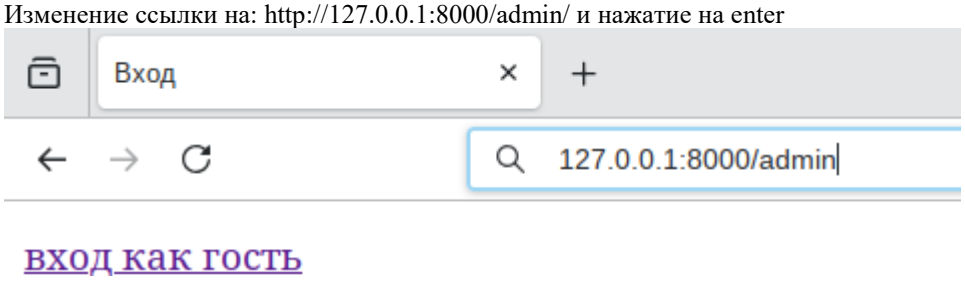
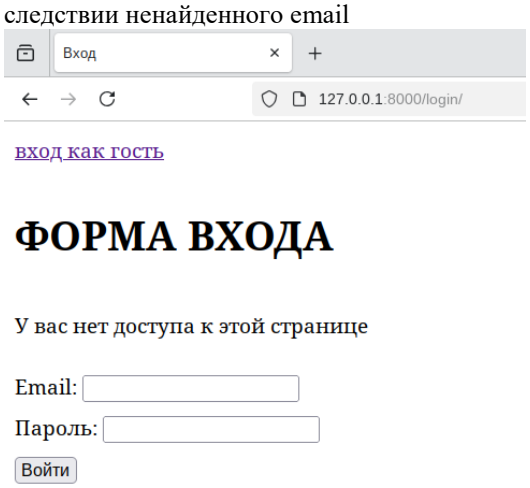
Результат шага №1	Отображение окна с наименованием "Вход", ссылкой на "вход как гость", заголовок "ФОРМА ВХОДА", поля для ввода с наименованиями "Email" и "Пароль", кнопка войти. 
Шаг №2	Изменение ссылки на: http://127.0.0.1:8000/admin/ и нажатие на enter 
Результат шага №2	Пользователь переадресовался на экран входа с сообщением об ошибке входа в следствии ненайденного email 
Поведение	Позитивное
Статус	Пройден

Таблица 1.4 - Тест-кейс №4

Номер	4
Название	Тестирование защиты входа и переадресации на страницу входа при ложном пароле

Продолжение таблицы 1.4

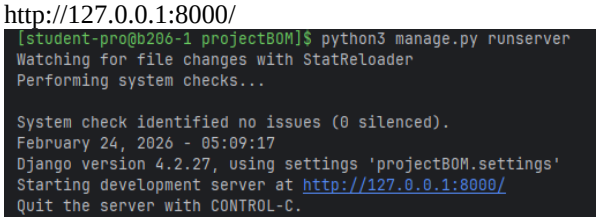
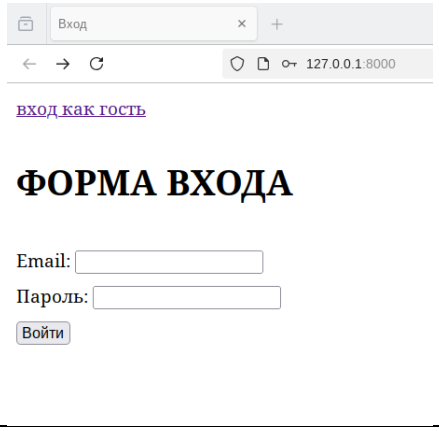
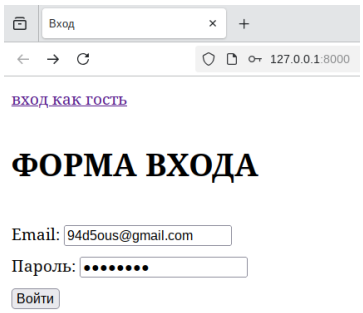
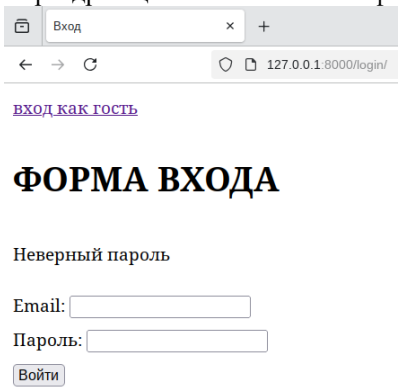
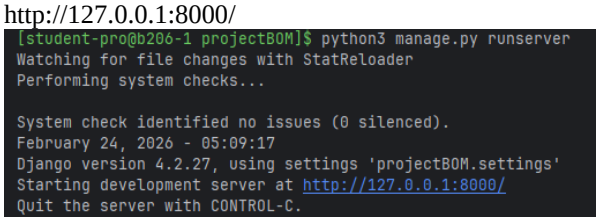
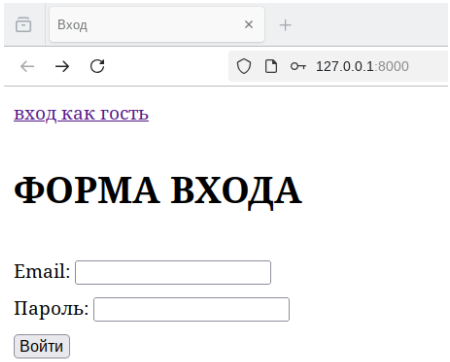
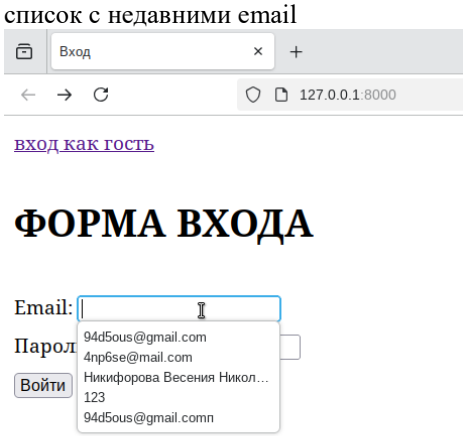
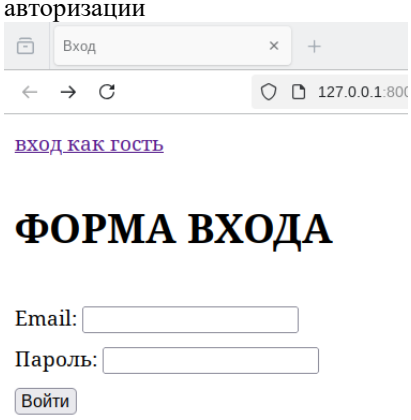
Предусловие	---
Шаг №1	Запустить программу через python3 manage.py runserver и перейти на на страницу http://127.0.0.1:8000/ 
Результат шага №1	Отображение окна с наименованием "Вход", ссылкой на "вход как гость", заголовок "ФОРМА ВХОДА", поля для ввода с наименованиями "Email" и "Пароль", кнопка войти. 
Шаг №2	Ввод верного email и ложных данных о пароле в форму входа, нажать на кнопку “Войти” 
Результат шага №2	Переадресация пользователя на форму входа с сообщением об неверном пароле 
Поведение	Позитивное
Статус	Пройден

Таблица 1.5 - Тест-кейс №5

Номер	5
Название	Тестирование защиты входа без email и пароля
Предусловие	---
Шаг №1	Запустить программу через python3 manage.py runserver и перейти на на страницу http://127.0.0.1:8000/ 
Результат шага №1	Отображение окна с наименованием "Вход", ссылкой на "вход как гость", заголовок "ФОРМА ВХОДА", поля для ввода с наименованиями "Email" и "Пароль", кнопка войти 
Шаг №2	При нажатии на кнопку входа курсор перемещается на пустое поле и открывает список с недавними email 

Продолжение таблицы 1.5

Результат шага №2	Пользователь остался на экране формы входа в ожидании ввода email и пароля для авторизации 
Поведение	Позитивное
Статус	Пройден

1.2 Модульное тестирование

Модульное тестирование подставляет собой метод проверки работоспособности наименьших изолированных частей кода, как функции, методы или классы. Оно нужно для выявления ошибок на ранних стадиях, упрощая отладку и безопасность документации кода. Каждый тест работает изолированно, используя заглушки вместо реальных зависимостей.

Структура теста включает в себя:

- Подготовка - инициализация переменных и настройка среды;
- Действие - вызов тестируемого метода;
- Утверждение - проверка соответствия полученного результата ожидаемому.

Листинг 1 - Тестирование модели “UnitMes”

```
from django.test import TestCase
from appBOM.models import UnitMes

class UnitMesTest(TestCase):
    """Тесты модели для единиц измерения"""

    def test_create(self):
        """Создание единицы измерения"""
        unit = UnitMes.objects.create(name='шт') #создание
объекта
```

```
        self.assertEqual(unit.name, 'шт') #добавление
        self.assertIsNotNone(unit.id)

    def test_str(self):
        """Проверка __str__ метода"""
        unit = UnitMes.objects.create(name='кг')
        self.assertEqual(str(unit), 'кг')

    def test_unique(self):
        """Проверка что можно создавать дубликаты"""
        UnitMes.objects.create(name='л')
        UnitMes.objects.create(name='л')

self.assertEqual(UnitMes.objects.filter(name='л').count(), 2)

    def test_max_length(self):
        """Проверка максимальной длины"""
        unit = UnitMes.objects.create(name='a' * 250)
        self.assertEqual(len(unit.name), 250)

    def test_delete(self):
        """Проверка удаления"""
        unit = UnitMes.objects.create(name='мл') #создание
        unit_id = unit.id
        unit.delete()

self.assertEqual(UnitMes.objects.filter(id=unit_id).count(), 0)
```

Работа тестового раннера django демонстрирует как изолированное развертывание временной базы данных успешно выполняет тест-кейсы с последующей очисткой окружения. Все тесты прошли удачно в соответствии с рисунком 2.1.

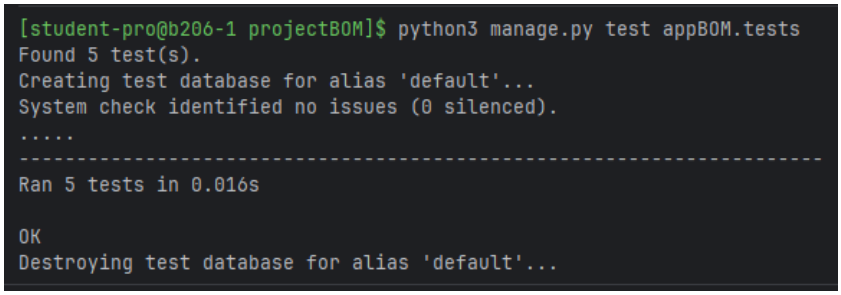


Рисунок 2.1 - Результаты тестирования модели единиц измерения

Листинг 2 - Тестирование гостевого шаблона show_guest.html

```
from django.test import TestCase, Client
from django.urls import reverse
from appBOM.models import Product, Artickles, ProductsNames,
UnitMes, Provider, Producer, CategoryProduct, Photos
from decimal import Decimal
```

```
class ShowGuestViewTest(TestCase):
    """Тестирование страницы гостя"""
    def setUp(self):
        """Подготовка тестовых данных"""
        self.client = Client()

        # Создаём зависимости для товара
        self.artickle = Artickles.objects.create(name='ART-001')
        self.prod_name =
ProductsNames.objects.create(name='Товар')
        self.unit = UnitMes.objects.create(name='шт')
        self.provider =
Provider.objects.create(name='Поставщик')
        self.producer =
Producer.objects.create(name='Производитель')
        self.category =
CategoryProduct.objects.create(name='Категория')
        self.photo = Photos.objects.create(name='photo.jpg')

        # Создаются товары
        self.product1 = Product.objects.create(
            artickle=self.artickle,
            name=self.prod_name,
            unit_mes=self.unit,
            price=1000,
            provider=self.provider,
            producer=self.producer,
            category=self.category,
            discount=Decimal('100.00'),
            count_store=50,
            description='Описание',
            photo_name=self.photo
        )
        self.product2 = Product.objects.create(
            artickle=self.artickle,
            name=self.prod_name,
            unit_mes=self.unit,
            price=2000,
            provider=self.provider,
            producer=self.producer,
            category=self.category,
            discount=Decimal('200.00'),
            count_store=30,
            description='Описание 2',
            photo_name=self.photo
        )

    def test_show_guest_status_code(self):
        """Тест 1: Проверка статуса ответа"""
        response = self.client.get('/guest/')  # или
```

```
reverse('show_guest')
        self.assertEqual(response.status_code, 200)

    def test_show_guest_template(self):
        """Тест 2: Проверка используемого шаблона"""
        response = self.client.get('/guest/')
        self.assertTemplateUsed(response, 'show_guest.html')

    def test_show_guest_context(self):
        """Тест 3: Проверка контекста"""
        response = self.client.get('/guest/')
        self.assertIn('products', response.context)
        self.assertEqual(len(response.context['products']), 2)

    def test_show_guest_products(self):
        """Тест 4: Проверка что товары передаются в шаблон"""
        response = self.client.get('/guest/')
        products = response.context['products']
        self.assertIn(self.product1, products)
        self.assertIn(self.product2, products)

    def test_show_guest_empty_products(self):
        """Тест 5: Проверка с пустым списком товаров"""
        Product.objects.all().delete()
        response = self.client.get('/guest/')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(len(response.context['products']), 0)
```

При прохождении тестирования передачи данных в шаблон работа тестового раннера django продемонстрировала успешное выполнение тест-кейсов с последующей очисткой окружения. Все тесты прошли удачно в соответствии с рисунком 2.2.

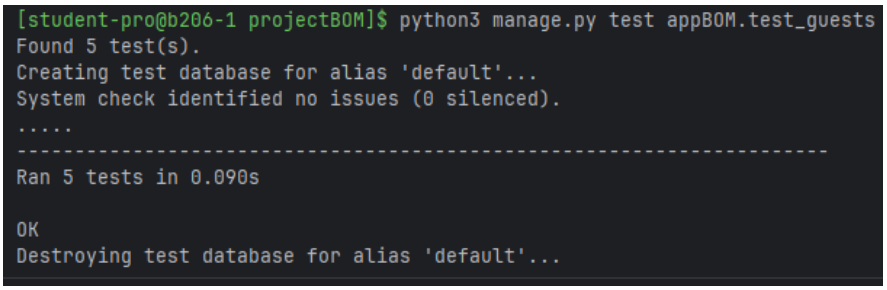


Рисунок 2.2 - Результаты тестирования отображения данных в шаблоне show_guest.py

Листинг 3 – Тестирование выхода из аккаунта на страницу входа

```
from django.test import TestCase, Client
from django.urls import reverse
```

```
from django.contrib.auth import get_user_model


class LogoutViewTest(TestCase):
    """Тестирование страницы выхода"""

    def setUp(self):
        """тестовые данные"""
        self.client = Client()
        User = get_user_model()
        self.user = User.objects.create_user(
            username='testuser',
            password='pass123'
        )

    def test_logout_redirect(self):
        """Тест 1: Проверка редиректа на login_view"""
        self.client.login(username='testuser',
password='pass123')
        response = self.client.get('/logout/')
        self.assertRedirects(response, reverse('login_view'))

    def test_logout_status_code(self):
        """Тест 2: Проверка ответа """
        self.client.login(username='testuser',
password='pass123')
        response = self.client.get('/logout/')
        self.assertEqual(response.status_code, 302)

    def test_logout_user_logged_out(self):
        """Тест 3: Проверка что пользователь вышел"""
        self.client.login(username='testuser',
password='pass123')
        self.client.get('/logout/')
        response = self.client.get('/logout/')
```

```
        # После выхода пользователь не аутентифицирован
        self.assertNotIn('_auth_user_id', self.client.session)

    def test_logout_without_login(self):
        """Тест 4: Выход без предварительного входа"""
        response = self.client.get('/logout/')
        self.assertRedirects(response, reverse('login_view'))
```

При прохождении тестирования, на выполняемость выхода из аккаунта на страницу входа, работа тестового раннера django показала, что успешно выполняет тест-кейсы с последующей очисткой окружения. Все тесты прошли удачно в соответствии с рисунком 2.3.

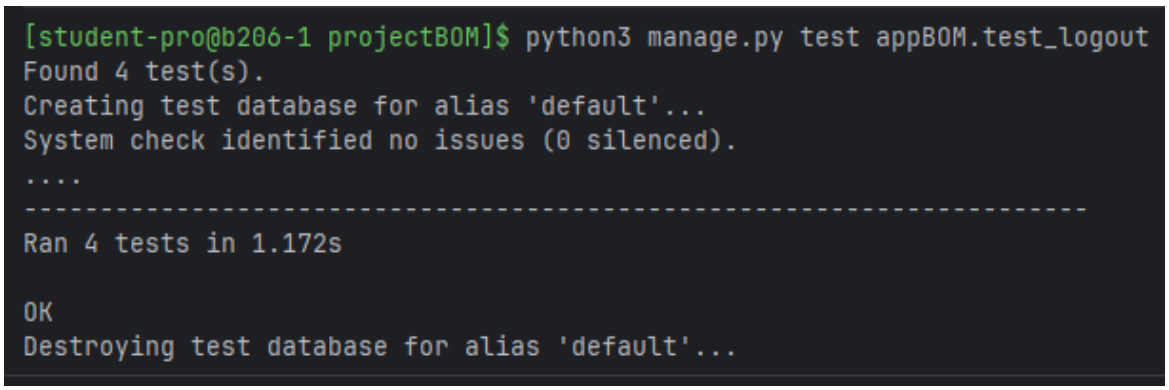


Рисунок 2.3 - Результаты тестирования выхода из аккаунта на страницу входа

ПРИЛОЖЕНИЯ

Программный код в системе контроля версий

Ссылка на Git - https://github.com/SnezhiK000/test-cases_practika