Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»

КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»

Лабораторная работа №4

«Информационные системы. Разработка программного кода. Рефакторинг»

ДИСЦИПЛИНА: «Методология программной инженерии»

Выполнил: студент гр. ИУК4-11М _____ (Сафронов Н.С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Белов Ю.С.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель работы: формирование навыков разработки информационной системы в соответствии с предъявляемыми требованиями.

Постановка задачи

Вариант 8

ИС «Торговля» (САБП, СЭДО).

В процессе выполнения лабораторной работы необходимо:

2. Произвести анализ разработанного программного кода.

3. Выявить фрагменты кода, требующие рефакторинга.

4. Провести рефакторинг кода.

5. Реализовать дополнительный функционал информационной системы (при необходимости)

Результат выполнения работы

Проблемы исходного кода:

1. Дублирование логики:

Одни и те же операции, такие как создание уведомлений или обработка ответов сервера, повторялись в нескольких местах.

2. Неунифицированные модели данных:

Результаты, возвращаемые различными сервисами, были неструктурированными, что усложняло их обработку.

3. Отсутствие единого формата ответа:

Сервисы возвращали данные в различных форматах, что приводило к ошибкам при их обработке на клиентской стороне.

4. Соответствие стандартам:

Хотя исходный код соответствовал стандарту PEP8, его структура могла быть улучшена для большего удобства чтения и поддержки.

5. Неэффективная обработка ошибок:

Обработка ошибок была разбросана по коду и не имела единой структуры, что затрудняло отладку и поддержку.

Результаты рефакторинга

1. Унификация моделей данных и результатов сервисов

До рефакторинга каждый сервис возвращал результаты в своём формате, что усложняло возврат данных клиенту. Разные форматы данных для разных сервисов приводили к необходимости дополнительных преобразований, что делало код менее читаемым и трудным для поддержки.

Решение:

В рамках рефакторинга были созданы унифицированные модели данных, используемые для всех ответов сервисов. Эти модели стандартизировали структуру данных, что обеспечило их последовательное использование на всех уровнях приложения. Унифицированные модели данных определяют четкую структуру ответа для различных типов данных, таких как клиенты, товары, заказы и другие сущности.

```python
@dataclass
class BaseModel:
    """Base model providing a to_dict method."""

    3 usages  keyone
    def to_dict(self) -> dict:
        """Convert the model instance to a dictionary."""
        keyone
        def serialize(obj: Any) -> Any:
            """Helper function to serialize nested objects."""
            if is_dataclass(obj):
                return asdict(obj)
            if isinstance(obj, list):
                return [serialize(item) for item in obj]
            if isinstance(obj, dict):
                return {key: serialize(value) for key, value in obj.items()}
            return obj

        return serialize(self)
```

Рисунок 1 – Созданная базовая модель

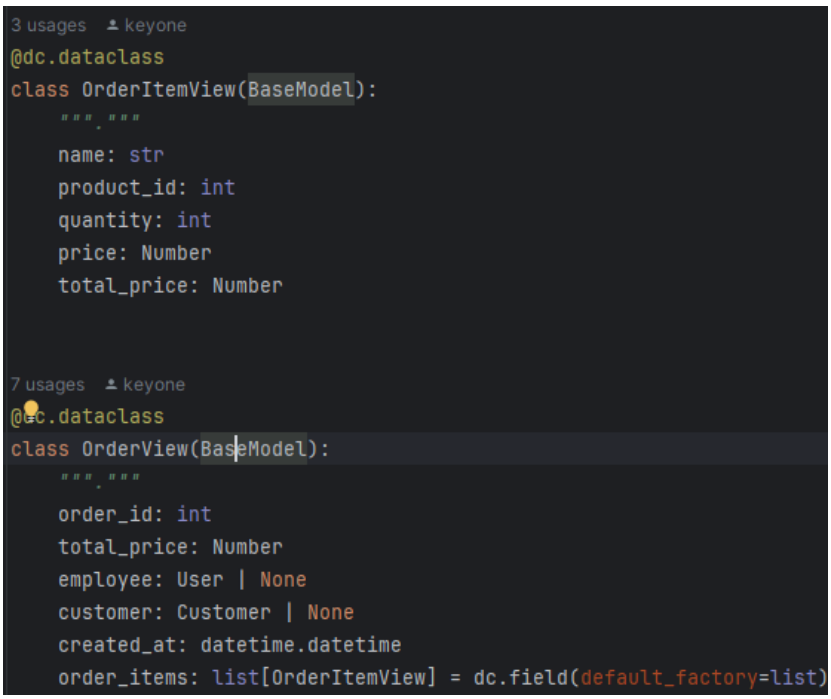

Рисунок 2 – Пример применения наследования от базовой модели

Рисунок 3 – Пример использования моделей в результатах сервисов

2. Централизация обработки ошибок и новая иерархия ошибок

В процессе рефакторинга была реализована централизованная обработка ошибок, что позволило улучшить диагностику и управление исключениями в системе. Ранее ошибки могли обрабатываться в разных частях приложения по-разному, что создавалло проблемы с унификацией и пониманием возникающих ошибок. Теперь все ошибки проходят через

централизованный механизм, что упрощает обработку, логирование и отображение сообщений об ошибках.

Решение:

В рамках рефакторинга была введена централизованная система обработки ошибок с использованием новой иерархии ошибок. Основной идеей стало разделение ошибок на несколько категорий и использование единой точки для их обработки. Также была обеспечена поддержка подробного логирования ошибок и информирования пользователей с учётом их типа и уровня.

Рисунок 4 – Новая иерархия классов ошибок

```python
keyone
@app.errorhandler(Exception)
def error_handler(error: Exception | ModuleException | HTTPException):
    """
    Обработчик ошибок для преобразования их в единый формат JSON.
    """
    if isinstance(error, ModuleException):
        response = flask.jsonify(error.to_dict())
        response.status_code = error.status_code
    elif isinstance(error, HTTPException):
        response = jsonify({
            "error": error.__class__.__name__,
            "data": {},
        })
        response.status_code = error.code
    else:
        response = jsonify({
            "error": "InternalServerError",
            "message": "Произошла внутренняя ошибка сервера.",
            "data": {
                'exception': error.__class__.__name__,
                'data': error.args,
            }
        })
        response.status_code = 500
    return response
```

Рисунок 5 – Созданный обработчик ошибок

3. Централизация обработки ошибок клиентом

В процессе рефакторинга была реализована централизованная обработка ошибок, что позволило улучшить диагностику и управление исключениями в системе. Ранее ошибки могли обрабатываться в разных частях клиентского приложения по-разному, что создавалло проблемы с унификацией и пониманием возникающих ошибок. Теперь все ошибки проходят через централизованный механизм, что упрощает обработку, логирование и отображение сообщений об ошибках.

Решение:

Централизация обработки ошибок на клиентской стороне была реализована путем создания единой функции для отображения сообщений об ошибках с использованием всплывающих уведомлений (тостов). Это позволило унифицировать отображение информации о проблемах, возникших в процессе взаимодействия с сервером, и повысило пользовательский опыт за счет единого стиля информирования о событиях.

```
// Функция для обработки ответа с сервера
async function handleResponse(response) {
    if (!response.ok) {
        const error = await response.json();
        const errorMessage = error.message || 'Неизвестная ошибка.';
        const errorData = error.data || {};
        console.error(`Ошибка: ${errorMessage}`, errorData);
        throw new Error(errorMessage);
    }
    return response.json();
}
```

Рисунок 6 – Функция обработки результата запроса

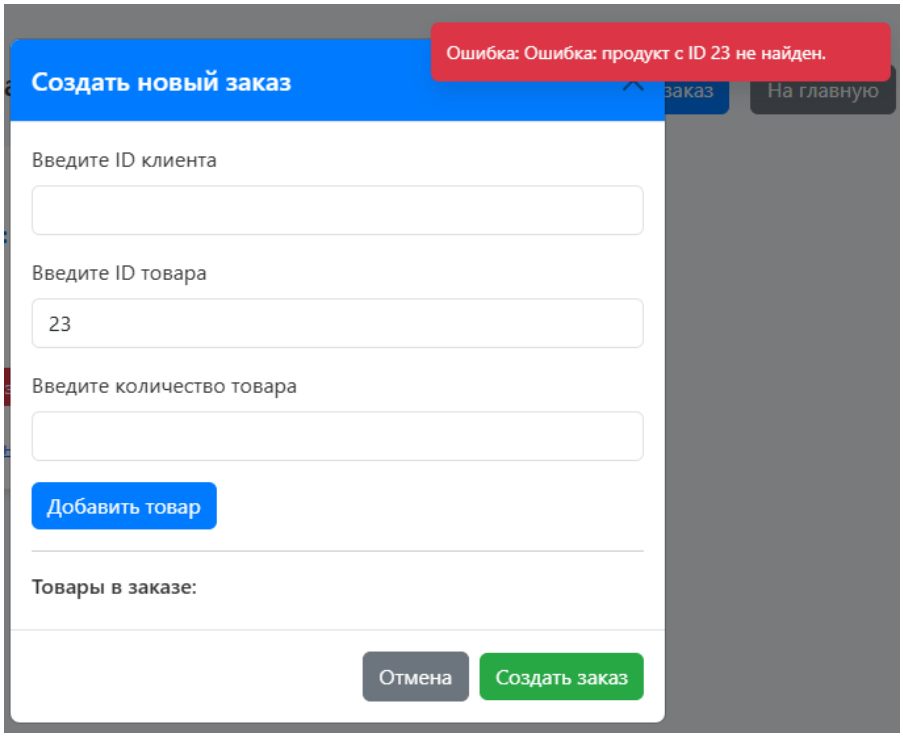

Рисунок 7 – Пример нового вывода информации об ошибке

4. Соответствие стандартам

В ходе работы над проектом было обеспечено соответствие кода принятым стандартам качества и стиля. Для этого применялись следующие инструменты:

Статический анализ с использованием mypy:

Все модули проекта были проверены на соответствие указанным аннотациям типов.

Использование mypy помогло обнаружить потенциальные ошибки, связанные с несоответствием типов, еще на этапе разработки.

Применение аннотаций улучшило читаемость кода и упростило работу с функциями и классами за счет явного определения их ожидаемых параметров и возвращаемых значений.

Проверка стиля кода с помощью flake8:

Код был приведен в соответствие стандарту PEP 8, включая требования к форматированию, именованию переменных, а также организации импортов.

flake8 позволил автоматизировать проверку кода и гарантировать, что он соответствует установленным правилам.

Устранены лишние пробелы, длинные строки разделены на более короткие, а структура модулей была упрощена для лучшей читаемости.

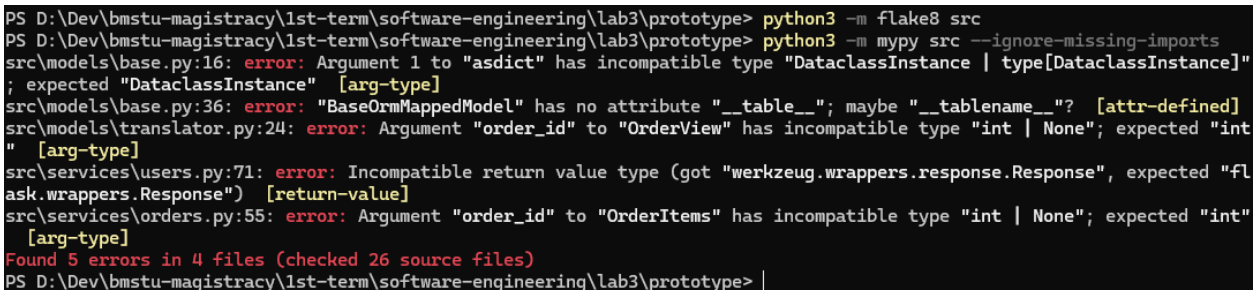

Рисунок 8 – Результат выполнения проверок flake8 и mypy

Вывод: в ходе выполнения лабораторной работы были сформированы навыки разработки информационной системы в соответствии с предъявляемыми требованиями.