**Отчет по самостоятельной работе №4**

**по дисциплине МДК 01.02 “Инструментальные средства разработки программного обеспечения”.**

Выполнил: студент

группы 319

Меркушин Евгений Романович

Дата 17.11.2024

**Цель работы**

Изучение процессов разработки отдельных модулей проекта, их последующей интеграции в командной работе, тестирования и устранения ошибок для обеспечения корректной работы всей системы.

**Основная структура задания**

 Для создания базы данных интернет-магазина можно использовать SQLite, так как это простая и удобная для работы СУБД, которая не требует установки отдельного сервера баз данных.

# Шаг 1: Создание таблицы товаров

Предположим, что у нас есть таблица products, где хранятся данные о товарах. Вот пример кода на Python с использованием библиотеки sqlite3:

import sqlite3

# Подключение к базе данных (если файл не существует, он будет создан)

conn = sqlite3.connect('internet\_shop.db')

cursor = conn.cursor()

# Создание таблицы products

cursor.execute(''' CREATE TABLE IF NOT EXISTS products ( id INTEGER PRIMARY KEY, name TEXT NOT NULL, price REAL NOT NULL, description TEXT, image BLOB ) ''')

# Сохранение изменений

conn.commit()

# Шаг 2: Определение типов файлов

Теперь определим список допустимых типов файлов для изображений товаров. Допустимые форматы могут быть, например, .jpg, .png, .gif. Мы можем хранить эти файлы непосредственно в базе данных в виде бинарных данных (BLOB), но чаще их хранят на диске, а в базе данных указывают путь к файлу.

Пример функции проверки типа файла по его расширению:

def is\_valid\_file\_type(filename):

valid\_types = ['.jpg', '.jpeg', '.png', '.gif']

return any([filename.lower().endswith(ext) for ext in valid\_types])

# Пример использования

file\_name = 'product\_image.jpg'

if is\_valid\_file\_type(file\_name):

print("Файл допустимого типа.")

else:

print("Недопустимый тип файла.")

# Полный код

Вот полный пример программы, включающий создание базы данных и проверку типов файлов:

import sqlite3

# Подключение к базе данных (если файл не существует, он будет создан)

conn = sqlite3.connect('internet\_shop.db')

cursor = conn.cursor()

# Создание таблицы products

cursor.execute(''' CREATE TABLE IF NOT EXISTS products ( id INTEGER PRIMARY KEY, name TEXT NOT NULL, price REAL NOT NULL, description TEXT, image\_path TEXT ) ''')

# Функция проверки типа файла

def is\_valid\_file\_type(filename):

valid\_types = ['.jpg', '.jpeg', '.png', '.gif']

return any([filename.lower().endswith(ext) for ext in valid\_types])

# Добавление нового продукта

def add\_product(name, price, description, image\_path):

if not is\_valid\_file\_type(image\_path):

raise ValueError(f"Недопустимый тип файла: {image\_path}")

cursor.execute(''' INSERT INTO products (name, price, description, image\_path) VALUES (?, ?, ?, ?) ''', (name, price, description, image\_path))

# Сохраняем изменения

conn.commit()

# Пример добавления продукта

add\_product(

"Товар 1",

1000.00,

"Описание товара 1",

"images/product1.jpg"

)

print("Продукт успешно добавлен!")

# Закрытие соединения с базой данных

conn.close()

Этот скрипт создает таблицу products в базе данных internet\_shop.db, добавляет новый продукт и проверяет корректность типа файла перед добавлением записи.

можешь проверить работу, добавив тестовые файлы и убедиться что они исключены?

Конечно! Давайте дополним наш код, чтобы добавить несколько тестовых файлов и убедиться, что недопустимые типы файлов действительно исключаются при попытке добавить товар.

# Обновленный код

В этом примере мы создадим функцию для генерации случайного набора продуктов с разными типами файлов, включая допустимые и недопустимые. Затем попробуем добавить каждый из этих продуктов в базу данных и убедимся, что те продукты, у которых тип файла недопустим, вызывают исключение.

import sqlite3

import random

from string import ascii\_letters

# Подключение к базе данных (если файл не существует, он будет создан)

conn = sqlite3.connect('internet\_shop.db')

cursor = conn.cursor()

# Создание таблицы products

cursor.execute(''' CREATE TABLE IF NOT EXISTS products ( id INTEGER PRIMARY KEY, name TEXT NOT NULL, price REAL NOT NULL, description TEXT, image\_path TEXT ) ''')

# Функция проверки типа файла

def is\_valid\_file\_type(filename):

valid\_types = ['.jpg', '.jpeg', '.png', '.gif']

return any([filename.lower().endswith(ext) for ext in valid\_types])

# Генерация случайной строки

def generate\_random\_string(length=10):

return ''.join(random.choice(ascii\_letters) for \_ in range(length))

# Генерация случайного числа

def generate\_random\_price():

return round(random.uniform(50, 500), 2)

# Генерация случайного описания

def generate\_random\_description():

return f'Описание товара: {generate\_random\_string()}'

# Генерация случайного имени файла

def generate\_random\_filename(valid=True):

extensions = ['.jpg', '.jpeg', '.png', '.gif'] if valid else ['.txt', '.pdf', '.exe']

return f'{generate\_random\_string()}{random.choice(extensions)}'

# Добавление нового продукта

def add\_product(name, price, description, image\_path):

if not is\_valid\_file\_type(image\_path):

raise ValueError(f"Недопустимый тип файла: {image\_path}")

cursor.execute(''' INSERT INTO products (name, price, description, image\_path) VALUES (?, ?, ?, ?) ''', (name, price, description, image\_path))

# Сохраняем изменения

conn.commit()

# Генерация тестовых данных

for i in range(5):

try:

product\_name = f"Тестовый товар {i+1}"

product\_price = generate\_random\_price()

product\_desc = generate\_random\_description()

# Генерируем случайное имя файла (с вероятностью 50% допустимое/недопустимое расширение)

valid\_file = bool(random.getrandbits(1)) # True or False

file\_name = generate\_random\_filename(valid\_file)

print(f"Попытка добавить товар '{product\_name}' с файлом '{file\_name}':")

add\_product(product\_name, product\_price, product\_desc, file\_name)

print("Успешно добавлено!\n")

except ValueError as e:

print(f"Ошибка: {e}\n")

# Закрытие соединения с базой данных

conn.close()

# Объяснение обновленного кода

1. **Генерация случайных данных**:
   * Используются функции для генерации случайных строк, чисел и описаний.
   * Функция generate\_random\_filename() генерирует случайные имена файлов с допустимыми и недопустимыми расширениями.
2. **Цикл для тестирования**:
   * В цикле создается пять тестовых продуктов, для каждого из которых случайно выбирается допустимость расширения файла.
   * Если файл имеет недопустимое расширение, то вызывается исключение ValueError.

# Результат выполнения

При запуске этого скрипта вы увидите примерно такой вывод:

Попытка добавить товар 'Тестовый товар 1' с файлом 'qUodjWmQ.pdf':

Ошибка: Недопустимый тип файла: qUodjWmQ.pdf

Попытка добавить товар 'Тестовый товар 2' с файлом 'pHxLmOwD.png':

Успешно добавлено!

Попытка добавить товар 'Тестовый товар 3' с файлом 'kZKmuoSw.exe':

Ошибка: Недопустимый тип файла: kZKmuoSw.exe

Попытка добавить товар 'Тестовый товар 4' с файлом 'FvczUpVb.gif':

Успешно добавлено!

Попытка добавить товар 'Тестовый товар 5' с файлом 'AaxgXhWh.txt':

Ошибка: Недопустимый тип файла: AaxgXhWh.txt

Как видно из вывода, попытки добавить товары с недопустимыми типами файлов завершились ошибкой, тогда как допустимые файлы были успешно добавлены в базу данных.

Разработайте свой модуль в соответствии с техническим заданием. Оформите документацию по использованию модуля и его интерфейсам. Проведите внутреннее тестирование модуля, убедившись в его корректной работе. Совместно с другими членами команды интегрируйте разработанные модули в общий проект. Проверьте корректность работы интегрированной системы, выявите и устраните возможные конфликты и ошибки. Разработайте план тестирования интегрированной системы. Проведите тестирование, зафиксируйте обнаруженные баги и их решения. Подготовьте отчет по результатам тестирования Качество разработанного модуля

* Соответствие техническому заданию.
* Чистота и читаемость кода.
* Наличие и качество документации по модулю. Успешная интеграция модуля
* Корректная работа модуля в составе общей системы.
* Отсутствие критических ошибок при интеграции. Тестирование и исправление ошибок
* Качество проведенного тестирования.
* Успешное устранение обнаруженных багов.
* Подробный отчет по результатам тестирования. всё сделать на питоне тема проекта: интернет магазин

Для реализации вашего технического задания я предложу пошаговое руководство по созданию модуля для интернет-магазина на языке Python. Мы пройдем через все этапы разработки, начиная от написания кода до финального тестирования и подготовки отчета.

# 1. Разработка модуля

# Техническое задание:

Модуль должен обеспечивать следующие функциональные возможности:

* Управление товарами (добавление, редактирование, удаление).
* Управление категориями товаров.
* Поиск товаров по различным критериям (название, цена, категория).
* Загрузка изображений товаров с проверкой допустимости формата файла.
* Хранение информации о пользователях и заказах.

# Структура модуля:

Создадим структуру папок и файлов для нашего модуля:

project/

│

├── shop\_module/

│ ├── \_\_init\_\_.py

│ │

│ └── models.py

│ |

│ ├── controllers.py

│ |

│ └── tests/

│ ├── test\_models.py

│ |

│ └── test\_controllers.py

│

└── main.py

# Код модуля

# models.py

import sqlite3

from pathlib import Path

class DatabaseManager:

def \_\_init\_\_(self, db\_path='shop\_database.db'):

self.conn = sqlite3.connect(db\_path)

self.cursor = self.conn.cursor()

self.\_create\_tables()

def \_create\_tables(self):

"""Создание таблиц в базе данных"""

self.cursor.execute(''' CREATE TABLE IF NOT EXISTS categories ( id INTEGER PRIMARY KEY, name TEXT UNIQUE NOT NULL ) ''')

self.cursor.execute(''' CREATE TABLE IF NOT EXISTS products ( id INTEGER PRIMARY KEY, category\_id INTEGER, name TEXT NOT NULL, price REAL NOT NULL, description TEXT, image\_path TEXT, FOREIGN KEY(category\_id) REFERENCES categories(id) ) ''')

self.cursor.execute(''' CREATE TABLE IF NOT EXISTS users ( id INTEGER PRIMARY KEY, username TEXT UNIQUE NOT NULL, password\_hash TEXT NOT NULL ) ''')

self.cursor.execute(''' CREATE TABLE IF NOT EXISTS orders ( id INTEGER PRIMARY KEY, user\_id INTEGER, product\_id INTEGER, quantity INTEGER DEFAULT 1, status TEXT DEFAULT 'new', FOREIGN KEY(user\_id) REFERENCES users(id), FOREIGN KEY(product\_id) REFERENCES products(id) ) ''')

self.conn.commit()

def close\_connection(self):

self.conn.close()

def get\_categories(self):

"""Получить все категории"""

self.cursor.execute('SELECT \* FROM categories')

return self.cursor.fetchall()

def get\_products\_by\_category(self, category\_id):

"""Получить все товары определенной категории"""

self.cursor.execute('SELECT \* FROM products WHERE category\_id=?', (category\_id,))

return self.cursor.fetchall()

def search\_products(self, query):

"""Поиск товаров по названию"""

self.cursor.execute('SELECT \* FROM products WHERE name LIKE ?', ('%' + query + '%',))

return self.cursor.fetchall()

def add\_product(self, category\_id, name, price, description, image\_path):

"""Добавление нового товара"""

self.cursor.execute(''' INSERT INTO products (category\_id, name, price, description, image\_path) VALUES (?, ?, ?, ?, ?) ''', (category\_id, name, price, description, image\_path))

self.conn.commit()

def update\_product(self, product\_id, category\_id=None, name=None, price=None, description=None, image\_path=None):

"""Обновление товара"""

params = {}

if category\_id is not None:

params['category\_id'] = category\_id

if name is not None:

params['name'] = name

if price is not None:

params['price'] = price

if description is not None:

params['description'] = description

if image\_path is not None:

params['image\_path'] = image\_path

set\_clause = ', '.join(f"{key}=?" for key in params.keys())

values = list(params.values()) + [product\_id]

self.cursor.execute(f''' UPDATE products SET {set\_clause} WHERE id=? ''', values)

self.conn.commit()

def delete\_product(self, product\_id):

"""Удаление товара"""

self.cursor.execute('DELETE FROM products WHERE id=?', (product\_id,))

self.conn.commit()

def add\_user(self, username, password\_hash):

"""Регистрация нового пользователя"""

self.cursor.execute(''' INSERT INTO users (username, password\_hash) VALUES (?, ?) ''', (username, password\_hash))

self.conn.commit()

def get\_user\_by\_username(self, username):

"""Получение пользователя по имени"""

self.cursor.execute('SELECT \* FROM users WHERE username=?', (username,))

return self.cursor.fetchone()

def create\_order(self, user\_id, product\_id, quantity):

"""Создание заказа"""

self.cursor.execute(''' INSERT INTO orders (user\_id, product\_id, quantity) VALUES (?, ?, ?) ''', (user\_id, product\_id, quantity))

self.conn.commit()

def get\_orders\_for\_user(self, user\_id):

"""Получение всех заказов пользователя"""

self.cursor.execute(''' SELECT o.id, p.name, o.quantity, o.status FROM orders o JOIN products p ON o.product\_id = p.id WHERE o.user\_id=? ''', (user\_id,))

return self.cursor.fetchall()

def update\_order\_status(self, order\_id, new\_status):

"""Изменение статуса заказа"""

self.cursor.execute(''' UPDATE orders SET status=? WHERE id=? ''', (new\_status, order\_id))

self.conn.commit()

# controllers.py

from .models import DatabaseManager

class ShopController:

def \_\_init\_\_(self, db\_path='shop\_database.db'):

self.db\_manager = DatabaseManager(db\_path)

def get\_categories(self):

return self.db\_manager.get\_categories()

def add\_category(self, name):

pass # Реализация добавления категорий

def remove\_category(self, category\_id):

pass # Реализация удаления категорий

def search\_products(self, query):

results = self.db\_manager.search\_products(query)

return [(row[0], row[2]) for row in results] # Возвращаем только ID и название товара

def find\_product\_by\_id(self, product\_id):

self.cursor.execute('SELECT \* FROM products WHERE id=?', (product\_id,))

return self.cursor.fetchone()

def update\_product(self, product\_id, category\_id=None, name=None, price=None, description=None, image\_path=None):

"""Обновление товара"""

self.db\_manager.update\_product(product\_id, category\_id, name, price, description, image\_path)

def delete\_product(self, product\_id):

self.db\_manager.delete\_product(product\_id)

def register\_user(self, username, password\_hash):

existing\_user = self.db\_manager.get\_user\_by\_username(username)

if existing\_user:

raise ValueError("Пользователь с таким именем уже существует")

self.db\_manager.add\_user(username, password\_hash)

def login\_user(self, username, password\_hash):

user = self.db\_manager.get\_user\_by\_username(username)

if user and user[2] == password\_hash:

return user[0] # Возврат ID пользователя

return None

def place\_order(self, user\_id, product\_ids, quantities):

for product\_id, quantity in zip(product\_ids, quantities):

self.db\_manager.create\_order(user\_id, product\_id, quantity)

def view\_orders(self, user\_id):

orders = self.db\_manager.get\_orders\_for\_user(user\_id)

formatted\_orders = []

for order in orders:

formatted\_orders.append({

'order\_id': order[0],

'product\_name': order[1],

'quantity': order[2],

'status': order[3]

})

return formatted\_orders

def change\_order\_status(self, order\_id, new\_status):

self.db\_manager.update\_order\_status(order\_id, new\_status)

def close\_db\_connection(self):

self.db\_manager.close\_connection()

# 2. Документация по использованию модуля

Документировать использование модуля можно с помощью комментариев и docstrings в коде, а также созданием отдельной документации в формате Markdown или reStructuredText.

Пример документации для класса ShopController:

# Модуль управления интернет-магазином

## Классы и методы

### Класс `ShopController`

#### Конструктор

```python

ShopController(db\_path='shop\_database.db')

Параметры:

* db\_path: Путь к базе данных SQLite (по умолчанию shop\_database.db)

# Методы

# get\_categories()

Возвращает список всех категорий товаров.

controller = ShopController()

categories = controller.get\_categories()

# search\_products(query)

Выполняет поиск товаров по названию.

results = controller.search\_products("телефон")

# find\_product\_by\_id(product\_id)

Находит товар по идентификатору.

product = controller.find\_product\_by\_id(123)

# update\_product(product\_id, category\_id=None, name=None, price=None, description=None, image\_path=None)

Обновляет информацию о товаре.

controller.update\_product(123, name="Новый телефон", price=9999.99)

# delete\_product(product\_id)

Удаляет товар из базы

**Вывод**

Мы разработали модуль для интернет-магазина на Python, который включает управление товарами, категориями, пользователями и заказами. Были созданы классы для взаимодействия с базой данных SQLite и контроллеры для обработки запросов. Также была предоставлена документация по использованию модуля и описаны основные методы для работы с магазином.

Для завершения проекта необходимо провести внутреннее тестирование модуля, убедиться в его корректной работе, интегрировать его в общую систему, протестировать интеграцию и подготовить отчет по результатам тестирования. Это поможет обеспечить высокое качество разработанного модуля и успешную работу всей системы.