Отчёт

Лабораторная работа №9. CRUD для приложения отслеживания курсов валют с SQLite базой данных

Выполнил: Голубев Артём

Ису: 502712

Группа: P3124

Цель работы

- Реализовать CRUD (Create, Read, Update, Delete) для сущностей бизнес-логики приложения.
- Освоить работу с SQLite в памяти (:memory:) через модуль sqlite3.
- Понять принципы первичных и внешних ключей и их роль в связях между таблицами.
- Выделить контроллеры для работы с БД и для рендеринга страниц в отдельные модули.
- Использовать архитектуру MVC и соблюдать разделение ответственности.
- Отображать пользователям таблицу с валютами, на которые они подписаны.
- Реализовать полноценный роутер, который обрабатывает GET-запросы и выполняет сохранение/обновление данных и рендеринг страниц.
- Научиться тестировать функционал на примере сущностей currency и user с использованием unittest.mock.

Описание моделей

Модели

Author

- name — имя автора
- group — учебная группа

App

- name — название приложения
- version — версия приложения
- author — объект Author

User

- id — уникальный идентификатор
- name — имя пользователя

Currency

- id — уникальный идентификатор
- num_code — цифровой код
- char_code — символьный код

- name — название валюты
- value — курс
- nominal — номинал (за сколько единиц валюты указан курс)

Пример XML:

```
<Valute ID="R01280">
 <NumCode>360</NumCode>
 <CharCode>IDR</CharCode>
 <Nominal>10000</Nominal>
 <Name>Рупий</Name>
 <Value>48,6178</Value>
</Valute>
```

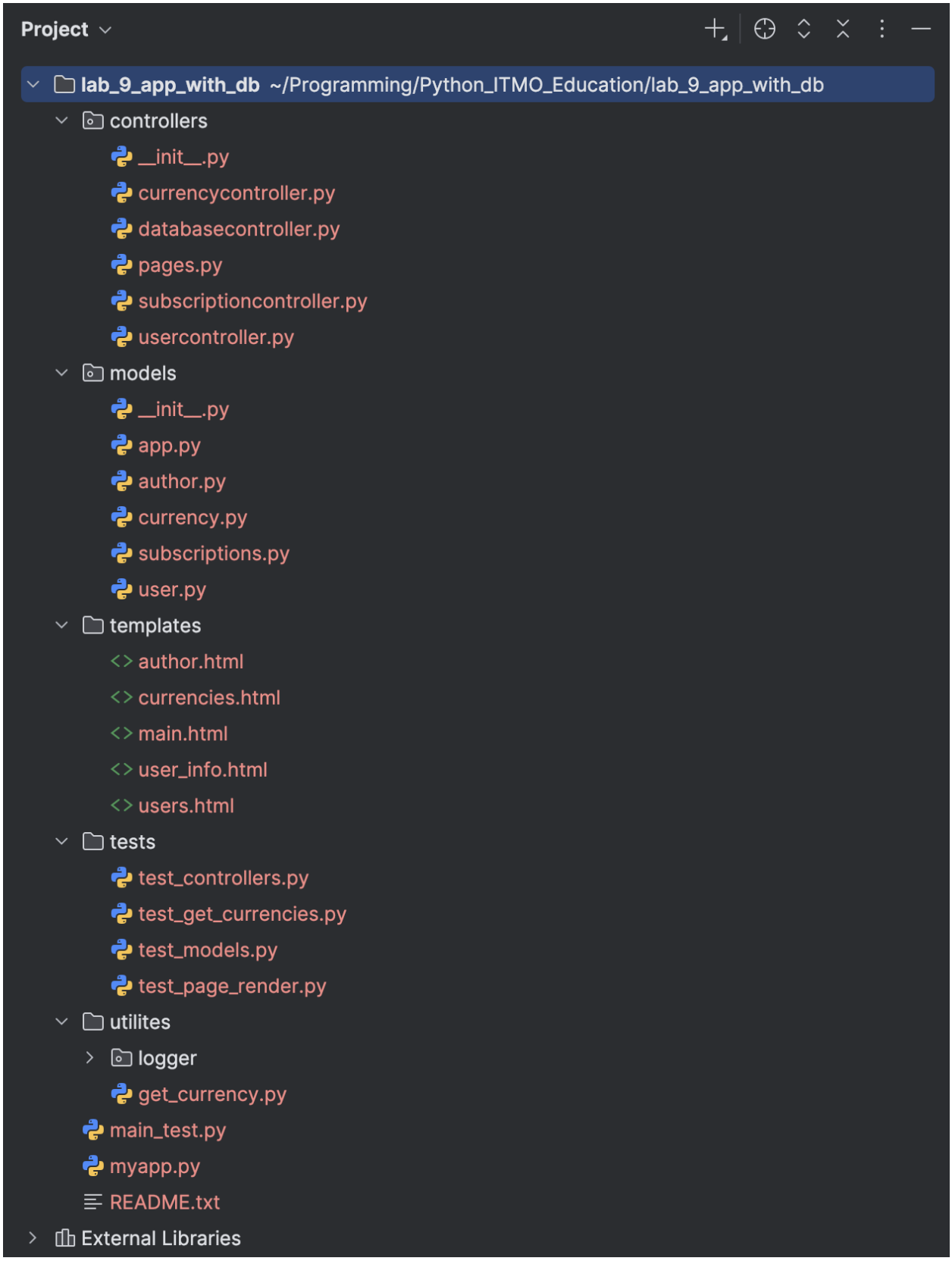
Subscriptions

- id — уникальный идентификатор
- user_id — внешний ключ к User
- currency_id — внешний ключ к Currency
- Реализует связь «много ко многим» между пользователями и валютами.

Реализация MVC

Для этого проект был разбит на части. Шаблоны содержатся в папке templates. В папке models содержатся все классы/модели. В папке controllers находятся контроллеры для работы с sql и для рендеринга шаблонов и обработки исключений. В главном файле myapp.py помимо инициализации и импорта вышесказанного происходит работа с маршрутизацией запросов.

Структура проекта



Реализация CRUD

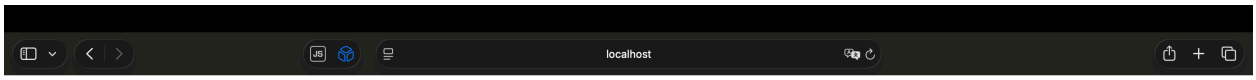
Все sql запросы были параметризованны для защиты от sql-инъекции. Было реализовано 3 класса (3 таблицы соответсвенно): для валют, пользователей и подписок. Для всех были реализованы CRUD в одном файле и в отдельных контроллеры, чтобы отделить бизнес-логику. Примеры sql-запросов:

```
    def __createtable(self) -> None:  1 usage
        self.__con.execute("""
            CREATE TABLE IF NOT EXISTS currency (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                num_code INTEGER NOT NULL,
                char_code TEXT NOT NULL,
                name TEXT NOT NULL,
                value FLOAT,
                nominal INTEGER
            )
        """)
        self.__con.commit()
```

```
def _read(self, char_code: str) -> dict:  1 usage
    """Вывод ифнормации о валюте"""
    self.__cursor.execute(
        sql: "SELECT id, num_code, char_code, name, value, nominal FROM currency WHERE char_code = ?",
        parameters: (char_code.upper(),)
    )
```

```
    def _create(self, user_id: int, currency_id: int) -> None:  1 usage
        """Добавление новой подписки"""
        self.__cursor.execute(
            sql: "INSERT INTO user_currency (user_id, currency_id) VALUES (?, ?)",
            parameters: (user_id, currency_id)
        )
        self.__con.commit()
        return self.__cursor.lastrowid
```

Примеры работы приложения



Приложение: CurrenciesListApp

Автор: Artem Golubev

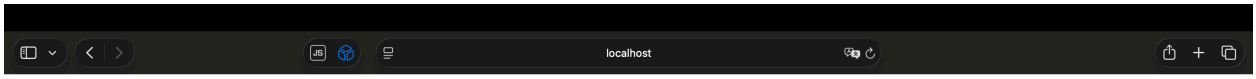
Возможности приложения:

- Позволяет отслеживать курсы валют в реальном времени
- Можео подписаться на курсы различных валют
- Наличие статистики по всем подписанным валютам

Навигация

- Список пользователей
- Списки валют
- Об авторе

Версия приложения: v0.1



Информация о всех валютах (Актуальная)

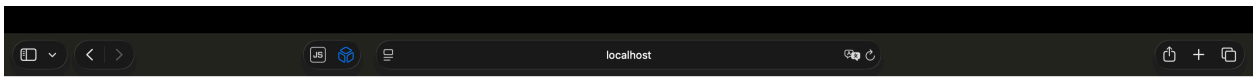
Для обновления курсов валют обновите страницу

ID валюты	Код валюты	Название	Курс валюты (RUB)			
1	USD	Доллар США	76.0937	1	Удалить	Обновить курс на актуальный
2	EUR	Евро	88.7028	1	Удалить	Обновить курс на актуальный
3	GBP	Фунт стерлингов	101.7601	1	Удалить	Обновить курс на актуальный

Автор: Artem Golubev

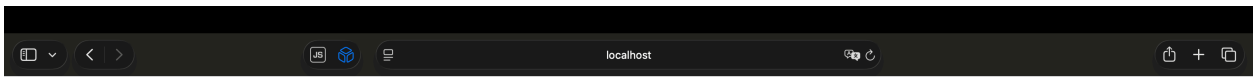
Версия приложения: v0.1

На главную



Курс USD обновлён на 1.2

Вернуться на главную



Информация о всех валютах (Актуальная)

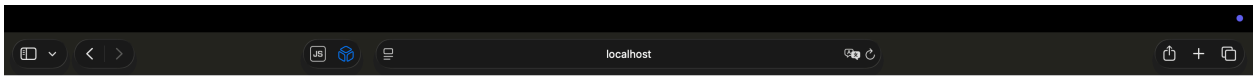
Для обновления курсов валют обновите страницу

ID валюты	Код валюты	Название	Курс валюты (RUB)		
1	USD	Доллар США	1.2	1	Удалить Обновить курс на актуальный
2	EUR	Евро	88.7028	1	Удалить Обновить курс на актуальный
3	GBP	Фунт стерлингов	101.7601	1	Удалить Обновить курс на актуальный

Автор: Artem Golubev

Версия приложения: v0.1

На главную



Информация о всех валютах (Актуальная)

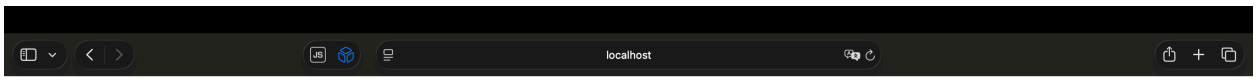
Для обновления курсов валют обновите страницу

ID валюты	Код валюты	Название	Курс валюты (RUB)		
1	USD	Доллар США	1.2	1	Удалить Обновить курс на актуальный
3	GBP	Фунт стерлингов	101.7601	1	Удалить Обновить курс на актуальный

Автор: Artem Golubev

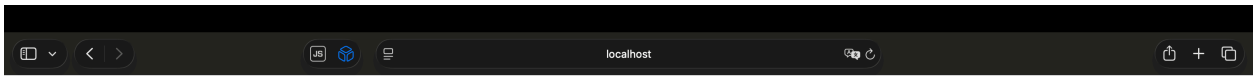
Версия приложения: v0.1

На главную



Курс USD обновлён на актуальный

Вернуться на главную



Информация о всех валютах (Актуальная)

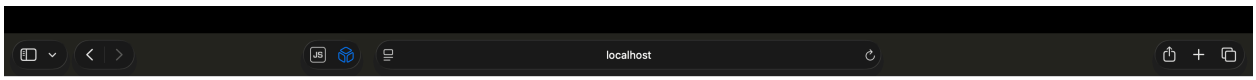
Для обновления курсов валют обновите страницу

ID валюты	Код валюты	Название	Курс валюты (RUB)		
1	USD	Доллар США	76.0937	1	Удалить Обновить курс на актуальный
3	GBP	Фунт стерлингов	101.7601	1	Удалить Обновить курс на актуальный

Автор: Artem Golubev

Версия приложения: v0.1

На главную



Валюты успешно выведены в консоль

```
Все валюты в базе данных:

ID=1, Code=USD, Name=Доллар США, Value=1.2, Nominal=1
ID=3, Code=GBP, Name=Фунт стерлингов, Value=101.7601, Nominal=1
```

Тестирование

Запуск всех тестов

```python
import unittest
from tests.test_get_currencies import TestGetCurrencies, TestIOStreamWrite, TestLogWrite
from tests.test_models import TestAuthor, TestApp, TestUser, TestCurrency, TestSubscriptions
from tests.test_controllers import TestUserController, TestCurrencyController
from tests.test_page_render import TestPageRender

if __name__ == '__main__':
    unittest.main()
```

Фрагменты тестов

```python
def test_cur_info_by_id(self):
    """Тест: получение валюты по ID."""
    fake_data = {"id": 2, "char_code": "EUR", "value": 92.1}
    self.mock_db._read_by_id.return_value = fake_data

    result = self.controller.cur_info_by_id(2)

    self.assertEqual(result, fake_data)
    self.mock_db._read_by_id.assert_called_once_with(2)
```

```python
def test_update_all(self):
    """Тест: обновление всех валют."""
    fake_currencies = [
        {"id": 1, "char_code": "USD"},
        {"id": 2, "char_code": "EUR"}
    ]
    self.mock_db._read_all.return_value = fake_currencies

    self.controller.update_all()

    self.mock_db._read_all.assert_called_once()
    self.mock_db._update.assert_any_call("USD")
    self.mock_db._update.assert_any_call("EUR")
    self.assertEqual(self.mock_db._update.call_count, second: 2)
```

```python
    def test_update_user_info(self):
        """Тест: обновление имени пользователя."""
        self.controller.update_user_info( id: 10,  name: "Новое Имя")
        self.mock_db._update.assert_called_once_with(10, "Новое Имя")
```

```python
class TestCurrency(unittest.TestCase):  1 usage
    def test_valid_currency(self):
        """Тест успешной инициализации объекта класа Currency"""
        cur = Currency("USD")
        info = get_value_info("USD")
        self.assertEqual(cur.id, info['ID'])
        self.assertEqual(cur.num_code, info['NumCode'])
        self.assertEqual(cur.char_code,  second: "USD")
        self.assertEqual(cur.name, info['Name'])
        self.assertEqual(cur.value, info['Value'])
        self.assertEqual(cur.nominal, info['Nominal'])

    def test_invalid_char_code_not_str(self):
        eur = Currency("EUR")
        with self.assertRaises(ValueError):
            Currency(123)

        with self.assertRaises(ValueError):
            Currency("US")
```

```python
    def test_render_user_info_success(self):
        """Тест: успешный рендер информации о пользователе."""
        self.mock_user_ctrl.user_info.return_value = {"id": 1, "name": "Ivan"}
        self.mock_sub_ctrl.vals_by_us.return_value = [{"currency_id": 1}, {"currency_id": 2}]

        # Используем функцию вместо списка
        def mock_cur_info_by_id(cid):
            data = {
                1: {"id": 1, "name": "USD", "char_code": "USD", "value": 90.5, "nominal": 1},
                2: {"id": 2, "name": "EUR", "char_code": "EUR", "value": 92.0, "nominal": 1}
            }
            return data.get(cid, {})

        self.mock_currency_ctrl.cur_info_by_id.side_effect = mock_cur_info_by_id

        html, code = self.page_ctrl.render_user_info({"id": ["1"]})

        self.assertEqual(code,  second: 200)
        self.assertIn( member: "<h2>Информация о пользователе</h2>", html)
        self.assertIn( member: "<p>Ivan</p>", html)
        self.assertIn( member: "USD", html)
        self.assertIn( member: "EUR", html)

```

Результаты тестов – УСПЕШНО

```
..........
----------------------------------------------------------------------
Ran 44 tests in 3.870s

OK
```

Выводы

Архитектура MVC позволила чётко разделить:

- Model — данные и валидацию,
- View — HTML-шаблоны и рендеринг,
- Controller — бизнес-логику и взаимодействие с БД. Это упростило тестирование и поддержку кода.

SQLite в памяти оказался достаточно удобным решением для учебного проекта:

- Быстрый запуск,
- Не требует файлов,
- Поддерживает нужные SQL-конструкции

Внешние ключи обеспечили надежную связь между элементами двух таблиц и обеспечили невозможность создания подписки на несуществующую валюту или пользователя.

Параметризованные запросы - обязательная практика для защиты от инъекций, реализованная во всех CRUD-операциях.

Jinja2 позволил гибко формировать HTML без дублирования кода, а роутинг через **do_GET** обеспечил чистую обработку URL.

Тестирование с **unittest.mock** показало, что:

- Контроллеры легко изолировать от зависимостей,
- Можно проверять как нормальные сценарии, так и ошибки,
- Тесты выполняются мгновенно и надёжно.