Отчёт

Лабораторная работа №8. Клиент-серверное приложение на Python с использованием Jinja2

Выполнил: Голубев Артём

Ису: 502712

Группа: P3124

Цель работы

1. Создать простое клиент-серверное приложение на Python без серверных фреймворков.
2. Освоить работу с HTTPServer и маршрутизацию запросов.
3. Применять шаблонизатор Jinja2 для отображения данных.
4. Реализовать модели предметной области (User, Currency, UserCurrency, App, Author) с геттерами и сеттерами.
5. Структурировать код в соответствии с архитектурой MVC.
6. Получать данные о курсах валют через функцию get_currencies и отображать их пользователям.
7. Реализовать функциональность подписки пользователей на валюты и отображение динамики их изменения.
8. Научиться создавать тесты для моделей и серверной логики.

Описание предметной области

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)

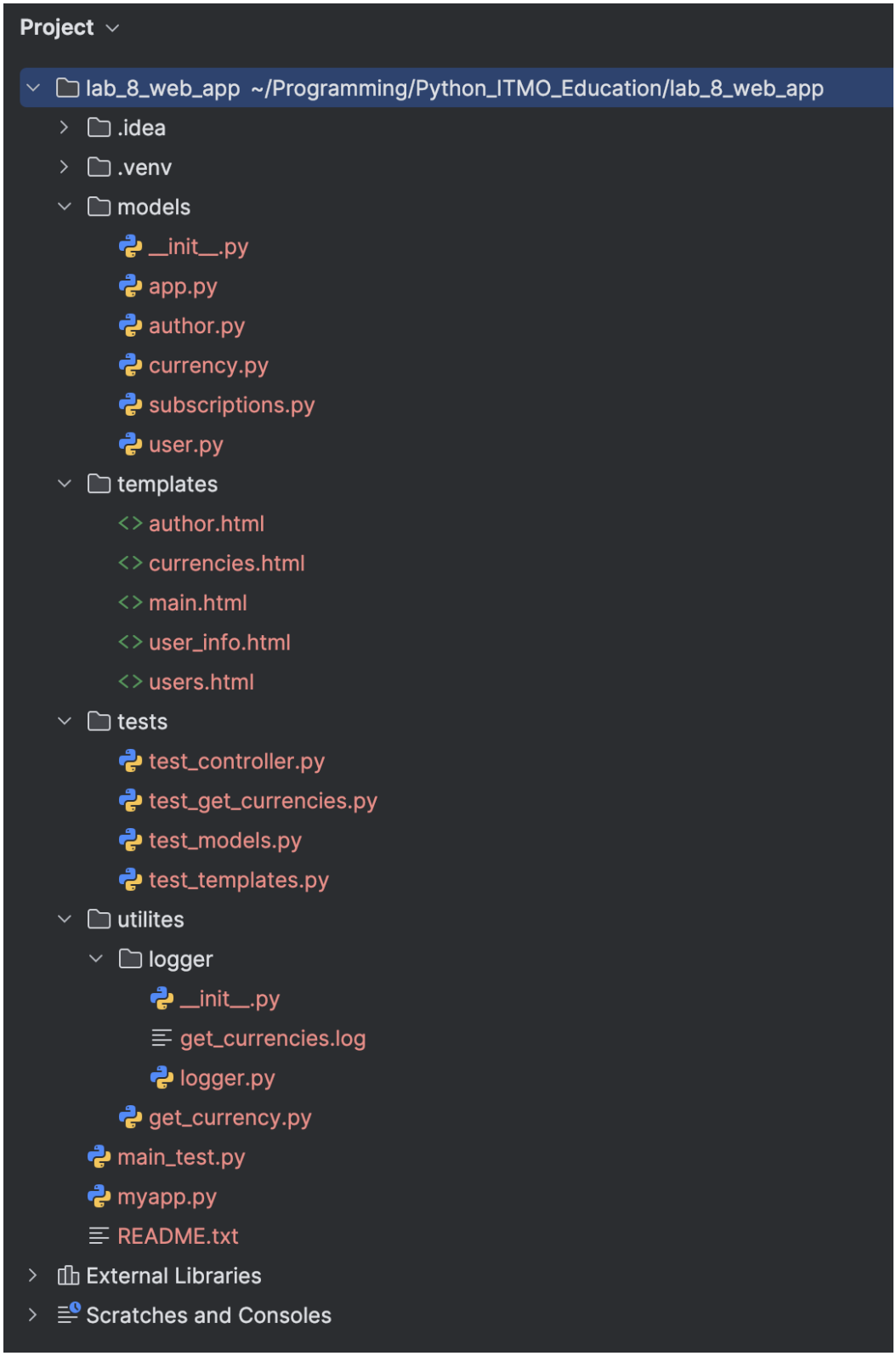
Пример XML:

```
<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.

Структура проекта



Описание реализации

Были реализованы класс (описано в описании предметной области). Все модели реализованы с использованием инкапсуляции:

1. Все атрибуты — приватные (__name, __group и т.д.)
2. Доступ через геттеры (@property)
3. Изменение через сеттеры (@name.setter) с валидацией:
    - Проверка типов (isinstance)
    - Проверка длины строк
    - Проверка положительности чисел
    - И т.п.

Пример:

```python
class Subscriptions():
    """
    Класс для хранения информации о подписках пользователей на валюты
    """
    def __init__(self, id: int, user_id: int, currency_id: list):
        self.id: int = id
        self.user_id: int = user_id
        self.currency_id: list = currency_id

    @property
    def id(self):
        return self.__id

    @id.setter
    def id(self, id: int):
        if type(id) is int and id > 0:
            self.__id = id
        else:
            raise ValueError('Ошибка при задании ID подписки')

    @property
    def user_id(self):
        return self.__user_id

    @user_id.setter
    def user_id(self, user_id: int):
        if type(user_id) is int and user_id > 0:
            self.__user_id = user_id
        else:
            raise ValueError('Ошибка при задании ID пользователя')
```

Так же, для класса Currency были реализованы дополнительные методы для обновления курса валюты и для получения всей информации о валюте в виде списка.

```
92      def update_value(self):
93          """
94          Обновляет значение курса валюты
95          """
96          self.__value = Decimal(str(get_currencies([self.char_code])[self.char_code]))
97
98
99      def get_full_info(self):
100         """
101         Возвращает список с полной информацией о валюте
102         """
103         return self.__id, self.__num_code, self.__char_code, self.__name, float(str(self.__value)), self.__nominal
104
```

Хранение денежных данных реализуется через библиотеку decimal. Это позволяет хранить данные и в будущем делать с ними различные операции без переживания за корректность их выполнения (из-за особенностей хранения float данных). На данный момент класс по запросу возвращает данные в типе float для удобства отображения их на сайте (пока нужды делать какие-то операции с деньгами нет).

Маршрутизация и обработка запросов

Реализована в классе SimpleHTTPRequestHandler:

1. Используется urlparse(self.path) для разбора пути
2. parse_qs(parsed.query) — для извлечения параметров (/user?id=1)
3. Поддерживаются маршруты:
    - / — главная страница
    - /users — список пользователей
    - /user?id=... — профиль пользователя
    - /currencies — курсы валют (с обновлением через get_currencies)
    - /author — информация об авторе

Обработка ошибок:

1. Некорректный id пользователя -> HTTP 404
2. Неизвестный маршрут -> HTTP 404

Шаблоны jinja2

Инициализация Environment один раз при старте и шаблонов:

```
118     # Инициализация окружения
119     env = Environment(
120         loader=PackageLoader( package_name: "myapp",  package_path: "templates"),
121         autoescape=select_autoescape())
122
123     # Инициализация шаблонов
124     main_page = env.get_template("main.html")
125     users_page = env.get_template("users.html")
126     user_info_page = env.get_template("user_info.html")
127     currencies_page = env.get_template("currencies.html")
128     author_page = env.get_template("author.html")
```

Преимущества:

- Объект Environment хранит кэш шаблонов и настройки Jinja2.
- Инициализация один раз при старте приложения повышает производительность, шаблоны не подгружаются заново при каждом запросе.
- Позволяет многократно рендерить шаблоны, передавая разные данные (модели, списки, переменные).

Шаблоны передают данные через render():

```
132     main = main_page.render(title='Главная страница', myapp=app_info.name,
133                             version=app_info.version, author=app_info.author,
134                             navigation=[{'name': 'Список пользователей', 'link': '/users'},
135                                         {'name': 'Список валют', 'link': '/currencies'},
136                                         {'name': 'Об авторе', 'link': '/author'}])
137
138     author = author_page.render(title='Об авторе', version=app_info.version,
139                                 author=app_info.author, group=main_author.group)
```

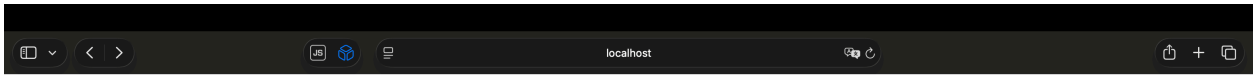
Интеграция функции получения курсов валют

Сама функция реализована в utilites/get_currency.py. Она полностью аналогична функции из Лабораторной работы №7. Помимо этого дополнительно была реализована функция для получения полной информации о валюте в виде словаря

```python
@logger(handle=log)
def get_value_info(currency_code: str, url: str = "https://www.cbr-xml-daily.ru/daily_json.js") -> dict:
    """
    Получает полную информацию о валюте с API Центробанка России.
    Args:
        currency_code (str): Символьный код валюты.
        url (str): Get запрос для получения курса валют (по умол. https://www.cbr-xml-daily.ru/daily_json.js).
    Returns:
        dict: Словарь, где ключи - параметры валюты, а значения - их значения.
              Возвращает None в случае ошибки запроса.
    """

    response = requests.get(url)
    response.raise_for_status()
    data = response.json()
    valute = {}

    if "Valute" in data:
        if currency_code in data["Valute"]:
            valute = data["Valute"][currency_code]

    return valute
```

Данные функции были интегрированы в класс Currency. Во-первых, для инициализации объекта класса нужно передать его символьный код (char_code). Остальные данные берутся из функции get_value_info. Также, функция используется для обновления значения курса валюты. Это используется на странице со всеми валютами, при каждом открытии/обновлении страницы курс валюты обновляется.

Примеры работы приложения



Приложение: CurrenciesListApp

Автор: Artem Golubev

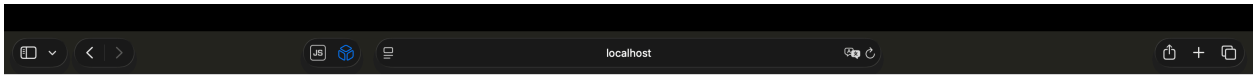
Возможности приложения:

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

Навигация

- Список пользователей
- Список валют
- Об авторе

Версия приложения: v0.1



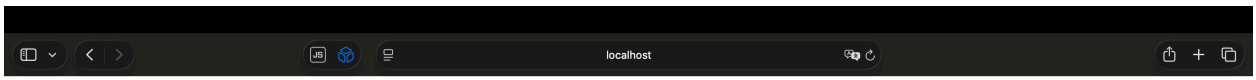
Об Авторе

Автор: Artem Golubev

Учебная группа: P3124

Версия приложения: v0.1

На главную



Список пользователей:

- id=1) Ivan Ivanov

  Подписки
- id=2) Vladimir Putin

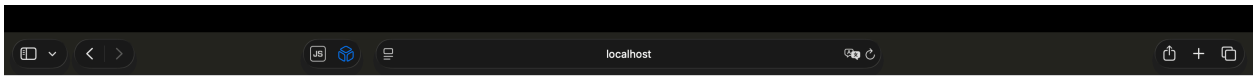
  Подписки
- id=3) Cristiano Ronaldo

  Подписки

Автор: Artem Golubev

Версия приложения: v0.1

На главную



Информация о всех валютах (Актуальная)

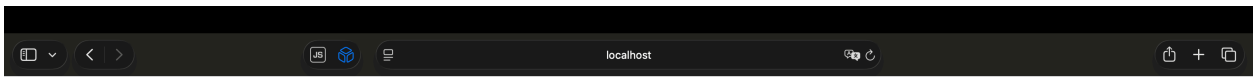
Для обновления курсов валют обновите страницу

ID валюты	Код валюты	Название	Курс валюты (RUB)
R01235	USD	Доллар США	76.0937
R01239	EUR	Евро	88.7028
R01035	GBP	Фунт стерлингов	101.7601

Автор: Artem Golubev

Версия приложения: v0.1

На главную



Информация о пользователе с id=1

Имя: Ivan Ivanov

id подписки: 1

Валюты в подписке: ['USD', 'EUR']

ID валюты	Код валюты	Название	Курс валюты
R01235	USD	Доллар США	76.0937
R01239	EUR	Евро	88.7028

Ко всем пользователям

Автор: Artem Golubev

Версия приложения: v0.1

На главную

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_controller import TestController
from tests.test_templates import TestTemplates

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

Фрагменты тестов

```python
    def test_root_page(self):
        """Тест главной страницы"""
        content, code = self.make_request('/')
        self.assertEqual(code, second: 200)
        self.assertIn( member: 'Главная страница', content)
        self.assertIn( member: 'CurrenciesListApp', content)

    def test_author_page(self):
        """Тест страницы об авторе"""
        content, code = self.make_request('/author')
        self.assertEqual(code, second: 200)
        self.assertIn( member: 'Об авторе', content)
        self.assertIn( member: 'P3124', content)  # группа автора
```

```python
    def test_connection_error(self):
        """
        Проверяет правильность вызова ConnectionError
        """

        currency_list = ['USD', 'EUR']

        with self.assertRaises(requests.exceptions.ConnectionError):
            get_currencies(currency_list, url='https://www.notcorrectapi.ru')

    def test_value_error(self):
        """
            Проверяет правильность вызова ValueError
        """

        currency_list = ['USD', 'EUR']

        with self.assertRaises(ValueError):
            get_currencies(currency_list, url='https://google.com')
```

```python
    def test_valid_subscription(self):
        sub = Subscriptions(1, 1, ["USD", "EUR"])
        self.assertEqual(sub.id,  second: 1)
        self.assertEqual(sub.user_id,  second: 1)
        self.assertEqual(sub.currency_id,  second: ["USD", "EUR"])

    def test_invalid_currency_id_not_list(self):
        with self.assertRaises(ValueError):
            Subscriptions(1, 1, "USD")

        with self.assertRaises(ValueError):
            Subscriptions(1, 1, [])
```

```python
    def test_users_template_renders(self):
        """тест рендера станицы с пользователями"""
        template = self.env.get_template('users.html')
        users = [
            User(1, 'Ivan Ivanov'),
            User(2, 'Artem Artemov')
        ]
        author = Author("Artem Golubev", "P3124")
        app = App("CurrenciesListApp", "v0.1", author.name)

        result = template.render(
            title='Список пользователей',
            author=app.author,
            version=app.version,
            navigation=users
        )

        self.assertIn( member: 'Список пользователей', result)
        self.assertIn( member: 'Ivan Ivanov', result)
        self.assertIn( member: 'Artem Artemov', result)
```

Результаты тестов – УСПЕШНО

```
..........
----------------------------------------------------------------------
Ran 31 tests in 44.462s

OK
```

Выводы

Проблемы

В ходе работы единственная проблема, которая возникла, это работа с относительными путями к файлам. Но после полного анализа структуры и правильного импорта файлов и модулей проблема была решена

Применение MVC

1. Model: классы в models/ — чистая бизнес-логика, без HTTP/UI.
2. View: шаблоны в templates/ — только отображение.
3. Controller: SimpleHTTPRequestHandler — маршрутизация + вызов моделей + рендеринг.

Новые знания

1. Научился работать с встроенным HTTPServer без фреймворков.
2. Освоил базово Jinja2. Оказался правда удобный инструмент для безопасного рендеринга.
3. Понял, как интегрировать внешние API в веб-приложение (как продолжения лабораторной работы №7).
4. Углубил понимание принципов ООП и их реализацию в Python.