Dokumentacja

Realizacja w portalu kalkulatora ”simple”

(dane wprowadzane z klawiatury).

Krzysztof Opalski

Spis treści

Część I
Ogólne informacje o kalkulatorze

- napisany jako duży projekt w ramach seminarium Finanse Obliczeniowe
- opiekun: prof. Andrzej Palczewski
- dostępny w internecie pod adresem: `http://weboctave.mimuw.edu.pl/mathfinance2`
- cel: obliczanie cen różnych instrumentów finansowych (głównie opcji)
- kalkulator posiada dwie części: obliczenia dla prawdziwych danych (z plików csv, wymagane logowanie) oraz dla danych z klawiatury (bez logowania, uproszczone)
- wykorzystywane technologie: python, django, octave, hg, javascript

1 Struktura projektu

Objaśnienie:

Pogrubioną czcionką zaznaczyłem części obsługujące kalkulator simple (wyłącznie)

Kursywą - części wspólne, które ten kalkulator istotnie wykorzystuje (w związku z tym były modyfikowane)

- `.hg/` - katalog tworzony przez system kontroli wersji
- `mf/` - pliki pythona
  - ...
  - `octave.py` - *funkcje pomocnicze, komunikacja octave z pythonem*
  - `forms_simple.py` **- formularze, dane, teksty pomocy do kalkulatora simple**
  - `views_simple.py` **- główny plik obsługujący kalkulator simple**
- `octavecodes/` - pliki octave
  - `fixed/` - rynek fixed income
  - `fx/` - rynek fx
  - `simple/` **- funkcje octave kalkulatora simple (rynki fx, equity, fixed income)**
- `static/` - pliki statyczne
  - `admin/` - pliki dla panelu administratora
  - `css/` - arkusze stylow

  - `js/` - *skrypty javascript*
  - `models/` - pliki statyczne dla danego modelu (np. domyslne csv)
- `/views` - pliki html (w podziale na części kalkulatora)
  - `fixed/`
  - `fx/`
  - `registration/`
  - `simple/`
- `/userdata` - plik z danymi użytkowników
- `manage.py` - skrypt wykonywalny służący do uruchomienia portalu i zarządzania nim
- `settings.py` - plik z ogólnymi ustawieniami django
- `urls.py` - *plik tłumaczący adresy URL na funkcje pythona*

2 Schemat działania kalkulatora simple

- użytkownik: wybiera na głównej stronie simple calculator (nie musi być zalogowany)
- python: wyświetla główną stronę `/simple` z wyborem rynku
- użytkownik: wybiera rynek (fx, equity, fixed income)
- python: wyświetla stronę `/simple/[rynek]` z listą instrumentów w podziale na kategorie
- użytkownik: wybiera konkretny instrument do wyceny
- python: uruchamia główną funkcję - calculate (jest w pliku `mf/views_simple.py`), która wyświetla stronę `/simple/[rynek]/[instrument]`
  - wybierany jest odpowiedni formularz do wprowadzania danych i pokazywany użytkownikowi - po lewej stronie
  - po prawej jest puste miejsce w którym pojawią się wyniki.
- użytkownik: wprowadza dane do formularza, klika przycisk
- javascript: skrypt `static/js/reload.js` wczytuje obrazek ładowania („kręcące się słoneczko”), zmienia kursor (na kursor oczekiwania) oraz przeładowuje stronę
- python: dane wracają do funkcji calculate

  - jeśli formularz jest wypełniony prawidłowo wczytujemy te dane do pythona
  - następnie wywołujemy obliczenia - służy do tego funkcja `mf_exec_anonymous_code` z pliku `mf/octave.py`. Tworzony jest nowy proces octave i w nim wywoływana odpowiednia funkcja (działa to tak samo jakbyśmy ten tekst wpisali w konsoli octave) z właściwymi parametrami

- octave: w każdym przypadku ywoływany skrypt jest nakładką na właściwą funkcję obliczeniową. najpierw wczytuje wymagane funkcje, wywołuje właściwe obliczenia a następnie prezentuje wyniki (wypisuje na stdout w formacie tabelki html). Ewentualne błędy są wypisywane na stderr.
- python: wynik z octave'a jest tekstem (złożonym ze strumieni stderr i stdout). Formatujemy go trochę żeby ładniej wyglądał i wypisujemy na stronę (po prawej stronie)

Część II
Szczegóły implementacji

3 Plik `urls.py`

Odpowiedzialny za wywołanie kontrolera odpowiadającemu adresowi URL. W tym przypadku istotne są wpisy:

- `url(r'^simple/$', 'mf.views.index_simple'),`
- `url(r'^simple/(?P<market>\w+)/$', 'mf.views_simple.index'),`
- `url(r'^simple/(?P<market>\w+)/(?P<function>\w+)/$', 'mf.views_simple.calculate'),`

4 Kontrolery pythona

Kontroler jest funkcją pythona wywoływaną przez dany adres url. Powinien zwrócić stronę html, do jej wygenerowania można użyć następujących funkcji: `HttpResponse('czysty tekst')`, `render_to_response('nazwapliku.html', slownik)`, `HttpResponseRedirect` — przekierowanie, `Http404` — zwrocenie bledu 404. Funkcja `render_to_response` została "opakowana" w funkcję `mf_render(request, view, dictionary)`, która znajduje się w pliku `/mf/views.py`.

Zasadnicza część kodu kalkulatora simple jest w pliku `mf/views_simple.py`, oprócz tego niektóre elementy (formularze, lista funkcji itp.) są przechowywane w pliku `mf/forms_simple.py` oraz funkcja `mf_exec_anonymous_code` w pliku `mf/octave.py`. Prosty skrypt javascript odpowiedzialny za bajery graficzne (obrazek symbolizujący obliczanie i zmianę kursora) jest w pliku `/static/js/reload.js`

Opis poszczególnych funkcji

- `mf/views_simple.py`, funkcja `calculate_strikes`: funkcja pomocnicza, dla danej opcji zwraca ilość strike'ów
- `mf/views_simple.py`, funkcja `is_barrier`: funkcja pomocnicza, dla danej opcji zwraca czy jest barierowa czy nie
- `mf/views_simple.py`, funkcja `choose_form`: funkcja pomocnicza, dla danego rynku i instrumentu zwraca odpowiedni formularz
- `mf/views_simple.py,` funkcja `index`: wyświetla główną stronę danego rynku, ładuje odpowiednie helpy, generuje z nich obiekty html (przy pomocy funkcji help_javascript z pliku mf/forms.py) i przekazuje do pliku html
- `mf/views_simple.py`, funkcja `calculate`: najważniejsza część modułu simple. Na początku wczytywane są odpowiednie helpy (analogicznie jak powyżej, z tym że generujemy zarówno help do rynku jak i do konretnej funkcji). Potem wybieramy odpowiedni formularz (funkcja `choose_form`) i wyświetlamy użytkownikowi. Dalsza część jest uruchamiana jeśli użytkownik wprowadzi dane i naciśnie przycisk: kontroluje to linia `if request.method == 'POST'`. `if load_forms.is_valid()` sprawdza czy formularz jest wypełniony (sprawdza też czy w polach liczbowych są liczby, ale nie sprawdza czy są dodatnie, ujemne itp.). Dalsza rozbudowana instrukcja warunkowa tworzy komendę octave (skleja ją z odpowiednich kawałków tekstu i danych) i przekazuje ją do wykonania
- `mf/octave.py`, funkcja `mf_exec_anonymous_code`: dostaje jako argument tekst komendy do wykonania w octave. Dokleja do niej początek (`octave --no-history (...)`) i całość wywołuje w bashu, przechwytując strumienie wyjściowe. W odróżnieniu od funkcji wykorzystywanej w kalkulatorze real funkcja nie używa plików. Dalszy ciąg funkcji to kosmetyka, formatowanie — strumień błędów wyświetlamy klasą `notify_no_background`, strumień stdout tabelą klasy `output_table`. Dodatkowo zakomentowujemy początek outputu (`"GNU Octave, version ..."`) zostawiając same wyniki.
- `static/js/reload.js` – bardzo prosty skrypt, pełni funkcje jedynie kosmetyczne. Jako argument dostaje nazwę diva oraz stronę. Skrypt zmienia kursor na kursor oczekiwania, wstawia na obecnej stronie do diva podanego jako argument obrazek ładowania a następnie ładuje stronę wskazaną jako argument funkcji. Ładowanie następuje z opóźnieniem, bez tego w niektórych przeglądarkach nie wczytywał się obrazek.

5 Pliki html (template'y)

Charakterystyczna dla django jest hierarchiczna struktura templatów, wykorzystuję ją także tutaj. Dzięki temu powtarzające elementy kodu piszemy tylko raz, potem odwołując się do wspólnego pliku. Ponadto, możemy sterować tym co jest wyświetlane w htmlu na podstawie słownika przekazanego przez kontroler przy wywołaniu wyświetlenia.

- `{% extends "xxx.html" %}` - renderuje plik xxx.html, podmieniając w nim zawartości bloków zadeklarowanych w xxx.html tymi zadeklarowanymi w danym pliku w następujący sposób
  - `{% block nazwa_bloku %}`
  - `zawartosc bloku`
  - `{% endblock %}`
- `{{ zmienna_ze_slownika_do_wyswietlenia }}` — wstawiamy do htmla zmienną
- ponadto dostępne są instrukcje sterujące - m.in. `{% if warunek %}` `{% else %}` `{% endif %}`. przy czym warunkiem można zastąpić identyfikatorem obiektu przekazywanym przez słownik (wtedy nie podanie obiektu będzie wyewaluowane do fałszu)
- `{% for element in lista %} powtorz dla kazdego element z lista {% endfor %}` — jeśli przekazaliśmy do strony listę, możemy ją obsłużyć w pętli. W ten sposób są obsługiwane zakładki pomocy.

Poszczególne pliki

- `views/index_simple`: strona główna kalkulatora simple, zawiera jedynie linki do poszczególnych linków. Wykorzystuje plik `views/layout.html`, w którym zapisane są ogólne informacje o wyglądzie strony (ten plik jest bezpośrednio lub pośrednio wykorzystywany przez wszystkie widoki html, jest wspólny dla całego kalkulatora, nie tylko części simple)
- `views/simple/header.html`: plik z częścią wspólną wszystkich podstron w kalkulatorze simple: górny pasek z linkami do pomocy i do zmiany rynku. Również odwołuje się bezpośrednio do layout.html
- `views/simple/[rynek].html`: strona danego rynku, zawiara tylko spis dostępnych instrumentów z linkami. Wykorzystuje plik header.html
- `views/simple/[rynek]_calculate.html`: strona konkretnej funkcji. Składa się z dwóch części, lewa (`<div class=leftdiv>`) wyświetla formularz wprowadzania danych (przekazany jako zmienna `load_forms`) oraz zawiera przycisk, który powoduje wysłanie formularza i uruchomienie javascriptu, odpowiednia instrukcja jest w nagłówku formularza: `onsubmit="wait_load('outputDiv_calculate'`

`'input'); return(false)`. Po prawej stronie wyświetlają się wyniki (jeśli zmienna `is_ready.result` ma wartość) lub napis "No calculations have been made yet"