FRACTAL
GENERATOR

Politechnika Wrocławska

Aplikacje Internetowe i Rozproszone

Generowanie fraktali z użyciem klastra obliczeniowego

Jakub Burzała
Krzysztof Cabała
Bartosz Cieśla
Adrian Frydmański
Dawid Gracek
Bartosz Kardas

prowadzący
dr hab. inż. Henryk Maciejewski

2 czerwca 2016

Spis treści

1 Cel projektu i wymagania

Celem projektu jest stworzenie aplikacji do generowania fraktali z użyciem technologii internetowych i zrównoleglonych obliczeń na klastrze MPI.

Użytkownik aplikacji powinien móc:

1. Wpisać dowolną funkcję zespoloną generującą fraktal
2. Zobaczyć wygenrowany fraktal
3. Przybliżyć obraz w pewnym punkcie płaszczyzny
   - klikając
   - suwakiem
   - zaznaczając pewien obszar
4. Określić:
   - czas trwania animacji
   - liczba klatek na sekundę
   - rozdzielczość
5. Zatrzymać, przewinąć, wznowić animację
6. Wybrać rodzaj animacji. Animowanie względem:
   - przybliżenia – wartość początkowa i końcowa
   - punktu centralnego (ścieżki) – wybranie dwóch punktów na płaszczyźnie (początek i koniec) oraz zdefiniowanie ścieżki pomiędzy nimi, która wskazuje w jaki sposób będzie poruszać się kamera
   - parametrów zespolonych – dowoloność we wprowadzaniu parametrów w równaniu zespolonym. Każdy z nich może być zmieniany z zadanym krokiem podczas animacji
   - kolorystyki – wybór kolorystyki fraktala, definiowanie własnych zasad kolorowania
   - kroków zbieżności – ilość maksymalnej ilość kroków zbieżności, potrzebnych do obliczenia koloru danego piksela
7. Łączyć powyższe animacje:
   - szeregowo – animacje odtwarzają się po sobie
   - równolegle – animacje wykonują się w tym samym momencie
8. Zobaczyć postęp prac przy generacji animacji w postaci paska postępu oraz przewidywany czas zakończenia
9. Zobaczyć skalę przybliżenia w postaci łatwej do wyobrażenia jednostki. Przykładowo punkt początkowy: 100 px odpowiada 1 km. W miarę prybliżania aktualizowanie jednostki do m, cm, mm itd.
10. Zapisać opis animiacji (funkcja, parametry itp) na swoje urządzenie
11. Odczytać wcześniej zapisany opis animacji
12. Zapisać animację na swoje urządzenie

13. Podzielić się wynikiem na Facebooku :)
14. Przygotowanie zadania, bez edytora tylko z pliku konfiguracyjnego
15. Logować się na serwer

Administrator powinien móc:

1. Płynnie przełączać się pomiędzy typem prostym double, a klasą mpf implementującą liczby zmiennoprzecinkowe dowolnej dokładności
2. Wybierać metodę zrównoleglania
    - piksele
    - linie
    - klatki
    - części animacji
3. Wybierać liczbę jednostek wykonawczych

Aplikacja powinna:

1. Zarządzać zadaniami, kolejkować je
2. Sekwencyjnie wykonywać zadania dla wielu użytkowników (z priorytetowaniem)

2 Prototyp

2.1 Opis ogólny

1. Przynajmniej 2 liczące slave'y
2. Może być jednoużytkownikowy system – zlecamy zadanie i trzymamy na serwerze, żeby można było wrócić do niego po jakimś czasie.
3. Wynikiem może być wykonanie fraktala (1 klatki) lub film – dowolność.
4. Zlecenie zadania z np. wczytywaną konfiguracją z pliku, przekazanie do serwera, zakończenie zadania i zwrócenie wyniku użytkownikowi.

2.2 Opis szczegółowy

1. Klaster złożony z co najmniej dwóch maszyn, na każdej wykorzystywane wszystkie wątki procesora
2. Interfejs webowy umożliwiający zapamiętanie kolejki zadań
3. Możliwość pobrania wygenerowanego filmu (ew. obrazka)
4. Możliwość wpisania dowolnej funkcji zespolonej
5. Określenie czasu trwania animacji
6. Generowanie animacji na podstawie klatek kluczowych
7. Każda klatka kluczowa opisana współrzędnymi dwóch rogów obrazka
8. Wybór rozdzielczości animacji
9. Wybór między odcieniami szarości a kolorowym obrazkiem
10. Opis sceny wczytany z pliku tekstowego przygotowanego w edytorze tekstu
11. Interfejs webowy bez podziału na użytkowników uruchomiony na komputerze z aplikacją master
12. Kolejka może być obsługiwana przez aplikację odpowiadającą za interfejs webowy

3 System w wersji finalnej

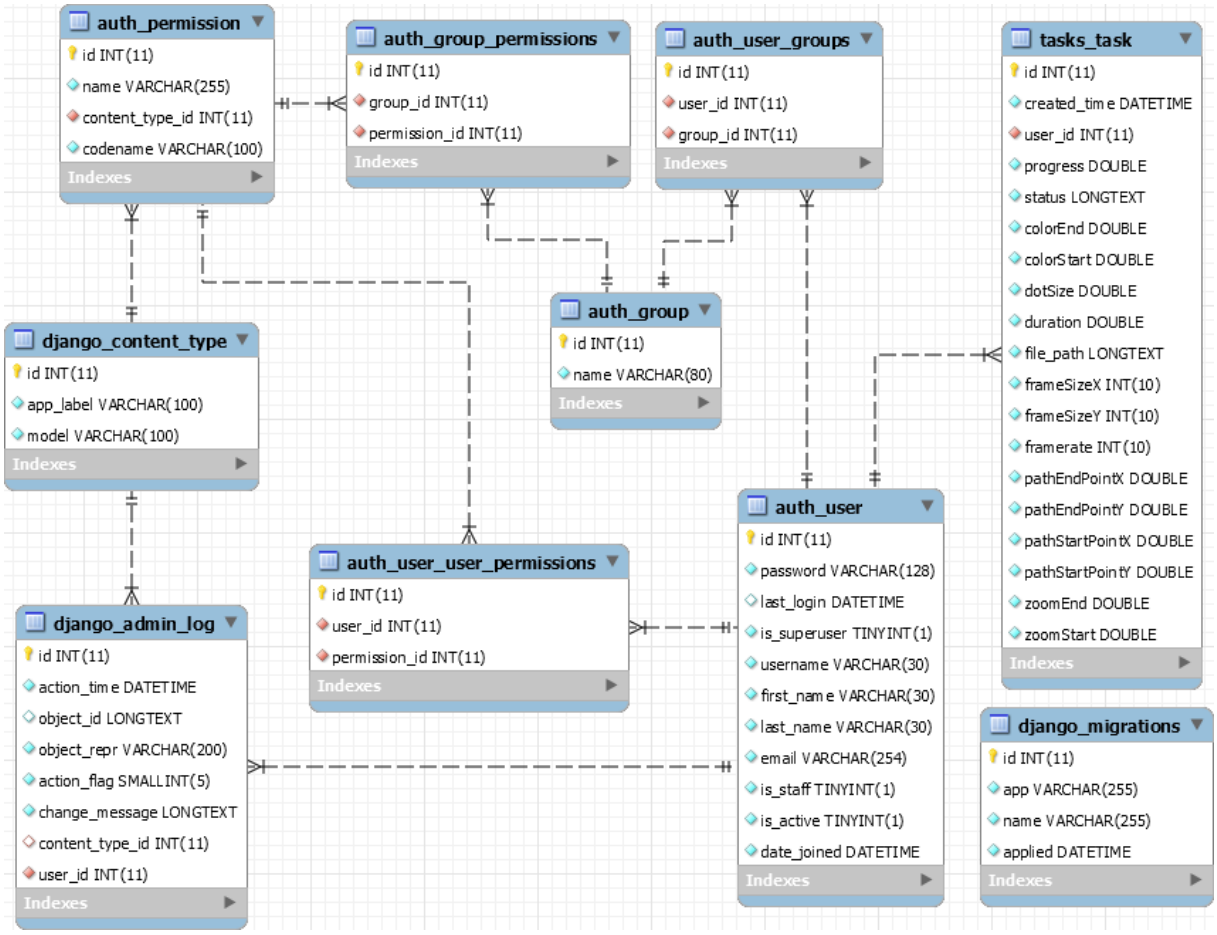
Aplikacja spełnia wymagania. Posiada czytelny interface użytkownika umożliwiający wprowadzanie nowych danych celem wykonania obliczeń. Wynikiem tychże jest animacja, której elementem jest fraktal.

Na działający serwis składa się kilka elementów. Są to:

- Serwer bazodanowy – MySQL Server
- Klaster MPI generujący animacje
- Serwer internetowy

3.1 Baza danych

Baza danych służy do przesyłania zadań pomiędzy aplikacją w *Django*, a aplikacją rozproszoną oraz do zarządzania użytkownikami systemu. Umieszczona jest na serwerze MySQL. Poniżej widoczny jest jej model fizyczny.



Rysunek 1: Model bazy danych

Kluczową dla zapewnienia funkcjonalności serwisu jest tabela tasks_task. Zawiera ona informację o zadaniach (zarówno parametry i status wykonania, jak i informację o dacie dodania i autorze).
Tabela auth_user zawiera użytkowników. Te i pozostałe tabele obsługuje serwer *Django*, między innymi na nich opierając swoje działanie.

3.2 MPI

Program co ustalony kwant czasu odpytuję bazę danych, sprawdzając czy są nowe zadania do wykonania. Jeśli znajdzie takowe, zaczyna je liczyć, zaprzęgając do pracy węzły MPI.

Główna funkcja programu jest dość prosta. W zależności od roli, jaką ma pełnić program, uruchamiany jest master lub slave.

3.2.1 Master

Master odpowiada za sprawdzanie bazy danych.

Jeśli znajdzie zadanie, pobiera jego parametry i rozsyła zadanie slave'om. Po odebraniu zadania aktualizowany jest jego status, dzięki czemu na stronie internetowej widać pasek postępu – o którym w dalszej części dokumentacji.

Slave'y pobierają zadania – pojedyncze klatki do wygenerowania. Wygenerowane obrazy odbierane są od slave'ów:

Kiedy wszystkie zadania sleve'ów zostaną wykonane, master łączy klatki w film za pomocą programu ffmpeg:
ffmpeg -framerate 30 -y -i images/%d.bmp -c:v libx264 -r 30 -pix_fmt yuv420p nazwa_pliku.mp4

3.2.2 Slave

Slave jest odpowiedzialny za generowanie fraktala. Wywołuje metodę liczącą fraktal – calcMandelbrotPart – z klasy FractalCalc. Po policzeniu odsyła wynik masterowi.

3.3 Django

Aplikacja internetowa napisana przy użyciu frameworku *Django* jest kolejnym ważnym elementem systemu. To ona odpowiada za komunikację *użytkownik – baza danych.*

Widoki dostępne w interfejsie użytkownika dzielą się na:
Ogólnodostępne:

- strona główna
- logowanie
- rejestracja

Wymagające zalogowania:

- lista zleconych zadań generacji
- dodawanie nowego zadania generacji
- przeglądanie wygenerowanych animacji

Dzięki systemowi szablonów, strony, których zawartość korzysta z powtarzającego się kodu, można sprowadzić do zapisu tylko różniących się fragmentów jako rozszerzenie głównego szablonu. Przykładem takiego zabiegu jest m.in. formularz rejestracji. Dodatkową funkcjonalnością jest autogeneracja kodu przez *Django*. W poniższym przykładzie w miejscu {{*form.as_p*}} zostanie wygenerowany kod formularza rejestracyjnego.

```
1 {% extends "tasks/tasks.html" %}
2 {% load i18n %}
3
4 {% block reg_log %}
```

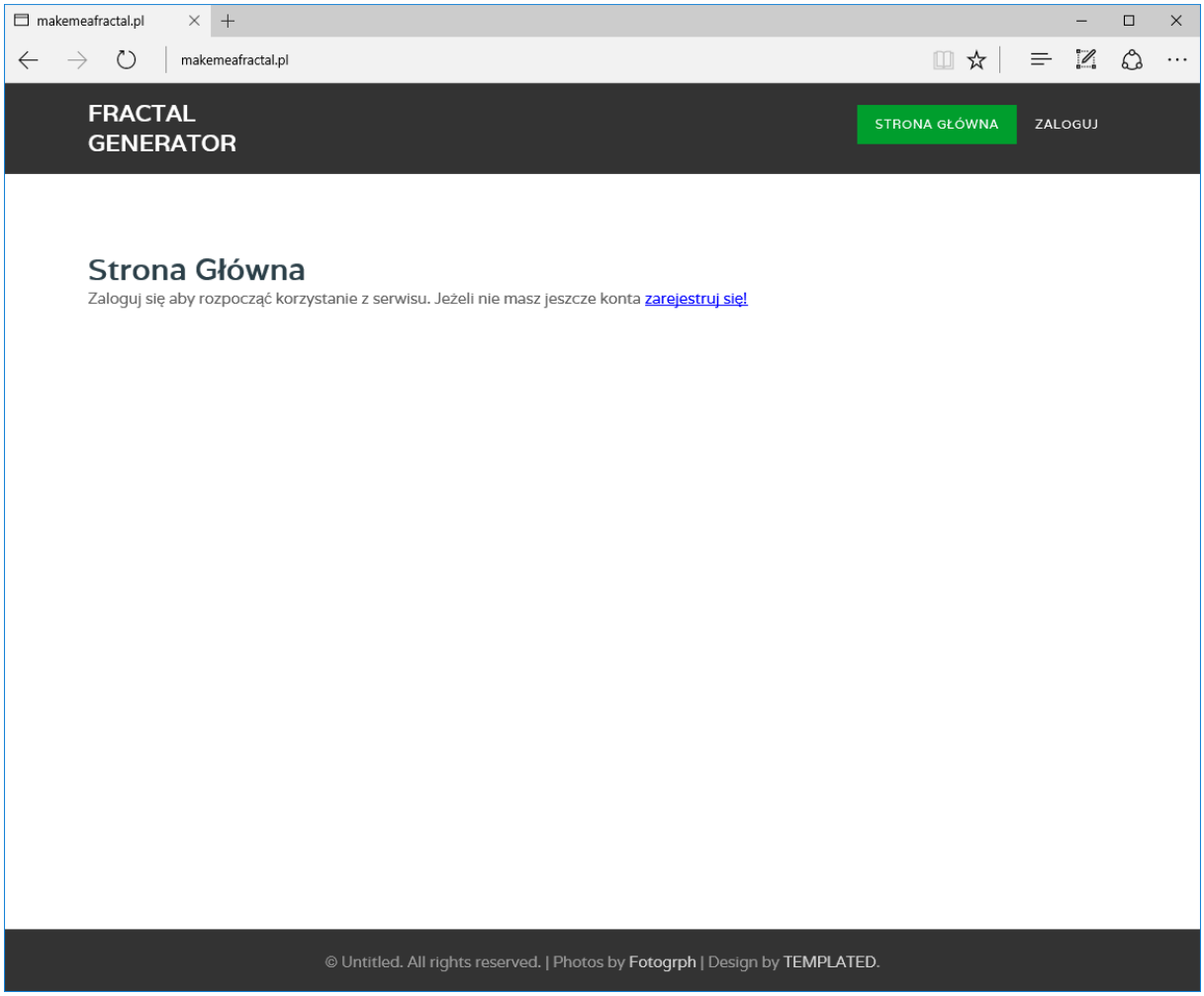
```
5           <form method="post" action=".">
6               {% csrf_token %}
7               {{ form.as_p }}
8               <label> </label><input type="submit" value="{% trans 'Submit' %}"/>
9           </form>
10  {% endblock %}
```

Za załadowanie odpowiedniego szablonu i umieszczenie w nim danych odpowiedzialny jest widok. Wszystkie widoki są zdefiniowane w pliku *views.py*.

Ustawienia całej aplikacji internetowej zawarte są w pliku *settings.py*.

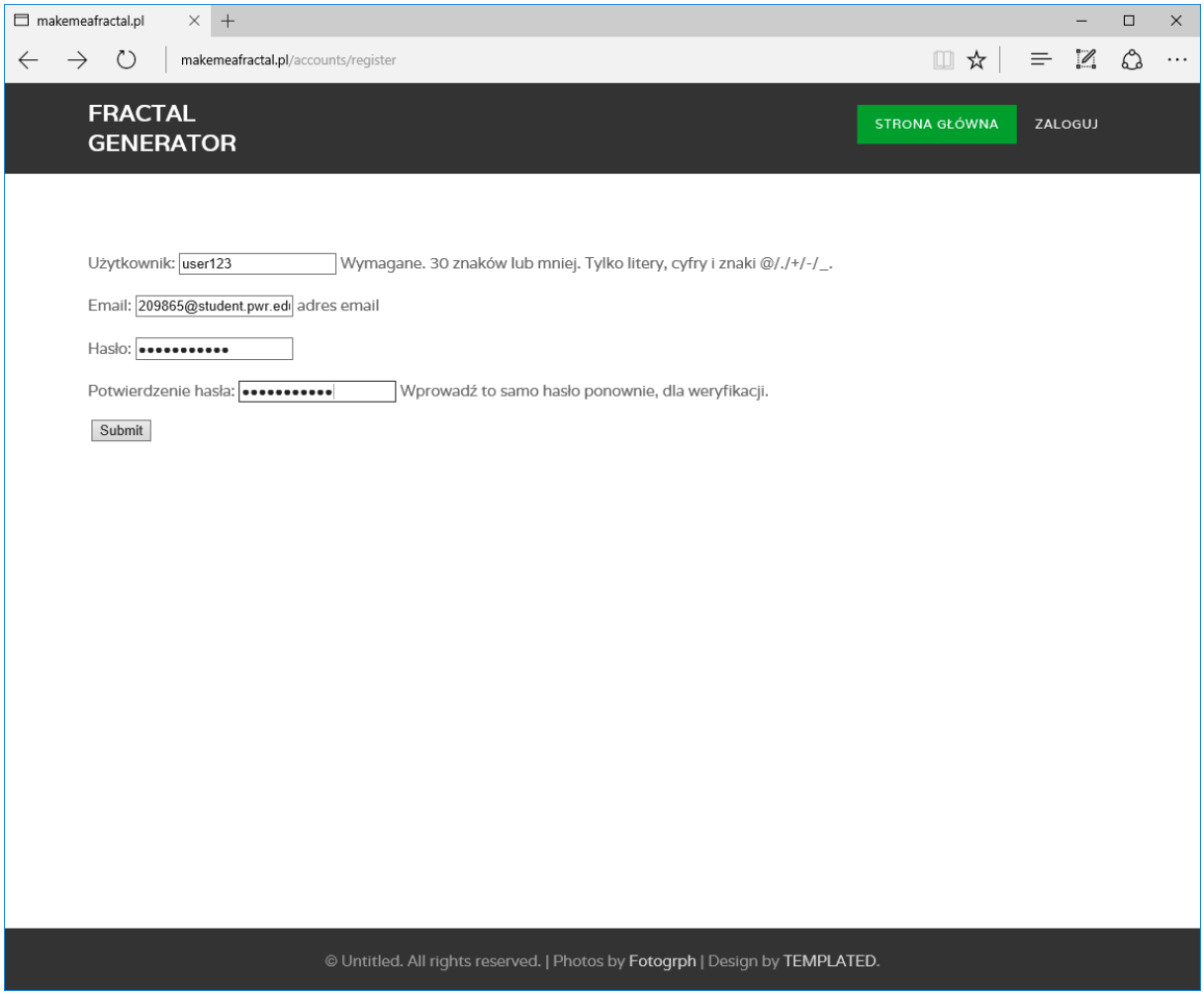
3.4 Strona internetowa – interfejs użytkownika

Poniższe zrzuty ekranu prezentują działanie serwisu:



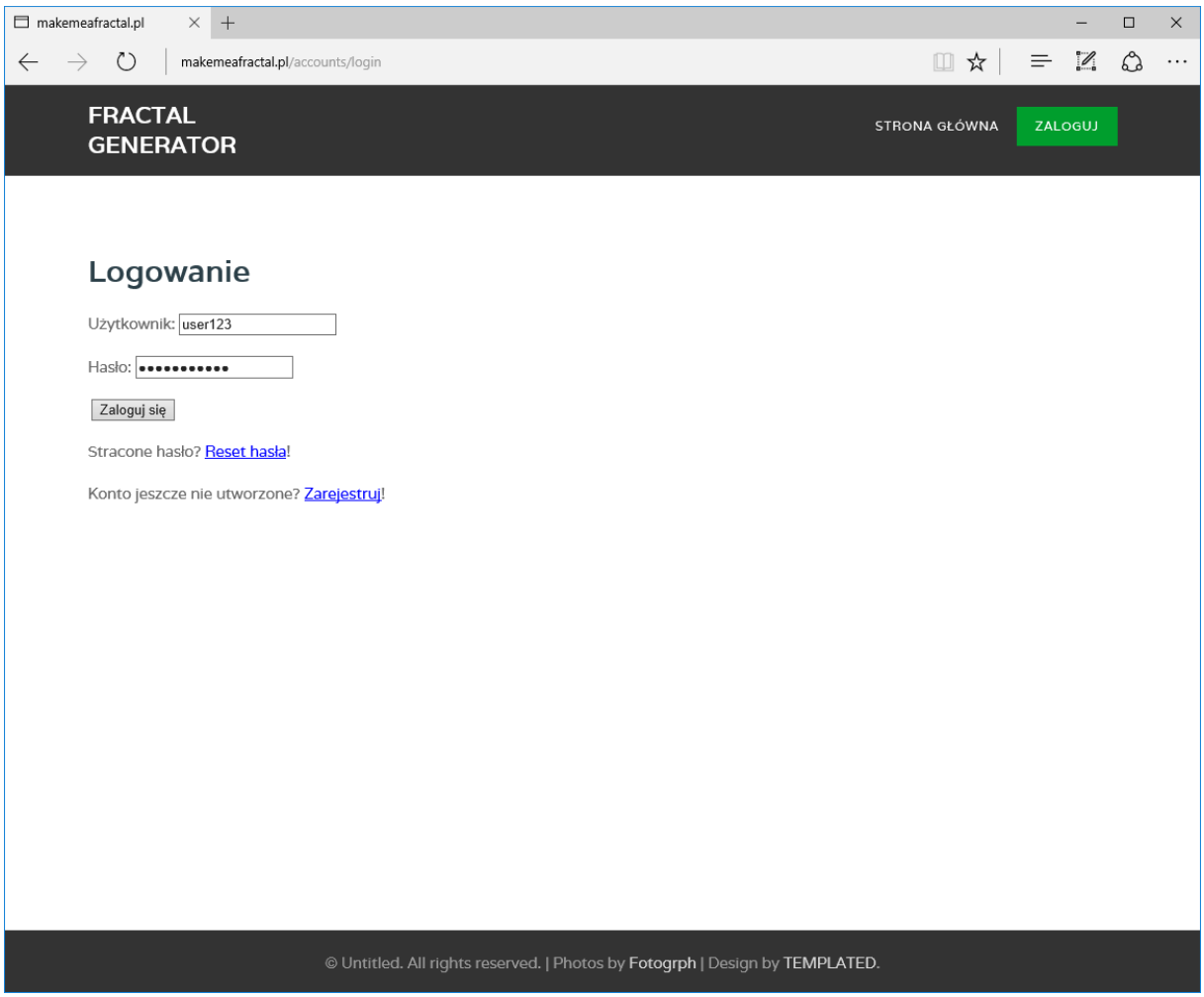
Rysunek 2: Strona główna

Na stronie głównej widoczna jest propozycja zalogowania się, lub rejestracji. Bez tego niemożliwe jest wysyłanie zadań do wykonania.

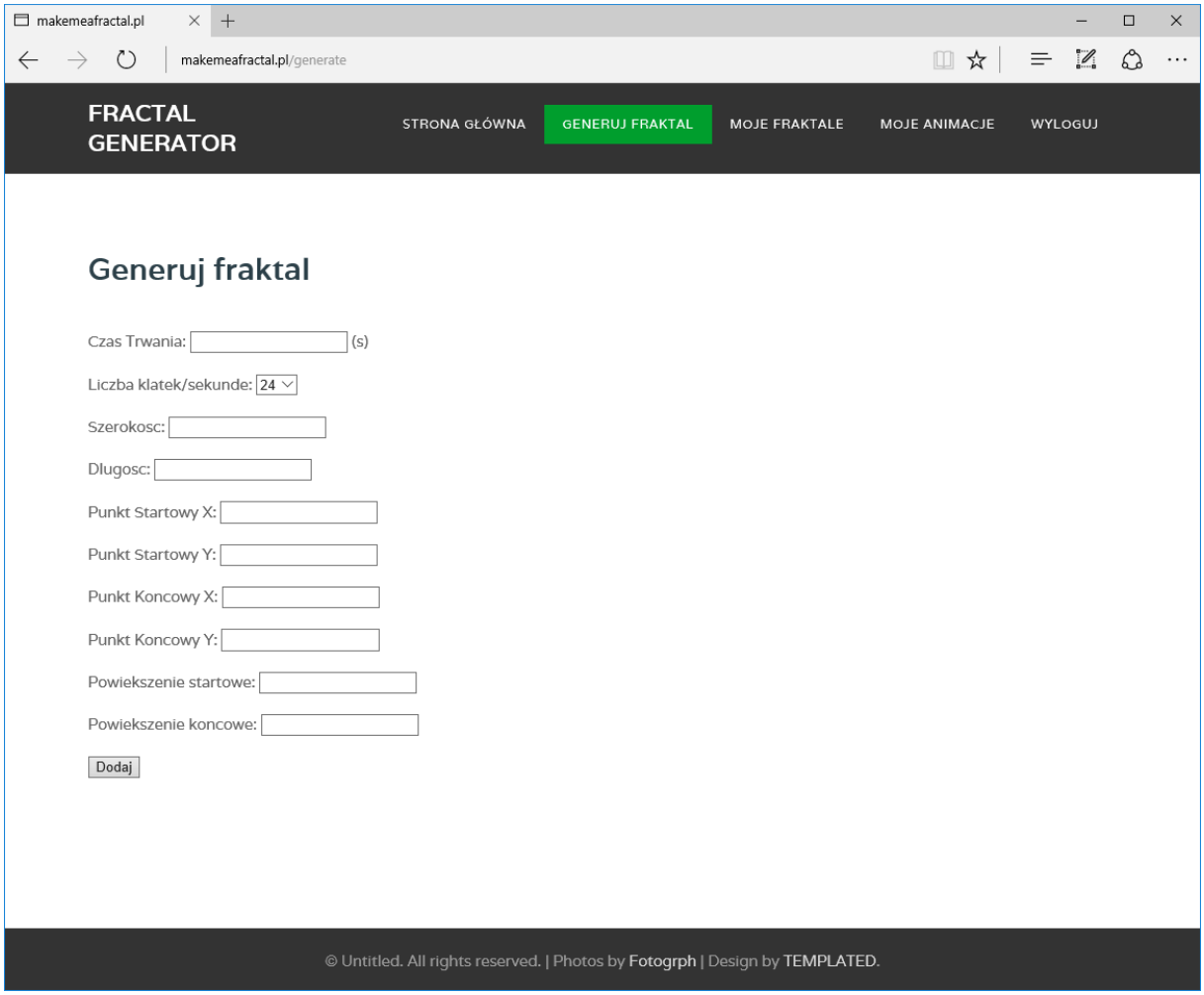


Rysunek 3: Rejestracja

Rejestracja wymaga podania nazwy użytkownika, adresu email i hasła. Na podany adres zostaje wysłany link aktywacyjny, po kliknięciu którego można korzystać z serwisu.

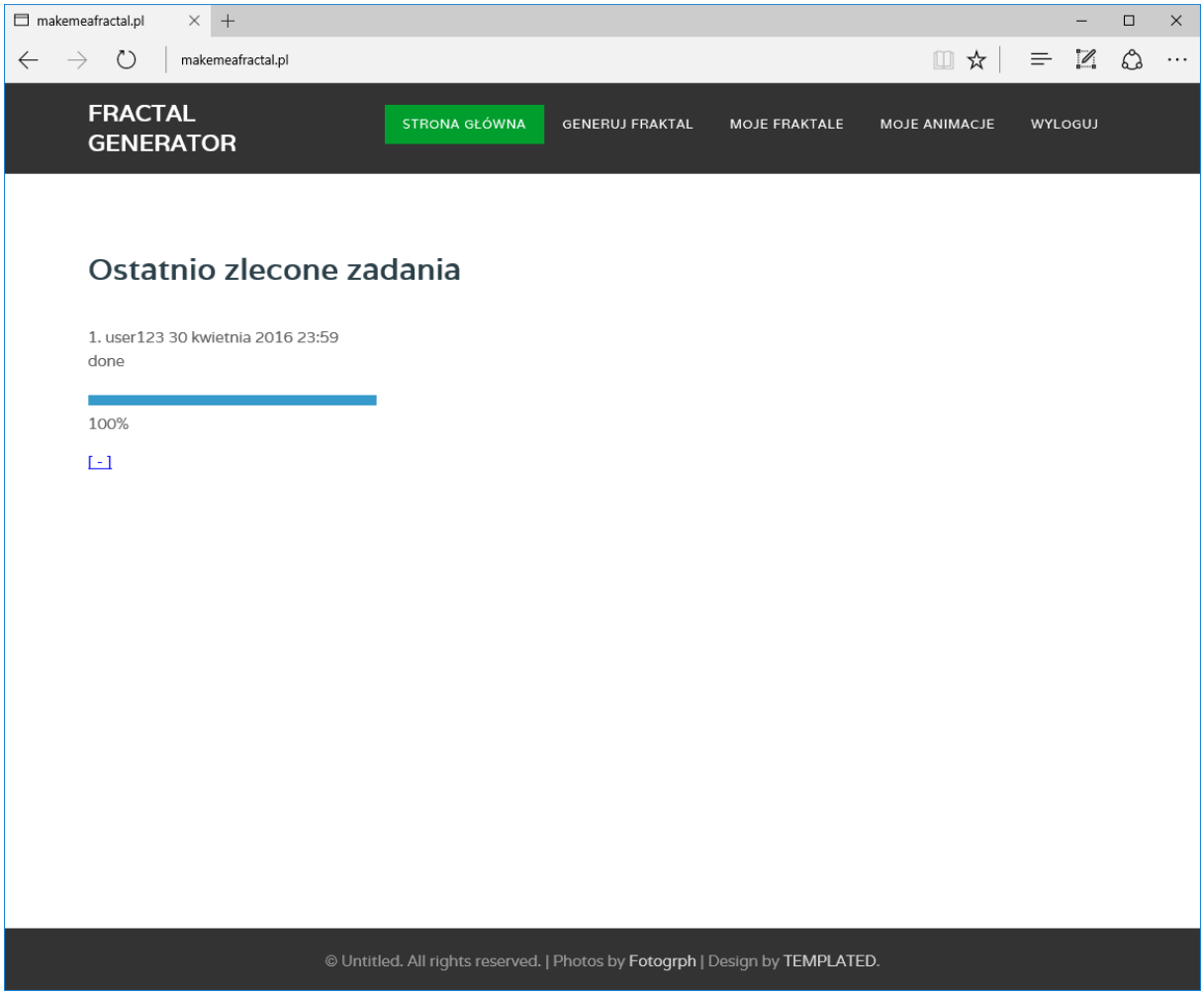


Rysunek 4: Logowanie



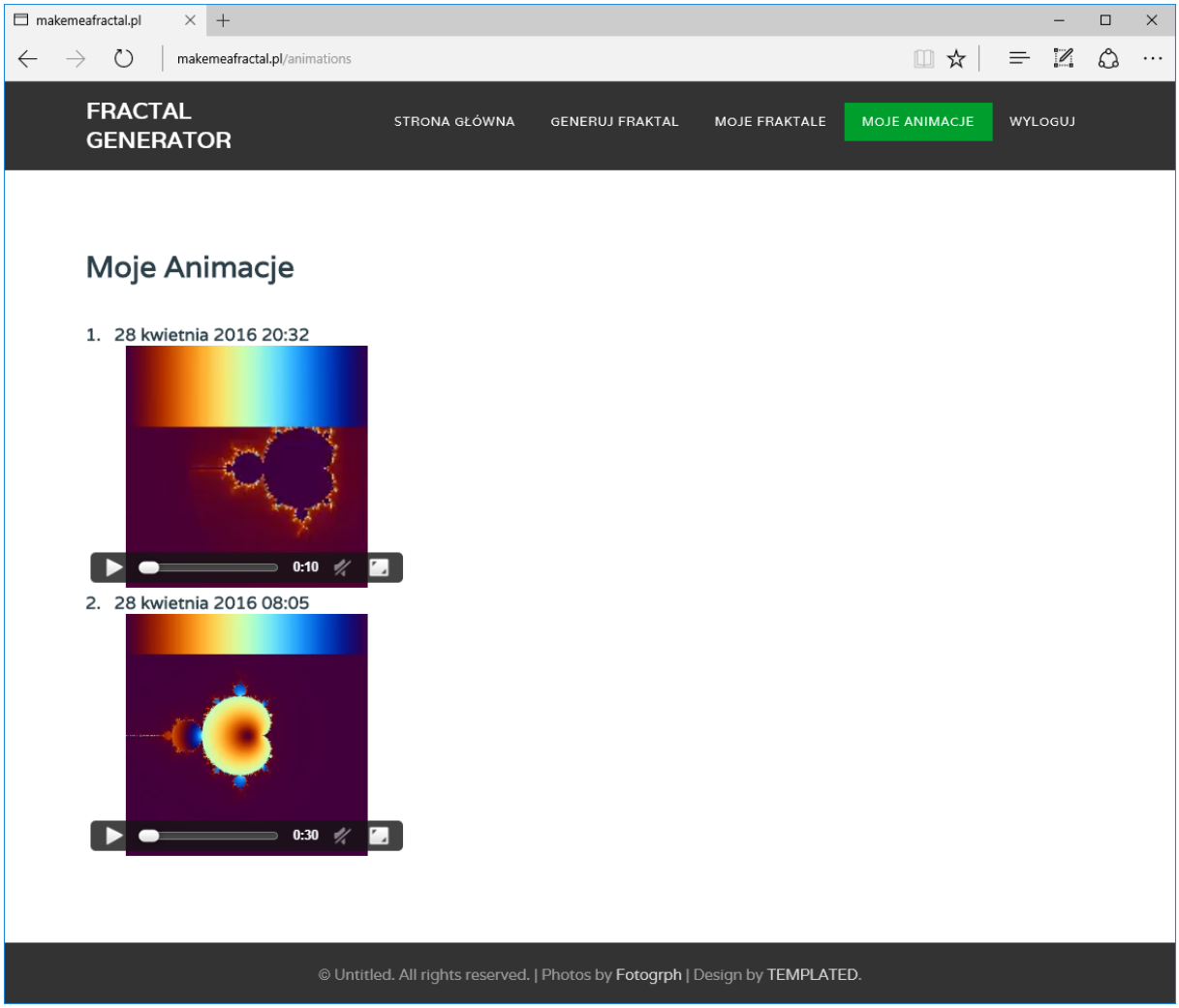
Rysunek 5: Generowanie fraktala

Na tej stronie istnieje możliwość dodania nowego zadania do wykonania. Podawane są parametry animacji – dotyczące zarówno pliku wideo, jak i samego fraktala i jego pozycji.



Rysunek 6: Kolejka zadań

Po zalogowaniu się na stronie głównej widoczne są zlecone zadania i ich status wykonania wraz z paskiem postępu.



Rysunek 7: Moje animacje

Na stronie Moje animacje widoczny jest podgląd wygenerowanych animacji.

4 Uruchamianie

4.1 MPI

W celu uruchomienia aplikacji MPI potrzebne są następujące składniki:

- System Linux (testowane na Debian/Mint)
- Serwer SSH na stacjach slave
- Logowanie do ssh slave przy użyciu klucza RSA (bez podawania hasła)
- Program rsync na stacji master
- Program ffmpeg
- Biblioteki:
  - libmysqlclient-dev
  - libstdc++6
  - openmpi-devel
  - gmp-devel
  - opengl
- Kompilator mpic++

Program obsługujący MPI znajduje się w katalogu Cluster. Uruchomienie klastra odbywa się za pomocą skryptu *Cluster/run.sh*. Przyjmuje on następujące parametry:

- liczbę slotów w klastrze (node'ów)
- wielkość ziarna jako część całej klatki np. 2 oznacza połowę klatki, 4 oznacza ćwierć, 10 oznacza jedną dziesiątą klatki itd.

Program obsługujący MPI znajduje się w katalogu Cluster. Uruchomienie klastra odbywa się za pomocą skryptu Cluster/run.sh. Przyjmuje on następujące parametry:

- liczbę slotów w klastrze (node'ów)
- wielkość ziarna jako część całej klatki np. 2 oznacza połowę klatki, 4 oznacza ćwierć, 10 oznacza jedną dziesiątą klatki itd.

W pierwszej kolejności wykonywana jest kompilacja projektu. W pliku źródłowym *Master.cpp* jest makrodefinicja *_LOCALHOST_ONLY_*, dzięki której można wyspecyfikować czy program ma łączyć się ze zdalną bazą danych, której adres zmodyfikować można w pliku *NetCpp/comm.h* czy opis sceny ma być pobierany z kodu źródłowego. Następnie pliki wykonywalne rozsyłane są za pośrednictwem programu rsync do poszczególnych node'ów. Polecenie to wygląda następująco:
rsync -av -e "ssh" ./main user@host: /main, gdzie
user - nazwa użytkownika
host - adres ip node'a
W ostanim etapie uruchamiany jest klaster za pośrednictwem polecenia mpirun.

4.2 Aplikacja internetowa (Django + baza danych)

Aplikacja internetowa w *Django* znajduje się w katalogu WebApp. Aby ją uruchomić wymagane są:

- Python w wersji 2.7
- Django w wersji min. 1.9.2
- MySQLdb
- django-registration

Przed włączeniem serwera należy utworzyć pustą bazę danych i wypełnić ją na podstawie modelu MySQL (Doc/mandel.mwb) lub poleceń w *Django*:
python manage.py makemigrations
python manage.py miagrate

5 Testy wydajności

Testy zostały wykonane przy użyciu trzech fizycznych maszyn. 2 z nich były wyposażone w 2 (Intel i5), a jedna 4 (Intel i7) rdzenie procesora. Zastosowana w nich technologia HyperTreading pozwala na jednoczesną pracę dwóch wątków na jednym rdzeniu, zatem można stwierdzić, że były dostępne odpowiednio 4 i 8 rdzeni Maszyna na której uruchamiany był master pracowała pod kontrolą systemu operacyjnego Linux Mint, natomiast na pozostałych były zainstalowane maszyny wirtualne z systemem Linux Debian. Komputery były spięte do jednego switcha z portami FastEthernet. Dostępny sprzęt pozwolił przetestować następujące konfiguracje:

1. 1 maszyna fizyczna, 4 rdzenie
2. 1 maszyna fizyczna, 8 rdzeni
3. 2 maszyny fizyczne, 8 rdzeni (4 + 4)
4. 2 maszyny fizyczne, 12 rdzeni (4 + 8)
5. 3 maszyny fizyczne, 12 rdzeni (4 + 4 + 4)
6. 3 maszyny fizyczne, 16 rdzeni (4 +4 +8)

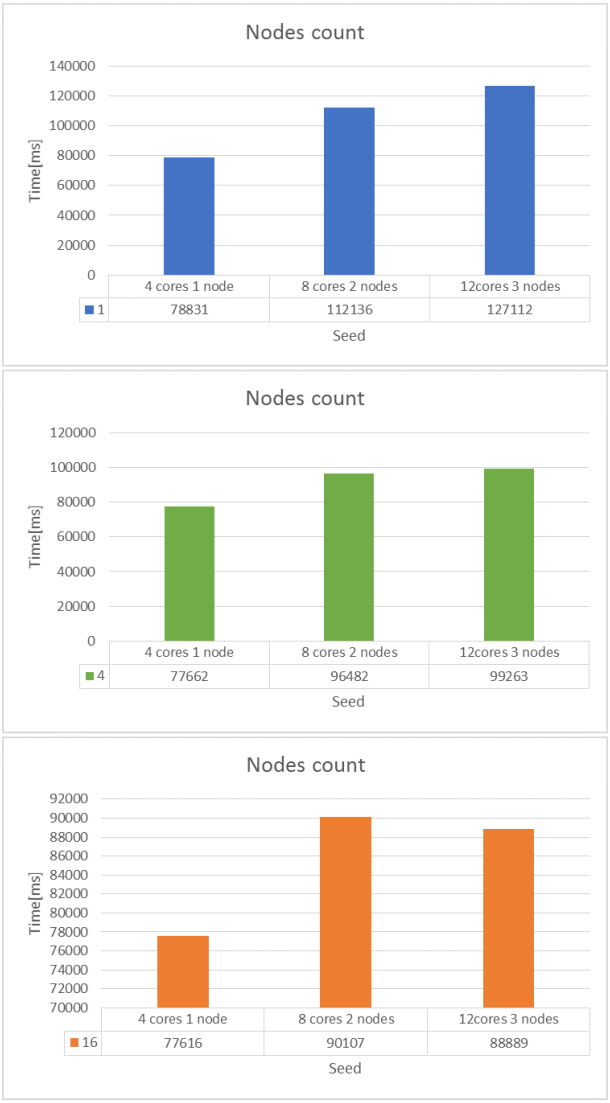
Na wszystkich maszynach zostało uruchomione zadanie wygenerowania 1-sekundowej animacji (30 klatek) w różnej rozdzielczości (400x400 – 2000x2000 px ze skokiem co 200 px). Drugi scenariusz testowy dotyczył podziału klatki na mniejsze podzadania. W tym przypadku rozdzielczość została ustalona na 2000 x 2000 px, natomiast klatka była dzielona na 1, 2, 4, 8, 16 podzadań (seed). Wszystkie pomiary zostały wykonane 10-krotnie, a następnie uśredniono wyniki. Widoczny na wykresach przypadek all dotyczył czasu wykonania całego zadania, wraz z generowaniem animacji z gotowych klatek, natomiast mpi dotyczy wyłącznie czasu obliczeń zrównoleglonych. Wykresy na których nie ma tego podziału pokazują czasy tylko obliczeń zrównoleglonych.

Rysunek 8: Porównanie czasu dla różnych konfiguracji

Z powyższych wykresów widać, że jeśli wszystkie rdzenie liczące znajdują się na jednej maszynie to podział zadania na różne fragmenty nie ma większego wpływu na czas obliczeń. Jeżeli jednak zwiększymy ilość maszyn liczących, to podział zadania na mniejsze pozwala zaobserwować spadek czasu obliczeń. Różnica ta zwiększa się wraz z dokładaniem do klastra kolejnych maszyn. Również liczba przydzielonych rdzeni procesora ma znaczący wpływ na szybkość omawianej zmiany.

Rysunek 9: Porównanie czasu dla różnych rozdzielczości

Porównanie czasu obliczeń w zależności od rozmiaru klatki obrazu ukazuje wyniki zgodne z oczekiwaniami. Ilość maszyn liczących i rdzeni procesora nią ma wpływu na charakter zmiany czasu obliczeń. We wszystkich przetestowanych przypadkach jest to zależność wielomianowa, zbliżona do funkcji kwadratowej. Ma to uzasadnienie w zmianie rozmiarze zadania – dwukrotne zwiększenie boku klatki powoduje czterokrotne zwiększenie rozmiaru zadania.



Rysunek 10: Porównanie czasu dla różnych liczb podzadań

Na wszystkich wykresach można zaobserwować wzrost czasu trwania wykonania zadania zależnie od ilości połączonych węzłów. Jest to spowodowane narzutem komunikacji pomiędzy węzłem master a wszystkimi węzłami slave. Patrząc jednak na wielkość ziarna zadania, czyli podziału jednej klatki na kolejno 1, 4 i 16 podzadań można zauważyć, że większe rozdrobnienie idzie w parze z przyspieszeniem czasu wykonania całości zadania, jest to lepiej widoczne im więcej węzłów połączymy ze sobą. Większa ziarnistość powoduje zmniejszenie zużycia łącza na przesłanie dużego fragmentu danych (przy ziarnie 1 przesyłana jest cała klatka, a przy 16 tylko 1/16). Przesyłanie trwa na tyle krótko żeby nie doszło do zbyt długiego blokowania węzła master podczas gdy inne węzły slave czekają na swoją kolej obsługi.

Rysunek 11: Porównanie czasu dla różnych rozmiarów klatek

Na wykresach czasu trwania wykonania zadania od wielkości klatki w pikselach możemy zauważyć dwie tendencje dla czasu wykonania. Pierwsza z nich to spadkowa, jeśli zwiększamy ilość węzłów i jest ona zauważalna dla rozmiarów klatek 400, 800, 1200. Druga z nich to tendencja wzrostowa jeśli zwiększamy ilość węzłów i widać ją dla rozmiaru klatek 1600 i 2000. Tendencja spadkowa przy małych klatkach jest spowodowana zwiększeniem mocy obliczeniowej klastra. Natomiast tendencja wzrostowa wynika ze zbyt długiego czasu przesyłania wyniku do węzła master. W tym czasie inne węzły mogą ukończyć zadanie i niestety muszą czekać dłużej na swoją kolej obsługi. Drugim, dość oczywistym wnioskiem, jest to że im większy rozmiar klatki tym więcej czasu potrzeba na wykonanie zadania.

6 Podsumowanie

Otrzymane wyniki są dość niespodziewane. Wydawać by się mogło, że dodatkowe węzły liczące powinny powodować redukcję czasu potrzebnego na obliczenia. Tymczasem, nasze pomiary nie wykazują tej własności. Najprawdopodobniej użyty do łączenia maszyn interfejs FastEthernet ma zbyt małą przepustowość przez co procesory nie są maksymalnie wykorzystane i występuje długi czas oczekiwania na komunikację pomiędzy masterem i slavem.

Cały projekt pokazał, że nie zawsze rozproszenie obliczeń skutkuje zwiększeniem wydajności. W tym celu należy odpowiednio dobierać parametry – do uzyskania optimum. Podczas pisania napotkano na różne trudności związane zarówno z tworzeniem oprogramowania, jak i z konfiguracją maszyn do współdziałania w ramach klastra. Utrwalone zostały umiejętności związane z technologiami webowymi (framework *Django*, język *Python*) i pokazane te związane z obliczeniami rozproszonymi (MPI).

Spis rysunków