**7. Aplikacja internetowa**

**Tworzenie nowego projektu**

W linii poleceń należy przejść do katalogu, w którym będzie znajdował się kod, następnie wpisując:

*django-admin.py startproject PI*

Utworzony zostanie katalog z projektem:

PI/

\_\_init\_\_.py

manage.py

settings.py

urls.py,

* \_\_init\_\_.py: Pusty plik informujący Pythona o tym, że katalog nadrzędny powinien być traktowany jako pakiet Pythona
* manage.py: Działające z linii poleceń narzędzie, które pozwala na interakcję z projektem Django na różne sposoby.
* settings.py: Ustawienia/konfiguracja dla tego projektu Django.
* urls.py: Deklaracja adresów URL dla tego projektu Django; “mapa serwisu” Twojej strony zbudowanej w oparciu o Django.

Przechodząc do katalogu PI, można sprawdzić czy server deweloperski działa, w tym calu należy wpisać:

*python manage.py runserver*

**Ustawienia bazy danych**

Django można używać bez bazy danych, jednak dostarczony jest wraz z systemem ORM (odwzorowanie relacyjno-obiektowe), za pomocą, którego można definiować tabele w bazie danych oraz pracować na nich pisząc i operując na klasach w Pythonie. Django współpracuje z następującymi serwerami bazodanowymi: PostgreSQL, MySQL, Oracle i SQLite. Ze względu na łatwość obsługi wybrany został MySQL, przez co należało doinstalować MySQLdb

***sudo apt-get install python2.7-mysqldb***

Co wymagało również zmiany silnika bazy danych w ustawieniach aplikacji na „django.db.backends.mysql” oraz z wnętrza powłoki bazy danych utworzenia potrzebnej bazy danych. Ze względu, że w pliku settings.py INSTALLED\_APPS zawarte są domyślne aplikacje, należy użyć komendy:

*python manage.py syncdb,* która dla w/w aplikacji tworzy tabele w bazie.

**Tworzenie modeli**

W katalogu PI, aby zacząć pracę nad projektem należy wprowadzić następujące polecenie:

*python manage.py startapp PI2*

Co utworzy podkatalog PI2, w którym będzie się znajdować struktura docelowej aplikacji.

W pliku models.py zostały stworzone następujące klasy:

class Task(models.Model):

userID = models.ForeignKey('auth.User')

numberOfPoints = models.BigIntegerField(default=0)

numberOfThreads = models.IntegerField(default=1)

add\_date = models.DateTimeField(default=timezone.now)

class DoneTask(models.Model):

userID = models.ForeignKey('auth.User')

numberOfPoints = models.BigIntegerField(default=0)

add\_date = models.DateTimeField(default=timezone.now)

value = models.TextField();

class UserProfile(models.Model):

user = models.OneToOneField(User)

def \_\_unicode\_\_(self):

return self.user.username

W tabeli Task znajdują się zadania, zlecone przez użytkowników, które oczekują w kolejce do pobrania i obliczenia przez silnik.   
*userID* – ID usera zgłaszającego zadanie

*numberOfPoints* – liczba punktów ??

*numberOfThreads* – liczba wątków, na których zostaną wykonane obliczenia

W pliku forms.py zostały dodane klasy obsługi rejestracji oraz zlecenia zadania:

*Litning. Klasa obsługi zdarzenia*

class CommisionTask(forms.ModelForm):

numberOfPoints = forms.IntegerField(label ="", min\_value=1)

numberOfThreads = forms.IntegerField(label ="", max\_value=12, min\_value=1)

class Meta:

model = Task

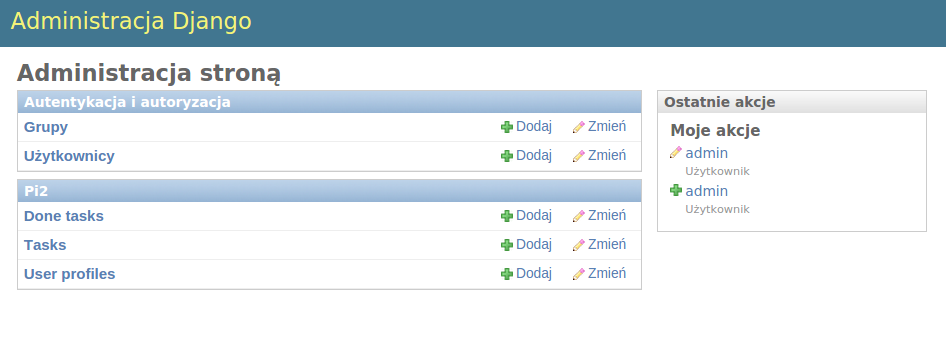
fields = ('numberOfPoints', 'numberOfThreads' )

Formy pobierające dane zawierają następujące ograniczenia:

- Liczba punktów nie może być mniejsza od 1

- Liczba wątków musi się znajdować w przedziale <1-12>

W pliku admin.py została dodana możliwość przeglądania przez administratora profilów użytkowników, zleconych zadań oraz skończonych zadań.



**Tworzenie widoków wraz z obsługą żądań**

W django do generowania widoków wykorzystywane są pliki z rozszerzeniem ".html". W aplikacji wykorzystano strony napisane w jezyku html w wersji 5 oparte o technologie twitter bootstrap.

Aby w django utworzyć strone internetową z utworzonymi plikami "html", należy utworzyć folder w katalogu projektu o nazwie "templetes". W folderze tym umieszcza się wszystkie pliki statyczne tzn. dodatkowe biblioteki javascript (z rozszerzeniem: jar, js) pliki styli "css" oraz wszystkie pliki "html" tworzonego portalu.

Nastepnie tworzy się plik urls.py w którym umieszcza się scieżki do plików html oraz podaje obsługujące je metody jezyka python zdefiniowane w pliku views.py:

*Listening pliku urls.py który wykorzystano w projekcie:*

*from django.conf.urls import patterns, include, url*

*from django.contrib import admin*

*urlpatterns = patterns('',*

*# Examples:*

*# url(r'^$', 'PI.views.home', name='home'),*

*# url(r'^blog/', include('blog.urls')),*

*url(r'^$', 'PI2.views.index'),*

*url(r'^odswiez/$', 'PI2.views.odswiez'),*

*url(r'^odswiez2/$', 'PI2.views.odswiez2'),*

*url(r'^admin/', include(admin.site.urls)),*

*url(r'^register/$', 'PI2.views.register', name='register'),*

*url(r'^login/$', 'PI2.views.user\_login', name='login'),*

*url(r'^logout/$', 'PI2.views.user\_logout', name='logout'),*

*url(r'^wyczysc/$', 'PI2.views.wyczysc', name='wyczysc'),*

*)*

W pliku views.py zdefiniowane są wszystkie kontrolery obsługujące scieżki url oraz request'y, gdyż korzystając z technologi ajax można wywołać dany kontroler requestem POST lub GET.

*Listening prezentujący fragment pliku views.py który wykorzystano w projekcie:*

*from PI2.models import Task, DoneTask*

*import PI2*

*from django.template import Context, loader*

*from django.shortcuts import render*

*from django.shortcuts import render\_to\_response*

*from django.utils import timezone*

*from django.template import RequestContext*

*from PI2.forms import UserForm, CommisionTask*

*from django.contrib.auth import authenticate, login*

*from django.http import HttpResponseRedirect, HttpResponse*

*from django.contrib.auth.decorators import login\_required*

*from django.contrib.auth import logout*

*from .forms import CommisionTask*

*from django.http import HttpResponse*

*from datetime import date*

*import parser*

*from django.forms.fields import DateTimeField*

*def index(request):*

*form = CommisionTask(request.POST)*

*if form.is\_valid():*

*Task = form.save(commit=False)*

*Task.userID = request.user*

*Task.save()*

*#Resetuje formularz po pobraniu poprawnym*

*form =CommisionTask(0);*

*return render\_to\_response('index.html', {'form': form }, context\_instance=RequestContext(request))*

*def wyczysc(request):*

*Task.objects.all().delete()*

*DoneTask.objects.all().delete()*

*return HttpResponse('Czyszczenie bazy wykonane!')*

*def odswiez(request):*

*"""Tutaj mamy pobieranie danych z tabeli i przekazywanie do strony. Pobieramy wszystkie dane z tabeli task czyli .all()*

*filtrujemy po userID = ID oraz sortujemy po dacie dodania malejaco od najmlodszego czyli descending a odpowiada za to:*

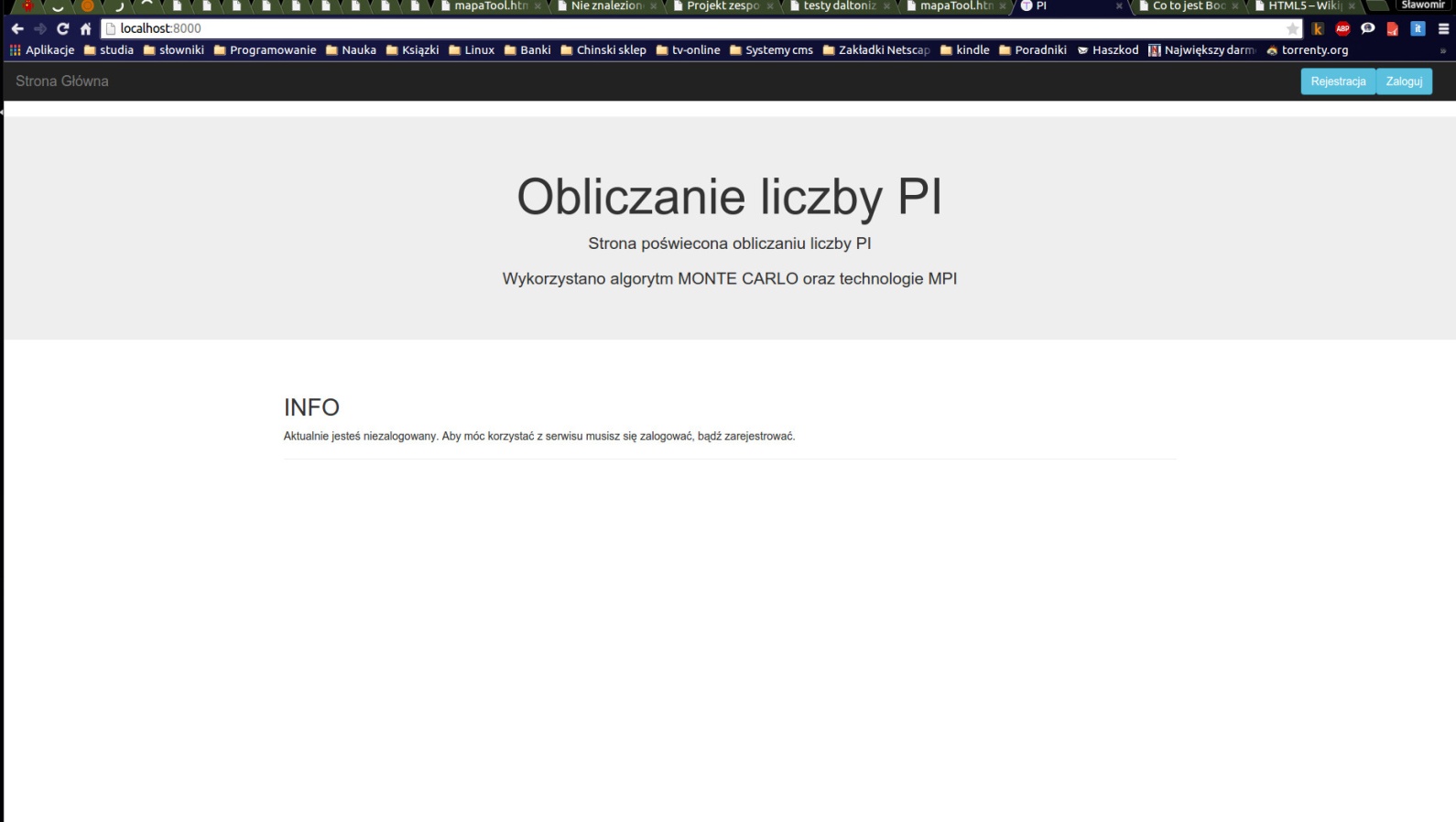
*minus myslnik - przed fieldem czyli rekordem add\_date"""*

*form2 = PI2.models.DoneTask.objects.all().filter(userID= request.user.id).order\_by('-add\_date')*

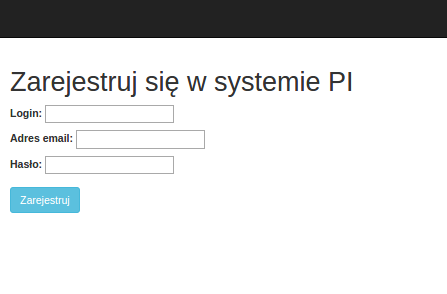
*return render\_to\_response('odswiez.html', {'form2': form2}, context\_instance=RequestContext(request))*

*...*

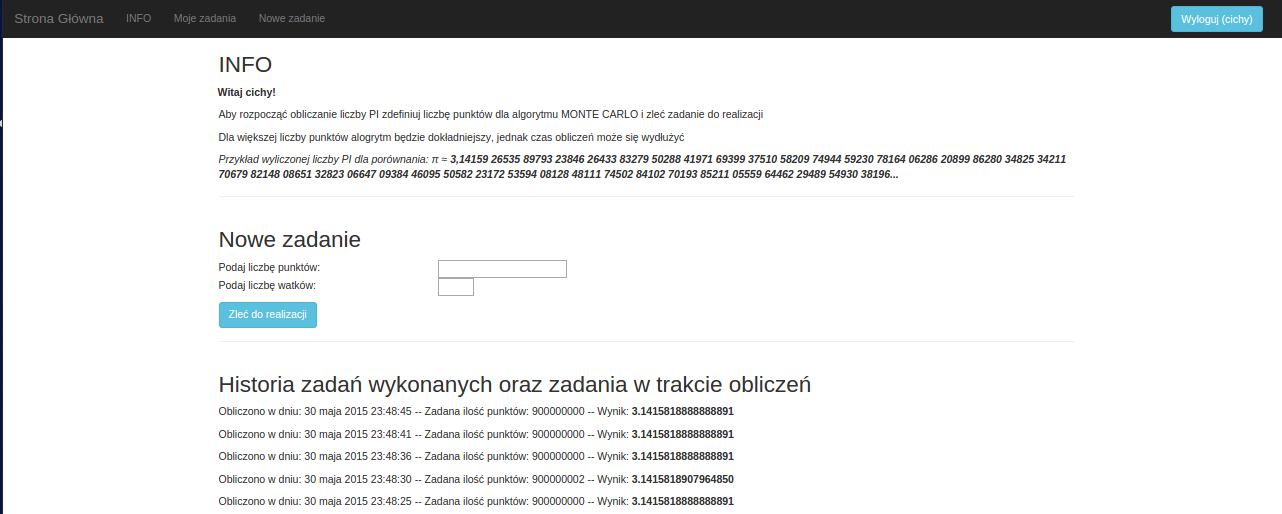
Po zdefiniowaniu bierzących plików oraz uruchomieniu serwera domyślnie startowa strona wyświetli dokument index.html. W przypadku naszego projektu zostanie wyświetlona strona www zaprezentowana na rysunku nr 1 .



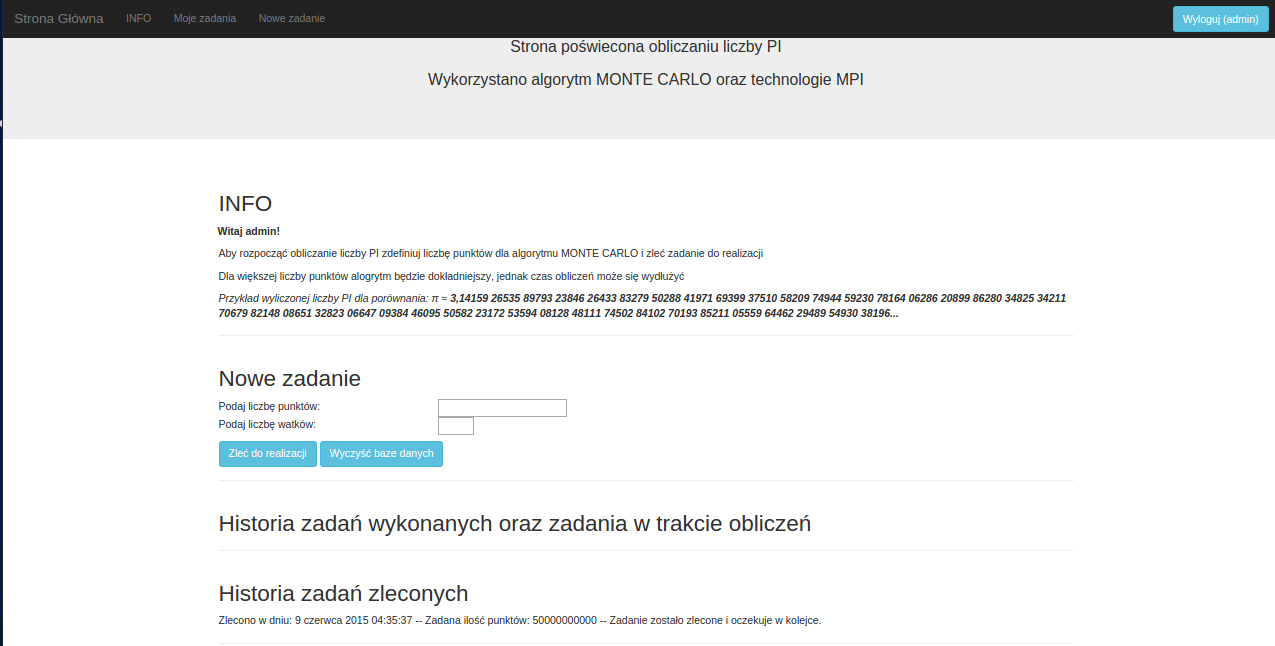
Rysunek

**Rejestracja**

**Logowanie**

****

**Administrator**

****