Drigan Documentation

Release dev

Drigan Team

February 14, 2015

Contents

Contents:

CHAPTER 1

Quick start

1.1 Installation

This document describes the process of setting up the environment. At the end of it there is an attachement with a quick list of commands that you need to run on some systems to get started even faster.

1.1.1 Pre-requirements

Altough most requirements will be installed for you automatically using PIP, there are some pre-requirements:

- Python >= 3.4
- PostgreSQL >= 9.3.4 with HStore enabled
- git
- Mercurial

Enabling HStore

To enable HStore in Postgresql type:

```
$ psql -d template1 -c 'create extension hstore;'
```

From now on all created databases would have HStore installed. You can also run this command only for one database, after creating it.

1.1.2 Setting up virtual environment

It is recommended to use virtual environments to decouple Python packages. For example, using venv (included in Python >= 3.3):

```
$ pyvenv /path/to/environment        #  create virtual environment
$ cd /path/to/environment            #  cd to its directory
$ source bin/activate                #  activate virtual environment
```

From now you can use only packages installed in this virtual environment. additionally, copies of *python* and *pip* binaries were created.

Take a look at virtualenvwrapper to make those commands even simpler.

1.1.3 Downloading project

Clone it using git:

```
$ git clone https://bitbucket.org/zeroos/drigan.git
```

1.1.4 Installing requirements

All requirements are in `requirements.txt`. You can install it with (don't forget about activating virtual environment!):

```
$ pip install -r requirements.txt
```

Hooray! Everything is installed, time to configure it and run.

1.2 Setup

Like in every Django application, you have to provide a `settings.py` file. There is a template for it in `drigan/settings_example.py`, just copy it and edit with your favourite editor:

```
$ cp drigan/settings_example.py drigan/settings.py
$ vim drigan/settings.py
```

1.2.1 Creating database

Now it's time to create a database. If you are doing it on your own just for development purposes you can for example use the following command. You have to issue it as a user with permissions to create PostgreSQL databases on your system (usually `postgres`).

```
postgres $ createdb drigan      # or other database name
```

If you ever need to reset your database to initial state you can ofcourse use Django management command (`reset`) or just recreate the database (probably more reliable):

```
postgres $ dropdb drigan      # drop the database
postgres $ createdb drigan      # and create it again
```

Before first stable version is released we are not going to use migrations, so you will have to reset the database after each model change.

1.3 Postgresql priveleges configuration

Since postgresql privelege configuration can be akward here is a quide.

Postgresql security config is in `pg_hba.conf` file, and by default (on ubuntu and debian) local socket connections use `peer` authentication so current sysytem user is named `foo` when connecting postgres will automatically authenticate `foo` database user. For this to work you'll need:

In `settings.py` set:

- `HOST` to empty string
- `DATABASE` to drigan

- USERNAME to foo (your system username)
- PASSWORD to empty string
- Add postgresql user that has the same username as you. In example we assume that your system username is foo.

  ```
  sudo createuser foo
  ```

  If you want to give your user superuser rights in postgres:

  ```
  sudo createuser -s foo
  ```

- Create database:

  ```
  sudo createdb -O foo drigan
  ```

1.3.1 settings.py

Every setting in the copied settings_example.py file is documented, so you can just go through them and adjust them.

If you are just trying to run it in developing mode, you don't have to change much – just adjust your database credentials if needed and everything should work.

However, if you'd like to set up a production environment, you should look over each setting. And don't forget to set DEBUG = False!

1.3.2 Database

```
$ python ./manage.py syncdb
```

1.3.3 Collecting static files

Note: You don't have to do it when DEBUG = False, i.e. in a development environmennt. In this case static files are served automatically by Django.

Before doing it make sure STATIC_ROOT is set correctly in settings.py.

```
$ python manage.py collectstatic
```

1.4 That's it!

And that's everything. If you're just running development instance you can run the server with

```
$ python manage.py runserver
```

and start coding!

If you are setting up a production environment you can use any technique that's used to deploy Django.

1.5 Testing

Django is recreating test database prior to each test run. This has unfortunate side-effect that `hstore` extension is missing. Until someone fixes this error you'll need to create `hstore` extension in `template1` database. If you do this **all future databases created in this system will contain this extension.**

```
psql template1 -c 'create extension hstore;'
```

CHAPTER 2

List of commands needed to configure environment on some systems

2.1 Debian/Ubuntu

1. sudo apt-get install python3.4-dev
2. sudo apt-get install python-pip
3. sudo pip install -U pip
4. sudo apt-get install postgresql-9.3
5. sudo apt-get install postgresql-contrib-9.3
6. sudo apt-get install postgresql-server-dev-9.3
7. sudo su - postgres

8. psql -d template1 -c 'create extension hstore;' <ctrl+D> to logout from postgres 9. sudo apt-get install mercurial 10. sudo apt-get install git 11. git config –global user.name "FIRST_NAME LAST_NAME" 12. git config –global user.email "MY_NAME@example.com" 13. sudo apt-get install python-virtualenv 14. mkdir ~/Drigan 15. cd ~/Drigan 16. virtualenv –python=/usr/bin/python3.4 environment 17. cd ~/Drigan/environment/ 18. source bin/activate

    You can skip 19 and 20 point and optionally clone repo over HTTPS: git clone https://zeroos@bitbucket.org/zeroos/drigan.git

19. Set up SSH Key. Tutorial here: https://confluence.atlassian.com/pages/viewpage.action?pageId=270827678
20. git clone git@bitbucket.org:zeroos/drigan.git
21. cd drigan
22. cp drigan/settings_example.py drigan/settings.py
23. vim drigan/settings.py
24. pip install -r requirements.txt
25. sudo su - postgres
26. createdb drigan
27. See: Postgresql priveleges configuration
28. sudo service postgresql restart
29. python ./manage.py syncdb
30. python manage.py collectstatic

31. python manage.py runserver

CHAPTER 3

Coding standards

3.1 Basic stuff

3.1.1 First solve bugs then add features

Bug fixes are more important then new features.

3.1.2 Code Reviews

Every story is created in it's own branch. Person developing code then adds pull request. This pull request must be accepted by someone else.

When asigning someone with a code-review, please mark your story as `In Review` and assign it to this person in JIRA.

3.1.3 Meaningfull commit messages

Use meaningfull commit messages. Please reference JIRA issue if appropriate.

3.1.4 Documentation

For now code is fairly undocumented, which is bad.

Every class, and public function needs a proper docstring. These docstrings are in sphinx format. We'll use autodoc to extract documentation from docstrings.

3.1.5 Unittests

Aim for 100% unittest coverage (for changed code).

3.1.6 Time logging

Please log time spent on developing this application. I need this information for political reasons (people at HQ don't have idea how much value we bring in terms of free (as in beer) software).

3.2 Django stuff

- Prefer class based views to function ones.
- Even better: use class based generic views.
- Try to keep inter application dependencies to minimum
- Separate scout code with generic code

3.3 Named urls

All urls should be named and referenced only by its name and application namespace.

There is no clearly defined convention for naming urls in Django. Even across documentation there are different standards. In Drigan, we have determined some simple url-naming rules:

- Try to keep the url name short and simple
- Always use hyphen (-) to separate words
- Use this set of words in a correct place to describe actions: * sufixes:
  - list (*not* index)
  - detail
  - prefixes: * create * update * delete

Furthermore, *always* specify application namespace in urls configuration and *always* use this namespace to reverse urls, eg.:

```
url(r"^pattern/", include('my_super_app.urls', app_name='my_super_app'))
```

And then when you reverse urls from this app:

```
reverse("my_supper_app:view-name")
```

CHAPTER 4

Specyfikacja techniczna

4.1 Pojęcia w systemie

4.1.1 Role w systemie

Użytkownik

Osoba która założyła sobie konto w aplikacji. Może ona rejestrować się na wydarzenia oraz tworzyć nowe wydarzenia (które podlegają akceptacji administratora)

Organizator

Oosba która tworzy dane wydarzenie. Jest ona zwykłym **użytkownikiem**, ale pojawia się ona dostateczne często by dostać własny rzeczownik ;)

Administrator

Osoba administrująca aplikacją. Może akceptować wydarzenia.

Osoba wdrażająca

Osoba która wdraża daną aplikację. Może ona robić proste zmiany i zna Pythona.

4.1.2 Organizator (Organizer)

By stworzyć jakieś wydarzenie należy podać jego organizatora, przy czym: organizator nijak nie ma się do osób kont użytkowników.

Dane jakie zbieramy o organizatorze są modyfikowalne przez osobę wdrażającą oprogramowanie. Zawsze organizator będzie zawierał takie dane:

- Nazwa instytucji
- Adres instytucji * Rozbite na atomowe pola :)
- Numer telefonu

Note: Może warto byłoby zrobić jakieś powiązanie organizaora i konta użytkownika, ale na razie tego nie robimy.

4.1.3 Wydarzenie (Event)

Wydarzenie jest zbiorem podwydarzeń, rejestrujemy się na podwydarzenia.

Note: W języku Harcerskim wydarzeniem będzie na przykład: Zlot Kadry 2012, mial on podwydarzenia: LAS, Zlot harcmistrzów, Zlot Akademików itp. Ogólnie może to na przykład być festiwal filmów, muzyki, biegów, lub nawet po prostu pojedynczy bieg, w ramach którego główną atrakcją też będzie ten bieg.

Podstawowe dane wydarzenia:

- Nazwę wydarzenia
- Opis Wydarzenia
- Numer edycji wydarzenia (na przykład nazwą będzie Zlot Kadry a edycją 2016). Pole edycja jest nieobowiązkowe.
- Dane organizatora.
- Datę rozpoczęcia i zakończenia wydarzenia
  - Zawsze podajemy datę rozpoczęcia
  - Data zakończenia jest opcjonalna, jeśli organizator zaznaczy że wydarzenie jest jednodniowe.

4.1.4 Podwydarzenie/atrakcja (Attraction)

Podwydarzenia są czymś na co uczestnicy się rejestrują. Podwydarzenie ma takie pola:

Podstawowe dane

- Nazwa podwydarzenia **obowiązkowe**
- Opis **obowiązkowe**
- Limit miejsc/brak limitu **obowiązkowe**

Dodatkowe dane

- Płatność/informacja o darmowości **obowiązkowe**, patrz: *Płantość*.
- Czas rezerwacji miejsca **obowiązkowe jeśli jest limit miejsc i płatność**.
  - Osoba która będzie się rejestrować na to podwydarzenie będzie miała tyle dni na dokonanie zapłaty. W przypadku gdy wpłata nie dotrze miejsce będzie zwalniane.
- Połączenie z rejestracją na wydarzenie
- Ograniczenia na uczestnika (walidacje), parz *Walidacje dostępu do wydarzenia*.

> **Warning:** Oprogramowanie tego może być skomplikowane w przypadku:
> - Manualnej rejestracji wpłat.
> - Automatycznego rejestrowania przekazów pocztowych

Podpinanie dynamicznych formularzy (dynamic forms)

Każde podwydarzenie ma możliwość podopięcia dynamicznego formularza zbierającego dodatkowe (wpisane przez Organizatora) dane o uczestnikach.

Informacje o tej funkcjonalności w *Dynamiczne dane do formularza rejestracji*.

Note: Limit miejsc prawdopowobnie wypadnie z pierwszej wersji aplikacji.

4.1.5 Dodatkowa konfiguracja wydarzenia

- Data rozpoczęcia rejestracji.
- Data zakończenia rejestracji.
  - Z opcjonalnym grace-period na wpłaty metodami nienatychmiastowymi.

Note: Opcjonalnie: czy nie rozważyć by te dane były określane per podwydarzenie.

4.1.6 Procesy powiązane z wydarzeniem

Tworzenie wydarzenia

tworzone

Kiedy wydarzenie jest **tworzone** nie wyświetla się na liście wydarzeń. Jest ono wtedy edytowalne dla osoby je tworzącej.

Do akceptacji

Kiedy osoba tworząca wydarzenie kliknęła odpowiedni guzik, wydarzenie uzyskuje status do akceptacji.

Wydarzenie przestaje być wtedy edytowalne (patrz: *Edytowalność wydarzenia a jego stan*).

W zależności od konfiguracji wydarzenie albo automatycznie przechodzi w status zaakceptowane, albo wymaga to kliknięcia przez administratora (patrz: *Weryfikacja wydarzena*).

Zaakceptowane

Wydarzenie nie jest edytowalne ale jest widoczne na liście wydarzeń.

Rejestracja otwarta/zamknięta

Rejestracja otwarta

W tym stanie możliwe jest rejestrowanie się.

Stan zmienia się na **rejestracja zamknięta**:

- w momencie w którym nadejścia zakończenia rejestracji (definiowane jako własność wydarzenia/atrakcji)
- w momencie przekroczenia limitu rejestracji
- pewnie inne.

Note: Możliwa jest również zamknięcie rejestracji pod wpływem odpowiednich validacji (przekroczenie limitu osób).

Rejestracja zamknięta

Nie ma możliwości rejestracji, stan przechodzi w **wydarzenie trwa/rejestracja zamknięta** w chwili rozpoczęcia wydarzenia.

Wydarzenie trwa

Note: To nie jest priorytet

Wydarzenie trwa

Stan ten ma dwa podstany:

- rejestracja trwa
- rejestracja zamknięta.

Wydarzenie zakończone

Stan po zakończeniu wydarzenia.

4.1.7 Edytowalność wydarzenia a jego stan

Na razie zamykamy wprowadzanie jakichkolwiek zmian do wydarzenia podczas jego trwania. Potem będzie trzeba włączyć częściową funkcjonalność zmiany wydarzenia.

Note: Decyzja po rozmowie z Michałem w REJCEN-29

4.1.8 Płantość

Note: Generalnie całkiem ważne może być wprowadzenie dynamicznej metody obliczania ceny. Tutaj nie mam pomysłu jak to uelastycznić w sposób sensowny.

Przykłady zastosowania:

- Rejestracja przed daną datą: mniejsza kwota
- Rejestracja dużej drużyny mniejsza kwota

Płatność zawiera dwie niezależne informacje:

- Kwotę opłaty.
- Metodę opłaty.
- Informacje powiązane z metodą opłaty.

Metoda opłaty (typ płatności)

Nie jest to element bazodanowy, a np. klasa instniejąca gdzieś w aplikajci, klasa ta odopwiada za obsługę danego rodzaju płatności.

Mamy takie metody opłaty:

Płatność darmowa

Specjalny rodzaj platności oznaczający coś bezpłatnego.

Rejestracja automatycznie przechodzi w stan: “Opłacone”

Płatność gotówką na miejscu

Z naszego punktu widzenia jest równoznaczna z płatnością darmową, ale wyświetlamy co innego uczestnikom.

Rejestracja automatycznie przechodzi w stan: “Nie wymagana opłata przez aplikację”.

Note: Wypada z pierwszej wersji apki.

Weryfikacja ręczna przez organizatora

Aplikacja w żaden sposób nie obsługuje płatności.

Organizator wypełnia pole tekstowe, które wyświetla się użytkownikowi gdy ma opłacić urzestnictwo.

Następnie za pomocą interfejsu administracyjnego zaznacza kto zapłacił.

Płatność przelewem

Nie różni się niczym od weryfikacji ręcznej... poza tym że zamiast pola tekstowego pojawia się pole na numer konta, która posiada walidację czy dany numer konta jest poprawny.

Płatności Dot Pay

Aplikacja obsługuję opłatę przez DotPay.

Organizator podaje numer konta Dot Pay na które będą przesyłane pieniądze, oraz inne dane konieczne do zrealisoania płatności.

Aplikacja samodzielnie rejestruję wpłtę.

4.1.9 Rejestracja

Rejestracja to wiersz w tabeli który zawiera łączy użytkownika z podwydarzeniem (atrakcją) i informuje o statusie rejestracji użytkownika na atrakcję.

Stany rejestracji:

nowa

Stan zaraz po stworzeniu

wypełniona

Po wypełnieniu ankiety

płatnść w toku

Użytkownik rozpoczął proces opłacania wydarzenia.

Rejestracja zakończona

Wszystkie kroki powiązane z rejestracją są zakończone.

Note: Stan ten można czasem wywnioskować ze stanu innych encji w systemie, ale nie zawsze. Przykładowo organizator może uznać że ktoś jest zapłacony (nawet jeśli dana atrakcja jest płacona przez dot pay więc weryfikacja płatności jest automatyczna) — powód może być taki że pewna grupa osób może mieć uprawnienie do darmowego uczestnictwa w zlocie.

4.1.10 Rejestracja na zajęcia

Niektóre atrakcje mogą wymagać dodatkowej rezerwacji na zajęcia/warsztaty czy coś podobnego.

Scenariusze użycia w ZHP które chcemy spełnić:

- Rejestracja na warsztaty podczas LAS.
- Rejestracja na zajęcia dla grup harcerzy na Zlocie w Krakowie.

Note: Wypada z pierwszej wersji.

4.1.11 Walidacje dostępu do wydarzenia

Note: Wydaje mi się że walidację należałoby rozbić na dwa etapy: przed płatnością i po platności. Na przykład walidacją przed płatnością byłoby sprawdzenie że użytkownik ma stopień harcmistrza (na przykład na Zlocie Harcmistrzów...) a walidacja po platności to sprawdzenie wykonania zadania przedrajdowego.

Na razie implementujemy walidację przed płatnością.

Note: TODO przemyśleć mechanizm uelastyczniania walidacji.

Lista walidacji jakie będą potrzebne w wersji harcwrsjiej:

- Sprawdzenie stopnia instruktorskiego
- Sprawdzenie wieku

4.1.12 Dynamiczne dane do formularza rejestracji

Generalnie każde wydarzenie będzie zbierało *jakieś* dodatakowe dane o każdym zgłoszeniu. Chcemy by organizator mógł do każdej atrakcji podpiąć dodatkowy formularz rejestracji z dynamiczną zawartością.

Synchronizacja dynamicznych dancyh między formularzami

Żeby użytkownik nie musial wpisywać danych wielokrotnie powinniśmy umożliwić mechanizm automatycznego uzupełniania danych które powtarzają się między ankietami.

Mechznizm ten działa następująco: Pole o nazwie `foo` otrzymuje początkowo wartość z pola o nazwie `foo` w ostatnio wypełnionej ankiecie zawierającej to pole.

Note: Potem może wymyślimy coś bardziej błyskotliwego.

4.1.13 Podstawowe dane

Podczas rejestracji użytkownik dla każdego wydarzenia podaje ten sam zestaw podstawowych danych.Podstawowe dane są konfigurowane na poziomie wdrożenia, ale dwa zestawy podstawowych danych będą zdefiniowane.

Dla wersji ogólnej będzie to:

- Imie
- Nazwisko
- Adres
  - Podzielony na atomowe dane

Dla wersji harcerskiej:

- Imie
- Nazwisko
- Numer PESEL
- Numer karty członkowskiej (organizator wybiera czy pole to jest obowiązakowe(
- Adres
  - Podzielony na atomowe dane
- Stopień harcerski
- Stopień instruktorski

Note: Całość można zamiemienić na system z wykorzystaniem dynamicznych ankiet. Reszta informacji o decyzji na `REJCEN-26`.

4.2 Generyczny mechanizm pluginów

Sporo rzeczy w tej aplikacji będzie zmienialne na poziomie wdrożenia, dobrze byłoby mieć jakiś wspólny mechanizm pluginów który pozwalałby elegancko podmieniać poszczególne używane modele w Django.

Na pewno za pomocą pluginów opisywane będą:

- Podstawowe informacje podawane podczas rejestracji (przez użytkownika)
- Podstawowe informacje o wydarzeniu
- Podstawowe informacje o podwydarzeniu
- Podstawowe informacje o organizatorze

Note: Prawdopodobnie większość z tych scenariuszy zastąpimy dynamicznymi ankietami.

Ale to jest otwarty temat.

4.2.1 Implementacja pluginów za pomocą dynamicznych formularzy

Osoba wdrażająca aplikację tworzy dynamiczny formularz. Następnie w adminie na poziomie bazy danych ustala że dynamiczny formularz o danym ID jest dodawany do każdej rejestracji.

Analogiczny mechanizm będzie działal dla danych organizatora.

Note: Procedura zmiany tego formularza wyglądałaby tak że: nowo tworzone rejestracje miałyby już doklejane nowe dane, a stare działałyby na danych starych.

4.3 Integracja z ESHD

4.3.1 Rejestracja jednoosobowa

Tutaj integracja jest prosta, za pomocą: numeru PESEL, numery karty, imienia i nazwiska sprawdzamy czy ktoś jest w ESHD. Jeśli go nie ma to odrzucamy osoba nie może się zarejestrować.

4.3.2 Rejestracja wieloosobowa

Note: Wypada.

4.4 Raporty

TODO

4.5 Scenariusze użycia

4.5.1 Logowanie i zakładanie konta

Logowanie

Note: Zasadniczo logowanie zostaje poza zakresem głównej aplikacji, powinna być możliwość doklejenia dowolnego mechanizmu logowania.

System pozwala na logowanie za pomocą dwóch metod:

Loginem i hasłem

By zalogować się należy podać swój login i hasło.

Logowanie i zakładanie konta robimy za pomocą `django-registration`.

Note: W przyszłości zrobimy logowanie emailem.

Za pomocą konta ‘‘zhp.net.pl‘‘ (mechanizm openid)

By zalogować się należy kliknąć w odpowiedni guzik, który wykona procedurę logowania open-id.

4.5.2 Utworzenie wydarzenia

Każdy ma prawo stworzyć nowe wydarzenie. Użytkownik klika guzik: dodaj wydarzenie i przenosi go na formularz dodawania wydarzenia.

Note: Ewentualnie można rozważyć wymaganie posiadania odpowiedniego przywileju django.

Formularze ten zawiera podstawowe dane wydarzenia oraz dane organizatora (opis tutaj: *Wydarzenie (Event)*, oraz *Organizator (Organizer)*).

Użytkownik wypełnia ten formularz i jeśli nie ma błędów wydarzenie dodaje się w stanie: ‘Nowe’.

4.5.3 Dodanie podwydarzenia

Użytkownik dodał już wydarzenie i teraz dodaje podwydarzenie. Znajduje swoje wydarzenie i klika: dodaj podwydarzenie.

Wypełnia podstawowe dane podwydarzenia (patrz: *Podwydarzenie/atrakcja (Attraction)*).

Jeśli dane są poprawne do wydarzenia dodaje się podwydarzenie.

Note: Na liśice wydarzeń organizator wydarzenia widzi jego status.

4.5.4 Dodanie płatności do podwydarzenia

Podwydarzenie domyślnie jest bezpłatne, po jego dodanoiu na liście podwydarzeń w wydarzeniu pojawia się guzik “Dodaj płatność”, po jego kliknięciu użytkownik widzi formularz zawierający:

- Typ płatności (patrz: *Metoda opłaty (typ płatności)*).
- Kwotę płatności (nie pojawia się dla darmowej płatności).
- Dodatkowe informacje określane przez typ płatności.

4.5.5 Dodanie (dynamicznej) ankiety do podwydarzenia

Domyślnie podwydarzenie nie ma dynamicznej ankiety.

Po dodaniu podwydarzenia na liście podwydarzeń w wydarzeniu pojawia się guzik “Dodaj ankietę”.

Po jego kliknięciu organizator widzi listę już dodanych pytań z możliwością ich edycji oraz formularz umożliwiający dodanie pytania.

TODO opisać dokladniej.

4.5.6 Wyłączenie edycji wydarzenia po włączeniu rejestracji

Administrator ma guzik: “Włącz rejestrację na wydarzenie” po jego kliknięciu widzi panel: “Po włączeniu rejestracji nie będziesz mógł modyfikować wydarzenia”.

Jeśli kliknie “OK”:

- zmienia się stan wydarzenia,
- można się na nie rejestrować,
- wydarzenie nie jest edytowalne.

4.5.7 Weryfikacja wydarzeń

Wysłanie wydarzenia do weryfikacji

Note: Jest to rozwinięcie scenariusza z punktu poprzedniego.

Po dodaniu wszystkich podwydarzeń organizator klika na guzik: rozpocznij zbieranie zgłoszeń.

Jeśli w konfigiuracji systemu *nie wymagamy* weryfikacji wydarzeń wydarzenie otrzymuje status: **zaakceptowane**. Jeśli data rozpoczęcia zbierania zgłoszeń już minęła status zmienia się na **otwarte**.

Jeśli wymagamy weryfkiacji to status zmienai się na: **Do akceptacji**, oraz:

- Administrator aplikacji otrzyma wiadomość e-mail o konieczności weryfikacji danego wydarzenia.
- Organizator dostanie e-maila o konieczności weryfikacji.

Weryfikacja wydarzena

Administrator w panelu administracyjnym ma listę wydarzeń do potwierdzenia.

Dla każdego z nich może zaakceptować je lub odrzucić.

- Zaakceptowane otrzymuje status **zaakceptowane**
- Odrzucone otzymuje status **nowe** (można ją zmienić i przesłać do akceptacji ponownie).

W obu przypadkach organizator otrzymuje wiadomość e-mail.

W przypadku odrzucenia rejestracji administrator musi podać powód, który pojawi się w mailu do organizatora.

4.5.8 Automatyczna zmiana stanów wydarzeń

Dodajemy komendę administracyjną django (django management command), która przy wywołaniu odświerza stan rejestacji.

Generalnie zakładam że przy niektórych zmianach stanu rejestracji, powiązanych z upływem czasu (otwarcie, zamknięcie) będziemy do użytkowników wysyłać wiadomości e-mail z informacją. Taka funkcjonalność musi siedzieć w cronie.

4.5.9 Obsługa płatności wersja 1.0

Uzytkownik po podaniu danych przekierowywany jest na widok z płatnością, zawartość tego widoku jest zależna od rodzaju płatności.

Zadanie techniczne: API płatności

Wykonanie API obsługującego typy płatności (patrz *Metoda opłaty (typ płatności)*).

Zadanie techniczne: obsługa rejestracji trwających długo

Płatność będzie odbywała się asynchronicznie, i może trwać kilka dni.

Zatem musi być jakaś obsługa tego schematu, żeby użytkownik najpierw widział ekran: “Płatność w realizacji”, a potem dostał wiaodmość e-mail oraz: “Płatność zakończona”

Note: Możliwe będą dodatkowe kroki rejestracji po płatności, na przykład wybór zajęć.

Podpinanie płatności do atrakcji

Organizator ma możliwość podpięcia płatności do atrakcji.

Obsługa bezpłatnej płatności

Informujemy użytkownika że dana atrakcja jest bezpłatna, wyświetlamy komunikat dodany przez organizatora. Po kliknięciu dalej użytkownik przechodzi na kolejny krok rejestarcji.

Obsługa płatności przelewem

Użytkownik widzi informację o konieczności oplaty przelewem. Do póki płatość nie zostanie ręcznie odnotowana przez administratora to ciągle widzi ekran: "Płatność w realizacji", po odnotowaniu płatności otrzymuje wiadomość e-mail o tym fakcie.

4.5.10 Rejestracja

Użytkownik z listy wydarzeń wybiera interesujące go wydarzenie, oraz wybiera podwydarzenie na które chce się zarejestrować.

Wypełnia dane do rejestracji i klika dalej, użytkownik jest zarejestrowany.

4.5.11 Wyświetlanie listy zarejestrowanych osób

Po wybraniu swojego wydarzenia organizator ma dostęp do strony na której może zarządzać wydarzeniem.

W ramach zadrządzania ma możliwość wyświetlenia listy osób które się zarejestrowały na to wydarzenie.

Może:

- filtorwać i sortować listę pod kątem: podstawowych danych i danych z dynamicznego formularza,
- widzieć w tabeli wszystkie dane o rejestracji (łącznie z dynamicznymi),
- ręcznie zatwierdzać płatności.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*