Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Information Technology

Department of Software Engineering

Bachelor's thesis

ElateMe - Backend

Yevhen Kuzmovych

Supervisor: Ing. Jiří Chludil

24th April 2017

Acknowledgements

THANKS (remove entirely in case you do not with to thank anyone)

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 24th April 2017 . . . .

Czech Technical University in Prague
Faculty of Information Technology
© 2017 Yevhen Kuzmovych. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Kuzmovych, Yevhen. *ElateMe - Backend*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

V několika větách shrňte obsah a přínos této práce v českém jazyce.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

Summarize the contents and contribution of your work in a few sentences in English language.

Keywords ElateMe, crowdfunding platform, social network, back-end API, RESTful, payments systems

Contents

Introduction **1**
ElateMe . . . 1
Aim of the thesis . . . 1
Motivation . . . 1

1 Analysis **3**
1.1 BI-SP1 and BI-SP2 subjects . . . 3
1.2 Functional requirements . . . 3
1.3 Non-functional requirement . . . 5
1.4 Use cases . . . 5
1.5 Domain model . . . 5
1.6 System structure . . . 6
1.7 Authentication . . . 7
1.8 Payments system . . . 7
1.9 Push notifications . . . 8

2 Design **11**
2.1 Authentication . . . 11
2.2 Server API . . . 12
2.3 Chosen technologies . . . 14
2.4 Class model . . . 15

3 Implementation **19**
3.1 Django project structure . . . 19
3.2 Social integration . . . 19

4 Testing **21**
4.1 Unit tests . . . 21
4.2 Apachebench . . . 23

Conclusion **25**

Bibliography **27**

A Acronyms **29**

B Contents of enclosed CD **31**

List of Figures

1.1 Component diagram 6
1.2 Mechanism of push notifications 8

2.1 Authentication activity diagram 12
2.2 Social integration class model 16
2.3 Push notification class model 17

Introduction

!!!! START !!!!

ElateMe

ElateMe is a new crowdfunding platform with elements of the social network. Unlike other similar projects like Kickstarter or Patreon that help bring creative, commercial projects to life by means of interested people, ElateMe is focused on fulfillment of personal wishes with the help of user's friends. The user can create a wish and set its cost, title, and a short description. His friends then will be able to contribute to his wish by donating money. When wish gathers needed amount, money will be transferred to the user bank account. The social part of the application is providing an ability for the user to connect with his friends, communicate with other users, rate and comment others' wishes.

Aim of the thesis

The aim of this thesis is to analyze functional, non-functional requirements and use cases of the project, design database model and server architecture, implement back-end Application Programming Interface (API) and payments system for this service.

Motivation

The main goal for the author of the thesis is to analyze and learn tools for web back-end development such as Python programming language and Django web framework, practice building complex systems using them, learn to design server architecture and explore various online payment systems.

Chapter 1

Analysis

This chapter will focus on analysis of the project as a part of a software development that connects customer's requirements to the system and its following design and development.

Analysis of software project is intended to define detailed description of the product, break it down into requirements to the system, their systematization, detection of dependencies, and documentation.

1.1 BI-SP1 and BI-SP2 subjects

The work on the ElateMe project started within the framework of the BI-SP1 subject. Our development team devided into groups: Android, iOS and Back-end developers. Our task was to define and document main client's requirements, implement functioning prototypes of mobile applications and back-end server API. During BI-SP1 subject, Maksym Balatsko was working on prototype of back-end server, so choise of used technologies was up to him. Then the technology stack was agreed with supervisor of the project. Chosen technologies will be discussed in the next chapter.

Because of changes in requirements and provided a new interface design of the mobile applications, analysis and its documentation has undergone certain changes. And at the start of BI-SP2 subject implementation of back-end API, on which Maksym and I worked, has started.

1.2 Functional requirements

1.2.1 Authorization

F1 Sign up via Facebook User will be able to sign up to ElateMe application with his Facebook account. Application will load user's data such as name, surname, email, date of birth, etc.

F2 Logout Authorized user will be able to log out. In this case he will also stop receiving any notifications from the application.

F3 Load friends from social network On initial login application will load list of user's friends that are already signed up in this application. This users will be considered as friends in ElateMe application.

1.2.2 Friendship management

F4 View friends list User will be able to view list of his Facebook friends that are already signed up in application.

F5 Create friends group User will be able to create friends group. Groups will be used for simplification of friends management.

F6 Delete friends group User will be able to delete friends group.

1.2.3 Wish management

F7 Create wish User will be able to create wish, set it's title, description, price(amount of money that he(user) wants to gather), and deadline.

F8 Delete wish User will be able to delete his wish if nobody will have donated money yet.

F9 Close wish User will be able to close his wish. Money that will have been gatherd on this wish will be refunded to donators.

F10 View users' wishes list User will be able to browse wishes lists of his friends.

F11 Create surprise wish User will be able to create surprise wish for one of his friends. In this case user to whom the wish was addressed will not have acces to it and will not know about it until the whole amount is collected.

F12 View contributed wishes list User will be able to view list of wishes he will have contributed to.

1.2.4 Feed

F13 View user's feed User will recieve feed with latest wishes of his friends.

1.2.5 Donation management

F14 Donate to wish User will be able to financially contribute to wishes of his friends.

F15 Refund In the case of the closure of the wish, all gatherd money will be refunded to donators.

1.2.6 Comments management

F16 View wishes comments list User will be able to view list of comments under the wish he will be browsing.
F17 Comment wish User will be able to leave a comment under the wish.
F18 Delete comment User will be able to delete his comment.

1.3 Non-functional requirement

1.3.1 Back-end API

N1 Representational State Transfer (REST)ful Back-end API will follow architectural constraints of REST architectural style.
N2 HyperText Transfer Protocol Secure (HTTPS) Server will comunicate with client via HTTPS.
N3 PostgreSQL database PostgreSQL will be used as the main DBMS.

N4 Performance Server will be able to serve 1500 requests per second.

1.3.2 Payments

N5 FIO-bank User will be able to make payments via FIO-bank.
N6 Bitcoin User will be able to make payments via Bitcoin.
N7 Secure payments System will ensure secure payments.
N8 Consistency Servers data about payments will be consistent with data in payments systems (FIO-bank, Bitcoin, etc.). System will react accordingly to errors appeared during payments.

1.4 Use cases

** insert Use cases diagram **

1.5 Domain model

** insert Domain model diagram **

1.6 System structure

The whole ElateMe application system is devided into components. Main components are server, Android and iOS clients.

Detailed structure of server and its connection with external interfaces are presented at the component diagram 1.1. As seen in the diagram, server provides interface for mobile applications to communicat via REST API. Server also uses interfaces of Facebook (Graph API) to recieve needed data about users and interfaces of payment systems (FIO-bank and Bitcoin) for payments processing.

Inside the server is divided into components that are responsible for storing and processing data of application entities. This components are called *apps* in Django. Apps communicate with database via Django *models*. Models in Django is an interface designed to simplify querying to database.

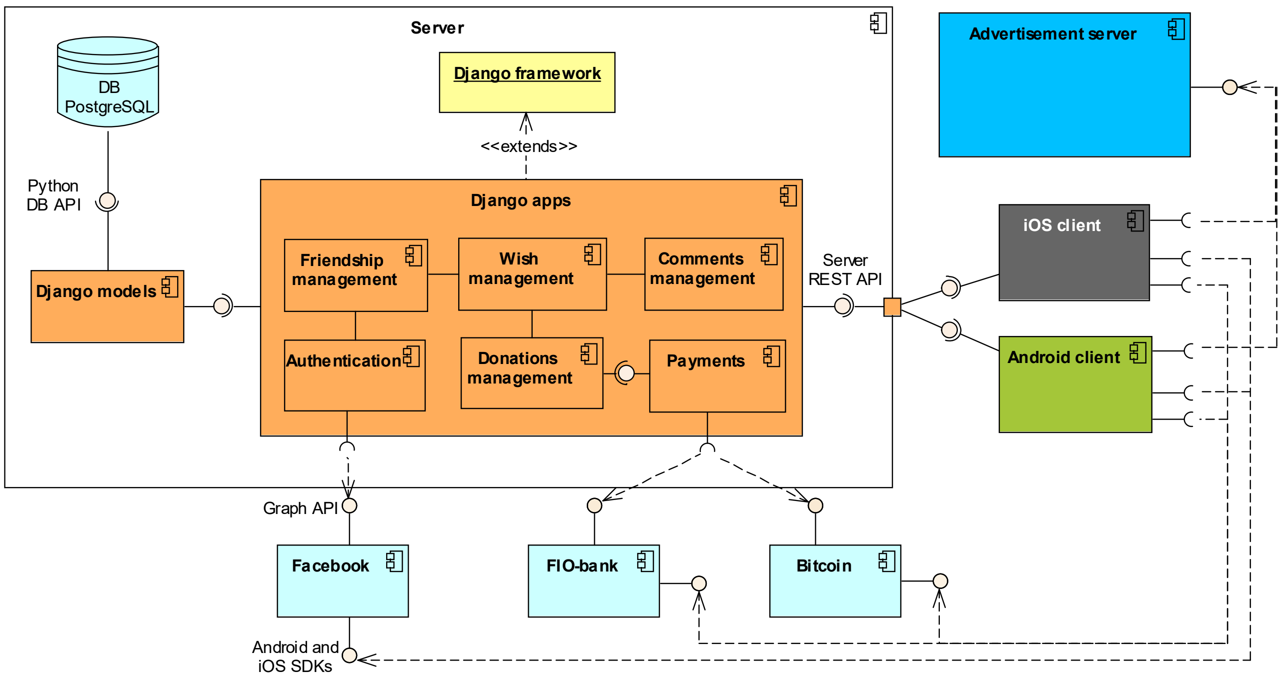

Figure 1.1: Component diagram

The diagram also shows the use of interfaces of Facebook and payment systems by mobile clients, but they are not a part of my work, so their design and implementation will not be described in this thesis.

1.7 Authentication

User has to be authorized to use the application. ElateMe application will not provide in-app registration. User authentication will be conducted exclusively through third-party systems. It is made to simplify the registration in the application.

1.7.1 Facebook

User authentication will be conducted through his Facebook account.

After first login, application will get from Facebook needed information about the user: first name, last name, email address, list of user's friends. User's Facebook friends who are already logged in to the application, automatically become his friends in the ElateMe.

Despite the lack of in-app registration, user's information recieved from Facebook will be stored in ElateMe system as well, because user will be able to add other users to his friend list, create friends groups independently from Facebook.

1.8 Payments system

The application is based on croweding. So application (and the server in particular) has to provide service for payments processing. According to the requirements, this service should use interfaces of FIO-banka and Bitcoin.

1.8.1 Use cases

In this application, the payment system participates in the following use cases:

- **Donation**

  During the donation, the money is transferred to the internal account of ElateMe, where it is stored until one of the following use cases.

- **Wish completion**

  In case of fulfillment of the wish, money is transferred to the account of the author of this wish.

- **Wish closing**

  There are two situations in which the wish is closed: closing of wish by its author and closing upon expiration of deadline. In both cases, already collected money is returned to the donators' accounts.

1.8.2 FIO-banka

1.8.3 Refundation

1.9 Push notifications

Push notifications are small important messages from the application or service, displayed by the operating system when the user does not directly work with the specified application or service. The advantage of such notifications is that there is no need to keep the program in memory, spending on it processor powers and memory.

In ElateMe application the user will receive information about the state of his wishes, new donations, comments, etc.

1.9.1 Mechanism of push notifications

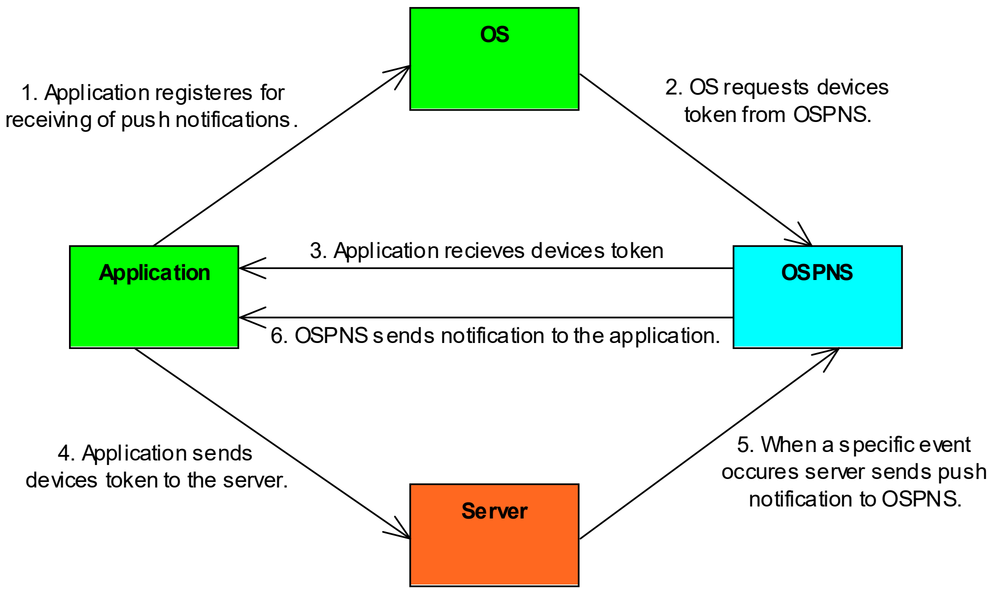
OS
1. Application registeres for receiving of push notifications.
2. OS requests devices token from OSPNS.
Application
3. Application recieves devices token
OSPNS
6. OSPNS sends notification to the application.
4. Application sends devices token to the server.
5. When a specific event occures server sends push notification to OSPNS.
Server

Figure 1.2: Mechanism of push notifications

For server to be able to send push notifications it needs to store a token. Token is a line of characters that serves as an address of specific application on the specific device. Token is generated by Operating system push notification service (OSPNS). After application is installed on the device it registers itself for receiving of push notifications, OS requests token from OSPNS, application receives token and sends it to the server.

1.9.2 Actors

- **OSPNS**

  Every operating system has its own service for processing push notifications. They are Google Cloud Messaging (GCM) for Android and Apple Push Notification service (APNs) for iOS. As shown on the diagram 1.2 OSPNS sends a token to the application when it registers in the service and sends the push notifications to the application itself.

- **Server**

  The server stores the tokens of each individual device and sends the push notifications to OSPNS.

- **Client application**

  The application is registered to receive push notification, receives a token from OSPNS and sends it to the server.

Chapter 2

Design

2.1 Authentication

As was mentioned earlier in this thesis, authentication of the user will be conducted through his Facebook account. Facebook provides interface for user authentication in third-party applications. This interface uses OAuth 2.0 protocol.

2.1.1 OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 supersedes the work done on the original OAuth protocol created in 2006. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. [1]

For server to be able to get list of friends and other information about user, mobile application needs to recieve token from Facebook with appropriate permissions and send it to the server. Token is a line generated by Facebook and by which Facebook provides access to certain data of certain user.

Diagram 2.1 shows mechanism of successful authentication via user's Facebook account.

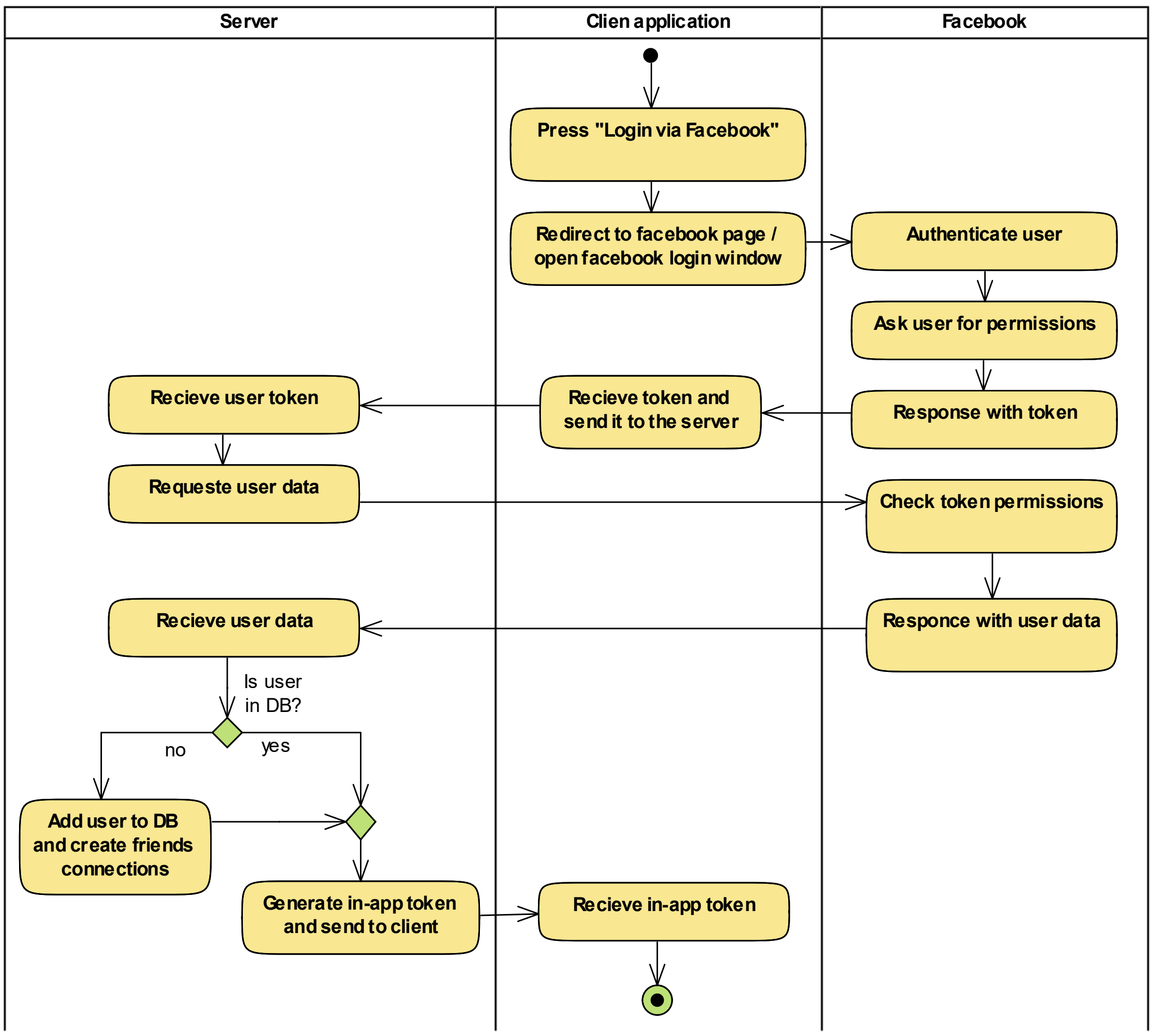

Figure 2.1: Authentication activity diagram

2.2 Server API

2.2.1 REST

Server API will be built on the basis of REST. REST is the architectural solution for the transfer of structural data between server and client [2]. API is considered RESTful if it follows certain rules [3]:

- **Client-Server**

  Client-Server defines a clear separation between a service and its consumers. Service (in this case server) offers one or more capabilities and listens for requests on these capabilities. A consumer (in this case mobile client) invokes a capability by sending the corresponding request message, and the service either rejects the request or performs the requested task before sending a response message back to the consumer.

- **Stateless**

  Statelessness ensures that each service consumer request can be treated independently by the service. The communication between service consumer (client) and service (server) must be stateless between requests. This means that each request from a service consumer should contain all the necessary information for the service to understand the meaning of the request, and all session state data should then be returned to the service consumer at the end of each request.

- **Cache**

  Responses may be cached by the consumer to avoid resubmitting the same requests to the service. Response messages are explicitly labeled as cacheable or non-cacheable. This way, the service and/or the consumer can cache the response for reuse in later requests.

- **Uniform Interface**

  All services and service consumers within a REST-compliant architecture must share a single, overarching technical interface. As the primary constraint that distinguishes REST from other architecture types, Interface is generally applied using the methods and media types provided by HTTP.

- **Layered System**

  A REST-based solution can be comprised of multiple architectural layers, and no one layer can "see past" the next. Layers can be added, removed, modified, or reordered in response to how the solution needs to evolve.

There is also an optional constraint **Code-On-Demand**. This constraint states that client application can be extended if they are allowed to download and execute scripts or plug-ins that support the media type provided by the server. Adherence to this constraint is therefore determined by client rather than the API [2].

2.2.2 Apiary

The apiary service will be used for the server API documentation. It is a powerful API design stack [4]. In the apiary, the Blueprint API is used to describe the structure of the APIs. API Blueprint is a powerful high-level API description language for web APIs [5].

2.3 Chosen technologies

As I mentioned before, the choice of used technology was not up to me so in this section I will not describe why certain technologies were chosen, but will describe their advantages (alternatively disadvantages) for this project.

2.3.1 Python

Python is a base of the server. It was chosen as a primary programming language because it was designed to be simple and highly readable which is very important for large-scale projects. Its syntax and standard library simplify and speed up a development.

2.3.2 Django

Django is an open source web framework for python. It provides a high level abstraction of common web development patterns. It follows Model-View-Controller (MVC) design pattern. Django uses MVC to separate model as a data and a business logic of the application, view as a representation of the information for the user, in this case, the client side of the application and controller as an interface of the application, in this case, set of URLs to communicate with front-end [6].

2.3.3 Django REST

Django REST framework is an open source project built on Django framework. It contains needed tools for implementation of RESTful API.

2.3.4 PostgreSQL

On initial stage of the development, SQLite will be used as a DataBase Management System (DBMS), because it does not require a standalone database server and is simple to set up. The database will be changed and migrated to PostgreSQL later.

PostgreSQL is powerful, open source relational DBMS. It has advanced features such as full atomicity, consistency, isolation, durability [7]. Django framework provides great API for working with PostgreSQL databases.

2.3.5 nginx

nginx [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server, originally written by Igor Sysoev [8]. According to Netcraft [9], nginx served or proxied 28.50% busiest sites in March 2017.

2.4 Class model

Class model of the project was built based on Django project structure. Most of the classes extend Django classes following certain rules and format. Therefore I will describe only parts that don't depend on Django structure. A complete model of classes can be found in attachments.

2.4.1 Authentication

As I said before, in frameworks of this work user authentication will be conducted through his Facebook account. But in the future, other social networks and/or authorization methods may be added. Since most of social networks support OAuth 2.0 [1], this allows you to make a universal interface for user authorization, that will process user authentication via social network using token provided by client (mobile or web application).

Thus, as shown in the diagram 2.2, classes that extend **AbstractSocialAPI** will provide interface for token processing. **AbstractSocialAPI** defines method *process* that receives token from client, requests information about the user from the corresponding social network, decides if user is already registered in the application, registers him (adds to database) if necessary, and authorizes user in application. Information about user is obtained using abstract methods like *request_data*, *get_social_id*, *get_friends*, etc.

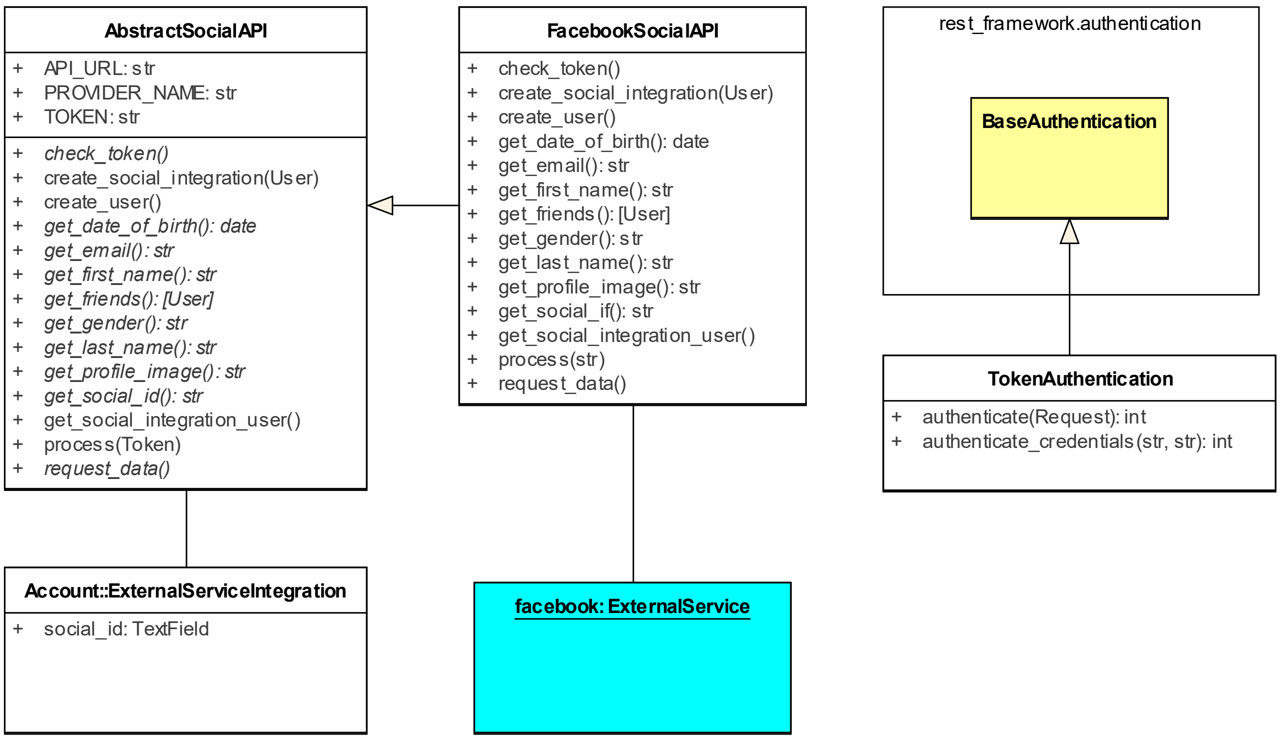

Figure 2.2: Social integration class model

2.4.2 Push notifications

As shown in the diagram 2.3, the **EventHandler** class will be responsible for processing of events which require user notification. This class uses interface of **AbstractNotificationService** for sending of push notifications to the corresponding OSPNSs. Since the user can have several devices with the installed application, there may be situations when one notification will be sent to several tokens and/or different OSPNSs. Classes that extend **AbstractNotificationService** and implement method *notify* will be responsible for sending push notifications to the specific OSPNSs.

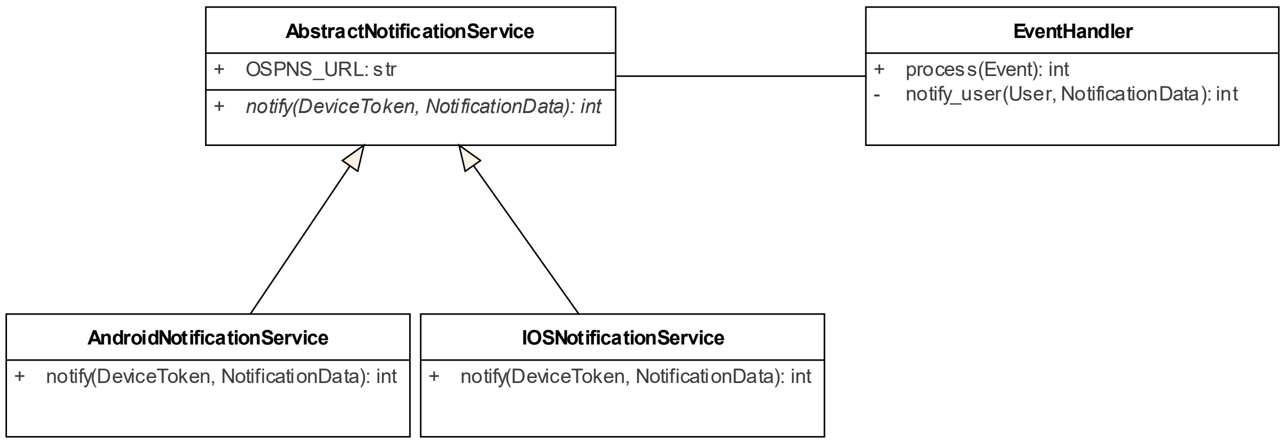
AbstractNotificationService
+ OSPNS_URL: str
+ notify(DeviceToken, NotificationData): int
EventHandler
+ process(Event): int
- notify_user(User, NotificationData): int
AndroidNotificationService
+ notify(DeviceToken, NotificationData): int
IOSNotificationService
+ notify(DeviceToken, NotificationData): int

Figure 2.3: Push notification class model

2.4.3 Payments

Chapter 3

Implementation

3.1 Django project structure

Django as a framework determines the structure of the whole system. Django project is divided into logical parts - apps. Apps contain a set of modules with classes, that implement interfaces and extend classes, that are provided by Django.

3.1.1 Django models

Django models is an interface for simplified querying to database. All models extend class **model** from *django.db* module and usually represent single table in a database [10].

3.1.2 Django views

3.2 Social integration

Chapter **4**

Testing

4.1 Unit tests

Alongside with the development automatic testing was performed using unit tests. Unit tests are designed to verify the correct functioning of the parts of the application.

Unit testing code means validation or performing the sanity check of code. Sanity check is a basic test to quickly evaluate whether the result of calculation can possibly be true. It is a simple check to see whether the produced material is coherent. [11]

4.1.1 Django REST tests

For testing, native tools were used, namely the Django REST framework tests. Similar to java JUnit tests Django tests are class-based. Every test case is a method of the class, that extends **APITestCase** from the module *rest_framework.test*. Classes can also contain the following methods:

- setUp: Method called to prepare the test fixture.
- tearDown: Method called immediately after the test method has been called and the result recorded.

For testing, Django creates a separate empty database independent of the main database. Sqlite DBMS is used, for testing in this project.

4.1.2 Auxiliary methods

For testing of the server API, I wrote a set of auxiliary methods that simulate HTTP requests to the server. This methods take URL to which the requester is sent and optionaly information (JSON) which is sent as the body of the request. Methods use **APIRequestFactory** to perform requests to the server.

Methods also use *force_authenticate* function that allows to authenticate user (in this case test user) in the system without involvement of Facebook. This function is used for testing of request that require authorization.

4.1.3 Test cases

As an example of a test, I'll take the creation of the wish by the user.

Initially in the method *setUp* I create test user, after that I make a POST request to the server with information about the wish in the body of the request. After the server responded, I check status code of the response, compare the information between the body of the request and the body of the response (body of the response contains the newly created wish) and check that wish is added to the database.

```
from django.urls import reverse
from rest_framework.test import APITestCase
from rest_framework import status
from account.models import User, UserManager
from wishes.models import Wish

# auxiliary methods for http requests
from util.test_requests import post, get, put, patch, delete

class WishesTest(APITestCase):

  def setUp(self):
    self.url = reverse('wishes:wishes')
    self.user = UserManager().create_user('test1@test.com', 'test')

  def test_create_wish(self):
    wish_data = {
      'title': "iPhone7",
      'description': "I don't need no jack",
      'amount': 19999
    }
    status_code, response_data = post(url=self.url,
                                      user=self.user,
                                      data=wish_data)

    self.assertEqual(status_code, status.HTTP_201_CREATED)
    self.assertEqual(response_data['title'], wish_data['title'])
    self.assertEqual(response_data['amount'], wish_data['amount'])
    self.assertEqual(Wish.objects.get().title, wish_data['title'])
```

This is a positive test, so the status code must be 201 (created), wish should be created and added to the database.

4.2 Apachebench

Conclusion

Bibliography

[1] Reference. OAuth 2.0 [online]. [Cited 2017-04-02]. Available from: `https://oauth.net/2/`

[2] DAIGNEAU, Robert. *Service design patterns: fundamental design solutions for SOAP/WSDL and RESTful web services.* Upper Saddle River: Addison-Wesley, c2012. Addison-Wesley signature series, 2012, ISBN 978-0-321-54420-9.

[3] Reference. What is REST? [online]. [Cited 2017-04-02]. Available from: `http://whatisrest.com/`

[4] Reference. Apiary [online]. [Cited 2017-04-22]. Available from: `https://apiary.io/`

[5] Reference. API Blueprint [online]. [Cited 2017-04-22]. Available from: `https://apiblueprint.org/`

[6] HOLOVATY, Adrian and Jacob. KAPLAN-MOSS. *The definitive guide to Django: Web development done right. 2nd ed.* Berkeley: Apress, c2009. Expert's voice in Web development., 2009, ISBN 978-1-4302-1936-1.

[7] OBE, Regina Obe and Leo Hsu. *PostgreSQL: up and running.* Sebastopol: O'Reilly, c2012, 2012, ISBN 978-1-449-32633-3.

[8] Reference. nginx [online]. [Cited 2017-04-08]. Available from: `https://nginx.org/en/`

[9] Reference. Netcraft [online]. [Cited 2017-04-08]. Available from: `https://news.netcraft.com/archives/2017/03/24/march-2017-web-server-survey.html`

[10] Reference. Django documentation [online]. [Cited 2017-04-18]. Available from: `https://docs.djangoproject.com/`

[11] ACHARYA, Sujoy. *Mastering unit testing using Mockito and JUnit: an advanced guide to mastering unit testing using Mockito and JUnit.* Birmingham, England: Packt Publishing, 2014. Community experience distilled., 2014, ISBN 978-1-78398-250-9.

APPENDIX **A**

Acronyms

API Application Programming Interface

MVC Model-View-Controller

HTTPS HyperText Transfer Protocol Secure

DBMS DataBase Management System

REST Representational State Transfer

OSPNS Operating system push notification service

GCM Google Cloud Messaging

APNs Apple Push Notification service

Appendix **B**

Contents of enclosed CD

```
 readme.txt ...................... the file with CD contents description
 exe ................................. the directory with executables
 src ................................. the directory of source codes
  wbdcm .................................. implementation sources
  thesis ........... the directory of LaTeX source codes of the thesis
 text ..................................... the thesis text directory
  thesis.pdf ...................... the thesis text in PDF format
  thesis.ps ........................ the thesis text in PS format
```