

# Модель базы данных для мобильного приложения

---

курсовой проект по дисциплине «Проектирование баз данных»

СТУДЕНТ **ВОРОНЕНКО А.С.**

ВАРИАНТ **2**

ГРУППА **8О-406 Б**

ПРЕПОДАВАТЕЛИ **ГАВРИЛОВ Е.С.**  
**ГРЕСС Е.С.**

ДАТА **10.12.2016**

ПОДПИСЬ

# Содержание

ВВЕДЕНИЕ И ИДЕЯ	3
Дизайн бриф: Change	
Интервью	
Opportunity area	
Идеи	
ИСХОДНЫЕ СЦЕНАРИИ	5
Общий поток данных	
Сценарии для пользователя мобильного приложения	
Графический прототип интерфейса	
РЕАЛИЗОВАННЫЕ СЦЕНАРИИ	7
Пользователь	
Модератор	
СХЕМА БАЗЫ ДАННЫХ	7
КЛИЕНТ С GUI	8
Структура приложения	
Экраны	
КОНСОЛЬНЫЙ КЛИЕНТ	12
Хранимая процедура	
Триггеры	
ВЫВОДЫ	13
ИСТОЧНИКИ	15
ПРИЛОЖЕНИЕ 1. КОД GUI КЛИЕНТА	16
ПРИЛОЖЕНИЕ 2. КОД КОНСОЛЬНОГО КЛИЕНТА	26

# Введение

Предварительное пользовательское исследование, область возможностей, сценарии, прототип графического интерфейса мобильного приложения и его последующее тестирование были выполнены мною в рамках финального проекта (capstone project) по специализации (серии курсов) Interaction Design (IxD), представленной UCSD (University of California San Diego) на онлайн образовательной платформе Coursera.

Ряд этих материалов включен в данную работу, чтобы представить идею тем, кто решит с ней ознакомиться.

## Дизайн-бриф: Change

В рамках IxD capstone project было предложено 3 брифа, из которых я выбрала "Перемены". Представьте себе хороший университетский кампус. Это одно из самых лучших мест для жизни. Все необходимое - от прачечной до спортзала - "под боком". Для любого времяпрепровождения легко найти компанию. (Здесь и далее для краткости я буду называть это инфраструктурой). Можем ли мы сделать обычный городской район комфортнее в этом отношении?

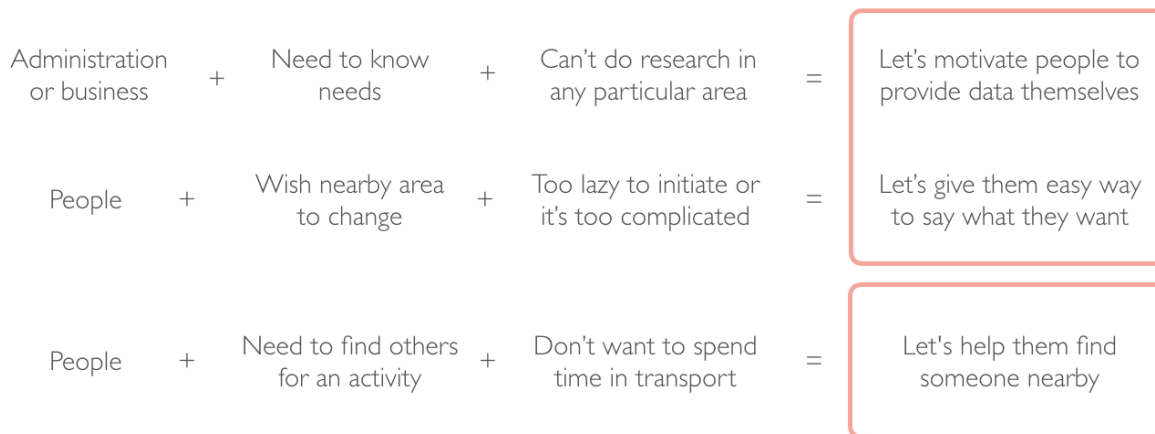
И что мы готовы для этого сделать? Многие позитивные стороны жизни на кампусе - результат инициативы и усилия не университета, а студентов и преподавателей, которые живут там или проводят большое количество времени.

## Интервью

Для разработки идеи я опросила 5 человек, живущих в разных странах: Россия, Украина, Бельгия / Израиль, Бразилия и США. Список вопросов и содержание ответов находятся за пределами данной работы, общие выводы:

- Пул проблем обычно регионально окрашен, исторически обусловлен и закреплен (транспортные проблемы в Атланте, вызванные тем, что люди по возможности живут за пределами города, "коммерческие кварталы" в Бразилии - весь малый бизнес одного типа собран в одном квартале, пресловутая проблема парковок в России, etc).
- Сами по себе люди скорее бездействуют, но гораздо лучше откликаются на чужую инициативу, чем это можно представить.
- Не "всякая домохозяйка знает, как управлять государством", но некоторые могут предложить не мало идей относительно собственного района.
- Многие проблемы можно решить за счет усилий обычных людей, не прибегая к помощи (или по крайней мере финансовой помощи) городской администрации.

# Opportunity area



## Идеи

1. Использование user-generated данных и контента для улучшения городской инфраструктуры: жалобы, запросы, сбор данных с прочих приложений. Пример последнего: решить спор о том, где строить велодорожки в Москве можно было, собрав данные со спорт-трекеров велосипедистов и аппроксимировав маршруты. Добровольцев передавать эти данные нашлось бы достаточно. Монетизируемый инструмент, так как "принимающей" стороной является городская администрация и локальный бизнес. Конкурентные примеры мне неизвестны.

2. Площадка для общения, а главное - совместных мероприятий с людьми, живущими поблизости. Пользователь видит только мероприятие, которое будет происходить в пределах заданного им радиуса. Инструмент монетизируемый, но с риском для удобства пользователей. Конкурентные примеры: [meetup.com](http://meetup.com), [next-door.com](http://next-door.com). Второй - прямой конкурент, но создан в этом году и запущен только в США и Нидерландах.

3. По умолчанию жалобы, запросы и прочее в п.1 - односторонняя связь. Мы не видим ни собственные, ни чужие запросы. В ходе тестирования интерфейса мне подавали идею сделать их видимыми (например, на карте) и дать возможность не только создавать собственный, но и поддерживать (лайкать?) чужой запрос. Эта идея в прототипе не реализована, и в модели баз учтена не будет.

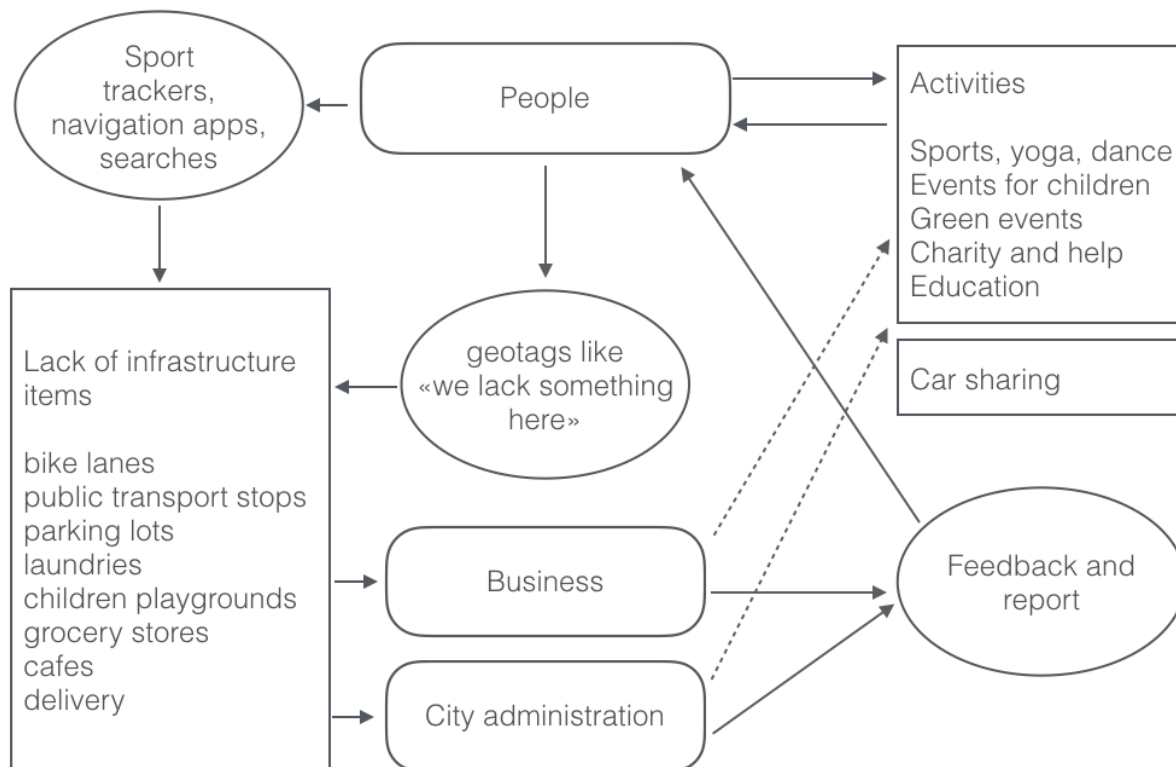
4. Wi-Fi чат. Открытый групповой чат с теми, кого вы видите в списке доступных wi-fi сетей. Немонетизируемый, конкурентные примеры неизвестны. В прототипе не реализовано.

Важно отметить, что в проделанной ранее и данной работе обсуждается только мобильное приложение для частного лица. Но на другом конце (городская администрация / бизнес) предполагается система, которая не просто собирает, но каким-то образом обрабатывает полученные данные. Это статистика по типам (запросов, жалоб), аппроксимация географических точек или маршрутов в рамках определенной области с учетом количества источников, система уведомлений (возможно, даже экстренных), формирование обратной связи ("мы узнали то и сделали это"), et cetera. То есть речь идет о сложной, в первую очередь "десктопной" системе с совместным доступом, вычислительными мощностями, безопасностью и неизвестными нам задачами. Выяснить их можно только привязавшись к конкретному региону и вникнув в процесс работы и цели соответствующей структуры (администрации / бизнеса). В данной работе эта система рассматривается как черный ящик. Процессы его работы обсуждаться не будут. Базы будут проектироваться только для данных, формируемых в рамках мобильного приложения.

# Исходные сценарии

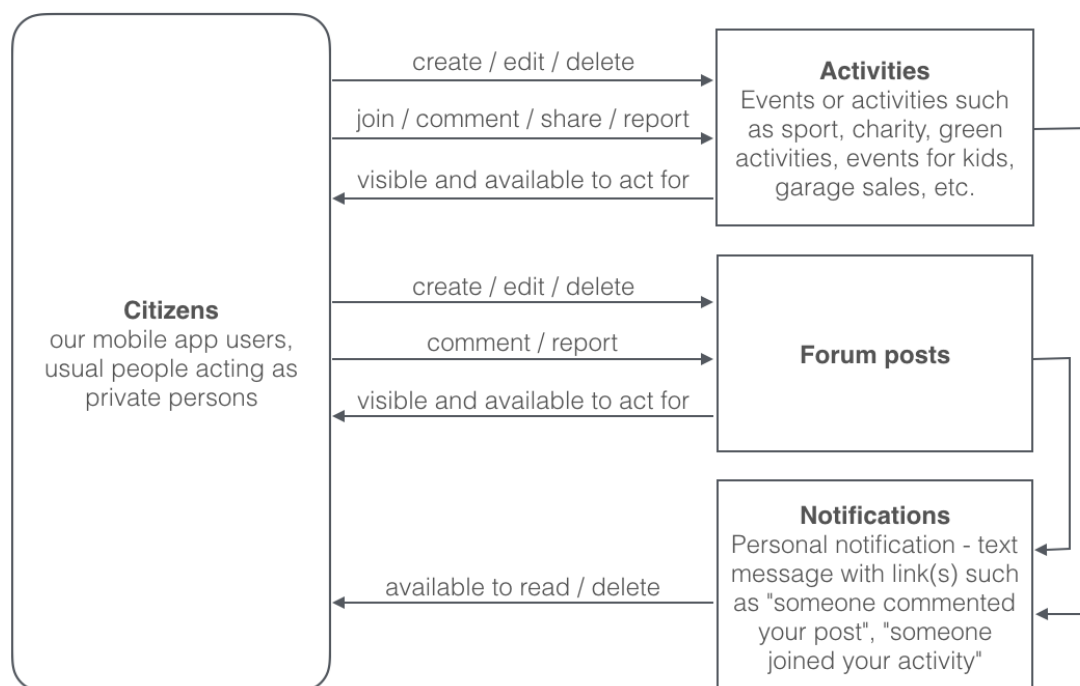
Сценарии в данном случае условное название, т.к. будут представлены скорее общие схемы потоков данных и действий - как основа для последующего проектирования данных.

## Общий поток данных

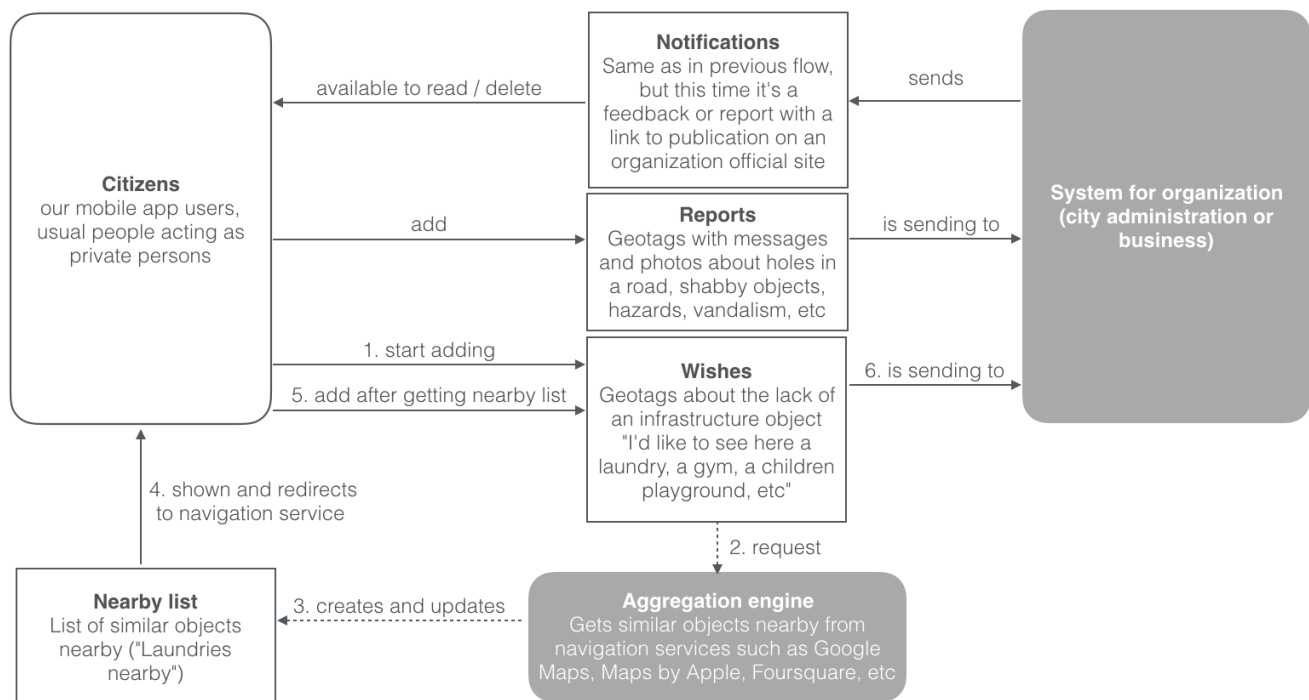


## Сценарии для пользователя мобильного приложения

### "Neighbors hangout" flow



## Citizen-to-organization flow



## Графический прототип интерфейса

Прототип интерфейса мобильного приложения опубликован на сервисе InVision и доступен по ссылке: <https://invis.io/CV851EDG4>. Он преимущественно выполнен в соответствии с гайдлайном Apple для iOS 9 с использованием одного популярного нарушения (используется в Waze, Prisma и других приложениях) - вынесения кнопки перехода к следующему шагу в сценарии из навигационной панели на место главного меню.

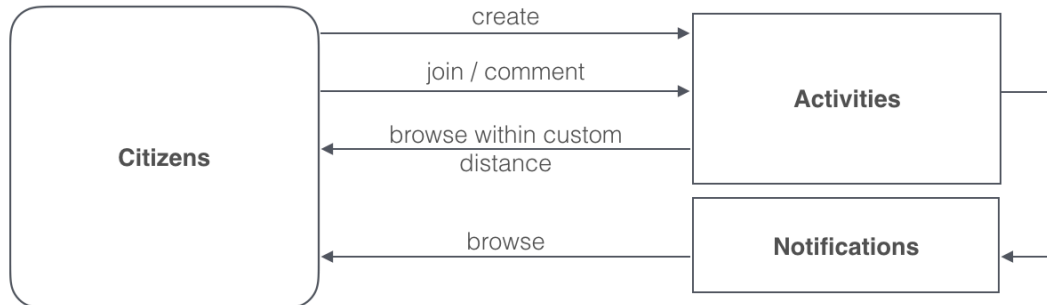
В ходе разработки проведена одна эвристическая (экспертная) оценка и два лабораторных теста: для 4 человек с протоколом "think aloud", и для 20 человек с A/B сравнением (10/10) времени выполнения задач и числа ошибок/возвратов. Статистически значимых результатов, к сожалению, получить не удалось, поэтому выбрана версия, выигрывающая по среднему.

Выбранный интерфейс содержит 5 основных экранов. Activities и Forum описаны выше и содержат фильтры по дистанции от точки, заданной как домашний адрес. Notifications и Profile содержат персональные уведомления (описаны выше) и профиль пользователя. Центральная кнопка меню - добавление нового event / forum thread / wish / report - то есть суть любого типа контента, создаваемого пользователем вручную.

# Реализованные сценарии

## Пользователь

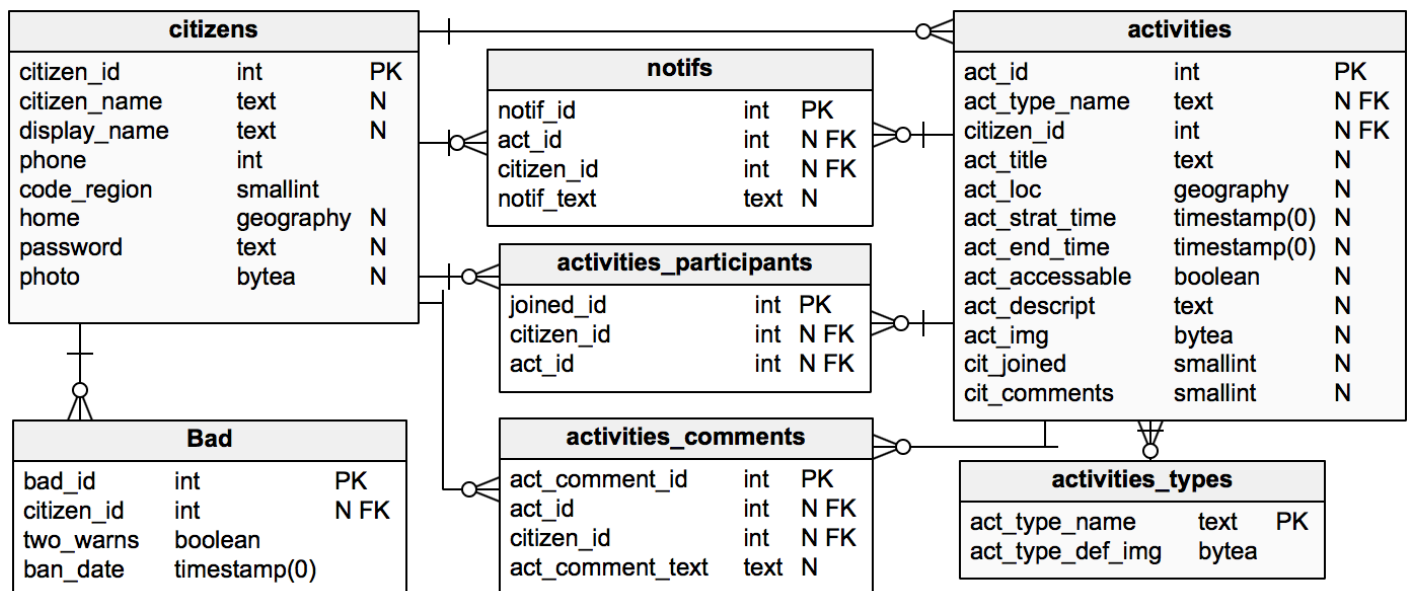
Сценарии реализованы в GUI клиенте



## Модератор

Сценарий реализован в консольном клиенте. Удаление конкретного (заданного модератором) события (activity) или комментария, нарушающего правила сервиса.

## Схема базы данных



База реализована на PostgreSQL 9.6. Дополнительно использован модуль PostGIS, реализующий тип geography. Экземпляр типа содержит координаты типа geometry и радиус Земли. Модуль также предлагает функции:

- перевода широты-долготы в geometry и обратно,
- расчета дистанции между реальными географическими координатами.

# Клиент с GUI

Инструменты:

- **Flask** - микрофреймворк Python для web-разработки.
- **Psycopg2** - модуль, который дает возможность делать SQL запросы непосредственно внутри скриптов Python.
- **Jinja2** - templating language для Python
- **WTForms** - модуль, облегчающий задачи создания форм ввода, обработки и валидации ввода для web-приложений на Python

## Структура приложения

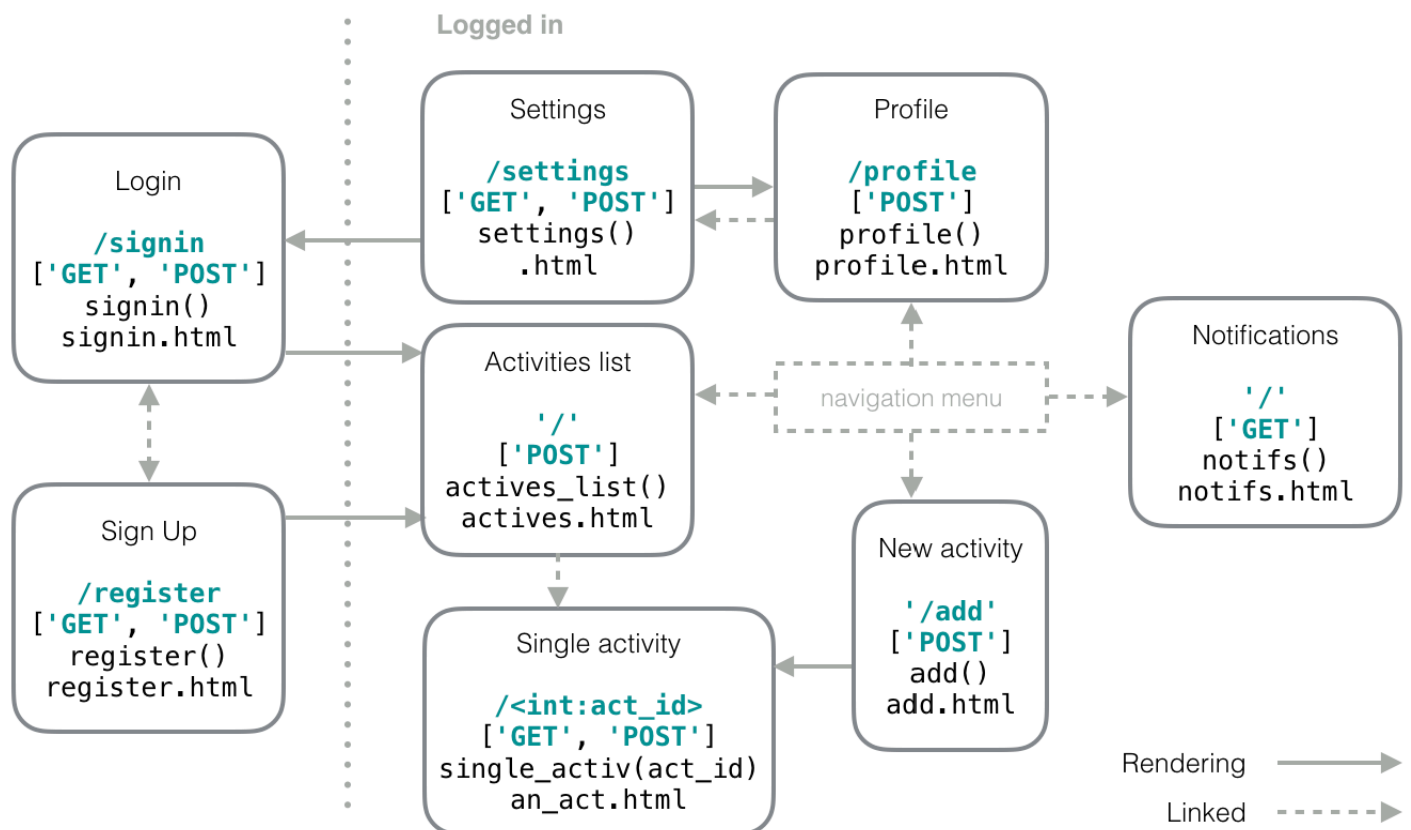
Проект Flask помимо кода по умолчанию содержит две директории - для стилей и html-файлов как таковых. Темплейт layout.html предназначен для хранения общих для абсолютно всех экранов элементов и структуры. Прочие темплейты являются его "расширением" с точки зрения Jinja2, но из Python вызываются напрямую.

Основной скрипт содержит 8 view functions. Каждым возвратом такой функции обязан быть "рендеринг" html-темплейта.

8 view functions однозначно соответствуют 8 темплейтам (экранам). Одному темплейту может соответствовать несколько view-functions, но обычно в этом нет необходимости, и некоторые разработчики считают это дурным тоном.

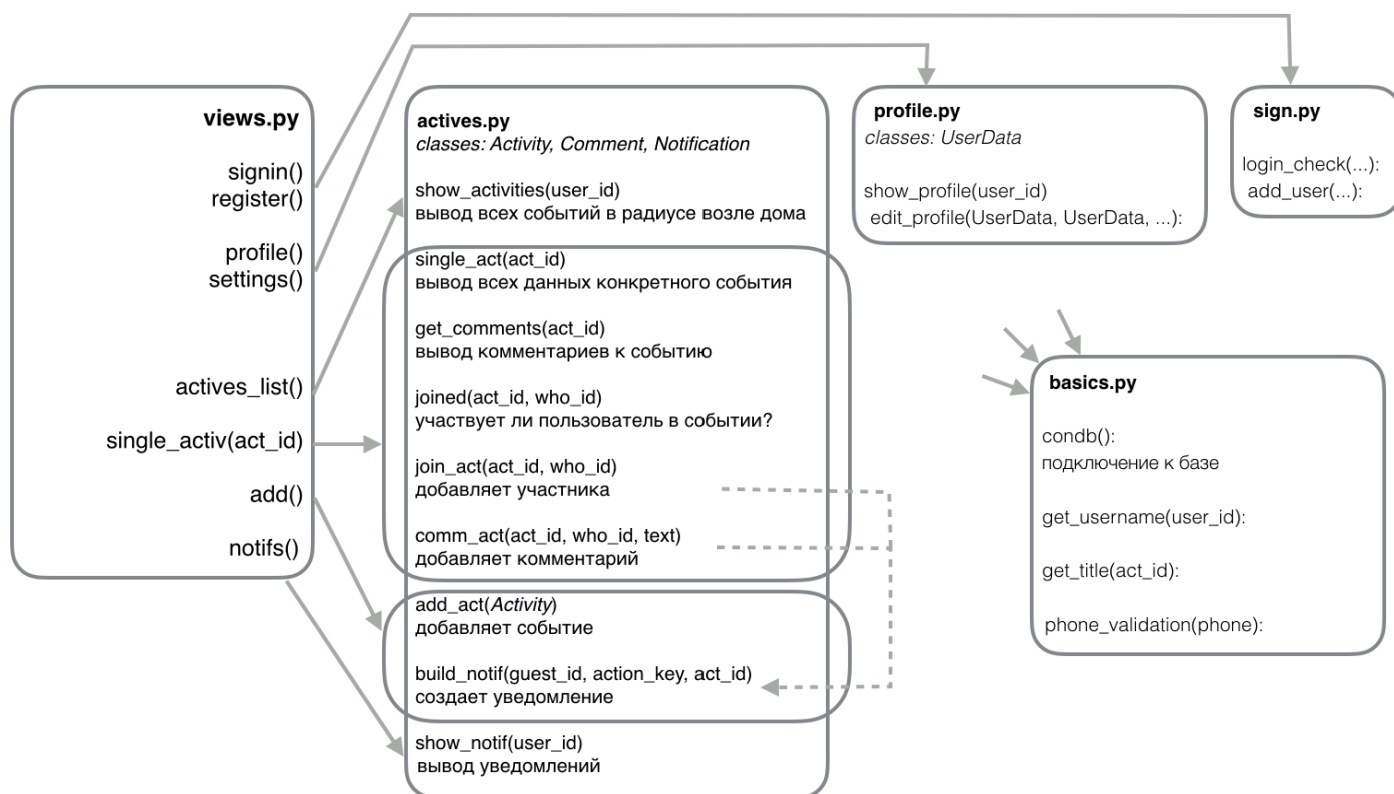
Ниже представлена структура клиента.

Белые блоки - экраны. В каждом блоке указаны название экрана, окончание URL, методы, имя обслуживающей его view-функции и название html-темплейта.





Чтобы облегчить чтение кода и структурировать его, все взаимодействие с базой и некоторые другие функции вынесены в отдельные скрипты. Так как "everything in Python is an object", на схеме ниже показано расположение объявлений и функций, и классов, то есть "оглавление" практически всего кода. Классы для форм - подклассы класса Form, реализованного WTForms, находятся во `views.py` и на схеме не представлены.



Хочется отметить, почему функции получения данных события и комментариев к нему разделены, хотя они никогда не используются по отдельности. Даже произойдет ошибка при получении комментариев из базы, страница откроется, и данные о событии будут отображены, но ниже появится ошибка о том, что подгрузить комментарии не удалось. Эту маленькую деталь не предусмотрели создатели Facebook, что раздражает невероятно: при низкой скорости соединения (просмотре с мобильного устройства) ты не можешь прочесть пост просто потому, что к нему слишком много комментариев.

Код полностью представлен в Приложении 1.

## @Home

[Login](#) [Sign up](#)

### Login

Phone:

+7 

Password:

Login

## @Home

[Login](#) [Sign up](#)

### Sign up

Full Name

Phone

+7 

Password

☐ I accept Terms of Service

SignUp

## @Home

[Activities](#) [Add](#) [Notifications](#) [Profile](#)

---

Games \*\*\* Thu, Dec 15, 19:00

[Dixit Club](#)

---

Sports \*\*\* Sat, Dec 17, 12:00

[Football](#)

---

Charity \*\*\* Sat, Dec 24, 11:00

[School crafts market](#)

[Activities](#) [Add](#) [Notifications](#) [Profile](#)

## John Doe

None

Name: John D. Doe

Phone: +79031234567

Home: 55.809837, 37.49319


[Settings](#)

[Activities](#) [Add](#) [Notifications](#) [Profile](#)

Display name:

Full name:

Phone:

+7 

---

Latitude:

Longitude:

---

UPDATE

LOGOUT

[Activities](#) [Add](#) [Notifications](#) [Profile](#)

---

2016, Dec 03, 05:41

Ann Fisherman commented on your activity:[School crafts market](#)

---

2016, Dec 03, 05:32

Ann Fisherman joined your activity:[School crafts market](#)

---

2016, Dec 03, 05:02

Lea Freeman commented your activity:[School crafts market](#)

---

2016, Dec 03, 05:01

Lea Freeman joined your activity:[School crafts market](#)

# @Home

[Activities](#) [Add](#) [Notifications](#) [Profile](#)

## Add activity

Type:

Sport 

Title:

Description:

Starts:

dd/mm/yyyy --:--

Ends:

dd/mm/yyyy --:--

Latitude:

Longitude:

☐ Accessible for wheelchairs:

POST

# @Home

[Activities](#) [Add](#) [Notifications](#) [Profile](#)

Charity \*\*\* Sat, Dec 24, 11:00 - Sat, Dec 24, 16:00

## School crafts market

By: Jane Doe

None

Our little artists sell their works to help Stop Cancer program. Join us for games, cakes and art of course!

Location: 55.809496, 37.495563

---

2 joined

JOIN

---

3 comments

### Lea Freeman

2016, Dec 03, 04:38

How about a little photo report for the District newspaper?

POST

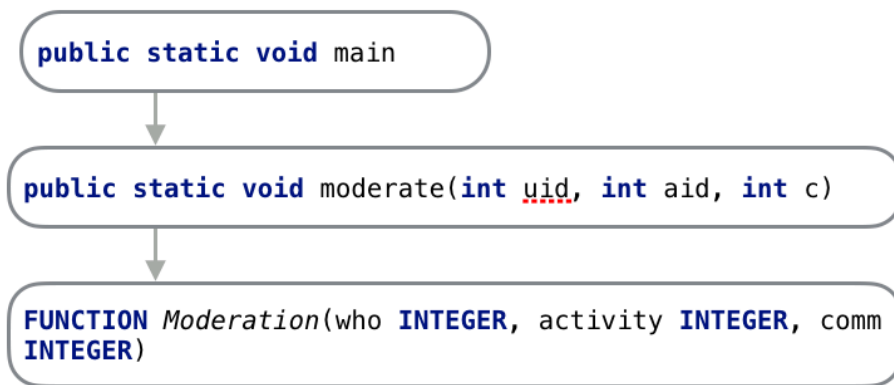
# Консольный клиент

## Хранимая процедура

Консольный клиент реализует вызов процедуры модерации событий и комментариев, нарушающих правила сервиса. Процедура как таковая реализована на SQL, функция в Java лишь вызывает ее.

События, нарушающие правила, удаляются полностью. Комментарии заменяются сообщением. За каждое нарушение пользователю выносится предупреждение.

Для хранения данных о нарушениях создана отдельная таблица. Пользователю выносятся 2 предупреждения, затем он лишается доступа к ресурсу на некоторое время, скажем - неделю. (База хранит только дату бана, время не фиксировано).



Код представлен в Приложении 2.

## Триггеры

Также, согласно заданию, реализовано 2 триггера для проверки дат. Триггеры не применены ни к одному из клиентов, просто хранятся вместе с процедурой.

Первый для таблицы `activities` следит за тем, чтобы дата начала была раньше даты окончания события, и оно проходило в ближайшие полгода (то никто не мог добавить событие в прошлом или, наоборот, через 3 года).

Второй триггер для таблицы `activities_participants` следит за тем, чтобы присоединиться (`join`) к активити в прошлом было невозможно.

Очевидно, проверку первого нужно проводить при валидации, а второе вообще не должно происходить - потому что прошедшие события не видны никому (или, по крайней мере, видны только организаторам).

Код представлен в Приложении 2 вместе с хранимой процедурой.

## Выводы

### "Проще просить прощения, чем разрешения"

При написании самой первой функции - регистрации - я собралась проверять, нет ли указанного пользователем номера телефона в базе, а уже затем добавлять. И что-то спросила по этому поводу у коллеги, на что он дал мне понять, что я занимаюсь ерундой. И велел сразу пытаться сделать INSERT - и сэкономить тем самым один запрос в случае успеха. Добавив, - "вообще питон это "проще попросить прощения, чем разрешения". В дальнейшем я пыталась применять этот принцип, что ведет к следующему личному "открытию".

### Constraints как способ валидации

Идея практически намеренного допущения ошибок сначала показалась мне ужасной. Затем пришло осознание, что если база как таковая содержит ограничения, то проверяя данные заранее, я по сути "дублирую" код. Здесь началось изменение исходной схемы - я заменила проверки ограничением UNIQUE на номер телефона и display name в таблице citizens.

### Geography type и auto increment

Фактически, продолжением этой идеи стали:

- исправление ошибки генерации PK - замена MAX(id)+1 на серию
- использование модуля PostGIS

В Postgres нет автоинкремента, но и создавать серию для генерации id тоже не обязательно. Для этих целей есть "тип" SERIAL, типом его назвать нельзя, потому что храним мы по-прежнему integer, но указывается он вместо типа. Но графический редактор таблиц PyCharm указать его в качестве типа не позволяет. Ну и дальше есть выбор - нажать много клавиш (идти в консоль и модифицировать колонки вручную) или поставить 1 галочку, которая обозначена как автоинкремент, но на деле формирует запрос для создания серии со всеми деталями, которые остается только отредактировать, если есть необходимость. Прогресс и лень, разумеется, побеждают.

О PostGIS. Изначально для "дома" пользователя и события хранились в чистом виде широта и долгота. И я уже нашла функцию, которая переводит их в градусы и вычисляет расстояние с использованием сферических координат - то есть расстояние по поверхности Земли, а не "насквозь". Для реализации требовалось импортировать один или два матмодуля Python, а также заменить тип данных во всех колонках широты и долготы в базе (на POINT). К счастью, реализовать этот кошмар я не успела - модуль PostGIS мне встретился раньше. Он реализует тип данных GEOGRAPHY (два экземпляра POINT и радиус планеты), функции перевода широты-долготы в этот GEOGRAPHY и обратно, а главное - функцию расчета "реалистичного" расстояния.

### Кто "заслуживает" таблицу

На этапе создания схемы возник вопрос - как поступать с данными профиля:

- Добавить их в таблицу с пользователями - и модифицировать ее колонки, если мы решим добавить новые поля (например, профили становятся публичными, и в них появляются биографии и так далее).
- Создать отдельную таблицу с отношением 1 к 1, в которой мы сможем в дальнейшем резвиться без угрозы login data, но придется составлять сложные запросы для всех действий над профилем.

Для курсовой работы, очевидно, без разницы, но мне стало интересно, как это решается в реальной жизни. Экспертные мнения, собранные среди знакомых, довели к первому решению, и по такому принципу в дальнейшем были решены аналогично (кроме случая, описанного в следующем пункте).

Аналогичный вопрос возникает о словарях, и здесь тоже решение было принято из соображений "на вырост". Типы событий (Sports, Dance, etc.) получили свою отдельную таблицу и вошли в таблицу событий как foreign keys - для удобства предполагаемой (а также пока не запланированной) фильтрации. Словарь же текстов уведомлений таблицы не получил - потому что число типов уведомлений будет большим, а также станет уместна генерация фраз из большого числа фрагментов. И по мере усложнения, решения с помощью словарей Python останутся изящными, а запросы к гипотетической таблице будут становиться все уродливее и "дороже" по ресурсам.

## Как на самом деле нужно было реализовывать бан

Так как я выполняла работу не по порядку, к моменту создания консольного клиента графический был уже готов, и последний хотелось его не затрагивать. Поэтому новая таблица bad, которая хранит предупреждения, хранит и дату бана. В "реальности" хранить дату бана логичнее в таблице citizens, и включить проверку при login. Допустим, бан у нас на неделю. (Также опустим вопрос с session.get['logged in']). Тогда при входе мы проверяем не только login data, но и нет ли бана. И если дата не null + неделя не прошла, не пускаем. Если прошла - делаем дату null.

## Something UI vs. тестирование

Что бы ни случилось, пользователь должен понимать, где он находится, как сюда попал - или хотя бы как вернуться назад. Одним словом, если что-то пошло не так, должен возвращаться какой-то экран (хотя бы тот же), с сообщением об ошибке - максимально содержательным, но все еще понятным. Короче говоря - ни в коем случае не internal server error. И если что-то пошло не так с базой - юзеру детали знать бессмысленно. Сообщение на этот случай - одна из немногих "глобальных" переменных в данной работе. (В кавычках - потому что в Python, строго говоря, нет глобальных переменных).

На самом деле об описанном выше есть отдельное правило из 10 "заповедей" юзабилити, сформулированных Якобом Нильсоном, но для меня это уже также естественно - как есть или спать. Поэтому все запросы к базе я с самого начала, не задумываясь, реализовала как try - except. На все ошибки при попытке отправить базе запрос у GUI один ответ - "простите, попробуйте позже". Очевидно, очень скоро я прокляла свое "безотказное" решение - его было совершенно невозможно тестировать. Пришлось пройти по всему коду и добавить содержательные выводы "скрипт вот тот: не взлетело вот это" - просто не на GUI, а себе на консоль.

## Источники

Vertabelo Knowledge base <http://support.vertabelo.com/forums/2-knowledge-base/>

Mark Pilgrim, "Dive into Python 3"

Flask documentation <http://flask.pocoo.org>

PostgreSQL documentation <https://www.postgresql.org/docs/>

PostgreSQL wiki <https://wiki.postgresql.org>

# Приложение 1

## views.py

```
from flask import Flask, request, session, redirect, url_for,
render_template, flash
from wtforms import Form, validators, BooleanField,
FloatField, StringField, SelectField, PasswordField,
SubmitField, TextAreaField
from wtforms_components import DateTimeField, DateRange,
TimeField
from datetime import datetime, timedelta
```

```
from athome import app, sign, actives, uprofile, basics
```

```
app.secret_key = 'OLOLO'
```

```
class RegForm(Form):
    fullname = StringField('Fullname', [validators.Length(min=4,
max=25)])
    code = SelectField('Code', choices=[('7', '+7'), ('38', '+38'),
('375', '+375')])
    phone = StringField('Phone', [validators.Length(min=10,
max=20)])
    password = PasswordField('Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords must
match')
    ])
    confirm = PasswordField('confirm')
    accept_tos = BooleanField('I accept the TOS',
[validators.DataRequired()])
```

```
class ProfileForm(Form):
    nickname = StringField('nickname',
[validators.DataRequired()]) #[validators.Length(min=4,
max=25)]
    #photo =
    fullname = StringField('fullname',
[validators.DataRequired()]) #, [validators.Length(min=4,
max=30)]
    code = SelectField('Code', choices=[('7', '+7'), ('38', '+38'),
('375', '+375')])
    phone = StringField('phone', [validators.DataRequired()]) #,
[validators.Length(min=10, max=20)]
    latitude = FloatField('latitude',
[validators.NumberRange(min=-90, max=90, message='Invalid
latitude'), validators.DataRequired()])
    longitude = FloatField('longitude',
[validators.NumberRange(min=-180, max=180,
message='Invalid longitude'), validators.DataRequired()])
    submit = SubmitField('UPDATE')
```

```
class AddForm(Form):
    type = SelectField('type', choices=[('Sport', 'Sport'),
('Games', 'Games'),
('Language', 'Language'),
('Practises', 'Practises'), ('Language',
('Dance', 'Dance'), ('Kids', 'Kids'),
('Eco', 'Eco'),
('Lost/Found', 'Lost/Found'), ('Event',
'Event'),
('Charity', 'Charity'), ('Music', 'Music'),
('Art', 'Art')])
```

```
#photo =
title = StringField('Title', [validators.DataRequired()])
description = TextAreaField('Description',
[validators.DataRequired()])
starts = DateTimeField('Start date', format='%Y-%m-%d')
start_time = TimeField('Start time', format='%H:%M')
ends = DateTimeField('End date', format='%Y-%m-%d')
end_time = TimeField('End time', format='%H:%M')
latitude = FloatField('Latitude',
[validators.NumberRange(min=-90, max=90, message='Invalid
latitude'),
        validators.DataRequired()])
longitude = FloatField('Longitude',
[validators.NumberRange(min=-180, max=180,
message='Invalid longitude'),
        validators.DataRequired()])
access = BooleanField('Accessable')
submit = SubmitField('POST')
```

```
class CommentForm(Form):
    comment = TextAreaField('comment',
[validators.DataRequired()])
    post = SubmitField('POST')
```

```
#####ACTIVITIES #####
```

```
@app.route('/', methods=['GET'])
def actives_list():
    if not session.get('logged_in') or not session.get('user'):
        return render_template('signin.html', error=None)
    [list, message] =
actives.show_actives(int(session.get('user')))
    return render_template('actives.html', entries=list,
error=message)
```

```
@app.route('/add', methods=['GET', 'POST'])
def add():
    if not session.get('logged_in') or not session.get('user'):
        return render_template('signin.html', error=None)
```

```
    form = AddForm(request.form)
    id = session.get('user')
    if request.method == 'GET':
        return render_template('add.html', form=form,
error=None)
    if request.method == 'POST' and form.validate():
```

```
        #photo =
        [starts] = form.starts.raw_data
        [stime] = form.start_time.raw_data
        [ends] = form.ends.raw_data
        [etime] = form.end_time.raw_data
```

```
        starts = starts + ' ' + stime
        ends = ends + ' ' + etime
```

```
        #validating if trying to add event in the next 6 months
        sd = datetime.strptime(starts, "%Y-%m-%d %H:%M")
        ed = datetime.strptime(ends, "%Y-%m-%d %H:%M")
        print(sd, ed)
        if sd > ed:
            print(sd, ed)
```



```

        return render_template('add.html', form=form,
error='Activity should end later than it starts')
        if sd < datetime.now() or ed > datetime.now() +
timedelta(6*365/12):
            return render_template('add.html', form=form,
error='Activity should start and end within next 6 months')

        new_activity = actives.Activity(None, form.type.data, id,
form.title.data,
                                starts, ends, form.latitude.data,
form.longitude.data,
                                form.description.data, 0, 0, None,
form.access.data)

        [new_id, error] = actives.add_act(new_activity)
        if not new_id:
            return render_template('add.html', form=form,
error=error)

        return render_template('an_act.html', act_id=new_id,
activity=new_activity, joined=True,
                                word='comments', comments=[],
form=CommentForm())
        if not form.validate():
            print(form.starts.data)
            flash_errors(form)
            return render_template('add.html', form=form,
error=None)

@app.route('/<int:act_id>', methods=['GET', 'POST'])
def single_activ(act_id):

    if not session.get('logged_in') or not session.get('user'):
        return render_template('signin.html', error=None)

    [act, error1] = actives.single_act(act_id)
    if act == None:
        abort(404)
    id = session.get('user')
    [im_in, whatever] = actives.joined(act_id, id)
    [comms, error2] = actives.get_comments(act_id)
    if act.c % 10 == 1 or act.c == 1:
        word = 'comment'
    else:
        word = 'comments'
    form = CommentForm(request.form)
    #form2= Form(prefix='form2')

    if request.method == 'GET':
        return render_template('an_act.html', activity=act, joined
= im_in, word=word,
                                comments=comms, error1=error1,
error2=error2, form=form)

    if request.method == 'POST' and form.validate():
        [done, error4] = actives.comm_act(act_id, id,
form.comment.data)
        if done:
            [comms, error2] = actives.get_comments(act_id)
            act.c = act.c + 1
            return render_template('an_act.html', act_id=act_id,
activity=act, joined=im_in, word=word,
                                comments=comms, form=form,
error1=error2, error2=None)
        else:
            return render_template('an_act.html', act_id=act_id,
activity=act, joined=im_in, word=word,
                                comments=comms, form=form,
error1=error4, error2=None)

```

```

        if request.method == 'POST' and request.form['btn'] ==
'JOIN':
            [joined, error3] = actives.join_act(act.id, id)
            if joined:
                act.j = act.j+1
                im_in = True
                return render_template('an_act.html', activity=act,
joined = im_in, word=word,
                                comments=comms, error1=error1,
error2=error2, form=form)
            else:
                return render_template('an_act.html', activity=act,
joined = im_in, word=word,
                                comments=comms, error1=error3,
error2=None, form=form)

##### SIGN IN / UP #####

@app.route('/signin', methods=['GET', 'POST'])
def signin():

    if request.method == 'GET':
        return render_template('signin.html', error=None)

    if request.method == 'POST':
        code = int(request.form['code'])
        phone = request.form['phone']
        password = request.form['password']
        phone = basics.phone_validation(phone)

        if not phone:
            return render_template('signin.html', error='Invalid
phone number')

        [uid, message] = sign.login_check(code, phone,
password) # check if there's 10 digits (we support CIS only
now)
        if not message: # check are the login data correct
            session['logged_in'] = True
            session['user'] = uid
            print("logged in:", session.get('logged_in'), ', as id:',
session.get('user'))
            return redirect(url_for('actives_list'))
        else:
            return render_template('signin.html', error=message)

@app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegForm(request.form)

    if request.method == 'GET':
        return render_template('register.html', form=form,
error=None)

    if request.method == 'POST' and form.validate():
        print(1)
        phone = form.phone.data
        phone = basics.phone_validation(phone)
        if not phone:
            print(2)
            return render_template('register.html', form=form,
error='Invalid phone number')

        code = form.code.data

```

```

code = int(code)

[uid, message] = sign.add_user([form.fullname.data,
phone, code, form.password.data])
if not message:
    print(3)
    session['user'] = uid
    session['logged_in'] = True
    flash("Welcome home! You can edit the data you've
added anytime in Profile")
    return redirect(url_for('actives_list'))
else:
    print(4)
    return render_template('register.html', form=form,
error=message)
if request.method == 'POST' and not form.validate():
    flash_errors(form)
    print(5)
    return render_template('register.html', form=form,
error=None)

```

##### PROFILE #####

```

@app.route('/profile', methods=['GET'])
def profile():
    if not session.get('logged_in') or not session.get('user'):
        return render_template('signin.html', error=None)
    else:
        id = int(session.get('user'))
        print(id)
        [userdata, error] = uprofile.show_profile(id)

        return render_template('profile.html', error=error,
profile=userdata)

@app.route('/settings', methods=['GET','POST'])
def settings():

    if not session.get('logged_in') or not session.get('user'):
        return render_template('signin.html', error=None)

    form1 = ProfileForm(request.form)
    id = session.get('user')
    [userdata, error] = uprofile.show_profile(id)

    if request.method == 'GET':
        return render_template('settings.html', form1=form1,
fn=userdata.f, nn=userdata.n,
                                cr=userdata.c, pn=userdata.p,
la=userdata.la, lo=userdata.lo, error=error)

    if request.method == 'POST' and form1.validate():

        #did phone or display name changed?
        phone = form1.phone.data
        clean_phone = basics.phone_validation(phone)

        if not clean_phone:
            return render_template('settings.html', form1=form1,
fn=form1.fullname.data,
                                nn=form1.nickname.data,
cr=form1.code.data, pn=userdata.p,
                                la=form1.latitude.data,
lo=form1.longitude.data,
                                ph=None, error='Invalid phone number')

        code = form1.code.data

```

```

code = int(code)

pc = userdata.p != clean_phone or userdata.c != code
#did phone number change?
dnc = userdata.n != form1.nickname.data #did display
name change?

#saving changes
newdata = uprofile.UserData(form1.fullname.data,
form1.nickname.data, code, clean_phone,
                                form1.latitude.data,
form1.longitude.data, None)

#trying to update
[done, upd_data, error] = uprofile.edit_profile(userdata,
newdata, id, pc, dnc)
if done and not error:
    return render_template('profile.html', profile=upd_data,
error=None)
else:
    return render_template('settings.html', form1=form1,
fn=upd_data.f, nn=upd_data.f,
                                cr=upd_data.c, pn=upd_data.p,
la=upd_data.la, lo=upd_data.lo,
                                ph=upd_data.ph, error=error)

if not form1.validate() and request.form['btn'] != 'LOGOUT':
    flash_errors(form1)
    return render_template('settings.html', form1=form1,
fn=form1.fullname.data,
                                nn=form1.nickname.data,
cr=form1.code.data, pn=form1.phone.data,
                                la=form1.latitude.data,
lo=form1.longitude.data,
                                ph=None, error=None)

if request.method == 'POST' and request.form['btn'] ==
'LOGOUT':

    session.clear()
    return render_template('signin.html', error=None)

@app.route('/notifs', methods=['GET'])
def notifs():
    if not session.get('logged_in') or not session.get('user'):
        return render_template('signin.html', error=None)
    [notif_list, error] = actives.show_notifs(session.get('user'))
    return render_template('notifs.html', error=error,
notifs=notif_list)

def flash_errors(form):
    for field, errors in form.errors.items():
        for error in errors:
            flash(u"Error in the %s field - %s" % (
                getattr(form, field).label.text,
                error
            ))

if __name__ == '__main__':
    app.run()

```

## actives.py

```
import pycpg2
from athome import basics

problem = "Error. Please, check your internet connection and
try again later"
notif_dict = {'join': 'joined your activity:', 'not': 'will not
participate in your activity:',
             'comm': 'commented on your activity:}'

class Activity:
    def __init__(self, id, type, user_id, title, starts, ends, lat,
long,
                descr, joined, comments, img, access):
        self.id = id
        self.ty = type
        self.uid = user_id
        self.u = basics.get_username(user_id)
        self.t = title
        self.s = starts
        self.e = ends
        self.la = lat
        self.lo = long
        self.d = descr
        self.j = joined
        self.c = comments
        self.ph = img
        self.a = access

class Comment:
    def __init__(self, user_id, posted, text, act_id):
        self.u = basics.get_username(user_id)
        self.uid = user_id
        self.t = text
        self.d = posted
        self.a = act_id

class Notification:
    def __init__(self, text, date, id):
        self.t = text
        self.d = date
        self.id = id
        self.ti = basics.get_title(id)

##### SHOW ACTIVITIES #####

def show_actives(id):
    #returns list of activities for current user (or error)

    dist = 3000 #have no filters so far - distance from home in
meters

    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()

    #checking
    try:
        cur.execute("""SELECT act_id, act_type_name,
citizen_id, act_title, to_char(act_start_time, 'Dy,FX Mon DD,
HH24:MI'),
                        to_char(act_end_time, 'Dy,FX Mon DD,
HH24:MI'), ST_Y(act_loc::geometry) as latitude,
                        ST_X(act_loc::geometry) as longitude,
act_descript, cit_joined, cit_comments, act_img, accessible
FROM activities WHERE
ST_Distance((SELECT home FROM citizens WHERE
citizen_id=%s), act_loc)<=%s
AND act_end_time>=current_timestamp
ORDER BY act_start_time;""", [id, dist])
        actives = cur.fetchall()
        conn.close()
        #if not actives:
        #    return [None, "Nothing happens @Home, try to
increase the distance"]
        act_list=[]
        for (i, act) in enumerate(actives):
            item = Activity(id=act[0], type=act[1], user_id=act[2],
title=act[3], starts=act[4],
                        ends=act[5], lat=act[6], long=act[7],
descr=act[8], joined=act[9],
                        comments=act[10], img=act[11],
access=act[12])
            act_list.append(item)
        return [act_list, None]
    except:
        conn.rollback()
        conn.close()
        print ("Activities: Can't get user home location or activities
list from DB")
        return [None, problem]

def single_act(act_id):
    #returns Activity for a particular activity (or error)

    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()

    try:
        cur.execute("""SELECT act_id, act_type_name,
citizen_id, act_title, to_char(act_start_time, 'Dy,FX Mon DD,
HH24:MI'),
                        to_char(act_end_time, 'Dy,FX Mon DD,
HH24:MI'), ST_Y(act_loc::geometry) as latitude,
                        ST_X(act_loc::geometry) as longitude,
act_descript, cit_joined, cit_comments, act_img, accessible
FROM activities WHERE act_id=%s
ORDER BY act_start_time;""", [act_id])
        [act] = cur.fetchall()
        conn.close()
        details = Activity(id=act[0], type=act[1], user_id=act[2],
title=act[3], starts=act[4],
                        ends=act[5], lat=act[6], long=act[7],
descr=act[8], joined=act[9],
                        comments=act[10], img=act[11],
access=act[12])
        return [details, None]
    except:
        conn.rollback()
        conn.close()
```

```

print("Actives: can't activity details from DB")
return [None, problem]

def get_comments(act_id):
    # returns list of comments for a particular activity (or error)

    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()

    try:
        cur.execute("""SELECT citizen_id, act_comment_text,
to_char(posted, 'YYYY,FX Mon DD, HH24:MI'), act_id
FROM activities_comments WHERE act_id=%s
ORDER BY posted ASC;""", [act_id])
        comments = cur.fetchall()
        if not comments:
            return [None, 'No comments yet']
        conn.close()
        comments_list = []
        for (i, com) in enumerate(comments):
            item = Comment(user_id=com[0], text=com[1],
posted=com[2], act_id=com[3])
            comments_list.append(item)
        return [comments_list, None]
    except:
        conn.rollback()
        conn.close()
        print("Actives: can't get comments from DB")
        return [None, 'Error loading comments']

```

#### ##### NOTIFICATIONS #####

```

def show_notifs(id):
    #returns list of notifications for current user (or error)

    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()

    #checking
    try:
        cur.execute("""SELECT notif_text, to_char(notif_date,
'YYYY,FX Mon DD, HH24:MI'), act_id
FROM notifs WHERE citizen_id=%s
ORDER BY notif_date DESC;""", [id])
        notifs = cur.fetchall()
        conn.close()
        #if cur.rowcount < 1:
        #    return [None, "\nYour notifications will be displayed
here"]
        notif_list=[]
        for (i, notif) in enumerate(notifs):
            item = Notification(notif[0], notif[1], notif[2])
            notif_list.append(item)
        return [notif_list, None]
    except:
        conn.rollback()
        conn.close()
        print ("Notifications: Can't get notifications from DB")
        return [None, problem]

```

```

def build_notif(guest_id, action_key, act_id):

```

```

#creates a row in notifications table and returns true/false
# inner, so no error messages in return

```

```

guest = basics.get_username(guest_id)
[activ, whatever] = single_act(act_id)
host_id = activ.uid
act_id = activ.id
text = guest + notif_dict.get(action_key)

```

```

conn = basics.condb()
if not conn or not guest: #or not host_id:
    print("Actives: can't create notification (can't connect to
DB)")
    return [False]

```

```

cur = conn.cursor()

```

```

try:
    cur.execute("""INSERT INTO notifs (notif_id, act_id,
citizen_id, notif_text, notif_date)
VALUES (DEFAULT, %s, %s, %s,
current_timestamp);""", [act_id, host_id, text])
    #comments = cur.fetchall()
    conn.commit()
    conn.close()
    return [True]

```

```

except:
    conn.rollback()
    conn.close()
    print("Notifs: can't create notification")
    return [False]

```

#### #####ADD ACTIVITY #####

```

def add_act(a):
    #gets Activity and creates a row in activities, returns new id
(or error)

```

```

    conn = basics.condb()
    if not conn:
        return [None, problem]

```

```

    cur = conn.cursor()
    print([a.ty, a.uid, a.t, a.s, a.e, a.d,
a.ph, a.j, a.c, a.la, a.lo, a.a]
)

```

```

    try:
        cur.execute("""INSERT INTO activities
VALUES (DEFAULT, %s, %s, %s,
to_timestamp(%s, 'YYYY-MM-DD HP24:MI'),
to_timestamp(%s, 'YYYY-MM-DD HP24:MI'),
%s, %s, %s, %s, (ST_MakePoint(%s,%s)),
%s) RETURNING act_id;""", [a.ty, a.uid, a.t,
a.s, a.e, a.d,
a.ph, a.j, a.c, a.la, a.lo, a.a])

```

```

    [(a.id,)] = cur.fetchall()
    conn.commit()
    conn.close()
    return [a.id, None]

```

```

except:
    conn.rollback()
    conn.close()
    print("Actives: can't add activity to DB")
    return [False, problem]

```

#### ##### JOINS AND COMMENTS #####

```

def joined (act_id, who_id):
    # lets us now do a user participates in activity
    #returns 'yes'/'no' (or error)

```

```

conn = basics.condb()
if not conn:
    return [None, problem]

cur = conn.cursor()

cur.execute("""SELECT citizen_id FROM
activities_participants
WHERE act_id=%s AND citizen_id=%s;""",
[act_id, who_id])
joined = cur.fetchall()
if joined:
    return [True, None]
else:
    return [False, None]

def join_act(act_id, who_id):
    # gets activity id and guest id, returns True/False (or error)

    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()

    try:
        print(act_id, who_id)
        # row to participants
        cur.execute("""INSERT INTO activities_participants
VALUES (DEFAULT, %s, %s);""", [who_id,
act_id])

        # upd column "joined" in activities
        cur.execute("""UPDATE activities SET (cit_joined) =
(cit_joined+1) WHERE act_id=%s;""", [act_id])
        conn.commit()
        conn.close()

        # create notification
        notified = build_notif(who_id, 'join', act_id)
        if not notified:
            print("Activities: joined, but notification wasn't created")
            return [True, None]
        except:
            conn.rollback()
            conn.close()
            print("Actives: can't insert participant to DB or update
count")
            return [False, problem]

def comm_act(act_id, who_id, text):
    # gets activity id, guest id and text, returns True/False (or
error)

    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()
    try:
        print(act_id, who_id, text)
        # row to comments
        cur.execute("""INSERT INTO activities_comments
VALUES (DEFAULT, %s, %s, %s,
current_timestamp)""", [act_id, who_id, text])

        # upd comments count in activities

```

```

cur.execute("""UPDATE activities SET (cit_comments) =
(cit_comments+1) WHERE act_id=%s;""", [act_id])
        conn.commit()
        conn.close()

        # create notification
        try:
            build_notif(who_id, 'comm', act_id)
        except:
            print("Activities: commented, but notification wasn't
created")
            return [True, None]
        except:
            conn.rollback()
            conn.close()
            print("Actives: can't insert comment to DB or update
count")
            return [False, problem]

```

## sign.py

```

import pycpg2
from athome import basics

problem = "Error. Please, check your internet connection and
try again later"

def login_check(code, phone, password):
    # returns current user id (or error)

    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()

    # checking
    try:
        cur.execute("""SELECT citizen_id FROM citizens
WHERE phone=%s AND code_region=%s AND
password=%s;""", [phone, code, password])
        result = cur.fetchall()
        # if failed, checking if user exist
        if not result:
            try:
                cur.execute("""SELECT citizen_id FROM citizens
WHERE phone=%s AND
code_region=%s;""", [phone, code])
                result2 = cur.fetchall()
                if not result2:
                    conn.rollback()
                    conn.close()
                    return [None, "Phone number doesn't exist.
Please, sign up"]
            else:
                conn.rollback()
                conn.close()
                return [None, "Incorrect password"]
        except:
            conn.rollback()
            conn.close()
            print("Sign: can't check if the user exist")
            return [None, problem]
    [(uid,)] = result
    conn.close()
    return [uid, None]
except:
    conn.rollback()

```

```

conn.close()
print("Sign: can't check login data")
return [None, problem]

def add_user(user_row):
    # gets a list [name, phone, code, password] and returns
    new id (or error)
    #using current location for now
    #so that's the only case we need to import stuff to get
    current location
    from urllib.request import urlopen
    import json

    # Automatically geolocate the connecting IP
    f = urlopen('http://freegeoip.net/json/')
    json_string = str(f.read())
    f.close()
    json_string = json_string.replace("b", "")
    json_string = json_string.replace("\n", "")
    location = json.loads(json_string)
    lat = location['latitude']
    long = location['longitude']
    print(lat, long)

    #adding current location them to list of user data
    user_row.insert(3, long)
    user_row.insert(4, lat)
    # copying name to display name (hoping it's free)
    user_row.insert(1, user_row[0])

    conn = basics.condb()
    if not conn:
        return [None, problem]
    cur = conn.cursor()

    N = 1 # increment for display name builder below

    # trying to INSERT and return id
    while True:

        try:
            cur.execute("""INSERT INTO citizens(citizen_id,
citizen_name, display_name, phone,
                        code_region, home, password)
                        VALUES (DEFAULT,%s,%s,%s,%s,
(ST_MakePoint(%s,%s)),%s)
                        RETURNING citizen_id;""", user_row)

            [(uid,)] = cur.fetchall()
            conn.commit()
            return [uid, None]

        except psycopg2.IntegrityError as e:
            conn.rollback()
            conn.close()
            if e.diag.message_detail.startswith('Key (phone)'):
                return [None, "Phone number already exists. Please,
sign in"]

            # if display name isn't unique, creating it
            elif e.diag.message_detail.startswith('Key
(display_name)'):
                next_nickname = user_row[0] + str(' ') + str(N)
                user_row[1] = next_nickname
                N += 1
            else:
                conn.rollback()

```

```

conn.close()
print ("Sign: unknown integrity error - trying to
insert")
return [None, problem]
except:
    conn.rollback()
    conn.close()
    print("Sign: can't add new user to DB :(")
    return [None, problem]

```

## uprofile.py

```

import psycopg2
from athome import basics

```

problem = "Error. Please, check your internet connection and try again later"

```

class UserData:
    def __init__(self, fullname, nickname, code, phone, lat, long,
photo):
        self.f = fullname
        self.n = nickname
        self.c = code
        self.p = phone
        self.la = lat
        self.lo = long
        self.ph = photo

```

```

def show_profile(id):
    # returns user data (or error)
    conn = basics.condb()
    if not conn:
        return [None, problem]

    cur = conn.cursor()
    try:
        cur.execute("""SELECT citizen_name as fullname,
display_name as nickname,
                        code_region as code, phone,
ST_Y(home::geometry) as latitude,
                        ST_X(home::geometry) As longitude, photo as
photo
                        FROM citizens WHERE citizen_id=%s;""", [id])
        res=cur.fetchall()
        conn.close()
        [tup]=res
        userdata = UserData(tup[0], tup[1], tup[2], tup[3], tup[4],
tup[5], tup[6])
        return [userdata, None]
    except:
        conn.rollback()
        conn.close()
        print("Uprofile: can't show user data")
        return [None, problem]

```

```

def edit_profile(userdata, newdata, id, pc, dnc): #pc = phone
changed, dnc = display name changed
# returns true/false (and/or 1/2 errors)

```

```

conn = basics.condb()
if not conn:
    return [False, False, problem]

cur = conn.cursor()

```

```

progress = ""

if pc:
    try: #check for another user with the number we wanna
add
        cur.execute("""SELECT citizen_id FROM citizens
            WHERE phone=%s AND code_region=%s
AND citizen_id !=%s;""", [newdata.p, newdata.c, id])
        res=cur.fetchall()

        if res: #keeping old phone
            newdata.p = userdata.p
            newdata.c = userdata.c
            progress = progress + "Phone number belongs to
another user.\nIf it's your's, please, login as that user.\n"
        except:
            conn.rollback()
            conn.close()
            print ("Uprofile: can't check if phone is occupied")
            return [False, False, problem]

    if dnc:
        try: #check if display name is free
            cur.execute("""SELECT citizen_id FROM citizens
                WHERE display_name=%s AND citizen_id!
=%s;""", [newdata.n, id])
            res = cur.fetchall()

            if res:
                newdata.n = userdata.n #keeping old display name
                progress = progress + "Display name is taken,
please, choose another one"

            except:
                conn.rollback()
                conn.close()
                print("Uprofile: can't check if display name is
occupied")
                return [False, False, problem]

        #finally, updating user data
        try:
            cur.execute("""UPDATE citizens SET (citizen_name,
display_name, phone, code_region,
                home, photo)=(%s,%s,%s,%s,
(ST_MakePoint(%s,%s)),%s)
                WHERE citizen_id=%s;""", [newdata.f,
newdata.n, newdata.p, newdata.c,
newdata.lo, newdata.la,
newdata.ph, id])

            conn.commit()
            conn.close()
            return [True, newdata, progress]
        except:
            conn.rollback()
            conn.close()
            print("Uprofile: can't update user's record")
            return [False, False, problem]

```

## basics.py

```

import pycpg2

def condb():
    #connects to DB returning cursor and message

```

```

try:
    conn = pycpg2.connect("dbname='athome'
user='Delilah' host='localhost' password='desophie'")
    return conn
except:
    print ("Basics: can't connect to DB")
    return [None]

def get_username(id):
    #getting user's display name

    conn=condb()
    if not conn:
        return None
    cur = conn.cursor()

    try:
        cur.execute("""SELECT display_name FROM citizens
WHERE citizen_id=%s;""", [id])
        [(dname,)] = cur.fetchall()
        conn.close()
        return dname
    except:
        conn.rollback()
        conn.close()
        print("Basics: can't get display name from DB")
        return None

def get_title(act_id):
    #getting user's display name

    conn=condb()
    if not conn:
        return None
    cur = conn.cursor()

    try:
        cur.execute("""SELECT act_title FROM activities WHERE
act_id=%s;""", [act_id])
        [(title,)] = cur.fetchall()
        conn.close()
        return title
    except:
        conn.rollback()
        conn.close()
        print("Basics: can't get activity title from DB")
        return None

def phone_validation(phone):
    #returns clean 10-digit number (or error)

    phone = phone.replace(" ", "")
    phone = phone.replace(".", "")
    phone = phone.replace("-", "")
    phone = phone.replace("+", "")
    phone = phone.replace(")", "")
    phone = phone.replace("(", "")
    try:
        x = int(phone) # check if it's numbers
        if 999999999 < x < 10000000000:
            return phone
    except:
        return None

```

## layout.html

```
<!doctype html>
<title>@Home</title>
<link rel=stylesheet type=text/css href="{{ url_for('static',
filename='style.css') }}">
<div class=page>
  <dl class="metanav">
    <dd><h1>@Home</h1>
    {% if not session.logged_in %}
      <dd><a href="{{ url_for('signin') }}">Login</a>
      <a href="{{ url_for('register') }}">Sign up</a>
    {% else %}
      <dd><a href="{{ url_for('actives_list') }}">Activities</a>
      <a href="{{ url_for('add') }}">Add</a>
      <a href="{{ url_for('notifs') }}">Notifications</a>
      <a href="{{ url_for('profile') }}">Profile</a>
    {% endif %}
  </dl>
  {% for message in get_flashed_messages() %}
    <div class=flash>{{ message }}</div>
  {% endfor %}
{% block body %}{% endblock %}
</div>
```

## signin.html

```
{% extends "layout.html" %}
{% block body %}
  <dl><dd><h2>Login</h2></dl>
  {% if error %}<p class=error><strong>Error:</strong>
  {{ error }}{% endif %}
  <form action="{{ url_for('signin') }}" method=post>

    <dd>Phone:
    <dd><select name="code">
      <option value=7>+7</option>
      <option value=38>+38</option>
      <option value=375>+375</option>
    </select>
    <input type=text name=phone>
    <dd><br>
    <dd>Password:
    <dd><input type=password name=password>
    <dd><br>
    <dd><input type=submit value=Login>
  </dl>
</form>
{% endblock %}
```

## register.html

```
{% extends "layout.html" %}
{# {% from "_formhelpers.html" import render_field %}#}
{% block body %}
  <dl><dd><h2>Sign up</h2>
  {% if error %}<p class=error><strong>Error:</strong>
  {{ error }}{% endif %}
  </dl>
  <form action="{{ url_for('register') }}" method=post>
    <dl>
      <dd>Full Name
      <dd>{{ form.fullname(placeholder='John Doe') }}
      <dd><br>
      <dd>Phone
      <dd>{{ form.code }}
      {{ form.phone(placeholder='111 111 1111') }}
      <dd><br>
```

```
    <dd>Password
    <dd>{{ form.password(placeholder='Password') }}
    {{ form.confirm(placeholder='Repeat password') }}
    <dd><br>
    <dd>{{ form.accept_tos(value=None)}} I accept Terms of
Service
    <dd><br>
    <dd><input type=submit value=SignUp>
  </dl>
</form>

{% endblock %}
```

## actives.html

```
{% extends "layout.html" %}
{% block body %}
  {% if error %}<p class=error><strong>Error:</strong>
  {{ error }}{% endif %}
  {% if session.logged_in %}
    <dl class=show_actives>
      {% if entries %}
        {% for entry in entries %}
          <dd>
            <dd>{{ entry.ty }} *** {{ entry.s }}
            <dd><a href="{{ url_for('single_activ',
act_id=entry.id) }}">{{ entry.tlsafe }}</a>
          {% endfor %}
        {% else %}
          <dd><em> Nothing happens @Home, try to increase the
distance</em>
        {% endif %}
      </dl>
    {% endif %}
  {% endblock %}
```

## an\_act.html

```
{% extends "layout.html" %}
{% block body %}
  {% if error1 %}<p class=error><strong>Error:</strong>
  {{ error1 }}{% endif %}
  {% if error2 %}<p class=error><strong>Error:</strong>
  {{ error2 }}{% endif %}

  <dl class=single_activ>
    <dd>{{ activity.ty }} *** {{ activity.s }} - {{ activity.e }}
    <dd><h2>{{ activity.tlsafe }}</h2>
    <dd>By: {{ activity.ulsafe }}
    <dd>{{ activity.ph }}
    <dd>{{ activity.dlsafe }}
    <dd>Location: {{ activity.lalsafe }}, {{ activity.lolsafe }}
    <dd>
      <dd>{{ activity.jlsafe }} joined
      {% if joined %} (You're going) {% else %}
      <form action="{{ url_for('single_activ', act_id=activity.id) }}"
      prefix="form2" method=post>
        <input type=submit name=btn value=JOIN>
      </form>{% endif %}
      <dd>
        <dd>{{ activity.clsafe }} {{ wordlsafe }}
        <dd><br>
        {% if comments %}{% for c in comments %}
          <dd><h4>{{ c.ulsafe }}</h4>
          <dd>{{ c.dlsafe }}
          <dd>{{ c.tlsafe }}
```



```

<dd>

{% endfor %}{% endif %}
<form action="{{ url_for('single_activ', act_id=activity.id) }}"
method=post>
    <dd><br>
    <dd>{{ form.comment }}
    <dd>{{ form.post }}
</form>
</dl>

{% endblock %}

```

## add.html

```

{% extends "layout.html" %}
{% block body %}
    {% if error %}<p class=error><strong>Error:</strong>
    {{ error }}{% endif %}
    <form action="{{ url_for('add') }}" method=post class="add">
    <dl>
    <dd><h2>Add activity</h2>
    <dd>Type:
    <dd>{{ form.type }}
    <dd>Title:
    <dd>{{ form.title(rows=3, cols=40) }}
    <dd>Description:
    <dd>{{ form.description(rows=5, cols=40) }}
    <dd>Starts:
    <dd>{{ form.starts (class='datepicker', type='date')}}
    {{ form.start_time }}
    <dd>Ends:
    <dd>{{ form.ends (class='datepicker', type='date')}}
    {{ form.end_time }}
    <dd> Latitude:
    <dd>{{ form.latitude }}
    <dd> Longitude:
    <dd>{{ form.longitude }}
    <dd>
    <dd>{{ form.access }} Accessible for wheelchairs:
    <dd> <br>
    <dd>{{ form.submit }}
    </dl>
    </form>
{% endblock %}

```

## profile.html

```

{% extends "layout.html" %}
{% block body %}
    {% if error %}<p class=error><strong>Error:</strong>
    {{ error }}{% endif %}
    <dl class=profile>
    <dd><h2>{{ profile.nlsafe}}</h2>
    <dd>{{ profile.ph}}
    <dd>Name: {{profile.flsafe}}
    <dd>Phone: +{{profile.c}}{{profile.plsafe}}
    <dd>Home: {{ profile.lalsafe}}, {{ profile.lolsafe}}
    <dd><a href="{{ url_for('settings') }}">Settings</a>
    </dl>

{% endblock %}

```

## settings.html

```

{% extends "layout.html" %}
{% block body %}

```

```

    {% if error %}<p class=error><strong>Error:</strong>
    {{ error }}{% endif %}
    <form action="{{ url_for('settings') }}" prefix="form1"
    name="form1" method=post>
    <dl>
    <dd> Display name:
    <dd>{{ form1.nickname(value=nn) }}
    <dd> Full name:
    <dd>{{ form1.fullname(value=fn) }}
    <dd> Phone:
    <dd>{{ form1.code(value=cr) }}
    {{ form1.phone(value=pn) }}
    <dd> _____
    <dd> Latitude:
    <dd>{{ form1.latitude(value=la) }}
    <dd> Longitude:
    <dd>{{ form1.longitude(value=lo) }}
    <dd> _____
    <dd>{{ form1.submit }}
    <dt>
    </dl>
    </form>
    <form action="{{ url_for('settings') }}" prefix="form2"
    method=post>
    <dl><dd><input type=submit name="btn"
    value=LOGOUT></dl>
    </form>
{% endblock %}

```

## notifs.html

```

{% extends "layout.html" %}
{% block body %}
    {% if error %}<p class=error><strong>Error:</strong>
    {{ error }}{% endif %}
    {% if session.logged_in %}
    <dl class=show_notifs>
    {% if notifs %}
    {% for n in notifs %}
    <dd> _____
    <dd>{{n.dlsafe}}
    <dd>{{n.tlsafe}}<a href="{{ url_for('single_activ',
    act_id=n.id) }}">{{n.tlsafe}}</a>
    {% endfor %}
    {% else %}
    <dd><em>Your notifications will be displayed here</em>
    {% endif %}
    </dl>
    {% endif %}
{% endblock %}

```

# Приложение 2

## Вызов хранимой процедуры

```
package moderation;
import java.sql.*;

public class SProcedure {

    private static Connection conn = null;

    public static void main(String[] args) throws Exception {
        // Loading the driver
        Class.forName("org.postgresql.Driver");
        // ...connecting to the DB
        conn = DriverManager.getConnection(
            "jdbc:postgresql://localhost/athome",
            "Delilah", "desophie");
        // ...calling moderation function
        moderate(2, 4, 5);
        // ...closing connection.
        conn.close();
    }

    /**
     *
     * @param uid   user id
     * @param aid   activity id
     * @param cid   comment id (=0 if it's activity deletion)
     * @throws SQLException DB error case
     */
    public static void moderate(int uid, int aid, int cid)
        throws SQLException {
        // Statement creation
        CallableStatement stmt
            = conn.prepareCall("{ ? = call Moderation(?,?,?) } ");
        // ...setting parameters
        stmt.registerOutParameter(1, Types.VARCHAR);
        stmt.setInt(2, uid);
        stmt.setInt(3, aid);
        stmt.setInt(4, cid);
        // ... statement execution
        stmt.execute();
        String progress = stmt.getString(1);
        System.out.println(progress);
    }
}
```

## Хранимая процедура

```
CREATE OR REPLACE FUNCTION Moderation(who INTEGER,
activity INTEGER, comm INTEGER)
    RETURNS TEXT AS $$
DECLARE
    warn BOOLEAN := FALSE;
    warned BOOLEAN := FALSE;
    finalcase TEXT := 'moderation failed';
BEGIN

    -- activity deletion case
    IF comm = 0 THEN
        IF NOT EXISTS(SELECT * FROM activities
            WHERE act_id=activity AND citizen_id=who)
        THEN finalcase:= 'User already deleted the activity';
        ELSE DELETE FROM activities WHERE act_id=activity;
            warn := TRUE;
        END IF;
    END IF;
```

```
-- comment burning case
IF comm != 0
THEN
    IF NOT EXISTS(SELECT * FROM activities_comments
        WHERE act_comment_id = comm
            AND act_id=activity
            AND citizen_id=who)
    THEN finalcase:= 'This comment does not exist'; -- +
    ELSE UPDATE activities_comments
        SET act_comment_text = 'This comment was removed by
moderator because it violates the Terms of Service'
        WHERE act_comment_id = comm AND act_id=activity AND
        citizen_id=who;
        warn := TRUE;
    END IF;
END IF;

-- if warning needed
IF warn IS TRUE
THEN
    -- 1st warning
    IF NOT EXISTS(SELECT * FROM bad WHERE citizen_id =
        who)
    THEN INSERT INTO bad VALUES (citizen_id = who,
        two_warn=FALSE);
        finalcase:= 'First warning added';
    -- 2d warning
    ELSEIF NOT (SELECT two_warn FROM bad WHERE
        citizen_id=who)
    THEN UPDATE bad SET two_warn = TRUE WHERE
        citizen_id=who;
        finalcase:= 'Second warning added';
    -- Ban
    ELSE UPDATE bad b SET b.ban_date = current_timestamp;
        finalcase:= 'User banned';
    END IF;
END IF;

RETURN finalcase;

END;
$$ LANGUAGE plpgsql;
```

## Триггер времени события

```
CREATE FUNCTION Timeline() RETURNS trigger AS
$timeline$
-- Trigger checks that: now < start time < end time < now + 6
month
DECLARE
    error_message TEXT;
BEGIN
    -- start < end
    IF activities.act_start_time > activities.act_end_time
    THEN
        error_message := 'End time should be later than start
time';
        RAISE EXCEPTION 'End time should be later than start
time';
    END IF;

    -- now < start
```

```

    IF current_timestamp > activities.act_start_time OR
activities.act_end_time > (current_timestamp + '6
month'::interval)
    THEN
        error_message := 'Activity should start and end within
next two months';
        RAISE EXCEPTION 'Activity should start and end within
next two months';
    END IF;

    RETURN error_message;
END;
$timeline$ LANGUAGE plpgsql;

CREATE TRIGGER Timeline BEFORE INSERT OR UPDATE ON
activities
FOR EACH ROW EXECUTE PROCEDURE Timeline();

```

### Триггер участия

```

CREATE FUNCTION Timeline_join() RETURNS trigger AS
$timelinej$
-- Trigger doesn't allow to join / leave a past activity
DECLARE
    error_message TEXT;
BEGIN
    -- start < end
    IF (SELECT a.act_end_time FROM activities a
        WHERE activities_participants.act_id = a.act_id) >
current_timestamp
    THEN
        error_message := 'You are trying to change the past, man';
        RAISE EXCEPTION 'Cannot join/leave activity because it
has over';
    END IF;

    RETURN error_message;
END;
$timelinej$ LANGUAGE plpgsql;

CREATE TRIGGER Timeline_join BEFORE INSERT OR
UPDATE ON activities_participants
FOR EACH ROW EXECUTE PROCEDURE Timeline_join();

```