 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Розрахунково графічна робота**

з дисципліни «Веб-технології та веб-дизайн-2»

**Виконав:**

студент гр. БС-81

Алфьоров А. І.

**Перевірив:**

ас. каф. БМК Матвійчук О. В.

ас. каф. БМК Давидько О. Б.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2020

**Зміст**

[Вибір та опис специфікації проекту. 4](#_Toc60035564)

[Функціональні вимоги 4](#_Toc60035565)

[Набір функціоналу 4](#_Toc60035566)

[Нефункціональні вимоги 4](#_Toc60035567)

[Запланована кількість користувачів 4](#_Toc60035568)

[Можливі downtime 4](#_Toc60035569)

[Інтеграції з іншими сервісами 4](#_Toc60035570)

[Доступність для людей з обмеженими можливостями 4](#_Toc60035571)

[Безпека 5](#_Toc60035572)

[Вимоги до тестування 5](#_Toc60035573)

[Вибір та обґрунтування технологій для реалізації веб сервісу. 5](#_Toc60035574)

[Вибір мови програмування для реалізації веб сервісу. 5](#_Toc60035575)

[Вибір фреймворка для обробки мережевих запитів. 6](#_Toc60035576)

[Опис механізмів виконання паралельних та асинхронних завдань в обраній мові. 6](#_Toc60035577)

[Опис архітектурних рішень що використовувались в ході розробки проекту. 6](#_Toc60035578)

[Опис високорівневої архітектури 6](#_Toc60035579)

[вибір архітектури 6](#_Toc60035580)

[діаграма компонентів та їх в взаємодії 7](#_Toc60035581)

[Опис низькорівневої архітектури для хоча б двох модулів у моноліті. 7](#_Toc60035582)

[Розгортання програмного продукту. 8](#_Toc60035583)

[Скрипт контейнеризації сервісу (dockerfile) 8](#_Toc60035584)

[Скрипт оркестрації сервісів (docker-compose.yml) 8](#_Toc60035585)

[Висновки. 9](#_Toc60035586)

[Додаток. 9](#_Toc60035587)

[Моделі 9](#_Toc60035588)

[models.py 9](#_Toc60035589)

[робота з Кешем 10](#_Toc60035590)

[фрагмент settings.py 10](#_Toc60035591)

[робота з БД 10](#_Toc60035592)

[models.py 10](#_Toc60035593)

[фрагмент settings.py 10](#_Toc60035594)

[обробники веб запитів 11](#_Toc60035595)

[urls.py (глобальні) 11](#_Toc60035596)

[urls.py (tours) 11](#_Toc60035597)

[views.py (tours) 11](#_Toc60035598)

# Вибір та опис специфікації проекту.

## Функціональні вимоги

### Набір функціоналу

Перегляд наявних пропозицій користувачем

Перегляд

Додаваня і видалення походу гідом

## Нефункціональні вимоги

### Запланована кількість користувачів

до 10 000 або 100 000 користувачів з даними в базі даних

та до 10 000 активних переглядачів сайту

### Можливі downtime

відсутні

### Інтеграції з іншими сервісами

redis

### Доступність для людей з обмеженими можливостями

відсутня

### Безпека

лише аутентифікований користувач може отримати доступ до своїх даних

### Вимоги до тестування

перевірка безпеки, та функціональних вимог

# Вибір та обґрунтування технологій для реалізації веб сервісу.

## Вибір мови програмування для реалізації веб сервісу.

Python

філософія мови (The Zen of Python, by Tim Peters):

* Гарне краще за потворне.
* Явне краще за неявне.
* Просте краще за складне.
* Складне краще за заплутане.
* Плоске краще за вкладене.
* Розріджене краще за щільне.
* Легкість читання має значення.
* Особливі випадки не є настільки особливими, щоб порушувати правила.
* Хоча практичність є важливішою за бездоганність.
* Помилки ніколи не повинні проходити непомітно.
* Якщо їх приховування не прописано явно.
* Зустрівши неоднозначність, опирайтесь спокусі вгадати.
* Має бути один — і, бажано, тільки один — очевидний спосіб зробити це.
* Хоча спочатку він може бути й не очевидним, якщо ви не голландець.
* Зараз — краще, ніж ніколи.
* Хоча ніколи, найчастіше, — краще, ніж просто зараз.
* Якщо реалізацію важко пояснити — задум поганий.
* Якщо реалізацію легко пояснити — можливо, задум добрий.
* Простори імен — чудова річ, тож робімо їх більше!

мені подобається ця філософія і сама мова python тому я обрав саме її

## Вибір фреймворка для обробки мережевих запитів.

Django бо він зручний для використання і з ним можна швидко написати сайт

## Опис механізмів виконання паралельних та асинхронних завдань в обраній мові.

Асинхронність може бути конкурентною (тобто хоч програма і виконується асинхронно але дві задачі не можуть одночасно виконуватись) або паралельною (коли дві і більше задач можуть виконуватись одночасно). При конкурентному запуску завдань за ними «слідкує» GIL (Global Interpreter Lock) і не дає новому завданню початись поки не виконалось старе. Для виконання завдань паралельно використовуються різні ядра CPU.

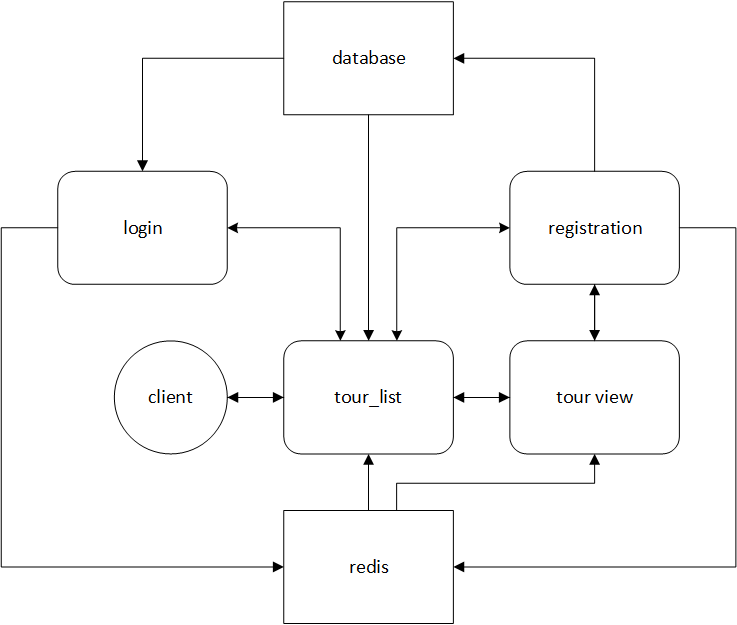
# Опис архітектурних рішень що використовувались в ході розробки проекту.

## Опис високорівневої архітектури

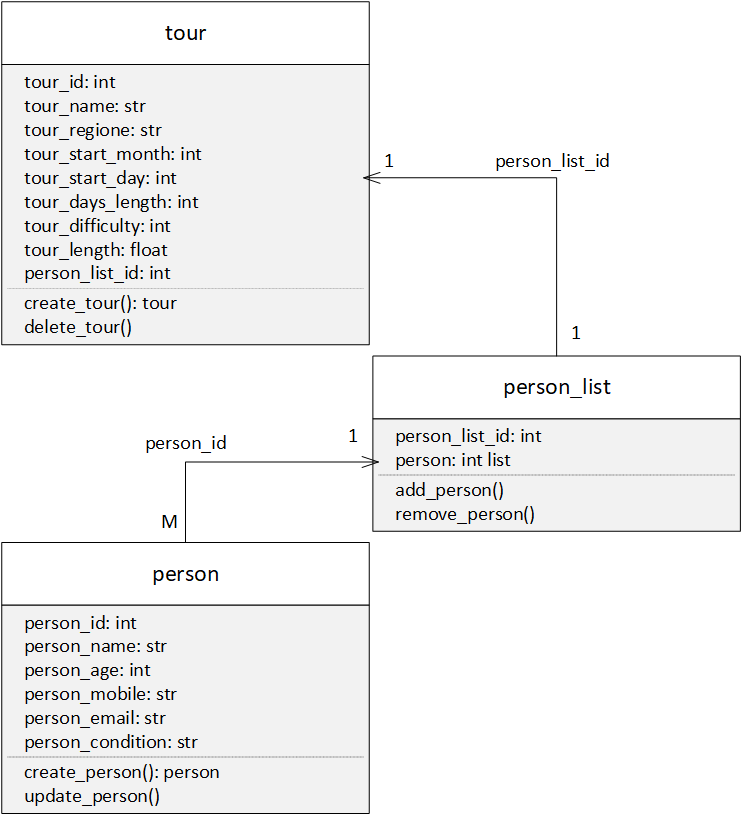
### вибір архітектури

моноліт бо додаток по суті дуже малий і не потребує розділеня на мікросервіси

### діаграма компонентів та їх в взаємодії



# Опис низькорівневої архітектури для хоча б двох модулів у моноліті.



# Розгортання програмного продукту.

## Скрипт контейнеризації сервісу (dockerfile)

FROM python:3.8.3-alpine

WORKDIR /usr/src/app

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

RUN apk update \

&& apk add postgresql-dev gcc python3-dev musl-dev

RUN pip install --upgrade pip

COPY ./requirements.txt .

RUN pip install -r requirements.txt

COPY ./entrypoint.sh .

COPY . .

ENTRYPOINT ["/usr/src/app/entrypoint.sh"]

## Скрипт оркестрації сервісів (docker-compose.yml)

version: '3.7'

services:

web:

build: ./app

command: python manage.py runserver 127.0.0.1:8000

volumes:

- ./app/:/usr/src/app/

ports:

- 8000:8000

env\_file:

- ./.env.dev

depends\_on:

- db

- redis

db:

image: postgres:12.0-alpine

volumes:

- postgres\_data:/var/lib/postgresql/data/

environment:

- POSTGRES\_USER=hello\_django

- POSTGRES\_PASSWORD=hello\_django

- POSTGRES\_DB=hello\_django\_dev

redis:

image: redis

ports:

"6379:6379"

# Висновки.

Обрана мова python показала себе добре впродовж розробки веб додатку. Фреймворк Django також був зручним, але для його нормального використання необхідно вивчити його вище базового рівня, на що було виділено не дуже багато часу. Вибір монолітної архітектури для даного випадку був вдалим, бо і справді розроблений додаток вкрай важко розділити щоб це було краще ніж зараз. Враховуючи що архітектурою обрано моноліт такого гнучкого масштабування веб додатку як у випадку мікросервісної архітектури не буде. Але для розробленого додатку це не є великою проблемою бо сам додаток є достатньо маленьким і не сильно потребує динамічного масштабування.

# Додаток.

## Моделі

### models.py

from django.db import models

class Tour(models.Model):

name = models.CharField('назва походу',max\_length = 100)

photo\_url = models.URLField('посилання на фото походу',max\_length = 1000) # maxlen 200 (default)

regione = models.CharField('територія де відбуватиметься похід',max\_length = 50)

month = models.PositiveSmallIntegerField('звичний місяць для проведення')

days\_len = models.PositiveSmallIntegerField('тривалість походу в днях')

difficulty = models.PositiveSmallIntegerField('складність')

tour\_len = models.PositiveSmallIntegerField('довжина маршруту в км')

def \_\_str\_\_(self):

return f'{self.id}\_{self.name}'

class Meta:

verbose\_name = 'Похід'

verbose\_name\_plural = 'Походи'

class Tour\_event(models.Model):

date = models.DateTimeField('дата створення події')

start\_date = models.DateTimeField('запланована дата початку походу')

tour = models.OneToOneField(Tour,on\_delete = models.CASCADE)

def \_\_str\_\_(self):

return f'{self.id}. {self.start\_date}\_{self.tour.name}'

class Meta:

verbose\_name = 'Подія походу'

verbose\_name\_plural = 'Події походу'

class Person(models.Model):

nickname = models.CharField('нік',max\_length = 30)

password = models.CharField('пароль',max\_length = 40)

name = models.CharField('ім`я',max\_length = 30)

age = models.PositiveSmallIntegerField('вік')

phone = models.CharField('мобільний номер',max\_length = 30)

email = models.EmailField('email',max\_length = 30)

condition = models.PositiveSmallIntegerField('фізичний стан')

tour\_event = models.ForeignKey(Tour\_event,on\_delete = models.SET\_NULL,blank=True,null=True) # одна людина не може бути записана одночасно в 2 походи (тому такий зв`язок є коректним)

def \_\_str\_\_(self):

return f'{self.id}. {self.name}({self.nickname})'

class Meta:

verbose\_name = 'Учасник'

verbose\_name\_plural = 'Учасники'

## робота з Кешем

вся робота з кешем виконується в *views.py*

### фрагмент settings.py

…

REDIS\_HOST = 'localhost'

REDIS\_PORT = 6379

…

## робота з БД

### models.py

(представлено вище)

### фрагмент settings.py

…

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

…

## обробники веб запитів

### urls.py (глобальні)

…

from django.contrib import admin

from django.urls import path,include

urlpatterns = [

path('tours/', include('tours.urls')),

# path('redis/', include('api.urls')),

path('admin/', admin.site.urls),

]

…

### urls.py (tours)

from django.urls import path

from . import views

app\_name = 'tours'

urlpatterns = [

path('',views.start, name = 'start'),

path('tour\_list',views.tour\_list\_view, name = 'tour\_list\_view'),

path('tour\_list/<int:tour\_id>',views.tour\_view, name = 'tour\_view'),

path('tour\_list/<int:tour\_id>/add\_person',views.add\_person, name = 'add\_person'),

path('register',views.register\_view, name = 'register\_view'),

path('register/create\_person',views.create\_person, name = 'create\_person'),

path('log\_in',views.log\_in\_view, name = 'log\_in\_view'),

path('log\_in/check',views.log\_in, name = 'log\_in'),

path('log\_out',views.log\_out, name = 'log\_out'),

]

### views.py (tours)

from django.shortcuts import render

from django.http import Http404, HttpResponseRedirect

from django.urls import reverse

from .models import Tour,Tour\_event,Person

import json

from django.conf import settings

import redis

redis\_instance = redis.StrictRedis(host=settings.REDIS\_HOST,port=settings.REDIS\_PORT, db=0)

# redis -> R

def Rget(key):

return redis\_instance.get(key)

def Rput(key,val):

redis\_instance.set(key, val)

def Rdelete(key):

redis\_instance.delete(key)

# http://127.0.0.1:8000

def start(request):

return HttpResponseRedirect(reverse('tours:tour\_list\_view'))

# http://127.0.0.1:8000/tour\_list

def tour\_list\_view(request):

tour\_list = Tour.objects.order\_by('difficulty')

PersonID = Rget('PersonID')

if PersonID:

login = Person.objects.get(id = int(PersonID))

return render(request, 'tours/tours\_list.html', {'tour\_list': tour\_list,'login':login})

return render(request, 'tours/tours\_list.html', {'tour\_list': tour\_list})

# http://127.0.0.1:8000/tour\_list/<int:tour\_id>

def tour\_view(request, tour\_id):

try:

tour = Tour.objects.get(id = tour\_id)

except:

raise Http404('Виникла помилка, походу з таким id не має)')

PersonID = Rget('PersonID')

if PersonID:

login = Person.objects.get(id = int(PersonID))

return render(request, 'tours/tour.html', {'tour': tour,'login': login})

return render(request, 'tours/tour.html', {'tour': tour})

# http://127.0.0.1:8000/tour\_list/<int:tour\_id>/add\_person

def add\_person(request, tour\_id):

return HttpResponseRedirect(reverse('tours:start'))

# http://127.0.0.1:8000/register

def register\_view(request):

return render(request, 'accounts/register.html')

# http://127.0.0.1:8000/register/create\_person

def create\_person(request):

user = Person(

nickname = request.GET['nickname' ],

password = request.GET['password' ],

name = request.GET['name' ],

age = request.GET['age' ],

phone = request.GET['phone' ],

email = request.GET['email' ],

condition = request.GET['condition'],

)

user.save()

Rput('PersonID',user.id)

return HttpResponseRedirect(reverse('tours:tour\_list\_view'))

# http://127.0.0.1:8000/log\_in

def log\_in\_view(request):

return render(request, 'accounts/log\_in.html')

# http://127.0.0.1:8000/log\_in/check

def log\_in(request):

try:

user = Person.objects.get(nickname=request.GET['nickname'])

if user.password == request.GET['password']:

Rput('PersonID',user.id)

return HttpResponseRedirect(reverse('tours:tour\_list\_view'))

else:

return Http404('Такого Акаунту з таким паролем не існує')

except:

raise Http404('Такого Акаунту з таким паролем не існує')

# http://127.0.0.1:8000/log\_out

def log\_out(request):

Rdelete('PersonID')

return HttpResponseRedirect(reverse('tours:tour\_list\_view'))