COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Rule-based Marketing Platform to Manage Call Detail Record

Chong Hu (ch3467)
Wenjie Chen (wc2685)
Yanchen Liu (yl4189)
Jiajing Sun (js5504)

TRANSCENDING DISCIPLINES, TRANSFORMING LIVES

Call Detail Record (CDR)

Produced by data generator

Data Generator

ID
CALLING_NUM
CALLED_NUM
START_TIME
DURATION
CALL_TYPE
CHARGE
CALL_RESULT

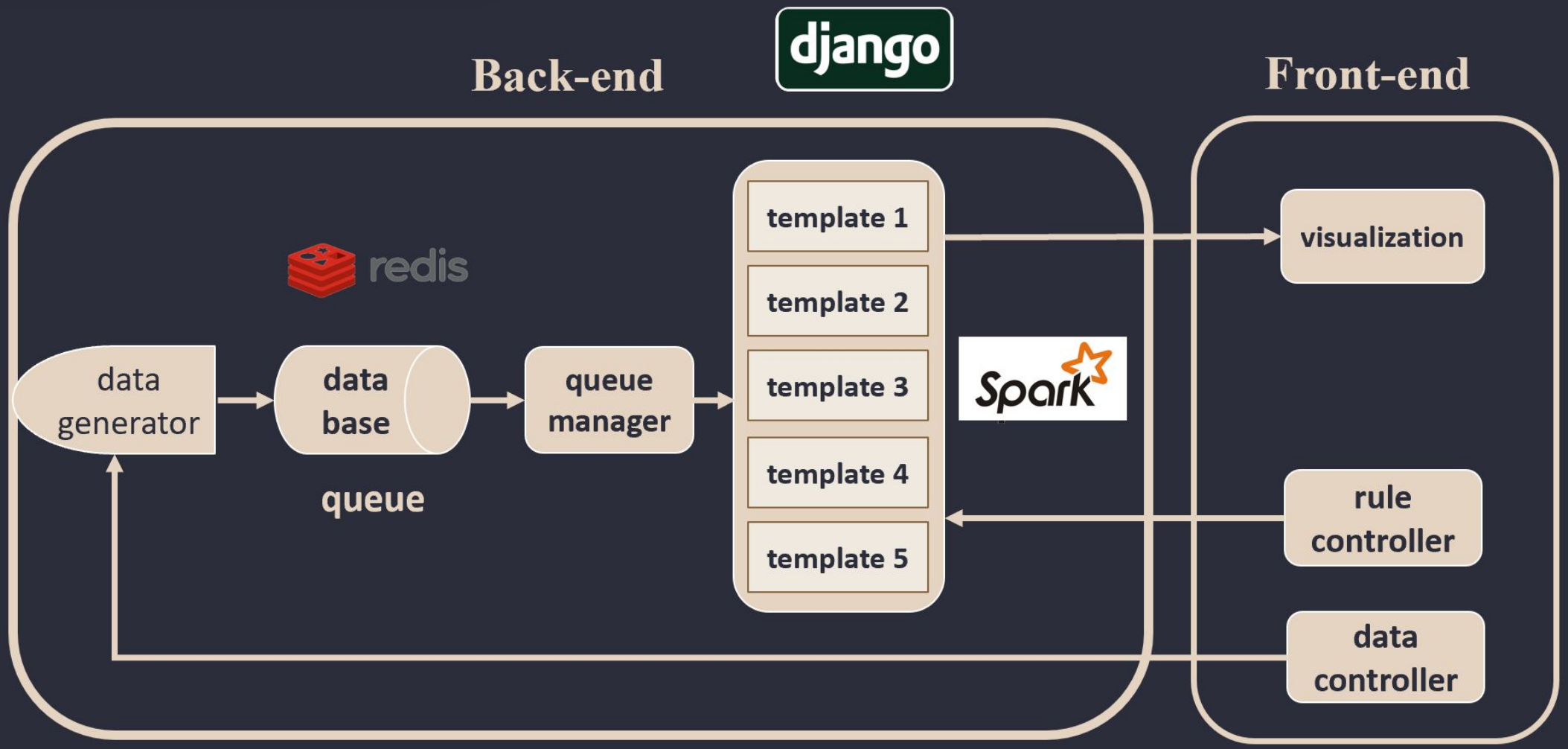
CDR

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Our Solution

template 1	Total duration Analysis
template 2	Business Type Analysis
template 3	International Analysis
template 4	Individual Tag Classification
template 5	Available Calling Time

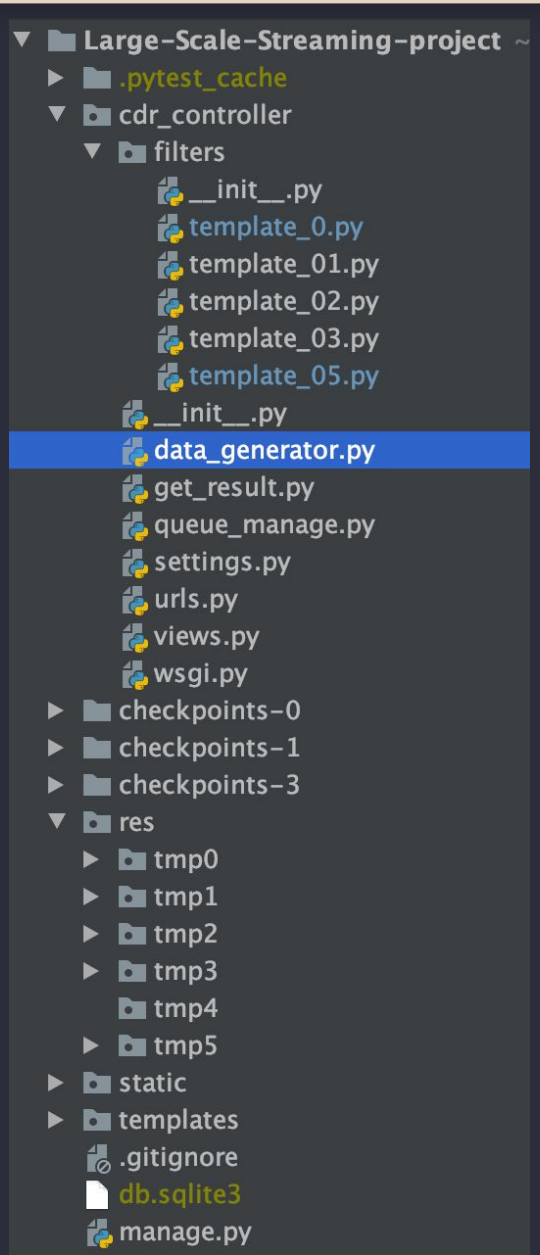
System Structure



Data generator

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Single CDR




```python
def __init__(self, ID=None, callednumber=None, teltime=None, teltype=None,
             charge=None, result=None, type=None,
             pick_type_distribution="default", rate_type_distribution=0.1,
             pick_call_distribution="default", delta_distribution="default",
             rate_place_distribution=0.7):

    self.ID = self.gen_ID()
    self.callednumber = self.gen_callednumber(rate_place_distribution)
    self.teltime = self.gen_teltime(pick_call_distribution,
                                    delta_distribution)
    self.teltype = self.gen_teltype()
    self.charge = self.gen_charge()
    self.result = self.gen_result()
    self.type = self.gen_type(pick_type_distribution,
                              rate_type_distribution)
```

```python
def gen_ID(self):
    return uuid.uuid1()

def gen_callednumber(self, rate_place_distribution):
    r1 = random.randint(1, 100)
    if r1 < 100 * rate_place_distribution:
        place = "1"
    else:
        place = random.randint(1, 300)
        while (region_code_for_country_code(place) == "ZZ"):
            place = random.randint(1, 300)

    first = str(random.randint(100, 999))
    second = str(random.randint(1, 888)).zfill(3)
    last = str(random.randint(1, 9998)).zfill(4)
    return '+{}-{}-{}-{}'.format(str(place), first, second, last)
```

```python
startt = 1588278283
pastt=1451624400
b = 133920 * startt - pastt
time1 = cur_t * 133920 - b
time2 = time1 + 133920
if call_distribution == "midnight mode":
    starttime = asctime(localtime(uniform(time1, time2)))
    while int(starttime.split(" ")[3].split(":")[0]) > 6 and i < 4:
        starttime = asctime(localtime(uniform(time1, time2)))
        i += 1
```

```python
def gen_teltype(self):
    if (random.randint(0, 1) == 0):
        return "SMS"
    else:
        return "VOICE"

def gen_charge(self):
    return random.random()

def gen_result(self):
    r = random.randint(1, 10)
    if (r < 9):
        return "AWNSERED"
    else:
        return "Busy"
```

```
np.random.exponential
np.random.poisson
np.random.binomial
```

Data generator

People

```
type = {
    # "Business",
    0: "Business",
    1: "Banking",
    2: "Financial agency",
    3: "Job",

    #  "Agency",
    4: "Legal agency",
    5: "Housekeeping and property management",

    #  "Education",
    6: "School",
    7: "Extracurricular training camp",

    # "Health",
    8: "Hospital (including health care)",
    9: "Clinic (including dentist)",

    # "AD",
    10: "Food (including takeaway)",
    11: "Dress code (booking and buying)",
    12: "Housing (including rental)",
    13: "Traveling",

    14: "Emergency",
    # "Private",
    15: "Private",
    16: "Private"
}
```

```
def gen_calltimes(self):
    return random.randint(10, 50)

def gen_data(self, ID=None, callednumber=None, teltime=None, teltype=None,
             charge=None, result=None, type=None,
             pick_type_distribution="default", rate_type_distribution=0.3,
             pick_call_distribution="default", delta_distribution="default",
             rate_place_distribution=0.7):
    for i in range(self.calltimes):
        data_generator_temp = data_generator(ID, callednumber, teltime,
                                             teltype, charge, result, type,
                                             pick_type_distribution,
                                             rate_type_distribution,
                                             pick_call_distribution,
                                             delta_distribution,
                                             rate_place_distribution)
        self.data.append(data_generator_temp)
```

```
def save_in_redis(self):
    for i in range(self.calltimes):
        tempdata = str(self.ID) + "|" + str(self.callnumber) + "|" \
                   + (str(self.data[i]))
        self.output_redis_1(self.data[i].ID, tempdata)
```

```
def output_redis_1(self, ID, tempdata):
    rds.lpush('ID_0',str(tempdata))
    rds.lpush('ID_01',str(tempdata))
    rds.lpush('ID_02',str(tempdata))
    rds.lpush('ID_03',str(tempdata))
    rds.lpush('ID_05',str(tempdata))
```

Template

```
def __init__(self, IP="localhost", interval=10, port=9000):
    # create spark context
    self.spark = SparkSession.builder.appName('template0').getOrCreate()
    self.sc = SparkContext.getOrCreate(SparkConf().setMaster("local[2]"))
    # create sql context, used for saving rdd
    self.sql_context = SparkSession(self.sc)
    # create the Streaming Context from the above spark context with batch interval size (seconds)
    self.ssc = StreamingContext(self.sc, 1)
    self.IP = IP
    self.interval = interval
    self.port = port
    # read data from port
    self.lines = self.ssc.socketTextStream(self.IP, self.port)
```

*

```
def count_duration(self):
    """
    This function is to read data from port 9000, then count the call time duration sum of every hour.
    """

    def updateFunc(new_values, last_sum):
        return sum(new_values) + (last_sum or 0)

    self.lines = self.lines.filter(lambda x: x)
    id_time_duration = self.lines.map(
        (lambda x: (x.split("|")[2], x.split("|")[4], x.split("|")[5])))
    temp_id_duration = id_time_duration.map(
        lambda x: (x[1].split(" ")[3].split(":")[0], int(x[2])))
    temp_id_duration_total = temp_id_duration.reduceByKey(
        lambda x, y: x + y).updateStateByKey(updateFunc)

    temp_id_duration_total.pprint()
    temp_id_duration_total.foreachRDD(
        lambda rdd: rdd.sortBy(lambda x: x[0]).toDF().toPandas().to_json(
            os.path.join(STORE_DIR, "tmp0",
                         "tmp0.json")) if not rdd.isEmpty() else None)
```

0

```
process_lines=self.lines.map(helper)
# process_lines.pprint()
people_type_count=process_lines.countByValue().map(lambda x: (x[0][0],x[0][1],x[1]))
# people_type_count.pprint()
# First, people with type
people_type_max=people_type_count.transform(lambda rdd: rdd.sortBy(lambda x: (x[0],-int(x[2]),x[1])).map(lambda x: (x[0],x[1])).reduceByKey(lambda x,y:x))

# people_type_max.pprint()
people_type_max.foreachRDD(lambda rdd: rdd.sortBy(lambda x: x[0]).toDF().toPandas().to_json(os.path.join(STORE_DIR, "tmp2", "pptype2.json")) if not rdd.isEmpty() else None)
# Second, people with tag
people_tag=people_type_count.map(mapper)
people_tag_max=people_tag.transform(lambda rdd: rdd.sortBy(lambda x: (x[0],-int(x[2]),x[1])).map(lambda x: (x[0],x[1])).reduceByKey(lambda x,y:x))
people_tag_max.pprint()
people_tag_max.foreachRDD(lambda rdd: rdd.sortBy(lambda x: x[0]).toDF().toPandas().to_json(os.path.join(STORE_DIR, "tmp2", "pptag2.json")) if not rdd.isEmpty() else None)
```

2

Template

1

```
def count_type(self):
    """
    This function is to read data and extract the call type.
    """
    def helper(x):
        rds_type = redis.Redis(host="localhost", port=6379,
                               decode_responses=True,
                               db=1)  # host是redis主机，需要redis服务端和客户端都启动 redis默认端口是6379
        res = "private" if rds_type.get(
            x.split("|")[3]) is None else rds_type.get(x.split("|")[3])
        rds_type.close()
        return res
    process_lines = self.lines.map(helper)
    def updateFunc(new_values, last_sum):
        return sum(new_values) + (last_sum or 0)

    resultstream = process_lines.map(
        lambda word: (word.lower(), 1)).reduceByKey(
        lambda x, y: x + y).updateStateByKey(updateFunc)
    resultstream.pprint()
    resultstream.foreachRDD(
        lambda rdd: rdd.sortBy(lambda x: x[0]).toDF().toPandas().to_json(
            os.path.join(STORE_DIR, "tmp1",
                         "type.json")) if not rdd.isEmpty() else None)
```

3

```
def count_place(self):
    def updateFunc(new_values, last_sum):
        return sum(new_values) + (last_sum or 0)
    callednumber = self.lines.map(lambda x: (x.split("|")[3]))
    place = callednumber.map(lambda x: region_code_for_country_code(
        int(x.split("-")[0].split("+")[1])))
    place_count = place.map(lambda place: (place, 1)).reduceByKey(
        lambda x, y: x + y).updateStateByKey(updateFunc)
    place_count.pprint()
    place_count.foreachRDD(
        lambda rdd: rdd.sortBy(lambda x: x[0]).toDF().toPandas().to_json(
            os.path.join(STORE_DIR, "tmp3",
                         "region.json")) if not rdd.isEmpty() else None)
```

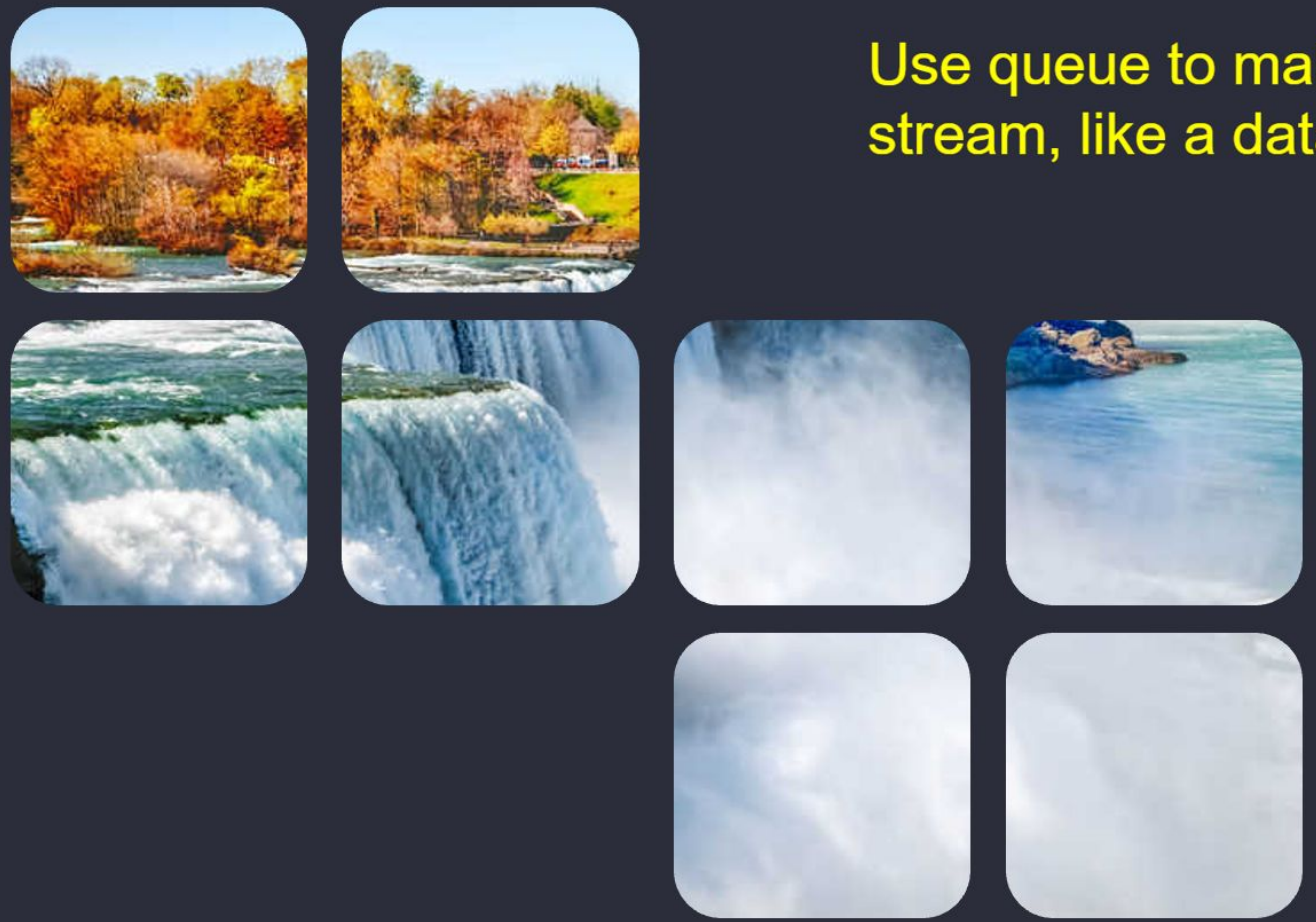
Template


```
people_calltime = self.lines.map(
    lambda x: (x.split("|")[0], x.split("|")[4]))

people_calltime_w = people_calltime.map(
    lambda x: (x[0] + ":" + x[1].split(" ")[0], 1))
people_calltime_d = people_calltime.map(
    lambda x: (x[0] + ":" + x[1].split(" ")[3].split(":")[0], 1))

people_calltime_w_count = people_calltime_w.reduceByKey(
    lambda x, y: x + y).map(
    lambda x: (x[0].split(":")[0], x[0].split(":")[1], x[1]))
people_calltime_d_count = people_calltime_d.reduceByKey(
    lambda x, y: x + y).map(
    lambda x: (x[0].split(":")[0], x[0].split(":")[1], x[1]))
people_calltime_w_count.foreachRDD(lambda rdd: rdd.sortBy(lambda x: (x[0], -x[2], x[1])).map(lambda x: (x[0], x[1])).distinct().reduceByKey(lambda x, y: x)
                                   .sortBy(lambda x: x[0]).toDF().toPandas().to_json(os.path.join(STORE_DIR, "tmp5", "day2.json")) if not rdd.isEmpty() else None)
people_calltime_w_count.pprint()
people_calltime_d_count.foreachRDD(lambda rdd: rdd.sortBy(lambda x: (x[0], -x[2], x[1])).map(lambda x: (x[0], x[1])).distinct().reduceByKey(lambda x, y: x)
                                   .sortBy(lambda x: x[0]).toDF().toPandas().to_json(os.path.join(STORE_DIR, "tmp5", "clock2.json")) if not rdd.isEmpty() else None)
people_calltime_d_count.pprint()
```

5

How was streaming used

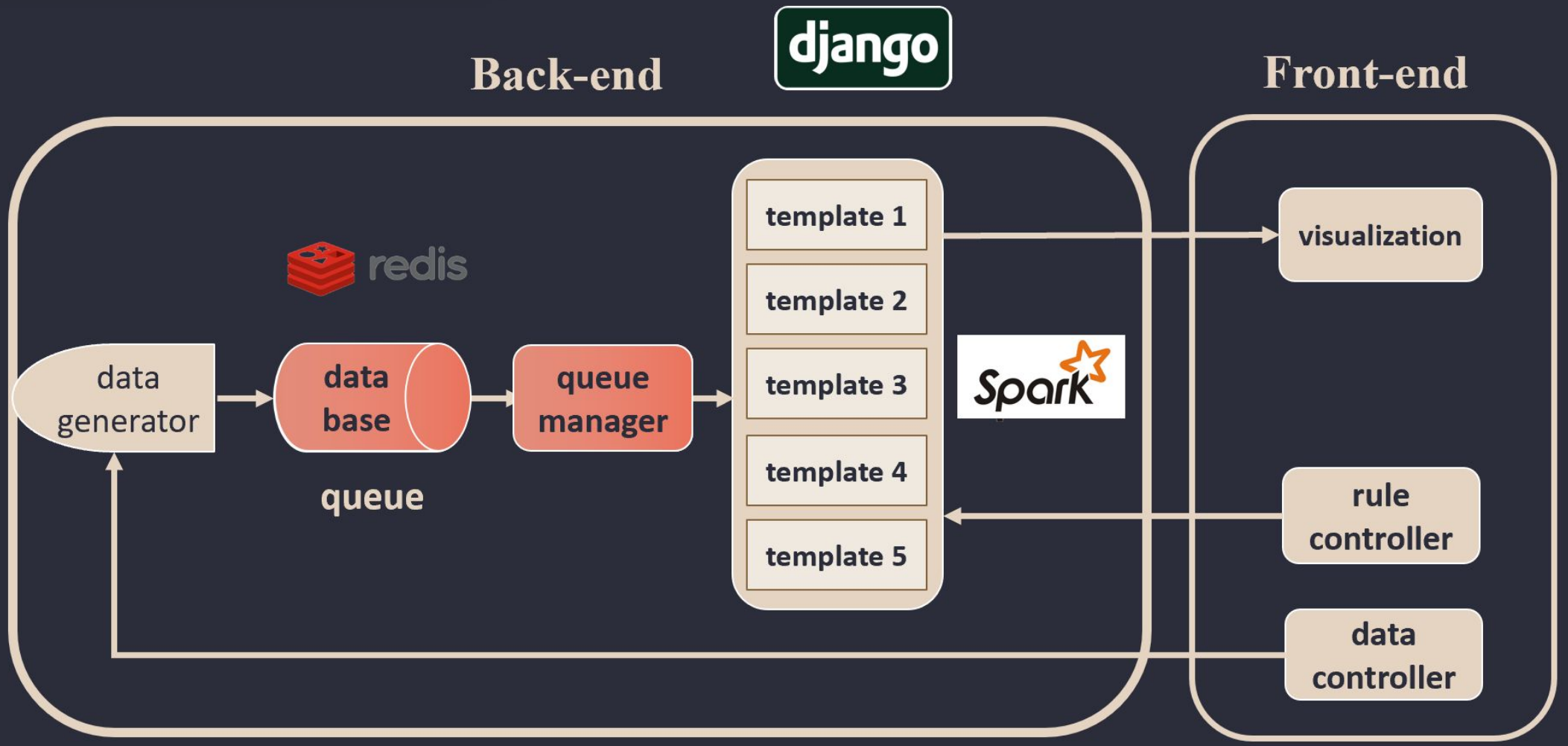
Database



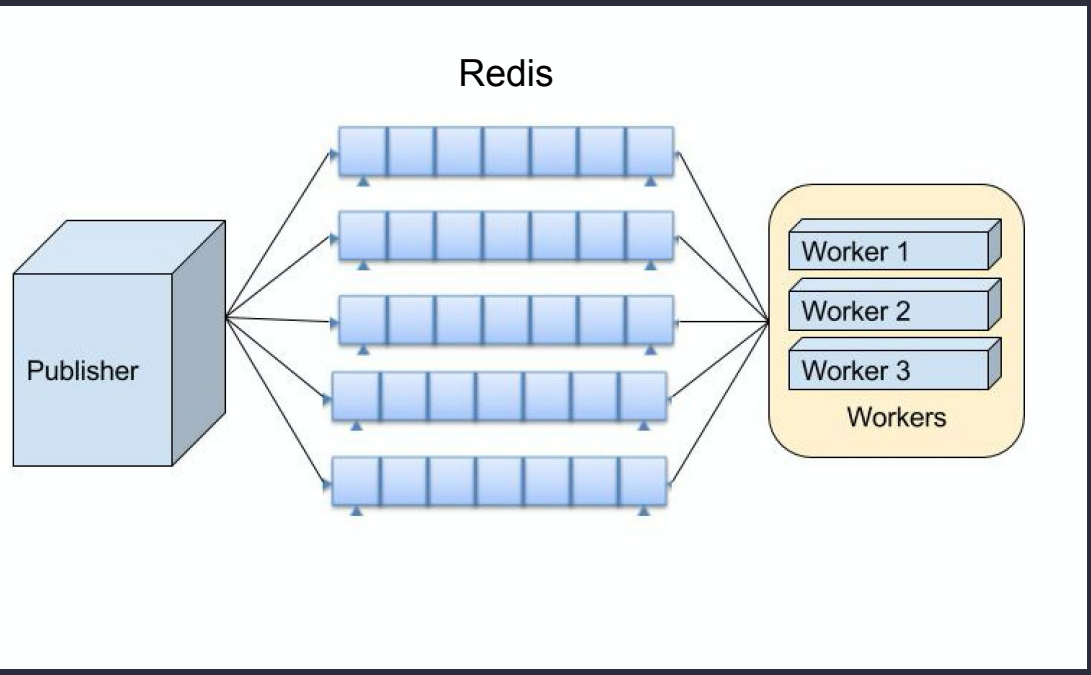
Use queue to make it like a stream, like a dataflow.

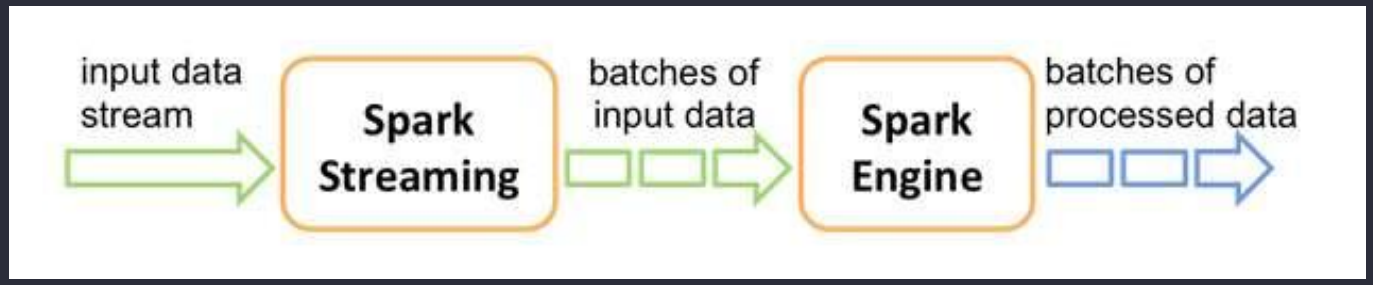
Streaming

System Structure

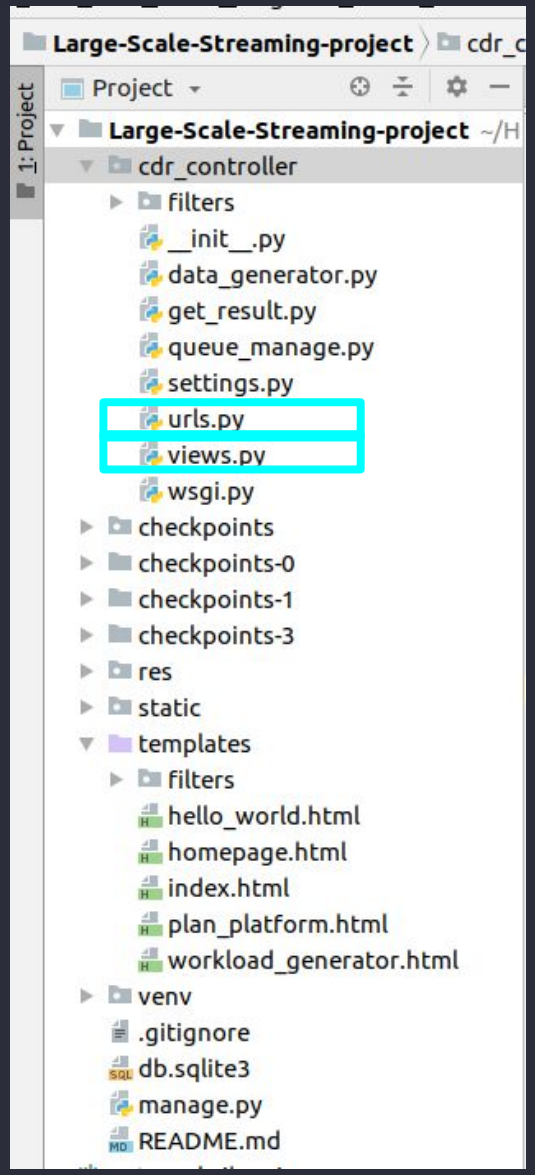


Redis Queue and Spark Streaming





Front-end Code

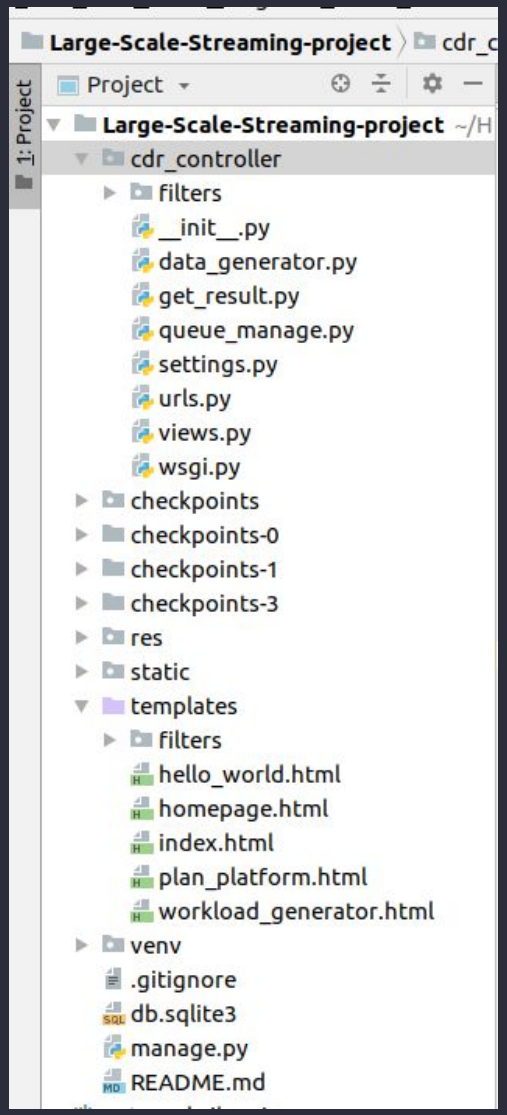



```python
from django.conf.urls import url
from django.contrib import admin
from django.urls import path

from . import views

urlpatterns = [
    # path('admin/', admin.site.urls),
    url(r'hello/', views.hello_world),
    url(r'admin/', admin.site.urls),
    url(r'index/', views.index, name="index"),
    url(r'workload_generator', views.workload_generator,
        name="workload_generator"),
    url(r'page1/', views.page1_view),
    url(r'page2/', views.page2_view),
    url(r'plan_platform', views.plan_platform, name="plan_platform"),
    url(r'show_info', views.show_info),
    url(r'^data_gen_start', views.data_gen_start),
    url(r'^data_gen_stop', views.data_gen_stop),
    # custom templates
    path(r"filters/template0", views.custom_template0, name="template0"),
    path(r"filters/template1", views.custom_template1, name="template1"),
    path(r"filters/template3", views.custom_template3, name="template3"),
    # data source
    path(r"data/template0", views.data_template0),
    path(r"data/template1", views.data_template1),
    path(r"data/template3", views.data_te
    url(r'', views.homepage, name="homepa
]
```

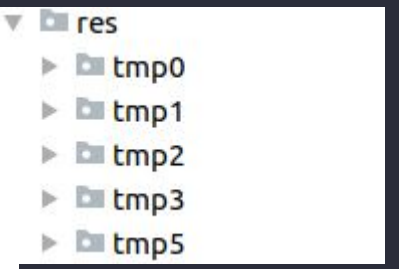
```python
85
86      def homepage(request):
87          return render(request, 'homepage.html', {})
88
89
90      def index(request):...
98
99
00      def workload_generator(request):...
30
31
32      def plan_platform(request):...
45
```

13 | CDR marketing platform

Front-end Code



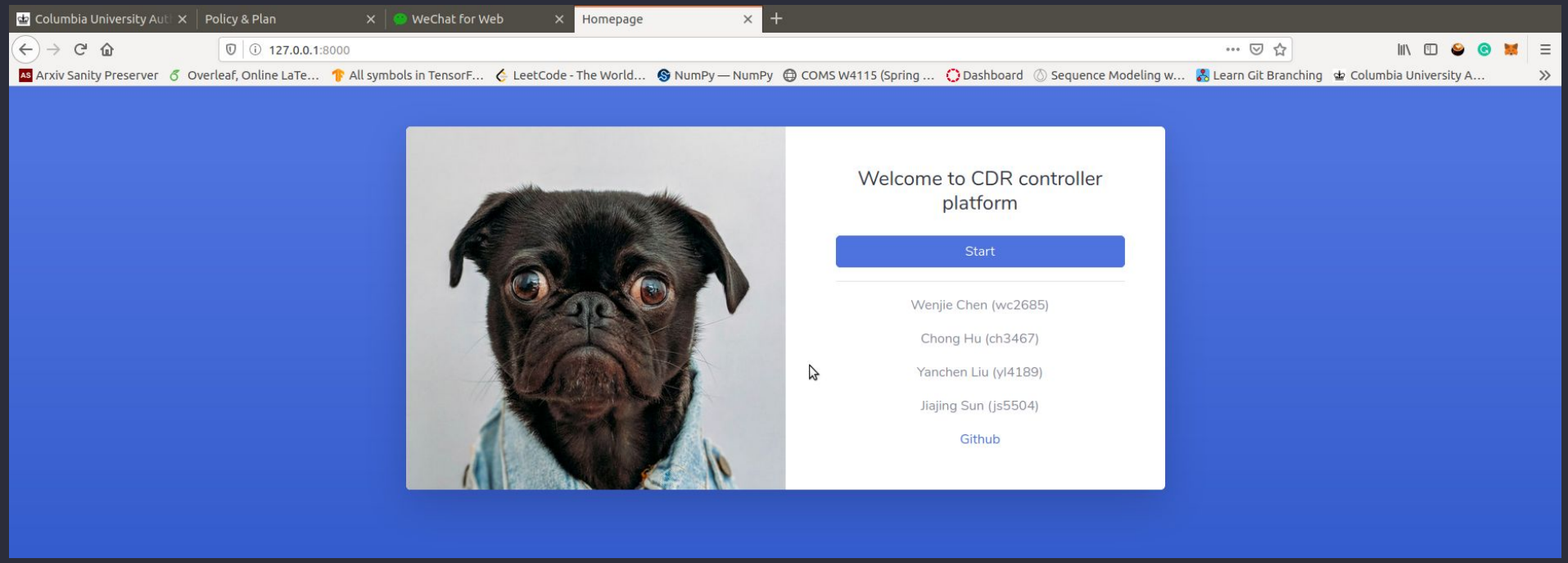

```python
# custom templates
def custom_template0(request):
    data_tmp0 = get_tmp0_data()
    return render(request, 'filters/template_00.html', data_tmp0)


def custom_template1(request):
    data_tmp1 = get_tmp1_data()
    return render(request, 'filters/template_01.html', data_tmp1)


def custom_template3(request):
    data_tmp3 = get_tmp3_data()
    return render(request, 'filters/template_03.html', data_tmp3)


# data source
def data_template0(request):
    data_tmp0 = get_tmp0_data()
    return HttpResponse(json.dumps(data_tmp0))


def data_template1(request):
    data_tmp1 = get_tmp1_data()
    return HttpResponse(json.dumps(data_tmp1))


def data_template3(request):
    data_tmp3 = get_tmp3_data()
    return HttpResponse(json.dumps(data_tmp3))
```



Final Results

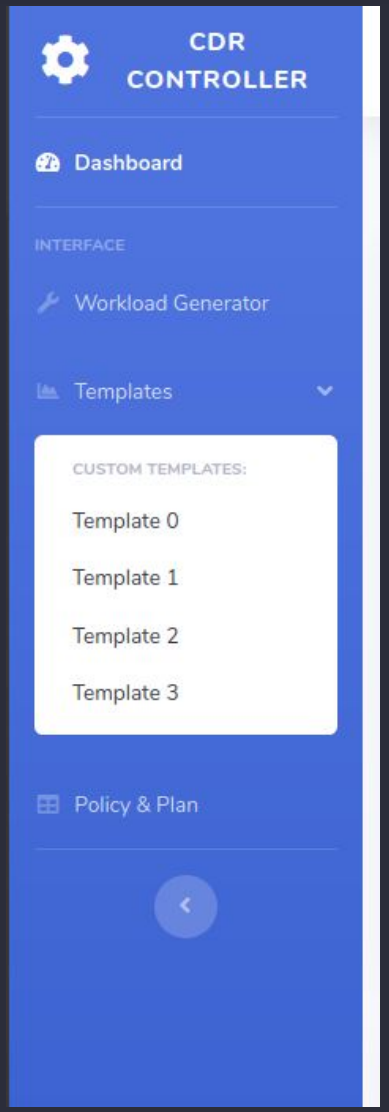
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

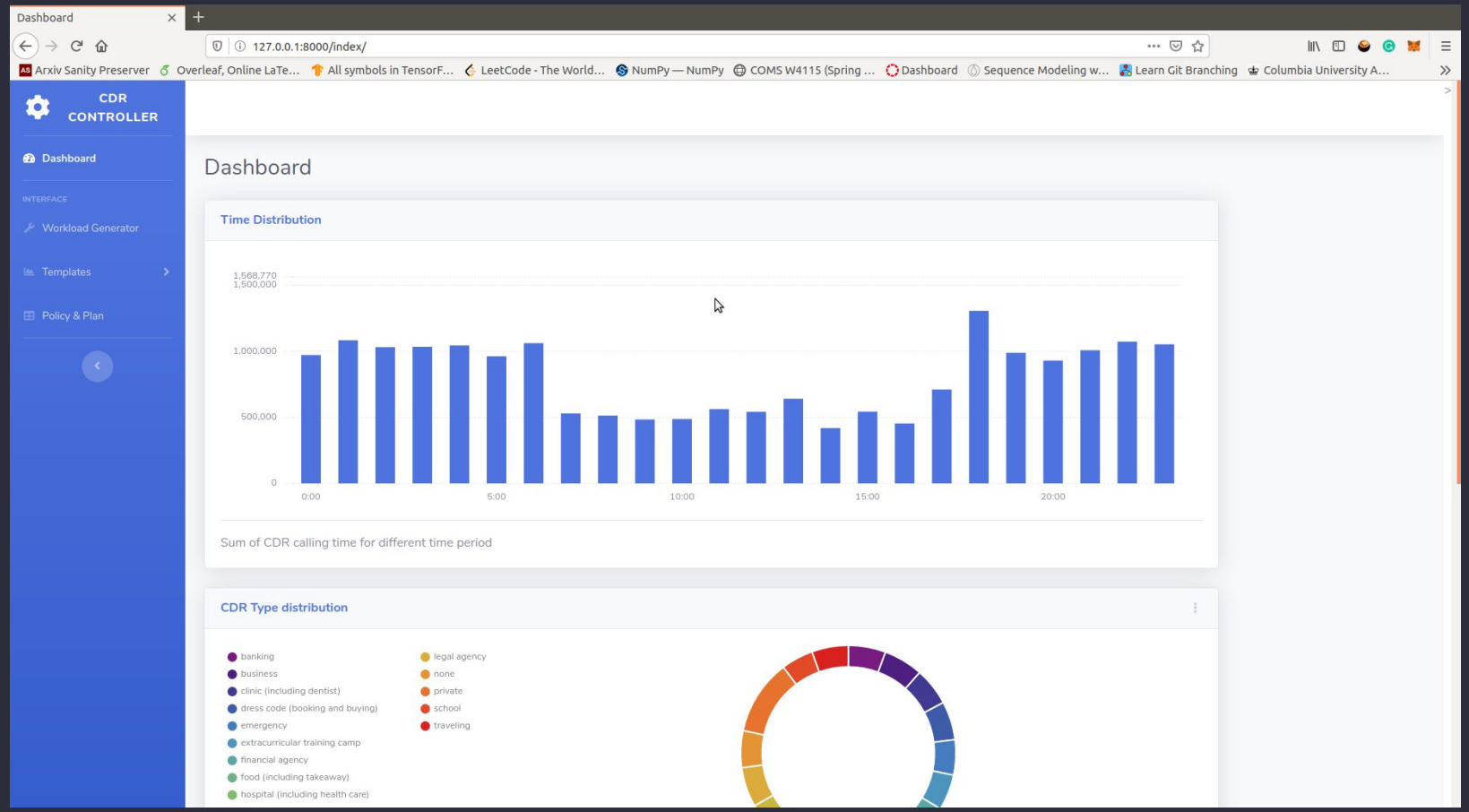
```
Performing system checks...

Watching for file changes with StatReloader
[INFO 2020-04-30 22:19:18,635] Watching for file changes with StatReloader
System check identified no issues (0 silenced).
April 30, 2020 - 22:19:18
Django version 3.0.5, using settings 'cdr_controller.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```



15 | **CDR marketing platform**

Final Results

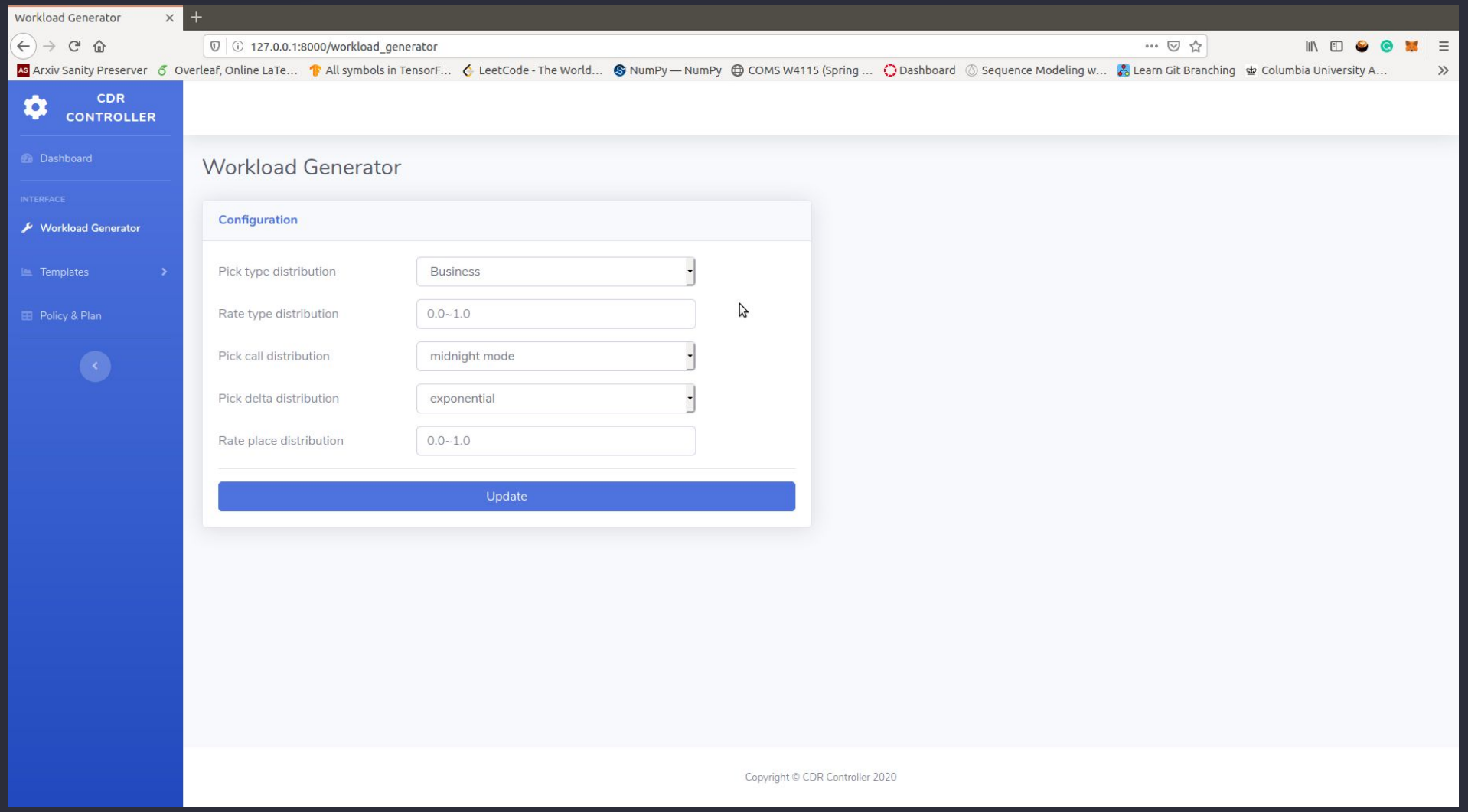




16 | **CDR marketing platform**

Final Results

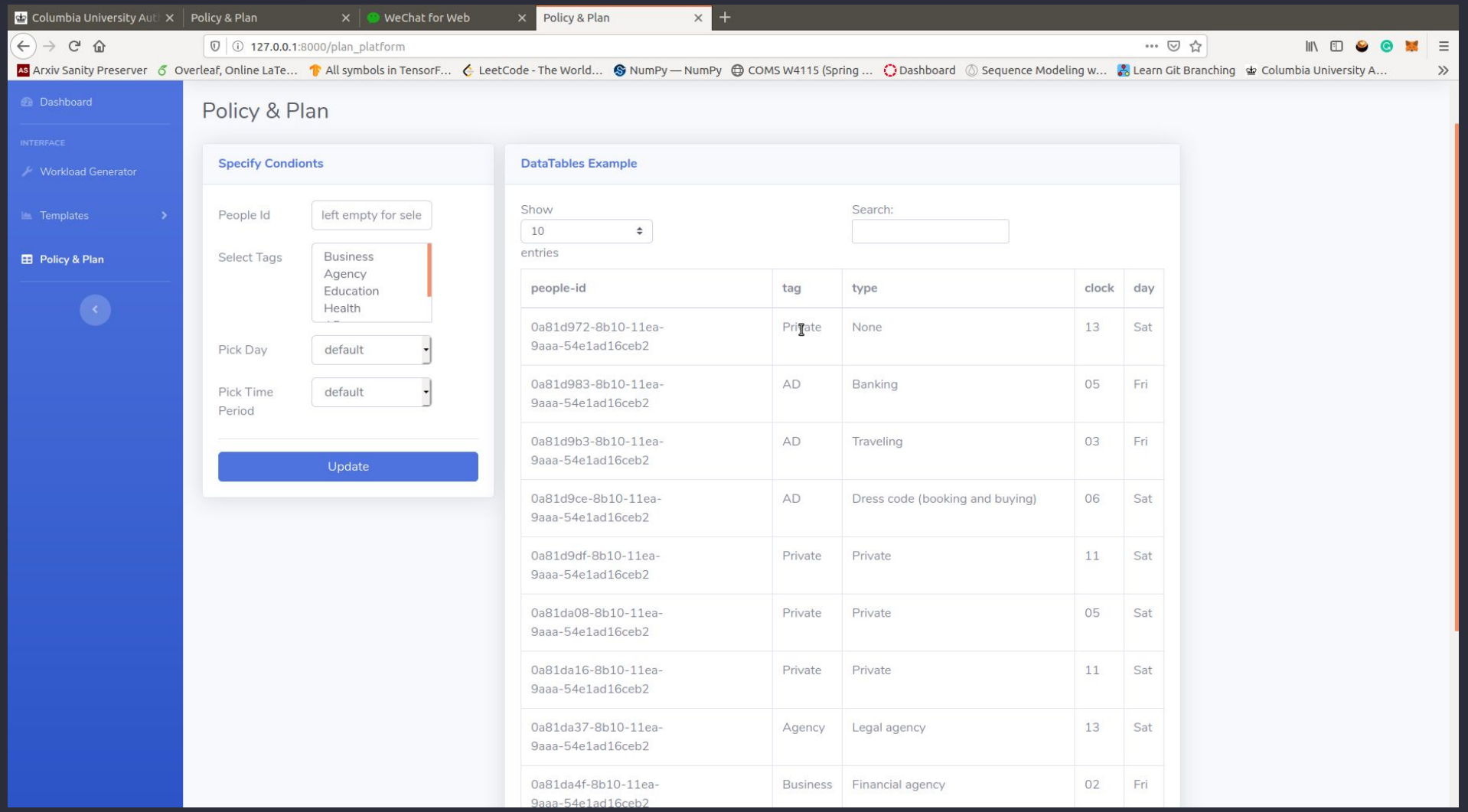
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science



17 | **CDR marketing platform**

Final Results

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science



18 | **CDR marketing platform**

Future Work

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

- System-Side:
  - Use kafka as message queue to deliver message for better extension ability and larger scale data stream
  - Optimize templates and data processing flow
  - Provide more api to control each templates
  - Back pressure

- Front-End:
  - More buttons and options to control templates
  - Login method


COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science
Thank You
TRANSCENDING DISCIPLINES, TRANSFORMING LIVES