Software Engineering Project
–Documentation–

Criminal Tracker with Suspect Prediction

Author: Daniel-Peter Reckerth

Team members:
Livia-Maria Mitrică
Daniel-Peter Reckerth
Group 30434/2

1 Abstract

Our project aims at implementing a system which helps investigators and law enforcement personnel carry their duties via a digitized interface.

As recent times showed, police forces placed a great emphasis on data and its application. Therefore the system was design in order to aid and supply the investigation corp with a common application. The main features of this application are as follows:

- add/edit/delete officers
- add/edit/delete cases, victims, suspects, criminals
- suspect prediction algorithm

The system contains many criminals and based on the profiling of those criminals the suspect prediction algorithm works. The suspect's data is retrieved supposedly at the place of the felony. Run through the system it should identify possible criminals already present in the system.

As far as technologies are concerned for developing this application we have used Python which is a powerful programming language and Django, a high-level web application framework enabling rapid development of web applications. Because of it's cleaner design it allowed a more easier administration of the application like templates, views, admin interface, support for database backend.

2 Source Code

2.1 Mini-project code

Earlier this semester we have provided a mini-project which comprises the MVT (Model-View-Template) architecture. This differs from the classical MVC and is specific to Django framework. The model follows the same guidelines, i.e. the logical data structure. Here the template (T) represents the presentation layer and view (V) is the data formatting part.

Below is the source code of the *models.py* file which is the source code of our models. As it can be seen many of the database parts were modeled here.

models.py

```
from django.db import models


# Create your models here.

class Criminal(models.Model):
    cid = models.CharField(max_length=20)
    cssn = models.IntegerField()
    cfirst_name = models.CharField(max_length=20)
    clast_name = models.CharField(max_length=20)
    cdob = models.DateField()
    cpob = models.CharField(max_length=15)

    class Meta:
        managed = True
        db_table = "criminal"

    def __str__(self):
        return 'Criminal #{}
        ↪  ({},{},{},{})'.format(self.cid,self.cfirst_name,self.clast_name,self.cdob,
        ↪  self.cpob)

class Victim(models.Model):
    vid = models.CharField(max_length=20)
    vssn = models.IntegerField()
    vfirst_name = models.CharField(max_length=20)
    vlast_name = models.CharField(max_length=20)
    vdob = models.DateField()
    vpob = models.CharField(max_length=15)

    class Meta:
        managed = True
        db_table = "victim"

    def __str__(self):
        return 'Victim #{} ({}
        ↪  {},{},{})'.format(self.vid,self.vfirst_name,self.vlast_name,self.vdob,
        ↪  self.vpob)
        #return self.vfirst_name
```

The view is presented below, and it follows the classic Django rules of creating views and returning a

html response, namely a page.

views.py

```
from django.http import *
from django.shortcuts import render
from .models import *


# Create your views here.

def home(request):
    # return HttpResponse("<h1>Hello world</h1>")
    return render(request, 'home.html', {'name': 'SE mini project'})


def add(request):
    value1 = int(request.GET['num1'])
    value2 = int(request.GET['num2'])
    res = value1 + value2

    return render(request, 'result.html', {'result': res})


def index(request):
    victims = Victim.objects.all()
    return render(request, 'result.html', {'obj': victims})
```

The urls file is a Django URL scheme called a URL configurtion, which is pure Python code and is actually a mapping between URL path expression to Python functions, i.e. the views.

urls.py

```
from django.urls import path

from . import views

urlpatterns= [
    path('', views.home, name='home')
    ,path('add', views.add, name='add')
    ,path('index', views.index, name='index')
]
```

In the apps file a registry of installed application which store the configurations is present. It maintains a list of models.

apps.py

```
from django.apps import AppConfig


class MyapptestConfig(AppConfig):
    name = 'myAppTest'
```

The admin interface is very helpful because it is automatic. The models are registered here. In this interface an administrator can add officers, cases, update the criminals database and input the suspect's

description for a specific case. We will present pictures later on, of how the admin interface looks like. Below is the code.

admin.py

```
from django.contrib import admin
from .models import *

# Register your models here.

admin.site.register(Victim)
admin.site.register(Criminal)
```

Some of this code was present in the mini-project. We had then a basic models, comprising of the criminal and victims class. Because we also have a suspect prediction algorithm we have developed other types of models, namely the suspect, case type, officer and case models.

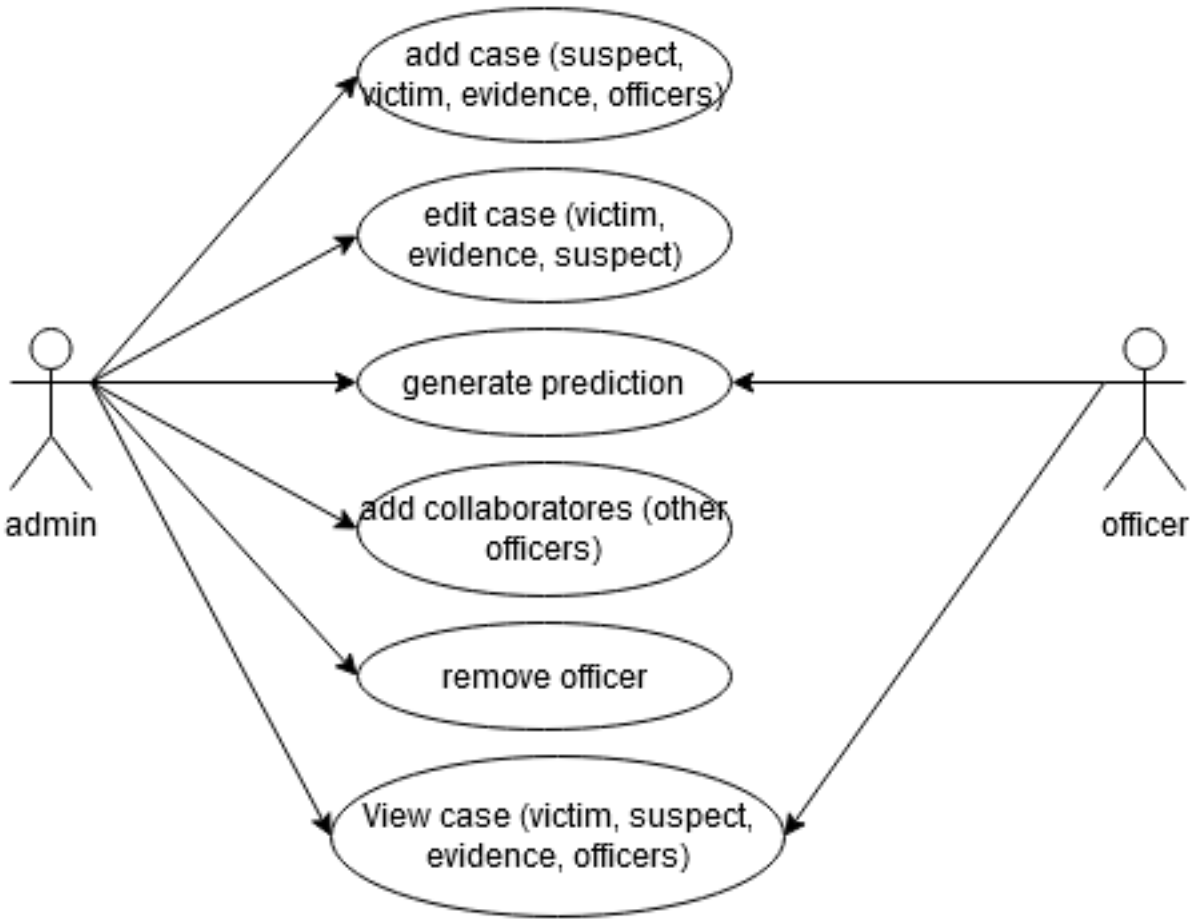
The view also contains different functions related to the final project besides the home one of the mini-project.

3 Final Project

3.1 Diagrams: Use Case

Presented already was a UML use-case diagram which portrays the main operations an admin or an officer can do in our system. The picture is shown below:

Figure 1: Use-case diagram



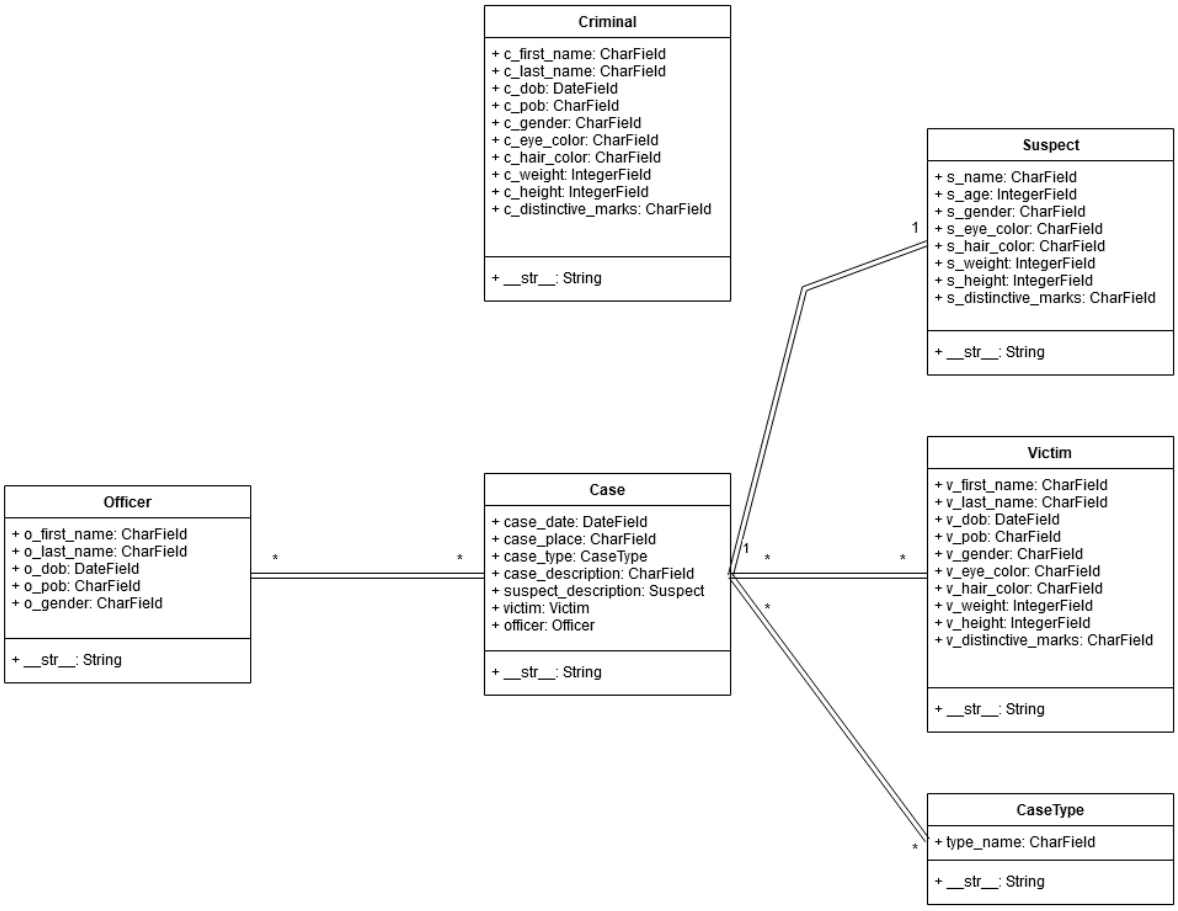
Most of this capabilities are achievable thorough the admin interface.

3.2 Diagrams: UML Class Diagram

We will present below the UML class diagram of the classes defined in our models. This diagram comprises all of the models not only those of the mini-project. The source code regarding the new models.py file is shown later.

As stated previously we have created six models, namely: *Suspect, Victim, CaseType, Officer, Case, Criminal*. We have mentioned early what they represent. The class diagram:

Figure 2: UML models class diagram



3.3 MVT Architecture

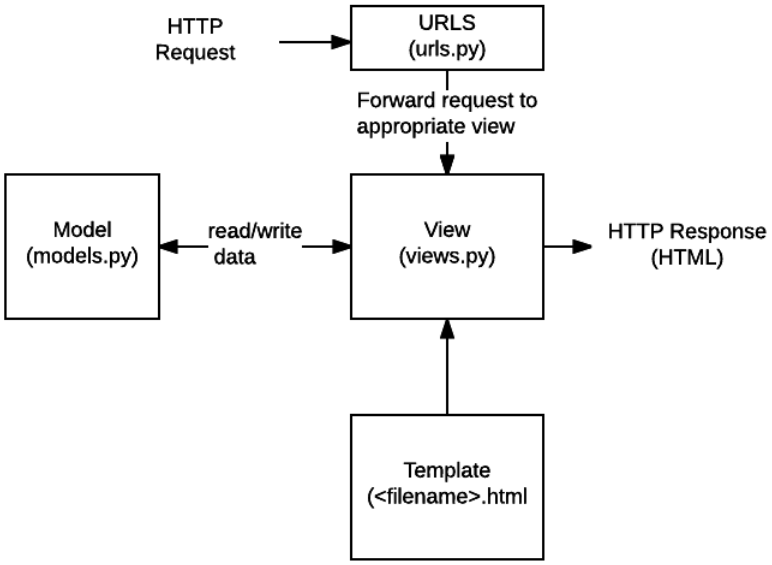
Our system follows the general MVT architecture.

The model acts as the link between the database and the server. It represent a representation of the data structure used by our website. As database is concerned we have used MySQL database integrated with XAMPP by Apache and we have created the database via the admin page of the XAMPP.

The view contains the logic to be displayed. The view is not exactly the controller in Django because Django Views are corresponding only to a particular template, so they are not selecting a model, per se.

The templates are specific to frontend. It contains the static parts of the HTML output as well as special syntax of dynamic content. It is very useful in generating HTML dynamically.

Figure 3: MVT Architecture



4 Source code of the final project

The models file contains all the models defined. It must be denoted that we have defined predetermined features (eye colors, hair colors, distinctive marks, etc.) for ease of usage in the admin database. Below is the source code.

models.py

```
from django.core.validators import MaxValueValidator, MinValueValidator
from django.db import models

# Create your models here.

# TODO eliminate the ids and create an auto increment
# TODO experiment with info retrieval

FEMALE = 'F'
MALE = 'M'
UNKNOWN = 'U'

GENDER_TYPE = [(FEMALE, 'female'),
               (MALE, 'male'),
               (UNKNOWN, 'U')]

BLUE = 'BLU'
GREEN = 'GRE'
BROWN = 'BR'
GRAY = 'GRA'
BLACK = 'BLA'

EYE_COLOR = [(BLUE, 'blue'),
             (BLACK, 'black'),
             (BROWN, 'brown'),
             (GRAY, 'gray'),
             (GREEN, 'green'),
             (UNKNOWN, 'U')]

BLONDE = 'BLO'
GINGER = 'GI'
WHITE = 'WH'
OTHER = 'O'

HAIR_COLOR = [(BLONDE, 'blonde'),
              (BLACK, 'black'),
              (BROWN, 'brown'),
              (GRAY, 'gray'),
              (GINGER, 'ginger'),
              (WHITE, 'white'),
              (OTHER, 'other'),
              (UNKNOWN, 'U')]

NONE = 'NONE'
FRECKLE = 'freckle'
WRINKLE = 'wrinkle'
MOLE = 'mole'
DARK_POINT = 'dark point'
```

```
49
50  DISTINCTIVE_MARKS = [(NONE, 'none'),
51                       (FRECKLE, 'freckle'),
52                       (WRINKLE, 'wrinkle'),
53                       (MOLE, 'mole'),
54                       (DARK_POINT, 'dark point'),
55                       (UNKNOWN, 'unknown')]
56
57  class Suspect(models.Model):
58      s_name = models.CharField(max_length=20)
59      s_age = models.IntegerField(validators=[MinValueValidator(5),
        ↪  MaxValueValidator(100)])
60      s_gender = models.CharField(max_length=15, choices=GENDER_TYPE, default=UNKNOWN)
61      s_eye_color = models.CharField(max_length=15, choices=EYE_COLOR, default=UNKNOWN)
62      s_hair_color = models.CharField(max_length=15, choices=HAIR_COLOR, default=UNKNOWN)
63      s_height = models.IntegerField(validators=[MinValueValidator(100),
        ↪  MaxValueValidator(250)], default=-1)  # in cm
64      s_weight = models.IntegerField(validators=[MinValueValidator(30),
        ↪  MaxValueValidator(250)], default=-1)  # in kgs
65      s_distinctive_marks = models.CharField(max_length=100, choices=DISTINCTIVE_MARKS,
        ↪  default="no specific marks")
66
67      class Meta:
68          managed = True
69          db_table = "suspect"
70
71      def __str__(self):
72          return 'Suspect ({},{},{},{} cm,{} kgs)'\
73              .format(self.s_name, self.s_gender, self.s_age, self.s_height, self.s_weight)
74
75
76  class Victim(models.Model):
77      v_first_name = models.CharField(max_length=20)
78      v_last_name = models.CharField(max_length=20)
79      v_dob = models.DateField()
80      v_pob = models.CharField(max_length=75, default=UNKNOWN)
81      v_gender = models.CharField(max_length=15, choices=GENDER_TYPE, default=UNKNOWN)
82      v_eye_color = models.CharField(max_length=15, choices=EYE_COLOR, default=UNKNOWN)
83      v_hair_color = models.CharField(max_length=15, choices=HAIR_COLOR, default=UNKNOWN)
84      v_height = models.IntegerField(validators=[MinValueValidator(100),
        ↪  MaxValueValidator(250)], default=-1)  # in cm
85      v_weight = models.IntegerField(validators=[MinValueValidator(30),
        ↪  MaxValueValidator(250)], default=-1)  # in kgs
86
87      class Meta:
88          managed = True
89          db_table = "victim"
90
91      def __str__(self):
92          return 'Victim ({} {},{},{})'.format(self.v_first_name, self.v_last_name,
            ↪  self.v_dob, self.v_pob)
93
94
95  class CaseType(models.Model):
```

```
96      type_name = models.CharField(max_length=15)
97
98      class Meta:
99          managed = True
100         db_table = "case_type"
101
102     def __str__(self):
103         return '{}'.format(self.type_name)
104
105
106 class Officer(models.Model):
107     o_first_name = models.CharField(max_length=20)
108     o_last_name = models.CharField(max_length=20)
109     o_dob = models.DateField()
110     o_pob = models.CharField(max_length=75, default=UNKNOWN)
111     o_gender = models.CharField(max_length=15, choices=GENDER_TYPE, default=UNKNOWN)
112
113     class Meta:
114         managed = True
115         db_table = "officer"
116
117     def __str__(self):
118         return 'Officer ({} {})' \
119             .format(self.o_first_name, self.o_last_name)
120
121
122 class Case(models.Model):
123     case_date = models.DateField()
124     case_place = models.CharField(max_length=75)
125     case_type = models.ManyToManyField(CaseType)
126     case_description = models.CharField(max_length=250)
127     suspect_description = models.ForeignKey(Suspect, on_delete=models.CASCADE)
128     victim = models.ManyToManyField(Victim)
129     officers = models.ManyToManyField(Officer)
130
131     class Meta:
132         managed = True
133         db_table = "case"
134
135     def __str__(self):
136         return 'Case ({},{},{},{},{},{})' \
137             .format(self.case_date,
138                     self.case_place,
139                     ", ".join(case_t.__str__() for case_t in self.case_type.all()),
140                     ", ".join(victim.__str__() for victim in self.victim.all()),
141                     self.suspect_description,
142                     ", ".join(officer.__str__() for officer in self.officers.all()))
143
144
145 class Criminal(models.Model):
146     # TODO add info about if he vas in prison
147     c_first_name = models.CharField(max_length=20)
148     c_last_name = models.CharField(max_length=20)
149     c_dob = models.DateField()
```

```
    c_pob = models.CharField(max_length=75, default=UNKNOWN)
    c_gender = models.CharField(max_length=15, choices=GENDER_TYPE, default=UNKNOWN)
    c_eye_color = models.CharField(max_length=15, choices=EYE_COLOR, default=UNKNOWN)
    c_hair_color = models.CharField(max_length=15, choices=HAIR_COLOR, default=UNKNOWN)
    c_height = models.IntegerField(validators=[MinValueValidator(100),
    ↪  MaxValueValidator(250)], default=-1)  # in cm
    c_weight = models.IntegerField(validators=[MinValueValidator(30),
    ↪  MaxValueValidator(250)], default=-1)  # in kgs
    c_distinctive_marks = models.CharField(max_length=100, choices=DISTINCTIVE_MARKS,
    ↪  default="no specific marks")
    c_record = models.ManyToManyField(CaseType)

    class Meta:
        managed = True
        db_table = "criminal"

    def __str__(self):
        return 'Criminal ({} {},{},{},criminal record:{})'\
            .format(self.c_first_name,
                    self.c_last_name,
                    self.c_dob,
                    self.c_pob,
                    ", ".join(case_t.__str__() for case_t in self.c_record.all()))
```

The *urls.py, admin.py, apps.py* have been modified accordingly in order to add our new paths and to register our new models. The *manage.py* file and *setting.py* come with Django and they are essential for running the web application. In *settings.py* we have defined the connection to our database, MYSQL and we specify apps and other features. Below are their source codes.

urls.py

```
from django.urls import path

from . import views

urlpatterns= [
    path('', views.home, name='home')
    ,path('add', views.add, name='add')
    ,path('index', views.index, name='index')
    ,path('generate_suspects', views.generate_suspects, name='generate_suspects')
    ,path('d_tree', views.d_tree, name='d_tree')
]
```

admin.py

```
from django.contrib import admin
from .models import *

# Register your models here.

admin.site.register(Victim)
admin.site.register(Criminal)
admin.site.register(Suspect)
admin.site.register(Officer)
admin.site.register(Case)
admin.site.register(CaseType)
```

apps.py

```
from django.apps import AppConfig


class MyapptestConfig(AppConfig):
    name = 'myAppTest'
```

settings.py

```
"""
Django settings for myApp project.

Generated by 'django-admin startproject' using Django 3.1.3.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/
"""
import os
from pathlib import Path


# Build paths inside the project like this: BASE_DIR / 'subdir'.
import export as export

BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '-b24cu%g2anvoc8v((gsa8k15))_zf0o5kl+3ftnwgl)r%ud4i'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myAppTest',
]

```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'myApp.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'myApp.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
        'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'se_proj_db',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
        'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'"
        }
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
```

```
        'NAME':
        ↪  'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]

STATIC_ROOT = os.path.join(BASE_DIR, 'assets')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'myApp.settings')
```

manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'myApp.settings')
```

```
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Perhaps the most interesting source code file is the *views.py* which contains the view. We will post it and then comment on it.

views.py

```
from dateutil.utils import today
from django.http import *
from django.shortcuts import render
from .models import *
from dateutil.relativedelta import relativedelta
import pandas as pd
from sklearn import tree
import pydotplus
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import matplotlib.image as pltimg

# Create your views here.

def home(request):
    # return HttpResponse("<h1>Hello world</h1>")
    return render(request, 'home.html', {'name': 'SE mini project'})


def add(request):
    value1 = int(request.GET['num1'])
    value2 = int(request.GET['num2'])
    res = value1 + value2

    return render(request, 'result.html', {'result': res})


def index(request):
    victims = Victim.objects.all()
    query_set_1 = Criminal.objects.prefetch_related('c_record')
    criminals = []
    for criminal in query_set_1:
        case_types = [case_t.type_name for case_t in criminal.c_record.all()]
        criminals.append({'id': criminal.id,
                          'c_first_name': criminal.c_first_name,
                          'c_last_name': criminal.c_last_name,
                          'c_pob': criminal.c_pob,
```

```
                            'c_dob': criminal.c_dob,
                            'record': case_types})

    cases_set = Case.objects.prefetch_related('case_type')

    cases = []
    for case in cases_set:
        case_types = [case_t.type_name for case_t in case.case_type.all()]
        cases.append({'id': case.id,
                      'case_date': case.case_date,
                      'case_place': case.case_place,
                      'case_type': case_types,
                      'case_description': case.case_description,
                      'suspect_description': case.suspect_description})

    query_set_victims = Case.objects.prefetch_related('victim')
    for case in query_set_victims:
        victims_set = [v.__str__() for v in case.victim.all()]
        cases[int(case.id)-1].update({'victim': victims_set})

    query_set_officers = Case.objects.prefetch_related('officers')
    for case in query_set_officers:
        officers_set = [o.__str__() for o in case.officers.all()]
        cases[int(case.id)-1].update({'officers': officers_set})

    return render(request, 'result.html', {'victims': victims, 'criminals': criminals,
    ↪  'cases': cases})

# algorithm of generating suspects based on their description and decision tree
↪  prediction of felony
def generate_suspects(request):
    case_id = int(request.GET['case_id'])
    case = Case.objects.get(id=case_id)
    suspect = case.suspect_description

    #criminals_same_gender = Criminal.objects.filter(c_gender=suspect.s_gender)
    query_set_1 = Criminal.objects.prefetch_related('c_record')
    criminals = []
    for criminal in query_set_1:
        case_types = [case_t.type_name for case_t in criminal.c_record.all()]
        if criminal.c_gender == suspect.s_gender:
            criminals.append({'id': criminal.id,
                              'c_first_name': criminal.c_first_name,
                              'c_last_name': criminal.c_last_name,
                              'c_gender': criminal.c_gender,
                              'c_pob': criminal.c_pob,
                              'c_dob': criminal.c_dob,
                              'c_eye_color': criminal.c_eye_color,
                              'c_hair_color': criminal.c_hair_color,
                              'c_height': criminal.c_height,
                              'c_weight': criminal.c_weight,
                              'c_distinctive_marks': criminal.c_distinctive_marks,
                              'record': case_types,
                              'match': 10})
```

```
90
91      case_related = Case.objects.prefetch_related('case_type').get(id=case_id)
92      case_types = [case_t.type_name for case_t in case_related.case_type.all()]
93
94      d = {1: 'murder', 2: 'arson', 3: 'assault', 4: 'robbery', 5: 'tax evasion', 6:
        ↪  'cybercrime', 7: 'homicide', 8: 'forgery', 9: 'blackmail', 10: 'harassment', 11:
        ↪  'domestic abuse'}
95      suspect_offense_prediction = predict_suspect(suspect)
96
97      offense = d[suspect_offense_prediction[0]]
98      print(offense)
99
100     for criminal in criminals:
101         physical_match = 0
102         match_prediction = 0
103         age_criminal = relativedelta(today(), criminal['c_dob']).years
104         if age_criminal - 5 <= suspect.s_age <= age_criminal + 5:
105             physical_match += 10
106         else:
107             physical_match += 1
108
109         if int(criminal['c_height']) - 5 <= int(suspect.s_height) <=
            ↪  int(criminal['c_height']) + 5:
110             physical_match += 10
111         else:
112             physical_match += 1
113
114         if int(criminal['c_weight']) - 5 <= int(suspect.s_height) <=
            ↪  int(criminal['c_weight']) + 5:
115             physical_match += 10
116         else:
117             physical_match += 5# losing or gaining weight is very possible
118
119         if criminal['c_eye_color'] == suspect.s_eye_color:
120             physical_match += 10
121         else:
122             physical_match += 1
123
124         if criminal['c_hair_color'] == suspect.s_hair_color:
125             physical_match += 10
126         else:
127             physical_match += 5# dyeing the hair or getting white is possible
128
129         if criminal['c_distinctive_marks'] == suspect.s_distinctive_marks:
130             physical_match += 10
131         else:
132             physical_match += 2
133
134         if criminal['record'][0] == offense:
135             match_prediction += 15
136         else:
137             match_prediction += 5
138
139         physical_match_percent = physical_match / 60 * 100
```

```
140             match_prediction_percent = match_prediction / 20 * 100
141
142             # average national recidivism rate for released prisoners is 43%
143
144             c_record = criminal['record']
145             record_for_this_type = any(item in c_record for item in case_types)
146             recidivism = 90 if record_for_this_type else 30# 90% if he has committed
                ↪  something similar, otherwise 30
147
148             match_percentage = 0.7 * physical_match_percent + 0.15 * recidivism + 0.15 *
                ↪  match_prediction_percent
149             criminal['match'] = round(match_percentage, 2)
150
151         sorted_criminals = sorted(criminals, key=lambda k: k['match'], reverse=True)
152         return render(request, 'case_suspects.html', {'case_id': case_id, 'suspect': suspect,
            ↪  'criminals': sorted_criminals})
153
154
155     def d_tree(request):
156         return render(request, 'decision_tree.html')
157
158
159     # helper function to generate csv file from database table 'criminal' and 'suspect'
160     def generate_csv():
161         # generate csv for criminals
162         df_criminals = pd.DataFrame.from_records(Criminal.objects.all().values())
163         # we have to take care of the record column as it is another table
164         criminals = Criminal.objects.all()
165         output = []
166         # for each criminal
167         for c in criminals:
168             # retrieve the records
169             records = c.c_record.values_list()
170             #print(records)
171             crimes = []
172             # for each record append it as a crime by ID
173             for r in records:
174                 crimes.append(r[0])
175             #print(crimes)
176             # append the crimes in the output list
177             output.append(crimes)
178         #print(output)
179         # add a new column with the records (criminal offenses by ID)
180         df_criminals['c_record'] = output
181         # flattened it if multiple -> WORKS but not helpfuk
182         flattened = pd.DataFrame([(index, value) for (index, values) in
            ↪  df_criminals['c_record'].iteritems() for value in values],
183                                  columns = ['index', 'c_record']).set_index('index')
184         df_criminals = df_criminals.drop('c_record', axis=1).join(flattened)
185         #print(df)
186         # save it as csv
187         df_criminals.to_csv('E:\AC\SE-miniproject-master\csv\criminals.csv', index=False)
188
189         # generate csv for suspects
```

```
    df_suspects = pd.DataFrame.from_records(Suspect.objects.all().values())
    df_suspects.to_csv('E:\AC\SE-miniproject-master\csv\suspects.csv', index=False)


# function to generate a decision tree based on the criminal csv
def generate_decision_tree():
    generate_csv()
    df = pd.read_csv('E:\AC\SE-miniproject-master\csv\criminals.csv')
    # making a decision tree => data must be numerical
    # retrieve only the year of birth
    df['c_dob'] = pd.to_datetime(df['c_dob'])
    df['c_dob'] = df['c_dob'].dt.year
    df['c_age'] = 2020 - df['c_dob']
    # drop the pob as it's not essential
    df.drop('c_pob', axis=1, inplace=True)

    # map() and convert gender to numerical
    d = {'M': 0, 'F': 1}
    df['c_gender'] = df['c_gender'].map(d)

    # map eye colors
    d= {'BLU': 1, 'GRE': 2, 'BR': 3, 'GRA': 4, 'BLA': 5}
    df['c_eye_color'] = df['c_eye_color'].map(d)
    #df = df.fillna(0)

    # map hair color
    d = {'BLO': 1, 'BLA': 2, 'BR': 3, 'GRA': 4, 'GI': 5, 'WH': 6, 'O': 7, 'U': 8}
    df['c_hair_color'] = df['c_hair_color'].map(d)


    # map distinctive marks
    d = {'NONE': 1, 'freckle': 2, 'wrinkle': 3, 'mole': 4, 'dark point': 5, 'U': 6}
    df['c_distinctive_marks'] = df['c_distinctive_marks'].map(d)

    #print(df)
    features = ['c_gender', 'c_age', 'c_height', 'c_weight', 'c_distinctive_marks',
    ↪  'c_eye_color', 'c_hair_color']
    x = df[features]
    y = df['c_record']

    dtree = DecisionTreeClassifier()
    dtree = dtree.fit(x, y)
    data = tree.export_graphviz(dtree, out_file=None, feature_names=features)
    graph = pydotplus.graph_from_dot_data(data)
    graph.write_png('static/criminals_dtree.png')
    return dtree

# function wich predicts in which felony class the given suspect (with his/her
↪  description) fits
# based on the decision tree of the criminals of the known database
def predict_suspect(suspect):
    # decision tree of the criminals
    dtree = generate_decision_tree()

```

```
242     sus = []
243     # this df is the suspect one
244     df = pd.read_csv('E:\AC\SE-miniproject-master\csv\suspects.csv')
245
246
247     # map() and convert gender to numerical
248     d = {'M': 0, 'F': 1}
249     sus.append(d.get(suspect.s_gender))
250     sus.append(suspect.s_age)
251     sus.append(suspect.s_height)
252     sus.append(suspect.s_weight)
253
254     d = {'NONE': 1, 'freckle': 2, 'wrinkle': 3, 'mole': 4, 'dark point': 5, 'U': 6}
255     sus.append(d.get(suspect.s_distinctive_marks))
256
257     d= {'BLU': 1, 'GRE': 2, 'BR': 3, 'GRA': 4, 'BLA': 5}
258     sus.append(d.get(suspect.s_eye_color))
259
260     d = {'BLO': 1, 'BLA': 2, 'BR': 3, 'GRA': 4, 'GI': 5, 'WH': 6, 'O': 7, 'U': 8}
261     sus.append(d.get(suspect.s_hair_color))
262
263     #print(sus)
264     return dtree.predict([[sus[0], sus[1], sus[2], sus[3], sus[4], sus[5], sus[6]]])
265
266
267
```

My focus in the prediction part of the project was developing a decision tree classifier of the criminals from the system. In achieving this I have used special libraries developed with this purpose from the Python including: pandas, scikit-learn, matplotlib and pydotplus. All of them helped in building the decision tree.

A decision tree is similar to a flow-chart diagram, in which internal nodes represent features/attributes and branches represent decision criteria (false/true), while leaves represent outcomes.

In order to create this decision tree we have to first convert our criminal database table to a .csv file, as pandas have special functions to read this file and will create a data frame based on it. Lines 159-191 retrieve the query set of the database and works on it creating a data frame (df) and then saving it as a .cvs. This can also be very helpful for later capabilities.

Next we create the decision tree per se, the function which generates the decision tree is defined on lines 195-234. Basically we read the criminals csv and then we have to remap the columns in the data frame as for creating the decision tree we need only numerical data and strings are not feasible. Then we make a distinction between which **features** and **target**, i.e. where to predict from and what we want to predict. Our features are: the gender, age, height, weight, eye color, hair color, distinctive marks of the criminals. Our target is their record felony. A classification takes place. The **gini** value which is seen in the following representation of a decision tree is the quality of our split, and is always a number between 0.0 and 0.5, where 0.0 would mean all of the samples got the same result, and 0.5 would mean that the split is done exactly in the middle.

In lines 238-264 we have the function which predicts where the suspect defined by its characteristic would fit in our criminal decision tree classification, i.e. what record felony is he most probable to have committed given all its traits.

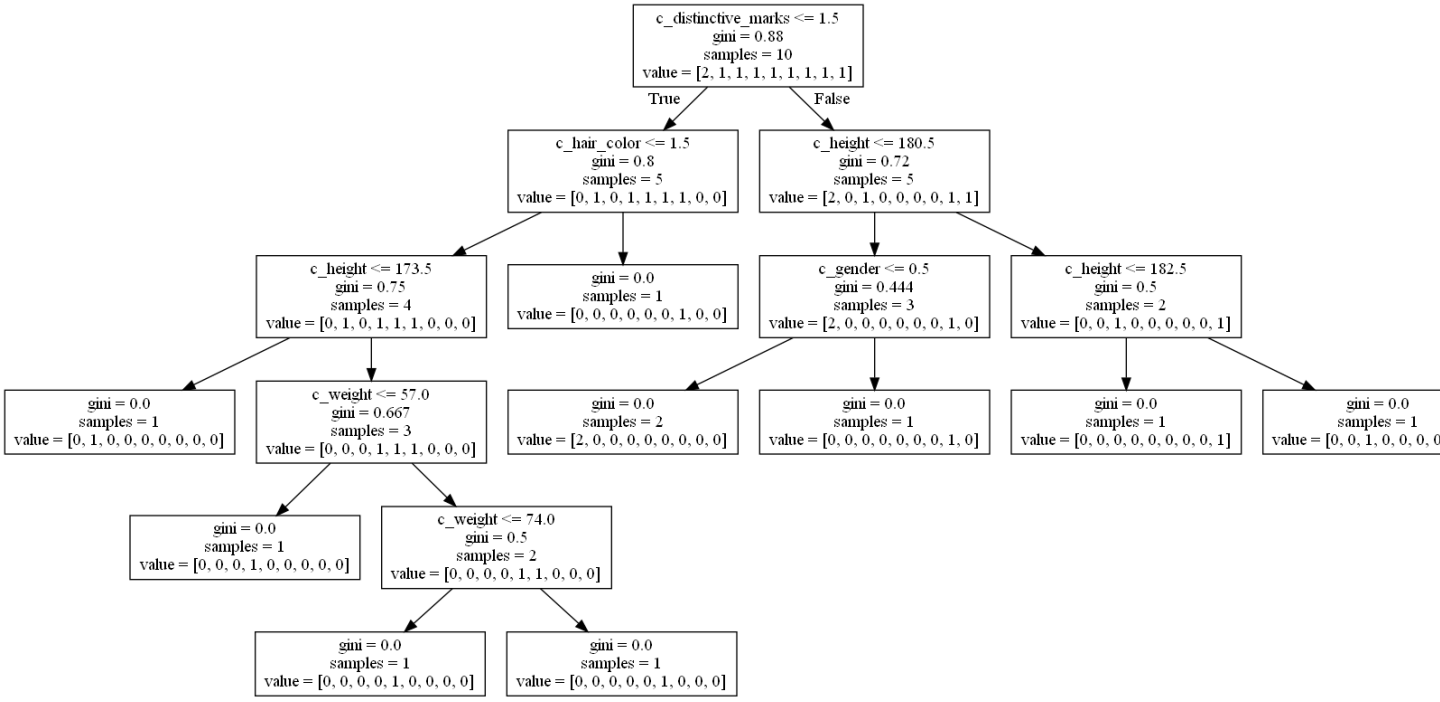
We present below the criminal file (csv) on which I have worked.

criminals.csv

```
id,c_first_name,c_last_name,c_dob,c_pob,c_gender,c_eye_color,c_hair_color,c_height,c_weight,c_distinctiv
1,John,Doe,1985-06-09,"Wayne, New Jersey",M,BLU,BLO,172,86,NONE,3
2,Edith,Stone,1968-08-23,"San Luis, CA",F,BLA,BLO,182,53,NONE,5
3,Stanley,Kornish,1952-03-16,"Saginaw, MI",M,BLA,GI,175,51,dark point,1
4,James,Calhoun,1971-10-13,"East Lansing, MI",M,BR,BLA,179,71,freckle,1
5,Nancy,Stevens,1992-10-13,"Manchester, MO",F,GRA,BLO,175,61,NONE,6
6,Dee,Matter,1980-11-02,"Milwaukee, WI",M,GRE,BR,183,90,wrinkle,4
7,Jocelyn,Lewis,1983-09-18,"Golden Valley, MN",F,GRA,BR,169,51,U,10
8,John,Baltz,1972-10-13,"Salinas, CA",M,BR,BLA,172,69,NONE,9
9,Charles,Webber,1950-07-09,"Sacramento, CA",M,GRA,BLO,182,87,NONE,8
10,Anna,Jackson,1967-10-23,"Johnson City, TN",F,GRA,WH,182,71,mole,11
```

The decision tree resulting from this records is:

Figure 4: Possible Decision Tree



We have to take into consideration the fact that the decision tree is directly influences by the number of actors on which it is made as well as by the number of different features they have. Even if we feed them with the same data they would sometimes give different answers. Because we base it on a probability of an outcome, than the prediction is not 100 percent sure.

We will furthermore present the html templates we have used to display our content. The base one is the backbone of a html document type and allows dynamic content to be added. The others are used for criminals display, decision tree displays, etc. Below is the source code of the main base html file.

base.html

```
1 {% load static %}
2
3 <!DOCTYPE html>
4 <html>
5 <head>
```

```
<style>
body {
  background-image: url({%static 'high.jpg' %});
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: cover;
}
</style>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0,
          ↪  maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    {% block content %}

    {% endblock %}
</body>
</html>
```

5 Conclusions

5.1 Observations and further development

The system is very raw in its capabilities and appearance. It can be developed more by making a better frontend, by making special pages to add the data of criminals, suspects, victims, cases.

In addition, the prediction algorithm can be further increased, by experimenting more with machine learning and decision trees. This was nothing but a trying example. Other functionalities such as gps and area observation of crime density based on categories can be implemented with the help of decision trees, as they provide classifications.

Django and Python can further develop more the application from routes to security and better flexibility.

6 Bibliography

http://www.jetir.org/papers/JETIR2003381.pdfhttp://www.jetir.org/papers
https://ijesc.org/upload/a34318960524a381b0a311d882104ae8.Criminal%20Investigation%20Tracker%20witl
https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052https://towardsdatascience
trees
https://www.victimsupport.org.uk/crime-info/types-crimehttps://www.victimsupport.org.uk/crime-info/types-crime
https://en.wikipedia.org/wiki/Recidivismhttps://en.wikipedia.org/wiki/Recidivism
https://www.w3schools.com/python/python_ml_decision_tree.asphttps://www.w3schools.com/python/py
https://www.predpol.com/how-predictive-policing-works/https://www.predpol.com/how-predictive-policing-works/
https://www.rand.org/content/dam/rand/pubs/research_briefs/RB9700/RB9735/RAND_RB9735.pdfhttp
https://www.ncjrs.gov/pdffiles1/nij/230414.pdfhttps://www.ncjrs.gov/pdffiles1/nij/230414.pdf