Intro to database Systems
Final project
Report
Team 02

Traffic accidents in Taipei

- Main idea
    - The purpose of your application

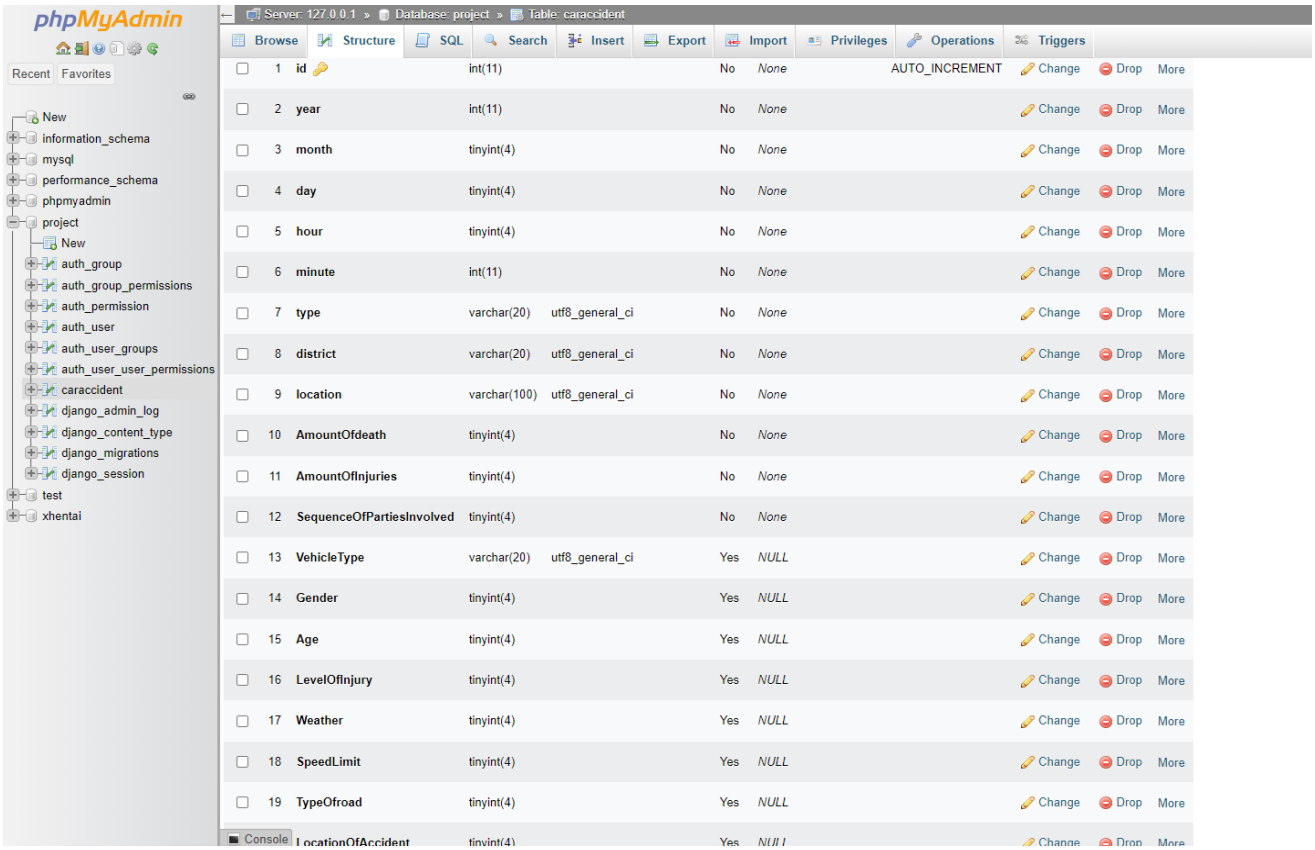
Our application seeks to help novice drivers navigate the roads safely by offering useful information about probable accident scenarios. Because car accidents are common, it can be difficult for inexperienced drivers to anticipate the types of situations they may face. Users receive access to a large database by using our program, which allows them to search for certain conditions and locations that are statistically more prone to accidents. For example, if a user is aware of a specific street or road where a high number of accidents have happened, they can drive with additional caution in that region. Drivers can be better prepared and aware by understanding the main elements that contribute to accidents, thereby promoting safer driving practices and lowering the possibility of accidents. Our app is a vital tool for educating and empowering novice drivers, allowing them to make more informed decisions and traverse roads with better confidence and awareness.

The web application will allow the user to access ID, year, month , day hour, the type of road, district , the location , amount of death , amount of injuries,and sequence of parties involved when the accident occurred. Other irrelevant data like the age and gender of the involved people would remain inaccessible. The user will be able to sort the data in any category such as death by gender, amount of accident per month, percentage of weather conditions under which accident happened , level of injuries for male, level of injuries for female . The administrator can create new records in the database, and might be able to add new attributes to the tables in the database.

- Data

The data we are using in the final project is the detail of traffic accidents in Taipei. The data we get from the Taipei City Police Department Traffic Division documents the location of the accidents, the occurring date, and the info of involved people. Each table represents a specific year of the data and the column means, the year of the accident, the month of the accident, the hour of the accident, the minute of the accident, how severe the accident was(1 meaning death in 24 hours, 2 meaning death after 24 hours or injured, 3 meaning no injury nor death), the area where the accident occurred, the exact location of the accident, the amount of death, the amount of injured, vehicle type, gender of involved people, level of injury, state of the weather, the speed limit, the type of road, and which part of the road did the accident occurred respectively. The data will also be updated once every year.

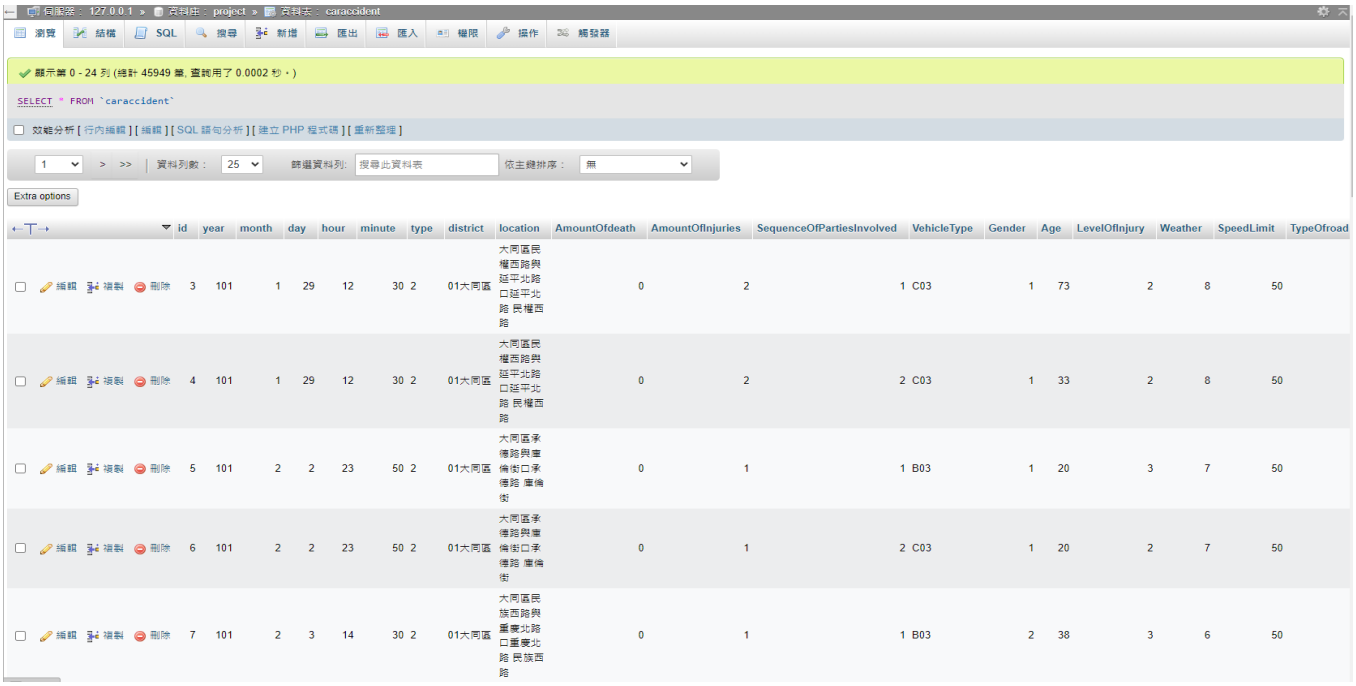
Below is the screenshot of the columns in table “caraccident”.



The "caraccident" table above is designed to store information related to traffic accidents. Here is a description of the table in plain words:
The "caraccident" table has the following columns:
"id": An auto-incremented unique identifier for each accident.
"year": The year in which the accident occurred.
"month": The month (represented as a number) in which the accident occurred.
"day": The day (represented as a number) on which the accident occurred.
"hour": The hour (represented as a number) at which the accident occurred.
"minute": The minute (represented as a number) at which the accident occurred.
"type": The type of accident, described using up to 20 characters.
"district": The district or area where the accident occurred, described using up to 20 characters.
"location": The specific location or address of the accident, described using up to 100 characters.
"AmountOfdeath": The number of deaths resulting from the accident.
"AmountOfInjuries": The number of injuries resulting from the accident.
"SequenceOfPartiesInvolved": A numeric representation indicating the sequence of parties involved in the accident.
"VehicleType": The type of vehicle involved in the accident, described using up to 20 characters. (This column allows NULL values, meaning it can be empty or unspecified.)
"Gender": The gender of the individuals involved in the accident. (This column allows NULL values.)
"Age": The age of the individuals involved in the accident. (This column allows NULL values.)
"LevelOfInjury": A numeric representation indicating the severity or level of injury sustained in the accident. (This column allows NULL values.)

"Weather": A numeric representation indicating the weather conditions during the accident. (This column allows NULL values.)
"SpeedLimit": The speed limit in effect at the accident location. (This column allows NULL values.)
"TypeOfroad": A numeric representation indicating the type of road where the accident occurred. (This column allows NULL values.)
"LocationOfAccident": A numeric representation indicating the specific location or direction of the accident. (This column allows NULL values.)

Below is a part of table contents:



伺服器： 127.0.0.1 » 資料庫： project » 資料表： caraccident

瀏覽 | 結構 | SQL | 搜尋 | 新增 | 匯出 | 匯入 | 權限 | 操作 | 觸發器

顯示第 0 - 24 列 (總計 45949 筆, 查詢用了 0.0002 秒。)

```
SELECT * FROM `caraccident`
```

效能分析 [行內編輯] [編輯] [SQL 語句分析] [建立 PHP 程式碼] [重新整理]

1 > >> | 資料列數： 25 篩選資料列: 搜尋此資料表 依主鍵排序： 無

Extra options

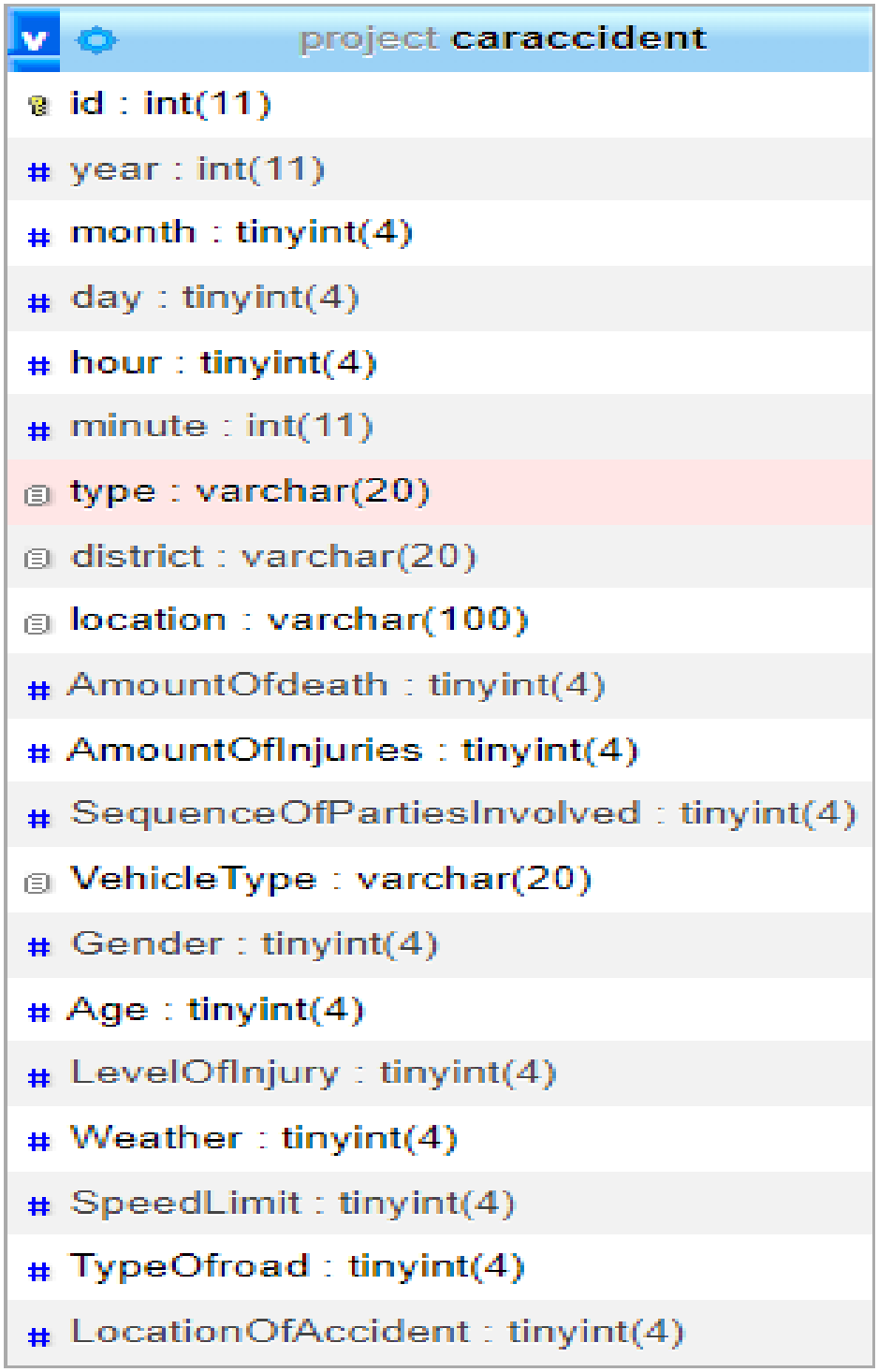
	id	year	month	day	hour	minute	type	district	location	AmountOfdeath	AmountOfInjuries	SequenceOfPartiesInvolved	VehicleType	Gender	Age	LevelOfInjury	Weather	SpeedLimit	TypeOfroad
編輯 複製 刪除	3	101	1	29	12	30	2	01大同區	大同區民權西路與延平北路口延平北路民權西路	0	2	1	C03	1	73	2	8	50	
編輯 複製 刪除	4	101	1	29	12	30	2	01大同區	大同區民權西路與延平北路口延平北路民權西路	0	2	2	C03	1	33	2	8	50	
編輯 複製 刪除	5	101	2	2	23	50	2	01大同區	大同區承德路與重倫街口承德路重倫街	0	1	1	B03	1	20	3	7	50	
編輯 複製 刪除	6	101	2	2	23	50	2	01大同區	大同區承德路與重倫街口承德路重倫街	0	1	2	C03	1	20	2	7	50	
編輯 複製 刪除	7	101	2	3	14	30	2	01大同區	大同區民族西路與重慶北路口重慶北路民族西路	0	1	1	B03	2	38	3	6	50	

Table “caraccident” above contains more than 45000 lines, I provide a part of it.

Source: open data platform of the Taiwan government
data.gov.tw/dataset/130110

Draw an ER model of your schema.

ER MODEL



The "caraccident" table is represented in the ER as an entity (rectangle) with each column serving as one of its attributes. The "id" column serves as the table's primary key and identifies each accident report uniquely.
The relationships between entities and the table's structure are depicted by the ER model, but not those between the columns themselves.

- Database
    - What database do you use (MySQL, SQLite, mongoDB, etc.)

    We use MySQL because we are familiar with it and MySQL can efficiently handle huge amounts of data and supports several different data types, including text, date/time, numeric, and more. It has attributes including transactions, indexing, data integrity requirements, and user administration.

    - Describe how you connect your database to your application as detail as possible

Django is a Python-based web framework that enables the rapid development of efficient web applications. It is also known as the "batteries included framework" because Django includes built-in features for everything, such as the Django Admin Interface, the default database - SQLlite3, and so on.

Installation

Set up the Mysql database. After downloading, run the setup and configure the admin and password.
https://dev.mysql.com/downloads/installer/

Install Django

pip install django

Then install another library to use the MySQL database:
pip install mysqlclient

Steps to connect MySQL to Django

Step 1: Create a new project:
django-admin startproject NYCU-Database-poject

Step 2: Move to the MyDB folder.
cd NYCU-Database-project

Step 3: Create a MySql database.

Step 4: Update the settings.py
Open settings.py here inside the DATABASES variable configure MySQL database values, and add values of your database.

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "project",
        "USER": "appuser",
        "PASSWORD": "",
        "HOST": "localhost",
        "PORT": "3306",
    }
}
```

First, we have replaced the 'django.db.backends.sqlite3' to 'django.db.backends.mysql'. This is basically indicating we shift SQLite to MySQL database.

NAME: It indicates the name of the database we want to connect.
USER: The MYSQL username is the one who has access to the database and manages it.
PASSWORD: It is the password of the database.
HOST: It is indicated by "127.0.0.1" and "PORT" "3306" that the MySQL database is accessible at hostname "0.0.1" and on port "3306."

Step 5: Run the server.
python manage.py runserver

Step 6: Run the migration command
python manage.py makemigrations

What if someone does something unexpected?
User Feedback: When something unexpected occurs, provide clear and informative error messages to the user. Avoid exposing sensitive information but provide enough details to help the user understand what went wrong and how to proceed.

- How do you maintain your database (update data, add new data etc.)

This code below (screenshot)represents a collection of functions/views that handle different aspects of a web application related to car accidents. Each function is described below:

emp(request): This function handles the creation of a new car accident record. If the request method is POST, it validates the form data and saves it to the database using a "CaraccidentForm". If the form is valid and the data is saved successfully, it redirects to the /show page. Otherwise, it renders the index.html template with the form.

show(request, page=1): This function retrieves all car accidents from the database using Caraccident.objects.all(). It then performs pagination by slicing the results based on the specified page number (page). The function renders the show.html template, passing the paginated car accidents along with information about the current page, previous page, next page, and the minimum and maximum page numbers.

edit(request, id): This function retrieves a specific car accident record based on the provided id from the URL parameter. It renders the edit.html template, passing the retrieved car accident as context.

update(request, id): This function handles the updating of an existing car accident record. It retrieves the specific record based on the provided id and populates a form (CaraccidentForm) with the existing data. If the form data is valid, it saves the changes to the database and redirects to the /show page. Otherwise, it renders the edit.html template with the form and the car accident object.

destroy(request, id): This function handles the deletion of a car accident record. It retrieves the specific record based on the provided id and deletes it from the database. It then redirects to the /show page.

index(request): This function serves as the main view for the home page of the application. It performs various data processing and visualization using the Pandas and Plotly libraries. It creates multiple plots (line chart, pie charts) based on data obtained from different functions (AmountAccidentbymonth(), GetDeathByGender(), Weather(), GetLevelofinjurybygender()) and generates their respective HTML representations using the Plotly's plot function. Finally, it renders the home.html template, passing the generated plot HTML representations as context.

index(request): This function serves as the main view for the home page of the application. It performs various data processing and visualization using the Pandas and Plotly libraries. It creates multiple plots (line chart, pie charts) based on data obtained from different functions (AmountAccidentbymonth(), GetDeathByGender(), Weather(), GetLevelofinjurybygender()) and generates their respective HTML representations using the Plotly's plot function. Finally, it renders the home.html template, passing the generated plot HTML representations as context.

database_test(request): This function is a test endpoint that returns JSON data. It can be used to test and retrieve specific data from the database. Currently, it returns JSON data representing the values of male injury counts obtained from the GetLevelofinjurybygender() function.

Overall, these functions are responsible for various parts of the web application, such as creating, reading, updating, and deleting car accident records, as well as generating visualizations and giving a test endpoint the ability to retrieve specific data.
In the provided code, there are a few instances where exception handling is used:

```
# Create your views here.
def emp(request):
    if request.method == "POST":
        form =  CaraccidentForm(request.POST)
        if form.is_valid():
            try:
                form.save()
                return redirect('/show')
            except:
                pass
    else:
        form =  CaraccidentForm()
    return render(request,'index.html',{'form':form})
def show(request, page=1):
    caraccidents = Caraccident.objects.all()
    row = 100
    min = 1
    max = math.ceil(len(caraccidents)/row)
    return render(request,"show.html",{'caraccidents':caraccidents[(page-1)*row:page*row],
                                       'page': page,
                                       'prev': page - 1 if page > min else min,
                                       'next': page + 1 if page < max else max,
                                       'min': min, 'max': max})
def edit(request, id):
    caraccident = Caraccident.objects.get(id=id)
    return render(request,'edit.html', {'caraccident':caraccident})
def update(request, id):
    caraccident = Caraccident.objects.get(id=id)
    form =  CaraccidentForm(request.POST, instance = caraccident)
    if form.is_valid():
        form.save()
        return redirect("/show")
    return render(request, 'edit.html', {'caraccident': caraccident})
```

```python
def destroy(request, id):
    caraccident = Caraccident.objects.get(id=id)
    caraccident.delete()
    return redirect("/show")

def index(request):
    df = pd.DataFrame(list(zip(list(AmountAccidentbymonth().keys()),list(AmountAccidentbymonth().values()))), columns=['month', 'count'])
    print(df)
    fig = px.line(df, y='count', x='month', title='Amount of accident per month (2012)')
    plot_div = plot(fig, output_type='div')

    x = ['rainy', 'sunny', 'snow']
    y = [52, 104, 33]
    df = pd.DataFrame(list(zip(list(GetDeathByGender().keys()),list(GetDeathByGender().values()))), columns=['gender', 'count'])
    print(df)
    fig = px.pie(df, values='count', names='gender', title='Death by gender(2012)')
    plot2_div = plot(fig, output_type='div')

    df = pd.DataFrame(list(zip(list(Weather().keys()),list(Weather().values()))), columns=['weather', 'count'])
    print(df)
    fig = px.pie(df, values='count', names='weather', title='Percentage of weather conditions under which accidents happened(2022)')
    plot3_div = plot(fig, output_type='div')

    df = pd.DataFrame(list(zip(list(GetLevelofinjurybygender()['male'].keys()),list(GetLevelofinjurybygender()['male'].values()))), columns=[
    print(df)
    fig = px.pie(df, values='count', names='male', title='Level of injury for male (2012)')
    plot4_div = plot(fig, output_type='div')

    df = pd.DataFrame(list(zip(list(GetLevelofinjurybygender()['female'].keys()),list(GetLevelofinjurybygender()['female'].values()))), colum
    print(df)
    fig = px.pie(df, values='count', names='female', title='Level of injury for female (2012)')
    plot5_div = plot(fig, output_type='div')
    return render(request, "home.html", context={'plot_div': plot_div,'a_div': plot2_div,'b_div': plot3_div , 'c_div': plot4_div, 'd_div': pl

def database_test(request):
```

- Application
  Create model:
  Put the following code into models.py file.
  models.py//

```python
> carAccident > models.py > GetLevelofinjurybygender
from django.db import models
from django.db.models import Avg, Count, Min, Sum, Q

class Caraccident(models.Model):
    id = models.AutoField(primary_key=True)
    year = models.IntegerField()
    month = models.IntegerField()
    day = models.IntegerField()
    hour = models.IntegerField()
    minute = models.IntegerField()
    type = models.CharField(max_length=20)
    district = models.CharField(max_length=20)
    location = models.CharField(max_length=100)
    amountofdeath = models.IntegerField(db_column='AmountOfdeath')  # Field name made lowercase.
    amountofinjuries = models.IntegerField(db_column='AmountOfInjuries')  # Field name made lowercase.
    sequenceofpartiesinvolved = models.IntegerField(db_column='SequenceOfPartiesInvolved')  # Field name made lowercase.
    vehicletype = models.CharField(db_column='VehicleType', max_length=20, blank=True, null=True)  # Field name made lowercase.
    gender = models.IntegerField(db_column='Gender', blank=True, null=True)  # Field name made lowercase.
    age = models.IntegerField(db_column='Age', blank=True, null=True)  # Field name made lowercase.
    levelofinjury = models.IntegerField(db_column='LevelOfInjury', blank=True, null=True)  # Field name made lowercase.
    weather = models.IntegerField(db_column='Weather', blank=True, null=True)  # Field name made lowercase.
    speedlimit = models.IntegerField(db_column='SpeedLimit', blank=True, null=True)  # Field name made lowercase.
    typeofroad = models.IntegerField(db_column='TypeOfroad', blank=True, null=True)  # Field name made lowercase.
    locationofaccident = models.IntegerField(db_column='LocationOfAccident', blank=True, null=True)  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'caraccident'

def GetLevelofinjurybygender():
    result1 = (Caraccident.objects
        .filter(gender__exact=1)
        .filter(levelofinjury__isnull=False)
        .values('levelofinjury')
        .annotate(count=Count('levelofinjury'))
        .order_by('levelofinjury')
```

```
        .order_by('levelofinjury')
    )
    result2 = (Caraccident.objects
        .filter(gender__exact=2)
        .filter(levelofinjury__isnull=False)
        .values('levelofinjury')
        .annotate(count=Count('levelofinjury'))
        .order_by('levelofinjury')
    )
    data1 = dict()
    data2 = dict()
    for row in result1:
        data1[row['levelofinjury']] = row['count']
    for row in result2:
        data2[row['levelofinjury']] = row['count']
    data = dict()
    data = {
        'male': data1,
        'female': data2
    }
    return data

def AmountAccidentbymonth():
    result = (Caraccident.objects
        .filter(sequenceofpartiesinvolved__exact=1)
        .values('month')
        .annotate(deadcount=Count('sequenceofpartiesinvolved'))
        .order_by('month')
    )
    data = dict()
    for row in result:
        data[row['month']] = row['deadcount']
    return data
```

```
    return data


def GetDeathByGender():
    result = (Caraccident.objects
        .filter(Q(levelofinjury=1) | Q(levelofinjury=5))
        .values('gender')
        .annotate(deadcount=Count('levelofinjury'))
        .order_by('gender')
    )
    data = dict()
    for row in result:
        if row['gender'] == 1:
            data['male'] = row['deadcount']
        elif row['gender'] == 2:
            data['female'] = row['deadcount']
    return data

def Weather():
    map = {
        1: "heavy rain",
        2: "strong wind",
        3: "wind and sand",
        4: "fog or smoke",
        5: "snowy",
        6: "rainy",
        7: "cloudy",
        8: "sunny"
    }
result = (Caraccident.objects
    .filter(sequenceofpartiesinvolved__exact=1)
    .filter(weather__isnull=False)
    .values('weather')
    .annotate(count=Count('weather'))
    .order_by('weather')
)
data = dict()
for row in result:
    data[map[row['weather']]] = row['count']
return data
```

Create a ModelForm:

form.py

```
project > carAccident > forms.py > CaraccidentForm
1   from django import forms
2   from .models import *
3   class CaraccidentForm(forms.ModelForm):
4       class Meta:
5           model = Caraccident
6           fields = "__all__"
```

Create View Functions:

views.py//

```
> carAccident > views.py > show
from django.shortcuts import render, redirect
from plotly.offline import plot
from plotly.graph_objs import Scatter

# Create your views here.
from django.http import HttpResponse
import plotly.express as px

from .models import *
import json
import math

import pandas as pd
from .forms import CaraccidentForm


# Create your views here.
def emp(request):
    if request.method == "POST":
        form =  CaraccidentForm(request.POST)
        if form.is_valid():
            try:
                form.save()
                return redirect('/show')
            except:
                pass
    else:
        form =  CaraccidentForm()
    return render(request,'index.html',{'form':form})
def show(request, page=1):
    caraccidents = Caraccident.objects.all()
    row = 50
    min = 1
    max = math.ceil(len(caraccidents)/row)
    return render(request,"show.html",{'caraccidents':caraccidents[(page-1)*row:page*row],
                                       'page': page,
                                       'prev': page - 1 if page > min else min,
                                       'next': page + 1 if page < max else max,
                                       'min': min, 'max': max})
def edit(request, id):
    caraccident = Caraccident.objects.get(id=id)
    return render(request,'edit.html', {'caraccident':caraccident})
def update(request, id):
    caraccident = Caraccident.objects.get(id=id)
    form =  CaraccidentForm(request.POST, instance = caraccident)
    if form.is_valid():
        form.save()
        return redirect("/show")
    return render(request, 'edit.html', {'caraccident': caraccident})
def destroy(request, id):
    caraccident = Caraccident.objects.get(id=id)
    caraccident.delete()
    return redirect("/show")

def index(request):
    df = pd.DataFrame(list(zip(list(AmountAccidentbymonth().keys()),list(AmountAccidentbymonth().values()))), columns=['month', 'count'])
    print(df)
    fig = px.line(df, y='count', x='month', title='Amount of accident per month (2012)')
    plot_div = plot(fig, output_type='div')
```

```
    df = pd.DataFrame(list(zip(list(GetDeathByGender().keys()),list(GetDeathByGender().values()))), columns=['gender', 'count'])
    print(df)
    fig = px.pie(df, values='count', names='gender', title='Death by gender(2012)')
    plot2_div = plot(fig, output_type='div')

    df = pd.DataFrame(list(zip(list(Weather().keys()),list(Weather().values()))), columns=['weather', 'count'])
    print(df)
    fig = px.pie(df, values='count', names='weather', title='Percentage of weather conditions under which accidents happened(2012)')
    plot3_div = plot(fig, output_type='div')

    df = pd.DataFrame(list(zip(list(GetLevelofinjurybygender()['male'].keys()),list(GetLevelofinjurybygender()['male'].values()))), columns=[
    print(df)
    fig = px.pie(df, values='count', names='male', title='Level of injury for male (2012)')
    plot4_div = plot(fig, output_type='div')

    df = pd.DataFrame(list(zip(list(GetLevelofinjurybygender()['female'].keys()),list(GetLevelofinjurybygender()['female'].values()))), colum
    print(df)
    fig = px.pie(df, values='count', names='female', title='Level of injury for female (2012)')
    plot5_div = plot(fig, output_type='div')
    return render(request, "home.html", context={'plot_div': plot_div,'a_div': plot2_div,'b_div': plot3_div , 'c_div': plot4_div, 'd_div': pl

def database_test(request):
    # return HttpResponse(json.dumps(AmountAccidentbymonth()))
    return HttpResponse(json.dumps(list(GetLevelofinjurybygender()['male'].values())))
```

Provide Routing:

Provide URL patterns to map with views function.

urls.py//

```
> carAccident > urls.py > ...
from django.urls import path
from django.contrib import admin

from . import views

urlpatterns = [
    path("", views.index, name="index"),
    path("database", views.database_test, name="database"),
    path('admin/', admin.site.urls),
    path('emp', views.emp),
    path('show',views.show),
    path('show/<int:page>',views.show),
    path('edit/<int:id>', views.edit),
    path('update/<int:id>', views.update),
    path('delete/<int:id>', views.destroy),
]
```

Organize Templates:

Create a templates folder inside the caraccident app and create three (index, edit, show) html files inside the directory. The code for each is given below.

index.html

```
> carAccident > <> index.html > html > body > form.post-form > div.container > div.form-group.row
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
    {% load static %}
    <link rel="stylesheet" href="{% static 'style.css' %}"/>
</head>
<body>
<form method="POST" class="post-form" action="/emp">
        {% csrf_token %}
    <div class="container">
<br>
    <div class="form-group row">
    <label class="col-sm-1 col-form-label"></label>
    <div class="col-sm-4">
    <h3>Enter Details</h3>
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Year:</label>
    <div class="col-sm-4">
      {{ form.year }}
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label">Month:</label>
    <div class="col-sm-4">
      {{ form.month }}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Day:</label>
    <div class="col-sm-4">
      {{ form.day }}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Hour:</label>
    <div class="col-sm-4">
      {{ form.hour }}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Minute:</label>
    <div class="col-sm-4">
      {{ form.minute }}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Type:</label>
    <div class="col-sm-4">
      {{ form.type }}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">District:</label>
    <div class="col-sm-4">
      {{ form.district }}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Location:</label>
    <div class="col-sm-4">
      {{ form.location }}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Amount of death:</label>
    <div class="col-sm-4">
      {{ form.amountofdeath}}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Amount of injuries:</label>
    <div class="col-sm-4">
      {{ form.amountofinjuries}}
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Sequence of parties involved:</label>
    <div class="col-sm-4">
      {{ form.sequenceofpartiesinvolved}}
    </div>
  </div>

    <div class="form-group row">
    <label class="col-sm-1 col-form-label"></label>
    <div class="col-sm-4">
    <button type="submit" class="btn btn-primary">Submit</button>
    </div>
  </div>
    </div>
</form>
</body>
</html>
```

//show.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Accident Records</title>
    {% load static %}
    <link rel="stylesheet" href="{% static 'style.css' %}"/>
</head>
<body>
<table class="table table-striped table-bordered table-sm">
    <thead class="thead-dark">
    <tr>
        <th>ID</th>
        <th>Year</th>
        <th>Month</th>
        <th>Day</th>
        <th>Hour</th>
        <th>Type</th>
        <th>District</th>
        <th>Location</th>
        <th>Amount of Death</th>
        <th>Amount of injuries</th>
        <th>Sequence of parties involved</th>

    </tr>
    </thead>
    <tbody>
{% for caraccident in caraccidents %}
    <tr>
        <td>{{ caraccident.id }}</td>
        <td>{{ caraccident.year }}</td>
        <td>{{ caraccident.month }}</td>
        <td>{{ caraccident.day }}</td>
        <td>{{ caraccident.hour }}</td>
        <td>{{ caraccident.type }}</td>
        <td>{{ caraccident.district }}</td>
        <td>{{ caraccident.location }}</td>
        <td>{{ caraccident.amountofdeath }}</td>
        <td>{{ caraccident.amountofinjuries }}</td>
        <td>{{ caraccident.sequenceofpartiesinvolved }}</td>
        <td>
            <a href="/edit/{{ caraccident.id }}"><span class="glyphicon glyphicon-pencil" >Edit</span></a>
            <a href="/delete/{{ caraccident.id }}">Delete</a>
        </td>
    </tr>
{% endfor %}
    </tbody>
</table>
<br>
<br>
<center>
    <a href="/show/{{ min }}">&lt;&lt;</a>
    <a href="/show/{{ prev }}">&lt;</a>
    {{ page }}
    <a href="/show/{{ next }}">&gt;</a>
    <a href="/show/{{ max }}">&gt;&gt;</a>
</center>
<center><a href="/emp" class="btn btn-primary">Add New Record</a>
</center>
</body>
```

```
55          <a href="/show/{{ next }}">&gt;</a>
56          <a href="/show/{{ max }}">&gt;&gt;</a>
57      </center>
58      <center><a href="/emp" class="btn btn-primary">Add New Record</a>
59      </center>
60      </body>
61      </html>
```

// edit.html

```
> carAccident > <> edit.html > html > body > form.post-form > div.container > div.form-group.row > div.col-sm-4 > input#id_sequenceofparties
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
    {% load static %}
    <link rel="stylesheet" href="{% static 'style.css' %}"/>
</head>
<body>
<form method="POST" class="post-form" action="/update/{{caraccident.id}}">
        {% csrf_token %}
    <div class="container">
<br>
    <div class="form-group row">
    <label class="col-sm-1 col-form-label"></label>
    <div class="col-sm-4">
    <h3>Update Details</h3>
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Year</label>
    <div class="col-sm-4">
        <input type="text" name="year" id="id_year" required maxlength="20" value="{{ caraccident.year}}"/>
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label">Month:</label>
    <div class="col-sm-4">
        <input type="text" name="month" id="id_month" required maxlength="20" value="{{ caraccident.month }}" />
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Day:</label>
    <div class="col-sm-4">
        <input type="text" name="day" id="id_day" required maxlength="20" value="{{ caraccident.day }}" />
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Hour:</label>
    <div class="col-sm-4">
        <input type="text" name="hour" id="id_hour" required maxlength="20" value="{{ caraccident.hour }}" />
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Minute:</label>
    <div class="col-sm-4">
        <input type="text" name="minute" id="id_minute" required maxlength="20" value="{{ caraccident.minute }}" />
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Type:</label>
    <div class="col-sm-4">
        <input type="text" name="type" id="id_type" required maxlength="100" value="{{ caraccident.type }}" />
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label">District:</label>
    <div class="col-sm-4">
        <input type="text" name="district" id="id_district" required maxlength="250" value="{{ caraccident.district }}" />
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Location:</label>
    <div class="col-sm-4">
        <input type="text" name="location" id="id_location" required maxlength="254" value="{{ caraccident.location}}" />
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-2 col-form-label">Amount of death:</label>
    <div class="col-sm-4">
        <input type="text" name="amountofdeath" id="id_amountofdeath" required maxlength="20" value="{{ caraccident.amountofdeath }}" />
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label">Amount of injuries:</label>
    <div class="col-sm-4">
        <input type="text" name="amountofinjuries" id="id_amountofinjuries" required maxlength="20" value="{{ caraccident.amountofinjuries}}"
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label">Sequence of parties involved:</label>
    <div class="col-sm-4">
        <input type="text" name="sequenceofpartiesinvolved" id="id_sequenceofpartiesinvolved" required maxlength="20" value="{{ caraccident.s
    </div>
  </div>
    <div class="form-group row">
    <label class="col-sm-1 col-form-label"></label>
    <div class="col-sm-4">
    <button type="submit" class="btn btn-success">Update</button>
    </div>
        </div>
          </div>
      </form>
      </body>
      </html>
```

Static Files Handling:
//style.css

```
# style.css
project > carAccident > static > # style.css > body
1   body {font:12px/1.4 Verdana,Arial; background:#eee; height:100%; margin:25px 0; padding:0}
2   h1 {font:24px Georgia,Verdana; margin:0}
3   h2 {font-size:12px; font-weight:normal; font-style:italic; margin:0 0 20px}
4   p {margin-top:0}
5   ul {margin:0; padding-left:20px}
6
7   #testdiv {width:600px; margin:0 auto; border:1px solid #ccc; padding:20px 25px; background:#fff}
8
9   #tinybox {position:absolute; display:none; padding:10px; background:#fff url(images/preload.gif) no-repeat 50% 50%; border:10px solid #e3
10  #tinymask {position:absolute; display:none; top:0; left:0; height:100%; width:100%; background:#000; z-index:1500}
11  #tinycontent {background:#fff}
12
13  .button {font:14px Georgia,Verdana; margin-bottom:10px; padding:8px 10px 9px; border:1px solid #ccc; background:#eee; cursor:pointer}
14  .button:hover {border:1px solid #bbb; background:#e3e3e3}
```

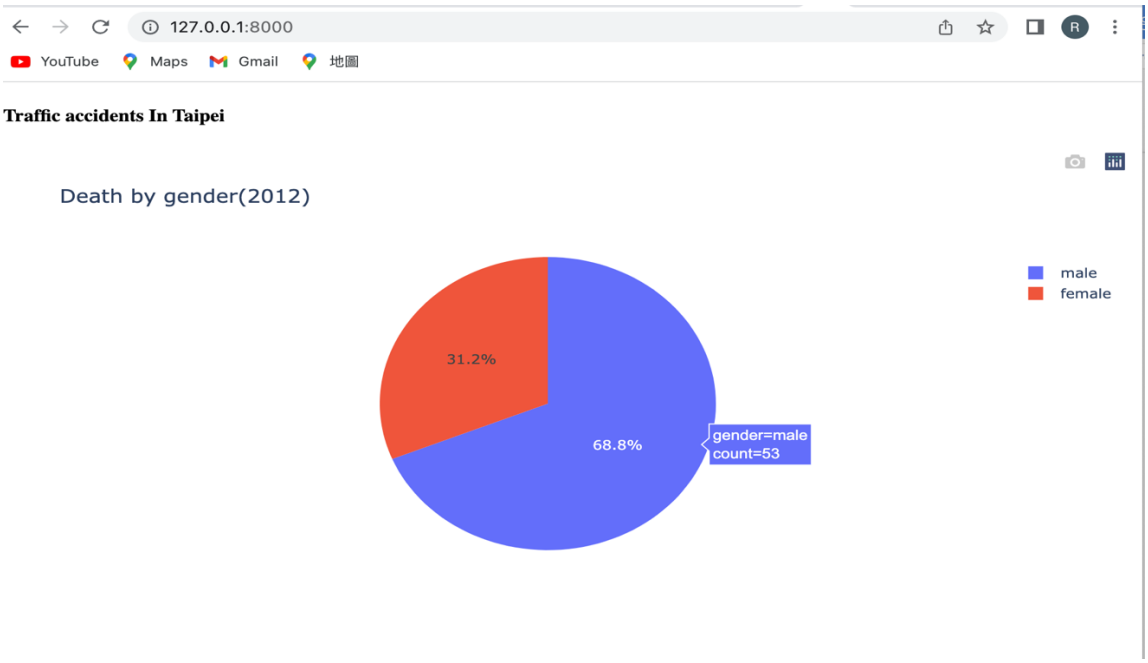
To run server use the following command: python manage.py runserver

```
PROBLEMS 12   OUTPUT   DEBUG CONSOLE   TERMINAL
(base) ralphkedywillensbuteau@Ralphs-MacBook-Pro project % python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified some issues:

WARNINGS:
?: (urls.W005) URL namespace 'admin' isn't unique. You may not be able to reverse all URLs in this namespace

System check identified 1 issue (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 15, 2023 - 23:29:23
Django version 4.2.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
Ln 78, Col 51  Spaces: 4  UTF-8  LF  Python
```

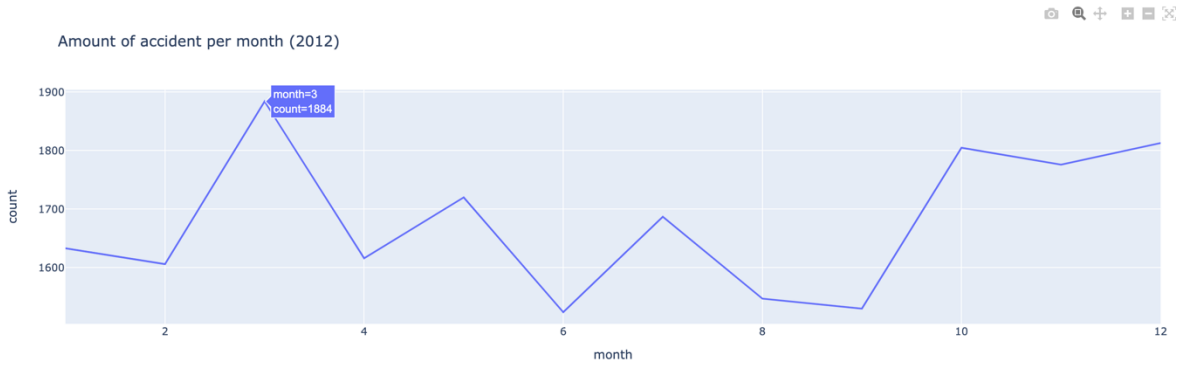
Access the browser:

Access the application by entering http://127.0.0.1:8000/ . it will show the interface about the traffic accidents in Taipei.

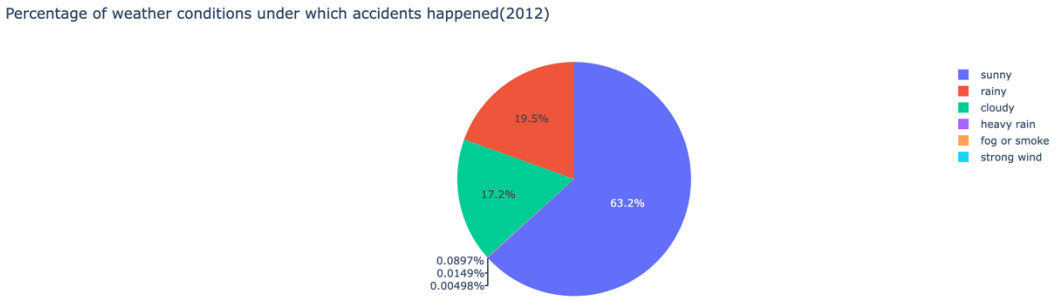
Below it shows the percentage and count of death by gender in 2012.



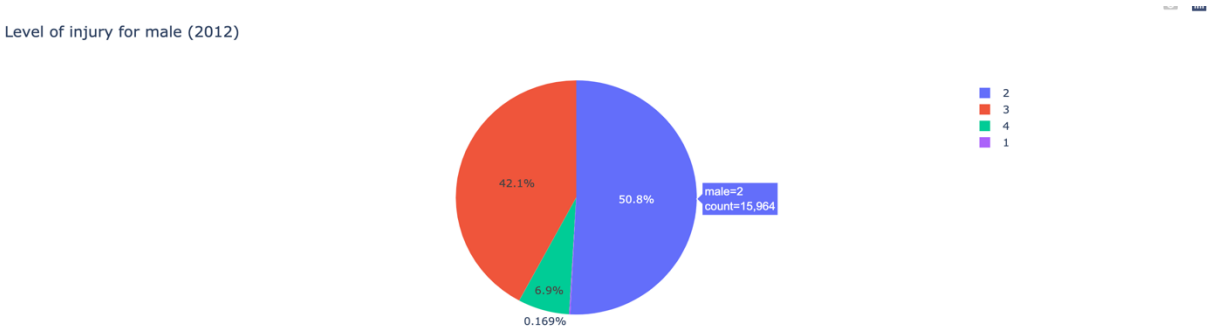
In the screenshot below, it shows the amount of accident per month in 2012



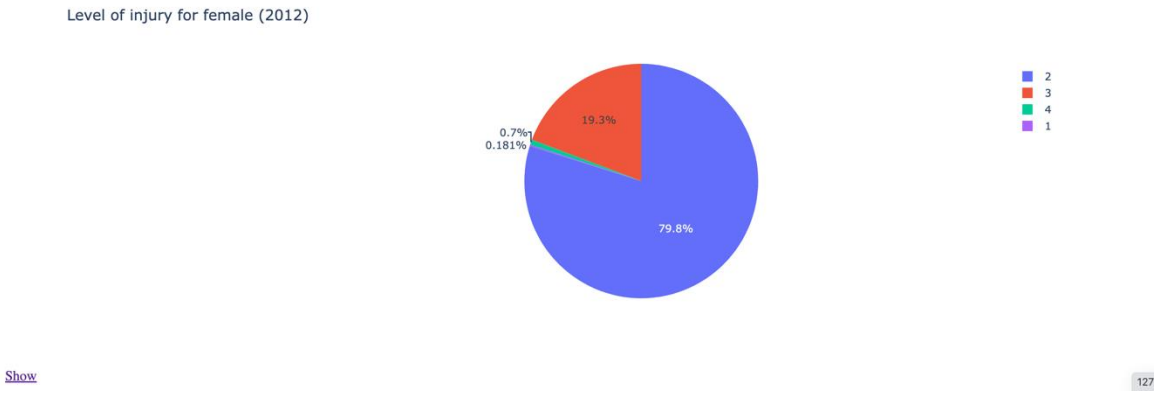
In the screenshot below, it shows the percentage of weather under which accidents happened.



In the screenshot below, it shows the level of injury for male.



In the screenshot below, it shows the level of injury for female.



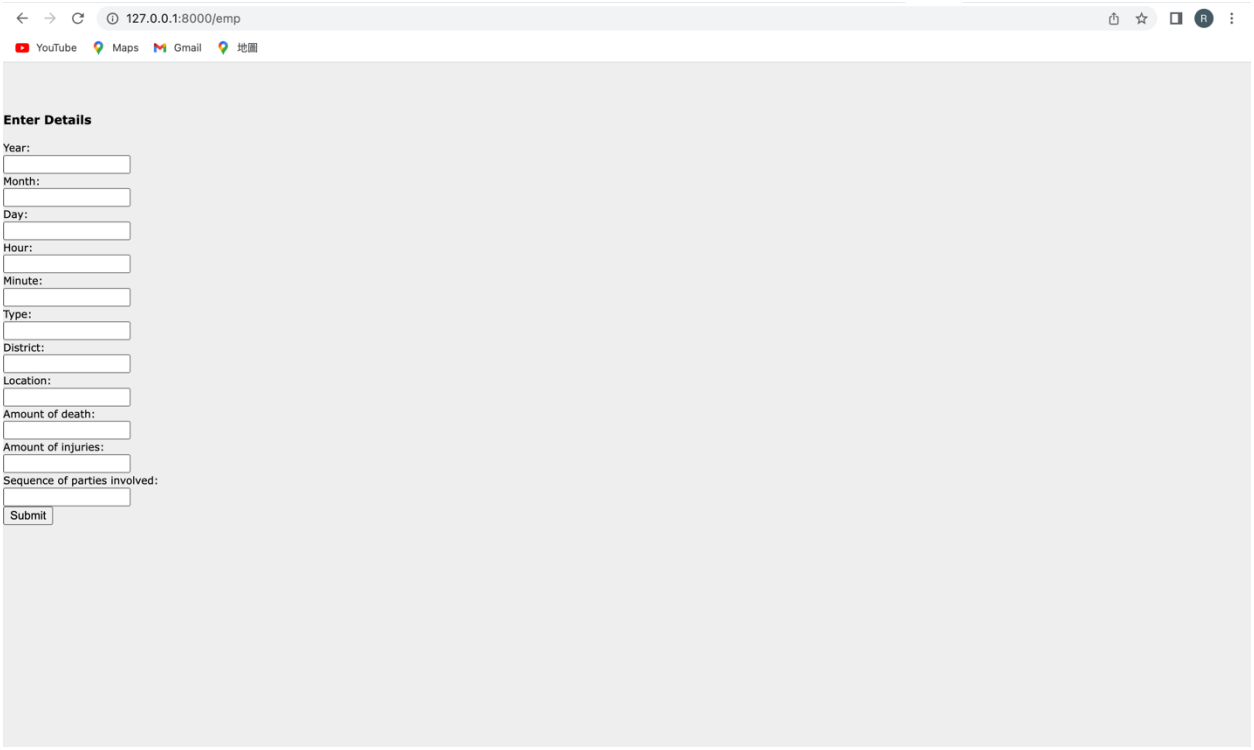
After clicking on the button “Show”, it will show all the available traffic accident records.

ID	Year	Month	Day	Hour	Type	District	Location	Amount of Death	Amount of injuries	Sequence of parties involved	
1	999	10	3	10	2	01大同區	大同區承德路與敦煌路口承德路 敦煌路	0	1	1	Edit Delete
2	101	10	3	10	2	01大同區	大同區承德路與敦煌路口承德路 敦煌路	0	1	2	Edit Delete
3	101	1	29	12	2	01大同區	大同區民權西路與延平北路口延平北路 民權西路	0	2	1	Edit Delete
4	101	1	29	12	2	01大同區	大同區民權西路與延平北路口延平北路 民權西路	0	2	2	Edit Delete
5	101	2	2	23	2	01大同區	大同區承德路與庫倫街口承德路 庫倫街	0	1	1	Edit Delete
6	101	2	2	23	2	01大同區	大同區承德路與庫倫街口承德路 庫倫街	0	1	2	Edit Delete
7	101	2	3	14	2	01大同區	大同區民族西路與重慶北路口重慶北路 民族西路	0	1	1	Edit Delete
8	101	2	3	14	2	01大同區	大同區民族西路與重慶北路口重慶北路 民族西路	0	1	2	Edit Delete
9	101	2	3	17	2	01大同區	大同區民權西路與台北橋口民權西路台北橋往台北市 機車道	0	1	1	Edit Delete
10	101	2	3	17	2	01大同區	大同區民權西路與台北橋口民權西路台北橋往台北市 機車道	0	1	2	Edit Delete
11	101	2	14	10	2	01大同區	大同區歸綏街232號	0	1	1	Edit Delete
12	101	2	14	10	2	01大同區	大同區歸綏街232號	0	1	2	Edit Delete
13	101	2	21	8	2	01大同區	大同區忠孝西路與環河北路口環河北路 忠孝西路	0	1	1	Edit Delete
14	101	2	21	8	2	01大同區	大同區忠孝西路與環河北路口環河北路 忠孝西路	0	1	2	Edit Delete
15	101	2	17	23	2	01大同區	大同區西寧北路與鄭州路口鄭州路 西寧北路	0	1	1	Edit Delete
16	101	2	17	23	2	01大同區	大同區西寧北路與鄭州路口鄭州路 西寧北路	0	1	2	Edit Delete
17	101	2	26	22	2	01大同區	大同區民權西路與蘭州街口民權西路 蘭州街	0	2	1	Edit Delete
18	101	2	26	22	2	01大同區	大同區民權西路與蘭州街口民權西路 蘭州街	0	2	2	Edit Delete
19	101	3	6	20	2	01大同區	大同區延平北路4段與迪化街2段口延平北路4段 迪化街2段254巷(延平北路4段111號前)	0	2	1	Edit Delete
20	101	3	6	20	2	01大同區	大同區延平北路4段與迪化街2段口延平北路4段 迪化街2段254巷(延平北路4段111號前)	0	2	2	Edit Delete
21	101	3	7	23	2	01大同區	大同區民權西路與延平北路3段口	0	1	1	Edit Delete
22	101	3	7	23	2	01大同區	大同區民權西路與延平北路3段口	0	1	2	Edit Delete
23	101	2	11	14	2	01大同區	大同區重慶北路與涼州街口重慶北路 涼州街	0	1	1	Edit Delete
24	101	2	11	14	2	01大同區	大同區重慶北路與涼州街口重慶北路 涼州街	0	1	2	Edit Delete
25	101	2	19	19	2	01大同區	大同區錦西街與雙連街口雙連街 錦西街	0	1	1	Edit Delete
26	101	2	19	19	2	01大同區	大同區錦西街與雙連街口雙連街 錦西街	0	1	2	Edit Delete
27	101	3	16	15	2	01大同區	大同區民樂街45號	0	2	1	Edit Delete
28	101	3	16	15	2	01大同區	大同區民樂街45號	0	2	2	Edit Delete
29	101	3	21	16	2	01大同區	大同區民生西路238號	0	1	1	Edit Delete
30	101	3	21	16	2	01大同區	大同區民生西路238號	0	1	2	Edit Delete
31	101	2	17	17	2	01大同區	大同區承德路與昌吉街口承德路 昌吉街	0	1	1	Edit Delete
32	101	2	17	17	2	01大同區	大同區承德路與昌吉街口承德路 昌吉街	0	1	2	Edit Delete
33	101	3	16	8	2	01大同區	大同區甘谷街與延平北路1段口	0	1	1	Edit Delete
34	101	3	16	8	2	01大同區	大同區甘谷街與延平北路1段口	0	1	2	Edit Delete
35	101	3	16	8	2	01大同區	大同區甘谷街與延平北路1段口	0	1	3	Edit Delete
36	101	3	16	16	2	01大同區	大同區哈密街95號	0	1	1	Edit Delete
36	101	3	16	16	2	01大同區	大同區哈密街95號	0	1	1	Edit Delete
37	101	3	16	16	2	01大同區	大同區哈密街95號	0	1	2	Edit Delete
38	101	4	2	14	2	01大同區	大同區承德路與敦煌路口承德路 敦煌路	0	2	1	Edit Delete
39	101	4	2	14	2	01大同區	大同區承德路與敦煌路口承德路 敦煌路	0	2	2	Edit Delete
40	101	4	13	18	2	01大同區	大同區大龍街222號	0	2	1	Edit Delete
41	101	4	13	18	2	01大同區	大同區大龍街222號	0	2	2	Edit Delete
42	101	3	25	13	2	01大同區	大同區長安西路與承德路口承德路 長安西路	0	1	1	Edit Delete
43	101	3	25	13	2	01大同區	大同區長安西路與承德路口承德路 長安西路	0	1	2	Edit Delete
44	101	3	26	15	2	01大同區	大同區台北橋台北橋機車道	0	2	1	Edit Delete
45	101	3	26	15	2	01大同區	大同區台北橋台北橋機車道	0	2	2	Edit Delete
46	101	4	28	2	2	01大同區	大同區長安西路與承德路口承德路 長安西路	0	2	3	Edit Delete
47	101	4	28	2	2	01大同區	大同區長安西路與承德路口承德路 長安西路	0	2	4	Edit Delete
48	101	4	29	17	2	01大同區	大同區重慶北路1段50號	0	1	1	Edit Delete
49	101	4	29	17	2	01大同區	大同區重慶北路1段50號	0	1	2	Edit Delete
50	101	5	8	19	2	01大同區	大同區長安西路與南京西路口長安西路54號 南京西路18巷口	0	1	1	Edit Delete

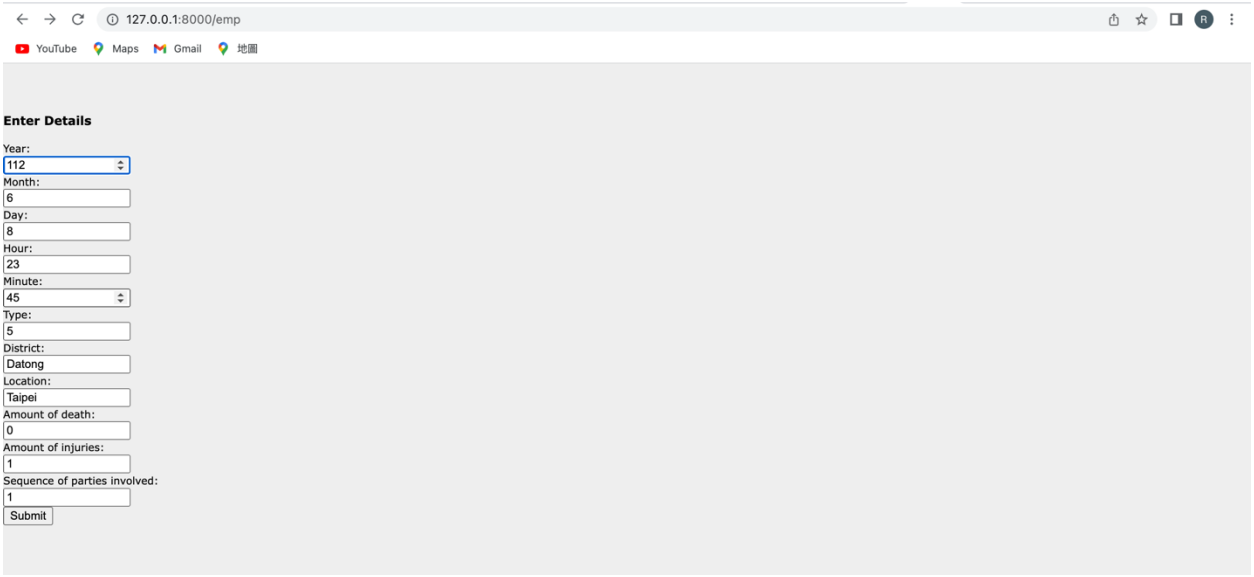
<< < 1 > >>
Add New Record

Adding Record:

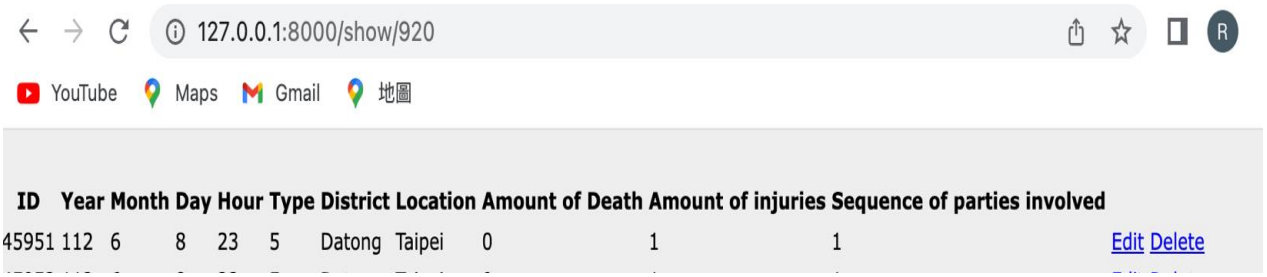
After clicking on the Add New Record button and fill the details:



Filling details:

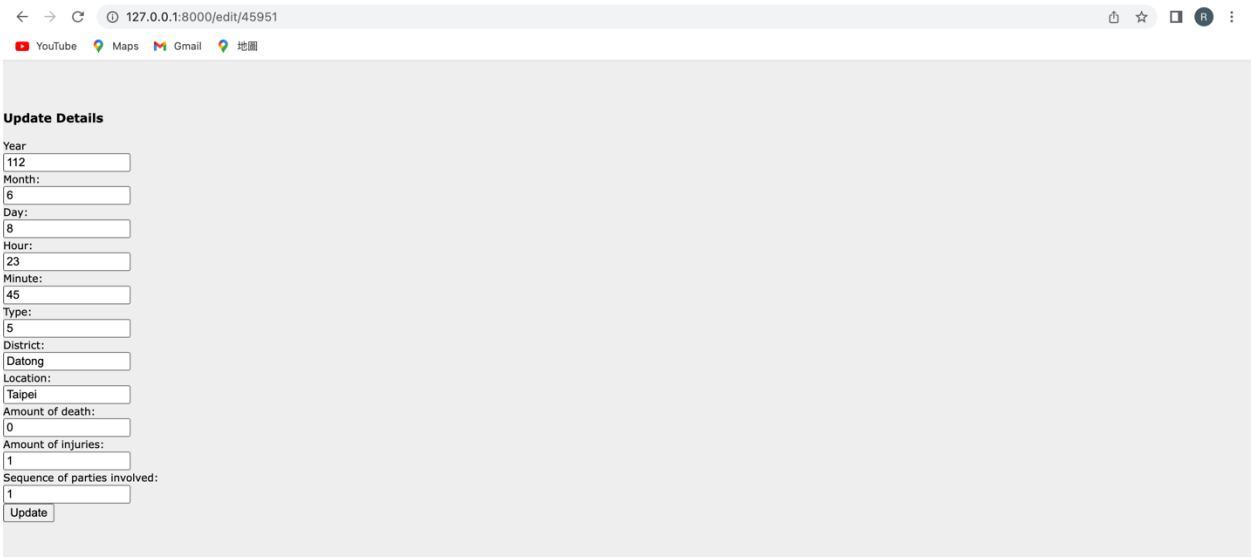


Submit the record and see, after submitting it shows the saved record.
This section also allows, update and delete records .



After saving couple of random records, now we have following records.

Update Record:
After clicking on Edit button whom ID is 45951, it will display record of 45951 in edit mode:

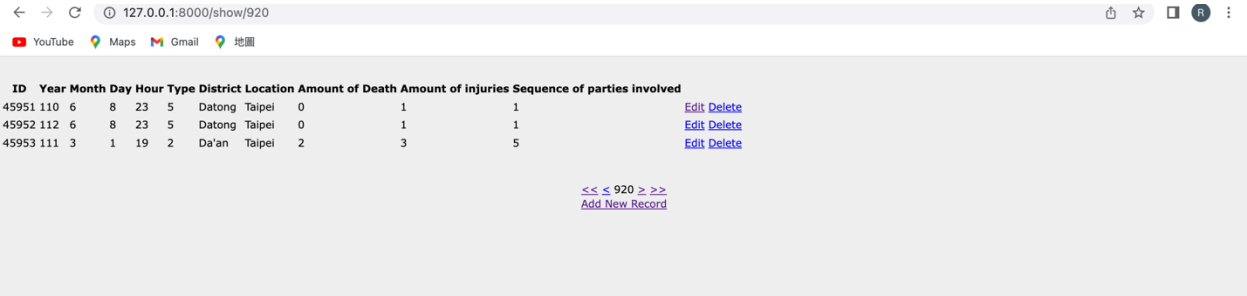


Let's suppose I update year to 110 then click on the update button. It updates the record immediately. See the example.

Update Details

Year
110
Month:
6
Day:
8
Hour:
23
Minute:
45
Type:
5
District:
Datong
Location:
Taipei
Amount of death:
0
Amount of injuries:
1
Sequence of parties involved:
1
Update

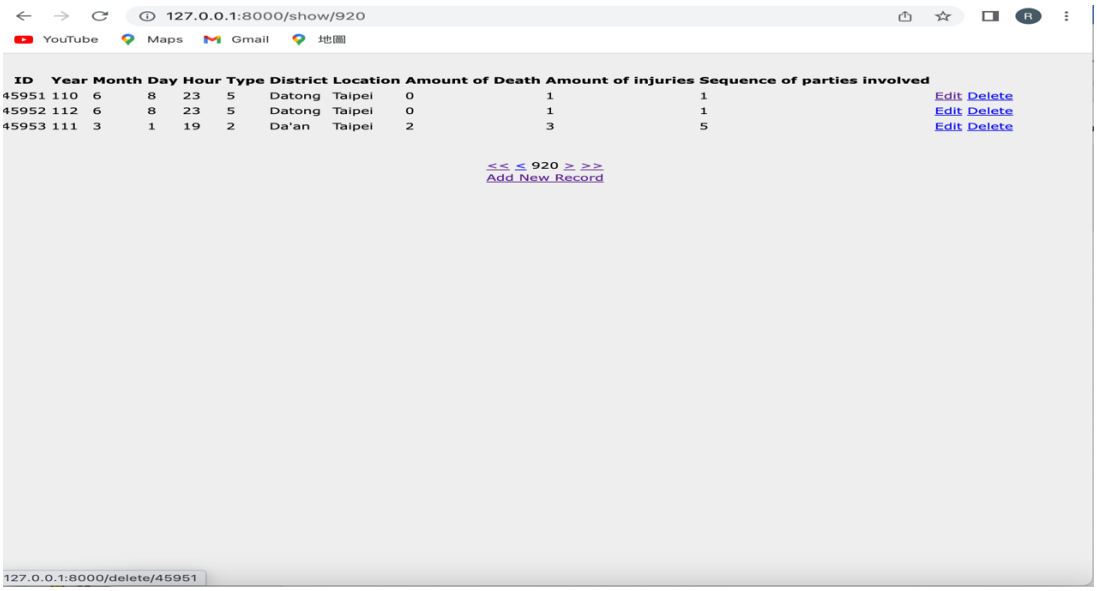
Click on update button and it redirects to the following page. See Year whom ID is 45951 is updated.



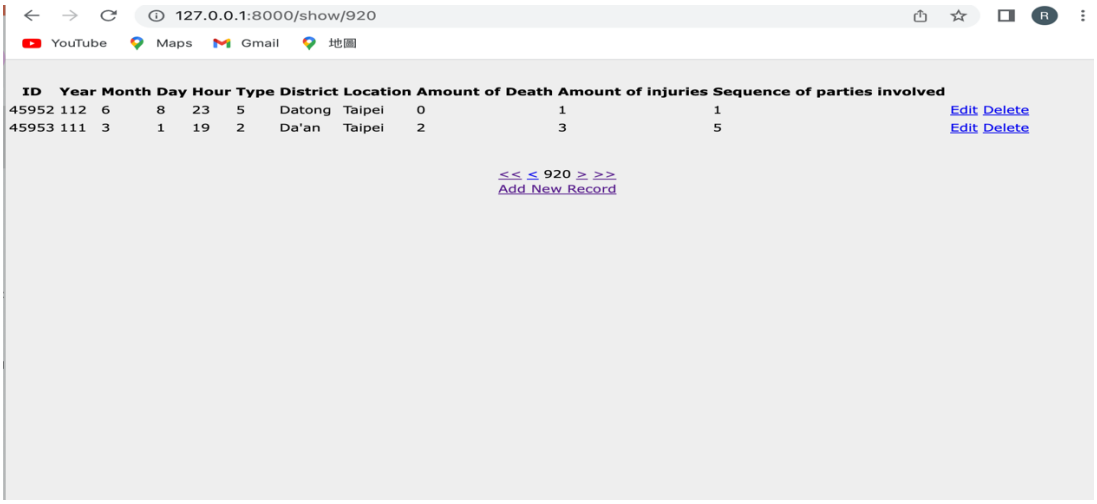
Same as, we can delete records too, by clicking the delete link.

Delete Record:

Suppose I want to delete 45951, it can be done easily by clicking the delete button. See the example.



After deleting, we left with the following records.



So, we've successfully built a Django CRUD application.

- query design (including discussion on “why the query is designed like this”)
  The query in the GetLevelofinjurybygender function is designed to retrieve the count of accidents based on the level of injury, categorized by gender. , providing insights into the distribution of injuries among males and females involved in car accidents.

- Exception handling
  Use Try-Except Blocks: Put a try-except block around any code that might raise an exception. Identify and manage specific exceptions that you think are going to occur. You

can display user-friendly error messages, log the error details for troubleshooting, or take other actions to manage the unexpected situation .

- Others
  Expected Progress:

  Database Design [####################] 100% complete
  Django Setup [####################] 100% complete
  Create Models [####################] 90% complete.
  Create Views [####################] 100% complete.
  Create Templates [#################] 80% complete.
  Implement Create Functionality [##############] 70% complete.
  Implement Read Functionality [#################] 80% complete.
  Implement Update Functionality [############] 75% complete.
  Implement Delete Functionality [############] 75% complete.
  Testing and Bug Fixing [###########] 60% complete

  Actual Progress:

  Database Design [####################] 100% complete
  Django Setup [####################] 100% complete
  Create Models [####################] 90% complete.
  Create Views [####################] 100% complete.
  Create Templates [#################] 80% complete.
  Implement Create Functionality [############] 80% complete.
  Implement Read Functionality [#################] 85% complete.
  Implement Update Functionality [############] 75% complete.
  Implement Delete Functionality [############] 75% complete.
  Testing and Bug Fixing [##########] 70% complete

  - What were the problems you met in the project, how did you solve them?

  1) About creating the table for the Database because all the information from the government data is in Chinese, I use chatgpt to help me translate.
  2) I encounter difficulties to connect the database to the web application, I read some documents online which help me solve it.

  - List the contribution of each team member in the project clearly
    Overall distribution by percentage
    卜銳凱：80%
    呂晟愷:10%
    張啟明:10%
    Distribution Tasks

    Report: 卜銳凱

Database design: 卜銳凱
CRUD: 卜銳凱
Web application: 卜銳凱

- Provide a link to your repository
  https://github.com/Buteau17/NYCU-Database-project/

- Provide a link to your discussion channel
  https://hackmd.io/lhCOkHHkSwiDQ9DGZyY5gA