**Software :**

**We have used Django framework for building the website which can display and store the data .**

**What is the Django Framework?**

Django is an open-source python web framework used for rapid development, pragmatic, maintainable, clean design, and secure websites. A web application framework is a toolkit of all components needed for application development.

The main goal of the **Django** framework is to allow developers to focus on components of the application that are new instead of spending time on already developed components. Django is fully featured than many other frameworks on the market. It takes care of a lot of hassles involved in web development; enables users to focus on developing components needed for their application.

**Top 5 Uses of Django**

**1. Easy to Use**

**2. It’s fast and simple**

**3. Excellent Documentation for real-world application**

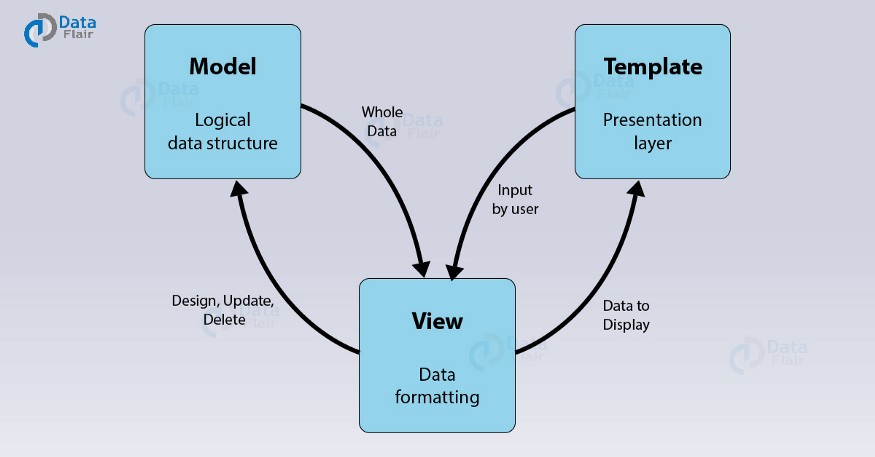
**4. It’s secure**

**5. It suits any web application project**

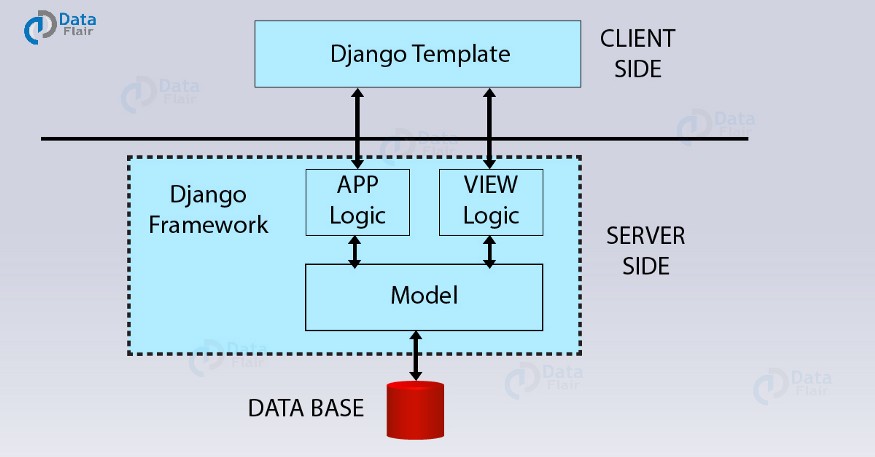
**The Django MTV architecture is composed of the following components:**

* **Model**: it defines the logical data structure. In practice, **a model is a Python class, which represents a single table in the database**. All the classes representing the logical structure of the database are stored in a script named **models.py**.
* **View**: it defines the business logic, in the sense that it communicates with the model and translates it into a format readable by the Template. **A view is a Python function, which takes a request as input and returns a Web response as output**. All the functions representing the business logic are stored in a script named **views.py**.
* **Template**: it defines the structure or the layout of a file, such as a HTML file. **It is a text document or a Python string encoded through the Django template language.**

The following figure illustrates the Django MTV architecture and how the Model, Template and View components interact each other:



Django Work Flow :



As from the above diagram, we have some components and two regions i.e., server side and client side. Here you will notice that the View is on the server-side part while the template is on the client side.

Now, when we request for the website, the interface through which we use to make that request via our browser was the Template. Then that request transmits to the server for the management of view file.

Django is literally a play between the requests and responses. So whenever our Template is updating it’s the input (request) we sent from here which on the server was seen by the View. And, then it transports to the correct URL. It’s one of the important components of Django MTV architecture. There, the URL mapping in Django is actually done in regular expressions. These expressions are much more understandable than IP addresses. It also helps with the SEO task which we have discussed in the Django Features Tutorial.

Now after the sending of a request to the correct URL, the app logic applies and the model initiates to correct response to the given request. Then that particular response is sent back to the View where it again examines the response and transmits it as an HTTP response or desired user format. Then, it again renders by the browser via Templates.

An easier real-life working of above functioning would be –

When you login in a website ([Django](https://www.djangoproject.com/start/) based), you open the login page. It again happens without the need of the Model. It is because Views will process the request and send it to the URL of the login page. Then, it will be a response by the server, from there to the browser.

After that, you enter your credentials in the given Template, HTML form. From there the data is again sent to the view, this time this request rectifies and the model is given data. Then the Model reads and verifies the data that the user provides within the connected database.

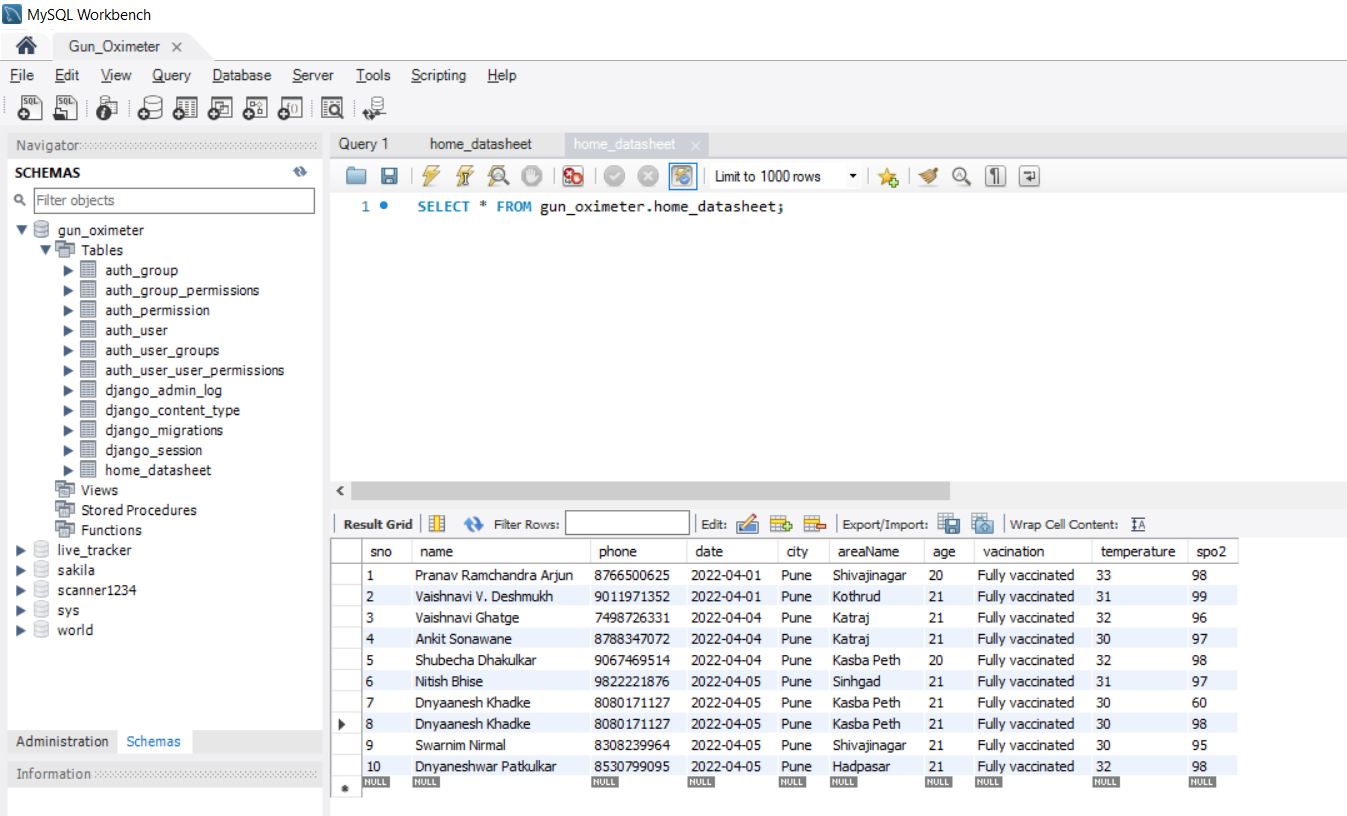
If the user data matches it will send the relevant user data like profile image, name and (other things depending on the type of website) to the Views. It will then format the same in desired response and will transmit the same to the client.

Otherwise, the Model will send a negative result to the Views. In turn, it will rout it to the login page again alongside an error message.

That’s how the Django MTV architecture is actually working.

Test Analysis :

My SQL DataBase :



**Code :**

**1. Arduino Code :**

//GUN OXIMETER.

// Adding libraries

#include <Wire.h>

#include "MAX30105.h"

#include "spo2\_algorithm.h"

#include <Adafruit\_MLX90614.h>

#include <LiquidCrystal.h>

int Contrast=145;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int temp;

#include<NewPing.h>

const int trigPin = 9;

const int echoPin = 10;

float duration, distance;

Adafruit\_MLX90614 mlx = Adafruit\_MLX90614();

MAX30105 particleSensor;

#define MAX\_BRIGHTNESS 255

#if defined(\_\_AVR\_ATmega328P\_\_) || defined(\_\_AVR\_ATmega168\_\_)

//Arduino Uno doesn't have enough SRAM to store 100 samples of IR led data and red led data in 32-bit format

//To solve this problem, 16-bit MSB of the sampled data will be truncated. Samples become 16-bit data.

uint16\_t irBuffer[50]; //infrared LED sensor data

uint16\_t redBuffer[50]; //red LED sensor data

#else

uint32\_t irBuffer[50]; //infrared LED sensor data

uint32\_t redBuffer[50]; //red LED sensor data

#endif

int32\_t bufferLength; //data length

int32\_t spo2; //SPO2 value

int8\_t validSPO2; //indicator to show if the SPO2 calculation is valid

int32\_t heartRate; //heart rate value

int8\_t validHeartRate; //indicator to show if the heart rate calculation is valid

byte pulseLED = 11; //Must be on PWM pin

byte readLED = 13; //Blinks with each data read

void setup() {

//LCD Set UP

analogWrite(6,Contrast);

lcd.begin(20, 4);

//Presence Sensor Setup

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

Serial.begin(9600);

// //HR and SPO2 Sensor Setup

Serial.begin(9600); // initialize serial communication at 115200 bits per second:

pinMode(pulseLED, OUTPUT);

pinMode(readLED, OUTPUT);

if (!particleSensor.begin(Wire, I2C\_SPEED\_FAST)) //Use default I2C port, 400kHz speed

{

Serial.println(F("MAX30105 was not found. Please check wiring/power."));

while (1);

}

Serial.read();

byte ledBrightness = 60; //Options: 0=Off to 255=50mA

byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32

byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green

byte sampleRate = 100; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200

int pulseWidth = 411; //Options: 69, 118, 215, 411

int adcRange = 4096; //Options: 2048, 4096, 8192, 16384

particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcRange);

//Configure sensor with these settings

mlx.begin();

}

void loop() {

digitalWrite(trigPin, LOW);

delayMicroseconds(1000);

digitalWrite(trigPin, HIGH);

delayMicroseconds(1000);

digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);

distance = (duration\*.0343)/2;

lcd.clear();

lcd.setCursor(0, 0);

lcd.print(distance);

lcd.print(",");

// Temperature

temp= mlx.readObjectTempC();

//LCD

delay(500);

lcd.clear();

lcd.setCursor(0, 2);

lcd.print("temperatre= ");

lcd.print(mlx.readObjectTempC());

lcd.print("C");

delay(1000);

//SPO2

long irValue = particleSensor.getIR();

bufferLength = 50; //buffer length of 100 stores 4 seconds of samples running at 25sps

//read the first 50 samples, and determine the signal range

for (byte i = 0 ; i < bufferLength ; i++)

{

while (particleSensor.available() == false) //do we have new data?

particleSensor.check(); //Check the sensor for new data

redBuffer[i] = particleSensor.getRed();

irBuffer[i] = particleSensor.getIR();

particleSensor.nextSample();

}

maxim\_heart\_rate\_and\_oxygen\_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

//Continuously taking samples from MAX30102. Heart rate and SpO2 are calculated every 1 second

while (1)

{

//dumping the first 25 sets of samples in the memory and shift the last 75 sets of samples to the top

for (byte i = 15; i < 50; i++)

{

redBuffer[i - 15] = redBuffer[i];

irBuffer[i - 15] = irBuffer[i];

}

//take 25 sets of samples before calculating the heart rate.

for (byte i = 25; i < 50; i++)

{

while (particleSensor.available() == false) //do we have new data?

particleSensor.check(); //Check the sensor for new data

digitalWrite(readLED, !digitalRead(readLED)); //Blink onboard LED with every data read

redBuffer[i] = particleSensor.getRed();

irBuffer[i] = particleSensor.getIR();

particleSensor.nextSample();

if (validSPO2){

Serial.print(distance);

Serial.print(",");

Serial.print(temp);

Serial.print(",");

Serial.print(spo2, DEC);

Serial.println(",");

}

delay(1000);

//LCD

lcd.setCursor(0, 3);

lcd.print("spo2= ");

lcd.print(spo2);

lcd.print("%");

}

maxim\_heart\_rate\_and\_oxygen\_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

}

}

}

**Python Code :**

import serial

import time

import schedule

from django.http import JsonResponse

import numpy as np

def Arduino(request):

arduino = serial.Serial('com3', 9600)

print('Established serial connection to Arduino')

data = arduino.readline()

data = str(data,'utf')

data=data.strip('\r\n')

data=data.split(",")

# Decalring the varibales

dist=(data[0])

temp = (data[1])

spo2 = (data[2])

arduino.close()

return JsonResponse({'temp': temp, 'spo2': spo2})



Fig. Temperature accuracy graph

Fig. Oxygen accuracy graph