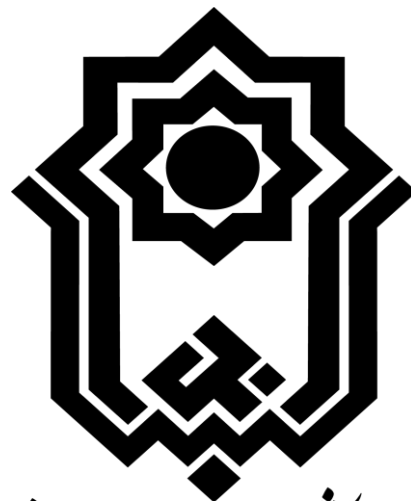


به نام خدا



دانشگاه بوعلی سینا

نام و نام خانوادگی: سید فرهاد حسینی

شماره دانشجویی : ۹۶۱۲۳۵۸۰۱۶

نام درس : پروژه کارشناسی

استاد مربوطه : دکتر منصوری زاده

موضوع : پیاده سازی سیستم پایش محیطی

اینجانب سید فرهاد حسینی دانشجوی رشته مهندسی کامپیوتر دانشگاه بوعلی سینا از تاریخ ۱۴۰۰/۰۴/۱ الی ۱۴۰۰/۰۶/۱۵ روی پروژه با موضوع سیستم پایش محیطی زیر نظر استاد راهنما جناب دکتر منصوری زاده فعالیت نموده و بدین وسیله تعهد می‌کنم که مطالب این پایان نامه همگی نتیجه فعالیت و تحقیقات اینجانب می‌باشد.

بدین وسیله گواهی می گردد که آقا/خانم..... دانشجوی رشته..... از تاریخ تا زیر نظر اینجانب تحقیق و فعالیت نموده و بدین وسیله ایشان را واجد شرایط برای دریافت مدرک کارشناسی می دانم و بانمره ی کارایشان را تایید می نمایم.

فهرست :

۵	• مقدمه :
۶	• توضیح صورت پروژه :
۶	• ویژگی ها و امکانات :
۷	• شماتیک بخش های سیستم :
۷	• Deployment model :
۸	• ابزارهای پیاده سازی :
۸	• واسط کاربری :
۸	• حسگر :
۸	• کنترلر :
۸	• polling :
۹	• Usecase diagram :
۱۰	• تحلیل و طراحی دیتابیس :
۱۰	• ER Diagram :
	• پیاده سازی :
۱۱	• حسگر :
۱۶	• polling :
۱۸	• معماری mvc :
۱۹	• Jobs and queue in laravel :
۲۰	• کنترلر :
۲۱	• روتها :
۲۲	• مدل ها :
۲۳	• کنترلر ها :
۲۹	• بخش ارسال ایمیل (job and queue) :
۳۰	• اجرا :
۳۰	• بخش سنسور :
۳۲	• بخش کنترلر :
۳۵	• بخش اقدام (ایمیل) :
۳۶	• آدرس فایل ها و کدها :
۳۶	• مراجع :

● مقدمه :

موضوع این پروژه پیاده سازی یک سیستم مانیتورینگ (پایش محیطی) در یک حالت عمومی است .
Surveillance system ها به منظور کنترل رفتار ها و فعالیت ها در یک مجموعه استفاده میشود تا
برخی رفتار ها که برای کاربر مهم هستند و احتمالاً مخرب اند را سریع تشخیص دهیم و اقدام مناسب را
انجام دهیم مثلاً تشخیص ورود و خروج افراد به یک مکان خاص ، تشخیص تغییر شکل ظاهری یک ماده
خاص در صنعت که با این کار براحتی از افزایش خسارت جلوگیری میشود . کاربرد های این سیستم صرفاً
برای جلوگیری از یک فعالیت مخرب نیست برای مثال میتوان با نصب دوربین در فروشگاه های بزرگ و
شمردن افراد در بخش های مختلف فروشگاه تحلیل های خاصی را به منظور تبلیغات موثرتر به عمل آورد .

• توضیح صورت پروژه :

هدف از این پروژه طراحی و پیاده سازی یک سیستم پایش محیطی است . این سیستم از ۴ بخش مجزا از هم تشکیل شده است . بخش اول یک دوربین است که یک نرم افزار به منظور تشخیص رخداد ها بر روی آن نصب است . این دوربین وظیفه دریافت اطلاعات از محیط را برعهده دارد و اگر شی مشکوکی را شناسایی کرد به بخش دوم یعنی کنترلر اطلاع میدهد . وظیفه اصلی بخش دوم بررسی و ثبت رخداد در پایگاه داده و نیز ارسال برخی اطلاعات به بخش سوم یعنی بخش action میباشد . این بخش وظیفه اطلاع رسانی به کاربر بکمک ارسال ایمیل را برعهده دارد . بخش چهارم نیز polling نام دارد . این بخش وظیفه این را به عهده دارد که سامانه را بررسی کند و در صورت وجود اشکال یا خرابی آنرا به کاربر اطلاع دهد .

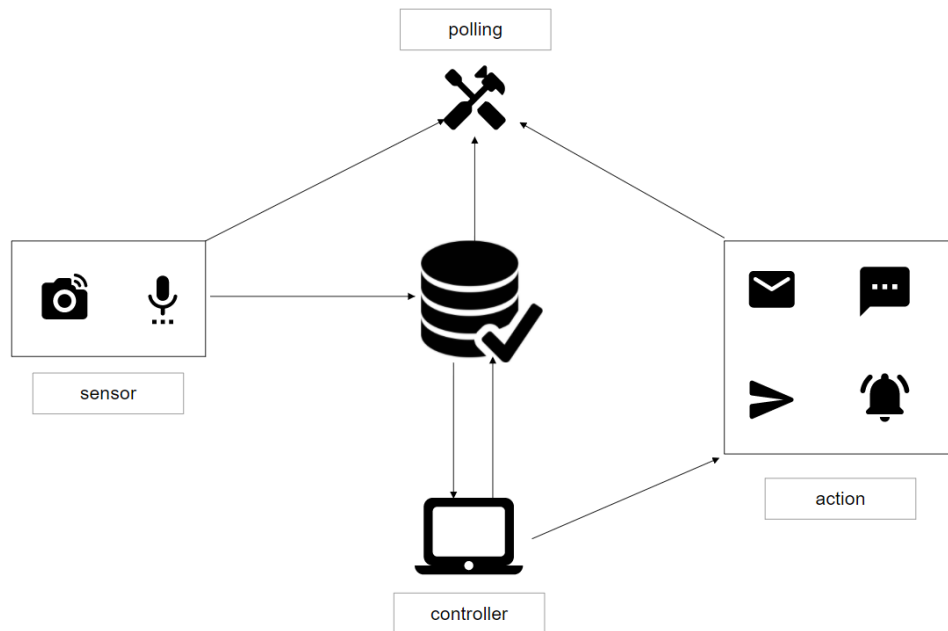
ویژگی ها و امکانات :

در این سیستم چندین ویژگی نیز پیاده سازی شده است . اولین ویژگی بحث احراز هویت (authentication) میباشد که کاربر با وارد کردن ایمیل و رمز عبور خود وارد سامانه شده و از آن استفاده میکند . همچنین بخش فراموشی رمز عبور نیز در آن پیاده سازی شده است .

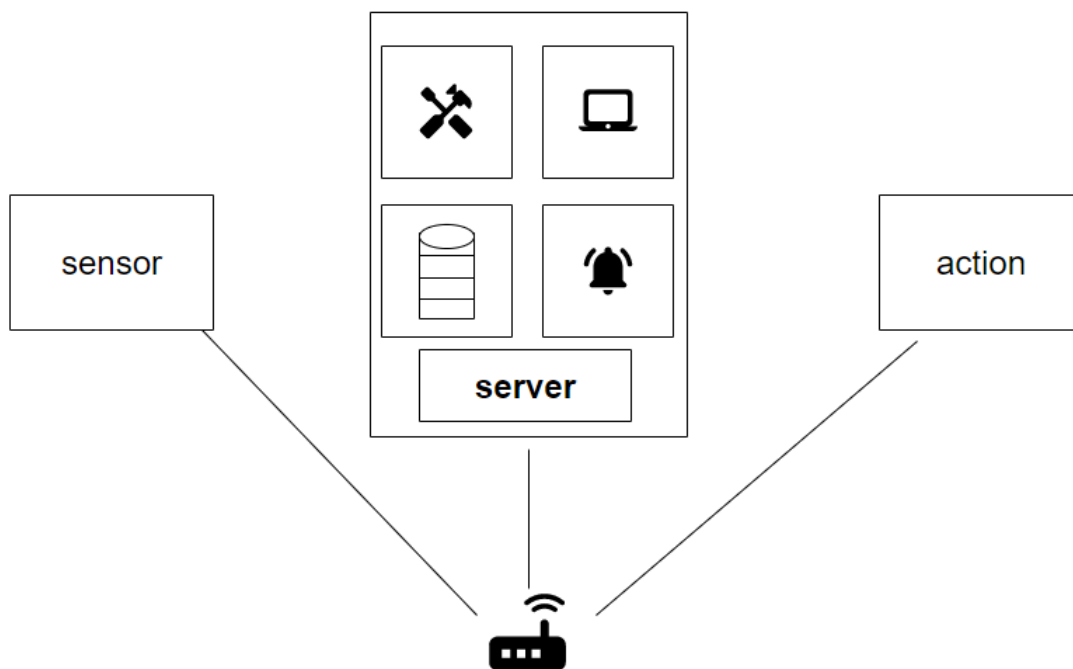
ویژگی دوم تنظیم و تغییر متن ایمیل ها میباشد . در واقع کاربر پس از ورود به سامانه در بخش رخداد ها و اقدام ها میتواند بر اساس نوع هر رخداد ، متن ایمیل آنرا به دلخواه خود تغییر دهد .

ویژگی بعدی اجرای برخی پرس و جو های پر کاربرد میباشد . کاربر میتواند لیست تمام رخداد ها را درخواست کند تا به شکل یک جدول آنرا مشاهده کند . کاربر میتواند لیست رخداد هایی که اقدامی برای آنها نشده را دریافت کند . منظور از اینکه اقدامی برای آن نشده این است که به ازای آن رخداد کاربر ایمیلی را دریافت نکرده است . پرس و جوی بعدی کمی پویاتر است و در آن کاربر میتواند با وارد کردن دو تاریخ لیست تمام رخدادهایی که در این بازه اتفاق افتاده اند را دریافت کند .

● شماتیک بخش های سیستم :



: *Deployment model*



• ابزارهای پیاده سازی :

برای پیاده سازی ها بخش از این سیستم از زبانها و تکنولوژی هایی استفاده شده که آنها را بررسی میکنیم .

واسط کاربری :

برای اینکه کاربر بتواند به آسانی با این نرم افزار کار کند بایستی یک واسط کاربری گرافیکی طراحی میشد .
برای اینکار از زبان های نشانه گذاری html و css استفاده شده و برای زیباتر کردن محیط آن از فریمورک bootstrap استفاده شده است .

حسگر :

همانطور که گفتیم سنسور دوربین نیازمند یک نرم افزار میباشد که به کمک آن اتفاقات را تشخیص دهد . برای پیاده سازی آن از زبان پایتون و همچنین برای تشخیص اتفاقات از کتابخانه cv2 استفاده شده است .

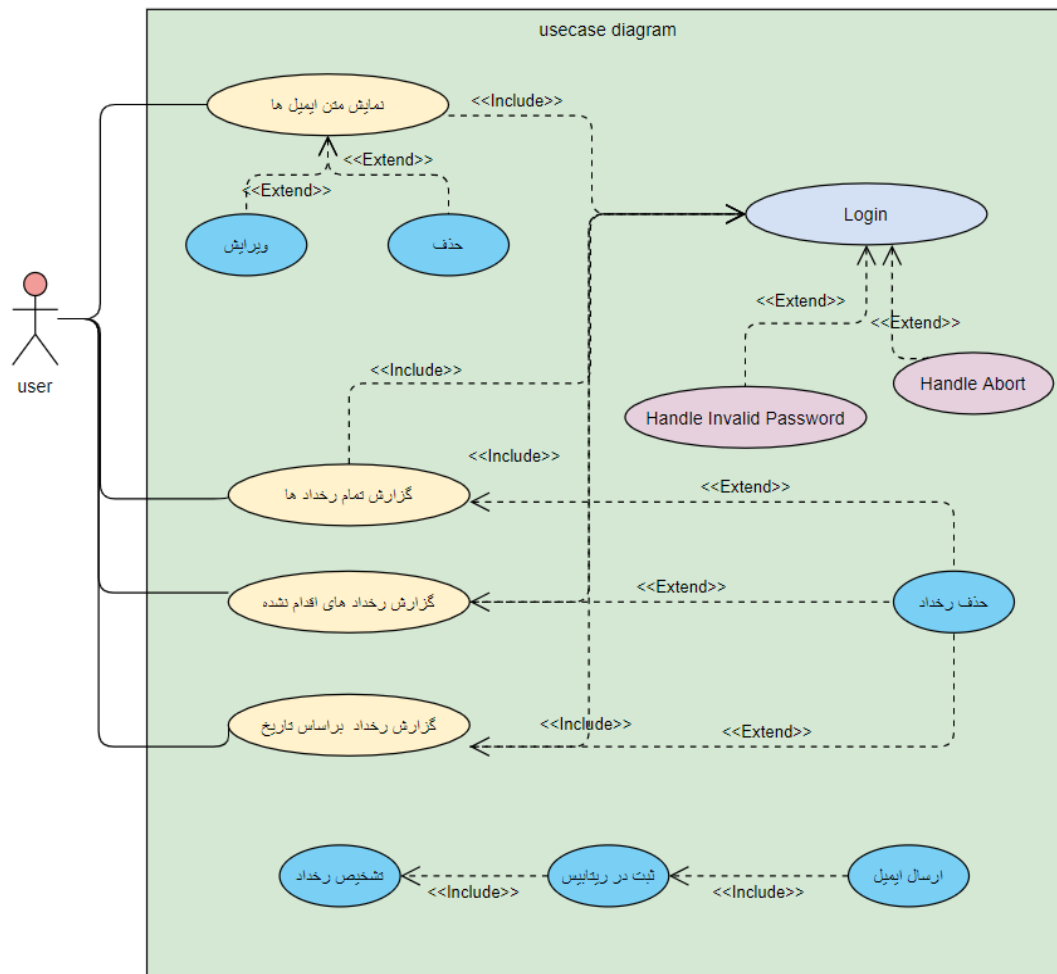
کنترلر :

این قسمت از سیستم که بزرگترین بخش است به کمک فریمورک لاراول که مبتنی بر زبان php است پیاده سازی شده و از برخی امکانات این فریمورک استفاده شده .

: polling

این بخش نیز همانند بخش حسگر با زبان پایتون پیاده سازی شده است .

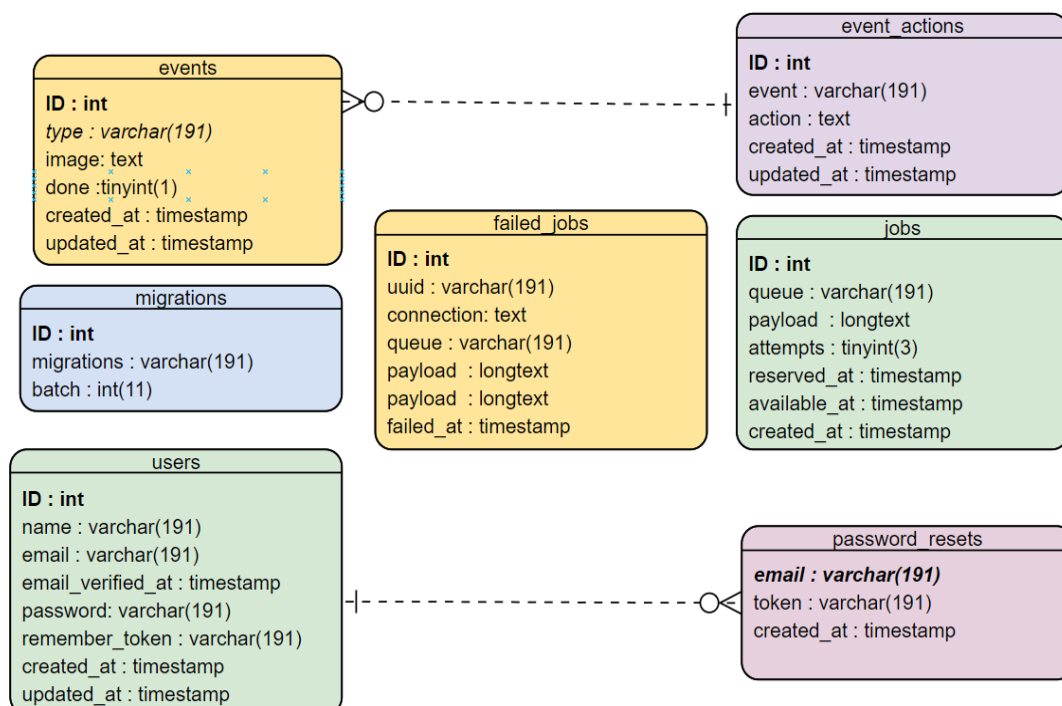
• Usecase diagram : Usecase diagram



• تحلیل و طراحی دیتابیس :

در این نرم افزار بدیل اینکه ما نیاز داریم برخی اطلاعات را برای مدتی طولانی در اختیار داشته باشیم تا از آنها در موارد مختلف استفاده کنیم بایستی این اطلاعات را در جایی ذخیره کنیم . برای ذخیره اطلاعات ما از پایگاه داده mysql استفاده میکنیم . همانطور که میدانیم این نوع دیتابیس ها برای ذخیره اطلاعات از جداول استفاده میکنند .

: ER Diagram



• پیاده سازی :

حسگر :

پیاده سازی بخش حسگر از دوفایل تشکیل شده است . یک فایل اصلی برنامه و یکی هم پیاده سازی توابع مورد استفاده .

ابتدا به بررسی توابع موجود در فایل func.py میپردازیم .

```
def cat(cat_classifier,gray,img,catCounter ,catTime):  
    # Detects faces of different sizes in the input image  
    cat = cat_classifier.detectMultiScale(gray, 1.3, 5)  
    for (x,y,w,h) in cat:  
        # To draw a rectangle in a face  
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)  
  
    if(len(cat) != 0):  
        catCounter += 1  
    if(catCounter >= 3):  
        print(cat)  
        catCounter=0  
        catTime = datetime.datetime.now()  
        sendEvent(img ,catTime , "cat" )  
  
    # Display an image in a window  
    cv2.imshow('img',img)  
    return catTime , catCounter
```

در تابع بالا تصویر خوانده شده از دوربین در یک پنجره به نمایش در میآید اما قبل از آن اگر صورت گربه ای در تصویر تشخیص داده شد دور آن یک مستطیل رسم میکند . نکته ای که باید به آن توجه داشت این است که برای جلوگیری از خطاهای احتمالی بررسی میشود که اگر در سه تصویر متوالی گربه را تشخیص داد آنگاه مطمئناً گربه را تشخیص داده است و پس از تشخیص گربه ، تصویر را به همراه مهر زمانی آن تصویر به تابع sendEvent ارسال میکند .

```
def fullBody(body_classifier,gray,img,bodyCounter ,bodyTime):
    # Detects faces of different sizes in the input image
    bodies = body_classifier.detectMultiScale(gray)

    # Extract bounding boxes for any bodies identified
    for (x,y,w,h) in bodies:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 90, 120), 2)

    if(len(bodies) != 0):
        bodyCounter += 1
    if(bodyCounter >= 3):
        print(bodies)
        bodyCounter=0
        bodyTime = datetime.datetime.now()
        sendEvent(img ,bodyTime ,"body" )

    cv2.imshow('img', img)
    return bodyTime , bodyCounter
```

در تابع بالا تصویر خوانده شده از دوربین در یک پنجره به نمایش در میآید اما قبل از آن اگر بدن انسان در تصویر تشخیص داده شد دور آن یک مستطیل رسم میکند . نکته ای که باید به آن توجه داشت این است که برای جلوگیری از خطاهای احتمالی بررسی میشود که اگر در سه تصویر متوالی بدن انسان را تشخیص داد آنگاه مطمئناً بدن را تشخیص داده است و پس از تشخیص بدن ، تصویر را به همراه مهر زمانی آن تصویر به تابع sendEvent ارسال میکند .

```
def face(face_classifier,gray,img,faceCounter , faceTime):
    # Detects faces of different sizes in the input image
    faces = face_classifier.detectMultiScale(gray, 1.0485258, 6)

    # Extract bounding boxes for any bodies identified
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x, y), (x+w, y+h), (40, 255, 20), 2)

    if(len(faces) != 0):
        faceCounter += 1
    if(faceCounter >= 3):
        print(faces)
        faceTime = datetime.datetime.now()
        faceCounter=0
        sendEvent(img ,faceTime , "face")

    cv2.imshow('img', img)
    return faceTime , faceCounter
```

در تابع بالا تصویر خوانده شده از دوربین در یک پنجره به نمایش در می‌آید اما قبل از آن اگر چهره انسان در تصویر تشخیص داده شد دور آن یک مستطیل رسم میکند . نکته ای که باید به آن توجه داشت این است که برای جلوگیری از خطاهای احتمالی بررسی میشود که اگر در سه تصویر متوالی چهره انسان را تشخیص داد آنگاه مطمئناً چهره را تشخیص داده است و پس از تشخیص چهره ، تصویر را به همراه مهر زمانی آن تصویر به تابع `sendEvent` ارسال میکند .

```
def sendEvent(img , time ,type):

    URL = "http://localhost:8000/api/event/create"

    time = time.strftime('%Y-%m-%d')

    address = type + '_' + str(time) + '_' + str(random.randint(100000)) + '.png'

    status = cv2.imwrite('images/'+
    address , img)
    cv2.imwrite('../controller/public/images/'+address , img)
    print(status)

    data = {'type': type ,
            'image' : address }

    r = requests.post(url = URL, data = data)

    # extracting response text
    pastebin_url = r.text
    print("The pastebin URL is:%s"%pastebin_url)
```

حال نوبت بررسی تابع `sendEvent` میرسد . در این تابع ابتدا یک نام بر اساس نوع تصویر ، تاریخ تصویر و یک عدد رندوم برای تصویر ساخته شده و سپس تصویر ذخیره میشود . در آخر این تصویر را به همراه نوع تصویر به کمک `api` ای که در کنترلر پیاده سازی شده است بصورت متود `post` به کنترلر ارسال میکند تا در آنجا بررسی شده و در دیتابیس ذخیره شود .

حالا به بررسی فایل اصلی بخش حسگر میپردازیم :

```
PICTURE_TIME = 1 # min
catCounter = 0
bodyCounter = 0
faceCounter = 0
```

ابتدا ۴ متغیر تعریف شده که بسیار مهم اند .

یکی از مشکلاتی که در این پیاده سازی وجود دارد این است که اگر برنامه سنسور بعنوان مثال گربه ای را تشخیص داد در تصاویر بعدی هم مکررا گربه را تشخیص میدهد و در یک ثانیه تعداد خیلی زیادی گربه را تشخیص میدهد که این موضوع اصلا جالب نیست . برای جلوگیری از این مشکل متغیر اول یک تایم را در خود نگه میدارد مثلا ۱ دقیقه . این بدین معناست که اگر گربه ای را برنامه تشخیص داد تا یک دقیقه بعد گربه ای را تشخیص ندهد .

سه متغیر بعدی هم قبلا توضیح داده شده . (تعداد تصاویر متوالی شامل شی خاص را می‌شمارد . اگر به ۳ رسید مطمئن میشود که شی را واقعا تشخیص داده است .)

```
while 1:

    # reads frames from a camera
    ret, img = cap.read()
    cv2.imshow('img',img)

    # convert to gray scale of each frames
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    t_X_mins_ago = datetime.datetime.now() - datetime.timedelta(minutes = PICTURE_TIME)
    if(catTime < t_X_mins_ago):
        catTime , catCounter = func.cat(cat_classifier,gray,img,catCounter,catTime)
    if(bodyTime < t_X_mins_ago):
```

```

        bodyTime , bodyCounter = func.fullBody(body_classifier,gray,img , bodyCounter,bodyTime)
        bodyTime , bodyCounter = func.upperBody(ubody_classifier,gray,img , bodyCounter,bodyTime)
        if(faceTime < t_X_mins_ago):
            faceTime , faceCounter = func.face(face_classifier,gray,img , faceCounter,faceTime)
        # func.body(img)
        # Wait for Esc key to stop
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

```

بخش اصلی برنامه از یک حلقه بینهایت تشکیل شده و هربار تصویر گرفته شده از دوربین را به نمایش در میآورد . هربار بررسی میکند که آیا به تازگی مثلاً گربه را دیده است یا خیر اگر ندیده بود تابع تشخیص شی گربه (که بالا تر بررسی کردیم) را صدا میزند تا اگر گربه ای وجود دارد بصورت محاط شده در یک مستطیل به نمایش در بیاید .

در آخر نیز منتظر میماند که اگر کاربر کلید q را فشرد از حلقه خارج میشود .

: polling

همان طور که گفتیم وظیفه بخش polling این است که اگر خرابی و یا نقصی در سیستم وجود داشت آنرا به کاربر گزارش دهد . اساس کار ما هم در این بخش به این صورت است که این برنامه چند وقت یکبار دیتابیس را چک میکند و آخرین رکورد در جدول events را به لحاظ زمانی پیدا کرده و تفاضل زمان آنرا با تاریخ و ساعت فعلی محاسبه کرده و اگر از یک مقداری بزرگتر شد به این معناست که مدت زمان زیادی از ثبت آخرین رکورد در دیتابیس میگذرد . پس این احتمال وجود دارد که خرابی ای در سیستم رخ داده است . پس به کاربر یک ایمیل میفرستد .


```
def last_event():

    mydb = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        port = 3308 ,
        database="surveillance_system"
    )

    mycursor = mydb.cursor()
    sql = "SELECT MAX(created_at) AS last_event FROM events"
    mycursor.execute(sql)
    a = mycursor.fetchall()

    return a[0][0]
```

در تابع بالا ابتدا به دیتا بیس متصل میشویم سپس کوئری مربوط به پیدا کردن آخرین رخداد را اجرا میکنیم.

```
lock = 0 # send email once a day
while 1:
    now = datetime.datetime.now()
    if(now - last_event() > datetime.timedelta(days=DELTA_TIME) and lock == 0):
        print('email')
        send_email()
        print('email')
        lock = 1

    if(now - last_event() > datetime.timedelta(days=DELTA_TIME + 1) and lock == 1):
        lock = 0
    time.sleep(5)
```

در کد بالا یک متغیر تعریف شده برای اینکه با مشاهده یک خرابی مکرراً پشت سر هم گزارش خرابی ندهد. مثلاً روزی یکبار گزارش دهد. همانطور که مشاهده میشود اگر اختلاف زمانی، از تایم تعریف شده بیشتر بود ایمیل ارسال میشد و پرچم قفل فعال شده تا مکرراً ایمیل ارسال نشود و اگر یک روز از این ماجرا گذشت آنگاه قفل ما غیر فعال شده و دوباره در صورت عدم رفع مشکل یک ایمیل دیگر به کاربر ارسال میشود و این چرخه همینطور ادامه میابد.

معماری *mvc* :

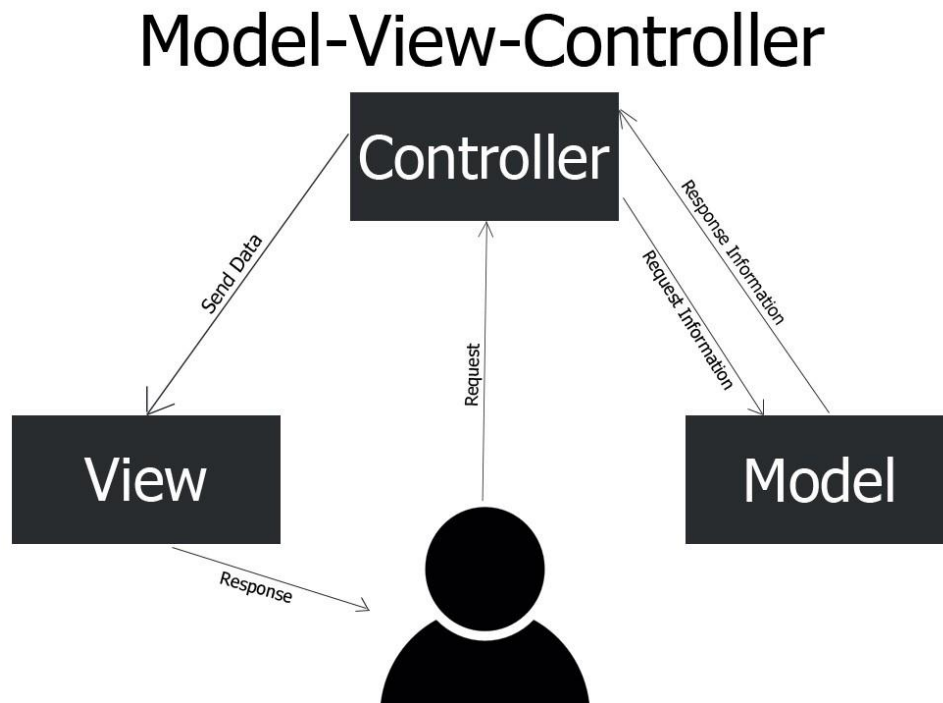
Model , view , controller یا به اختصار **mvc** نوعی روش معماری نرم‌افزار است که در توسعه وب اپلیکیشن‌ها بسیار پرکاربرد است و ورود آن به صنعت توسعه نرم‌افزار به دهه ۱۹۷۰ بازمی‌گردد. امروزه فریمورک‌های مطرحی که در توسعه نرم‌افزارهای کوچک و بزرگ مورد استفاده قرار می‌گیرند مبتنی بر این معماری‌اند. به طور خلاصه، می‌توان گفت که هدف از معماری سه‌لایه ام‌وی‌سی مجزاسازی بخش‌های مختلف نرم‌افزار از یکدیگر است به طوری که بتوان هر کدام از این بخش‌ها یا ماژول‌ها را به صورت مستقل توسعه داد و در نهایت مابین آن‌ها ارتباط برقرار ساخت. به عبارت دیگر، چندین و چند دولوپر مختلف می‌توانند روی پروژه‌هایی با این نوع معماری کار کنند بدون آنکه در کار یکدیگر تداخلی ایجاد نمایند.

Model را به نوعی می‌توان به منزله مغز اپلیکیشن در نظر گرفت به طوری که اصطلاحاً **Business Logic** یا به عبارتی «آنچه اپلیکیشن به خاطرش توسعه یافته است» در این لایه طرح‌ریزی می‌شود. مسلماً نیاز به توضیح نیست که مثلاً در یک وب اپلیکیشن بخشی از منطق نرم‌افزار مرتبط با ارتباط با دیتابیس به منظور انجام عملیات **CRUD** است که تسک‌هایی از این دست در مدل عملی می‌گردند.

نکته سرواژه **CRUD** برگرفته از کلمات **Create, Read, Update, Delete** است که به ترتیب به منظور «ثبت»، «فراخوانی»، «به‌روزرسانی» و «حذف» داده‌ها از دیتابیس مورد استفاده قرار می‌گیرند. **View** همان‌طور که از نام آن مشخص است، این وظیفه را دارا است تا دیتایی که در مدل ساخته و پرداخته شده را در معرض دید کاربران وب اپلیکیشن قرار دهد و به عبارتی می‌توان گفت که همان **User Interface** یا به اختصار **UI** است.

Controller در این معماری سه‌لایه نقش واسط را دارا است به طوری که ریکوئست‌ها را از بخش ویو گرفته و در اختیار مدل قرار می‌دهد و پس از آنکه مدل پردازش‌هایش را روی ریکوئست (درخواست) ورودی انجام داد،

ریسپانس (پاسخ) را مجدد در اختیار کنترلر قرار داده و کنترلر هم پاسخ نهایی را در اختیار ویو می‌گذارد تا در معرض دید کاربران قرار دهد.



: Jobs and queue in laravel

سیستم پیاده سازی پردازش به صورت صف یا Queue System یک مفهوم عمومی در برنامه نویسی است که در لاراول نحوه پیاده سازی آن در Queue Backend های مختلف نظیر Redis , DataBase کاملاً مشابه هم و پیکره بندی آن هم نزدیک به هم است.

Queue یک صف انتظار از پردازش ها (Job یا Listeners) برای اجرا می باشد. به گونه ای که پردازش ها در زمان اجرا به صف انتظار می روند و سرعت اجرای متد را کم نمی کنند و در Background برنامه اجرا می شوند.

چون در پشت برنامه Queue ها به اجرا در می آیند و تا حدودی از دید کاربر پنهان می شوند به آن ها Background Processing نیز می گوییم.

معمولاً در هر Queue تعدادی Job وجود دارد. این Job ها در زمان اجرا به دو دسته تقسیم می شوند.

۱- **Failed_jobs** : پروسه ها (Job) هایی که در زمان اجرا به خطا خورده باشند. این پروسه ها ذخیره و قابل پیگیری می باشند. در **database** جدولی با نام **failed_jobs** برای این مساله داریم.

۲- **Jobs** : پروسه هایی که در صف انتظار برای اجرا می باشند. در **database** ما جدولی با نام **jobs** داریم.

QUEUE_CONNECTION یا سرویسی که ما قرار است از آن جهت پیاده سازی سیستم **Queue** استفاده کنیم می باشد که به پیرو آن سایر تنظیمات مانند **FailedJobs** و **Jobs** نیز انجام می شود. برخی سرویس پیاده سازی **Queue** یا **QUEUE_CONNECTION** که در فایل **queue.php** به آن ها اشاره شده است:

Sync : انجام همزمان پردازش ها . در این مورد عملا ما از تکنیک **queue** استفاده نمی کنیم و صرفا برای تست می باشد. **Database** : برای استفاده از پایگاه داده . **beanstalkd** و **sqs** و **redis** هر **queue connection** می تواند **queue attribute** های مربوط به خود را داشته باشد.

تعریف پروسه و قرار دادن آن ها در صف انتظار (**queue**) درواقع ما پروسه های زمانبر را در قالب **job** تعریف می کنیم و در جایی که به آن ها نیاز داشته باشیم از آن ها استفاده می کنیم.

به طور مثال ما می خواهیم به ازای برخی اعمال کاربر ایمیلی حاوی اطلاعات کاربر جاری به مدیر وب سایت بزنیم. این برخی اعمال کاربر یعنی مکان هایی که ما قرار است **job** را **dispatch** کنیم. این مکان ها در میان متد های **controller** می تواند باشد.

برای فراخوانی مستقیم **Job** باید آن را بوسیله کلاسش **dispatch** کنیم. این فراخوانی می تواند در هر کجای برنامه نظیر **controller** یا **Route** باشد. گوش دادن به **queue** یعنی وقتی ما سیستم **Queue** را پیاده سازی کردیم هر **Job** به صف **Queue** می رود و در آنجا که ما به آن **Background** می گوئیم منتظر برپایی می باشد.

کنترلر :

مهم ترین بخش در این سیستم بخش کنترلر میباشد . این بخش با فریمورک لاراول پیاده سازی شده است . در این بخش درخواست ثبت و ذخیره تصویر را از بخش سنسور دریافت کرده و بعد از بررسی آن ، آنرا در دیتابیس ذخیره کرده و پس از تشکیل یک صف ، ایمیل مربوط به آنرا در صف ایمیل ها قرار میدهد . زمانی که نوبت هر

ایمیل شد آنرا به سرویس ارسال ایمیل خارجی میفرستد و نهایتاً این موضوع که ایمیل برای رخداد مربوطه به کاربر ارسال شده ، در دیتابیس ذکر میشود .

روتها :

آدرس ها و لینک های هر بخش از وبسایت به کنترلر هر بخش در اینجا متصل میشود . درواقع برای هر فعالیت در وبسایت باید یک آدرس در مرورگر داشته باشیم که در فریمورک لاراول آنها را در این دایرکتوری ثبت میکنیم .

ابتدا روت مربوط به api ای که از طریق آن کنترلر سیستم به بخش سنسور متصل است را مشاهده میکنید :

```
Route::post('/event/create', [EventController::class, 'store'])->>name('event_create');
```

اکنون به بررسی روت های داخلی بخش کنترلر سیستم میپردازیم :

```
Route::get('/', function () {
    return view('back.index');
})->middleware('auth')->name('login');

Auth::routes();

Route::prefix('actions')->middleware('auth')->group(function() {
    Route::get('/', [EventActionController::class, 'index'])->>name('actions');
    Route::get('/create', [EventActionController::class, 'create'])->>name('actions.create');
    Route::post('/store', [EventActionController::class, 'store'])->>name('actions.store');
    Route::get('/edit/{action}', [EventActionController::class, 'edit'])->>name('actions.edit');
    Route::put('/update/{action}', [EventActionController::class, 'update'])->>name('actions.update');
    Route::delete('/destroy/{action}', [EventActionController::class, 'destroy'])->>name('actions.destroy');
```

```
});

Route::prefix('events')->middleware('auth')->group(function(){
    Route::get('/',[EventController::class , 'index']->name('events'));
    Route::get('/no_action',[EventController::class , 'no_action']->name('events.no_action');
    Route::get('/show-image/{image}',[EventController::class , 'show_image']->name('events.show_image');
    Route::delete('/destroy/{event}',[EventController::class , 'destroy']->name('events.destroy');
    Route::get('/date_page',[EventController::class , 'date_page']->name('date_page');
    Route::post('/date_query',[EventController::class , 'date_query']->name('date_query');
});
```

در ابتدا این نکته ذکر شده که کاربر بایستی به صفحه لاگین برود .

بخش authentication در این فریمورک بصورت آماده پیاده سازی شده و فقط با یک خط کد بایستی روت های آنرا به برنامه معرفی کنیم .

در دو تکه کد بعدی اعمال crud بر روی مدل event و action آدرس سازی شده اند .

مدل ها :

مدل ها در لاراول در پوشه models قرار دارند . ما سه مدل اصلی به نام های event_action و event و user داریم . مدل user بصورت پیشفرض قرار دارد و ما فقط دو مدل دیگر را بررسی میکنیم . در لاراول تمامی مدل هایی که میسازیم از کلاس اصلی Model ارث بری میکنند که ما به بررسی کلاس مدل نمیپردازیم .

یکی از اهداف مدل ها نگاشت کلاس ها بر روی دیتابیس است که در لاراول این کار بسیار راحت شده است . همانطور که در بخش تحلیل دیتابیس دیدیم بین دو کلاس event و event_action یک رابطه یک به چند وجود دارد که علاوه بر دیتابیس باید این موضوع را در مدل ها هم ذکر کنیم تا بتوانیم از این روابط در طول برنامه به سادگی استفاده کنیم .

در کد زیر این موضوع ذکر شده که هر `event_action` میتواند چند `event` داشته باشد. در واقع از یک نوع رخداد میتواند چندین و چند رخداد اتفاق بیوفتد که به معنای رابطه `one to many` میباشد. پس درون کلاس `event_action` باید کد زیر را قرار دهیم تا بتوانیم به `event` های مربوطه دسترسی داشته باشیم:

```
public function events() {  
    return $this->hasMany(Event::class, 'type', 'event');  
}
```

همچنین در کد فوق ذکر شده که فیلد `event` کلید خارجی در جدول دیگر میباشد.

حال باید بررسی کنیم که هر `event` متعلق از به یک `action_event` پس به کلاس `event` رفته و کد زیر را قرار میدهیم:

```
public function event_action() {  
    return $this->belongsTo(event_action::class, 'type', 'event');  
}
```

در کد فوق این نکته ذکر شده که فیلد `type` کلید خارجی است و به فیلد `event` در جدول دیگه ارجاع دارد.

کنترلر ها :

در این بخش منطق برنامه و نحوه ارتباطات مدل ها و ویوها پیاده سازی شده است. کنترلر ها در پوشه کنترلر قرار دارند. ما دو کنترلر اصلی بجز کنترلر های مربوط به بخش `authentication` داریم که آنها را بررسی میکنیم.

ابتدا کنترلر مربوط به `event_action`:

```
public function index()  
{  
    $actions = event_action::all();  
    return view('back.actions.actions', compact('actions'));  
}
```

در متود فوق ابتدا تمام اکشن ها از دیتابیس بازیابی شده و در متغیر ریخته میشود سپس همین متغیر به `view` مربوطه فرستاده میشود و در آنجا فیلد های مختلف در تگ های مربوط به خود جایگذاری میشوند.

```
public function create()
{
    return view('back.actions.create');
}
```

متود فوق فرم مربوط به ساخت اکشن ها را به کاربر نشان میدهد .

```
public function store(Request $request)
{
    $request->validate([
        'event'=>'required',
        'action'=>'required'
    ]);

    $action = new event_action();
    try {

        $action->create($request->all());
    }
    catch(Exception $ex){
        $msg1='کنید اقدام مجددا لطفا شد مواجه مشکل با سازی ذخیره';
        return redirect(route('actions.create'))->with('save_error',$msg1);
    }

    $msg='شد انجام موفقیت با جدید بندی دسته ذخیره';
    return redirect(route('actions'))->with('success',$msg);
}
```

متود فوق مقادیر ثبت شده توسط کاربر را دریافت ، آنها را اعتبار سنجی کرده و در صورت معتبر بودن ، در دیتابیس ذخیره میکند و پیام های مناسب را به کاربر نشان میدهد .

```
public function edit(event_action $action)
{
    return view('back.actions.edit',compact('action'));
}
```

متود فوق فرم مربوط به آپدیت کردن مقادیر را به کاربر نمایش میدهد .


```
public function update(Request $request, event_action $action)
{
    $request->validate([
        'event'=>'required',
        'action'=>'required'
    ]);

    try {
        $action->update($request->all());
    }
    catch(Exception $ex){
        $msg1='کنید اقدام مجددا لطفا شد مواجه مشکل با سازی ذخیره';
        return redirect(route('actions.edit'))->with('save_error',$msg1);
    }

    $msg='شد انجام موفقیت با ویرایش';
    return redirect(route('actions'))->with('success',$msg);
}
```

متود فوق اطلاعات را از کاربر گرفته و پس از اعتبار سنجی آنها را در دیتابیس آپدیت میکند و پیغام های مناسب را چاپ میکند .

```
public function destroy(event_action $action)
{
    $action->delete();
    $msg='شد انجام موفقیت با حذف';
    return redirect(route('actions'))->with('success',$msg);
}
```

متود فوق نیز یک اکشن را دریافت کرده و پس از یافتن آن در دیتابیس آنرا حذف کرده و پیغام مناسب را به کاربر نمایش میدهد .

حال میپردازیم به بررسی متود های کنترلر مربوط به event :

```
public function index()
{
    $events = Event::all();
}
```

```
return view('back.events.events', compact('events'));
}
```

متود فوق تمامی رخداد هارا از دیتابیس بازیابی کرده و در متغیری ریخته و نهایتا آنرا به view مربوطه اش ارسال میکند تا از آن در تگ های مناسب استفاده شود .

```
public function no_action()
{
    $events = Event::where('done', false)->get();
    return view('back.events.events', compact('events'));
}
```

متود فوق تمام رخداد هایی که برای آنها اقدامی نشده و ایمیلی ارسال نشده را از دیتابیس بازیابی کرده و آنها را به view مربوطه ارسال میکند .

```
public function show_image($image)
{
    return view('back.events.show_image', compact('image'));
}
```

هنگامی که کاربر میخواهد تصویر یک رخداد را بصورت بزرگ و واضح مشاهده کند این متود فراخوانی میشود .

```
public function store(Request $request)
{
    $request->validate([
        'type'=>['required'],
        'image'=>['required'],
    ]);

    $event = new Event;

    $event->create([
        'type'=>$request->type,
        'image' => $request->image,
        'done' => false,
    ]);
}
```

```
]);

$content = event_action::where('event',$request->type)-
>first()->action;

SendEmailJob::dispatch($request->type , $request->image ,
$content);

return response()->json([
    'message' =>$request->type.'Event created successfully'
] , 201);
}
```

هنگامی که بخش سنسور یک رخداد را شناسایی میکند درخواست خود را به این متود ارسال میکند . ابتدا درخواست دریافت شده اعتبار سنجی میشود سپس آنرا در جدول مربوط به رخداد ها ذخیره میکند . سپس متن ایمیل مربوط به این رخداد را بازیابی کرده و job مربوط به ارسال ایمیل را فراخوانی میکند . نهایتاً هم پاسخ موفقیت آمیز بودن را به بخش سنسور میفرستد .

```
public function sendEmail($request)
{
    Mail::to('farhad@gmail.com')->send(new LogMail($request-
>type));
    return response()->json([
        'message' =>$request->type.'Event created successfully'
    ] , 201);
}
```

در این بخش از خاصیت کار با ایمیل ها که در لاراول تعبیه شده استفاده میکنیم و یک متود برای ارسال ایمیل به آدرس ایمیل کاربر مینویسیم .

```
public function date_page()
{
    return view('back.events.date_events');
}
```

متود فوق فرم مربوط به دریافت دو تاریخ از کاربر را به نمایش در میآورد .

```
public function date_query(Request $request)
{
    $from = $request->from;
    $to = $request->to;
    $events = Event::whereBetween('created_at', [$from, $to])->get();
    return view('back.events.events', compact('events'));
}
```

متود فوق دو تاریخ را دریافت کرده و تمامی رخدادهای بین این دو تاریخ را بازیابی کرده و به صفحه **view** مربوطه ارسال میکند .

```
public function destroy(Event $event)
{
    if ($this->removeImage($event->image)) {
        $event->delete();
        $msg='    شد حذف موفقیت با    ';
        return redirect(route('events'))->with('success', $msg);
    }
}
```

متود فوق رکورد مربوط به رخداد را از پایگاه داده حذف میکند . نکته ی مهمی که اینجا وجود دارد این است که باید به هنگام حذف سطر مربوطه از دیتابیس ، تصویر رخداد مربوطه هم از دایرکتوری تصاویر حذف شود .

```
public function removeImage($img)
{
    if(file_exists(public_path('images/' . $img))) {
        unlink(public_path('images/' . $img));
        return true;
    }
    return false;
}
```

متود فوق نام یک تصویر را از ورودی گرفته و آنرا جستجو کرده و نهایتاً حذف میکند . این موضوع که ما حذف تصویر را در متود جداگانه نوشته ایم به افزایش انسجام برنامه (Cohesion) کمک میکند .

بخش ارسال ایمیل (job and queue):

```
public function handle()
{
    //      sleep(30);

    Mail::to('farhad@gmail.com')->send(new LogMail($this->type
, $this->content));

    DB::table('events')->where('image',$this->image)->update([
        'done' => true
    ]);
}
```

در این متود ما job مربوط به ارسال ایمیل را handle میکنیم . ابتدا فرستادن ایمیل را فراخوانی کرده و سپس فیلد مربوط به اقدام شدن رخداد را تیک میزنیم بدین معنا که ایمیل مربوط به این رخداد ارسال شده است.

حال به بخش ایمیل ها رفته و کد زیر را مینویسیم :

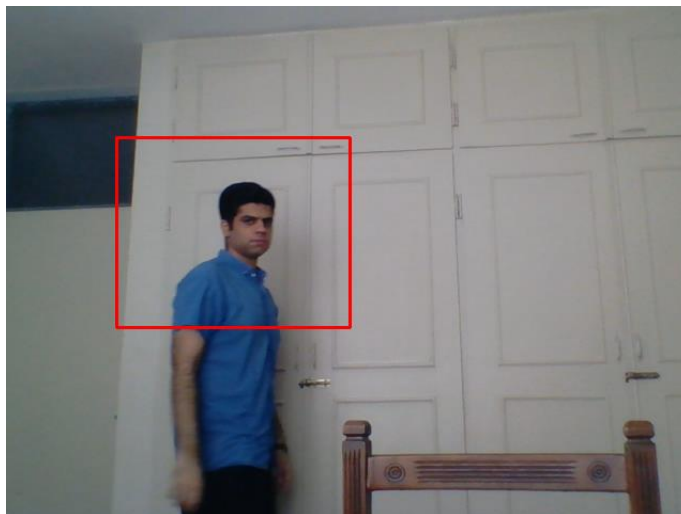
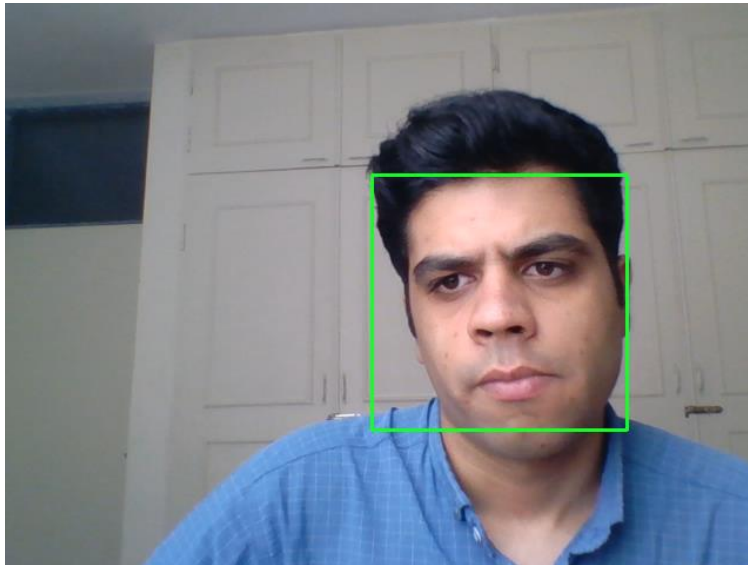
```
public function build()
{
    return $this->view('Mail.emailBody')->subject($this->type) -
>with('content',$this->content);
}
```

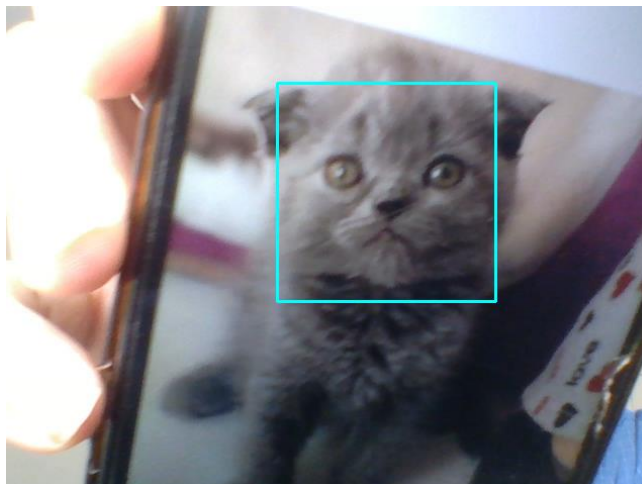
در این متود ما فعالیت مربوط به ایمیل را پیاده سازی میکنیم . درواقع متن ایمیل و عنوان ایمیل را به سرویس خارجی ایمیل میفرستیم .

• اجرا:

بخش سنسور :

برنامه مربوط به بخش سنسور را اجرا میکنیم . این برنامه به کمک دوربین رخدادهای شناسایی میکند نمونه هایی از سه نوع رخداد را مشاهده میکنید :



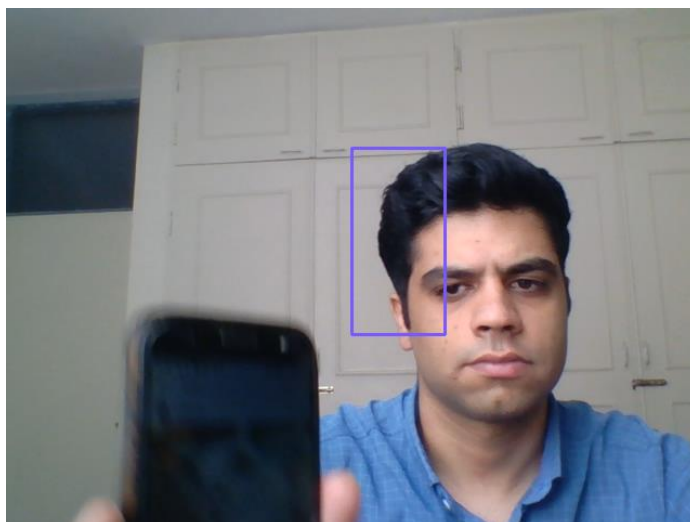


بدلیل در اختیار نبودن گربه ، از تصویر گربه در صفحه موبایل استفاده شده است .

پیغام های چاپ شده در کنسول هم بصورت زیر میباشد :

```
The pastebin URL is:{"message":"faceEvent created successfully"}  
[[319 129 86 172]]  
True  
The pastebin URL is:{"message":"bodyEvent created successfully"}  
[[268 76 216 216]]  
True  
The pastebin URL is:{"message":"catEvent created successfully"}  
[[312 146 217 217]]  
True
```

البته این سامانه تشخیص ،خطاهایی هم دارد . برای مثال تصویر زیر را بعنوان بدن انسان در نظر گرفته :



بخش کنترلر :

ابتدا سایت را serve میکنیم :

```
E:\books\مرتب\8\ژورنال\prj\controller>php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Thu Sep 2 16:44:05 2021] 127.0.0.1:3238 [200]: /favicon.ico
[Thu Sep 2 16:44:08 2021] 127.0.0.1:5066 [200]: /assets/vendors/iconfonts/mdi/css/materialdesignicons.min.css
[Thu Sep 2 16:44:08 2021] 127.0.0.1:5097 [200]: /assets/vendors/iconfonts/ionicons/css/ionicons.css
[Thu Sep 2 16:44:08 2021] 127.0.0.1:10911 [200]: /assets/vendors/iconfonts/typicons/src/font/typicons.css
```

صفحه مدیریت سایت بعد از لاگین بصورت زیر میباشد :







The screenshot shows the Staradmin dashboard. The main content area displays a table titled 'مدیریت رخدادها' (Event Management). The table has columns for 'نوع' (Type), 'تصویر گرفته شده' (Image Taken), 'آثار به اینستاگرام' (Effects to Instagram), 'تاریخ رخداد' (Event Date), and 'حذف' (Delete). The table lists several events, each with a delete button labeled 'حذف'.

نوع	تصویر گرفته شده	آثار به اینستاگرام	تاریخ رخداد	حذف
cat		1	09-09-39 2021-08-29	حذف
face		0	12-11-10 2021-09-02	حذف
body		0	12-12-17 2021-09-02	حذف
cat		0	12-12-55 2021-09-02	حذف
face		0	12-13-10 2021-09-02	حذف
face		0	12-13-30 2021-09-02	حذف
body		0	12-13-39 2021-09-02	حذف

The sidebar on the right contains the following links: 'مدیریت اصلی' (Main Management), 'حساب کاربری' (User Account), 'ایست رخداد و اقدامات' (Event and Actions), 'پرس و جو' (FAQ), 'ایست ظاهر رخدادها' (Event Appearance), 'ایست رخدادهای اقدام نشده' (Event Actions Not Done), 'بستجو بر اساس تاریخ' (Search by Date), and 'خروج' (Logout).

اکنون با زدن نوشتن دستور شروع صف ، مواردی که ایمیل آنها ارسال نشده شروع به ایمیل شدن میشود .








درواقع اگر این دستور را بنیم برای همیشه هر رخدادی که اتفاق بیفتد ایمیل آنرا بصورت خودکار ارسال میکند. در این گزارش برای توضیح بهتر، ابتدا ما آنرا متوقف کردیم تا ایمیل های ارسال نشده را مشاهده کنیم:

نوع	تصویر گرفته شده	اندام به اپلود	تاریخ رخداد	حالت
face		0	12:11:10 2021-09-02	مخفی
body		0	12:12:17 2021-09-02	مخفی
cat		0	12:12:55 2021-09-02	مخفی
face		0	12:13:10 2021-09-02	مخفی
face		0	12:13:30 2021-09-02	مخفی
body		0	12:13:39 2021-09-02	مخفی

اکنون دستور صف را اجرا میکنیم:

```
E:\books\م رت 8\ژورپ\ایسان شراک\prj\controller>php artisan queue:work
[2021-09-02 12:47:29][133] Processing: App\Jobs\SendEmailJob
[2021-09-02 12:47:42][133] Processed: App\Jobs\SendEmailJob
[2021-09-02 12:47:42][134] Processing: App\Jobs\SendEmailJob
[2021-09-02 12:47:48][134] Processed: App\Jobs\SendEmailJob
[2021-09-02 12:47:48][135] Processing: App\Jobs\SendEmailJob
[2021-09-02 12:47:54][135] Processed: App\Jobs\SendEmailJob
[2021-09-02 12:47:54][136] Processing: App\Jobs\SendEmailJob
```

با اجرای این دستور ایمیل ها ارسال میشوند.

نوع	تصویر گرفته شده	اندام به اپلود	تاریخ رخداد	حالت
cat		1	09-09:39 2021-08-29	مخفی
face		1	12:11:10 2021-09-02	مخفی
body		1	12:12:17 2021-09-02	مخفی
cat		1	12:12:55 2021-09-02	مخفی
face		1	12:13:10 2021-09-02	مخفی
face		1	12:13:30 2021-09-02	مخفی
body		1	12:13:39 2021-09-02	مخفی

اکنون جستجو بر اساس رخدادها را بررسی میکنیم:

پروژه پایانی کارشناسی

منوی اصلی

- حساب کاربری
- لیست رخداد و اقدام ها
- پرس و جو
- لیست تمام رخداد ها
- لیست رخداد های اقدام نشده
- جستجو بر اساس تاریخ
- خروج

از: 08/19/2021

تا: 09/01/2021

Submit

نتیجه بصورت زیر خواهد شد :

تاریخ رخداد	تاریخ رخداد	تاریخ رخداد	تاریخ رخداد	تاریخ رخداد	تاریخ رخداد
cat	09-09-39 2021-08-29	1	تصویر گرفته شد	حذف	حذف

حالا به بررسی و مشاهده و تغییر متن ایمیل ها میرویم :

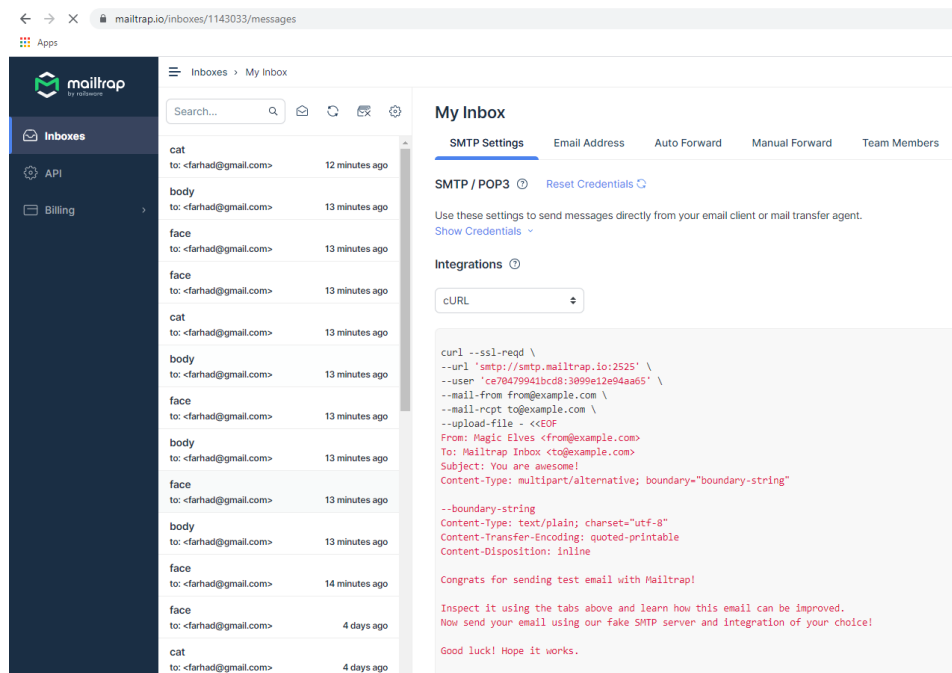
مدیریت رخداد - اقدام	مدیریت رخداد - اقدام	مدیریت رخداد - اقدام	مدیریت رخداد - اقدام	مدیریت رخداد - اقدام	مدیریت رخداد - اقدام
cat	سلام یک گریه دیده شد خسته نباشید ...	حذف	حذف	حذف	حذف
body	سلام یک انسان دیده شد :)))	حذف	حذف	حذف	حذف
face	سلام چهره یک انسان دیده شد	حذف	حذف	حذف	حذف

همانطور که مشاهده میشود قابلیت حذف و ویرایش هم برای آن قرار داده شده است .

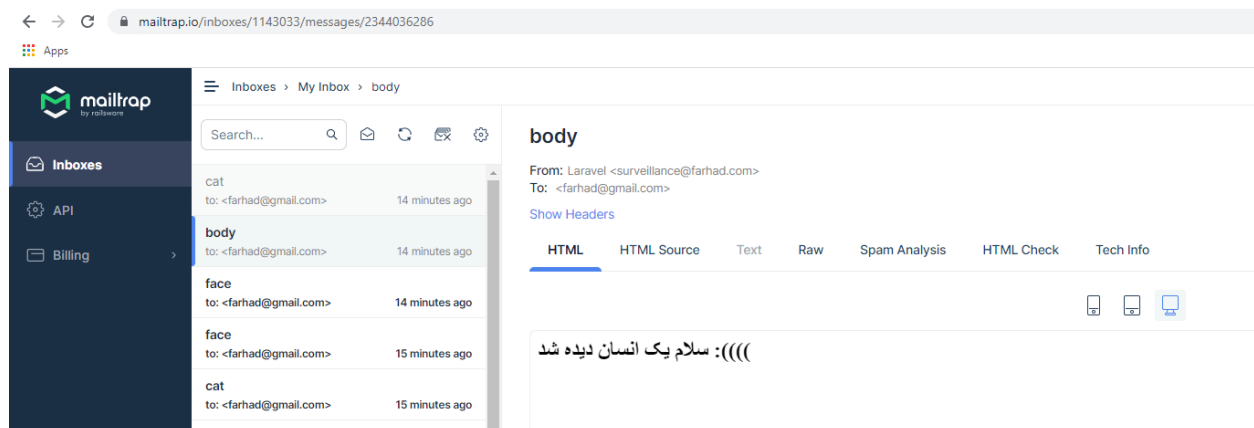
بخش اقدام (ایمیل) :

برای اینکه پروژه بر روی هاست محلی است و در مرحله ی develop میباشد. ما از سایت <https://mailtrap.io> استفاده میکنیم. این سایت امکانی را برای ما فراهم میکند تا بخش ایمیل نرم افزار های خود را اعتبار سنجی کنیم.

تصویر زیر لیست ایمیل های ما را در باکس ایمیل ها نشان میدهد :



برای مثال متن یک ایمیل را باز میکنیم :



• آدرس فایل ها و کدها :

https://github.com/farhadhsn8/surveillance_system

• مراجع :

<https://laravel.com/docs/8.x/authentication>

<https://laravel.com/docs/8.x/middleware>

<https://laravel.com/docs/8.x/controllers>

<https://laravel.com/docs/8.x/eloquent-relationships>

<https://laravel.com/api/8.x>

<https://docs.python.org/3>

<https://www.e-consystems.com/blog/camera/how-to-access-cameras-using-opencv-with-python>

https://docs.opencv.org/4.5.1/dd/d43/tutorial_py_video_display.html

<https://gist.github.com/tedmiston/6060034>

<https://mailtrap.io/inboxes/1143033/messages/2344036286>

<https://sokanacademy.com/academy/courses/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D9%85%D8%B9%D9%85%D8%A7%D8%B1%DB%8C-mvc/%D9%85%D9%82%D8%AF%D9%85%D9%87-106/%D9%85%D8%B9%D9%85%D8%A7%D8%B1%DB%8C-mvc-%DA%86%DB%8C%D8%B3%D8%AA>

<https://gnutec.net/queue-job-in-laravel>

<http://www.tahlildadeh.com/ArticleDetails/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D8%B3%D8%B1%D9%88%DB%8C%D8%B3-Queue-%D9%88-%D8%B5%D9%81-%D8%A8%D9%86%D8%AF%DB%8C-%D8%B9%D9%85%D9%84%DB%8C%D8%A7%D8%AA-%D8%AF%D8%B1-Laravel>