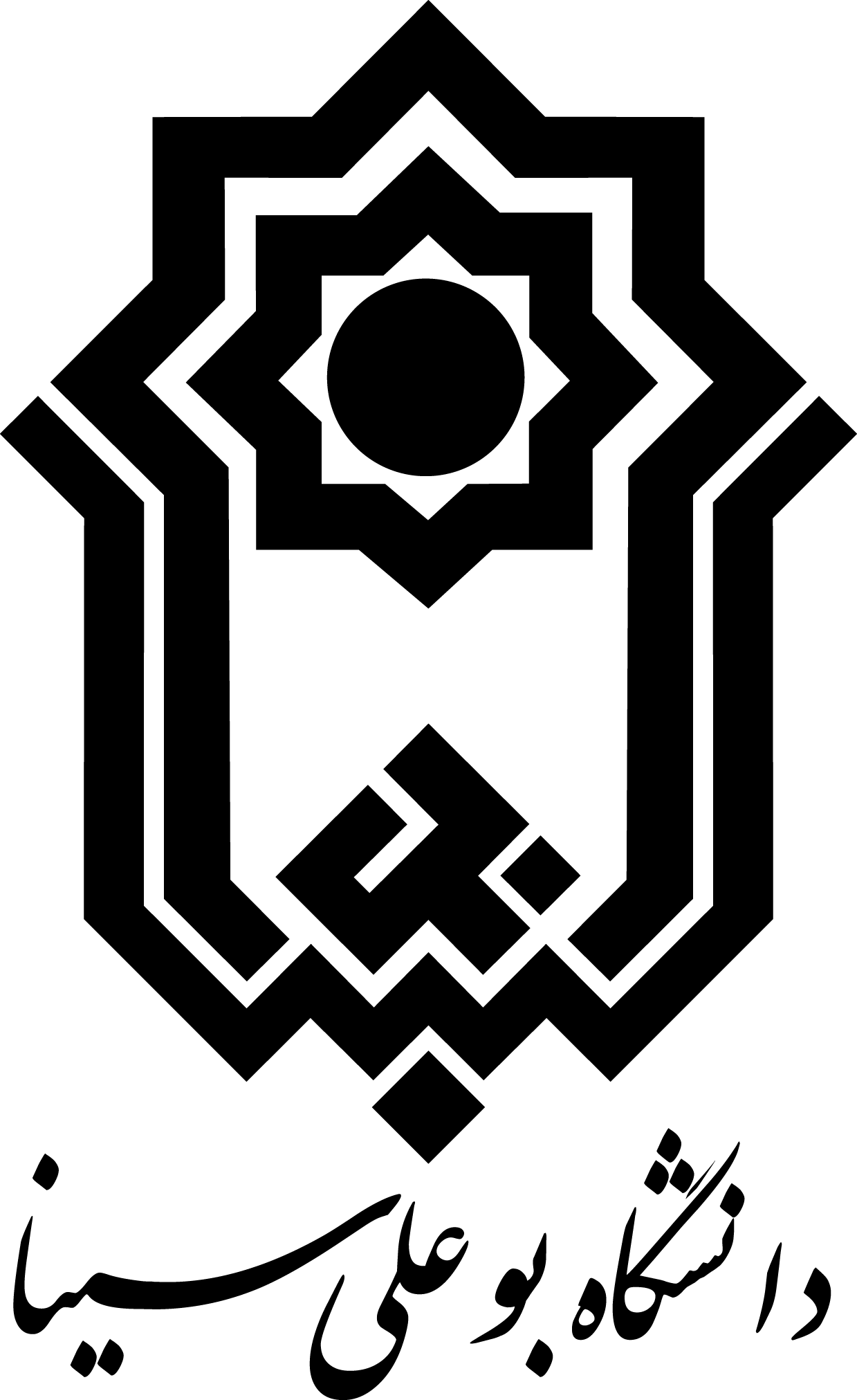
به نام خدا



نام و نام خانوادگی:سید فرهاد حسینی

شماره دانشجویی : 9612358016

نام درس : پروژه کارشناسی

استاد مربوطه : دکتر منصوری زاده

موضوع : پیاده سازی سیستم پایش محیطی

اینجانب سید فرهاد حسینی دانشجوی ‌رشته‌مهندسـی‌کامپیوتر‌دانشـگاه بوعلی‌سینا‌از‌تاریخ‌1/04/1400 الی‌15/06/1400 روی‌پروژه‌ با‌ موضـوع‌ سیستم پایش محیطی زیر‌نظر‌استاد‌ راهنما‌ جناب دکتـر‌منصوری زاده فعالیت نموده ‌و‌بدین‌ وسیله‌ تعهد‌می‌کنم‌که‌مطالب‌این‌پایان‌نامه‌همگی نتیجه‌فعالیت‌و‌تحقیقات‌ اینجانب می‌باشد‌.

بدین وسیله‌ گواهی‌ می‌ گردد‌که‌آقا/خانم..............‌دانشجوی رشته............‌از‌تاریخ ..........‌تا‌ ...........زیر نظر اینجانب تحقیق و‌فعال‌یــت نمــوده‌و‌بدین وسیله ایشــان را‌واجد‌شرایــط‌ برای دریافــت مدرک‌کارشــناسی‌می‌ دانــم و با‌نمره‌ی ..........‌کار‌ایشان را‌تایید می‌ نمایم.‌

**فهرست :**

[ **چکیده :** 5](#_Toc81436553)

[ **توضیح صورت پروژه :** 6](#_Toc81436554)

[ویژگی ها و امکانات : 7](#_Toc81436555)

[ شماتیک بخش های سیستم : 8](#_Toc81436556)

[Deployment model : 9](#_Toc81436557)

[ **ابزارهای پیاده سازی :** 9](#_Toc81436558)

[واسط کاربری : 9](#_Toc81436559)

[حسگر : 9](#_Toc81436560)

[کنترلر : 10](#_Toc81436561)

[polling : 10](#_Toc81436562)

[ **Usecase diagram :** 10](#_Toc81436563)

[ **تحلیل و طراحی دیتابیس :** 11](#_Toc81436564)

[ER Diagram : 12](#_Toc81436565)

[ **پیاده سازی :**................................................................................................................13](#_Toc81436566)

[حسگر : 13](#_Toc81436567)

[polling : 19](#_Toc81436568)

[معماری mvc : 20](#_Toc81436569)

[Jobs and queue in laravel : 22](#_Toc81436570)

[کنترلر : 23](#_Toc81436571)

[ بخش چهارم(polling) : 24](#_Toc81436572)

[ شماتیک بخش های سیستم : 24](#_Toc81436573)

[ Deployment model : 25](#_Toc81436574)

[ سیستم از دید کاربر : 25](#_Toc81436575)

[ جداول دیتابیس برنامه : 26](#_Toc81436576)

[1-جدول مربوط به کاربران 26](#_Toc81436577)

# **مقدمه :**

موضوع این پروژه پیاده سازی یک سیستم مانیتورینگ(پایش محیطی) در یک حالت عمومی است . system Surveillance ها به منظور کنترل رفتار ها و فعالیت ها در یک مجموعه استفاده میشود تا برخی رفتار ها که برای کاربر مهم هستند و احتمالا مخرب اند را سریع تشخیص دهیم و اقدام مناسب را انجام دهیم مثلا 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 صرفا 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 منظورتبلیغات موثرتر به عمل آورد .

# **توضیح صورت پروژه :**

هدف از این پروژه طراحی و پیاده سازی یک سیستم پایش محیطی است . این سیستم از 4 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 بعهده دارد . بخش چهارم نیز polling نام دارد . این بخش وظیف این را به عهده دارد که سامانه را بررسی کند و در صورت وجود اشکال یا خرابی آنرا به کاربر اطلاع دهد .

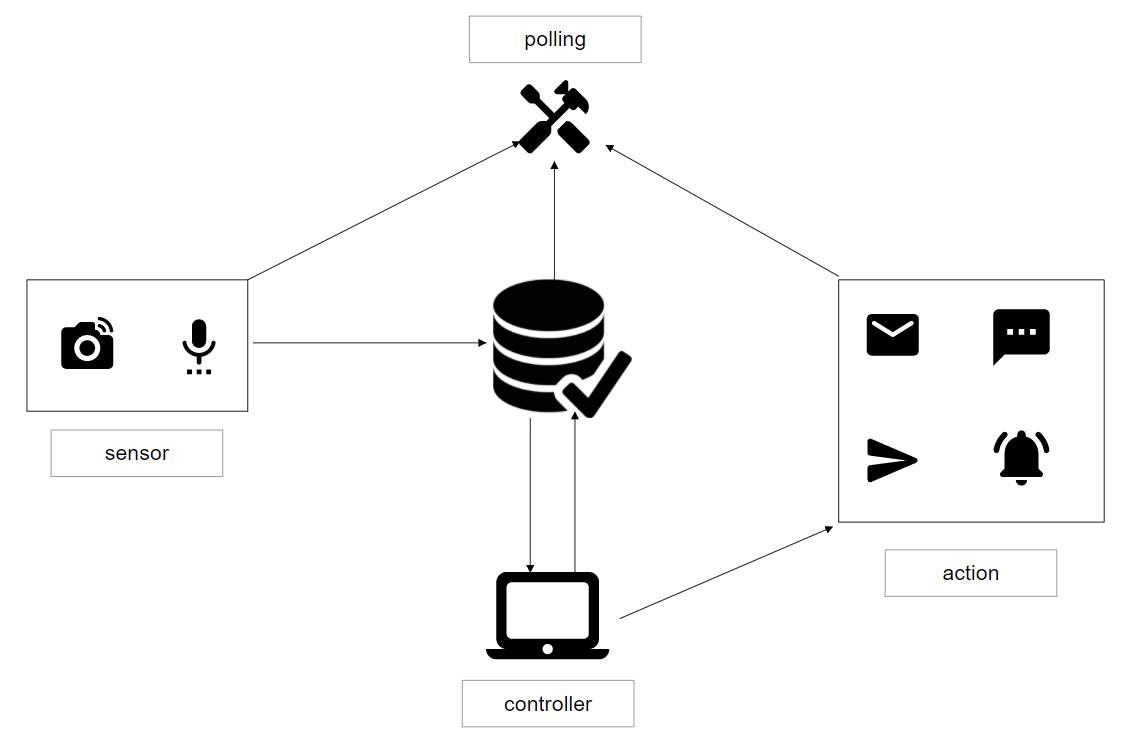
## ویژگی ها و امکانات :

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 .

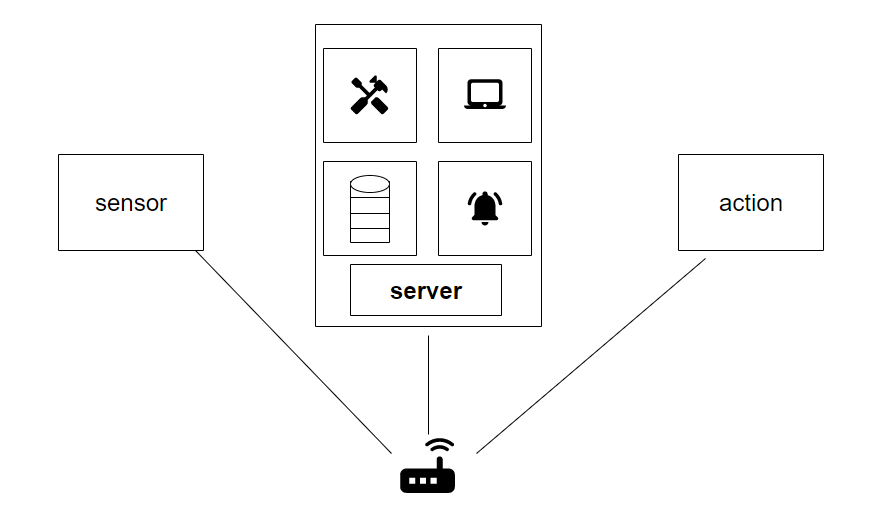
ویژگی دوم تنظیم و تغییر متن ایمیل ها میباشد . در واقع کاربر پس از ورود به سامانه در بخش رخداد ها و اقدام ها میتواند بر اساس نوع هر رخداد ، متن ایمیل آنرا به دلخواه خود تغییر دهد .

ویژگی بعدی اجرای برخی پرس و جو های پر کاربرد میباشد . کاربر میتواند لیست تمام رخداد ها را درخواست کند تا یه 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 یکنرم 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 .

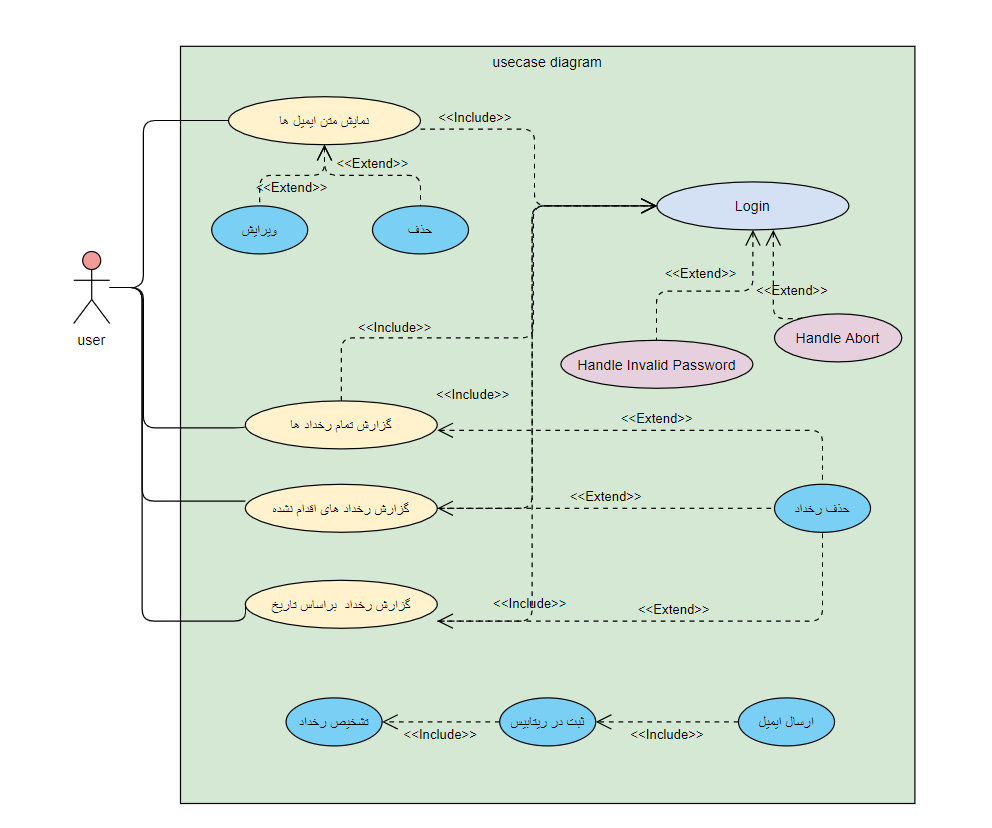
## کنترلر :

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 .

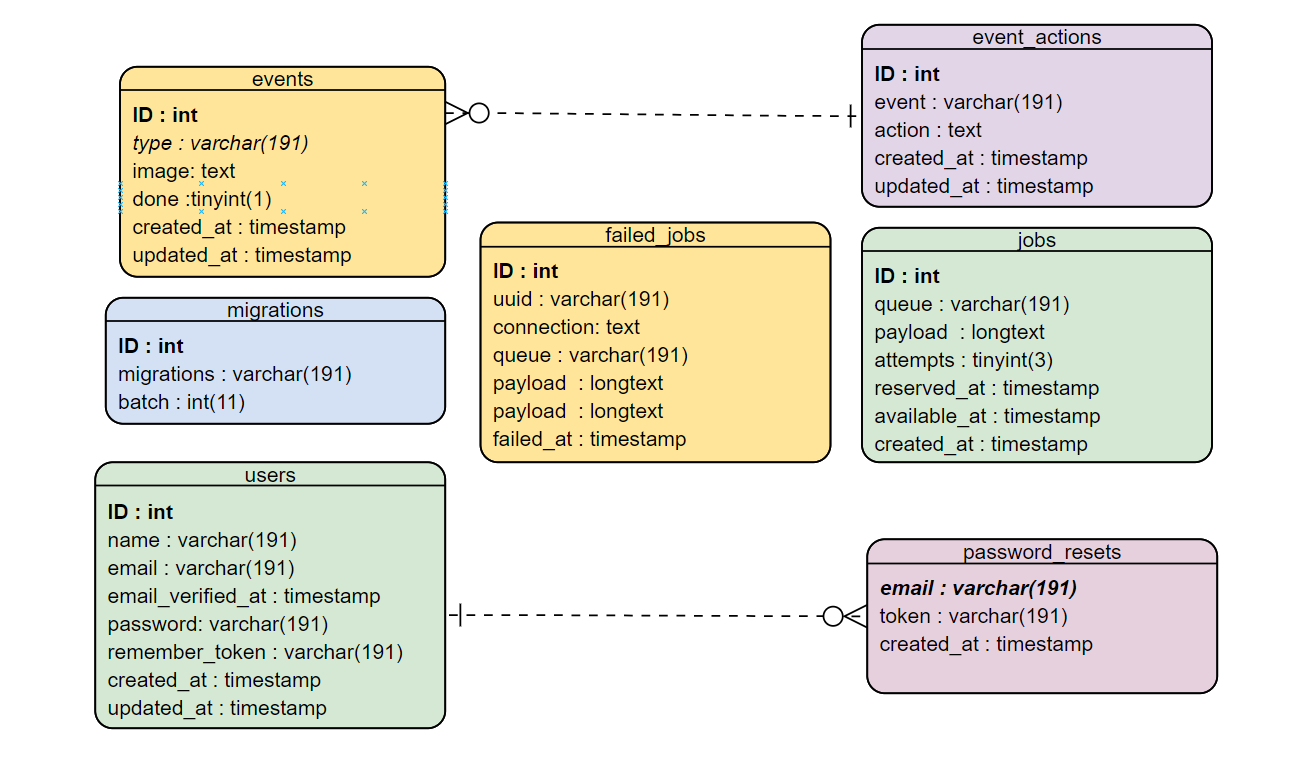
# **Usecase diagram :**



# **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 مطمئناگربه 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 مطمئنا 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 مطمئنا 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 4 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 1 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 3 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 ایا به تازگی مثلا 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 ویا نقصی در سیستم وجود داشت آنرا به کاربر گزارش دهد . اساس کار ما هم در این 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 اجرل 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 مکررا پشت سر هم گزارش خرابی ندهد . مثلا 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 مکررا 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  m**vc**نوعی روش معماری نرم‌افزار است که در توسعه وب اپلیکیشن‌ها بسیار پرکاربرد است و ورود آن به صنعت توسعه نرم‌افزار به دهه 1970 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 تَسک‌هایی از این دست در مدل عملی می‌گردند.

نکته سرواژه CRUD برگرفته از کلمات **U**pdate ،**R**ead ،**C**reate و **D**elete 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 **U**ser **I**nterface 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 DataBase , Redis کاملا 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 Proccessing نیز می گوییم.

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

1. 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.
2. 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 میخواهیم 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 گوییم منتظر برپایی می باشد.

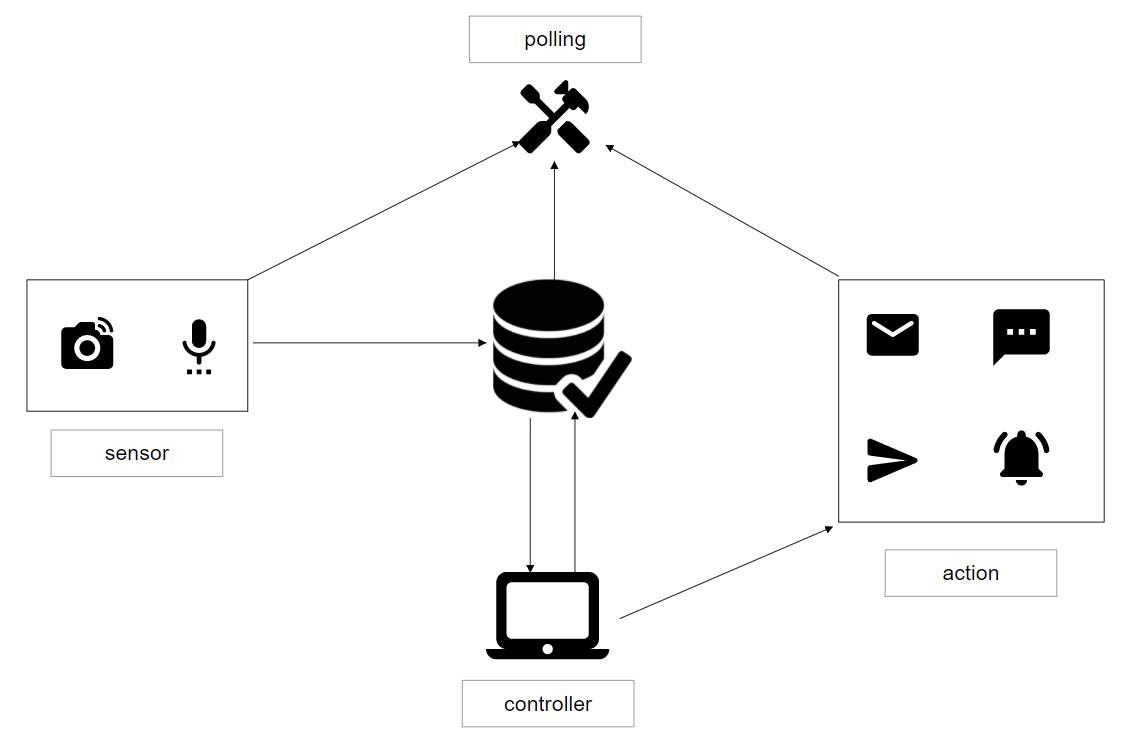
## کنترلر :

بعد از اینکه بخش کنترلر تغییری را در دیتابیس متوجه شد بر اساس آن رخداد اکشن مربوط به آن رخداد را صدا میزند . اکشن ها میتوانند فعالیت های مختفی مثل ارسال یک sms ، به صدا در آمدن آژیر ،ارسال ایمیل ، اجرا شدن برنامه ای خاص و ... باشند .

# بخش چهارم(polling) :

این بخش به هر سه بخش بالایی متصل است و ناظر بر صحت عمکرد آنها میباشد . مثلا اگر بعد از گذشت یک مدت زمان خاص فیلدی به دیتا بیس اضافه نشود این احتمال وجود دارد که سیستم بدرستی کار نمیکند و اتفاقات را در دیتا بیس ثبت نمیکند پس polling باید به نحوی این موضوع را به کاربر برساند .

# شماتیک بخش های سیستم :



# Deployment model :

# سیستم از دید کاربر :

کاربر ابتدا با صفحه احراز هویت روبرو میشود و پس از وارد کردن نام کاربری و رمز عبور وارد میشود .

کاربر در پنل کاربری خود 4 بخش کلی را مشاهده میکند .

بخش اول مربوط به تنظیمات اکانت است . کارهایی مثل تغییر نام کاربری یا تغییر رمز عبور و یا حتی افزودن کاربر جدید .

بخش دوم مربوط به افزودن یا حذف کردن event ها و action ها میباشد . به این صورت که بایستی دوتایی هایی از event , action را وارد کند . به این معنی که اگر سیستم متوجه event شد اکشن مربوطه اش را اجرا کند . در این بخش علاوه بر افزودن و حذف کردن ، کاربر میتواند عمل آپدیت را هم انجام دهد .

بخش سوم مربوط به برخی تنظیمات است مثل تعیین دوره زمانی چک کردن دیتابیس توسط سیستم ،که کاربر بایستی آنها را انجام دهد .

بخش چهارم هم مربوط به اجرای برخی از کوئری ها میباشد . مثلا مشاهده رخدادهایی که از فلان تاریخ تا فلان تاریخ اتفاق افتاده .

# جداول دیتابیس برنامه :

# 1-جدول مربوط به کاربران

2-جدول مربوط به event و action ها

3-جدول مربوط به رخدادها ( ماژول سنسور با مشاهده یک رخداد آنرا در این جدول ثبت میکند و کنترلر هم با همواره چک کردن این جدول اتفاقات جدید را متوجه میشود ) .

4 – جدول مربوط به تنظیمات