# 1. Общата картина

css

[Външен Хардуер (бариера, камери, код за разпознаване)] <----> [Django DRF сервер] <----> [Vue.js (мултиклиентски UI)]

Реализацията включва:

Backend: Django + DRF за основните CRUD операции, нов API endpoint-и за приемане на автоматични събития

Hardware Layer: Софтуер (Python скриптове или малък отделен service), който получава данни от бариерата и камерите

Обработка: Локално/на сървъра върви скрипт, който реже снимки, праща POST/PUT към Django чрез REST API

Frontend: Vue.js UI, който просто периодично обновява статуса от сървъра както сега

2. Външни хардуерни интеграции (Software Layer)

a) Инфрачервена бариера

Свързва се към компютър (например Raspberry Pi, Windows/Linux PC със изнесен вход, Serial/USB/IO).

Локален daemon/service (Python) слуша NC релето и при промяна (т.е. bariera is triggered) праща сигнал (HTTP POST) към DRF.

b) Камери (Dahua, ONVIF, RTSP)

Работиш с Python lib-ове като opencv, onvif, ffmpeg-python:

Една камера (за разпознаване) постоянно си „дърпа” кадър и го дава през AI модел (YOLO, EasyOCR) за номер.

Втора камера непрекъснато записва кадри и при събитие от бариерата записва текущия кадър/мини-клип като „фотофиниш”.

c) Интеграция между слоевете

Python service слуша бариерата → изчаква нов трайгър → изпраща снимки + timestamp + (евентуално получени номера) към endpoint на Django (например /api/finish\_event/)

Django REST API приема multipart/form-data (image + metadata).

3. Django/DRF слоеве

Създаваш специален endpoint (напр. /api/hardware-events/):

POST с информация: време на финиширане, снимки от камерата, разпознати номера от OCR/AI

Свързва тези данни с конкретно състезание/heat, т.е. записва кой спортист (по read-код/номер) е пресекъл.

Моделите съдържат:

Event (timestamp, снимка, резултат OCR, статус – автоматичен/ръчен)

Връзка към Athlete/User, Race/Start/Heat

4. Frontend (Vue.js) и синхронация

Не се променя особено – UI периодично тегли последния статус от сървъра.

Когато нов event дойде от автоматика, всички клиенти моментално виждат обновения дневник, времена, снимки.

5. Може да добавиш...

WebSocket канал за live-feed (Django Channels или sse/starlette за съобщения в реално време).

Автокласифициращо UI за операторите (да посочват/коригират автоматични резултати при нужда).

Админ портал за история и анализи.

СХЕМА:

scss

[Инфра бариера] ------+

| (POST с event)

[Камери] -------------+----> [Малък Python сървис] ----(REST/HTTP)----> [DRF Backend /api/hardware-events/]

|

(DB)

|

<---------------------------

[Vue.js клиент] (Ajax polling за нови данни)

Следващи стъпки

1. Избери начин за свързване на бариерата към сървъра (IO/USB/COM/PIO/RPI? дадена ли е и къде ще бъде този сървър?)
2. Тествай в Python да „виждаш” събитие от бариерата.
3. Тествай в Python да вземеш кадър от камерата (ONVIF/RTSP/статичен файл).
4. Базов OCR на номера върху примерна снимка.
5. Django endpoint, който приема такива данни и запазва в бд/файлова система.
6. Коригирай модела на UI за да може да се визуализира снимката и OCR/данните в близо реално време.

Аз:

искам да уточня някои неща. първо - може ли сканирането (проверката през определен интервал от време на нещо) да се направи като част от Django сървъра или трябва да отделно приложени, комуникиращо с Django? Второ - ако е възможно, то кое от двете решения е по-добро

GPT-4.1

Страхотни въпроси – това са важни архитектурни решения!  
**Ще отговоря подробно и прагматично:**

## 1. ****Може ли сканирането (периодичните проверки) да се изпълнява като част от Django сървъра?****

### - ****Чисто технически – да,****

можеш да стартираш фонова задача (например с threading, multiprocessing, Celery worker, управляван scheduler или дори custom management command като процес, който върви успоредно със сървъра).

**Но:**

* Django е уеб framework по дизайн, не е направен за непрекъснато работещи нескончаеми цикли вътре в основния си процес.
* WSGI сървъри (Gunicorn, uWSGI), които стартират Django за production, се очаква да обработват уеб заявки, а не да държат дълги "while True" цикли.
* Подобно сканиране може да доведе до странни бъгове, неочаквано поведение (пример: прекарваш upgrade/reload на уеб сървъра, целият процес се прекъсва; работят няколко процеса и всеки си мисли, че трябва да сканира).

## 2. ****По-добре ли е да е външно приложение?****

### ****Да – почти винаги е по-добрият, сигурен и „разделен” вариант!****

* Правиш малък Python скрипт (или FastAPI, или дори Go/C++/Node – всякакъв език, който говори HTTP).
* „Сканирането” (напр. четене на GPIO, четене камера, периодично пращане към backend) се държи в отделен процес/сървис, може да върви като systemd service, в Docker container или просто като дълъг Python процес.
* Когато нещо се „улови” – скриптът се държи просто като “клиент” на DRF - прави REST POST/PUT/GET заявки.

#### ****Това дава:****

* Много по-ясна структура и надeждност.
* Безопасно рестартиране на уебсайта (или на скрипта), без да се нарушават други процеси.
* Лесна подмяна/отделяне/мащабиране: може да пуснеш няколко такива „хардуерни/сканиращи” процеса, ако хардуерът го изисква.

## 3. ****Има ли случаи, когато Django вътрешно сканиране е ОК?****

* **Кратки или рядко нужни периодични задачи** (няколко пъти в час/ден) – може да се използва Celery Beat или Django-Q.
* **Обработка на “софтуерни” задачи** без нужда от хардуерно взаимодействие.

Пример: разпращане на имейли, преизчисляване на класации, кеширане.

## 4. ****Реални аргументи за/против (TL;DR):****

Таблица

|  | **Django вътре** | **Външен скрипт** |
| --- | --- | --- |
| **Изпълнение** | Runnable, но рисковано | Най-чист и надежден |
| **Мащабируемост** | Лоша | Лесна |
| **Поддръжка** | Трудна | Лесна |
| **Изолираност** | Липсва | Отлична |
| **Production-Ready** | Не се препоръчва | Препоръчва се |

## ****Обобщено решение:****

* **Моят съвет**:  
  **Прави „сканирането” като отделен скрипт/service**, който комуникира със сървъра (DRF) по HTTP.  
  Django остава чист, отговаря само за уеб/API логика.

### ****Модел на реализация****:

* hardware\_listener.py (скрипт на Python), който си върши сканирането + връзка с GPIO или камера.
* Django DRF – приема външни POST заявки, обработва резултата, пази в БД.
* Frontend – просто получава всичко от сървъра.

АЗ:

разполагам с Arduino nano - дай ми предложение за него. искам да се регистрират две събития за бариерата: 1: преход от не задействана към задействана 2: преход от задействана към не задействана

GPT-4.1

Много добре – **Arduino Nano** е отличен избор за такъв интерфейс!  
Ще ти дам напълно работеща примерна схема и код за Arduino, както и как би приел събитията от PC страна (Python).

## ****1. Схема на свързване****

* **NO/NC бариера (релеен изход) към пин D2 на Arduino Nano**.
* Популярен вариант:
  + **GND на бариерата** към **GND на Arduino**.
  + **NC контакт на бариерата** към **D2 на Arduino**.

Таблица

| **Бариера (NC контакт)** | **Arduino D2** |
| --- | --- |
| Бариера GND | Arduino GND |

NC (Normally Closed) означава, че веригата е затворена когато бариерата НЕ е задействана.

## ****2. Примерен код за Arduino Nano****

Ще използваме **Digital Input с включен pullup** и ще изпращаме различни команди към PC през Serial при двата прехода (DOWN/UP):

cpp

const int BARRIER\_PIN = 2;

int lastState = HIGH; // При PULLUP - HIGH = незадействано

void setup() {

pinMode(BARRIER\_PIN, INPUT\_PULLUP);

Serial.begin(9600);

}

void loop() {

int currentState = digitalRead(BARRIER\_PIN);

// Преход от NE задействана (HIGH) към задействана (LOW)

if (lastState == HIGH && currentState == LOW) {

Serial.println("ACTIVATED"); // Бариера задействана

delay(20); // debounce

}

// Преход от задействана (LOW) към НЕ задействана (HIGH)

if (lastState == LOW && currentState == HIGH) {

Serial.println("DEACTIVATED"); // Бариера освободена

delay(20); // debounce

}

lastState = currentState;

delay(2); // Main loop debounce

}

### ****Ключови моменти:****

* INPUT\_PULLUP: Ардуино "дърпа" входа нагоре – без външен резистор.
* NC контакт: HIGH = незадействана (контактът е свързан към GND), LOW = задействана (веригата е прекъсната).
* Забавяне за debounce (може да се финишира според времето на твоята бариера).
* Ако логиката на бариерата е обратна (NO, а не NC), може да размениш HIGH и LOW.

## ****3. Python скрипт за слушане на събитията през Serial****

python

import serial

ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1) # провери точния порт!

while True:

line = ser.readline().decode().strip()

if line == "ACTIVATED":

print("БАРИЕРА ЗАДЕЙСТВАНА!")

# Тук – HTTP POST към Django за 'бариера задействана'

elif line == "DEACTIVATED":

print("БАРИЕРА ОСВОБОДЕНА!")

# Тук – HTTP POST към Django за 'бариера освободена'

## ****4. Към Django****

Всеки път когато получиш събитие, можеш да изпратиш нещо такова:

python

import requests

import time

def send\_barrier\_event(state):

data = {

"event\_type": state, # "barrier\_activated" или "barrier\_deactivated"

"timestamp": time.time()

}

# Например на http://localhost:8000/api/barrier\_event/

response = requests.post(

"http://localhost:8000/api/barrier\_event/",

json=data,

headers={"Authorization": "Token <you\_token\_if\_needed>"}

)

print(response.status\_code, response.text)

Използвай това във всяко събитие от скрипта по-горе.

## ****5. Ако работиш под Windows****

* Променяш /dev/ttyUSB0 с примерно COM4 или съответния COM порт на Arduino Nano.

## ****6. Как ще изглеждат събитията:****

В твоето Django приложение ще пристигат два типа събития:

* **barrier\_activated** (бариерата е ЗАДЕЙСТВАНА – т.е. някой е пресекъл)
* **barrier\_deactivated** (бариерата НЕ Е задействана – свободна е)

## ****Какво дава това?****

* Перфектно локализиране на точните моменти на преход.
* 100% надежден watchdog: винаги знаеш кога има промяна, без да ползваш delay цикли за постоянно четене на входа.
* Кратък дебоунс за електрически шум.
* Лесно разпознаваш „висящ” релей (ако по някаква причина бариерата остане задействана), можеш да алармираш оператора.

когато изпращам към Django POST заявка, трябва да има и CSRF token. в горния случай, скриптът, който чете серийния порт и изпраща заявки към сървъра как ще получи актуалния токен

GPT-4.1

Много добър въпрос – това e често срещано притеснение при автоматизирани заявки от външен процес към Django.

## ****CSRF и сценарият със скрипт от Arduino****

**CSRF защита** (Cross-Site Request Forgery) е механизъм, който предпазва браузър клиенти при изпращане на POST/PUT/PATCH заявки, когато те работят в браузър сесия.

**Но:** За автоматизирани външни скриптове (като твоя Python скрипт, който слуша Arduino), ползващи Django REST API, най-доброто решение е да активираш **автентикация с токен** и да изключиш CSRF за API endpoint-ите.

### ****С две думи:****

* Скриптове, които не са браузър клиенти, не могат да ползват лесно CSRF токени.
* **Django REST Framework** има вградена поддръжка за Token Authentication (и други като JWT).
* За Token Authentication не се изисква CSRF токен — само HTTP header с токена.

## ****Решение:****

### ****1. Активирай Token Authentication във Django REST****

**Инсталация:**

shell

pip install djangorestframework

pip install djangorestframework-authtoken

**В settings.py:**

python

INSTALLED\_APPS = [

# ...

'rest\_framework',

'rest\_framework.authtoken',

]

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.TokenAuthentication',

],

}

**Миграция:**

python manage.py migrate authtoken

**Генериране на токен за даден потребител:**

shell

python manage.py drf\_create\_token <username>

Ще получиш токен като:

ff7f6bee14320c13c6d40d13b3e38bce892d139b

### ****2. Изпращай токена с всеки POST (без CSRF)****

**Python скрипт:**

python

import requests

TOKEN = "ff7f6bee14320c13c6d40d13b3e38bce892d139b"

headers = {

"Authorization": f"Token {TOKEN}",

"Content-Type": "application/json",

}

data = {

"event\_type": "barrier\_activated",

"timestamp": 1234567890,

}

response = requests.post(

"http://localhost:8000/api/barrier\_event/",

json=data,

headers=headers

)

print(response.status\_code, response.text)

## ****Важно!****

* За твоята архитектура:  
  **API endpoint-ите, които приемат заявки от хардуер/скриптове, винаги да са върху TokenAuthentication (или друго подходящо non-browser auth), а не да изискват CSRF!**
* Ако маркетираш API само за скриптове - не е нужно CSRF, нужна е сигурна идентификация (т.е. токен).

## ****Ако (по изключение) искаш да ползваш CSRF...****

* Ще трябва да имитираш сесия на браузър (login, cookie, get csrf...), което е излишно сложно, тромаво и не е „RESTful“.
* Най-често не се препоръчва за машини и хардуер.

## ****ДОБРА ПРАКТИКА****

**→ За външни скриптове, използвай винаги токен или друг тип non-browser автентикация. Не CSRF!**

### ****Пояснение за прости случаи****

* **CSRF защита** е за защита на браузърни POST заявки от злоумишлени сайтове и не е приложима или нужна при машини/скриптове.
* Ако някой от външните ти API-та изисква да бъде защитен с CSRF (което по default не трябва да се налага!), можеш без риск да използваш декоратора

python

@csrf\_exempt

(но \_това е за необезпечени демо тестове, а не production!).

### ****Обобщено:****

* Скрипт, слушащ бариера/Arduino, изпраща заявки с токен в Authorization header.
* Не се занимаваш с CSRF токени – не са нужни извън браузърен свят.

АЗ:

правилно ли разбирам, че токена трябва да се генерира всеки път, когато сървъра се сартира

GPT-4.1

Не, **това не е необходимо и не е правилно**.  
**Token-а се създава еднократно за всеки потребител и може да се използва многократно, докато не бъде изтрит или сменен.**  
Той НЕ зависи от рестартиране на сървъра или от жизнения цикъл на Python/Django приложението.

## ****Как работи Token Authentication в DRF:****

1. **Генерираш токена веднъж** за даден потребител (суперпотребител, “hardware” потребител, оператор и т.н.) с команда като:

php

python manage.py drf\_create\_token <username>

– или ако използваш по-стария начин – през authtoken shell:

python

from rest\_framework.authtoken.models import Token

from django.contrib.auth.models import User

user = User.objects.get(username='hardware')

token = Token.objects.create(user=user)

print(token.key)

1. **Пазиш този токен** (ключ) на сигурно място (например като ENV променлива, конфигурационен файл или директно записан в Python скрипта).
2. **Всички скриптове** използват този токен при всяка заявка чрез HTTP header:

makefile

**Authorization: Token <полученият\_токен>**

1. **Токенът не се сменя автоматично при рестарт на Django сървъра или операционната система,** освен ако ти не го изтриеш, не създадеш нов нарочно или не смениш някакви ключови части от БД.

### ****Може да обобщим така:****

* **Token-ът е постоянно уникално "парче текст", което идентифицира твоя хардуерен скрипт/потребител към сървъра.**
* Генерираш го веднъж.
* Използваш го докато искаш, или докато решиш да го смениш (например, ако има компрометиране на сигурността).

#### ****Изключения:****

* Можеш нарочно да регенерираш токена (ако имаш причина ‒ компрометиране, ротация на ключове и т.н.), тогава ще трябва да го смениш и в скриптовете си.

**Най-често ще създадеш един “hardware” (или друг избран) потребител и ще генерираш един token за този потребител. Скриптовете на бариерата и камерите ще го ползват във всичките си заявки.**

**Ако имаш нужда — покажи примерната си команда за създаване и ще ти опиша целия цикъл.**

АЗ:

включването на Token Authentication в DRF ще промени ли използването на вече създаденото API, което се използва от клиентите и използва CSRF

GPT-4.1

Чудесен въпрос!  
**Краткият отговор е:**  
**Не, добавянето на Token Authentication само по себе си няма да промени начина, по който вече работят браузър-базираните Vue.js клиенти, които използват CSRF и session authentication – ако не направиш промяна в настройките на DEFAULT\_AUTHENTICATION\_CLASSES или логиката на API-тата.**

## ****Подробно обяснение:****

### 1. ****Какво означава добавянето на TokenAuthentication?****

* Ако в DEFAULT\_AUTHENTICATION\_CLASSES сложиш еднакво:

python

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.SessionAuthentication',

'rest\_framework.authentication.TokenAuthentication',

],

}

... **тогава и двата вида authentication работят едновременно** –

* + **Клиентите с браузър (Vue.js, работа с forms, CSRF)** — продължават да работят с cookie/session и CSRF токени
  + **Автоматичните скриптове/хардуер** — могат да ползват Token authentication през HTTP header

#### ****Обърни внимание:****

* **Редът няма значение** – първият успешен метод за автентикация ще бъде използван.
* Ако не подадеш токен, се гледа за сесия/cookie; ако няма сесия, се гледа токен.

### 2. ****Какво ще се промени за съществуващите ти клиенти?****

* **Нищо!**  
  Те пак ще си ползват стария начин (login, session cookie, CSRF).
* Новите (твоя Python скрипт/хардуер) — ще ползват токен.

### 3. ****Кога ще има промяна/проблеми?****

* **Само ако** премахнеш SessionAuthentication от списъка с DEFAULT\_AUTHENTICATION\_CLASSES **и оставиш само TokenAuthentication** — тогава вече няма да работят стандартните session/CSRF заявки.
* Ако сложиш само Token или само JWT и изключиш сесиите, тогава Vue клиентите трябва да се пренапишат да ползват токен, което не ти е нужно за този проект.

### 4. ****Примерна настройка (препоръчителна за комбинирана архитектура):****

python

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.SessionAuthentication',

'rest\_framework.authentication.TokenAuthentication',

],

}

* **Сега:**
  + Vue.js клиентите работят както досега (CSRF + сесии).
  + Външните Python/хардуер скриптове работят с токен.

## ****В обобщение:****

1. Просто добави TokenAuthentication без да махаш SessionAuthentication.
2. Всичко, написано за обикновените (браузърни) клиенти, ще си работи както досега!
3. Само ако имаш крайна причина за FULL API only достъп без сесии, тогава боли да се сменят клиентите.

АЗ:

ще използвам IP camera Dahua, модел DH-IPC-HFW1239TL1-A-IL. искам да се свържа към нея от python скрипт. искам да използвам офлайн библиотеки (т.е. скрипта да работи и без връзка към интернет)

GPT-4.1

Страхотен избор на камера! Dahua DH-IPC-HFW1239TL1-A-IL е ONVIF IP камера с възможности за RTSP стрийминг.  
**Можеш напълно офлайн (без интернет) да се свържеш към нея от Python скрипт**, за да гледаш видео потока, да правиш снимки (кадри) или да управляваш някои функции през ONVIF.

Ще ти дам примери как да:

* Достъпиш RTSP видеопотока и обработваш кадри (напр. за разпознаване/заснемане)
* Използваш ONVIF за някои настройки/достъп
* Използваш само офлайн библиотеки – нищо облачно, нито Dahua Cloud

## ****1. Видео стриймване/снимане с OpenCV (най-често)****

### ****Вземане на RTSP потока:****

1. **Провери RTSP адреса на камерата (най-често):**

php

rtsp://<user>:<password>@<ip\_address>:554/cam/realmonitor?channel=1&subtype=0

* + <user>: потребителското име (най-често admin)
  + <password>: паролата, която си задал (!!!)
  + <ip\_address>: адресът на камерата в твоята локална мрежа
  + channel=1 и subtype=0 – главният стрийм; subtype=1 – втори, по-ниско качество

1. **Примерен Python скрипт с OpenCV (работи офлайн!):**

python

import cv2

# RTSP линк на камерата

rtsp\_url = 'rtsp://admin:YOUR\_PASSWORD@192.168.1.20:554/cam/realmonitor?channel=1&subtype=0'

# Свържи се към потока

cap = cv2.VideoCapture(rtsp\_url)

if not cap.isOpened():

print("Не може да се свърже с камерата!")

exit(1)

while True:

ret, frame = cap.read()

if not ret:

print("Грешка при четене на видео!")

break

# Показване на видео, пауза с 'q'

cv2.imshow('Kamera', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

* + Можеш лесно да записваш снимки с:

python

cv2.imwrite('frame.png', frame)

## ****2. ONVIF – за администриране & функции (опционално)****

Dahua поддържа протокола ONVIF, така че може да ползваш и python-onvif за някои задачи (например да търсиш настройки, движения и т.н., но не всички Dahua камери дават пълен ONVIF control).

**Инсталирай библиотека:**

sh

pip install onvif-zeep

**Примерно търсене на профили:**

python

from onvif import ONVIFCamera

mycam = ONVIFCamera('192.168.1.20', 80, 'admin', 'YOUR\_PASSWORD') # Dahua често е на порт 80

media\_service = mycam.create\_media\_service()

profiles = media\_service.GetProfiles()

for profile in profiles:

print(profile)

## ****3. Защо не ти трябва интернет****

* **Всичко работи по LAN.** Нито RTSP, нито ONVIF биха искали връзка към външен сървър.
* Може да работиш със скрипта напълно офлайн, стига да имаш необходимите Python библиотеки и камерата да е достъпна в локалната мрежа.

## ****4. Какво ти трябва инсталирано?****

* **OpenCV:**

sh

pip install opencv-python

* **ONVIF (по желание):**

sh

pip install onvif-zeep

## ****5. Безопасност и добри практики****

* Пази потребителя и паролата на камерата само на сигурно място!
* Можеш да сложиш статично IP на камерата.
* Не включвай Dahua cloud в настройките на камерата, щом всичко ти трябва да е офлайн.

## ****6. Полезни RTSP примери за други потоци:****

* **Главен high-res:**

bash

rtsp://admin:password@ip:554/cam/realmonitor?channel=1&subtype=0

* **Втори low-res поток:**

bash

rtsp://admin:password@ip:554/cam/realmonitor?channel=1&subtype=1