Kamerový systém se záznamem a detekcí pohybu

Alexander Mateides

ČVUT–FIT

mateial1@fit.cvut.cz

29. prosince 2024

1 Úvod

Tento report se týká semestrální práce z předmětu BI-PYT v semestru B241. Cílem práce je vytvořit aplikaci, přes kterou se bude dát vzdáleně přistoupit k Wi-Fi kamerám a umožní sledování živého přenosu z kamery a manuální ovládání některých prvků (světlo, night-vision).

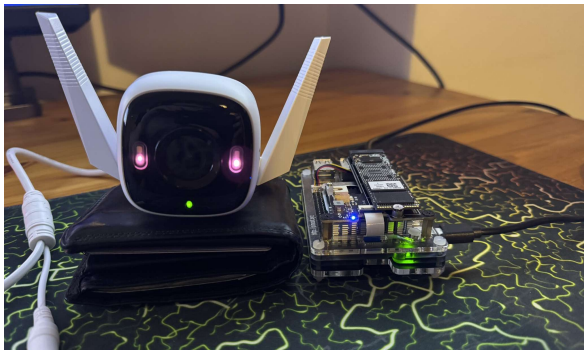
Dále také aplikace podporuje detekci pohybu, při které se automaticky zapne ukládání záznamu a zároveň pošle upozornění na e-mail (původně byla zamýšlena SMS, ale implementace je z programovací stránky prakticky stejná a za SMS se musí platit).

2 Síťové zapojení systému

Zde vysvětlím jak moje aplikace řeší zapojení kamer. Jak bývá zvykem, tak jsou Wi-Fi kamery připojeny k nějakému routeru. Dále je k tomuto routeru připojen ještě server, na kterém běží moje aplikace. Aplikaci jsem rozdělil na frontend a backend část. Komunikace potom probíhá následovně:

client ↔ frontend ↔ backend ↔ kamery

Kde frontend je napsán v React.js (není tedy předmětem práce). Backend potom využívá knihovnu FastAPI a jak s frontendem tak s kamerami komunikuje pomocí REST(ful) API. Aktuálně je v aplikaci naimplementován interface pro kameru TP-Link Tapo320WS (bude fungovat i pro kamery ze stejné rodiny).



Obrázek 1: Tapo320WS a RaspberryPi 5 server

3 Implementace backendu

Při implementaci jsem pro komunikaci s kamerami použil komunitou vytvořenou knihovnu **pytapo**. Tato knihovna potom zprostředkovává většinu komunikace mezi backendem a kamerami z TPLink Tapo rodiny. Vytvořit tedy API endpointy pro základní úkony jako zapínání světla nebyl velký problém.

3.1 Přenos streamu z kamery

Na první obtíže jsem narazil právě při získávání obrazu. Tapo kamery totiž streamují přes rtsp protokol, který je pro běžně nastavený prohlížeč nečitelný. Pokud si tedy frontend vyžádá přenos, je backendem tento rtsp stream konvertován pomocí opencv2 na stream jpeg obrázků, který už běžný prohlížeč dokáže zobrazit. Frontendu je potom otevřen websocket s tímto streamem.

Obrázek 2: Přenos obrazu z kamery

3.2 Detekce pohybu a alarm

Další netriviální záležitostí byla potom detekce pohybu. Naštěstí samotné Tapo kamery mají detekci pohybu již zabudovanou, tak jsem ji nemusel implementovat od nuly. Interface pro kameru má metodu getEvents, která vrátí momenty, kdy kamera zaznamenala pohyb za posledních n minut. Po celou dobu běhu serveru je tedy zapnutý listener loop, který v pravidelných časových intervalech provolává všechny kamery a když narazí na event, tak

spustí "poplach". To spočívá v odeslání e-mailu na specifikovanou adresu.

3.3 Stahování záznamů z kamery

Stahování záznamů byla poslední větší překážka, kterou jsem musel překonat. Problém je v tom, že záznamy se normálně nenachází na serveru, ale na kameře. Při vyžádání záznamu frontendem je tedy nejprve záznam stáhnut serverem. Na serveru je potom zkomprimován pomocí knihovny ffmpeg (zhruba 50Mb na 2Mb pro 1 min záznamu). Tento komprimovaný záznam je potom odeslán na frontend, kde je stažen clientem.

4 Závěr

Výsledkem je tedy poměrně dobře fungující backend, který umožní uživateli základní interakce s kamerami. Modulární implementace umožňuje případné rozšíření API pro jiné kamery. Dokonce by bylo možné mít zapojeno najednou více kamer ruzných druhů. V budoucnu by se určitě dalo vylepšit například zabezpečení API nebo vyladit dekódování rtsp streamu. Osobně jsem nejen s výsledky, ale hlavně procesem vývoje práce velmi spokojen. Vyzkoušel jsem si celý vývojový proces fullstack aplikace s přidanými prvky zpracování a přenosu video streamu a souborů. V projektu plánuji pokračovat i nad rámec semestrální práce a ideálně ho někdy v budoucnu nasadit například na monitoring chaty.

Reference

[1] ad1s0n. Tapo api reverse engineering. online, 2024. [cit. 2024–12–30] `https://dev.to/ad1s0n/reverse-engineering-tp-link-tapos-rest-api-part-1-4g6`.

[2] Juraj Nyiri. Pytapo. online, 2019-2024. [cit. 2024–12–30] `https://github.com/JurajNyiri/pytapo`.