**Vyšší odborná škola**

**a Střední průmyslová škola elektrotechnická**

**Plzeň, Koterovská 85**

**Dlouhodobá maturitní práce s obhajobou**

Téma: **Arduino Webcam**

**Autor práce: Ota VLNA**

**Třída: 4. P**

**Vedoucí práce: Jan Drexler**

**Dne: 31. 3. 2022**

**Hodnocení:**

**Vyšší odborná škola**

**a Střední průmyslová škola elektrotechnická**

**Plzeň, Koterovská 85**

**Zadání dlouhodobé maturitní práce**

**Žák: Ota VLNA**

**Třída: 4. P**

**Studijní obor:** **26-41-M/01 Elektrotechnika**

**Zaměření:****Internet věcí**

**Školní rok:** **2021 - 2022**

*Téma práce:*  **„Arduino Webcam*“***

***Pokyny k obsahu a rozsahu práce:***

1. ***Vytvoření programu pro kompilaci kódu a nahrávání výsledného programu do Arduina (12. listopadu 2021)***
2. ***Vytvoření webové aplikace pro psaní kódu a její propojení s programem (16. ledna 2022)***
3. ***Zapojení kamery a zprovoznění živého vysílání (20. února 2022)***
4. ***Výsledná realizace včetně doplnění příkladů kódu do webové aplikace (15. března 2022)***

***Požadavek na počet vyhotovení maturitní práce:*** *2 výtisky*

*Termín odevzdání:* ***31. března 2022***

*Čas obhajoby:* ***15 minut***

Vedoucí práce: **Ing. Radek MATOUŠEK**

Projednáno v **katedře ODP** a schváleno ředitelkou školy.

V Plzni dne: 30. září 2021 Ing. Naděžda Mauleová, MBA, v.r.

*ředitelka školy*

# Anotace

V dlouhodobé maturitní práci se věnuji tvorbě webové aplikace, jejíž účelem je usnadnit seznámení s platformou Arduino. Prostřednictvím webové aplikace je možné vzdáleně programovat desku Arduino Uno s několika připojenými periferiemi. V aplikaci je zabudováno živé vysílání umožňující sledovat dění na desce Arduino a periferiích v reálném čase.

# Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně a použil literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací. Souhlasím s využitím mé práce učiteli VOŠ a SPŠE Plzeň k výuce.

V Plzni dne: …..................... Podpis: …..........................

# Obsah

1. 1. Úvod
2. 2. Frontendová část webové aplikace
3. 2.1 JavaScriptový framework
4. 2.2 Souborová hierarchie
5. 2.3 CSS framework
6. 2.4 Vue.js komponenty webové aplikace
7. 2.5 Webserver Nginx
8. 3. Backendová část webové aplikace
9. 3.1 Node.js
10. 3.2 WebSocketová komunikace s frontendem, reverse proxy
11. 3.3 Sestavování výsledného programu ze zadaného kódu
12. 3.3.1 Jazyk Wiring
13. 3.3.2 Jazyk C
14. 4. Živé vysílání
15. 4.1 Kamera
16. 4.2 Twitch
17. 4.3 OBS
18. 5. Hardwarová část
19. 5.1 Raspberry PI
20. 5.2 Arduino UNO
21. 5.3 Zapojené periferie
22. 5.3.1 LED
23. 5.3.2 RGB LED
24. 5.3.3 Sedmisegmentový displej
25. 5.3.4 Fotorezistor
26. 5.3.5 Piezo bzučák
27. … ještě nevím co tam bude
28. 6. Příkladové programy
    1. 7. Použitý software a knihovny společné pro obě části aplikace

# 1 Úvod

Cílem mé dlouhodobé maturitní práce je vytvořit platformu, která umožní vyzkoušet si programování desky Arduino s několika vstupními a výstupními periferiemi i absolutním začátečníkům bez nutnosti počáteční investice.

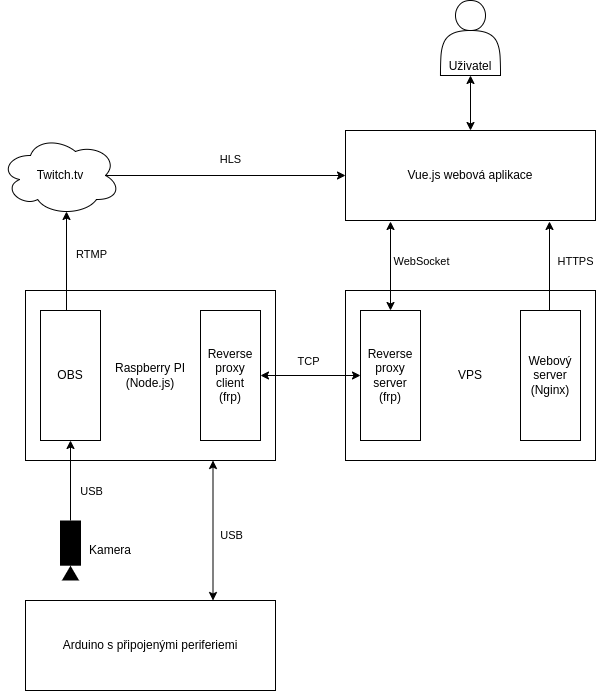
Projekt je složen z několika na sobě závislých částí.

Uživatel provádí interakci s webovou aplikací, která je psána v JavaScriptovém frameworku Vue.js. Webová aplikace je poskytována webovým serverem používajícím technologii Nginx, který je provozován na virtuálním privátním serveru (VPS) se statickou a veřejně přístupnou IP adresou.

Sestavování uživateli zadaného kódu, jeho nahrávání do Arduina a autoritativní řízení provozu webové aplikace má na starost vlastní server psaný v Node.js, který běží na Raspberry PI, které není veřejně přístupné z internetu. Pro umožnění obousměrné komunikace mezi webovou aplikací a programem na Raspberry PI pomocí WebSocketů jsem zvolil použití reverzní proxy frp mezi serverem na Raspberry PI a VPS.

Arduino je připojeno k Raspberry PI pomocí USB. Server na Raspberry PI sestaví program dle požadavku uživatele a potenciální fronty (v případě přístupu více uživatelů k aplikaci naráz) a nahraje ho do Arduina.

Kamera je připojena k Raspberry PI přes USB a snímá Arduino a připojené periferie. Program OBS Studio bežící na Raspberry PI streamuje video do služby Twitch. Ve službě Twitch je video zprocesováno a v co nejkratší možné době distribuováno uživatelům do přehrávače, který je vložen do webové aplikace.



# 2 Frontendová část webové aplikace

## 2.1 JavaScriptový framework

Framework (aplikační rámec) je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji.

https://cs.wikipedia.org/wiki/Framework

Pro vývoj webových aplikací je vhodné použít JavaScriptový framework. Měl jsem na výběr z několika možností. Mezi nejznámější a nejpoužívanější se řadí React a Vue.js. Tyto dva frameworky mají společnou vlastnost – virtuální DOM (Document Object Model).

Virtuální DOM je JavaScriptová reprezentace webového DOM umožňující rychlejší změny v reálném DOM. Framework provádí změny ve virtuálním DOM a poté porovná virtuální DOM s reálným DOM a v reálném změní pouze to, co je potřeba. Virtuální DOM musí běžet celou dobu paralelně se zbytkem webové aplikace.

Odlišný přístup volí nový framework Svelte, který kód psaný podobně jako v klasických frameworcích kompiluje do běžných JavaScriptových funkcí tak, aby žádný virtuální DOM nebyl potřeba. Výsledná sestavená webová aplikace je ve finále obvykle menší a rychlejší.

Pro svou webovou aplikaci jsem zvolil framework Vue.js 3, protože s ním mám největší zkušenosti a kód v něm psaný je obvykle kratší než v Reactu.

**Instalace nvm**

Vue.js vyžaduje prostředí Node.js, které jsem nainstaloval pomocí nvm (Node Version Manager), což je shell script umožňující jednoduchou instalaci prostředí Node.js v linuxovém prostředí. Nvm umožňuje správu různých verzí Node.js, podobně jako pyenv pro Python. Pro instalaci nvm jsem do terminálu zadal následující příkaz:

$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash

**Instalace Node.js**

Po instalaci nvm jsem nainstaloval Node.js. Pro instalaci nejnovější verze stačí zadat do terminálu následující příkaz:

$ nvm install node

Úspěšnou instalaci Node.js jsem ověřil pomocí příkazu:

$ node –-version

Pokud se vypíše verze nainstalovaného Node.js prostředí, instalace proběhla úspěšně. S Node.js se nainstaloval i správce balíčků npm (Node Package Manager), který bude využit pro instalaci knihoven a samotného Vue.js frameworku.

**Vytvoření projektu ve Vue.js**

Pro vytvoření projektu ve Vue.js frameworku jsem zadal následující příkaz:

$ npm init vue@latest

V pozadí se nainstaloval create-vue, oficiální Vue.js nástroj pro vytváření projektů. Vybral jsem nastavení projektu:

✔ Project name: … arduino-webcam-frontend

✔ Add TypeScript? … **No** / Yes

✔ Add JSX Support? … **No** / Yes

✔ Add Vue Router for Single Page Application development? … **No** / Yes

✔ Add Pinia for state management? … **No** / Yes

✔ Add Vitest for Unit Testing? … **No** / Yes

✔ Add Cypress for both Unit and End-to-End testing? … **No** / Yes

✔ Add ESLint for code quality? … No / **Yes**

✔ Add Prettier for code formatting? … **No** / Yes

Zde je možné nakonfigurovat projekt podle potřeb – přidat podporu TypeScriptu, JSX, Vue routeru a dalších nástrojů. To zjednodušuje základní konfiguraci projektu.

Pro pokročilejší aplikace je možné použít Nuxt.js, což je framework, který staví na frameworku Vue.js, a umožňuje velmi jednoduše nastavit pokročilé funkce jako je server-side renderování. Také stanovuje souborovou hierarchii projektu, a např. dynamicky nastavuje Vue router podle hierarchie souborů v podsložce „pages“. Ve větších projektech Nuxt.js umožňuje úsporu kódu a ještě více zjednoduší základní konfiguraci. Známější sourozenec frameworku Nuxt.js je framework Next.js, který dělá to samé, akorát staví na frameworku React. V mé aplikaci jsem se rozhodl Nuxt.js nevyužít, jelikož neplánuji využívat Vue router, a proto by Nuxt.js byl spíše přítěží (další část na které by byl projekt závislý).

## 2.2 Souborová hierarchie

Po dokončení vytváření Vue.js projektu se vytvoří adresář s názvem, který jsme specifikovali v nastavení „Project name“. V mém případě arduino-webcam-frontend. Pro vstup do adresáře a výpis souborů jsem použil následující příkazy:

$ cd arduino-webcam-frontend

$ ls

Příkaz ls vypsal soubory a adresáře, následující seznam obsahuje některé z nich a také některé, které se objeví teprve později:

* Adresář src – Zde se budou nacházet všechny zdrojové kódy aplikace.
* Adresář public – Zde se nachází statické soubory, které se v procesu sestavování aplikace přidávají k sestaveným kódům aplikace, např. index.html (základní html soubor, do kterého se vkládá sestavená Vue.js aplikace), favicon.ico (ikona webové stránky).
* Adresář node\_modules – Do této složky instaluje npm všechny knihovny (dependencies) projektu (objeví se až po prvním provedení příkazu npm install.
* Adresář dist – Do této složky se umístí výsledná sestavená webová aplikace po provedení příkazu npm run build.
* package.json – V tomto souboru se nachází hlavní informace o aplikaci, jako je název a verze. Obsahuje také seznam všech dependencies (závislostí) projektu a definici scriptů, které se spouštějí pomocí npm run <název\_scriptu>. Field „private“ nastavený na „true“ specifikuje, že tento npm balíček nemá být publikován do npm Registry (nebude k dispozici ostatním vývojářům k instalaci přes npm install).

{  
 "name": "arduino-webcam-frontend",

"version": "0.1.0",

"private": true,

"scripts": {

"serve": "vue-cli-service serve",

"build": "vue-cli-service build --mode development

"lint": "vue-cli-service lint",

"dev": "npm run serve"

},

"dependencies": {

"@fortawesome/fontawesome-svg-core": "^1.2.36",

"@fortawesome/free-solid-svg-icons": "^5.15.4",

"@fortawesome/vue-fontawesome": "^3.0.0-5",

"autoprefixer": "^9.8.8",

"core-js": "^3.6.5",

"postcss": "^7.0.39",

"tailwindcss": "npm:@tailwindcss/postcss7-compat@^2.2.17",

"vue": "^3.0.0",

"vue-plugin-load-script": "^2.1.0",

"vue3-ace-editor": "^2.2.1"

},

"devDependencies": {

"@vue/cli-plugin-babel": "~4.5.0",

"@vue/cli-service": "~4.5.0",

"@vue/compiler-sfc": "^3.0.0",

"@vue/eslint-config-standard": "^5.1.2",

"babel-eslint": "^10.1.0",

"eslint": "^6.7.2",

"eslint-plugin-import": "^2.20.2",

"eslint-plugin-node": "^11.1.0",

"eslint-plugin-promise": "^4.2.1",

"eslint-plugin-standard": "^4.0.0",

"eslint-plugin-vue": "^7.0.0"

}

}

* package-lock.json – Soubor, který se vygeneruje po provedení příkazu npm install, obsahuje seznam všech knihoven, na kterých závisí projekt i každá jeho knihovna a ukládá si její instalovanou verzi.
* README.md – Obsahuje informace určené pro uživatele usilující o zprovoznění projektu (ve formátu Markdown).
* .eslintrc.js – Nastavení linteru ESLint, který pomáhá hledat chyby v kódu
* .browserslistrc – Obsahuje seznam prohlížečů, u kterých chceme, aby byly podporovány webovou aplikací. Toto nastavení si přečte Babel a na základě toho zkompiluje JavaScript tak, aby ho přečetly i starší prohlížeče.
* .gitignore – Soubor obsahující seznam souborů a adresářů, které nechci ukládat do gitu (jedná se hlavně o adresář node\_modules, který může nabývat velmi velkých rozměrů a adresář dist).
* .editorconfig – Soubor nastavení editoru kódu vygenerovaný editorem Visual Studio Code.
* tailwind.config.js – Obsahuje nastavení CSS frameworku Tailwind CSS.

## 2.3 CSS framework

Tailwind CSS zjednodušuje psaní CSS pomocí předvytvořených CSS tříd (classes). Také vkládá vlastní výchozí konfiguraci CSS vlastností, což pokládá základ moderněji vypadající webové aplikace než s výchozím nastavením CSS.

Při sestavování aplikace vezme všechny CSS soubory a optimalizuje je tak, aby výsledná sestavená aplikace měla co nejmenší velikost.

**Instalace Tailwind CSS**

Pomocí následujících příkazů jsem nainstaloval Tailwind CSS.

$ npm install -D tailwindcss

$ npx tailwindcss init

V souboru tailwind.config.js jsem provedl konfiguraci. Ve fieldu „purge“ jsem specifikoval, které typy souborů mají být zpracovávány Tailwind CSS frameworkem.

module.exports = {

purge: ['./index.html', './src/\*\*/\*.{vue,js,ts,jsx,tsx}'],

darkMode: false,

theme: {

extend: {

flex: {

left: '1 0 60%',

right: '1 0 40%'

},

colors: {

background: '#232323',

buttonHover: '#2d2d2d',

primary: '#E2E2E2'

},

fontFamily: {

monospace: 'monospace'

}

}

},

variants: {

extend: {}

},

plugins: []

}

Pro ikony jsem se rozhodl využít knihovnu Font Awesome, kterou jsem nainstaloval pomocí příkazu:

$ npm i --save @fortawesome/fontawesome-svg-core

$ npm i --save @fortawesome/free-solid-svg-icons

$ npm i --save @fortawesome/vue-fontawesome@latest

První příkaz nainstaloval jádro knihovny, druhý příkaz samotné ikony a třetí příkaz Vue.js komponent, pomocí kterého budu jednoduše vkládat ikony do aplikace.

## 2.4 Vue.js komponenty webové aplikace

Hlavním JavaScriptovým souborem je **main.js**, který importuje komponent App, což je hlavní Vue.js komponent webové aplikace. Taktéž importuje některé použité knihovny, jako je font awesome (ikony), vue-plugin-load-script (pro načtení Twitch.tv přehrávače) a hlavní CSS soubor main.css.

import { createApp } from 'vue'

import App from './App.vue'

import './main.css'

import LoadScript from 'vue-plugin-load-script'

import { library } from '@fortawesome/fontawesome-svg-core'

import { faUpload, faCamera, faFileUpload, faFileDownload, faWifi, faExclamationCircle } from '@fortawesome/free-solid-svg-icons'

import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'

library.add([faUpload, faCamera, faFileUpload, faFileDownload, faWifi, faExclamationCircle])

const app = createApp(App).component('font-awesome-icon', FontAwesomeIcon)

app.use(LoadScript)

app.mount('#app')

Po importování těchto závislostí se Vue.js aplikace vloží do divu s id „app“ v index.html.

<!DOCTYPE html>

<html lang="">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width,initial-scale=1.0">

<link rel="icon" href="<%= BASE\_URL %>favicon.ico">

<title><%= htmlWebpackPlugin.options.title %></title>

</head>

<body class="bg-background h-screen">

<noscript>

<strong>Je nutno používat prohlížeč s podporou JavaScriptu.</strong>

</noscript>

<div id="app" class="h-full"></div>

<!-- aplikace se vloží zde -->

</body>

</html>

# 3 Backendová část webové aplikace

npm init @eslint/config

Need to install the following packages:

@eslint/create-config

Ok to proceed? (y)

✔ How would you like to use ESLint? · style

✔ What type of modules does your project use? · esm

✔ Which framework does your project use? · none

✔ Does your project use TypeScript? · No / Yes

✔ Where does your code run? · node

✔ How would you like to define a style for your project? · guide

✔ Which style guide do you want to follow? · standard

✔ What format do you want your config file to be in? · JavaScript

# 4 Živé vysílání

# 5 Hardwarová část

## 5.1 Raspberry PI

Na micro SD kartu jsem použitím programu Rapberry PI Imager nainstaloval operační systém Raspberry PI OS, kartu jsem vložil do mikropočítače Rapsberry PI Model B o velikosti operační paměti 2 GB. Po zapojení napájení systém naběhl, což jsem pozoroval na připojeném monitoru přes micro HDMI redukci do HDMI kabelu vedoucího do monitoru. Po chvilce běhu jsem zjistil, že systém nepracuje tak rychle, jak by měl, což bylo způsobeno nedostatečným výkonem napájecího zdroje. Vyměnil jsem tedy 5 V 2 A DC zdroj za 5 V 3 A DC zdroj, což vyřešilo problém.

Mikropočítač jsem připojil k internetu a nainstaloval potřebný software, jehož instalace je popsána v předchozích kapitolách.

Stáhnul jsem repozitář svého projektu a provedl instalaci jeho backendové části a spustil jsem ji. Následně jsem spustil živé vysílání.

# 7 Použitý software a knihovny společné pro obě části aplikace

**Node.js** je JavaScriptový runtime umožňující používat JavaScript mimo webový prohlížeč (např. na webovém backendu, v příkazové řádce, či v mobilních aplikacích). V mé aplikaci je Node.js použit ve vývojovém prostředí a na backendu.

**Git** je software sloužící ke správě verzí projektu. Prostřednictvím příkazů v příkazové řádce umožňuje jednoduše přidávat nové změny (commity), případně synchronizovat změny se vzdáleným serverem pro umožnění spolupráce více lidí, či pro zálohování projektu. V mém případě jsem jako git server využil školní GitLab server.

**Babel** je JavaScriptový compiler, který se stará o sestavování JavaScriptu do produkčního prostředí. Umožňuje jednoduše nastavit prohlížeče, které je potřeba podporovat. Babel se postará o to, aby moderní kód, který napíšeme a který nemusí být podporovaný staršími prohlížeči, byl zkompilován do JavaScriptu čitelného těmito prohlížeči.