**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: **Jan DREXLER**

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

[Anotace 3](#_Toc101035437)

[Prohlášení 3](#_Toc101035438)

[Úvod 6](#_Toc101035439)

[1 Frontendová část webové aplikace 8](#_Toc101035440)

[1.1 JavaScriptový framework 8](#_Toc101035441)

[1.2 Souborová hierarchie 9](#_Toc101035442)

[1.3 CSS framework 12](#_Toc101035443)

[1.4 Vue.js komponenty webové aplikace 13](#_Toc101035444)

[2.5 Webserver Nginx 24](#_Toc101035445)

[2 Backendová část webové aplikace 26](#_Toc101035446)

[2.1 Node.js 26](#_Toc101035447)

[2.2 WebSocketová komunikace s frontendem, služba localhost.run 26](#_Toc101035448)

[2.3 Sestavování výsledného programu ze zadaného kódu 30](#_Toc101035449)

[2.3.1 Jazyk Wiring 30](#_Toc101035450)

[2.3.2 Jazyk C 30](#_Toc101035451)

[3 Živé vysílání 32](#_Toc101035452)

[4 Hardwarová část 33](#_Toc101035453)

[4.1 Raspberry PI 33](#_Toc101035454)

[4.2 Arduino UNO 33](#_Toc101035455)

[5 Příkladové kódy 34](#_Toc101035456)

[5.1 V jazyce Wiring 34](#_Toc101035457)

[5.2 V jazyce C 37](#_Toc101035458)

[6 Použitý software a knihovny společné pro obě části aplikace 39](#_Toc101035459)

# 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.

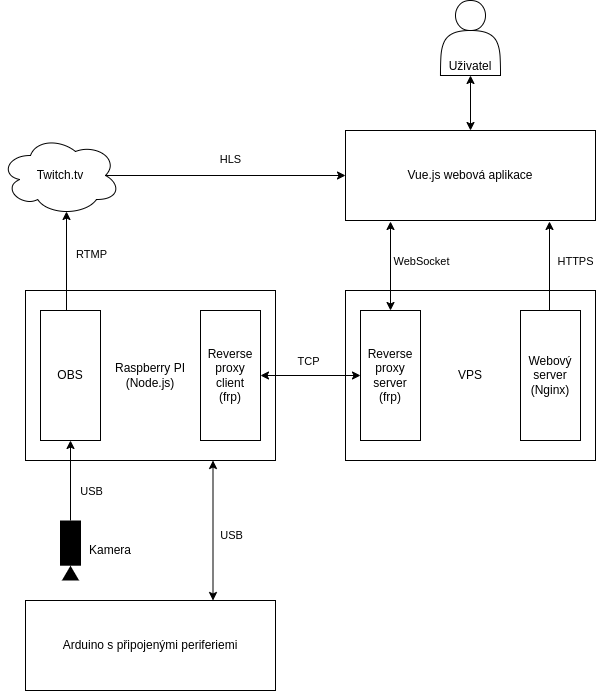
Projekt je složen z několika na sobě závislých částí.

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 služby localhost.run.

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 počítači přes USB a snímá Arduino a připojené periferie. Program OBS Studio běžící na počítači 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.



# 1 Frontendová část webové aplikace

## 1.1 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. 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).

## 1.2 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": "1.0.0",

    "private": true,

    "scripts": {

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

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

      "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",

      "file-saver": "^2.0.5",

      "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

module.exports = {

    root: true,

    env: {

      node: true

    },

    extends: [

      'plugin:vue/vue3-essential',

      '@vue/standard'

    ],

    parserOptions: {

      parser: 'babel-eslint'

    },

    rules: {

      'no-console': process.env.NODE\_ENV === 'production' ? 'warn' : 'off',

      'no-debugger': process.env.NODE\_ENV === 'production' ? 'warn' : 'off'

    }

  }

* .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.

## 1.3 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',

          btn: '#303030',

          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.

## 1.4 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>

Hlavním komponentem je App.vue, který definuje základní strukturu HTML stránky, obstarává komunikaci přes WebSocket a obsahuje další 3 komponenty - Editor.vue, Log.vue a Player.vue.

<template>

  <header class="py-4">

    <div class="lg:mx-20 mx-5 flex">

      <h1 class="text-primary">

        <font-awesome-icon icon="camera" />

        Arduino Webcam

      </h1>

      <div class="ml-5">

        <span v-if="websocketReady" class="text-green-600">

          <font-awesome-icon icon="wifi" />

          Connected

        </span>

        <span v-else class="text-red-600">

          <font-awesome-icon icon="exclamation-circle" />

          Trying to reconnect...

        </span>

      </div>

      <div class="ml-auto">

        <span class="text-primary mr-5" v-if="isRunning">

            Current user time left: {{ queueTimeLeft }} s

        </span>

        <span class="text-green-600" v-if="myTurn">Your turn</span>

        <span class="text-yellow-600" v-else-if="queueLength > 0 && !waitingForTurn">Queue length: {{ queueLength }}</span>

        <span class="text-yellow-600" v-else-if="queueLength > 0 && waitingForTurn">Queue position: {{queuePosition}}/{{ queueLength }}</span>

        <span class="text-primary" v-else>Queue empty</span>

      </div>

    </div>

  </header>

  <main class="main flex flex-row">

    <Editor

      class="flex-grow"

      v-model:code="code"

      v-model:lang="lang"

      :canUpload="websocketReady && !waitingForTurn"

      :waitingInQueue="queuePosition > 0"

      :currentCode="currentCode"

      :currentLang="currentLang"

      @upload="upload"

    />

    <div class="log-player-container flex flex-col">

      <Log :messages="messages" class="bg-white overflow-y-auto flex-grow" />

      <Player />

    </div>

  </main>

</template>

<script>

import Editor from './components/Editor.vue'

import Log from './components/Log.vue'

import Player from './components/Player.vue'

export default {

  name: 'App',

  components: {

    Editor,

    Log,

    Player

  },

  data () {

    return {

      code: '',

      lang: 'wiring',

      messages: [{ stdout: 'Waiting for input... <br />' }],

      ws: null,

      websocketReady: false,

      queueLength: null,

      timeoutStartTime: null,

      timeoutLength: null,

      isRunning: false,

      dateNow: 0,

      myTurn: false,

      queuePosition: null,

      currentCode: ''

    }

  },

  computed: {

    queueTimeLeft () {

      return (((this.timeoutStartTime + this.timeoutLength) - this.dateNow) \* 0.001).toFixed(0)

    },

    waitingForTurn () {

      return this.queuePosition !== null

    }

  },

  methods: {

    establishConnection () {

      this.ws = new WebSocket('wss://arduino-webcam.lhr.rocks/')

      this.ws.addEventListener('open', (event) => {

        console.log(this.ws)

        console.log('socket connected')

        this.websocketReady = true

      })

      this.ws.addEventListener('message', (event) => {

        const message = JSON.parse(event.data)

        console.log(message)

        switch (message.type) {

          case 'log':

            this.messages.push(this.formatMessage(message))

            break

          case 'queue':

            this.timeoutStartTime = message.timeoutStartTime

            this.timeoutLength = message.timeoutLength

            this.queueLength = message.queueLength

            this.isRunning = message.isRunning

            break

          case 'requestComplete':

            this.myTurn = false

            this.queuePosition = null

            break

          case 'yourTurn':

            this.myTurn = true

            break

          case 'queuePosition':

            this.queuePosition = message.position

            break

          case 'currentCode':

            this.currentCode = message.code

            this.currentLang = message.lang

            break

        }

      })

      this.ws.addEventListener('error', (event) => {

        console.log('socket encountered error')

        this.ws.close()

      })

      this.ws.addEventListener('close', (event) => {

        console.log('socket closed, trying to reconnect')

        this.websocketReady = false

        setTimeout(() => {

          this.establishConnection()

        }, 1000)

      })

    },

    upload () {

      this.ws.send(JSON.stringify({ code: this.code, lang: this.lang }))

    },

    formatMessage (message) {

      if (message.stdout) {

        // eslint-disable-next-line no-control-regex

        message.stdout = message.stdout.replace(/\n/g, '<br />').replace(/[\u001b\u009b][[()#;?]\*(?:[0-9]{1,4}(?:;[0-9]{0,4})\*)?[0-9A-ORZcf-nqry=><]/g, '')

      }

      if (message.stderr) {

        // eslint-disable-next-line no-control-regex

        message.stderr = message.stderr.replace(/\n/g, '<br />').replace(/[\u001b\u009b][[()#;?]\*(?:[0-9]{1,4}(?:;[0-9]{0,4})\*)?[0-9A-ORZcf-nqry=><]/g, '')

      }

      return message

    },

    dateNowUpdate () {

      this.dateNow = Date.now()

    }

  },

  created () {

    setInterval(this.dateNowUpdate.bind(this), 100)

    this.establishConnection()

  }

}

</script>

<style scoped>

.main {

  height: calc(100% - 56px);

}

.log-player-container {

  width: 832px;

}

</style>

Editor.vue se stará o vstup uživatelem psaného kódu a výběr úryvků kódu (snippetů). Snippety se načítají ze souboru snippets.js. Využívá se editoru Ace, který je open-source.

<template>

  <div class="flex flex-col">

    <v-ace-editor

      :value="shownCode"

      :options="{ printMargin: false, fontSize: '17px', useWorker: true }"

      :readonly="editDisabled"

      @update:value="$emit('update:code', $event)"

      @init="editorInit"

      lang="c\_cpp"

      theme="twilight"

      ref="ace"

      class="flex-1"

    />

    <div class="mt-1 mx-3 flex">

      <div>

        <div class="inline" v-if="!editDisabled">

          <label class="text-primary whitespace-nowrap" for="language">Language: </label>

          <select

            class="

              text-primary

              bg-background

              mr-3

              border-solid border-2 border-primary

            "

            name="language"

            id="language"

            @input="selectLang($event.target.value)"

            :value="lang"

          >

            <option value="wiring" selected>Wiring</option>

            <option value="c">C</option>

          </select>

        </div>

        <div class="inline" v-if="!editDisabled">

          <label class="text-primary whitespace-nowrap" for="snippet">Code snippet: </label>

          <select

            class="

              text-primary

              bg-background

              border-solid border-2 border-primary

              mr-3

            "

            name="snippet"

            id="snippet"

            :value="selectedSnippet.name"

            @input="selectSnippet($event.target.value)"

          >

            <option

              v-for="(snippet) in currentSnippets"

              :key="snippet.name"

            >

              {{ snippet.name }}

            </option>

          </select>

        </div>

      </div>

       <div class="inline">

          <label class="text-primary whitespace-nowrap" for="selectedCode">Show: </label>

          <select

            class="

              text-primary

              bg-background

              mr-3

              border-solid border-2 border-primary

            "

            name="selectedCode"

            id="selectedCode"

            @input="selectedCode = $event.target.value"

            :value="selectedCode"

          >

            <option value="my" selected>My code</option>

            <option value="current">Current code</option>

          </select>

        </div>

        <p v-if="editDisabled && currentLang" class="text-primary mr-2">Language: {{ currentLang }}</p>

        <p v-if="editDisabled" class="font-bold text-red-600">Showing currently running code. Editing disabled!</p>

      <div class="ml-auto">

        <button

          class="

            mr-2

            bg-btn

            hover:bg-buttonHover

            text-white

            py-2

            px-4

            rounded

          "

          @click="saveFile()"

        >

          <font-awesome-icon icon="file-download" /> Download

        </button>

        <button

          v-if="!editDisabled"

          class="

            border-solid border-2 border-blue-500

            mr-2

            bg-btn

            hover:bg-buttonHover

            text-blue-500

            py-2

            px-4

            rounded

          "

          :class="!canUpload ? 'opacity-30 cursor-not-allowed' : ''"

          @click="handleUpload"

        >

          <font-awesome-icon icon="upload" :disabled="!canUpload || waitingInQueue" />

          Upload to Arduino

        </button>

      </div>

    </div>

  </div>

</template>

<script>

import { VAceEditor } from 'vue3-ace-editor'

import 'ace-builds/src-noconflict/mode-c\_cpp'

import 'ace-builds/src-noconflict/theme-twilight'

import snippets from '../snippets'

import { saveAs } from 'file-saver'

export default {

  name: 'Editor',

  components: {

    VAceEditor

  },

  data () {

    return {

      selectedCode: 'my',

      selectedSnippet: null

    }

  },

  computed: {

    snippets () {

      return snippets

    },

    currentSnippets () {

      if (this.lang === 'c') return this.snippets.c

      return this.snippets.wiring

    },

    shownCode () {

      if (this.selectedCode === 'my') {

        return this.code

      } else if (this.selectedCode === 'current') {

        return this.currentCode

      }

      return ''

    },

    editDisabled () {

      return this.selectedCode !== 'my'

    }

  },

  created () {

    this.selectedSnippet = this.currentSnippets[0]

    this.$emit('update:code', this.selectedSnippet.code)

  },

  props: ['code', 'lang', 'canUpload', 'waitingInQueue', 'currentCode', 'currentLang'],

  emits: ['update:code', 'update:lang', 'upload'],

  methods: {

    editorInit () {

      console.log('editor init')

    },

    handleUpload () {

      if (this.canUpload) {

        this.$emit('upload')

      }

    },

    saveFile () {

      const file = new Blob([this.shownCode])

      let fileExtension

      if (this.editDisabled) {

        fileExtension = this.currentLang === 'wiring' ? 'ino' : 'c'

      } else {

        fileExtension = this.lang === 'wiring' ? 'ino' : 'c'

      }

      saveAs(file, `code.${fileExtension}`)

    },

    selectSnippet (snippet, makePrompt = true) {

      this.selectedSnippet = this.currentSnippets.find((s) => s.name === snippet)

      if (makePrompt) {

        if (confirm('This action will overwrite your current code. Continue?')) {

          this.$emit('update:code', this.selectedSnippet.code)

        }

      } else {

        this.$emit('update:code', this.selectedSnippet.code)

      }

    },

    selectLang (lang) {

      if (confirm('This action will overwrite your current code. Continue?')) {

        this.$emit('update:lang', lang)

        this.$nextTick(() => {

          this.selectSnippet('Default (empty)', false)

        })

      }

    }

  }

}

</script>

Log.vue se stará o výpis zpráv přicházejících z backendu.

<template>

<div>

  <div class="bg-black min-h-full px-3 py-2">

    <ul class="text-white font-monospace" ref="messages">

      <li v-for="(m, i) in messages" :key="i" class="inline">

        <span v-if="m.stdout" v-html="m.stdout"></span>

        <span v-if="m.stderr" v-html="m.stderr" class="text-red-600"></span>

      </li>

    </ul>

  </div>

</div>

</template>

<script>

export default {

  name: 'Log',

  props: {

    messages: Array

  },

  watch: {

    messages: {

      deep: true,

      handler () {

        this.$nextTick(() => {

          this.$refs.messages.children[this.$refs.messages.children.length - 1].scrollIntoView({ behavior: 'auto', block: 'end', inline: 'nearest' })

        })

      }

    }

  }

}

</script>

Player.vue se stará o přehrávání živého vysílání.

<template>

  <div class="flex flex-col">

    <div ref="twitchVideo" />

  </div>

</template>

<script>

export default {

  name: 'Player',

  data () {

    return {

      player: null,

      streamWidth: 1280,

      streamHeight: 720,

      windowWidth: 0,

      windowHeight: 0

    }

  },

  computed: {

    sizeCoef () {

      if (this.windowWidth < 600) {

        return 0.25

      } else if (this.windowWidth < 900) {

        return 0.3

      } else if (this.windowWidth < 1200) {

        return 0.4

      } else if (this.windowWidth < 1500) {

        return 0.5

      } else return 0.65

    },

    options () {

      return {

        width: this.streamWidth \* this.sizeCoef,

        height: this.streamHeight \* this.sizeCoef,

        channel: 'arduinowebcam',

        parent: ['localhost', 'sudety.ch', 'www.sudety.ch']

      }

    }

  },

  methods: {

    startStream () {

      this.$loadScript('twitch.js')

        .then(() => {

          this.player = new window.Twitch.Player(

            this.$refs.twitchVideo,

            this.options

          )

          this.player.setVolume(1)

          console.log(this.player)

        })

        .catch(() => {

          console.log('failed to load twitch script')

        })

    }

  },

  mounted () {

    this.windowHeight = window.innerHeight

    this.windowWidth = window.innerWidth

    this.startStream()

  }

}

</script>

## 2.5 Webserver Nginx

Sestavený frontend je provozován na virtuálním privátním serveru s operačním systémem Debian pomocí programu Nginx. Tento server má IP 162.19.137.183 a přes DNS je k němu přiřazená doména sudety.ch, kterou jsem použil z důvodu redukce výdajů spojených s prací.

Nginx je společně s Apache nejrozšířenější webserver na světě. S výchozím nastavením jsou soubory produkčně sestavené webové aplikace vloženy do adresáře /var/www/html.

# 2 Backendová část webové aplikace

## 2.1 Node.js

Backendovou část webové aplikace jsem napsal v JavaScriptu běžícím v prostředí Node.js. Projekt jsem zahájil pomocí příkazu:

$ npm init

Následně jsem pomocí npm nainstaloval WebSocketovou knihovnu pro komunikaci s frontendem:

$ npm install ws

Do projektu jsem přidal eslint, což je nástroj pro analýzu kódu, pomocí kterého můžu jednoduše udržovat jednotný formát kódu.

$ npm init @eslint/config

## 2.2 WebSocketová komunikace s frontendem, služba localhost.run

Celý kód backendu se nachází v souboru index.js. Tento kód spustí WebSocketový server na portu 1337, který následně reaguje na přicházející požadavky. Následně se pomocí příkazu, který je spouštěn pomocí funkce exec, nalezne připojené Arduino:

$ ls /dev | grep ttyUSB

Implementován je systém fronty, který řeší stav, kdy se k webové aplikaci připojí více uživatelů najednou. Kdo zadá svůj příkaz první, půjde první na řadu. V případě úspěšného nahrání kódu do Arduina nemůže být po dobu 30 sekund nahráván další kód. Po uplynutí tohoto času je na řadě další uživatel ve frontě. V případě neúspěšného nahrání (např. pokud se kód nepodaří zkompilovat) je tento čas pouze 10 sekund.

Protože backend běží na Raspberry PI, které nemá veřejnou IP adresu, rozhodl jsem se použít službu localhost.run, která po spuštění příkazu vytvoří SSH tunel mezi localhostem a vybranou doménou (v mém případě arduino-webcam.lhr.rocks):

$ ssh -R arduino-webcam.lhr.rocks:80:localhost:1337 plan@localhost.run

Služba localhost.run umožňuje použít buď svou vlastní doménu, nebo doménu třetího řádu na lhr.rocks.

**index.js:**

import WebSocket, { WebSocketServer } from 'ws'

import { exec, spawn } from 'child\_process'

import fs from 'fs'

import { promisify, inspect } from 'util'

import { randomUUID } from 'crypto'

const execPromise = promisify(exec)

const TIMEOUT\_NORMAL = 30000

const TIMEOUT\_ERROR = 10000

const wss = new WebSocketServer({ port: 1337 })

const queue = []

let currentRequest = null

let timeout = null

let timeoutStartTime = 0

let timeoutLength = 0

let isRunning = false

let device = ''

execPromise('ls /dev | grep ttyUSB').then(({ error, stdout, stderr }) => {

  if (stderr) {

    console.error(stderr)

  } else if (error) {

    console.error(error)

  }

  device = stdout.trim()

})

wss.on('connection', function connection (ws) {

  ws.id = randomUUID()

  sendQueueInfo()

  ws.on('message', async function message (data) {

    const request = JSON.parse(data)

    if (queue.find((r) => r.ws.id === ws.id) || (currentRequest && currentRequest.ws.id === ws.id)) {

      return

    }

    queue.push({ ...request, ws })

    sendQueueInfo()

    ws.send(JSON.stringify({ type: 'uploadSuccess' }))

    if (!timeout) {

      queueNext()

    }

  })

})

async function processC (request) {

  fs.writeFileSync('code-c/code.c', request.code)

  const command = `avr-gcc -Os -DF\_CPU=16000000UL -mmcu=atmega328p -c -o code.o code.c && avr-gcc -mmcu=atmega328p code.o -o code && avr-objcopy -O ihex -R .eeprom code code.hex && avrdude -F -V -c arduino -p ATMEGA328P -P /dev/${device} -b 115200 -U flash:w:code.hex 2>&1`

  return new Promise((resolve) => {

    const process = spawn(command, { cwd: 'code-c', shell: true })

    process.stdout.on('data', (data) => {

      broadcast(wss, { type: 'log', stdout: data.toString('utf8') })

    })

    process.stderr.on('data', (data) => {

      broadcast(wss, { type: 'log', stderr: data.toString('utf8') })

    })

    process.on('close', (code) => {

      resolve(code)

    })

    process.on('error', (err) => {

      console.error(inspect(err))

    })

  })

}

async function processWiring (request) {

  fs.writeFileSync('code-wiring/code.ino', request.code)

  const command = `export ARDUINO\_DIR=../../arduino-1.8.16 && export ARDMK\_DIR=./Makefile && export AVR\_TOOLS\_DIR=/usr/include && make && avrdude -F -V -c arduino -p ATMEGA328P -P /dev/${device} -b 115200 -U flash:w:build-uno/code-wiring\_.hex 2>&1`

  return new Promise((resolve) => {

    const process = spawn(command, { cwd: 'code-wiring', shell: true })

    process.stdout.on('data', (data) => {

      broadcast(wss, { type: 'log', stdout: data.toString('utf8') })

    })

    process.stderr.on('data', (data) => {

      broadcast(wss, { type: 'log', stderr: data.toString('utf8') })

    })

    process.on('close', (code) => {

      resolve(code)

    })

    process.on('error', (err) => {

      console.error(inspect(err))

    })

  })

}

async function queueNext () {

  if (currentRequest) {

    currentRequest.ws.send(JSON.stringify({ type: 'requestComplete' }))

  }

  currentRequest = queue[0]

  if (!currentRequest) {

    timeout = null

    isRunning = false

    sendQueueInfo()

    return

  }

  currentRequest.ws.send(JSON.stringify({ type: 'yourTurn' }))

  broadcast(wss, { type: 'currentCode', code: currentRequest.code, lang: currentRequest.lang })

  let exitCode = null

  if (currentRequest.lang === 'c') {

    exitCode = await processC(currentRequest)

  } else if (currentRequest.lang === 'wiring') {

    exitCode = await processWiring(currentRequest)

  }

  let waitTime

  if (exitCode === 0) {

    waitTime = TIMEOUT\_NORMAL

  } else {

    waitTime = TIMEOUT\_ERROR

  }

  timeout = setTimeout(queueNext, waitTime)

  timeoutStartTime = Date.now()

  timeoutLength = waitTime

  console.log('waiting for ' + waitTime \* 0.001 + ' seconds')

  isRunning = true

  queue.shift()

  sendQueueInfo()

}

function broadcast (wss, message) {

  wss.clients.forEach(function each (client) {

    if (client.readyState === WebSocket.OPEN) {

      client.send(JSON.stringify(message))

    }

  })

}

function queueBroadcastMessage () {

  return { type: 'queue', queueLength: queue.length, timeoutStartTime, timeoutLength, isRunning }

}

function sendQueuePositions () {

  for (let i = 0; i < queue.length; i++) {

    queue[i].ws.send(JSON.stringify({ type: 'queuePosition', position: i + 1 }))

  }

}

function sendQueueInfo () {

  broadcast(wss, queueBroadcastMessage())

  sendQueuePositions()

}

## 2.3 Sestavování výsledného programu ze zadaného kódu

### 2.3.1 Jazyk Wiring

Pro sestavení kódu v jazyce Wiring jsem se rozhodl pro použití Arduino.mk, což je makefile pro sestavování Wiring kódu. Bylo nutno stáhnout Arduino IDE pro ARM procesory a nainstalovat systémové balíčky potřebné pro sestavování programů:

$ sudo apt install gcc-avr binutils-avr avr-libc avrdude arduino-mk

Další potřebné programy jsou na operačním systému Raspberry PI OS již nainstalovány.

Sestavování se zahájí v Node.js, kde se vytvoří child process s příkazem:

$ export ARDUINO\_DIR=../../arduino-1.8.16 && export ARDMK\_DIR=./Makefile && export AVR\_TOOLS\_DIR=/usr/include && make && avrdude -F -V -c arduino -p ATMEGA328P -P /dev/${device} -b 115200 -U flash:w:build-uno/code-wiring\_.hex 2>&1

Tento příkaz nejdříve sestaví program pomocí make a poté ho pomocí avrdude flashne do Arduina. V Javascriptu se vloží správná „adresa“ Arduina.

### 2.3.2 Jazyk C

Sestavování kódu v jazyce C je mnohem snazší, pomocí avr-gcc se sestaví program a opět se pomocí avrdude flashne do Arduina:

$ avr-gcc -Os -DF\_CPU=16000000UL -mmcu=atmega328p -c -o code.o code.c && avr-gcc -mmcu=atmega328p code.o -o code && avr-objcopy -O ihex -R .eeprom code code.hex && avrdude -F -V -c arduino -p ATMEGA328P -P /dev/${device} -b 115200 -U flash:w:code.hex 2>&1

Stejně jako v minulém případě poslední část příkazu „2>&1“ slouží k odstranění ANSI escape kódů, které slouží k formátování výstupu. Jelikož výstup je zobrazován v HTML, kde ANSI escape kódy nefungují, musí být odstraněny.

# 3 Živé vysílání

Pro vysílání jsem se rozhodl použít software OBS Studio, které lze jednoduše nastavit pro streamování videa na Twitch prostřednictvím Auto Configuration Wizard, kam se pouze vloží hlavní kód z Twitche, kde bylo nutno vytvořit účet, a zbytek se nastaví automaticky.

Prvotní myšlenkou bylo streamovat přímo z Raspberry PI, pomocí kamery připojené přes USB. Zjistil jsem však, že procesory s architekturou ARM, který používá Raspberry PI, nejsou oficiálně podporovány. Rozhodl jsem se sestavit si program OBS Studio podle návodu na Raspberry PI, avšak výsledný program nebyl dostatečně stabilní. Proto bylo nutno nainstalovat OBS Studio na druhém počítači, který má podporovanou architekturu x86\_64, kde s provozem OBS Studia nebyl problém. Jako kamera je využívána HD webkamera.

# 4 Hardwarová část

## 4.1 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 a SSH tunel pomocí služby localhost.run.

## 4.2 Arduino UNO

Arduino UNO je zapojeno přes USB do Raspberry PI. Do Arduina jsou zapojeny následující periferie:

* Červená, oranžová a zelená LED
* RGB LED
* Tlačítko
* Fotorezistor

Sestavený kód se nahrává do flash paměti Arduina, která má omezenou životnost. Backend stanovuje interval nahrávání nového kódu na minimálně 30 sekund, což částečně znemožňuje provedení útoku, ve kterém by uživatel nahrával rychle zkompilovatelný kód bez pauzy a tím flash paměť zničil. Po určité době však bude nutné Arduino vyměnit, jelikož i při běžném užívání teoreticky jednoho dne dojde ke zničení flash paměti.

# 5 Příkladové kódy

V aplikaci je několik příkladových kódů, které je možné vybrat pomocí select vstupu ve spodní části webové aplikace. V závislosti na vybraném jazyce se zobrazí dostupné příkladové kódy.

## 5.1 V jazyce Wiring

Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 1Možnosti příkladových kódů v jazyce Wiring

Default (empty)

void setup() {

}

void loop() {

}

Blink

constexpr int LED\_PIN = 7;

void setup() {

  pinMode(LED\_PIN, OUTPUT);

}

void loop() {

    digitalWrite(LED\_PIN, HIGH);

    delay(1000);

    digitalWrite(LED\_PIN, LOW);

    delay(1000);

}

Button Pull-Up

constexpr int BTN\_PIN = 2;

constexpr int LED\_PIN = 7;

void setup() {

    pinMode(BTN\_PIN, INPUT\_PULLUP);

    pinMode(LED\_PIN, OUTPUT);

}

void loop() {

  int sensorVal = digitalRead(BTN\_PIN);

  if (sensorVal == HIGH) {

    digitalWrite(LED\_PIN, LOW);

  } else {

    digitalWrite(LED\_PIN, HIGH);

  }

}

Traffic Lights

constexpr int RED\_LED = 3;

constexpr int YELLOW\_LED = 4;

constexpr int GREEN\_LED = 5;

void setup() {

    pinMode(RED\_LED, OUTPUT);

    pinMode(YELLOW\_LED, OUTPUT);

    pinMode(GREEN\_LED, OUTPUT);

}

void loop() {

  digitalWrite(RED\_LED, HIGH);

  delay(3000);

  digitalWrite(YELLOW\_LED, HIGH);

  delay(1000);

  digitalWrite(RED\_LED, LOW);

  digitalWrite(YELLOW\_LED, LOW);

  digitalWrite(GREEN\_LED, HIGH);

  delay(3000);

  digitalWrite(GREEN\_LED, LOW);

  digitalWrite(YELLOW\_LED, HIGH);

  delay(1000);

  digitalWrite(YELLOW\_LED, LOW);

}

RGB LED

constexpr int RED\_LED = 9;

constexpr int GREEN\_LED = 10;

constexpr int BLUE\_LED = 11;

void setup() {

    pinMode(RED\_LED, OUTPUT);

    pinMode(GREEN\_LED, OUTPUT);

    pinMode(BLUE\_LED, OUTPUT);

}

void loop() {

  for(int i = 0; i < 256; i++) {

      analogWrite(RED\_LED, i);

      delay(4);

  }

  for(int i = 0; i < 256; i++) {

      analogWrite(GREEN\_LED, i);

      delay(4);

  }

  for(int i = 0; i < 256; i++) {

      analogWrite(BLUE\_LED, i);

      delay(4);

  }

  for(int i = 255; i >= 0; i--) {

      analogWrite(RED\_LED, i);

      delay(4);

  }

  for(int i = 255; i >= 0; i--) {

      analogWrite(GREEN\_LED, i);

      delay(4);

  }

  for(int i = 255; i >= 0; i--) {

      analogWrite(BLUE\_LED, i);

      delay(4);

  }

}

Photoresistor

constexpr int LED\_1 = 3;

constexpr int LED\_2 = 4;

constexpr int LED\_3 = 5;

constexpr int LED\_4 = 6;

constexpr int LED\_5 = 7;

void setup() {

    pinMode(LED\_1, OUTPUT);

    pinMode(LED\_2, OUTPUT);

    pinMode(LED\_3, OUTPUT);

    pinMode(LED\_4, OUTPUT);

    pinMode(LED\_5, OUTPUT);

}

void loop() {

    digitalWrite(LED\_1, LOW);

    digitalWrite(LED\_2, LOW);

    digitalWrite(LED\_3, LOW);

    digitalWrite(LED\_4, LOW);

    digitalWrite(LED\_5, LOW);

    int reading = analogRead(A0);

    if(reading > 50) {

      digitalWrite(LED\_1, HIGH);

    }

    if(reading > 100) {

      digitalWrite(LED\_2, HIGH);

    }

    if(reading > 200) {

      digitalWrite(LED\_3, HIGH);

    }

    if(reading > 350) {

      digitalWrite(LED\_4, HIGH);

    }

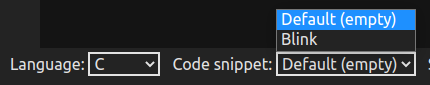
    if(reading > 600) {

      digitalWrite(LED\_5, HIGH);

    }

}

## 5.2 V jazyce C



Obrázek 2Možnosti příkladových kódů v jazyce C

Default (empty)

#include <avr/io.h>

int main(void) {

}

Blink

#include <avr/io.h>

#include <util/delay.h>

#define MS\_DELAY 200

int main(void)

{

  DDRD |= \_BV(DDD7);

  while (1)

  {

    PORTD |= \_BV(PORTD7);

    \_delay\_ms(MS\_DELAY);

    PORTD &= ~\_BV(PORTD7);

    \_delay\_ms(MS\_DELAY);

  }

}

# 6 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 kompilátor, 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.