Gymnázium, Praha 6, Arabská 14
Obor programování

MATURITNÍ PRÁCE

Vladimír Samojlov

Záběry fotek

Duben 2025

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne ..... Vladimír Samojlov .....

Název práce: Záběry fotek

Autoři: Vladimír Samojlov

Abstrakt: Cílem této práce bylo vytvořit webovou aplikaci, která umožňuje generování klipů z jednotlivých částí jednoho nebo více obrázků vybraných uživatelem. U každé části lze nastavit směr a délku trvání pohybu, například posun zespodu nahoru během 2 sekund. Každou část lze dále upravovat pomocí nástrojů určených pro práci s obrázky. Spojením jednotlivých klipů, mezi nimiž je možné nastavit přechody, vznikne finální video. Aplikace také poskytuje uživatelům možnost registrace a přihlášení, což jim zpřístupňuje funkce pro správu a sledování jejich vytvořených videí. Videa vytvořená z obrázků mohou sloužit například jako prezentační videa, reklamní spoty nebo úvodní sekvence pro různé projekty.

Klíčová slova: Přechod, Fotografie, Klip, Záběr, React, JavaScript, Express, SQLite

Title: Photo Shots

Authors: Vladimír Samojlov

Abstract: The goal of this project was to develop a web application that allows users to generate clips from selected parts of one or more images. For each part, the direction and duration of movement can be set, such as a slide from bottom to top over 2 seconds. Each part can also be further edited using tools designed for image manipulation. By combining individual clips with transitions that can be configured between them, a final video is created. The application also provides users with the ability to register and log in, granting them access to features for managing and viewing videos they have created. Videos created from images can serve various purposes, such as presentation videos, advertisements, or introductory sequences for different projects.

Keywords: Transition, Photography, Clip, Shot, React, JavaScript, Express, SQLite

Obsah

1. Úvod

Tento dokument představuje návrh a realizaci webové aplikace zaměřené na tvorbu videí z obrázků nebo jejich jednotlivých částí. Aplikace umožňuje uživatelům snadno upravovat obrázky, rozdělovat je na části, vytvářet klipy a následně je spojovat do finálního videa. Inspirací pro tento projekt byla autorova snaha posunout prezentaci svých uměleckých děl na vyšší úroveň – místo jednoduchého sdílení fotografií na sociálních sítích chtěl nabídnout dynamickou formu prezentace prostřednictvím krátkých klipů.

Tato práce dále detailně popisuje klíčové funkce aplikace, použité technologie i vizuální podobu uživatelského rozhraní. V textu jsou rovněž uvedeny technické výzvy, se kterými se autor během vývoje setkal, a způsoby, jak byly řešeny.

Zadání

Webová aplikace má mít níže uvedenou funkcionalitu:

1. Nahrání obrázku z lokálního úložiště nebo prostřednictvím URL.
2. Hlavní scéna – skládá se ze čtyř klíčových funkcí:
    a) Upravit obrázek za pomoci daných filtrů.
    b) Rozdělit obrázku na jednotlivé částice.
    c) Výběr částic pro tvorbu klipu.
    d) Sestavení výsledného videa z vytvořených klipů a možnost jeho uložení nebo stažení.
3. Uživatelská plocha – po úspěšné registraci nebo přihlášení získá uživatel přístup ke správě svých videí.

2. Architektura a technologie

V této kapitole je vysvětlena struktura aplikace, použitá architektura a zvolené technologie. Jsou zde uvedeny klíčové komponenty, jejich propojení a důvody výběru konkrétních nástrojů a technologií, které byly využity při vývoji.

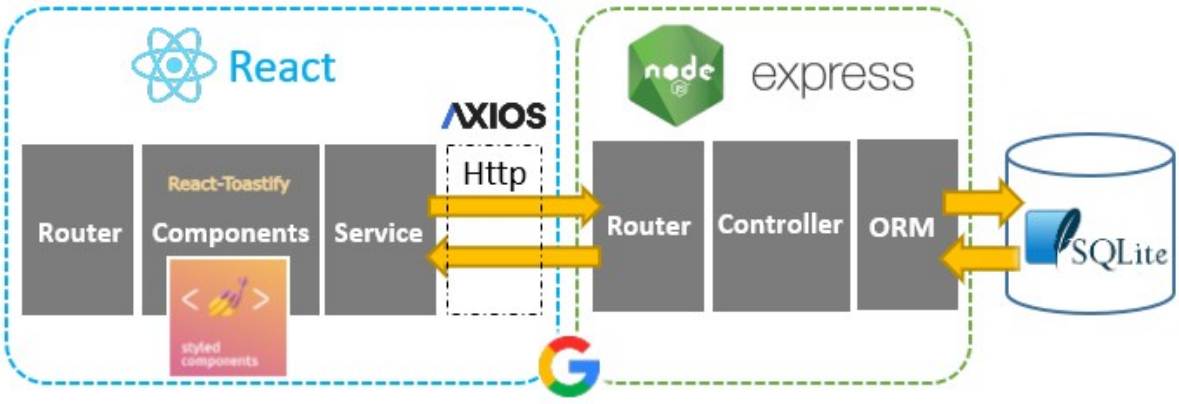
2.1 Architektura řešení

Webová aplikace byla navržena tak, aby byla uživatelsky přívětivá, snadno ovladatelná a především umožňovala efektivní komunikaci mezi klientskou částí (frontend) a serverovou částí (backend). Dílčí komponenty spolu komunikují za pomoci internetového protokolu HTTP, který slouží pro přenos hypertextových dokumentů, ale také dalších datových formátů, jako jsou JSON, XML, obrázky a další typy souborů. [1]

Nyní si představíme základní strukturu aplikace:

- **Frontend** – Uživatelské rozhraní, které interaguje s uživatelem a zobrazuje data.
- **Backend** – Zpracovává logiku aplikace, zajišťuje autentifikaci uživatelů.
    - **Databáze** – Ukládá a načítá data potřebná pro správnou funkčnost aplikace.

Na obrázku níže je vyobrazeno celé schéma aplikace včetně použitých technologií.



Obrázek 2.1: Struktura webové aplikace Záběry fotek

2.2 Technologie

2.2.1 Frontend

Pro dynamické vykreslování stránky (renderování) byl zvolen framework React, který využívá princip Single Page Application (SPA). Tento přístup umožňuje rychlé načítání a aktualizaci obsahu bez nutnosti znovu načítat celou stránku. Na rozdíl od běžného přístupu, kdy jsou HTML a JavaScript uloženy v samostatných souborech, React spojuje vše potřebné do jednoho souboru, kde komponenta obsahuje jak HTML, tak i logiku napsanou v JavaScriptu, případně v TypeScriptu. V případě autora byl zvolen programovací jazyk JavaScript (zkráceně *JS*), se kterým měl již předchozí zkušenosti.

Jelikož knihovna React se neustále v dnešní době vyvíjí, obsahuje mnoho zajímavých funkcí, které lze do projektu implementovat. Jednou z nich jsou *React Hooks*, jenž slouží k využívání různých funkcí daných komponent. Nyní si popíšeme hlavní funkce vybraných *Hooks*, které jsou v práci implementovány:

- ***State*** – Uchování a aktualizace stavu komponenty.
- ***Context*** – Poskytování informací komponentám od vzdálených rodičů.
- ***Ref*** – Uchovávání hodnot, které nejsou přímo používány pro vykreslování.
- ***Effect*** – Synchronizace komponent s externími systémy.
- ***Performance*** – Optimalizace výkonu aplikace tím, zabraňují zbytečnému znovu vykreslování. [2]

Na základě výše uvedených funkcí lze konstatovat, že *React Hooks* primárně slouží k usnadnění sdílení logiky mezi jednotlivými komponentami. Na obrázku 2.1 je uveden příklad komponenty v jazyce JavaScript, která využívá *State Hook* pro správu stavu textového pole. K nabízeným funkcím React patří také stylování jednotlivých komponent, pro které byla v tomto případě použita knihovna *Styled-components*. Tato knihovna umožňuje psát kaskádové styly (*CSS*) přímo v JavaScriptu a přiřazovat styly konkrétním komponentám.

Knihovna *Styled-components* zároveň umožňuje dynamicky měnit styly na základě stavu nebo vlastností komponenty, což je klíčové pro vytváření interaktivních uživatelských rozhraní.

Dalším důležitým aspektem frontendu je zobrazování dat z databáze, která jsou poskytována ze strany backendu. K jejich získání byla použita knihovna *Axios*, která slouží k vytváření HTTP požadavků z webového prohlížeče nebo Node.js. Zjednodušuje proces odesílání asynchronních požadavků na server a následné zpracování odpovědi. [3]

```
import React, { useState } from "react";

/**
 * Komponenta pro vstupní pole.
 * Umožňuje zadávání textu a zobrazení aktuálně zadané hodnoty.
 */
const TextInput = () => {

 // State Hook pro správu hodnoty textového pole
 const [text, setText] = useState('');

 // Funkce pro aktualizaci hodnoty textového pole zavolána při změně
 const handleChange = (event) => {
  setText(event.target.value);
 };

 return (
  <div>
   <h1>Zadaný text: {text}</h1>
   <input
    // Typ vstupního pole
    type="text"
    // Hodnota input zobrazuje aktivní hodnotu
    value={text}
    // Funkce pro aktualizaci stavu
    onChange={handleChange}
   />
  </div>
 );
};
export default TextInput;
```

Výpis kódu 2.2: Ukázka komponenty v jazyce *JavaScript* využívající knihovnu *React*

2.2.2 Backend

Stejně jako u frontendu byl pro backend zvolen staticky typovaný jazyk JavaScript, zejména díky rychlosti vývoje a širokému ekosystému knihoven dostupných v asynchronním runtime prostředí *Node.js*. [3]

Elementárním způsobem komunikace s frontendem je REST API, které bylo vytvořeno pomocí frameworku *Express*. Ten je v současnosti jedním z nejvíce používaných webových frameworků, a to ve spojení s *Node.js*. [4] Tento framework slouží jako prostředník mezi databází *SQLite* a klientskou aplikací *React*. REST API poskytuje jednoduché a přehledná data prostřednictvím standardních HTTP metod, jako jsou GET, POST, PUT a DELETE. Odpovědi jsou následně vráceny ve formátu JSON, který zaručuje bezpečnou integraci se stranou frontend.

Pomocí *Express* jsou v aplikaci prováděny také databázové operace, jako například filtrování videí podle jména uživatele. Tyto operace jsou realizovány prostřednictvím dotazů na *SQLite*. Díky jednoduchosti *SQLite* je možné snadno upravit strukturu databáze tak, aby vyhovovala specifickým potřebám aplikace. [5] Na obrázku 2.3 je zobrazen příklad kódu pro provedení dotazu na *SQLite*.

2.2.3 Databáze

Jak již bylo zmíněno v předchozí podkapitole, pro tento projekt byla vybrána databáze *SQLite*, a to především kvůli její nízké náročnosti na konfiguraci a efektivitě při práci s menšími a středně velkými datovými sadami. Tento typ databáze není přímo srovnatelný s databázovými systémy SQL typu klient/server, jako jsou *MySQL*, *PostgreSQL* a podobně, jelikož *SQLite* klade důraz především na efektivitu, spolehlivost, nezávislost a jednoduchost. [5] Díky své architektuře navíc nevyžaduje spuštění samostatného serveru, což minimalizuje samotné požadavky na konfiguraci.

```
// Načtení a zpřístupnění knihovny sqlite3
const sqlite3 = require('sqlite3').verbose();

// Vytvoření připojení k databázi
const db = new sqlite3.Database('data.db');

// Příklad ID daného klipu
const id_clip = 1;

// Vytvoření dotazu pro vyhledání klipu podle id_clip
db.get('SELECT * FROM clips WHERE id = ?', [id_clip], (err, row) => {

 // Vypsání případné chyby
 if (err) {
  console.error(err.message);
  return;
 }

 // Vypíše celý záznam uživatele, pokud existuje
 if (row) {
  console.log(row);

 } else {
  console.log('Uživatel nenalezen');
 }
});

// Uzavření připojení k databázi
db.close();
```

Výpis kódu 2.3: Ukázka kódu pro provedení dotazu na databázi *SQLite* v *Node.js*

3. Webová aplikace

Tato kapitola se zaměřuje na popis použitých metod při vývoji webové aplikace, zahrnující jak backend, tak frontend. Nejprve představuje části samotné aplikace a poté detailně objasňuje zvolené přístupy k jejich implementaci.

3.1 Obrazovky

3.1.1 Domovská stránka

Domovská stránka, často označovaná jako homepage, hraje zásadní roli v oblasti UX/UI designu a jeho psychologických aspektů. Jejím primárním cílem je zaujmout uživatele na první pohled, vytvořit pozitivní první dojem a navodit pocit, že je návštěvník stránky vítán. Současně poskytuje důležité informace a jasnou navigaci, která by měla výrazně uživateli usnadnit orientaci na webu. [6]

Na základě zmíněných pravidel grafického designu byla struktura webu rozdělena do struktury skládající se z těchto částí:

1. Header – Záhlaví stránky
2. Hero section – Úvodní sekce
3. Sekce s obsahem – Přehled klíčových funkcí
4. Testimonial Section – Sekce s referencemi
5. Footer – Patička stránky

3.1.2 Autentizace

Přestože přihlášení není nutné pro vytvoření obrázků a stažení výsledného videa, uživatelé se mohou přihlásit nebo zaregistrovat buď běžným způsobem, nebo pomocí účtu Google. Registrovaní uživatelé mají výhodu v podobě přístupu ke svým projektům, nastavením účtu a dalším personalizovaným funkcím.

Prvním způsobem je klasická registrace, do které je zapotřebí zadat osobní údaje – e-mail, heslo a celé jméno. Tento způsob je dle mého názoru ideální pro ty, kdo chtějí mít nezávislý přístup ke svým datům a projektům. Pro zajištění ochrany aplikace jsou dále implementována opatření proti útokům, jako je *SQL injection*, umožňující neoprávněnou manipulaci s databází, a *XSS* (*Cross-Site Scripting*), který může ohrozit bezpečnost uživatelských dat.

Druhou možností je registrace přes službu Google, která umožňuje uživatelům rychlý přístup k funkcím aplikace bez nutnosti vytvářet samostatný účet s vlastním heslem. Při přihlášení je nutné ověřit, zda uživatel s Google účtem již má v aplikaci vytvořený účet. To lze zjistit na základě sloupce *type* v tabulce *user*, který určuje typ uživatele (default, google). Celé schéma tabulky je zobrazeno na obrázku č. ___.

V případě zapomenutí hesla si uživatel může obnovit přístup přes e-mail, kam obdrží šestimístný jednorázový OTP kód platný po dobu 5 minut. Tímto by se měla výrazně zvýšit bezpečnost vůči neoprávněným přístupům nebo zneužití při resetování hesla.

3.1.3 Správa uživatelského účtu

3.1.3.1 Přehled projektů

3.1.3.2 Konfigurace účtu

3.1.4 Vyzkoušení aplikace

3.1.5 Informační scény

3.1.5.1 O projektu

3.1.5.2 Služby

3.1.5.3 Kontakt

3.2 Videoprodukce a nástroje

Tato kapitola se zaměřuje na klíčovou funkcionalitu webové aplikace, která umožňuje sestavení výsledného videa z jednotlivých klipů nebo obrázků. Uživatelské rozhraní stránky je rozděleno do dvou částí: ***Sidebar*** (umístěný vlevo) a ***CanvasContent*** (nachází se napravo od *Sidebaru* a pokrývá zbývající plochu stránky).

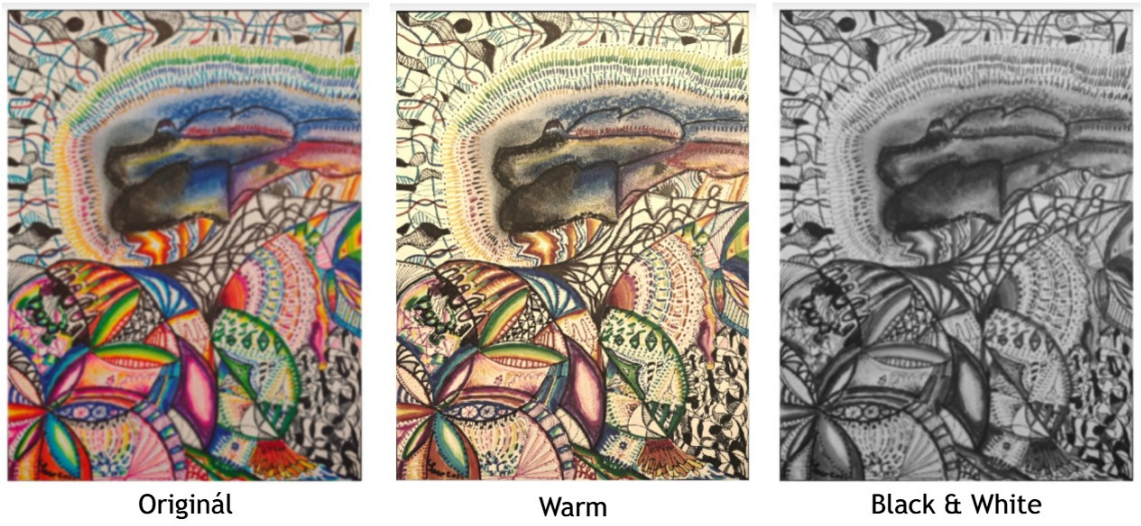
Sidebar zobrazuje jednotlivé kroky potřebné k vytvoření videa. Při výběru konkrétního kroku v *Sidebaru* se automaticky aktualizuje obsah komponenty *CanvasContent*, aby odpovídal aktuálně zvolené fázi.

V následujících kapitolách podrobně popíši kroky sestavení videa, nástroje pro závěrečnou fázi a proces exportu vytvořených klipů do videa.

3.2.1 Fáze pro vytvoření videa

3.2.1.1 Úprava obrázku

V této sekci si uživatel může vybrat z široké škály filtrů pro nahraný obrázek, například *Warm* nebo *Black & White*, jak je znázorněno na obrázku 3.1. Po výběru filtru v komponentě *Sidebar* se kaskádové styly pro filtr okamžitě aplikují na element *canvas*, jenž obsahuje daný obrázek.



Obrázek 3.1: Ukázka vybraných filtrů pro nahraný obrázek.

3.2.1.2 Rozdělení fotografie na částice

Následuje proces rozdělení, který byl navržen pro zachycení snímku odpovídajícího konkrétní částici z daného obrázku. Pro získání částic lze pomocí dvou tlačítek přidávat linie ve formě sloupců nebo řádků. Každá linie má tři parametry: id, typ (vertikální nebo horizontální) a pozice (souřadnice x nebo y). Na základě typu linie se její pozice určuje buď na polovině šířky, nebo výšky plochy *canvas*. Pokud je však na ploše již umístěna nějaká linie, nová linie se přidá do poloviny vzdálenosti S mezi původní linií a začátkem plochy. Počátek plochy je definován pro vertikální linii – [0, S], pro horizontální linii – [S, 0].

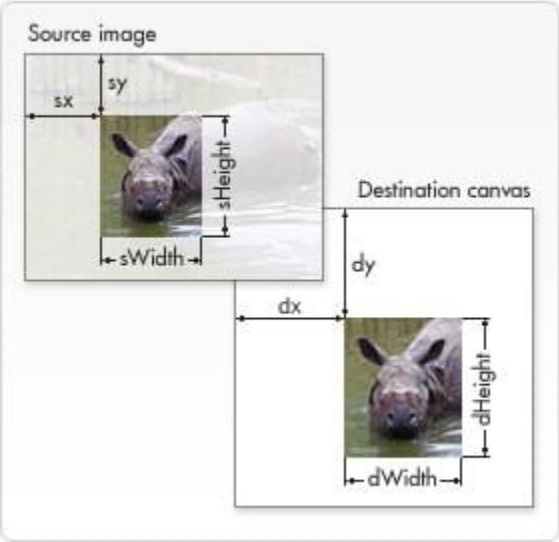
Přidané linie je možné libovolně posouvat v příslušném směru pomocí operace *drag-resize*. K vytvoření této funkce je v jazyce JavaScript nutné uplatnit tři základní typy událostí:

- *mousedown* – Detekce zahájení tahu.
- *mousemove* – Tah je v pohybu nad prvkem.
- *mouseup* – Ukončení tahu.

Kombinací těchto událostí s pohybem kurzoru myši vzniká plně funkční metoda *drag-resize*, která umožňuje přetahování jednotlivých linií na ploše *canvas*. Jakmile uživatel dokončí úpravy a přejde k další fázi, dojde k automatickému generování dílčích částic. Tyto dílky jsou vytvořeny pomocí algoritmu, který generuje částice na základě zadaných souřadnic linií a ukládá je ve formě URL adres, jež lze kdykoliv načíst. Algoritmus prochází obrázek po řádcích, to znamená, že pro každý řádek postupuje zleva doprava, a po dokončení zpracování jednoho řádku se posune o jednu pozici níže, aby začal nový řádek. Konkrétní části algoritmu, včetně zobrazení částic na plochu, jsou uvedeny v kódu 3.3.

Pro vykreslení jednotlivých částic bylo využito rozhraní (interface) *CanvasRendering-Context2D*, které je součástí služby *Canvas API*. Interface poskytuje 2D kontext pro vykreslování na kreslicí ploše prvku *canvas* a slouží k vytváření tvarů, textů, obrázků a dalších grafických objektů. [9]

V tomto rozhraní byla použita metoda *drawImage()*, která umožnila vyříznutí jednotlivých dílků obrázku na základě vzdáleností (*sWidth*, *sHeight*) od počátku [0, 0] (horní levý roh plochy) k souřadnicím určeným liniemi [*sx*, *sy*], nebo mezi dvěma liniemi [$sx_2 - sx_1$, $sy_2 - sy_1$]. Následně byl dílek zobrazen na plochu s odpovídající šířkou a výškou (*dWidth*, *dHeight*), přičemž jeho pozice na plátno byla nastavena do počátku [0, 0] ([*dx*, *dy*]). Pro lepší vizualizaci parametrů metody *drawImage()* jsou znázorněny na obrázku 3.2.



Obrázek 3.2: Proces výřezu částice z obrázku. [10]

3.2.1.3 Výběr částic

Po vyobrazení jednotlivých částic má uživatel možnost určit pořadí, ve kterém se tyto částice objeví na časové ose *Timeline*, která je součástí závěrečné fáze pro vytvoření videa (*Vytvoření klipů*). Na začátku výběru se uživateli zobrazí všechny částice s příslušnými parametry, přičemž každá je umístěna v samostatné kartičce, na jejímž vrchním okraji je zobrazeno pořadové číslo. Pokud uživatel zruší výběr některé z kartiček, která již měla přiřazené pořadí, program automaticky upraví pořadí zbývajících prvků tak, aby odpovídalo aktuálnímu stavu v daném seznamu kartiček.

```
    // Souřadnicová pole
    const finalPositionX = [0, ...positionX, image.width];
    const finalPositionY = [0, ...positionY, image.height];

    // Pole pro částice
    const pieces = [];

    // ID částice
    let idImg = 0;

    // Procházení řádků
    for (let i = 0; i < finalPositionY.length - 1; i++) {

        // Procházení sloupců
        for (let j = 0; j < finalPositionX.length - 1; j++) {

            // Pozice x, y na ploše
            const x = finalPositionX[j];
            const y = finalPositionY[i];

            // Šířka a výška částice
            const width = finalPositionX[j + 1] - x;
            const height = finalPositionY[i + 1] - y;

            // Nová plocha
            const pieceCanvas = document.createElement('canvas');
            pieceCanvas.width = width;
            pieceCanvas.height = height;

            // Nová částice
            const pieceContext = pieceCanvas.getContext('2d');
            pieceContext.filter = getFilter();

            // Vykreslení částice do plochy canvas
            pieceContext.drawImage(image, x, y, width, height, 0, 0,
                                   width,height);

            // Přidání údajů o částicích do pole
            pieces.push({
                id: idImg,
                src: pieceCanvas.toDataURL()
            });
            idImg++;
        }
    }
```

Výpis kódu 3.3: Algoritmus pro generování částic a jejich vykreslení na plochu *canvas*.

3.2.1.4 Sestavení videa

Jak již bylo zmíněno, video se skládá z jednotlivých částic, které tvoří fotografie nebo klipy. Tato sekce stránky, jejímž cílem je umožnit uživateli vytvořit plně funkční video pomocí dostupných nástrojů, je rozdělena do tří hlavních částí: časové osy *Timeline*, nástrojů pro práci s částicemi či videem a plochy *canvas*, která zobrazuje obsah propojený s komponentou *Timeline*.

Na prvku *Timeline* jsou vidět dílčí vygenerované částice, jež jsou rozloženy po celé její délce. Pro každou částici lze libovolně měnit její šířku, přičemž maximální hranice je stanovena tak, aby nedocházelo k překrytí s jinými částicemi na této ose. Dále je možné pro danou částici aplikovat klip, přičemž je zapotřebí použít dva nástroje: směry a kamera. Pro lepší pochopení fungování tohoto klipu si ukážeme na obrázku 3.4, kde je zobrazená jedna z částic výstřižku původní fotografie. V pravém dolním rohu je znázorněn čtvercový prostor představující velikost kamery, který se v běžícím programu zobrazuje jako průhledná oblast pohybující se plynule ve stanoveném směru.

Obrázek 3.4: Příklad vyříznuté částice z fotografie, s velikostí kamery (čtverec) a směrem jejího procházení přes částici.

Pro plynulost klipu jsme použili knihovní metodu *requestAnimationFrame()*, která informuje prohlížeč, že je žádost o provedení animace. Tato metoda zároveň požaduje, aby byla zavolána uživatelem definovaná zpětná funkce (callback) před dalším přetřením obrazovky (repaint). Tím se zajistí, že klip běží synchronizovaně s vykreslováním obrazovky. [11] Místo funkce *callback* byla v programu volána funkce *createClip*, která vykreslovala na plátno zachycený snímek kamery během procházení dráhy vyříznuté částice. Tato funkce je znázorněna na obrázku 3.5 s podrobnými komentáři, které zahrnují i popis jednotlivých částí metody *drawImage()*.

```
/** Funkce pro vytvoření klipu **/
const createClip = () => {

    // Očištění plochy před novým vykreslením
    ctx.clearRect(0, 0, canvas.width, canvas.height);


    // Vykreslení konkrétního snímku
    ctx.drawImage(
        img,                    // Nahraný obrázek
        coordinateX,            // Aktuální pozice kamery na ose X (sx)
        coordinateY,            // Aktuální pozice kamery na ose Y (sy)
        cameraWidth,            // Šířka kamery (sWidth)
        cameraHeight,           // Výška kamery (sHeight)
        0,                      // Počátek osy X (dx)
        0,                      // Počátek osy Y (dy)
        canvas.width,           // Šířka plochy (dWidth)
        canvas.height           // Výška plochy (dHeight)
    );
}
// Zavolání requestAnimationFrame pro plynulé vykreslování
requestAnimationFrame(createClip);
```

Výpis kódu 3.5: Funkce pro vytvoření klipu za využití metody *requestAnimationFrame*.

3.2.2 Nástroje

Tato podkapitola představuje a popisuje nástroje, které rozšiřují možnosti práce s částicemi nebo nabízejí další funkce pro tvorbu videa.

3.2.2.1 Kamera

Kamera má vždy počáteční a cílový bod, mezi kterými se plynule pohybuje po předem určené dráze. Jejím cílem je zachytit konkrétní detaily z vybraného výřezu během procházení úseku. Kromě toho musí kamera obsahovat i rychlost pohybu po dráze, která je automaticky vypočítána na základě celkového času trvání pohybu částice na časové ose a vzdálenosti mezi startovním a cílovým bodem. V sekci nástrojů lze pro každý klip nastavit přesné parametry velikosti kamery na základě pixelů (px) nebo poměru vůči výřezu.

3.2.2.2 Směry

Dalším důležitým nástrojem jsou směry, které umožňují nastavit orientaci pohybu, a to ve vertikálním, horizontálním nebo diagonálním směru. Kromě základních směrů jsou k dispozici také specifičtější možnosti. Patří mezi ně například oddálení a přiblížení, rotace ve směru či proti směru hodinových ručiček nebo vytvoření vlastního lineárního směru. Vlastní směr si uživatel může nastavit pomocí souřadnic dvou bodů [X, Y], což mu umožňuje přizpůsobit pohyb zcela podle svých představ.

3.2.2.3 Přechody

Podobně jako u směrů i zde rozlišujeme různé typy efektů přechodů mezi dvěma částicemi (např. efekt rozmazání, převrácení atd.). Proces vytváření přechodu začíná výběrem typu přechodu a následným označením dvou sousedních částic. Program následně zkontroluje, zda jsou obě částice skutečně sousední a zda jsou vzájemně propojeny v komponentě *Timeline*. Pokud je vše v pořádku, mezi těmito dvěma prvky se zobrazí spojovací linie. V opačném případě se zobrazí notifikace o chybě.

3.2.2.3 Rozměry plátna

Vzhledem k rychlému rozvoji technologií obsahuje aplikace nástroj, který umožňuje dynamicky měnit velikost plátna *canvas*. Uživatelé si mohou přizpůsobit rozměry plátna pomocí nabídky běžně používaných formátů, jako jsou širokoúhlé obrazovky (21:9), mobilní zařízení (16:9) nebo platformy s unikátními poměry stran, například 4:3 a další.

3.2.2 Exportování do videa

Proces exportu se zahájí stisknutím tlačítka „Export“, čímž dojde k automatickému procházení časové osy od jejího počátku až k poslední částici. Mezitím se na stránce zobrazí dialogové okno, ve kterém má uživatel možnost pojmenovat video a přidat volitelný popis. Po dokončení průchodu časovou osou se uživateli nabídnou tři možnosti prostřednictvím tlačítek – „Uložit projekt“, „Stáhnout“ a „Pokračovat“. Pro neregistrované uživatele je k dispozici možnost „Stáhnout“, která umožňuje exportovat video ve formátu MP4, stejně jako volba „Pokračovat“, jež přesměruje uživatele zpět do závěrečné fáze procesu tvorby videa.

Pro úplné zachování projektu je nezbytné převést video z nezpracovaného formátu Blob do řetězce Base64. Base64 bylo zvoleno primárně z důvodu efektivního ukládání a přenosu dat, neboť umožňuje textovou reprezentaci binárního obsahu. Tento převod je realizován pomocí metody *readAsDataURL()* objektu *FileReader*, která načte obsah souboru a zakóduje jej do formátu datového URL. [12] Po dokončení konverze dojde k uložení dat do databáze prostřednictvím API požadavků odeslané ze strany klienta. Uložená videa se přihlášenému uživateli zobrazí v sekci „Přehled projektů“ na stránce správy účtu.

3.2 Komunikace

3.2.1 Endpointy backendu

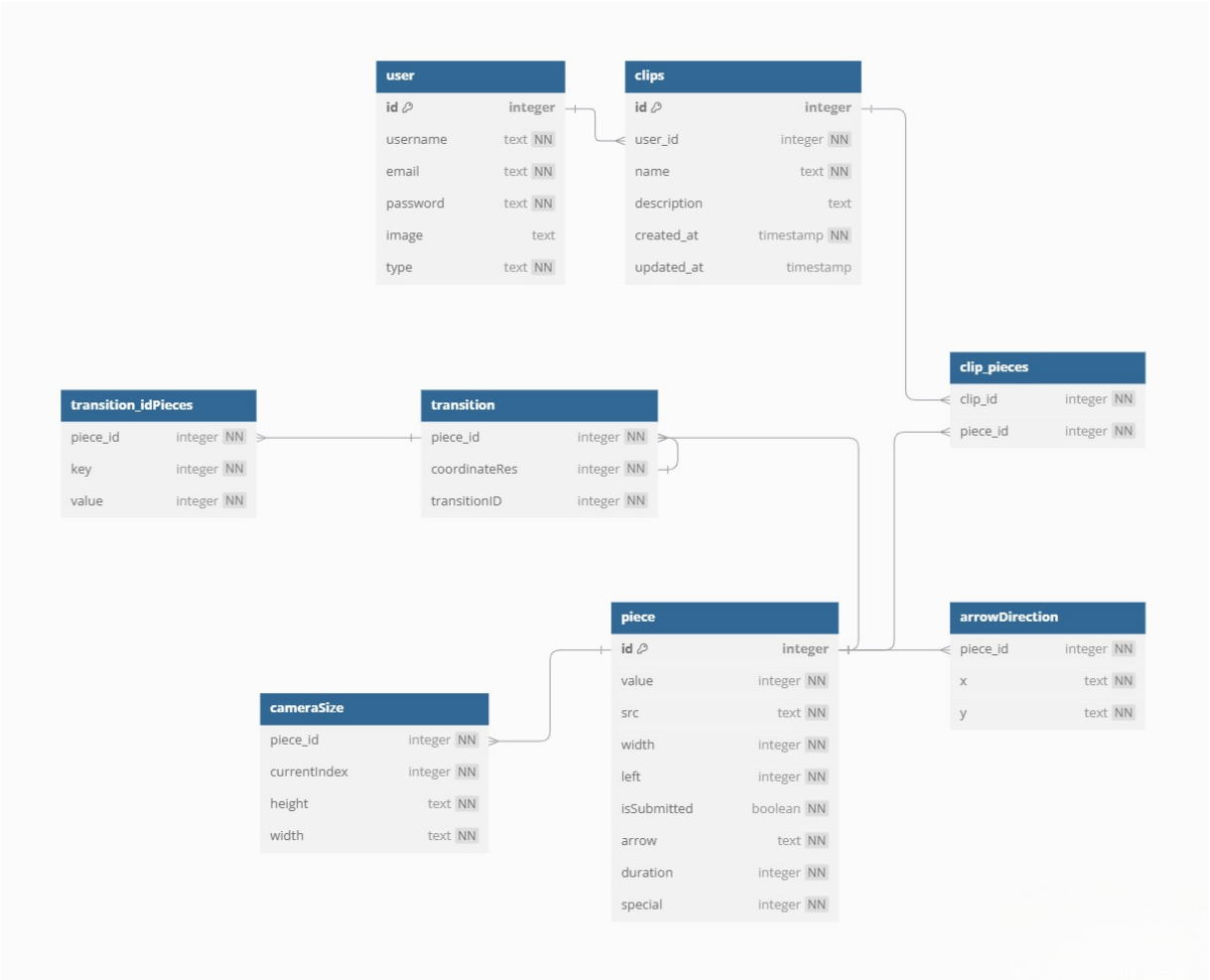
Na straně backendu lze rozlišit dva typy endpointů – auth a data. Endpointy začínající na adrese */auth* slouží k operacím souvisejícím s uživatelem, jako je autentifikace, obnovení hesla apod. Zatímco cesta */data* se zaměřuje na práci s jednotlivými videi, jako je jejich přidávání, aktualizace nebo mazání. Výjimkou je přihlášení přes službu Google, kde endpoint obsahuje přístupový token, který je generován při úspěšném ověření uživatele prostřednictvím *Google OAuth 2.0*.

Níže jsou uvedeny popisy vybraných typů endpointů:

- POST *auth/loginUser* se vztahuje k běžnému přihlášení do aplikace pomocí e-mailu a hesla.
- GET *oauth2/v1/userinfo?access_token=${accessToken}* slouží k ověření uživatele na základě přístupového tokenu.
- POST *auth/changePersonalData* změní na žádost uživatele vybraná data.
- POST *auth/resetPassword* odešle uživateli do e-mailové schránky kód pro resetování hesla (OTP).
- POST *data/addClip* přidá nově vytvořené video do tabulky *clips*.
- GET *data/getClips* načte všechna videa vytvořená uživatelem.
- POST *data/updateClip* obnoví informace o daném videu.

3.2.2 Interakce mezi backendem a databází

Jednou z hlavních problematik je navrhnout jednotlivé tabulky v databázi tak, aby byla komunikace mezi serverem a databází správně zajištěna. Nejprve si schéma databáze a následně si vysvětlíme, jak probíhají procesy ukládání, načítání, aktualizace a mazání dat.



Obrázek 3.6: Schéma databázových tabulek

Struktura databáze byla vytvořena tak, aby každá tabulka obsahovala parametr *id*, podle kterého je možné zajistit efektivní propojení mezi tabulkami pomocí unikátních klíčů. Na základě tohoto způsobu lze dále provádět dané operace formou *SQL* příkazů na straně backendu. Klíčovou roli v této struktuře (při úpravě daného videa) hraje tabulka *clip_pieces*, která funguje jako spojovací prvek mezi tabulkami *clips* (videoklipy) a *piece* (části videí), čímž umožňuje, že jedno video může obsahovat více částí.

3.3 Uživatelské rozhraní

Tato kapitola se ze začátku zaměřuje na zpracování dat a jejich dynamické vyobrazení na obrazovku. Dále jsou popsány hlavní komponenty aplikace, které tvoří základní strukturu a funkčnost. Nakonec je uveden návrh grafického designu a jeho implementace v aplikaci.

3.3.1 Dynamický obsah a získávání dat

Jedním z klíčových faktorů pro moderní webové aplikace je schopnost dynamicky načítat a zobrazovat data na základě interakce uživatele nebo změny stavu aplikace. Toho bylo ve frameworku *React* dosaženo pomocí již vysvětlených *React hooks* v kapitole 2.2.1. Nyní si detailněji přiblížíme jak dynamicky načítat (GET) a odesílat (POST) data pomocí knihovny *Axios*. Při zavolání jedné z těchto funkcí jsou data uchována ve formátu objektu, který *Axios* převede na JSON a následně ho odešle v těle HTTP požadavku na zadanou URL adresu. Tento požadavek je dále zpracován na straně serveru, a na základě vráceného stavu lze zjistit, zda celý proces proběhl úspěšně, nebo došlo k případné chybě.

Nádherným příkladem může být aplikace filtrů pro videa vytvořená uživateli, kde na základě aktuální volby uživatele dochází na backendu ke změně pořadí videí, které jsou následně vráceny a uloženy pomocí *State* hooku na frontendové straně. Pro zajištění aktivního zobrazení těchto videí využijeme další typ hooku zvaný *useEffect*, do kterého přidáme závislost na předchozím stavu uloženém v *useState*. To znamená, že při každé změně hodnoty v *useState* dojde k aktivaci *useEffect* a načtení nových dat.

Celý proces je zobrazen ve výpisu kódu 3.6, který představuje jeden z příkladů dynamického načítání dat z API na straně klienta. Podobným způsobem můžeme pomocí knihovny *Axios* daná data ze serveru přijímat pomocí funkce *get()*, která se liší některými parametry od funkce *post()*.

```
/** Načítání nejnovějších verzí klipů **/
useEffect(() => {

    // Pokud je aktivní sekce 'Moje projekty'
    if (activeItem === 'Moje projekty') {

        // Zpracování GET požadavku na serveru
        axios
            .get('http://localhost:4000/data/getClips', {

                // Data
                params : {
                    user_id: userData?.id, // ID uživatele
                    sortBy: sortCriteria,  // Filtr
                }
            })
            .then((res) => {

                // Uložení získaných videí (useState)
                setClips(res.data?.clips);

            })
            .catch((err) => {

                // Notifikace pro chybu
                toast.error(err.response?.data?.error);
            });
    }

}, [sortCriteria, activeItem]); // Hook se spustí při změně těchto hodnot
```

Výpis kódu 3.6: Aktivní získávání dat při změně výběru filtru videí.

3.3.2 Klíčové komponenty

V této podkapitole jsou popsány komponenty, které hrají v aplikaci důležitou roli a zabraňují principu neopakování se (DRY – Don't-Repeat-Yourself). Některé z nich byly vyvinuty od samého začátku, zatímco jiné knihovní komponenty byly poupraveny tak, aby lépe vyhovovaly specifickému prostředí aplikace.

3.3.2.1 Pop up

Element *Pop up* představuje grafický ovládací prvek, který zprostředkovává vyskakovací okno s určitým typem informací. Často se používá k zobrazení důležitých zpráv nebo akcí, které je třeba okamžitě řešit. [13] V aplikaci se objevuje v souvislosti s potvrzením změny údajů nebo výběrem z určitých možností, jako je například režim exportování videa. Jádro této komponenty tvoří kontejner knihovny *Reactjs-popup*, jež lze jednoduše naimportovat do kódu. Na základě této struktury lze vzhled vylepšit pomocí CSS. Pro oddělení obsahu obrazovky od kontejneru byl, v závislosti na typu sekce stránky, použit buď efekt stínování (box-shadow), nebo rozmazání (blur).

3.3.2.2 

Toast

Pro implementaci notifikačního systému byla do aplikace naimportována knihovna *React-Toastify*, známá také pod názvem *Toast*. Tato knihovna poskytuje snadné vytváření a použití různých typů oznámení, jako jsou informace, úspěch, varování, chyby nebo výchozí (default) zprávy. Podobně jako u knihovny *Reactjs-popup* je možné pro úpravu struktury obsahu notifikace přidat vlastní komponenty do samostatného kontejneru, což je umožněno díky kompatibilitě s CSS. Pro zobrazení konkrétního oznámení stačí zavolat metodu *toast(' '),* kde uvnitř uvozovek bude vepsána daná zpráva (viz výpis kódu 3.7).

```
<ToastContainer
    position="bottom-center"     // Pozice
    autoClose={5000}             // Automatické zavření
    hideProgressBar={false}      // Zobrazení průběhu trvání
    newestOnTop={false}          // Umístění nové notifikace
    closeOnClick={false}         // Zavření notifikace po kliknutí
    draggable                    // Přetahování
    theme="light"                // Vizuální styl
    transition={Bounce}          // Animace přechodu
/>
```

Výpis kódu 3.7: Ukázka importování kontejneru knihovny *React-Toastify*.

3.3.2.3 Navbar

Jednou z hlavních součástí je navigační lišta, známá také jako *Navbar*. Jejím primárním účelem je co nejvíce usnadnit uživateli orientaci a pohyb na webových stránkách. Strukturu této částice tvoří logo aplikace a uživatelské menu spolu s navigačními odkazy na jednotlivé sekce stránky. Odkazy v liště se mění podle stavu: zda je uživatel přihlášen (zobrazení ikony uživatele), nebo odhlášen (zobrazení tlačítek pro autentizaci).

Pro dosažení příjemnějšího uživatelského rozhraní byl kód pro komponentu napsán výhradně pomocí knihovny *Styled-components*, jejíž výhody byly již zmíněny v kapitole 2.2.1. Tato komponenta je zároveň plně responzivní, což znamená, že se styly v liště dynamicky přizpůsobují velikosti obrazovky různých zařízení. Příkladem může být mobilní zařízení, kde se místo vypsání všech prvků ze seznamu zobrazí možnost sbalení (hamburger menu), po jejímž kliknutí se prvky seřadí pod sebe.

3.3.2.3 React Icons

Přestože knihovna *React Icons* přímo nesouvisí s principem DRY, byla vybrána kvůli rozsáhlé nabídce populárních ikon pro framework *React*. Všechny ikony jsou ve formátu komponent, což přináší výhodu v tom, že není potřeba spravovat velké množství souborů v oddělené složce. Komponenty ikon lze navíc dále upravovat pomocí kaskádových stylů prostřednictvím parametru *style={}*. Přehled všech dostupných ikon lze najít v oficiální dokumentaci na webové stránce *https://react-icons.github.io/react-icons/*, odkud byly vybrané ikony čerpány z různých nabízených sad, jako například *Heroicons*. Dokumentace zároveň umožňuje snadné vyhledávání ikon podle názvu a poskytuje ukázky jejich použití.

3.3.3 Grafický design

Tato podkapitola se zaměřuje na hlavní prvky designu, které v aplikaci zajišťují uživatelsky přívětivé prostředí. Vysvětlíme si zde použité atributy kaskádových stylů a základní význam barev v kontextu aplikace.

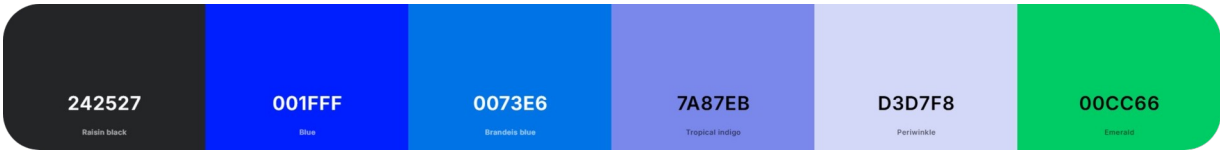
3.3.3.1 Animace

Pro zdokonalení vzhledu aplikace byly pro komponenty uplatněny různé efekty pomocí *CSS* pseudotříd (nejznámější – *:hover*, *:focus*), jejichž účelem je definovat speciální stav daného prvku. S pomocí těchto pseudotříd lze dále pro plynulost animace aplikovat atribut *transition*, který umožňuje definovat délku, typ a zpoždění vizuálního efektu. Dalšími významnými vlastnostmi pro vylepšení vzhledu komponenty jsou *border* (definování okraje), *border-radius* (zaoblení rohů) a *box-shadow* (přidání stínu). Tyto vlastnosti byly využity především u tlačítek, kde pomáhají zvýraznit interaktivní prvky a zlepšit uživatelský dojem.

Obrázek 3.8: Srovnání tlačítek podle zvolené pseudotřídy – vlevo původní stav, vpravo efekt při najetí myší (*:hover*).

3.3.3.2 Barevné schéma

Webová aplikace je z velké části rozložena na bílém pozadí tak, aby text byl jasně čitelný a vynikal kontrast barev mezi komponenty a danou obrazovkou. Autor zvolil zejména různé odstíny modré barvy, jelikož dle jeho názoru barva příliš nevyniká a celkově vytváří pocit lehkosti. V psychologii barev je modrá barva často spojována s pocity důvěry, bezpečí a klidu. [14] Kromě modré barvy byla použita také zelená, která symbolizuje aktivní výběr uživatele v sekci produkce videa, a tmavý odstín šedé, jenž byl nejčastěji využíván pro text. Na obrázku 3.9 je zobrazená celá paleta nejčastěji používaných barev v prostředí aplikace.



Obrázek 3.9: Paleta barev.

Závěr

Výsledná aplikace byla úspěšně dokončena a hra funguje bez problémů. Vytváření celého programu se obešlo bez velkých obtíží. Při průběhu projektu bylo ve vývojovém prostředí uplatněno spousta způsobů, podle kterých byla hra vylepšena. Tyto úpravy zlepšily nejen vzhled, ale také funkčnost samotné hry. Během tvorby tohoto projektu si autor výrazně zlepšil své dovednosti a získal tak spoustu nových poznatků ve frameworku *JavaFX*. Projekt by šlo rozšířit o několik dalších funkcí ve hře, mezi které patří například vytvoření útvaru částice puzzle.

Splnění zadání

Výsledná aplikace splňuje zadání, které bylo uvedeno v úvodu. Při projektu bylo nejtěžší zkontrolovat, zda vygenerované částice puzzle jsou správně umístěny na jednotlivých pozicích v hrací ploše. Během řešení této úlohy autor narazil na několik dalších problémů, které byly následně opraveny a výsledná aplikace se tak vyvinula nad očekávání autora.

Bibliografie

[1] *HTTP* https://www.seoprakticky.cz/slovnik-pojmu/http/ [cit. 2025-01-19].

[2] *Built-in React Hooks* https://react.dev/reference/react/hooks/ [cit. 2025-01-19].

[3] *About Node.js®* https://nodejs.org/en/about [cit. 2025-01-19].

[4] Mozilla Corporation. *Express/Node introduction* https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction [cit. 2025-01-19].

[5] *Appropriate Uses For SQLite* https://www.sqlite.org/whentouse.html [cit. 2025-01-19].

[6] Huei-Hsin Wang. *Homepage Design: 5 Fundamental Principles* https://www.nngroup.-com/articles/homepage-design-principles/ [cit. 2025-01-21].

[7] Kritika Sharma. *SQL injection and cross-site scripting: The differences and attack anatomy* https://www.manageengine.com/log-management/cyber-security/sql-injection-web-application-attack-and-cross-site-scripting-the-differences-and-attack-anatomy.html [cit. 2025-01-24].

[8] Mozilla Corporation. *Drag Events* https://developer.mozilla.org/en-US/docs/Web/API/DragEvent [cit. 2025-01-24].

[9] Mozilla Corporation. *CanvasRenderingContext2D* https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D [cit. 2025-01-26].

[10] Mozilla Corporation. *CanvasRenderingContext2D: drawImage() method* https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage [cit. 2025-01-26].

[11] Suprabha. *RequestAnimationFrame in JavaScript* https://builtin.com/software-engineering-perspectives/requestanimationframe [cit. 2025-01-28].

[12] Mozilla Corporation. *FileReader: readAsDataURL() method* https://developer.mozilla.org/en-US/docs/Web/API/FileReader/readAsDataURL [cit. 2025-01-29].

[13] Ahmet Loca. *What's pop-up?* https://medium.com/@locaahmet/whats-pop-up-1b3554f02295 [cit. 2025-01-30].

[14] Software Development Academy. *UX/UI Design a jeho vliv na rozhodovací procesy uživatelů* https://sdacademy.cz/blog/ux-ui-design/

Seznam obrázků