Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Aplikace pro převod fotografie na kreslený obrázek

2023

Jakub Jurka

Vedoucí práce:
Mgr. Markéta Trnečková, Ph.D.

Studijní program: Informatika,
Specializace: Programování a vývoj software

Bibliografické údaje

Autor:	Jakub Jurka
Název práce:	Aplikace pro převod fotografie na kreslený obrázek
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Informatika, Specializace: Programování a vývoj software
Vedoucí práce:	Mgr. Markéta Trnečková, Ph.D.
Počet stran:	36
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Jakub Jurka
Title:	An application for transferring photography to painting
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Computer Science, Specialization: Programming and Software Development
Supervisor:	Mgr. Markéta Trnečková, Ph.D.
Page count:	36
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Vytváření kreslených obrázků z fotografií je v poslední době čím dál tím víc populární. Tato práce se zabývá jednou z jednoduššíchčástí tohoto tématu. Ukazuje, jak se vytváří jednoduché filtry, které s každým obrázkem pracují stejně. Mimo jiné se také zabývá tvorbou webové aplikace a API serveru, který komunikuje mezi aplikací a filtry.

Synopsis

The creation of painted pictures from photography is getting more and more popular these days. This thesis focuses on one of the simpler parts of this topic. It shows how simple filters which work with every single image in the same way are made. Additionaly, it shows development of a web application and an API server which enables the communication between the application and the filters.

Klíčová slova: převod fotografie na kresbu; webová aplikace; API server

Keywords: transferring photography to painting; web application; API server

Rád bych poděkoval své vedoucí práce za rady při vytváření aplikace a psaní textu. Dále děkuji své rodině a přátelům za testování aplikace.

Odevzdáním tohoto textu jeho autor místopřísežně prohlašuje, že celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

1 Úvod

Slavné osobnosti kreslí velké množství jejich fanoušků a v minulosti byla kresba jediný způsob, jak osobu vizuálně zachytit. Kdyby ale někdo z nás chtěl svůj kreslený portrét, musel by nejspíše někomu zaplatit. Proces kresby navíc trvá dlouho. Když člověk přesně neví, jaký styl by si přál, nemusí být s výsledkem spokojen. V tuto chvíli ale mohou přijít ke slovu stroje, které takovou kresbu mohou vytvořit velice rychle a díky tomu dovolují jednoduché změny ve výsledku. Nevýhodami ale může být horší kvalita nebo chybějící lidský element, který dodává například jemné nepřesnosti.

Téma kresby je mi blízké, protože sám ve volném čase občas kreslím. Mezi mé zájmy patří také počítačová úprava obrazu. Zajímalo mě tedy, jak tento proces funguje z pohledu programátora. Navíc mě baví navrhovat a designovat aplikace, takže mi tento způsob prezentace přišel ideální.

1.1 Cíle

Rozhodl jsem se vytvořit aplikaci, která dovolí uživateli jednoduše převést libovolnou fotografii na kreslený obrázek. Vydal jsem se cestou, kde aplikace bude nabízet více jednodušších filtrů, místo pár složitých. Tyto složité filtry by ale mělo jít jednoduše přidat. Takto bude moct aplikace fungovat jako základní kámen pro větší projekt. Chtěl jsem, aby aplikace dosáhla určitých vlastností, zejména:

- jednoduchost – ovládání by mělo být intuitivní, uživatel by neměl mít problém se v aplikaci zorientovat; bylo by dobré, kdyby se uživatel nemusel přihlašovat, to je ale na úkor možnosti dlouhodobého ukládání na serveru
- minimalismus – vzhled aplikace by měl být minimalistický bez velkého množství zbytečných elementů
- přístupnost – k aplikaci by měl mít uživatel jednoduchý přístup kdekoliv; nabízí se proto webová aplikace, díky které může uživatel pracovat jak na počítači, tak i na mobilním telefonu
- nastavitelnost – uživatel by měl mít možnost nastavit efekty filtrů pro ideální výsledky
- rychlost – filtry by měly být rychlé, aby dovolovaly změnu parametrů bez zbytečného čekání
- vrstvení filtrů – v aplikaci by mělo být možné kombinovat filtry pro získání unikátních výsledků
- rozšiřitelnost – mělo by být jednoduché pro správce přidat další filtr
- samostatnost filtrů – knihovna s filtry by měla fungovat i bez aplikace, každý kdo si ji stáhne by měl mít možnost je použít zvlášť

2 Aplikace s podobnou funkcionalitou

Jelikož je mnoho stylů kreseb, každá aplikace bude implementovat odlišné typy filtrů. Některé z nich se specializují na jeden určitý typ a jiné zase na velkou škálu filtrů. U lepších aplikací jsou dokonce některé filtry uzavřené za platební branou a požadují registraci pro uložení vytvořeného obrázku.

2.1 BeFunky

Když jsem hledal aplikace, které pracují podobně, nejvíce mě zaujala webová aplikace BeFunky [1]. Nabízí velké množství uměleckých filtrů, které dosahují kvalitních výsledků.

Hlavním problémem je dle mého názoru potřeba předplatného pro použití těchto filtrů. Navíc většina z nich nabízí minimální možnost úpravy nastavení. To může v kombinaci s některými obrázky vytvářet nežádoucí výsledky. Při změně jakéhokoliv nastavení filtru je obrázek ihned překreslen. To vede hlavně u složitějších filtrů, které se načítají déle, ke zbytečnému a dlouhému překreslování, když měníme více hodnot najednou.

Dále aplikace dovoluje přímo manipulovat s plochou obrázku myší, což přidává možnost mít filtry, které by bez tohoto nebyly možné, jako například výběr barvy nebo plochy.

Verze zdarma nabízí pouze některé základní operace s obrázky, jako například ořezání a rozmazání. Ty jsou navíc na obrázek přidány přímo bez možnosti další úpravy, protože aplikace v neplacené verzi neobsahuje vrstvy.

Aplikace dokonce nabízí mobilní verzi, která je vylepšena, když si ji uživatel přímo stáhne jako mobilní aplikaci.

Se vším všudy se jedná hlavně o foto editor. Ten navíc nabízí převod na kresbu, který nejspíš není hlavním záměrem této aplikace. I tak ho ale editor zvládá v některých ohledech o dost lépe než konkurence.

2.2 Fotor

Další webová aplikace Fotor [2] se naopak hlavně specializuje na kreslené filtry. Podobně jako u předešlé aplikace je potřeba předplatného pro plné využívání. V tomto případě jsou ale omezení menší, protože filtry lze použít i bez placení. Obrázek můžeme dokonce stáhnout, jen je v horší kvalitě a obsahuje vodoznak.

Z hlediska možnosti nastavení filtrů aplikace velice pokulhává, protože nabízí pouze viditelnost filtru přes výchozí obrázek. Podobně na tom je i s vrstvením filtrů, které aplikace nenabízí.

Velmi mě zaujalo načítání při aplikování filtru, které občas trvá déle, ale v mezičase nabízí detailnější ukázku filtru a náhodné citáty známých umělců.

Aplikace nabízí i mobilní verzi na webu, která mi přijde velice příjemná na používání, dokonce i přehlednější než verze desktopová.

2.3 Prisma

Mezi nejpopulárnější čistě mobilní aplikace patří Prisma [3]. Stejně jako ostatní aplikace nabízí předplatné, které odemkne všechny dostupné filtry. Podobně jako u Fotor je vyšší kvalita obrázků také uzavřena za předplatným. I přesto nabízí aplikace mnoho filtrů zdarma. I když nemáme výsledné obrázky v plné kvalitě, vypadají lépe, než výsledky z předešlé aplikace bez placení. To je ale nejspíše výsledkem práce na mobilním telefonu, kde bude obrázek se stejným rozlišením menší, a tak bude i vypadat lépe.

Možnosti nastavení filtrů jsou skoro neexistující, podobně jako u Fotor. Aplikace ale naštěstí obsahuje možnost úpravy základních hodnot obrázku, jako je například jas a kontrast.

Hodně mě překvapilo velice přesné automatické rozdělování obrázku na popředí a pozadí, které nám dovoluje aplikovat daný filtr jen na některou část obrázku nebo změnit pozadí.

2.4 Porovnání

Hlavním rozdílem mezi většinou dostupných aplikací a mým řešením je nutnost platby pro plné používání aplikace. Hodně z nich nabízí použití pouze jednoho filtru v jednu chvíli. To mi přijde omezující, protože při používání mé aplikace jsem došel k tomu, že některé kombinace vypadají zajímavě.

Nastavování filtrů je také funkcionalita, kterou často tyto aplikace nemají. Při kombinaci některých filtrů a obrázků toto může chybět a výsledek nakonec nebude vypadat vůbec dobře.

Ve většině aplikací trvá použití filtru celkem dlouho, to ale většinou omlouvá komplexnost daného filtru. Protože jsou mé filtry celkem jednoduché, jejich aplikace je rychlejší.

Mé aplikaci navíc chybí manipulace s obrázkem přímo myší, jako například nabízí BeFunky. Tato absence nedovoluje mé aplikaci být přímo editorem pro fotografie. To ale ani nebyl záměr práce. I tak by se tato funkcionalita hodila u některých filtrů, například pro výběr barvy z obrázku.

Má aplikace naopak nabízí některé zajímavé efekty, které nejsou přímo uměleckou kresbou, ale buď samy nebo v kombinaci s ostatními vytvářejí zajímavé výsledky.

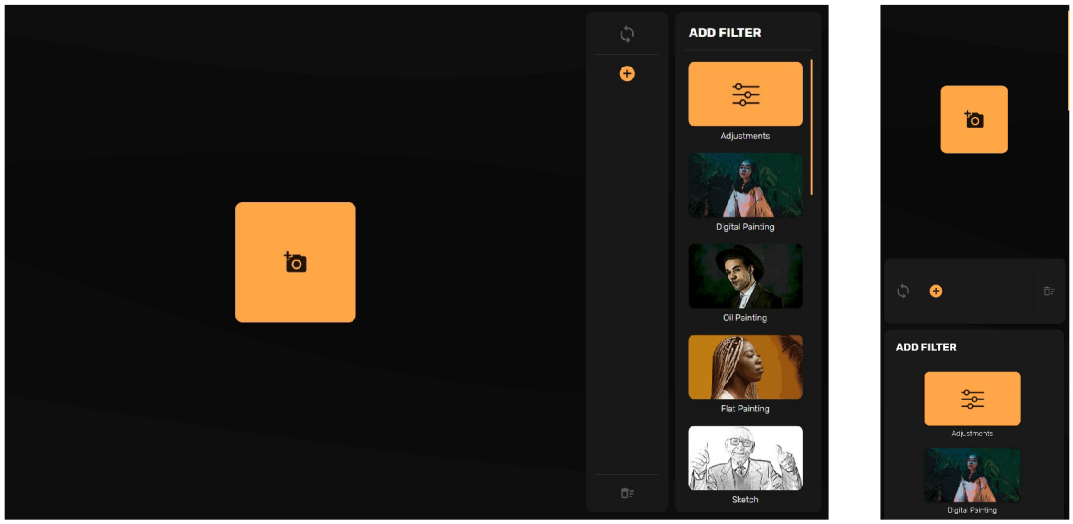
3 Uživatelská dokumentace

Tato část textu slouží jako uživatelská příručka pro práci s aplikací. Aplikace je stavěna velice minimalisticky a se snahou být jednoduše srozumitelná. Uživatel, který už dříve pracoval s jinou aplikací na úpravu obrázků, by neměl mít problém rychle porozumět ovládání. Naopak uživatel bez zkušeností může mít z počátku lehké potíže i přes to, že je aplikace navržena tak, aby byla intuitivní. Celá tato část je hlavně cílena pro druhou skupinu uživatelů.

3.1 Rozložení

Celá aplikace pracuje v jediné obrazovce, která je rozdělena do tří oblastí: Náhled obrázku, Manažer filtrů a Panel s akcemi. Jednotlivé oblasti jsou podrobněji rozebrány v následujících kapitolách.

V desktopové verzi jsou oblasti seřazeny horizontálně vedle sebe a zabírají celou výšku obrazovky. Jejich pořadí je stejné jako ve kterém byly vyjmenovány výše. Bylo zvolené, jelikož simuluje pořadí, kterým se aplikace využívá – výběr obrázku, zvolení akce, práce se zvolenou akcí. Mobilní verze je velice podobná, jen se oblasti řadí vertikálně a orientace Manažer filtrů je horizontální.



Obrázek 1: Desktopové a mobilní rozložení aplikace

3.2 Náhled obrázku

Největší oblastí je prostor pro zobrazení obrázku, který je momentálně upravován. Pro nahrání obrázku do aplikace slouží velké tlačítko ve středu této oblasti. Po vybrání obrázku se změní tlačítko na ukazatel načítání a následně se na tomto místě zobrazí zvolený obrázek. Ukazatel načítání se zobrazí i v případě, kdy aplikace čeká na překreslení obrázku. Pokud nechce uživatel čekat, může kliknout na tento ukazatel a tím zrušit načítání a odstranit obrázek.

Po kliknutí na zobrazený obrázek se objeví menu možností s třemi tlačítky uprostřed. Tlačítka, postupně zleva, slouží pro stažení obrázku z aplikace, odstranění obrázku a zrušení menu.

Když nastane chyba na serveru, zobrazí se na místě obrázku ukazatel s vykřičníkem. Tento stav může být zrušen kliknutím na tento ukazatel. To odstraní obrázek a dovolí uživateli ho znovu nahrát.

3.3 Manažer filtrů

Tato oblast slouží pro zobrazení aktuálně využívaných filtrů a výběru toho, co zobrazuje Panel s akcemi. V horní části (v mobilní verzi nejvíce vlevo) se nachází tlačítko pro překreslení obrázku s aktuálními filtry. Pokud byla provedena jakákoliv změna filtrů bez překreslení, je toto tlačítko označeno. Pod tímto tlačítkem (napravo v mobilní verzi) se nachází seznam aktuálně používaných filtrů, který je řazen shora dolů (zleva doprava v mobilní verzi). Filtry jsou seřazeny v pořadí, v jakém budou aplikovány. Na konci seznamu se vždy nachází tlačítko pro přidání dalšího filtru a zobrazení Přidání filtru. Ve spodní části (v mobilní verzi nejvíce vpravo) se nachází tlačítko pro odstranění všech aktuálně používaných filtrů.

Pokud chce uživatel změnit nastavení jednoho z filtrů, může ho vybrat kliknutím. Toto změní obsah Panel s akcemi na možnost Nastavení filtru pro zvolený filtr.

3.4 Panel s akcemi

Poslední oblast je nejvíce interaktivní, jelikož zobrazuje všechna nastavení, se kterými může uživatel pracovat, a mění v průběhu používání aplikace svůj obsah. Momentálně zobrazený obsah je závislý na zvoleném tlačítku v Manažer filtrů. Všechny možnosti, které může tento panel zobrazovat jsou popsány v následujích podsekcích.

3.4.1 Přidání filtru

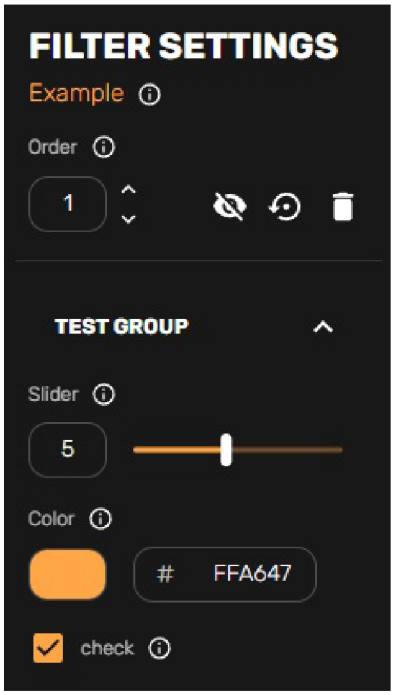
Při zvolení tlačítka pro přidání filtru tato oblast zobrazuje seznam všech dostupných filtrů. Po kliknutí na jeden z nich se přidá do seznamu v Manažer filtrů, vybere se jako aktivní a tato oblast se změní na Nastavení filtru.

3.4.2 Nastavení filtru

Tato oblast zobrazuje všechna možná nastavení daného filtru. Mezi možnosti nastavení hodnot patří posuvník (slider), přepínač (switch) a výběr barvy. Každá hodnota má k dispozici i nápovědu, která se zobrazí po najetí na ikonku se znakem „i“. Nápověda blíže popíše, co daná hodnota změní. Změny v nastavení se projeví až po překleslení obrázku tlačítkem v Náhled obrázku.

V horní části této oblasti se nachází tři tlačítka. První změní viditelnost filtru. Po kliknutí na přeškrknutou ikonku je filtr označen jako schovaný a při vykreslování nebude aplikován. Tento stav je zároveň označen v Manažer filtrů, kde se zobrazí místo pořadí daného filtru. Opětovným kliknutím na tlačítko, které v tuto chvíli ukazuje nepřeškrknutou ikonku, je filtr přidán do vykreslování. Další tlačítko vrátí nastavení daného filtru do výchozího nastavení. Poslední tlačítko filtr smaže.

Dále nabízí uživateli schovat jednotlivé skupiny vstupů. Schování skupiny neovlivní její vykreslování, slouží pouze pro větší přehlednost u filtrů s více vstupy.



Obrázek 2: Nastavení filtru

4 Použité technologie

V této části jsou stručně shrnuty použité jednotlivé technologie a knihovny, které aplikace využívá. Hlavně se zabývá tím, proč byla daná technologie zvolena a k čemu je využívána.

4.1 Python

Back-end je postaven na jazyku Python [4]. Pro úpravu obrázků je Python jeden z nejpopulárnějších jazyků hlavně díky velkému množství knihoven, které s obrázky dokážou pracovat. Toto platí i pro vývoj API serveru. Pro Python je vytvořeno mnoho frameworků přesně pro tyto účely.

Díky velké podpoře tohoto jazyka v požadovaných oblastech jsem mohl vytvořit ucelenější back-end, a tím usnadnit komunikaci mezi oběma částmi. Dalším důvodem zvolení tohoto jazyka byla osobní preference a větší zkušenosti.

4.2 Pillow

Jedna z nejpoužívanějších knihoven pro Python na úpravu obrázků je knihovna Pillow [5]. Jedná se o fork knihovny PIL (Python Imaging Library). Nabízí jak mnoho základních, tak i pokročilejších operací s obrázky.

Na této knihovně jsem založil většinu své knihovny na úpravu obrázků. Využil jsem hlavně základní operace (např. změna velikosti a úprava jasu), ale v některých případech jsem použil i operace pokročilejší z důvodu jejich vysoké optimalizace (např. rozmazání a maskování).

4.3 NumPy

Knihovna NumPy [6] patří k nejzákladnějším knihovnám jazyka Python. Využívá se hlavně k vědeckým výpočtům nebo k práci s maticemi a vektory. Na rozdíl od struktur jazyka Python je práce se strukturami NumPy rychlejší. To se hodí při práci s větším počtem dat.

Jelikož se bitmapový obrázek reprezentuje maticí jeho pixelů, mohl jsem využít efektivity této knihovny pro urychlení některých operací s obrázky. Toto mi navíc ušetřilo čas, protože jsem nemusel definovat vlastní funkce nad maticemi. Knihovna Pillow navíc dovoluje přímý převod mezi svou reprezentací obrázku a polem z NumPy, což také ulehčilo mou práci.

4.4 FastAPI

Jak už bylo dříve řečeno, API server zajišťující komunikaci s aplikací je vytvořen v jazyce Python. Pro svůj server jsem zvolil framework FastAPI [7] hlavně z důvodu jeho rychlosti, jednoduchosti a narůstající popularitě v poslední době. Jelikož je využití serveru velice povrchové (pouze posílá a přijímá obrázky nebo

data ve formátu JSON), nebylo by ho těžké přepracovat do jiného frameworku, ideálně jazyka Python.

4.5 React

Celá front-end část je vytvořena jako aplikace knihovny React [8], která dovoluje psát složitější webové aplikace v jazyce JavaScript, který je jedním ze základních jazyků pro web. Oproti vývoji v samostatném JavaScriptu umožňuje knihovna React jednodušší práci. React rozděluje aplikaci do jednotlivých komponent, díky kterým je znovupoužití elementů v aplikaci jednoduché. Dále nabízí hodně nástrojů pro snadné sledování změn v aplikaci a reakce na ně.

Knihovnu jsem vybral, zejména protože je jednoduchá a vhodná pro můj návrh aplikace, který vyžadoval časté opakování stejných elementů a reakce na interakce s nimi. Navíc jsem podobně jako u jazyka Python měl s touto knihovnou už dřívější zkušenosti.

4.6 Material UI

Pro jednodušší vývoj a hezčí výsledky webových aplikací se často využívá různých CSS frameworků. Při využívání knihovny React se ale nabízí rovnou využít knihovnu, která poskytuje komponenty s upravenými vzhledy. Mezi tyto knihovny například patří Material UI [9], která nabízí mnoho komponent dodržující zásady Material Design [10] od společnosti Google.

Material Design mi přijde jako jeden z nejlepších designových jazyků pro vytváření UI hlavně díky jeho jednoduchosti a minimalistickému vzhledu. Knihovna navíc nabízí stylování předvytvořených komponent, takže umožňuje jednoduché vytvoření komponent specifických pro mou aplikaci.

4.7 Další knihovny

Aplikace využívá i další knihovny, které není potřeba více představovat, jelikož jejich obsah buď není moc rozsáhlý, nebo byly využity jen minimálně.

- React Color [11] – Knihovna pro React, která nabízí různé jednoduché UI nástroje pro výběr barev.

- scikit-learn [12] – Rozsáhlá knihovna pro strojové učení pro Python. Využil jsem pouze funkce KMeans pro získání shluku barev pro paletu.

5 Programátorská dokumentace aplikace

Celá aplikace je postavena na technologii React, takže rozděluje veškeré své elementy do komponent, na kterých jsou React aplikace postaveny. Jak bylo řečeno výše v Uživatelská dokumentace, aplikace je rozdělena do tří oblastí (prakticky jsou to čtyři komponenty, jelikož poslední oblast mění svůj obsah mezi dvěma funkcemi). Každá tato oblast je reprezentována jednou komponentou a navzájem komunikují pomocí nadřízené komponenty `ImageEditApp`, která řídí běh celé aplikace.

Jelikož jsou React komponenty funkce, využívají návratovou hodnotu na určení toho, co mají zobrazit. Zobrazení komponent se realizuje pomocí JFX, které dovoluje přímo psát HTML elementy do JavaScript kódu.

Jelikož využívám knihovny Material UI, tak namísto klasických HTML elementů používám přímo komponenty z této knihovny. Tyto komponenty dovolují využívat buď přednastavených stylů, které jsou nastavitelné pomocí argumentů nebo vlastního stylování pomocí přidání CSS kódu jako argument.

5.1 Struktura projektu

Hlavní kód aplikace je uložen ve složce `src`, zbývající soubory byly automaticky vytvořeny při vytváření React projektu. Složka `src` obsahuje tři podsložky `components`, `img` a `setup-files`.

Ve složce `components` se nachází kódy jednotlivých komponent rozdělené podle komponent, které je spravují. `StyledComponents` obsahuje upravené verze komponent z Material UI pro potřeby aplikace. Tyto komponenty jsou využity ve více částech aplikace, takže nespadají přímo pod žádnou komponentu.

Všechny obrázky, které se v aplikaci zobrazují,[1] jsou uloženy ve složce `img`. V této složce se musí nacházet všechny obrázky k filtrům, jinak nebude přidaný filtr fungovat a vyhodí chybu aplikace. Navíc je nutné, aby byly obrázky uložené se stejným jménem, jaké mají přiřazené v `filters.js`, a ve formátu .jpg.

Poslední složka `setup-files` obsahuje soubory, které pomáhají konzistenci hodnot v aplikaci, a další data, která se v aplikaci zobrazují. Soubor `filters.js` udává, které filtry se zobrazí, jaké mají k dispozici nastavení a základní hodnoty těchto nastavení. Nové filtry se přidávají pomocí tohoto souboru.[2] Dalším souborem je `fontSizes.js`, který dovoluje upravovat hodnoty tří velikostí písma. Toto nastavení funguje pouze v případě, že elementu byla zvolena velikost písma z tohoto souboru. Posledním souborem je `themes.js`, který obsahuje nastavení témat aplikace pomocí `ThemeProvider` v Material UI. Mezi tyto hodnoty patří například barvy, zakulacení rohů nebo styl posuvníku (scrollbar). To dovoluje do budoucna jednoduše přidat další témata.

[1] S výjimkou těch nahraných uživatelem.

[2] Více v Možnosti rozšíření.

5.2 ImageEditApp

Jak už bylo na začátku kapitoly řečeno, hlavní logiku aplikace spravuje komponenta `ImageEditApp`. Kromě zmíněné komunikace mezi ostatními komponentami řeší i komunikaci s back-end serverem nebo celkové rozložení aplikace.

5.2.1 Komunikace mezi komponentami

Komunikace mezi ostatními komponentami je řízena pomocí useState hooks, které jsou uloženy právě v této komponentě. Tyto hooks jsou předávány pomocí Context hook podřazeným komponentám. Ve své podstatě useState hooks fungují jako proměnné, jen s tím rozdílem, že po dosazení jiné hodnoty je tato změna zobrazena na všech místech, kde je tato proměnná použita. To dovoluje snadnou komunikaci mezi komponentami a reakci na změny.

Všechny useState hooks jsou inicializovány na začátku kódu komponenty. Mezi těmito hodnotami se nachází například ukazatel, jestli je obrázek nahraný, cesta ke staženému obrázku ze serveru, určení, co má zobrazovat, Panel s akcemi nebo jaký je aktuálně vybraný filtr.

Pod tímto úsekem se nachází rozšířené verze těchto hooks, které při nastavení hodnoty provedou i další akce. Příkladem může být zrušení vybraného filtru, když se mění obsah Panel s akcemi pomocí `setActivePanel`.

5.2.2 Komunikace se serverem

Dále tato komponenta zařizuje komunikaci s API serverem. Jelikož je komunikace omezena jen na odesílání a přijímání obrázku, odesílání konfigurace filtrů a ukončování spojení, nepřišlo mi důležité oddělit tuto komunikaci do samostatného kódu. Navíc funkce na komunikaci se serverem obsahují pouze minimální úpravy dat a samotné volání metody `fetch()`, která slouží na odesílání a přijímání požadavků.

Zvláštností je funkce `sessionTimedOut`, která se při ukončení komunikace ze strany serveru pokusí toto spojení obnovit tím, že znovu zašle obrázek. Protože aplikace funguje bez přihlášení a server ukládá pouze aktuálně ukládaný obrázek,[3] je nutné tyto data ze serveru mazat po ukončení komunikace.

Toto se dá vyřešit pomocí zaslání požadavku po vypnutí stránky, který data na serveru smaže. V tomto přístupu jsem ale nalezl problém, kdy prohlížeč na mobilním telefonu nezašle tento požadavek, pokud je uzavřen bez přímého zavření stránky. Takto se při dalším zapnutí stránky vytvoří nové spojení, ale neukončí se předešlé. Proto server po určité době od poslední komunikace data uživatele automaticky smaže.

Zde ale může dojít k situaci, kdy má uživatel nahraný obrázek, nekomunikuje déle než je interval serveru na pročištění dat, a poté chce obrázek aktualizovat. V tomto případě by došlo k chybě. Proto, když klient žádá o překreslení obrázku,

[3] Obrázek by nemusel server ukládat vůbec, ale tento přístup urychluje komunikaci. Takto klient nemusí pokaždé posílat ten stejný obrázek.

který už nemá server k dispozici, odpovídá server statusem 410 (gone). Klient na to reaguje opětovným zasláním obrázku a uživatel ani nepozná, že jeho spojení bylo přerušeno.

Pokud nastane chyba při vytváření obrázku, odpovídá server statusem 502. V tomto případě se objeví chybový stav na místě obrázku.

5.2.3 Rozložení aplikace

Tato komponenta zobrazuje pouze rozložení aplikace a udává, kde se má která podřazená komponenta zobrazit. Nejvýše postavený element `<ThemeProvider>` dovoluje určit, jaké bude téma celé aplikace. Další element `<ImageEditContext.Provider>` předává Context podřazeným komponentám. Rozložení aplikace je realizováno pomocí `display: grid`.

5.3 ImagePreview

Komponenta pro Náhled obrázku je rozdělena do komponent podle jednotlivých stavů, ve kterých může tato komponenta nacházet. Výběr toho, která komponenta se zobrazuje, je realizován pomocí useState hooks z Context předaného z ImageEditApp. Tyto podřazené komponenty jsou `ImageAdd` (možnost přidání obrázku), `ImageLoading` (načítání obrázku), `ImageError` (chyba ze strany serveru) a `ImageActive` (zobrazený obrázek). První tři se zobrazují jako upravené tlačítko se společným základem v pomocné komponentě `ImageBox`, liší se pouze v zobrazované ikonce a akci po kliknutí. Pokud je obrázek vybraný a načtený, zabírá všechen volný prostor tak, aby nebyl zdeformovaný.

Po najetí na zobrazený obrázek se změní některé jeho vlastnosti jako například velikost a průhlednost. Když na něj uživatel klikne, objeví se menu s možnostmi obrázku.[4] V tuto chvíli už prohlížeč neregistruje, že je na obrázek najeto myší, takže se zruší efekty. To ale není žádoucí efekt. Problém jsem vyřešil pomocným useState hook `grayed`. Ten je nastaven jako pravdivý, když se otevře menu, a jako nepravdivý, když se zruší menu. Takto se ale efekt začal po zrušení menu zobrazovat znovu, jako by uživatel poprvé najel na obrázek. Proto jsem přidal prodlevu u nastavení pomocného hooku na nepravdu a efekt vypadá plynule.

5.4 FilterManager

Výběr aktivního filtru se odehrává v Manažer filtrů, který je realizován touto komponentou. Komponenta samotná nemá moc logiky. Její hlavní účel je určit vzhled této oblasti a vytvořit kartičku `SelectFilter` pro každý přidaný filtr v aktuálním pořadí.

Každá kartička má alternativní vzhled v závislosti na tom, jestli je momentálně zvolena. Neaktivní kartička je zvolena kliknutím. Jelikož má každý přidaný

[4] Realizované komponentou `ImagePopover`.

filtr unikátní id, může být porovnán s id aktivního filtru. Takto se určí, jestli se má zobrazit jako zvolený.

Tlačítka pro překreslení obrázku `UpdateImage`, přidání filtru `AddFilter` a smazání všech filtrů `DeleteAllFilters` jsou realizována jako vlastní třídy, které přidávají tlačítkům specifické akce. Původně jsem pro všechny tři používal jednu komponentu s argumenty, ale to se ukázalo jako neefektivní kvůli různorodosti jednotlivých tlačítek.

5.5 FilterChoice

Když není zvolen žádný filtr, zobrazuje Panel s akcemi ve svém obsahu Přidání filtru. Tato komponenta zobrazuje tento stav. Komponenta je koncipována podobně jako FilterManager. Zobrazované kartičky jsou třídy `FilterAdd`, které kromě definice svého vzhledu řeší jen reakci na kliknutí. Po kliknutí na jednu z nich se přidá nový filtr pomocí funkce z `ImageEditApp` a vybere se jako aktivní.

5.6 FilterSettings

Pokud je naopak nějaký filtr zvolen, zobrazuje se Nastavení filtru pomocí této komponenty. Po hlavní komponentě ImageEditApp je tato nejsložitější. Kromě definice vzhledu poskytuje logiku pro dvě dostupná tlačítka – odstranění filtru a nastavení výchozích hodnot filtru. Tato tlačítka jsou realizována společnou komponentou `SettingsIconButton`. Liší se rozdílnou ikonkou a akcí na klik v argumentech.

5.6.1 Formulář

Další funkcionalitou této komponenty je vytváření formuláře pro nastavení hodnot aktuálního filtru. Formulář se vytváří ve dvou krocích. Prvně se najde konfigurace pro typ aktivního filtru v `filters.js`.[5] Poté se projde každá skupina nastavení a vytvoří se jednotlivé vstupy. Tyto vstupy se vloží do skupiny, která se přidá do formuláře.

5.6.2 Vstupy

Jednotlivé vstupy se vytváří podle svého typu. Momentálně jsou k dispozici tři varianty: posuvník `InputSlider`, zaškrtávací políčko `InputCheck` a výběr barvy `InputColor`. Každý je vlastní komponentou. Systém je připraven na přidání více vstupů.[6]

Logika komponent vstupů je v základu podobná – při změně aktivního filtru se aktualizují jejich hodnoty na hodnoty nově zvoleného filtru pomocí useEffect hook a při manupulaci se vstupem jsou tyto hodnoty uloženy do useState hook,

[5] Zmíněný v Struktura projektu.
[6] Více v Možnosti rozšíření.

který ukládá hodnoty pro všechny aktivní filtry.[7] První dva vstupy jsou založeny na vstupech z Material UI, takže obsahují i jejich potřebnou logiku. Vstup na barvy je vytvořen pomocí knihovny react-color a také obsahuje logiku potřebnou pro práci s touto komponentou.

Největším problémem bylo zařídit, aby u posuvníku a výběru barvy fungoval jak samotný vstup z posuvníku nebo okna na barvy, tak i z textového pole. Pro provázání těchto dvou možností vstupu bylo nutné vytvořit useState hook, který se měnil s každou změnou jak vstupu tak i textového pole. Tento přístup ale vytvořil nežádoucí chování, když chtěl uživatel vše smazat nebo začal psát nepovolené znaky. Problém jsem vyřešil upravením funkce pro reakci na změny v textovém poli tak, že neměnila hodnotu, když byl na vstupu neplatný řetězec. Navíc jsem povolil prázdný řetězec, který se ale po odkliknutí textového pole převedl na výchozí hodnotu vstupu.

5.6.3 Změna pořadí filtru

Poslední akcí v tomto panelu je možnost změny pořadí aktuálního filtru. K tomu slouží textový vstup v horní části pod názvem filtru a šipky k němu připojené. Logiku řeší komponenta `OrderSettings`.

Protože jsem chtěl, aby vstup ihned aktualizoval pořadí, narazil jsem na podobný problém jako u posuvníku a výběru barvy. Jelikož byl problém skoro stejný, mohl jsem využít podobného řešení.

[7] Hook se jmenuje `appliedFilters` a udržuje objekt s klíči podle id filtrů. Každý záznam je znovu objekt s klíči odpovídající id vstupu.

6 Programátorská dokumentace serveru

Vstupním bodem do API serveru je soubor `main.py`, který po spuštění zprovozní server pomocí služby uvicorn [13]. Hlavní kód je obsažen v souboru `api.py`. Pod importem potřebných knihoven se nachází základní nastavení FastAPI. Server po každém spuštení promaže obrázky od uživatelů, které mohly z důvodu chyby zůstat uložené.

Klienti jsou rozlišováni pomocí jejich IP adresy. Při každém zahájení spojení[8] se klientovi vytvoří složka, ve které bude uložen neupravený obrázek i poslední verze upraveného obrázku. Aby nebyla přímo vidět IP adresa klienta, využívám hashe této adresy jako jeho id. Po tom, co je přijat obrázek od klienta, se přidá záznam do `last_client_responces`, kde se naváže jeho id na aktuální čas. V tuto chvíli se zahájí pravidelná kontrola,[9] jestli daný čas nepřekročil povolenou dobu neaktivity.[10] Při každé komunikaci je tento čas aktualizován.

Server reaguje na čtyři požadavky. Prvním je POST na `/img`, pomocí kterého dokáže server přijmout obrázek a vytvořit klientovi složku. Na stejné adrese pod GET požadavkem reaguje server zasláním poslední verze klientova upraveného obrázku. Dalším požadavkem je POST na adrese `/imgConfig`, který přijímá nastavení filtrů a ihned je aplikuje na klientův uložený obrázek. Pokud obrázek neexistuje, vrací status 410. Na to může klient reagovat, jak bylo řečeno dříve v Komunikace se serverem. Posledním požadavkem je na adresu `/clear`, který ukončí komunikaci s klientem a smaže jeho složku.

Aplikace filtrů probíhá pomocí funkce `handler`. Tato funkce postupně prochází jednotlivé nastavení filtrů přijaté od klienta a podle typu filtru aplikuje na obrázek daný filtr s příslušným nastavením.

[8] Přijetí obrázku od klienta.
[9] Každou minutu.
[10] Deset minut.

7 Dokumentace knihovna s filtry

Všechny filtry se nachází ve složce `scripts/filters`. Jednotlivé filtry jsou rozděleny do souborů podle jejich stylu. Každá funkce znázorňuje jeden filtr s výjimkou pomocných funkcí začínajících na „podtržítko“ a těch ve složce `misc`.

Všechny neupravené obrázky v této části byly získány ze stránky Pexels [22], která nabízí různé fotografie volně k využití.

7.1 Adjustments

První filtr v aplikaci dovoluje uživateli aplikovat různé základní úpravy obrázku. Mezi ně patří změna jasu a kontrastu, zostření nebo rozmazání, ekvalizace histogramu, invertování barev, vodorovné a horizontální překlopení obrazu. Všechny tyto operace se provádí pomocí funkcí knihovny Pillow.

7.2 Digital painting

Hlavním kreslícím filtrem je digitální kresba. Tento filtr byl vytvořen první a byly na něm založeny další. Proto také jiné filtry využívají stejné způsoby hledání obrysů a úpravy barev jako tento.

7.2.1 Kreslené obrysy

Obrysy vytváří pomocí funkce `make_outline`. Tato funkce hledá okraje pomocí kombinace Sobelových filtrů [14]. Jedná se o kernel filtry [15], které poznávají vysoké změny hodnot na obrázku v jednom směru. Příklad pravého Sobelova filtru, který zvýrazní hranice na pravé straně a potlačí na levé:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Kernel filtry se aplikují na každý pixel zvlášť, a tím kombinují hodnotu jeho a okolí. Právě vypočítávaný pixel odpovídá prostřední hodnotě ve filtru. Hodnoty filtru jsou koeficienty, kterými se násobí hodnoty pixelu a jeho okolí. Výsledný pixel se spočítá sečtením těchto vynásobených hodnot.

Při hledání hran pomocí Sobelových filtrů se většinou využívá kombinace jednoho horizontálního směru a jednoho vertikálního. Já ale využil kombinace všech čtyř, protože vytvářely výsledek více připomínající kreslenou čáru. Každý směr filtru se aplikuje na výchozí obrázek zvlášť a poté se spojují pomocí výpočtu přepony trojúhelníku tak, že vždy spojujeme směry na sebe kolmé.

Protože hledání hran pomocí této techniky vytváří vždy stejně tlusté čáry, musel jsem vyřešit změnu tloušťky pomocí změnění velikosti vstupního obrázku a následného vrácení do původní velikosti.

Pokud je obrázek před hledáním hran rozmazán, působí hrany hladčeji, a pokud je obrázku zvýšen kontrast, je hledání přesnější.

Nakonec je odebrána černá barva a ponechána pouze zvolená barva na místě bílé pomocí funkce `remove_black`.

7.2.2 Změna barev

Filtr provádí změnu barev ve dvou krocích pomocí dvou funkcí `get_palette` a `apply_palette`.

K získání určitého počtu barev z obrázku je využíván algoritmus k-means, který se používá pro shlukování dat. Jelikož jsem chtěl omezit počet barev na určitý počet, využil jsem tohoto algoritmu k vytvoření shluků barev obrázku. V těchto shlucích jsem vybral prostřední hodnotu, a tím dostal požadovaný počet barev. Protože tento filtr neměl vytvořit úplně ploché barvy, ke každé barvě byl přidán světlý a tmavý odstín.

Barevná paleta se poté aplikuje pomocí funkcí knihovny Pillow. Prvně se vytvoří obrázek z barev palety, a poté se pomocí něj a metody `quantize` aplikuje paleta na obrázek.

Obrázek 3: Obrázek před a po aplikování Digital painting

7.3 Oil painting

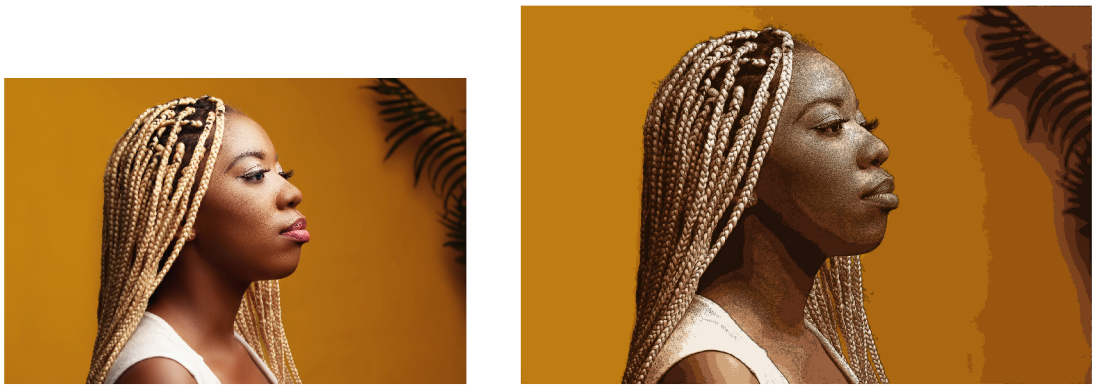
Dalším kreslícím filtrem je olejomalba. Kreslené obrysy se získávají stejně jako u předešlého filtru. Hlavní změnou je proces změny barev, který je u tohoto filtru jednodušší. Využívá totiž techniky posterizace [16], která snižuje počet barev pro každý barevný kanál. Tento proces společně s rozmazáním vytvoří efekt připomínající olejomalbu.

Obrázek 4: Obrázek před a po aplikování Oil painting

7.4 Flat painting

Alternativní verze Digital painting. Využívá stejných funkcí pro změnu barev s tím rozdílem, že vytváří ploché barvy, takže nepřidává další odstíny.

Tento filtr využívá jiný způsob získávání obrysů. Místo funkce `make_outline` používá `make_sketch`, která je založená filtru Sketch. Náčrtek je navíc invertován aby mohla být použita funkce `remove_black` pro odstranění pozadí a ponechání náčrtku samotného. Následně je spojen s obrázkem.



Obrázek 5: Obrázek před a po aplikování Flat painting

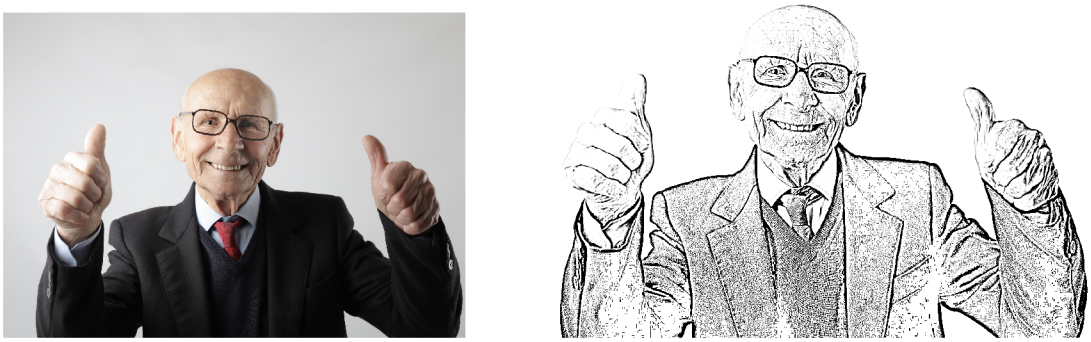
7.5 Sketch

Filtr Sketch [17] simuluje náčrt tužkou. Prvně převede obrázek na šedotónový a poté ho vydělí rozmazanou verzí sebe sama. Rozmazáním ztrácí obrázek detail a každý pixel získá informace o svém okolí. Vydělením získá hodnoty bližší nule, na místech kde je větší rozdíl s okolím, a bližší jedničce, tam kde jsou okolní hodnoty podobné. Poté vynásobením celého obrázku číslem 255[11] získá výsledný

[11] Hodnota pro bílou barvu v šedotónovém obrázku.

náčrt, protože oblasti s velkými rozdíly budou černé a s malými rozdíly budou bílé.

V závislosti na síle rozmazání budou čáry výraznější a po přidání kontrastu na konci procesu zase viditelnější. Proto byly tyto dvě hodnoty přidány jako argument.



Obrázek 6: Obrázek před a po aplikování Sketch

7.6 Filtry založené na vkládání objektů

Následující skupina čtyřech filtrů je založena na modulu `ImageDraw` z knihovny Pillow. Postupně prochází obrázek a podle aktuálního pixelu vytvoří objekt v jeho barvě. Pro zajímavější efekt jsou tyto objekty postupně otáčeny, aby nevypadaly monotónně. Tento průchod obstarává funkce `brush_strokes`, který má simulovat tahy štětcem. Štětci lze nastavit šířka, délka tahu, tvar a jak moc se má každý další tah otočit. Tvarem může být čára nebo elipsa. Každému tahu se pomocí goniometrických operací vypočítá místo, kam má být vložen, aby bylo dodrženo pravidelné otočení.

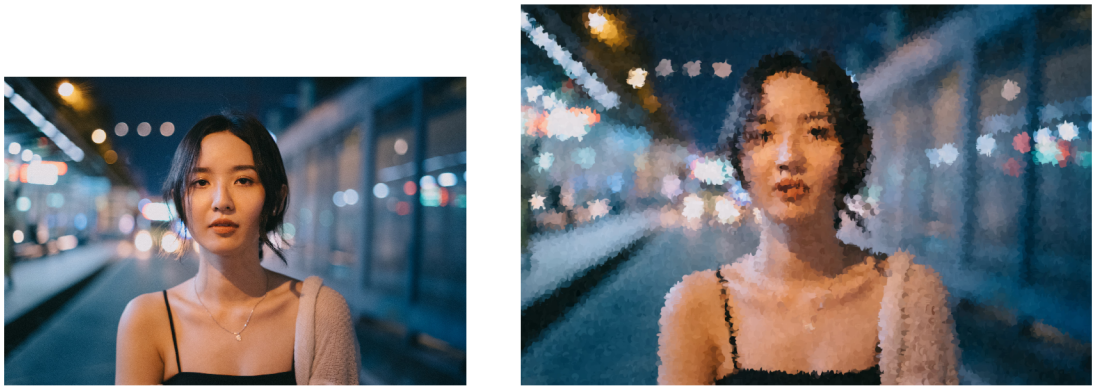
7.6.1 Brushstroke painting

Prvním z těchto filtrů je Brushstroke painting, který simuluje tahy štětcem, takže přímo využívá funkce `brush_strokes` bez větších úprav. Jako tvar štětce používá čáry. Pod vytvořené tahy štětcem je vložen rozmazaný původní obrázek, aby vyplnil možné mezery, které při výrobě tahů mohly vzniknout. Mezery jsou velice malé, takže tento podklad nenaruší efekt. Nakonec je celý obrázek uhlazen, aby tahy nebyly příliš ostré.

Obrázek 7: Obrázek před a po aplikování Brushstroke painting

7.6.2 Watercolor painting

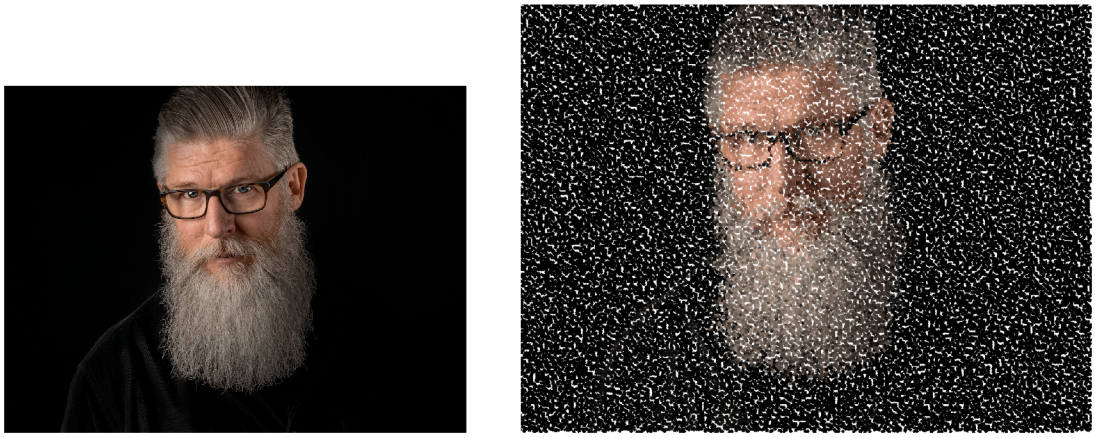
Druhý filtr funguje velice podobně jako filtr předešlý, ale s tím rozdílem, že využívá možnosti funkce `brush_strokes` na vkládání elips místo čar. Navíc obrázek nakonec rozmaže. To vytváří efekt, jakoby byl obrázek nakreslen vodovými barvami.



Obrázek 8: Obrázek před a po aplikování Watercolor painting

7.6.3 Pointillism

Třetím využitím funkce `brush_strokes` je filtr připomínající pointilismus [18]. Tento filtr funguje velice podobně jako Brushstroke painting. Na rozdíl od něj ale nabízí jen jednu hodnotu pro velikost štětce. Protože vytváří jednotlivé body, musí mít tahy stejnou tloušťku i délku. Dalším rozdílem je, že podkladem není původní obrázek, ale barva zvolená uživatelem, nejčastěji bílá.

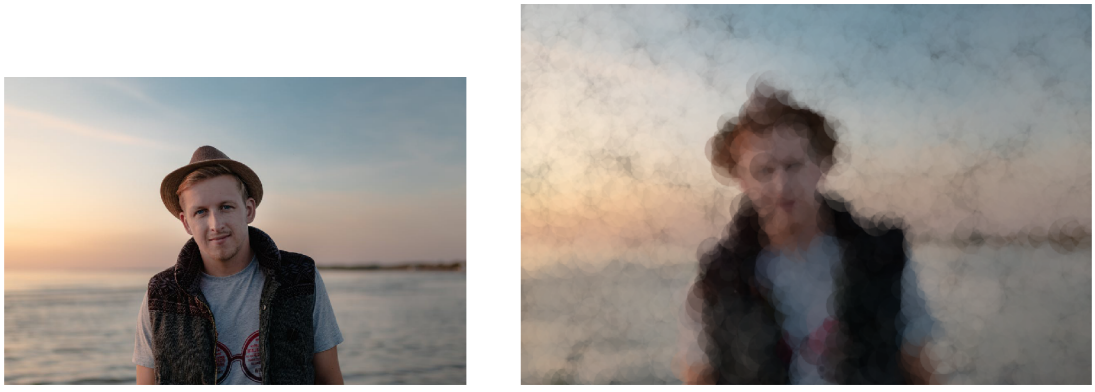


Obrázek 9: Obrázek před a po aplikování Pointillism

7.6.4 Bubbles

Poslední filtr z této skupiny imituje vrstvené bubliny. Používá upravenou verzi `brush_strokes`, která vždy kreslí kolečka. Ty místo otáčení průběžně mění svou velikost. Navíc jsou z části průhledné, což dopomáhá jejich vrstvení. Modul `ImageDraw` nedovoluje přímo vkládat průhledné tvary do obrázku, takže musel být pro každou bublinu vytvořen obrázek, který se přiložil na výsledek.

Tento filtr navíc dovoluje vybrat, ze které strany budou bubliny vkládány.



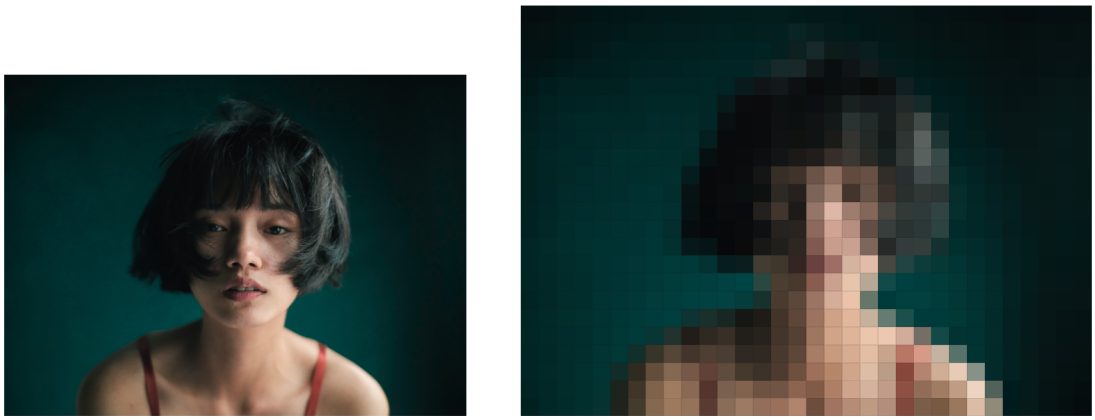
Obrázek 10: Obrázek před a po aplikování Bubbles

7.7 Pixel art

Další filtr převede obrázek na takzvaný Pixel art [19]. Tento styl je založen na skládání barevných čtverečků (pixelů) pro vytvoření výsledného obrázku. Tento styl byl v minulosti využíván ve videohrách z důvodu limitů technologie té doby.

Můj algoritmus prvně určí delší stranu obrázku a té přidělí uživatelem zadaný počet pixelů pomocí `pix_count`. Množství pixelů na kratší straně je dopočítáno

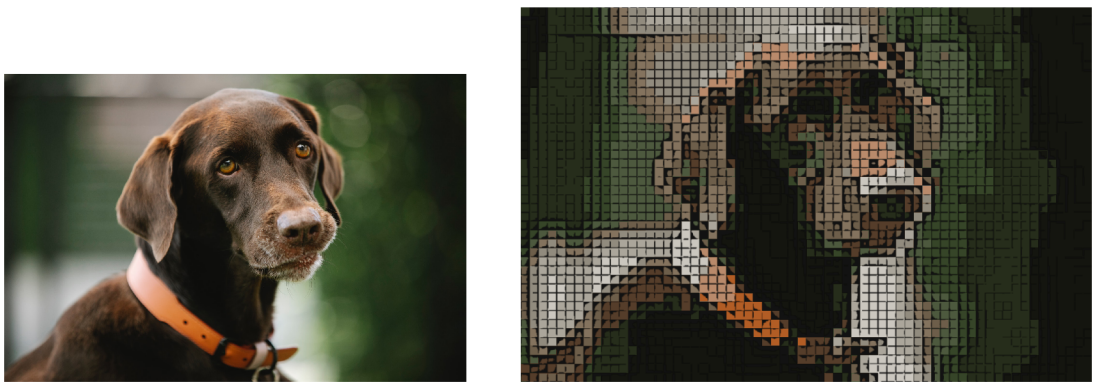
tak, aby byl přibližně zachován poměr stran. Následně je změněna velikost obrázku podle těchto hodnot, aby ho dále mohl algoritmus rozdělit do jednotlivých čtverečků. Poté se projde každý čtvereček a zjistí se medián[12] jeho barev v původním obrázku. Tato hodnota je zvolena jako barva ve výsledku na odpovídajícím čtverečku, který má velikost odpovídající uživatelem zadané hodnotě `points_per_pix`, díky které se dá obrázek jednoduše škálovat. Nakonec je čtverečku přidán okraj, pokud byl na vstupu povolen.



Obrázek 11: Obrázek před a po aplikování Pixel art

7.8 Block art

Při zkoušení kombinací filtrů jsem narazil na zajímavou interakci mezi Pixel art a Flat painting. Spojení těchto filtrů vytvořilo efekt jako by byl obrázek postaven z kostiček. Tato kombinace byla přidána s lehkou úpravou argumentů jako samostatný filtr.



Obrázek 12: Obrázek před a po aplikování Block art

[12]Medián byl zvolen pro jeho hezčí výsledky a zachování barev oproti průměru hodnot.

7.9 Filtry založené na práci s šedotónem

Skupina filtrů pracujících s šedotónovou variantou obrázku,[13] nebo přesněji pouze s jeho jasovou hodnotou. Tyto filtry navíc dovolují nahradit přechod z černé do bílé přechodem mezi dvěma libovolnými barvami.

7.9.1 Colorize

První z nich dělá přímo jen to, co bylo popsáno výše, bez dalších úprav. Využívá funkci z knihovny Pillow. Při základním nastavení vrací šedotónový obrázek.

Obrázek 13: Obrázek před a po aplikování Colorize

7.9.2 Halftoning

Podobně jako Pixel art simuluje tento filtr efekt z doby minulých, který byl zapříčiněn limitací technologií. Efekt polotónování [20], kterého tento filtr využívá, se používal při simulování barevného přechodu mezi černou a bílou pomocí různě velkých teček. Při využití výběru barvy pozadí a teček tato metoda nabízí zajímavý efekt.

K získání teček využívá podobného způsobu jako Pixel art, a to změnění velikosti obrázku tak, aby v něm šla udělat pravidelná mřížka čtverečků. Poté, co je obrázek převeden na šedotónový, je každému čtverečku zjištěna průměrná intenzita a podle ní je určena velikost kolečka, které bude vloženo do výsledku na odpovídající místo. Filtr je založen na obyčejném polotónování, které přidává černé tečky na bílé pozadí. Pro získání barevného efektu je tento černobílý obrázek použit jako maska pro vrstvu barvy popředí. Tento obrázek je poté vložen na vrstvu barvy pozadí.

[13] Pokud vynecháme Sketch, který s šedotónem pracuje jinak.

Obrázek 14: Obrázek před a po aplikování Halftoning

7.9.3 Thresholding

Poslední filtr ze skupiny těch, které pracují s jasem obrázku, využívá metody prahování [21]. Tento princip je založen na tom, že každému pixelu přidělí buď barvu bílou, pokud má jasovou hodnotu vyšší než zvolený práh, nebo barvu černou, pokud je hodnota nižší. Moje verze nabízí uživateli vybrat, jaké dvě barvy budou zvoleny. Navíc si uživatel může zvolit vyšší hloubku obrazu, která přidá každé barvě další odstín – tmavší náhradě za černou a světlejší náhradě za bílou. Pokud je hloubka zvolena, rozsah pro každou barvu je rozdělen na půl, a tím je přidán další práh.

Obrázek 15: Obrázek před a po aplikování Thresholding

8 Možnosti rozšíření

Aplikace vytvořena s úmyslem, aby bylo jednoduché přidávat další filtry. Nejjednodušší je přidání filtru napsaném v jazyce Python a postaveném (alespoň v základu) na knihovně Pillow, stejně jako všechny momentálně přidané filtry.

8.1 Přidání filtru do backendu

Když je použita knihovna Pillow, přidání filtru je velice jednoduché. Samotný filtr musí být importován do `handler.py`. Aby se filtr použil v průběhu aplikace, musí pro něj být přidána větev do `match`. String, který se přidává do `case`, odpovídá názvu filtru, který bude mít na frontendu.

Pokud by nebyl filtr založen na knihovně Pillow, je potřeba obrázek uložit v příslušné `case` větvi filtru pomocí knihovny Pillow. Poté filtr pracuje s cestou, kde byl obrázek uložen. Po dokončení práce filtru bude zase potřeba obrázek otevřít pomocí Pillow, aby nebyl narušen průchod ostatními filtry.

Použití jiného jazyka by bylo větší komplikací, protože by v průběhu aplikování filtrů v `handler.py` bylo potřeba tento jiný jazyk volat přímo z Pythonu. To by se ale dalo vyřešit například dalším API serverem pro tento jazyk.

8.2 Přidání filtru do frontendu

Aby se filtr zobrazoval na frontendu, musí být přidáno jeho vlastní nastavení do `filters.js` a přidán obrázek s jeho jménem ve formátu jpg do složky `src/img`. Pro rychlejší načítání a zobrazování, by měl obrázek mít stejnou velikost jako obrázky, které už se v aplikaci nachází, a to 200 × 115px. Obrázek je povinný.

Pro přidání nastavení filtru je nutno přidat záznam do `FILTERS`. Klíčem bude jeho název[14] a hodnotou bude objekt. Ten bude obsahovat klíč `description` s hodnotou, která udává popisek filtru potom, co je na něj najeto v aplikaci a druhý klíč `groups`, který bude obsahovat skupiny filtrů v seznamu. Každá skupina je objektem s názvem pod klíčem `name` a seznamem jednotlivých vstupů pod klíčem `settings`. Vstup je znovu objekt s klíči:

- `type` – určení typu vstupu; `slider` pro posuvník, `color` pro výběr barvy a `check` pro zaškrtávací políčko
- `id` – označení filtru, musí odpovídat argumentu funkce, která je volána na backendu při aplikaci filtru, když je filtr volán standardním způsobem
- `name` – zobrazený název
- `tooltip` – popis filtru, který se zobrazí po najetí na ikonku se znakem „i“

[14] Stejný, jaký byl zvolen na backendu.

- `baseVal` – výchozí hodnota vstupu, když je filtr přidán nebo nastaven na výchozí nastavení; při typu `slider` musí být číslo, při `color` barva v HEX kódování a při `check` pravdivostní hodnota
- `minVal` a `maxVal` – tyto hodnoty jsou potřeba pouze při typu `slider`, označují nejnižší a nejvyšší možnou hodnotu

```
"Example": {
    "description": "nothing.",
    "groups": [
        {
            "name": "test group",
            "settings": [
                {
                    "type": "slider",
                    "id": "slider",
                    "name": "Slider",
                    "tooltip": "slider",
                    "minVal": 1,
                    "maxVal": 10,
                    "baseVal": 5
                },
                {
                    "type": "color",
                    "id": "color",
                    "name": "Color",
                    "tooltip": "color",
                    "baseVal": "#ffa647"
                },
            ],
        },
        {
            "name": "test group2",
            "settings": [
                {
                    "type": "check",
                    "id": "check",
                    "name": "check",
                    "tooltip": "check",
                    "baseVal": true
                }
            ]
        }

    ]
},
```

Obrázek 16: Ukázkový kód pro přidání filtru do frontendu

Závěr

Výsledná aplikace splňuje všechny cíle, které jsem si stanovil. I když filtry pracují jen v jednotkách sekund, je rychlost jedním z jejich největších nedostatků. Tím pádem pokud by se obrázky automaticky překreslovaly při změně jejich nastavení, narušovalo by to plynulost používání aplikace. Ostatní body byly podle mého názoru splněny úspěšně.

Knihovna s filtry je nezávislá na aplikaci, ale v kombinaci s ní dovoluje přívětivější používání. Dala by se považovat za rozšíření knihovny Pillow, protože všechny operace aplikuje na reprezentaci obrázku z této knihovny.

Při vývoji mi zabralo nejvíce času vytváření filtrů, zejména jejich optimalizace. Došel jsem k tomu, že práce s obrázky je časově náročná, hlavně pokud chceme zachovat vyšší kvalitu výsledků.

Filtry, které jsem vytvořil, pracují s každým obrázkem stejně, a to postupným procházením, takže v obrázcích nehledají žádné objekty nebo vzory. Tento přístup je jednodušší a dovolil mi vytvořit více filtrů rychleji. Díky tomu aplikace působí robustněji. Složitější filtry lze přidat bez větších problémů právě díky jednoduché rozšířitelnosti. Aplikace slouží jako dobrý základ pro větší systém.

Conclusions

Completed application fulfils all the established goals. It mostly lacks in terms of the speed of the filters, even though it takes only a couple of seconds. With that being said, if the image was automatically redrawn after changing the settings of the filters, it would make the application usage unpleasant for the users. Other goals were fulfilled successfully, in my opinion.

The filter library is independent from the application, however, when combined, the user experience is enhanced. It could be considered as an expansion of the Pillow library because it applies all the operations on the image representation from this library.

While developing the application, creating the filters was the most challenging task, especially the optimisation. I came to the conclusion that manipulating with images is very time demanding process, more so when wanting to conserve the high quality of the images.

Filters, which i created, manipulate with every single picture in the same way by going step by step through it, therefore they do not look for any objects or patterns. This simpler approach allowed me to create more filters faster. Thanks to that, the application appears to be richer. Complicated filters can be added very easily because of the simple extensibility of the application. Therefore, the application serves as a great foundation for a bigger system.

A Obsah elektronických dat

Součástí práce jsou i všechna data potřebná pro zprovoznění aplikace a vytvoření PDF dokumentu textu. Odevzdaná data mají strukturu:

`app/`
: Adresář se soubory potřebnými pro zprovoznění backendové a frontendové části aplikace. Adresář obsahuje dva podadresáře odpovídající názvem jednotlivým částem aplikace – `backend/` a `frontend/`.

`text/`
: Adresář s textem práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce a všechny soubory potřebné pro bezproblémové vytvoření PDF dokumentu textu, tj. zdrojový text textu, vložené obrázky, apod.

`README.txt`
: Textový soubor s informacemi ke zprovoznění software na lokální síti. Obsahuje návod s kroky popisujícími, který software a knihovny je potřeba nainstalovat pro spuštění aplikace. Dále obsahuje informace o zprovozněné testovní verzi na školní síti.

Literatura

[1] BeFunky [online]. [cit. 2023-04-17]. Dostupné z: https://www.befunky.com/create/

[2] Fotor [online]. [cit. 2023-04-17]. Dostupné z: https://goart.fotor.com/

[3] Prisma [online]. [cit. 2023-04-17]. Dostupné z: https://prisma-ai.com/

[4] Python [online]. [cit. 2023-04-04]. Dostupné z: https://www.python.org/

[5] Pillow [online]. [cit. 2023-04-04]. Dostupné z: https://pillow.readthedocs.io/en/stable/index.html

[6] NumPy [online]. [cit. 2023-04-04]. Dostupné z: https://numpy.org/

[7] FastAPI [online]. [cit. 2023-04-04]. Dostupné z: https://fastapi.tiangolo.com/

[8] React [online]. [cit. 2023-04-04]. Dostupné z: https://react.dev/

[9] MaterialUI [online]. [cit. 2023-04-04]. Dostupné z: https://mui.com/

[10] Material Design [online]. [cit. 2023-04-04]. Dostupné z: https://m3.material.io/

[11] React Color [online]. [cit. 2023-04-04]. Dostupné z: https://casesandberg.github.io/react-color/

[12] scikit-learn [online]. [cit. 2023-04-04]. Dostupné z: https://scikit-learn.org/stable/

[13] Uvicorn [online]. [cit. 2023-04-04]. Dostupné z: https://www.uvicorn.org/

[14] FISHER, Robert; PERKINS, Simon; WALKER, Ashley; WOLFART, Erik. Sobel Edge Detector [online]. [cit. 2023-04-04]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

[15] POWELL, Victor. Image Kernels [online]. [cit. 2023-04-04]. Dostupné z: https://setosa.io/ev/image-kernels/

[16] Posterization. Wikipedia [online]. [cit. 2023-04-04]. Dostupné z: https://en.wikipedia.org/wiki/Posterization

[17] ACES PVGCOET. Convert a photo to different types of sketches using python. Medium [online]. [cit. 2023-04-04]. Dostupné z: https://medium.com/1-hour-blog-series/convert-a-photo-to-different-types-of-sketches-using-python-26f005a75f66

[18] Pointillism. Wikipedia [online]. [cit. 2023-04-16]. Dostupné z: https://en.wikipedia.org/wiki/Pointillism

[19] Pixel art. Wikipedia [online]. [cit. 2023-04-05]. Dostupné z: https://en.wikipedia.org/wiki/Pixel_art

[20] Halftone. Wikipedia [online]. [cit. 2023-04-05]. Dostupné z: https://en.wikipedia.org/wiki/Halftone

[21] Thresholding (image processing). Wikipedia [online]. [cit. 2023-04-05]. Dostupné z: https://en.wikipedia.org/wiki/Thresholding_(image_processing)

[22] Pexels [online]. [cit. 2023-04-04]. Dostupné z: https://www.pexels.com/