Praktické nasazení Google Vision API

Bc. Marek Jedlinský

Diplomová práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Marek Jedlinský**
Osobní číslo:	**A17228**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**prezenční**
Téma práce:	**Praktické nasazení Google Vision API**
Téma anglicky:	**The Practical Deployment of the Google Vision API**

Zásady pro vypracování:

1. **Prostudujte možnosti cloudové platformy společnosti Google, konkrétně služby Vision API, v oblasti klasifikace obrázků a uveďte také alternativní služby jiných dodavatelů.**
2. **V práci uveďte také aktuální platební podmínky služby Vision API či případných alternativ.**
3. **Seznamte se s webovou aplikací společnosti WrapStyle s.r.o., která slouží jako databáze obrázků s automobilovými designy a navrhněte možnosti automatické klasifikace obrázků automobilových designů.**
4. **S využitím některého z výše uvedených API implementujte modul pro webovou aplikaci společnosti WrapStyle s.r.o., který by umožňoval nahrávání obrázků a jejich automatickou klasifikaci.**
5. **Otestujte také možnost trénování klasifikace dle vlastních kategorií a v případě, že to bude vybrané API umožňovat, implementujte k tomuto účelu uživatelské rozhraní.**
6. **Proveďte reálný test klasifikace obrázků pomocí implementovaného modulu a výsledky shrňte.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **RAVULAVARU, Arvind. Google Cloud AI Services Quick Start Guide: Build intelligent applications with Google Cloud AI services. Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-661-3.**
2. **CIABURRO, Giuseppe. Hands-On Machine Learning on Google Cloud Platform: Implementing smart and efficient analytics using Cloud ML Engine. Birmingham: Packt Publishing, 2018. ISBN 978-1-78839-348-5.**
3. **PS, Legorie Rajan. Google Cloud Platform Cookbook: Implement, deploy, maintain, and migrate applications on Google Cloud Platform. Birmingham: Packt Publishing, 2018. ISBN 978-1-78829-199-6.**
4. **THODGE, Sanket. Cloud Analytics with Google Cloud Platform: An end-to-end guide to processing and analyzing big data using Google Cloud Platform. Birmingham: Packt Publishing, 2018. ISBN 978-1-78883-968-6.**
5. **Machine Learning. SAS [online]. SAS Institute, 2018 [cit. 2018-11-28]. Dostupné z: https://www.sas.com/en_hk/insights/analytics/machine-learning.html**
6. **Laravel [online]. Taylor Otwell [cit. 2018-11-28]. Dostupné z: https://laravel.com/**
7. **BISHOP, Christopher M. Pattern recognition and machine learning. New York: Springer, c2006. Information science and statistics. ISBN 978-0387310732.**
8. **Microsoft Azure: Computer Vision [online]. Microsoft, 2018 [cit. 2018-11-28]. Dostupné z: https://docs.microsoft.com/en-gb/azure/cognitive-services/computer-vision/home**

Vedoucí diplomové práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2018**

Termín odevzdání diplomové práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13. 5. 2019 Marek Jedlinský, v. r.

ABSTRAKT

Tato diplomová práce se zabývá praktickým využitím cloudové služby poskytující strojové učení v podobě počítačového vidění pro klasifikaci obrázků na úrovni klasifikace výrobce, modelu, karosérie a barvy vozidel. Práce se také zabývá možností naučení vlastního klasifikátoru pro klasifikaci vlastních tříd. V teoretické části práce je popsáno strojové učení a různé metody a algoritmy určené pro strojové učení. V praktické části je rozebrán návrh řešení, samotná implementace a jsou zhodnoceny výsledky vytvořeného programu.

Klíčová slova: Strojové učení, počítačové vidění, klasifikace, Google Cloud Vision, Google Cloud AutoML Vision, PHP, Laravel

ABSTRACT

This Master's thesis deals with practical use of cloud services which provides machine learning in form of computer vision used for image classification on the level of vehicle's manufacturer, model, bodywork and colour. The thesis also deals with possibility of training own classifier for classification of custom classes. In the theoretical part of thesis are described machine learning in general and different methods and algorithms used for machine learning. In practical part is described implementation plan, implementation itself and also there are evaluated results of developed program.

Keywords: Machine learning, computer vision, classification, Google Cloud Vision, Google Cloud AutoML Vision, PHP, Laravel

Velmi rád bych chtěl na tomto místě poděkovat vedoucímu mé práce, panu Ing. Radku Valovi, Ph.D., za odborné vedení, cenné rady a připomínky, které mi pomohly tuto práci zkompletovat.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 STROJOVÉ UČENÍ **11**
1.1 CONFUSION MATRIX 12
1.2 METODY UČENÍ 13
1.2.1 Učení s učitelem 13
1.2.2 Učení bez učitele 13
1.2.3 Učení s poloučitelem 13
1.2.4 Učení s posilováním 14
1.3 VYBRANÉ UČÍCÍ ALGORITMY 14
1.3.1 K-NearestNeighbors (KNN) 14
1.3.2 Naivní Bayesovský klasifikátor 14
1.3.3 K-Means 15
1.4 COMPUTER VISION 15
2 SLUŽBY POSKYTUJÍCÍ POČÍTAČOVÉ VIDĚNÍ **16**
2.1 GOOGLE CLOUD VISION 16
2.1.1 AutoML Vision 16
2.1.2 Ceník služeb 17
2.2 MICROSOFT AZURE – COGNITIVE SERVICES 18
2.2.1 Computer Vision 19
2.2.2 Ceník 20
2.3 AMAZON REKOGNITION 20
2.3.1 Ceník 21
3 POUŽITÉ TECHNOLOGIE **22**
3.1 PHP 22
3.2 LARAVEL 22
3.2.1 Composer 23
3.3 MVC 23
3.3.1 Model 24
3.3.2 View 24
3.3.3 Controller 24
3.4 VERZOVACÍ SYSTÉM 25
3.4.1 Git 25
II PRAKTICKÁ ČÁST **27**
4 WRAPSTOCK APLIKACE **28**
5 ZADÁNÍ A NÁVRH ŘEŠENÍ **30**
5.1 ZADÁNÍ 30
5.2 POŽADAVKY 30
5.3 REŠERŠE SLUŽEB POČÍTAČOVÉHO VIDĚNÍ 31
5.4 NÁVRH IMPLEMENTACE 32
5.4.1 Případ užití 32

5.4.2 Návrh struktury balíčku a databáze 33
6 IMPLEMENTACE ŘEŠENÍ 35
6.1 TVORBA BALÍČKU PRO LARAVEL 35
6.2 DATOVÝ TOK MEZI BALÍČKEM A GOOGLE VISION API 36
6.2.1 Tvorba požadavku pomocí Guzzle 37
6.2.2 Požadavky směřující na Google Cloud API 37
6.2.3 Odezva z Google Cloud Vision API 40
6.2.4 Odezva z Google Cloud AutoML Vision API 41
6.3 IMPLEMENTACE BALÍČKU 41
6.3.1 Třída VisionFunctions 41
6.3.2 Třída Authorization 44
6.3.3 Třída ColorCalculator 45
6.3.4 Modelové třídy 45
6.3.5 Konfigurační soubor 46
6.4 UČENÍ VLASTNÍHO KLASIFIKÁTORU POMOCÍ GOOGLE CLOUDAUTOML VISION 46
6.5 IMPLEMENTACE DO APLIKACE 49
6.5.1 Instalace balíčku 50
6.5.2 Implementace do aplikace Wrapstock 51
7 TESTOVÁNÍ 55
7.1.1 Testování balíčku 55
7.1.2 Testování v aplikaci Wrapstock 55
ZÁVĚR 57
SEZNAM POUŽITÉ LITERATURY 59
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 64
SEZNAM OBRÁZKŮ 66
SEZNAM TABULEK 67

ÚVOD

Cílem této práce je zmapování možností počítačového vidění v oblasti služeb poskytujících klasifikační modely pro implementaci do vlastní aplikace. Práce se tedy zaměřuje na aplikování některé ze zmapovaných služeb do praktického využití.

V práci je popsán potřebný teoretický aparát týkající se obecně strojového učení. V současné době má totiž strojové učení širokou škálu oblastí, ve kterých se může uplatnit. Strojové učení pokrývá relativně jednoduché oblasti, jako například predikce počasí, tak i oblasti komplexní, jako například zdravotnictví. Základem je vytvořit a použít správně naučený model na požadovanou oblast své činnosti. Různé metody učení a algoritmy používané k učení jsou rozebrány v první části této práce.

Jednou z oblastí uplatnění strojového učení je i takzvané počítačové vidění, které má za cíl klasifikovat a identifikovat údaje v požadovaných obrazových datech, nejčastěji obrázcích. Strojové učení tedy plní funkci člověka, který například identifikuje všechny objekty na uvedeném obrázku. V práci jsou popsány cloudové služby poskytující v rámci svých nabízených produktů i strojové vidění v oblasti klasifikace obrázků. Protože se jedná o placené služby, jsou v práci uvedeny i aktuální platební podmínky jednotlivých služeb.

Protože cílem práce je praktická aplikace některé ze služeb poskytujících klasifikaci obrázků, je součástí práce popsána i vytvořená aplikace implementující funkce zvolené služby. Před samotným popisem programové části je popsán i návrh celé implementace včetně požadavků na technologie a funkcionalitu vytvořeného programu. Součástí vývoje programu je dále kromě návrhu a následné implementace i testování vytvořeného řešení. V této závěrečné části práce je popsáno testování vytvořeného programu implementovaného do praktické aplikace. Kromě samotného testování jsou v této části práce také popsány a zhodnoceny výsledky jak použití připraveného klasifikátoru poskytnutého zvolenou službou, tak i výsledky klasifikátoru naučeném na vlastních datech pro klasifikaci uživatelsky zvolených tříd.

I. TEORETICKÁ ČÁST

1 STROJOVÉ UČENÍ

Strojové učení je podoblast umělé inteligence, která zahrnuje algoritmy umožňující strojům učit se na základě poskytnutých dat. Pomocí strojového učení lze rozpoznávat tvary a vzory, klasifikovat a třídit data, předpovídat budoucí stav na základě předcházejících zkušeností bez větší účasti člověka. [1, 2, 3, 4]

Cílem strojového učení je vytvoření modelu, který bude schopen, na základě naučení ze vstupních dat, predikovat nový výstup z neznámých vstupních dat. Modelem se zde chápe výsledek z učícího algoritmu. Pro vytvoření modelu je tedy důležité zvolit vhodný učící algoritmus na základě řešeného problému. Algoritmy jsou často vyvíjeny tak, aby napodobovali způsob lidského učení. [3, 5, 4]

Vstupem jak do učících algoritmů, tak následně do naučeného modelu, jsou data s určitými atributy. Tyto atributy představují konkrétní naměřené nebo jiným způsobem specifikované vlastnosti jedince. Jedincem se rozumí jedna instance dat. Atributy jedinců mohou nabývat různých hodnot, podle kterých jsou atributy členěny. [3]

- Binární atributy – Atributy nabývají přesně dvou hodnot, a to Pravda / Nepravda. [3]
- Nominální atributy – Atributy z neuspořádané množiny prvků o libovolné velikosti. [3]
- Numerické – Atributy nabývající reálných nebo celočíselných hodnot. [3]
- Ordinální – Atributy nabývající hodnot z konečné množiny uspořádaných prvků. [3]

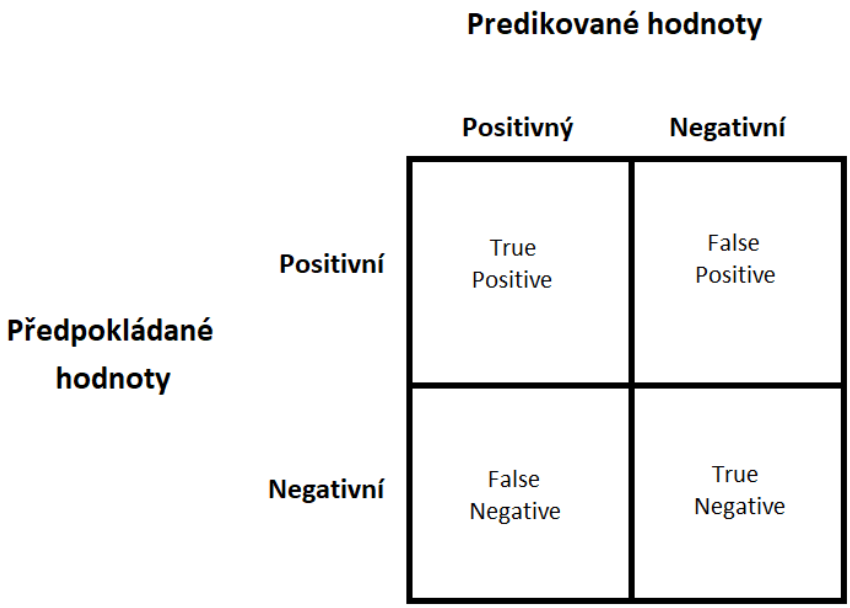
Pro naučení modelu se musí zvolit vhodný soubor dat (dataset) obsahující instance podobné těm, které má naučený model rozpoznávat. Tento dataset se v rámci učení rozděluje na dvě nebo tři množiny. První množina je trénovací, která obsahuje již klasifikovaná data. Na základě těchto dat, jak již z názvu množiny vyplývá, se model učí. Druhou množinou je množina testovací. Tato slouží k ověření úspěšnosti naučení modelu. Data v testovací množině jsou již také klasifikována, aby se rozpoznalo, zdali je model naučen správně. Data obsažená v testovací množině nesmí být obsažena v množině trénovací a jsou použita, až je model zcela naučen. Poslední množinou dat je validační množina. Pomocí těchto dat jsou parametry modelu upravovány, ale na jejich základu není model učen. Jedná se o jiný dataset, než je testovací. Validační množina se používá například při cross-validaci.[6, 7, 4]

V rámci učení může nastat nežádaný jev nazývaný přeučení, neboli overfitting. Je to případ, kdy se model naučí predikovat pouze konkrétní případy dat. Nastane tak, pokud se model příliš učí na datech a jejich šumu v trénovací množině. Tento šum se v nových vstupních datech neprojevuje stejně, proto je model přeučen. Opačným případem přeučení je underfitting. Jedná se o případ, kdy naučený model není schopen správně rozpoznávat ani vstupní data v trénovací případně testovací množině.[8]

1.1 Confusion Matrix

Důležitým ukazatelem kvality modelu je přesnost predikce. Ta lze získat prostým poměrem správně klasifikovaných tříd vůči celkovému počtu. K tomuto účelu slouží testovací, případně validační množina dat. Pro případy, kde je třeba klasifikovat pouze dvě třídy, je tato metoda dostačující. Protože jsou ale tyto případy spíše vzácné, využívá se takzvaná Confusion matrix. Jedná se o matici skládající se z četností jednotlivých tříd, klasifikovaných pomocí naučeného modelu. Díky této matici lze zjistit, jakých chyb klasifikace se model dopouští. [9]

Matice se sestavuje tak, že se po sloupcích uvádí predikované třídy získané z naučeného modelu. Po řádcích se uvádí třídy očekávané. Správně predikované hodnoty jsou ty, které se nachází na pozici, kde se kříží sloupec predikované hodnoty s řádkem požadované hodnoty. Tato matice lze použít na problémy dvou tříd a více. [9]



Obrázek 1 Confusion Matrix [10]

1.2 Metody učení

V rámci strojového učení existuje několik různých metod, kterými lze požadovaný model naučit. Dvě hlavní skupiny jsou učení s učitelem a učení bez učitele. Dále existuje učení s poloučitelem a učení s posilováním. [2, 11, 4]

1.2.1 Učení s učitelem

Algoritmy v této skupině pracují na takovém principu, že se učí na označené trénovací množině, tedy takové množině dat, kde jsou pro zvolené vstupy známy požadované výstupy. Využíváním metod jako jsou například klasifikace, regrese nebo predikce se pomocí nalezených vzorů v označených vstupních datech model upravuje tak, aby se minimalizovala chyba predikce vstupních neoznačených dat. [2, 11, 4]

Klasifikační úlohy slouží k predikci diskrétních hodnot, jako je například klasifikace elektronické pošty na legitimní zprávu či spam. Dalším příkladem klasifikačních úloh je rozpoznávání hlasu nebo computer vision, tedy zpracování obrazu. Běžně používané algoritmy pro klasifikační úlohy jsou například Support Vector Machines, rozhodovací stromy, k-nearest neighbor, neuronové sítě nebo Naivní Bayesovský klasifikátor. [11]

Regresní úlohy slouží oproti klasifikačním k predikci spojitých hodnot jako je například změna teploty nebo vývoj elektrické spotřeby. Algoritmy používané pro tento typ úloh jsou například lineární nebo nelineární regrese, rozhodovací stromy nebo stejně jako u klasifikačních úloh neuronové sítě. [11]

1.2.2 Učení bez učitele

Učení bez učitele využívá takových algoritmů, které pracují s neznámými, tedy neoznačenými daty. Tyto algoritmy hledají strukturu a vazby mezi vstupními daty. Nejběžnější metoda učení bez učitele je clustering, tedy seskupování dat na základě vzorů. Clustering lze využít například v obchodní sféře, kde na základě nákupů zákazníků lze zjistit různé zájmové skupiny. Algoritmy používané pro clustering jsou například k-means, samoorganizující se mapy, DBScan, a jiné. [2, 11, 4]

1.2.3 Učení s poloučitelem

Pro učení lze využít tzv. učení s poloučitelem, které lze využít na stejné problémy, na které by se využilo učení s učitelem, tedy klasifikaci, regresi a predikci. Tato metoda využívá jak označených dat, tak i dat neoznačených pro natrénování, kde počet označených dat je ale

malý. Malý počet označených dat je například z důvodu složitého a nákladného vyhodnocování a následném značení. [2]

1.2.4 Učení s posilováním

Algoritmy využívající tohoto učení pracují na principu pokus omyl, kde se zjišťuje, která akce přinesla nejlepší výsledek. Často se této techniky využívá v robotice nebo navigaci. Cílem tedy je, aby agent, což je učící prvek, vybíral takové akce, aby se maximalizoval výsledek. Pokud bude agent vybírat správné akce, dosáhne cíle v mnohem kratším čase, než při výběru špatných akcí. [2, 4]

1.3 Vybrané učící algoritmy

V této kapitole je popsáno několik vybraných algoritmů, které jsou využívány strojovým učením.

1.3.1 K-NearestNeighbors (KNN)

KNN je velmi jednoduchý klasifikátor, který patří do skupiny algoritmů učení s učitelem. Algoritmus klasifikuje nové vstupní data podle počtu nejbližších již klasifikovaných jedinců. KNN je nejvhodnější použít v případě, kdy má problém skupiny tříd v jednotlivých dimenzích snadno oddělitelné. Lze jej také využít i v případech složitějších, s ale menší úspěšností. [12]

Množství jedinců „K“, se kterými je nový porovnáván, může být libovolné, ale musí být liché. Hodnota „K“ se může nastavit i na jednoho jedince. V tomto případě je novému jedinci následně přiřazena třída nejbližšího souseda. V případě, že se „K“ nastaví příliš vysoké, bude nový jedinec klasifikován jako třída s největším zastoupením v datasetu. [12]

1.3.2 Naivní Bayesovský klasifikátor

Naivní Bayesovský klasifikátor je algoritmus patřící do skupiny algoritmů učení s učitelem. Pracuje na principu pravděpodobnosti příslušnosti jednotlivých atributů do klasifikační třídy. Předpokládá se, že jsou všechny atributy jedince na sobě nezávislé, čehož v reálném případě nelze docílit. Z tohoto důvodu se algoritmus označuje naivní. Nový jedinec je tedy klasifikován podle pravděpodobnosti příslušnosti jednotlivých atributů do klasifikačních tříd. Výhodou tohoto algoritmu je i schopnost pracovat s neúplnými daty z důvodu předpokládané pravděpodobnostní nezávislosti jednotlivých atributů. Pokud jsou chybějící

části náhodné, přesnost řešení se blíží hraniční pravděpodobnosti. V případě že chybějící části nejsou náhodné, je třeba zvolit jiný způsob vytvoření modelu. [13, 14]

1.3.3 K-Means

Algoritmus K-Means patří do skupiny algoritmů učení bez učitele a je používaný pro clustering. Protože se jedná o algoritmus učení bez učitele, vytváří clustery bez zásahu uživatele. Počet clusterů, které mají být vytvořeny, specifikuje uživatel před spuštěním algoritmu. [15, 16]

Algoritmus využívá takzvaných centroidů. Jedná se o K bodů, které indikují střed clusteru. V prvním kroku algoritmu jsou centroidy vybrány například náhodně ze vstupních dat. Clustery se následně tvoří kolem zvolených, nebo v dalších krocích vypočítaných, centroidů. Jedinec se přiřazuje do clusteru, k jehož centroidu má nejmenší vzdálenost. V dalších krocích se centroidy vypočítají jako střední hodnota všech bodů náležících do clusteru. Algoritmus je dokončen, pokud žádný z centroidů nezmění svou pozici. [15, 16]

1.4 Computer vision

Computer vision, neboli počítačové vidění, je jedna z oblastí umělé inteligence. Jeho cílem je interpretace lidského vidění pomocí počítače. Pro dosažení cíle je potřeba několika činností. Nejprve jsou získány prostředky pro zpracování, tedy obrazová data. Tato mohou být jak statické fotky, tak i nahrané video. Dále jsou aplikovány algoritmy pro zpracování obrazu z důvodu usnadnění práce s daty. Na závěr jsou aplikovány algoritmy pro analýzu obrazu a rozhodování, kde se již využívá učících algoritmů pro získání požadovaných informací. [17]

Důležitou součástí počítačového vidění je zpracování obrazu. Jedná se o aplikaci matematických funkcí a transformací nad zdrojovým obrazem. Příkladem těchto transformací a funkcí je zvýšení ostrosti, vyhlazování, detekce hran apod. Zpravidla se tedy jedná o aplikování konvolučních matic na celý obraz. [17, 18]

2 SLUŽBY POSKYTUJÍCÍ POČÍTAČOVÉ VIDĚNÍ

Pro vytvoření klasifikačního modelu lze zvolit několik různých postupů. Vedle možnosti natrénování vlastního modelu zcela od začátku, lze zvolit i možnost využít již natrénované modely. Tato část práce se zabývá službami poskytující natrénované modely k využití pro počítačové vidění.

2.1 Google Cloud Vision

Google Cloud Vision je součástí platformy Google Cloud Platform, která poskytuje řadu služeb, mezi které patří například cloudové úložiště, databázové služby a různé další možnosti. Služba Google Cloud Vision se konkrétně zabývá strojovým viděním. [19]

Cloud Vision poskytuje již předučenou síť přístupnou pomocí REST API, a také umožňuje vytvářet vlastní modely pomocí modulu AutoML Vision. API rychle klasifikuje vstupní obrázky na různé kategorie, je schopné rozpoznat jednotlivé objekty nebo tváře a disponuje schopností přečíst tištěný text. Pomocí těchto možností lze například vytvořit filtry na nevhodný obsah nebo automaticky vytvářet popis fotek v katalogu. [19]

Kromě kategorizování objektů v obrázku je API schopno rozpoznat známé krajinné body. Dále poskytuje OCR s velkou škálou jazyků s možností automatického rozpoznání jazyka. OCR pracuje jak s běžnými obrázky, tak i s PDF soubory. Lze také využít pro rozpoznání ručně psaného textu. V rámci kategorizování je možné zjistit i počet a umístění daného objektu v obrázku. API také umožňuje získat obecné atributy obrázků, jako například zastoupení barev nebo jak vhodně obrázek oříznout. [19]

2.1.1 AutoML Vision

AutoML Vision umožňuje snadné naučení vlastních modelů, které po nahrání a označení obrázků mohou být škálovány dle potřeby. Tato možnost lze využít, pokud je potřeba klasifikovat obrázky podle kategorií, které nejsou dostupné v Cloud Vision. Učení probíhá metodou učení s učitelem, kde musí být vstupním obrázkům přiřazeny kategorie, které má systém určovat. [19, 20]

Jako nezbytné minimum pro naučení modelu je třeba mít alespoň 100 obrázků na každou kategorii. Pro správné naučení je ale nejvhodnější mít 1000 a více obrázků. Dále je potřeba, aby bylo rozložení počtu obrázků na jednotlivé kategorie rovnoměrné. Rovnoměrné rozložení by mělo zůstat zachováno, i když je v datasetu větší množství pro jednu katego-

rii. Pokud se nedá zaručit rovnoměrné rozložení dat v datasetu, lze použít dataset tak, že nejméně četná kategorie bude mít minimálně 10% počtu nejčetnější kategorie. Při nespecifikování rozdělení datasetu na trénovací, validační a testovací množinu, bude dataset rozdělen na 80% trénovací, 10% validační a 10% testovací množinu. [20]

Pokud jsou v datasetu k dispozici pouze obrázky bez kategorie, lze využít službu ruční kategorizace. Ruční kategorizaci provádí tým lidí na základě zákazníkových požadavků. Díky této službě se získají testovací data použitelné pro naučení vlastního modelu. [19]

2.1.2 Ceník služeb

Služba Cloud Vision umožňuje platit pouze za potřebné funkce, které jsou každá ceněna zvlášť. Může být tedy účtována pouze jedna funkce nebo kombinace různých funkcí. Jednotky se uvádí jako jedno provedení funkce nad obrázkem. Provedení dvou funkcí nad jedním obrázkem se tedy počítá jako dvě jednotky. V rámci vícestránkových dokumentů se každá strana počítá jako jeden samostatný obrázek. [21]

Tabulka 1 Ceník služby Cloud Vision [21]

	Cena za 1000 jednotek měsíčně		
Funkce	1 – 1 000 jednotek/měsíc	1 001 – 5 000 000 jednotek/měsíc	5 000 001 – 20 000 000 jednotek/měsíc
Kategorizace	Zdarma	$1.50	$1.00
Detekce textu	Zdarma	$1.50	$0.60
Detekce vhodného obsahu	Zdarma	Zdarma s kategorizací, nebo $1.50	Zdarma s kategorizací, nebo $0.60
Detekce obličeje	Zdarma	$1.50	$0.60
Detekce krajinných prvků	Zdarma	$1.50	$0.60
Detekce loga	Zdarma	$1.50	$0.60
Vlastnosti obrázku	Zdarma	$1.50	$0.60
Tipy na ořezání	Zdarma	Zdarma s vlastnostmi obrázku, nebo$1.50	Zdarma s vlastnostmi obrázku, nebo $0.60

Detekce webu	Zdarma	$3.50	Po domluvě
Detekce dokumentového textu	Zdarma	$1.50	$0.60
Lokalizace objektů	Zdarma	$2.25	$1.50

Služba AutoML Vision je zpoplatněna dvojím způsobem. První je platba za učení modelu a druhá je platba za predikci z již naučeného modelu. Cena za učení se uvádí za výpočetní hodiny, jež jsou určeny čistým časem výpočtu, a ne tedy reálným časem. První hodina na 10 různých modelech každý měsíc je zdarma. Každá další započatá hodina učení je $20. Pokud se pokračuje v učení na již naučeném modelu, cena je $20 / hod.[22]

Tabulka 2 Ceník služby ruční kategorizace [22]

Služba	Počet hodnotících	Cena
Zdarma (prvních 100 obrázků měsíčně)	3	Zdarma
Základní	1	$35 / 1 000 obrázků
Základní	3	$100 / 1 000 obrázků
Prémiová	Po domluvě	

Tabulka 3 Ceník klasifikace naučeného vlastního modelu [22]

	0 - 1 000 obrázků/měsíc	1 001 - 5 000 000 obrázků/měsíc	Více než 5 000 000 obrázků/měsíc
Klasifikace obrázků	Zdarma	$3 / 1 000 obrázků	Po domluvě

2.2 Microsoft Azure – Cognitive services

Microsoft Azure je cloudová platforma poskytující řadu služeb, mezi které například patří úložiště dat, databázové nástroje, umělá inteligence se strojovým učením, a jiné. Každá poskytovaná služba je účtována zvlášť, takže uživatel platí pouze za to, co potřebuje. [23]

Cognitive services je jedním z produktů poskytovaných platformou Microsoft Azure. Produkt samotný poskytuje řadu různých služeb jako například klasifikaci obrázků, převod

mluveného slova na text a zpět, analýzu textu, automatický překladač, a další. Služby jsou seskupeny podle principu jejich funkcionality. [24]

2.2.1 Computer Vision

Služba strojového vidění umožňující po nahrání obrázku nebo poskytnutí URL adresy obrázku jeho analýzu. Obsah analýzy obrázku odvisí od požadavku uživatele jako například detekce obličeje. Implementace služby může být formou nativního SDK nebo pomocí REST API. [25, 26]

Poskytováno je několik možností:

- Označení obrázků podle kategorií, kde API vrací i velmi stručný popis kontextu scény, a také i kategorizaci všech předmětů na obrázku. [26]
- Detekci objektů, která je podobná označování obrázků, ale s tím rozdílem, že u této možnosti se v obrázku označuje pomocí souřadnic detekovaný objekt. [26]
- Detekce značek na základě již existující databáze. [26]
- Kategorizace obrázku jako celku pomocí struktury realizovanou formou rodič/potomek. [26]
- Popis obrázku pomocí vytvořené věty na základě objektů na obrázku. [26]
- Detekce obličejů. [26]
- Detekce parametrů obrázku. [26]
- Vytvoření náhledu obrázku na základě analyzované oblasti zájmu na obrázku. [26]
- Analýza oblasti zájmu realizovaná pomocí souřadnic. [26]

Pro analýzu nahraného obrázku musí obrázek splňovat několik požadavků. Musí být ve formátu JPEG, PNG, GIF nebo BMP, velikost souboru musí být menší než 4 MB a rozměry obrázku musí být větší než 50 x 50 pixelů. [26]

2.2.2 Ceník

Tabulka 4 Ceník služby Computer Vision – Microsoft Azure [27]

<table>
<tr><th></th><th colspan="3">Cena za každých 1 000 transakcí</th></tr>
<tr><th>Funkce</th><th>0 – 1 000 000 transakcí</th><th>1 000 000 – 5 000 000 transakcí</th><th>5 000 000 a více transakcí</th></tr>
<tr><td>Kategorizace</td><td rowspan="6">€0.844</td><td rowspan="6">€0.675</td><td rowspan="6">€0.549</td></tr>
<tr><td>Detekce tváří</td></tr>
<tr><td>Získání náhledu</td></tr>
<tr><td>Barva</td></tr>
<tr><td>Typ obrázku</td></tr>
<tr><td>Získat oblast zájmu</td></tr>
<tr><td>OCR</td><td rowspan="6">€1.265</td><td rowspan="6">€0.844</td><td rowspan="6">€0.549</td></tr>
<tr><td>Detekce obsahu pro dospělé</td></tr>
<tr><td>Celebrity</td></tr>
<tr><td>Krajinné prvky</td></tr>
<tr><td>Detekce objektů</td></tr>
<tr><td>Detekce značek</td></tr>
<tr><td>Rozpoznání textu</td><td colspan="3">€2.109</td></tr>
</table>

2.3 Amazon Rekognition

Amazon Rekognition poskytuje služby počítačového vidění aplikované jak na video, tak i na obrázky. Amazon Rekognition Video slouží k analýze videa a Amazon Rekognition Image k analýze obrázků. [28]

Amazon Rekognition Image poskytuje API pro detekci objektů, krajiny a obličejů. Dále umožňuje získání textu z obrazu, rozpoznávání celebrit. Je schopné rozlišit i nevhodný obsah obrázků. Všechny tyto akce lze provést na libovolném obrázku. Tento může být umístěn na cloudovém úložišti Amazonu nebo v lokálním úložišti uživatele. [29, 30]

Amazon Rekognition Video poskytuje funkce pro detekci aktivit, pohybu osob, rozpoznání osob a objektů. Dále lze stejně jako u obrázků rozpoznávat nevhodný obsah. Analýzu videa lze provést pouze na videa uložená v cloudovém úložišti Amazon S3, nebo videa streamovaná pomocí platformy Amazon Kinesis Video Streams. [31]

2.3.1 Ceník

Ceny se dělí podle používaného produktu. Samostatně je ceněno zpracování videa a samostatně zpracování obrázků. Amazon Rekognition umožňuje pro nové uživatele využívat službu zdarma po dobu jednoho roku s omezením. Pro zpracování videa je toto omezení stanoveno na 1000 minut videozáznamu měsíčně. Pro zpracování obrazu je omezení stanoveno na 5000 obrázků měsíčně a 1000 metadat tváří. Cenění služby při překročení tohoto limitu nebo uplynutí doby jednoho roku závisí na regionu. [32]

Tabulka 5 Ceník služby Amazon Rekognition Image pro oblast EU [32]

Počet analyzovaných obrázků za měsíc	Cena na 1000 zpracovaných obrázků
První 1 milion	$1.00
Dalších 9 milionů	$0.80
Dalších 90 milionů	$0.60
Přes 100 milionů	$0.40

3 POUŽITÉ TECHNOLOGIE

V této kapitole jsou popsány technologie, které byly použity pro vytvoření praktické části této práce. Technologie byly zvoleny hlavně z důvodu, že se jedná o open-source technologie s přehlednou dokumentací a velkou komunitou uživatelů. Dalším kritériem výběru byla již existující aplikace, na kterou se výstup této práce napojuje, a proto musely být zvoleny kompatibilní technologie se zmíněnou aplikací.

3.1 PHP

PHP je open-source skriptovací jazyk převážně používaný pro vývoj webových aplikací a může být obsažen v HTML dokumentech. Vložením do HTML stránek lze dosáhnout dynamického obsahu, tedy lze je i vytvářet. Rozdílem oproti tvorbě stránek pomocí JavaScriptu je, že PHP příkazy se vykonávají na serveru. Na serveru se tak zhotoví kompletní HTML stránka, která se pošle klientovi. Na straně klienta tedy není žádný způsob, jak zjistit, jakým způsobem byla stránka vygenerována. [33]

Jsou tři oblasti, kde lze PHP využít. Nejpoužívanější je spouštění skriptů na serveru, kde je potřeba PHP parser, webový server a webový prohlížeč. Prohlížeč zde slouží k zobrazování výstupů z PHP skriptů. Další možnost je spouštět skript v příkazové řádce, tedy bez webového serveru nebo prohlížeče. Pro tento způsob je ale stále potřeba mít PHP parser. Poslední možností je tvorba desktopových aplikací. Tento způsob využití PHP ale není příliš vhodný. Pro využití v desktopových aplikacích je vhodné použít rozšíření jazyka PHP-GTK. [34]

PHP může být použito na všech hlavních operačních systémech, jako jsou Linux, Microsoft Windows, MacOS, atd. Dále podporuje většinu webových serverů, jako jsou Apache, IIS a další. PHP má také podporu velkého množství databázových systémů. Na tyto lze v rámci vývoje přistupovat například pomocí specifických rozšíření nebo pomocí PDO. [34]

3.2 Laravel

Laravel je open-source PHP framework s licencí MIT. Framework využívá návrhového vzoru MVC, tedy Model-View-Controller. Laravel značně usnadňuje spoustu činností, jako například směrování HTTP požadavků (routing), dependency injection, vytváření a udržování relací, snadnou práci s databází pomocí ORM atd. Velkou výhodou Laravelu je

jasně strukturovaná a rozsáhlá dokumentace, kde u většiny příkazů je i praktická ukázka kódu. [35, 36]

3.2.1 Composer

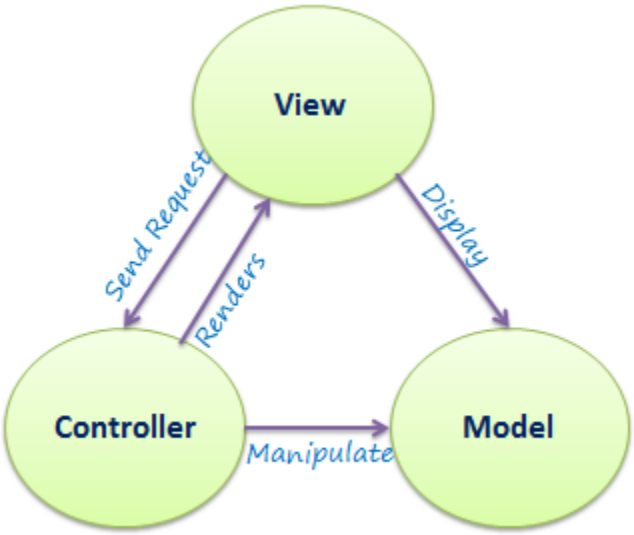
Výhodou Laravelu je také snadné přidávání nástrojů a knihoven třetích stran za pomoci nástroje Composer. Tento nástroj spravuje knihovny vždy v rámci jednoho projektu, kde se soubory požadovaného balíku instalují do složky „Vendor“. [37]

Nástroj využívá souboru „composer.json“, umístěném v kořenovém adresáři projektu, ve kterém jsou uváděny všechny závislosti. Soubor obsahuje klíč „require“, který je tvořen JSON objektem. V tomto objektu jsou následně namapovány jména balíků na jejich verze. Pomocí tohoto souboru Composer prohledá zvolené repozitáře nebo využije výchozí balíčkovací repozitář Packagist. Jména balíčků musí být v určitém formátu, aby se zajistila jednoznačná identifikace. Tento formát se skládá ze jména autora a názvu projektu, a je následující: jmeno1/projekt1. [38]

Composer také využívá souboru „composer.lock“, jehož účelem je uzamčení verze balíku nebo knihovny tak, aby se automaticky nestáhla nejnovější verze knihovny. Je to z důvodu, aby práci na projektu nepřerušila nějaká nečekaná aktualizace některého z balíků. Díky tomuto souboru tedy všichni pracující na projektu budou mít nainstalovanou jednu konkrétní verzi všech závislostí uvedených v souboru „composer.json“. [38]

3.3 MVC

MVC je návrhový vzor, který dělí aplikaci na tři části, a to model, view a controller. Jednotlivé části mezi sebou komunikují. Základní myšlenkou tohoto vzoru je oddělit uživatelské rozhraní od logiky programu a od dat. [39, 40]



Obrázek 2 MVC architektura [41]

3.3.1 Model

Modelem lze chápat data aplikace a logiku samotných dat, jako je získávání a ukládání dat do databáze. U malých aplikací lze modelem označit i samotný dataset, pokud má být cílem aplikace pouze zobrazování dat ve view komponentě. Co se týče komunikace, tak model nemůže nijak ovlivňovat controller nebo view, pouze oba informuje, pokud v něm nastala nějaká změna. Pro příklad lze jako model označit vzor pro jednotlivé úkoly aplikace úkolníčku. [39, 40]

3.3.2 View

View část slouží k prezentování aplikace uživateli, tedy jedná se o uživatelské rozhraní. Obvykle je view tvořeno pomocí modelů. V rámci komunikace, view komponenta nesmí přímo modifikovat model, tedy data. Jakákoliv akce uživatele tedy musí být odeslána do controlleru. [39, 40]

3.3.3 Controller

Tato část se stará o samotnou logiku programu. Zpracovávají akce uživatele, upravují model a starají se o zobrazení view pro uživatele. Controller přijímá odeslaná data, která odeslal view po zadání uživatele, například do formuláře. Přijatá data následně zpracuje požadovaným způsobem. Dále může uživatelský vstup posílat na model, kde bude parsován například na dotaz vložení do databáze [39, 40]

3.4 Verzovací systém

Verzovací systém slouží k zaznamenávání změn v souboru nebo změn souborového systému aplikace v čase. Díky těmto záznamům lze snadno zpětně zjišťovat provedené změny a hlavně lze vrátit aplikaci do předchozího stavu, pokud například některá ze změn zapříčinila špatnou funkčnost aplikace. Verzovací systém lze ale použít nejen na vývoj aplikací, ale i na libovolné soubory. Dalším příkladem jiným než vývoj software je grafický design nebo psaní knih. [42]

Existuje několik typů verzovacích systémů. Jedním je lokální verzovací systém, který vytváří verzovací databázi pouze na lokálním počítači. Tento typ slouží spíše jako náhrada za „kopírovací zálohu“, tedy že se jednotlivé verze aplikace zkopírují vždy do vedlejší složky. Výhodou lokálního verzovacího systému oproti kopírování je hlavně lepší přehlednost. [42]

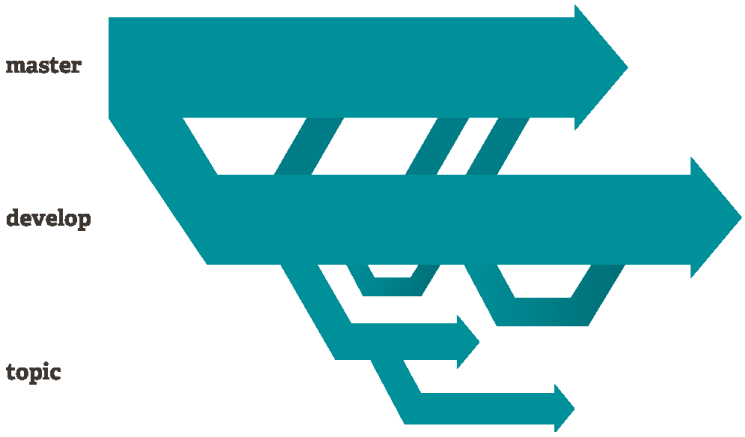
Dalším typem je centralizovaný verzovací systém, který je tvořen jednou centralizovanou databází verzí, ke které může přistupovat libovolný klient. Oproti lokální verzi je tato lepší pro práci v týmu, kde každý člen může pracovat na stejné verzi. Dále je také globálně lépe spravovatelná díky centralizovanému principu. Nevýhoda tohoto typu ale je, pokud by nastala závada na verzovacím serveru. V tomto případě po dobu výpadku nemůže žádný klient vytvářet další záznamy. Dále je zde nebezpečí ztráty projektu bez řádné zálohy stejně jako u lokálního verzovacího systému. [42]

Řešením nevýhod centralizovaného typu je použití distribuovaného verzovacího systému, kde každý klient má u sebe celou funkční kopii serverové verzovací databáze. Díky tomu lze serverovou databázi jednoduše zpětně zprovoznit od libovolného klienta. Na tomto principu pracuje například Git. [42]

3.4.1 Git

Git je open source distribuovaný verzovací systém pod licencí GNU GPL verze 2. První verzi Gitu napsal Linus Torvalds. [43, 44, 45]

Git má řadu vlastností, kde nejvýraznější je větvení projektu. Projekt se člení na několik místních větví, které jsou na sobě zcela nezávislé. Větve mohou být snadno sloučeny do sebe. Typicky se vytváří nová větev, pokud je potřeba zásadním způsobem změnit samotný projekt, při experimentování s různými řešeními nebo se rozděluje produkční větev od větve vývojové. [42, 46]

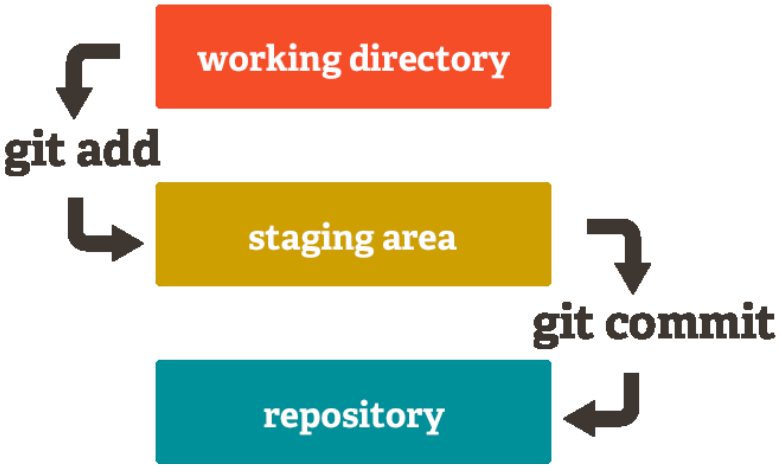


Obrázek 3 Ukázka větvení [47]

Protože je Git distribuovaný verzovací systém, tak lze pomocí větvení využít libovolného pracovního postupu při vývoji. Lze například použít centralizovaný přístup, kde všichni uživatelé commitují změny pouze do jedné větve. Git má tento způsob ošetřen tak, že uživatel nemůže poslat commit na server, pokud mezitím nahrál jiný uživatel nějaké změny. Dalším možným postupem je, že na server může změny nahrávat pouze jedna osoba. Uživatelé, kteří provedou změny, následně zažádají správce o nahrání jejich změn na server. [42, 48]

Pro zajištění integrity dat jsou všechny soubory a commity označeny kontrolním součtem. Každá další nahraná změna pak vychází z předchozích kontrolních součtů. Díky tomu nelze v historii nic změnit bez toho, aby se změnily následující části. [42, 49]

Git obsahuje přípravnou oblast pro změny nazývanou „staging area“. Přípravná oblast je určena k tomu, aby se na server nenahrávaly všechny změněné soubory, ale pouze ty, které uživatel sám zvolí. Je mezivrstvou mezi pracovní oblastí a repozitářem. [42, 50]

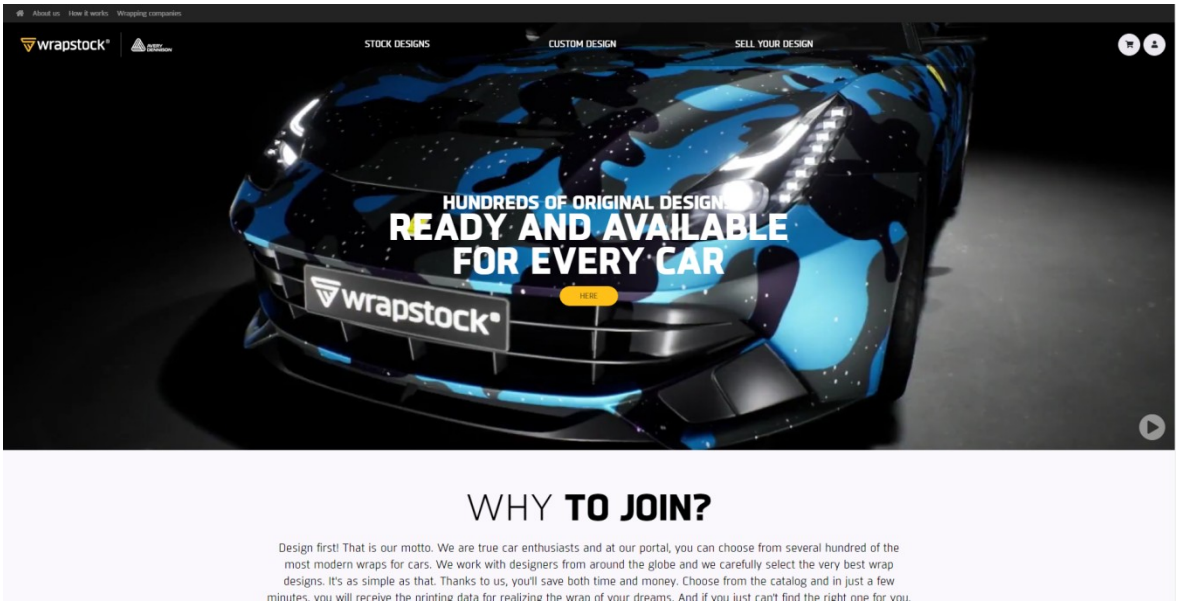


Obrázek 4 Zobrazení abstraktních vrstev Gitu [51]

II. PRAKTICKÁ ČÁST

4 WRAPSTOCK APLIKACE

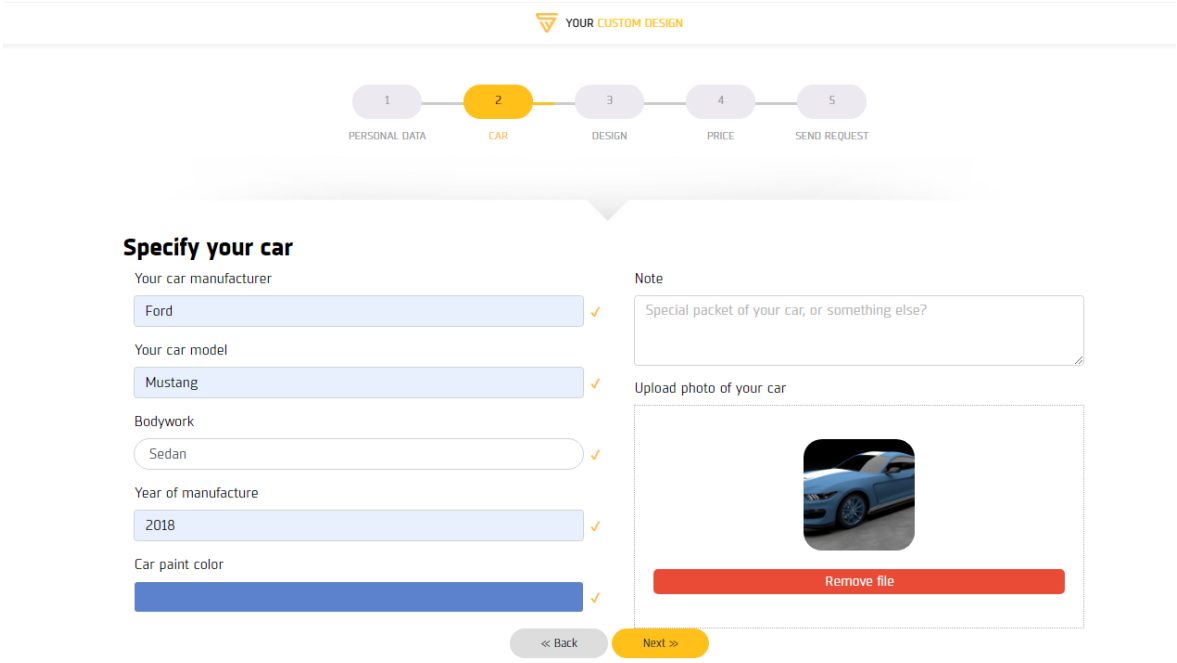
Wrapstock je internetový obchod zabývající se prodejem polepů na vozidla tvořené registrovanými designéry. Obchod slouží právě jako hub pro všechny designéry, kteří chtějí své designy zpeněžit.



Obrázek 5 Úvodní stránka Wrapstocku

Zákazník může v aplikaci využít obchod s již hotovými nabízenými designy. V nabídce jsou zveřejněny jak všechny schválené polepy externích registrovaných designérů, tak i polepy navrhnuté interními designéry. Tato sekce obchodu se dělí na dvě části, a to část se specifickými designy na konkrétní automobily, a také univerzální designy, které lze aplikovat a upravit na libovolný typ vozu.

Další funkcí aplikace je zažádání o nový design na míru dle požadavků. Prvním krokem je, pokud uživatel není zaregistrovaný, uživatelova registrace. Po registraci nebo přihlášení se musí nejprve specifikovat výrobce, model, typ karosérie, rok výroby a barva vozidla, na které má být design vytvořen. Pro snadnější rozpoznání vozidla a barvy lze vložit také fotku automobilu. V dalším kroku se specifikuje poptávaný design. Určuje se zde kategorie, specifické značky a barvy designu. Lze také označit oblíbené designy, kterými se má designér při návrhu inspirovat. V závěrečném kroku před samotnou platbou se zvolí druh polepu podle plochy a tvaru požadovaného designu.



Obrázek 6 Ukázka formuláře pro požadavek na vlastní design

Pro nahrání nového designu designér využije další část aplikace. Nejprve se nahraje obrázek s náhledem designu a zdrojové soubory designu. V dalším kroku se zadá název nového designu, jeho popis a specifikuje se, pro které vozidlo je design určen. Na závěr designér zvolí druh polepu podle jedné z uvedených kategorií a jeho cenu. Jakmile jsou všechny kroky splněny, odesílá se požadavek na zahrnutí designu do nabídky.

5 ZADÁNÍ A NÁVRH ŘEŠENÍ

Tato kapitola diplomové práce se bude zabývat samotným návrhem řešení a zadání z pohledu vytvářené aplikace. Jsou zde rozebrány jednotlivé požadavky na úspěšnou implementaci do aplikace Wrapstock.

5.1 Zadání

Cílem implementace a využití Google Cloud Vision API je usnadnění a částečná automatizace vyplňování formulářových polí pro zákazníky v aplikaci Wrapstock. Konkrétně se jedná o proces vytváření nového designu určeného k prodeji, a také o proces vytváření požadavku na vlastní unikátní design.

V obou uvedených případech je povinnou součástí formuláře uvést výrobce a model daného vozidla, jeho karosérii a rok výroby. Dále se specifikuje i barva vozidla v případě požadavku na vlastní design, nebo v případě nahrávání nového designu jeho převládající barvy. Součástí formulářů je i nahrání obrázku vozidla, respektive designu. Služby počítačového vidění lze uplatnit v obou případech tak, že se na základě nahraných obrázků klasifikuje právě výrobce a model vozidla, jeho barva a případně i karosérie. Vzhledem k tomu, že jsou designy klasifikovány podle druhu na specifické kategorie, nebudou tyto klasifikační třídy zahrnuty v naučených modelech služeb, které poskytují možnost klasifikace obrázků. Z důvodu, že si jsou designy v těchto kategoriích značně podobné, lze zde využít možnosti natrénování vlastního klasifikačního modelu, který bude mít za úkol klasifikovat designy pouze na tyto specifické kategorie.

5.2 Požadavky

Jednotlivé požadavky na výslednou implementaci funkcí některé ze služeb poskytujících klasifikaci obrázků jsou uvedeny v této kapitole jako funkční a nefunkční požadavky. Požadavky jsou sestaveny na základě zadání a technologií, které jsou kompatibilní, případně stejné, s těmi, na kterých je aplikace Wrapstock vystavěna.

Funkční požadavky, tedy ty, které označují poskytovanou funkcionalitu implementovaného řešení:

- Odesílání požadavku na API
- Zpracování odezvy z API – výrobce, model, karosérie a barva vozidla
- Extrahovaní požadovaných hodnot

- Vrácení hodnot uzpůsobených pro formulář
- Výpočet vzdálenosti barev
- Přiřazení libovolné barvy podle vzdálenosti k barvě v databázi
- Klasifikace podle vlastních značek – full wrap, stripes, partial

Dalšími požadavky jsou nefunkční požadavky, tedy ty, které určují omezení a technologie pro implementaci řešení:

- Jazyk PHP ve verzi 7.1.7
- Framework Laravel ve verzi 5.4
- Dotazy na API čistě v serverové časti aplikace
- Zachování MVC architektury
- Použití stávající databáze bez výrazných změn
- Použití vhodné cloudové služby poskytující počítačové vidění
- Možnost konfigurace parametrů

5.3 Rešerše služeb počítačového vidění

Součástí této práce je i prozkoumání alternativních řešení ke službě Google Cloud Vision. Vyhledávány byly pouze služby, které poskytují možnost klasifikace obrázků na kategorie, detekci barev a případně umožňují vytvořit vlastní klasifikační model. Kritériem pro výběr je také cena a i úroveň zpracování dokumentace.

Služby, které poskytují klasifikaci obrázků, byly nalezeny a zvoleny tři, včetně služby Google Cloud Vision. Nalezené služby jsou popsány v kapitole 2 této práce, a jsou to tedy Google Cloud Vision, Microsoft Azure – Cognitive services a Amazon Rekognition.

Všechny výše uvedené služby poskytují klasifikaci obrázků na kategorie, ačkoliv pouze Google poskytuje možnost natrénování vlastního klasifikátoru za pomoci AutoML Vision. Na rozdíl od služby na platformě Microsoft Azure, poskytují zbylé dvě služby výhodné platební podmínky, a také obě služby poskytují klasifikaci obrázků zdarma do určitého počtu měsíčně. Dalším rozdílem mezi těmito službami je možnost naučení vlastního klasifikátoru. Touto možností disponují služby na platformách Google Cloud a Microsoft Azure. Služba Amazon Rekognition tuto možnost neposkytuje. Z pohledu dokumentace jsou nejlépe zdokumentovány služby Google Cloud Vision a Microsoft Azure – Cognitive services. Na základě této rešerše byla zvolena služba Google Cloud Vision pro implementaci do této práce, která se jeví jako ideální volba pro potřeby tohoto projektu.

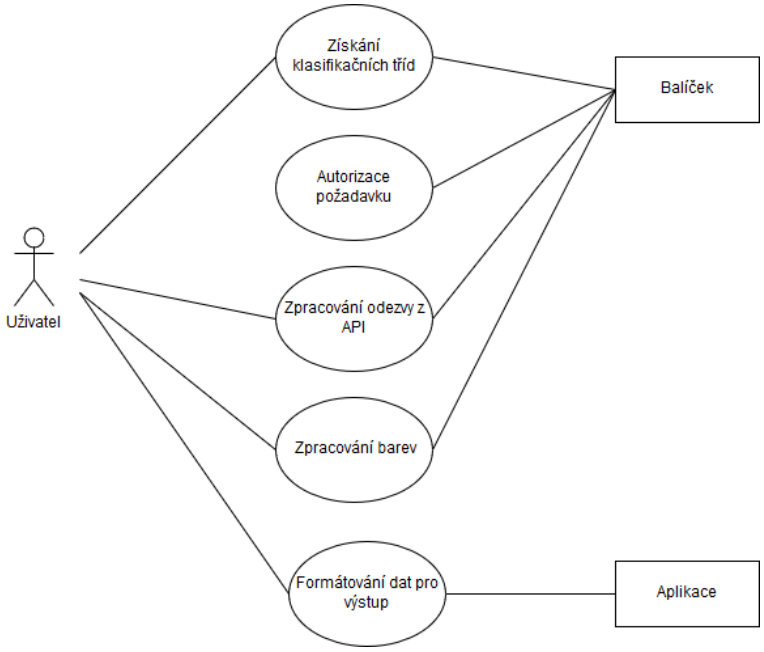
5.4 Návrh implementace

Návrh implementace vychází z analyzovaných požadavků a jedná se o konkrétní návrh řešení zadaného problému. Tento návrh je relativně nezávislý na zvolené službě za předpokladu, že služba poskytuje kromě klasifikace objektů i výčet převažujících barev. Jediná změna, která by v návrhu musela být změněna při volbě jiné služby, je autorizování dotazů na API.

Protože jedním z požadavků je, aby výsledné řešení bylo tvořeno pro framework Laravel, byla zvolena implementace formou instalovatelného balíčku pro tento framework, který umožňuje snadnou škálovatelnost pomocí různých modulů a s tím i spojenou jejich tvorbu. Díky tomu, že zadaný problém bude řešen právě pomocí balíčku, se dosáhne velmi snadného zprovoznění v případných dalších projektech zabývajících se stejnou nebo podobnou tématikou.

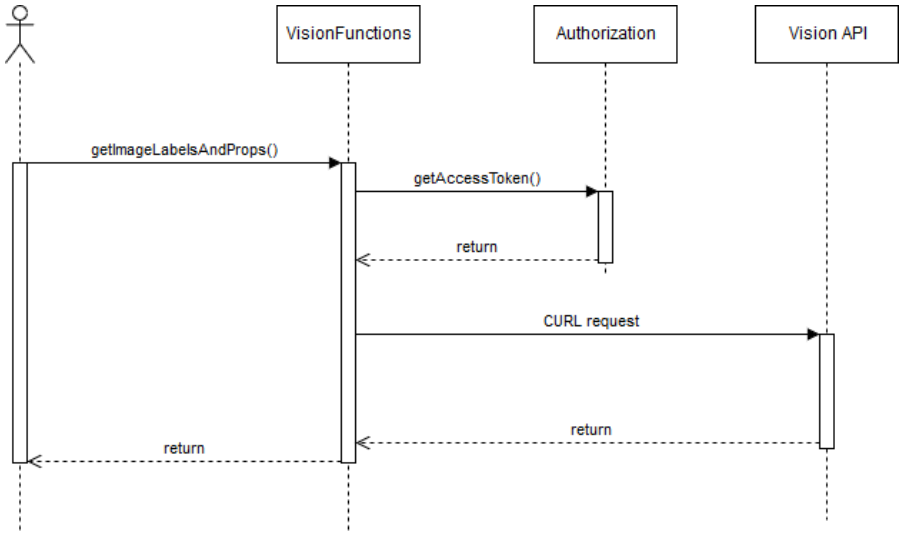
5.4.1 Případ užití

Typickým případem užití, který lze vidět na obrázku č. 7 je použití balíčku se serverovou aplikací. Hlavním aktérem je zde uživatel používající funkce balíčku a samotné aplikace. Dalšími aktéry jsou na opačné straně vytvořený balíček a aplikace implementující balíček do celkového řešení. Balíček musí umožňovat uživateli získat klasifikační třídy a barvy. Balíček musí sám řešit autorizaci požadavků na API bez zásahu uživatele. Dále musí poskytovat funkce pro zpracování odezvy z API tak, aby byly získány zjištěné modely a výrobci vozidel, jejich karosérie a barvy, například formou pole. Další uživatelem použitelná funkce balíčku je zpracování získaných barev tak, aby se získala reprezentace nejbližší barvy uložené v databázi. V rámci uvedeného případu užití je i formátování a poskytnutí dat zpracovaných balíčkem do aplikace implementující celé řešení.



Obrázek 7 Diagram případu užití

Získání klasifikačních tříd a jejich zpracování je rozděleno do dvou samostatných funkcí. Tímto je uživateli balíčku umožněno získat čistou odpověď z API a zpracovat ji zcela dle svého uvážení. Během tohoto kroku se využívá autorizační třída, protože je součástí funkce volání API.

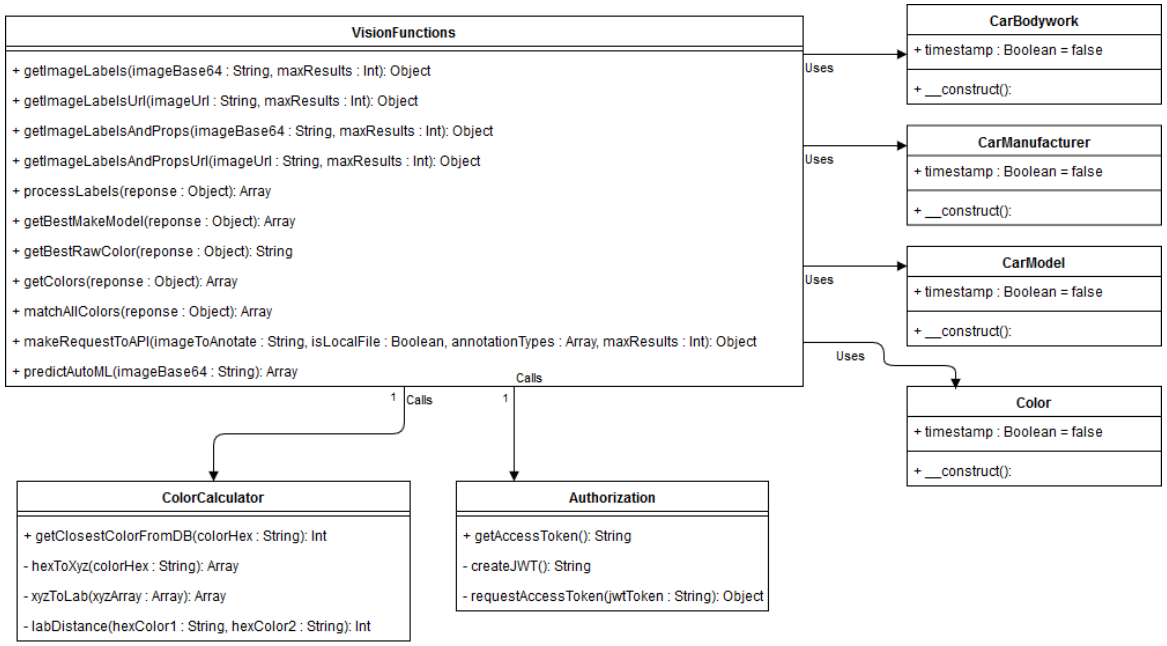


Obrázek 8 Sekvenční diagram – volání Google Vision API

5.4.2 Návrh struktury balíčku a databáze

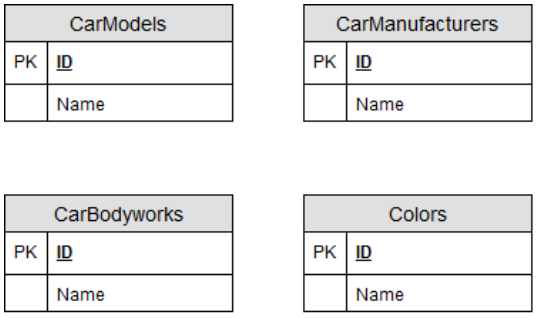
Na obrázku č. 9 je uveden diagram tříd vytvářeného balíčku. Je zde patrná hlavní třída VisionFunctions obsahující hlavní funkce pro komunikaci s API. Dále jsou v diagramu vidět dvě pomocné třídy ColorCalculator a Authorization. Třída ColorCalculator slouží

k výpočtu vzdáleností a následnému přiřazení zadané barvy k barvě z databáze. Třída Authorization slouží pro tvorbu autorizačních tokenů potřebných k volání API funkcí. Další částí v diagramu jsou 4 Modely sloužící pro reprezentaci databázových dat. Jsou to CarBodywork, CarManufacturer, CarModel a Color.



Obrázek 9 Diagram tříd

Ve vytvářeném balíčku jsou použity celkem čtyři tabulky obsahující data potřebná pro zpracování odezvy z API. Protože se musí porovnávat názvy jednotlivých získaných klasifikačních tříd s daty v těchto tabulkách, je struktura databáze značně zjednodušena, to znamená, že nejsou vyžadovány žádné vazby mezi tabulkami. Jedním z požadavků je použití existující databáze aplikace. Proto jsou názvy tabulek a sloupců nastavovány v konfiguračním souboru a na následujícím schématu je uvedena pouze orientační struktura databáze. Díky konfiguračnímu souboru je dalším možným, avšak zcela nevhodným řešením, je použití jedné tabulky obsahující čtyři sloupce s potřebnými daty.



Obrázek 10 Orientační struktura databáze

6 IMPLEMENTACE ŘEŠENÍ

Tato kapitola se věnuje implementaci navrhnutého řešení, které bylo rozebráno v předcházející kapitole. Před popisem samotné implementace je zde popsána obecná tvorba balíčku pro Laravel.

6.1 Tvorba balíčku pro Laravel

Framework Laravel umožňuje díky své modularitě nejen snadnou integraci různých knihoven, ale i možnost tvorby vlastní knihovny. Pro úspěšnou tvorbu vlastního balíčku je zapotřebí absolvovat několik kroků. Před zahájením vývoje je vhodné využít existujícího nebo nově vytvořeného projektu v Laravelu pro rychlé otestování funkčnosti vyvíjené knihovny a usnadnění její tvorby. Usnadnění spočívá ve využití nástroje Artisan.

Prvním krokem je navrhnout adresářovou strukturu balíčku. Ta vyžaduje hlavní adresář, který je pojmenován například podle autora. Pojmenován je tak z důvodu seskupení několika knihoven od jednoho autora. Hlavní adresář musí obsahovat podadresář, který svým názvem indikuje název celého balíčku. Dále v textu bude tento adresář pro zjednodušení označován jako kořenový. Následujícím krokem je vytvořit v kořenovém adresáři soubor *composer.json*. V tomto souboru jsou uvedeny všechny potřebné údaje o balíčku. Díky tomuto souboru lze balíček nainstalovat pomocí nástroje Composer. Soubor lze vytvořit ručně s použitím struktury uvedené níže, nebo využít příkazu *composer init* v kořenovém adresáři balíčku. Po zavolání uvedeného příkazu se spustí průvodce vytváření požadovaného souboru. Pro vytvoření souboru následně stačí zodpovědět na otázky položené průvodcem. V souboru se uvádí název balíčku, jeho popis, aktuální verze, požadované nainstalované balíčky a cesta ke složce sloužící pro automatické zavedení balíčku do projektu. Ukázková struktura souboru *composer.json* je následující:

```
{
  "name": "autor/nazevbalicku",
  "description": "Ukazkova struktura souboru composer.json",
  "minimum-stability": "dev",
  "require": {
      "laravel/framework": "5.4.*",
      "laravel/tinker": "~1.0",
      "laravelcollective/html": "^5.4"
  },
  "autoload": {
      "psr-4": {
          "\\autor\\nazevbalicku\\": "src/"
      }
  }
}
```

Dalším krokem je vytvoření adresářů a složek souvisejících s funkcionalitou samotného balíčku. Vytvoří se třídy s funkcemi, modely, kontroléry, pohledy, atd. Pokud jsou součástí balíčku i pohledy, konfigurační soubory,… které se musí publikovat do aplikace, tedy mají být nakopírovány do příslušných složek, musí být vytvořena i třída dědící od třídy *ServiceProvider*. S její pomocí budou zavedeny všechny potřebné části na určená místa v aplikaci po zavolání příkazu *php artisan vendor:publish*. Tuto třídu lze vytvořit stejně jako soubor *composer.json* ručně nebo s použitím příkazu *php artisan make:provider nazevServiceProvider*. Pokud se vytváří soubor ručně, musí mít následující strukturu:

```
<?php

namespace autor\nazevbalicku;

use Illuminate\Support\ServiceProvider;

class nazevServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap the application services.
     *
     * @return void
     */
    public function boot()
    {
      // Implement
    }

    /**
     * Register the application services.
     *
     * @return void
     */
    public function register()
    {
      // Implement
    }
}
```

V případě že balíček bude obsahovat nějaký ServiceProvider, musí být každý po instalaci balíčku zaveden do konfiguračního souboru */config/app.php*, ve kterém jsou zaregistrovány všechny ServiceProvider třídy z různých nainstalovaných balíčků.

6.2 Datový tok mezi balíčkem a Google Vision API

Před samotnou tvorbou balíčku je důležité nejprve porozumět datovým tokům mezi vytvářeným balíčkem a veřejným Google Vision API, které je pro balíček stěžejní. Vzhledem k tomu, že se jedná o server-side balíček, který bude vyžadovat odpověď z Google Vision API pro další funkcionalitu, využijí se pro komunikaci s API synchronní CURL požadavky. Pro zjednodušení tvorby HTTP požadavků a použití CURL je využit balíček Guzzle,

který požadované funkce značně ulehčuje. Navíc slouží jako abstraktní vrstva nad HTTP komunikací, což znamená, že nemusí nutně být využit CURL, ale je možné využít například stream atd.

6.2.1 Tvorba požadavku pomocí Guzzle

Obecně se požadavky s využitím Guzzle sestavují tak, že nejprve vytvoří instance klienta. Dále je zavolána její funkce pro vytvoření požadavku, která bere jako parametr metodu požadavku (GET, POST, …), cílovou URL adresu a pole s nastavením požadavku, jako například hlavička nebo tělo požadavku. Funkce po odeslání požadavku a vyřízení na vzdáleném serveru vrací právě odpověď z daného serveru. Je vhodné tuto funkci zapouzdřit do klauzulí try-catch. Je totiž možné, že se vyskytne chyba při samotném spojení nebo přímo na vzdáleném serveru. Pokud chyba nastane, může se využít například logování nebo jakýkoliv jiný úkon. Pro lepší přehlednost je vzorová struktura kódu sloužící k vytvoření požadavku uvedena níže. Je v ní uveden i příklad vnořených objektů v těle požadavku.

```
$client = new Client();
try {

    $resp = $client->request('POST', 'https://automl.googleapis.com/v1beta1/'.$parent.'/datasets',[
        'headers' => [
            'content-type' => 'application/json',
            'Authorization' => 'Bearer '.$accessToken
        ],
        'json' => [
            'displayName' => $datasetName,
            'imageClassificationDatasetMetadata' => [
                'classificationType' => 'MULTICLASS'
            ]
        ]
    ]);

} catch (\Exception $ex) {
    \Log::debug('error');
    \Log::debug((string) $ex->getResponse()->getBody());
    return (string) $ex->getResponse()->getBody();
}
```

6.2.2 Požadavky směřující na Google Cloud API

V balíčku se v rámci použití Google Cloud Vision API využívá vytvoření několika požadavků na Google Cloud platformu. Pro volání kterékoliv služby, které poskytuje Google Cloud Platform, je třeba každý požadavek autorizovat. Autorizace je možná u některých služeb pomocí API klíče získaného z platformy, nebo využitím OAuth2 autorizace pomocí takzvaných *access tokenů*. Protože autorizace pomocí OAuth2 poskytuje lépe spravovatelný přístup k API, byla zvolena tato možnost. Access token se získá zasláním POST poža-

davku na adresu *https://www.googleapis.com/oauth2/v4/token*. Struktura požadavku vytvářeného v kódu balíčku je uvedena níže. Protože se jedná o url-encoded požadavek a ne o JSON, je pro lepší přehlednost uvedená struktura právě formou kódu programu. Do těla požadavku se vkládá JWT Token, jehož tvorba bude popsána v následující kapitole.

```
$accessTokenRequestResponse = $client->request('POST',
        'https://www.googleapis.com/oauth2/v4/token',[
        'headers' => [
             'content-type' => 'application/x-www-form-urlencoded'
        ],
        'form_params' => [
             'grant_type' => 'urn:ietf:params:oauth:grant-type:jwt-bearer',
             'assertion' => $jwtToken
        ]
]);
```

Dalším vytvářeným požadavkem na Google API je přímo na službu Google Cloud Vision pro klasifikování uvedeného obrázku. Stejně jako v případě předchozího požadavku se i zde jedná o POST požadavek na adresu: *https://vision.googleapis.com/v1/images:annotate*. Tentokrát ale není požadavek typu url-encoded, ale je typu JSON. Do hlavičky požadavku se vkládá, vedle specifikování typu obsahu, token získaný pomocí autorizačního požadavku.

```
'headers' => [
  'content-type' => 'application/json',
  'Authorization' => 'Bearer '.$token
]
```

Vzhledem k tomu, že tělo požadavku je formou JSON objektu, který je přehledně čitelný, je následující struktura uvedena v této formě.

```
{
  "requests": [
      {
          "image": {
              "content": "iVBORw0KGg... další obsah obrázku ... rjQAAAABJRU5ErkJggg=="
          },
          "features": [
              {
                  "type": "LABEL_DETECTION",
                  "maxResults": 5
              },
              {
                  "type": "IMAGE_PROPERTIES",
                  "maxResults": 5
              }
          ]
      }
  ]
}
```

Tělo požadavku obsahuje parametr „image“, který udává obrázek určený ke zpracování. Dále se v těle požadavku vyskytuje pole „features“, do kterého se udávají požadované

úkony prováděné službou Google Cloud Vision na obrázku specifikovaném v „image“. Obrázek zde specifikovaný může mít dvě formy, které upravují strukturu parametru. První forma, která je uvedena ve struktuře výše, je base64 zápis obrázku. Tato možnost je zvolena v případě, že obrázek je uložen na disku uživatele a je k němu přístup. Druhou možností je zvolit vzdálený obrázek dostupný na uvedené URL, případně obrázek uložený v cloudovém úložišti na platformě Google Cloud. V případě využití druhé možnosti se tedy musí parametr „image“ změnit následovně:

```
"image": {
  "source": {
      "imageUri": "https://adresa-obrazku.cz/images/obrazek.png"
  }
}
```

Pole „features“ obsahuje, jak již bylo zmíněno, výčet úkonů, které se mají provést na uvedeném obrázku, a také počet výsledků, které má API vrátit. Google Cloud Vision API přijímá pouze několik přesně specifikovaných hodnot. Jejich výčet je následující:

- FACE_DETECTION – Detekce obličeje
- LANDMARK_DETECTION – Detekce krajinných útvarů
- LOGO_DETECTION – Detekce loga
- LABEL_DETECTION – Detekce klasifikačních tříd
- TEXT_DETECTION – Detekce textu nebo OCR
- DOCUMENT_TEXT_DETECTION – Detekce textu dokumentu OCR
- SAFE_SEARCH_DETECTION – Detekce potencionálně nevhodného obsahu
- IMAGE_PROPERTIES – Detekce vlastností obrázku jako např. převládající barvy
- CROP_HINTS – Návrh na ořezání obrázku
- WEB_DETECTION – Hledání podobných obrázků na webu
- PRODUCT_SEARCH – Hledání produktu
- OBJECT_LOCALIZATION – Lokalizace detekovaných objektů

Pro klasifikaci vlastní třídy se vytváří požadavek na službu AutoML Vision. Stejně jako u předcházejících požadavků se i zde jedná o požadavek POST s tělem formou JSON objektu. Požadavek se musí rovněž autorizovat pomocí získaného access tokenu. Adresa, na kterou se požadavek zasílá, má následující tvar:

https://automl.googleapis.com/v1beta1/projects/{$projectID}/locations/us-central1/models/{$modelID}:predict

V adrese se musí specifikovat ID projektu platformy Google Cloud, a také ID naučeného modelu. Tělo požadavku, jak již bylo zmíněno, má tvar JSON objektu. Jediný nastavovaný atribut je obrázek, který se má klasifikovat. Tento musí být kódován do tvaru base64. Struktura požadavku je následující:

```
{
    'payload': {
        'image': {
            'imageBytes': $image
        }
    }
}
```

6.2.3 Odezva z Google Cloud Vision API

Google Cloud Vision API vrací odpovědi ve formě jasně strukturovaného JSON objektu. Objekt se skládá pouze z pole „responses“, obsahující pouze ty objekty s požadovanými informacemi, o které bylo zažádáno v požadavku na toto API. Pro příklad lze ukázat strukturu požadavku na klasifikaci tříd a detekci vlastností obrázku. Jedná se pouze o ukázkový příklad odezvy zkrácený o některé výsledky.

```
{
  "responses": [
      {
          "labelAnnotations": [
              {
                  "mid": "/m/01prls",
                  "description": "Land vehicle",
                  "score": 0.9961627,
                  "topicality": 0.9961627
              }
          ],
          "imagePropertiesAnnotation": {
              "dominantColors": {
                  "colors": [
                      {
                      "color": {
                          "red": 234,
                          "green": 235,
                          "blue": 93
                      },
                      "score": 0.12354478,
                      "pixelFraction": 0.008888889
                      }
                  ]
              }
          },
          "cropHintsAnnotation": {
              "cropHints": [
                  {
                      "boundingPoly": {
                          "vertices": [
                              {},
                              {
                                  "x": 1919
```

```
                        },
                        {
                            "x": 1919,
                            "y": 1079
                        },
                        {
                            "y": 1079
                        }
                    ]
                },
                "confidence": 0.79999995,
                "importanceFraction": 1
            }
        ]
    }
  ]
}
```


6.2.4 Odezva z Google Cloud AutoML Vision API

Stejně jako odpověď Google Cloud Vision API, tak i odpověď z AutoML Vision API má jasnou strukturu. API vrací jednoduchý JSON objekt, který obsahuje pole klasifikovaných údajů. Zde je uveden název klasifikované třídy, a také pravděpodobnost příslušnosti k dané třídě.


```
{
    'payload': [
        {
            'displayName': 'Full wrap'
            'classification': {
                'score': 0.9994371
            }
        }
    ]
}
```


6.3 Implementace balíčku

Jelikož bylo řešení navrhnuto v předchozích kapitolách, byl vývoj balíčku řízen přesně podle požadavků a navrhnutých postupů. Jednotlivé třídy balíčku následují navrhnutou strukturu.

6.3.1 Třída VisionFunctions

Hlavní třídou balíčku je *VisionFunctions*, která obsahuje funkce pro volání Google Cloud Vision API, a také pro zpracování vrácené odezvy.

Volání API se uskutečňuje pomocí funkce *makeRequestToAPI*, která bere jako parametry obrázek na zpracování, buď ve formě base64 nebo adresy URL. Další parametr je typu Boolean a slouží právě pro rozlišení formy zadávání vstupního obrázku. Dalším paramet-

rem je pole obsahující textové řetězce indikující požadované činnosti prováděné API na zadaném obrázku. Posledním parametrem je celé číslo Integer, udávající počet výsledků, které má API vrátit. Součástí funkce je i vytváření instance třídy *Authorization* a volání její funkce pro získání *access tokenu*. Funkce vrací objekt odpovědi nástroje Guzzle. Tato funkce je veřejná z toho důvodu, aby mohl uživatel volat i jiné funkce API, než jsou připravené v tomto balíčku.

Třída dále obsahuje čtyři funkce s podobnou funkcionalitou. Tyto slouží pro ulehčení práce s balíčkem, protože obsahují volání funkce *makeRequestToAPI* s již nastavenými parametry tak, aby se z API získaly jen klasifikační třídy nebo klasifikační třídy společně s vlastnostmi obrázku. Všechny čtyři obsahují dva parametry, a to textový řetězec reprezentující obrázek (formou base64 nebo v případě názvu funkce končící URL právě adresou obrázku) a počet vrácených výsledků. Jedná se o tyto funkce, s ukázkou první:

- getImageLabels
- getImageLabelsURL
- getImageLabelsAndProps
- getImageLabelsAndPropsURL

```
public static function getImageLabels($imageBase64, $maxResults = 10){
  $labels = self::makeRequestToAPI($imageBase64, true, ['LABEL_DETECTION'], $maxResults);
  $labels = json_decode($labels->getBody()->responses);
  return $labels;
}
```

Protože se musí získaná odezva zpracovat, obsahuje třída také funkce pro jejich takové zpracování, aby byly splněny všechny stanovené požadavky. Všechny dále uvedené funkce mají pouze jeden parametr, jímž je objekt odpovědi získaný jednou z předpřipravených funkcí uvedených výše.

Jednou z funkcí určených pro zpracování odpovědi je funkce *getBestMakeModel*, která v odpovědi z API nalezne první hodnotu výrobce a modelu vozidla s nejvyšší pravděpodobností. Protože ale API nevrací výsledky týkající se pouze výrobců a modelů vozidel, musí se procházet všechny vrácené klasifikační třídy v odpovědi. Každá se použije pro vyhledání záznamu v databázi, zdali se nejedná o jednu z požadovaných hodnot. Nejprve se hledá výrobce vozidla, a pokud je nalezen, hledá se dále model odpovídající danému výrobci. Na základě testování bylo zjištěno, že model vozidla je v odpovědi uveden současně s jeho výrobcem jako jeden textový řetězec. Proto vyhledávání modelu probíhá na základě vyhledávání konkrétní části textu v textovém řetězci u každé klasifikační třídy.

Jakmile jsou nalezeny oba záznamy, procházení odpovědi skončí a hodnoty jsou vráceny formou pole.

Ne vždy stačí získat pouze nejlepší nalezený výsledek, ale je potřeba získat více možností například pro uživatelský výběr. Proto je implementována funkce *processLabels*. Princip její činnosti je velmi podobný jako u předcházející funkce. To znamená, že se opět prochází všechny klasifikační třídy získané v odpovědi z API a hledají se v ní požadované hodnoty. Při průchodu odpovědí se hledá výrobce a karosérie vozidla. Pokud je výrobce nalezen, vyhledají se všechny modely vozidla stejně jako u předcházející funkce, tj. prohledávání tříd podle části textu. Všechny nalezené modely se ukládají do pole formou jejich ID záznamu z databáze. Následně se takto sestavené pole vloží do dalšího pole s reprezentací klíče pomocí ID záznamu výrobce vozidla. Výstupem funkce jsou právě pole výrobců a příslušných modelů a pole nalezených karosérií. Následující ukázka kódu obsahuje část této funkce, přesněji cyklus procházení získané odpovědi z Google Cloud Vision API připravené pro čtení klasifikačních tříd.

```
foreach($labels as $label){
  $lowercaseLabel = strtolower($label);

  foreach($manufacturersArray as $manufacturerID => $manufacturer){
      if(stripos($label, $manufacturer) !== false && !isset($makeModels[$manufacturerID])){
          $selectedManufacturerID = $manufacturerID;
          $selectedManufacturerName = $manufacturer;
          break;
      }
  }

  $bodyworkID = array_search($lowercaseLabel, $bodyworksArray);
  if($bodyworkID){
      $foundBodyworks[$bodyworkID] = $label;
      $bodyworkID = false;
  }

  if($selectedManufacturerID){
      $models = [];
      $potentialModels = preg_grep("*".$selectedManufacturerName."*", $labelsLowerCaseArray);
      foreach ((array) $potentialModels as $model) {
          $selectedModelID = array_search(str_replace($selectedManufacturerName.' ', '', $model),
$modelsLowerCaseArray);
          if($selectedModelID && !in_array($selectedModelID, $models))
              array_push($models, $selectedModelID);
      }
      $makeModels[$selectedManufacturerID] = $models;
  }
}
```

Další funkcí třídy je *predictAutoML*, která využívá služby Google Cloud AutoML Vision ke klasifikaci vlastních tříd. Parametrem funkce je obrázek ve formě base64. Tělo funkce se skládá ze dvou částí, kde první je vytváření a odeslání požadavku na API, a druhá část je

i následné zpracování odpovědi. Zpracováním je zde myšleno získání názvu klasifikační třídy a pravděpodobnosti příslušnosti k dané třídě. Funkce vrací pole obsahující takto zpracované údaje.

Součástí třídy jsou i funkce zpracovávající část odpovědi API týkající se vlastností obrázku, konkrétně dominantních barev. Jak bylo ukázáno ve vzorové odpovědi z API v předcházející kapitole, barvy jsou v odpovědi reprezentovány formou zápisu samostatných RGB složek. Pro snadnější další použití jsou barvy převáděny do hexadecimálního zápisu. První funkce *getBestRawColor* slouží pouze k vrácení nejvíce zastoupené barvy, převedené právě do hexadecimálního tvaru. Druhá funkce *matchAllColors* využívá další třídy balíčku, a to *ColorClaculator*, k přiřazení každé vrácené barvy k nejbližší barvě uložené v databázi.

6.3.2 Třída Authorization

Třída *Authorization*, jak již z názvu vyplývá, slouží k autorizaci požadavků na server. V případě autorizace bylo možné využít knihovny poskytované přímo od společnosti Google, ale pro potřeby vytvářeného balíčku by se jednalo o přebytečné řešení. Proto jsou v třídě vytvořeny funkce obstarávající přístupový token ke službám Google Cloud API.

Pro získání přístupového tokenu, se musí zaslat požadavek na Google autorizační API s vytvořeným JWT tokenem. Tento je vytvářen v soukromé funkci *createJWT*. Token se skládá ze tří částí: hlavičky, těla a podpisu. Samotný token je pak sestaven jako textový řetězec ve tvaru: *hlavička.tělo.podpis*. Všechny uvedené části jsou kódovány do tvaru base64. Pro lepší čitelnost a přehlednost budou jednotlivé části ukázány před kódováním. Hlavička má vždy stejný tvar, bez ohledu na dotazovanou službu. Jedná se o JSON objekt:

```
{
  'alg': "RS256",
  'typ': "JWT"
}
```

Další částí tokenu je jeho tělo. Stejně jako v případě hlavičky se jedná o JSON objekt, ve kterém se již specifikuje uživatelem část Google Cloud služeb, která má být zpřístupněna a čas trvání a expirace platnosti tokenu. V následující ukázce kódu je znázorněna tvorba těla tokenu, který požaduje přístup ke Google Cloud Platform službám a přístup ke cloudovému úložišti v režimu pouze pro čtení.

```
$jwtClaimSet = json_encode([
  'iss' => \Config::get('googlevision.ServiceAccountEmail'),
```

```
  'scope' => 'https://www.googleapis.com/auth/cloud-platform htt-
ps://www.googleapis.com/auth/devstorage.read_only',
  'aud' => 'https://www.googleapis.com/oauth2/v4/token',
  'exp' => time() + 3600,
  'iat' => time(),
]);
$jwtClaimSetBase64 = base64_encode($jwtClaimSet);
```

Poslední částí tokenu je podpis, který je vytvořen ze zakódovaného textového řetězce částí tokenu *hlavička.tělo*. Pro podepsání se použije algoritmus RSA s SHA256. Samotný podpis je následně také zakódován do base64 a připojen k podpisovanému řetězci, čímž je vytvořen JWT token.

Třída *Authorization* dále pomocí soukromé funkce *requestAccessToken* volá Google autorizační API. Struktura tohoto požadavku byla již popsána v kapitole Datový tok mezi balíčkem a Google Vision API, proto nebude znovu popisována. Veřejnou funkcí této třídy je *getAccessToken*, která po zavolání kontroluje, zdali není platný přístupový token uložen v session. Pokud není, je vytvořen JWT token, který je zaslán v požadavku na Google autorizační API. Jakmile je přístupový token získán, je uložen do session a vrácen funkcí.

6.3.3 Třída ColorCalculator

Protože je součástí požadavku i přiřazení získané barvy k nejbližší barvě uložené v databázi, byla vytvořena třída ColorCalculator. Tato obsahuje privátní funkce pro převod barev z prostoru RGB do prostoru XYZ, následně funkci pro převod z prostoru XYZ do prostoru LAB (více k barevnému prostoru viz. [52]). Další privátní funkcí je výpočet vzdálenosti dvou barev v prostoru LAB. Byla testována varianta počítání vzdálenosti v barevném prostoru RGB, tato se ale po otestování jeví jako zcela nedostatečná.

Veřejná funkce této třídy *getClosestColorFromDB* bere jako parametr barvu v hexadecimálním tvaru. Funkce prochází všechny barvy uložené v databázi a počítá vzdálenosti s barvou zadanou. Výstupem funkce je ID záznamu z databáze nejbližší nalezené barvy.

6.3.4 Modelové třídy

Pro zachování vzoru MVC obsahuje balíček i čtyři modelové třídy. Z důvodu konfigurovatelnosti balíčku tyto modelové třídy neobsahují žádnou logiku a slouží čistě jako reprezentace dat z databáze. Následující ukázka třídy znázorňuje jednu z modelových tříd.

```
class Color extends Model
{
    public function __construct()
    {
      $this->table = \Config::get('googlevision.ColorsTableName','colors');
```

```
        $this->fillable = [
          \Config::get('googlevision.ColorNameColumn','name')
        ];
    }

    public $timestamps = false;
}
```

6.3.5 Konfigurační soubor

Součástí balíčku je i konfigurační soubor umožňující vlastní nastavení některých proměnných využívaných balíčkem. Pro zajištění větší bezpečnosti, případně snadnější vývoj ve více lidech, odkazují uživatelské údaje v konfiguraci do souboru prostředí serveru *.env*. Pro ukázku konfiguračního souboru je zvolena část, která obsahuje volání proměnných prostředí serveru.

```
    /*
        Service account credentials
    */
    'ServiceAccountEmail' => env('GOOGLE_VISION_SERVICE_ACCOUNT_EMAIL', 'exam-
ple@project.iam.gserviceaccount.com'),
    'PrivateKeyJWT' => env('GOOGLE_VISION_SERVICE_ACCOUNT_KEY','123abc'),

    /*
        AutoML Vision project information
    */
    'autoMLProjectID' => 'googleCloudProject',
    'autoMLModelID' => 'modelID_123456789',

    /*
        Table names for models
    */
    'ModelTableName' => 'car_model',
    'ManufacturerTableName' => 'car_manufacturers',
    'BodyworkTableName' => 'car_bodyworks',
    'ColorsTableName' => 'colors',
```

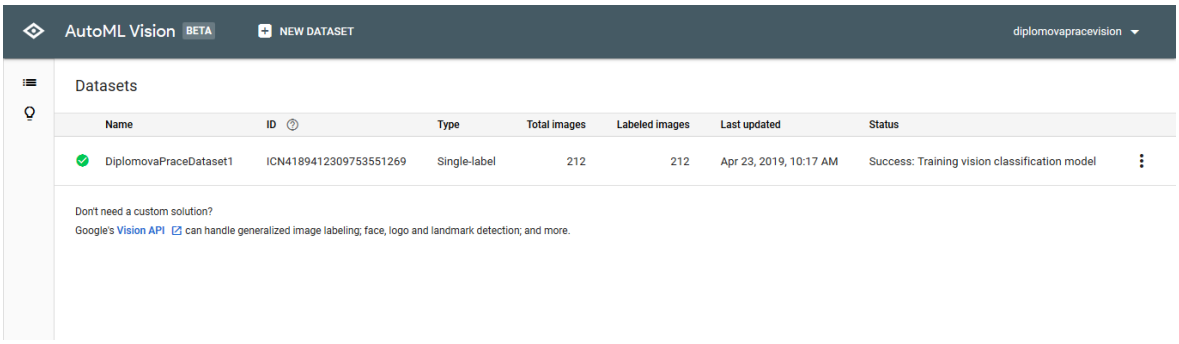
6.4 Učení vlastního klasifikátoru pomocí Google CloudAutoML Vision

Služba Google Cloud Vision byla zvolena také díky možnosti využít služby AutoML Vision, která poskytuje možnost vytvořit vlastní klasifikátor. V současné době je tato služba ale označena jako beta verze, což znamená, že její funkcionalita se může časem změnit natolik, že nebude zpětně kompatibilní.

Před vytvořením vlastního klasifikátoru se musí použít stávající nebo nově vytvořený Google Cloud projekt. Pro tento projekt musí být vytvořeno cloudové úložiště dat, do kterého se budou nahrávat data potřebná pro vytvoření datasetů. Toto úložiště by mělo být pro správnou funkcionalitu služby AutoML Vision pojmenováno v následujícím tvaru: *id-*

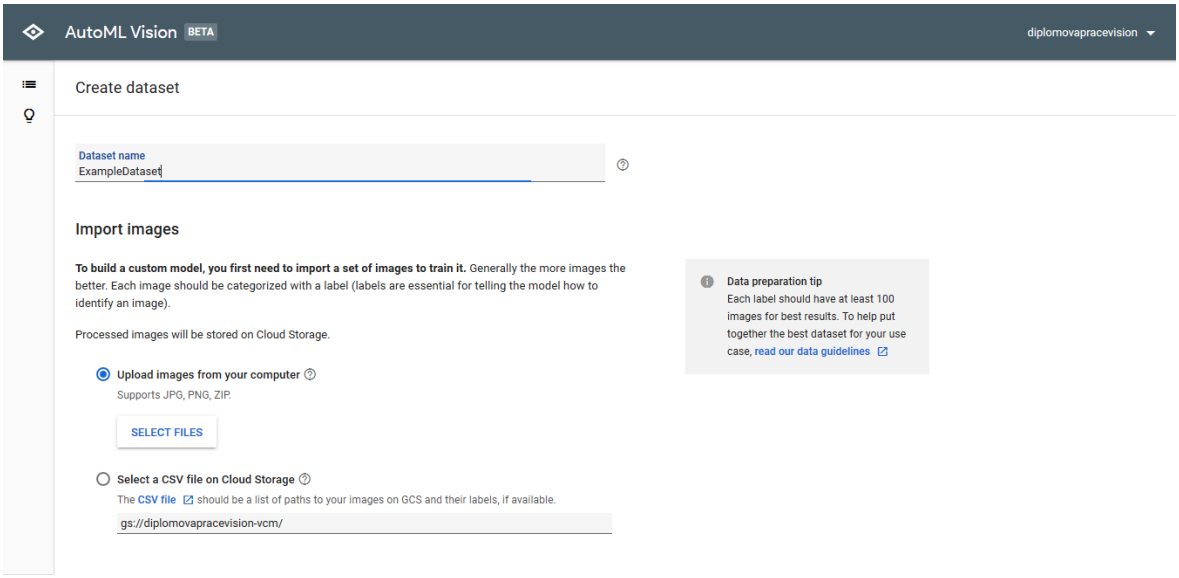
projektu-vcm. Dále musí být v projektu povolen přístup k *CloudAutoML API,* a také k *Storage API*.

Pro vytvoření vlastního klasifikátoru se využívá webového rozhraní poskytovaného společností Google na své platformě. Pro přístup k tomuto rozhraní je třeba se přihlásit do vývojářské konzole Google Cloud Platform a v projektu určeném ke klasifikaci vlastních dat zvolit v nabídce služeb v sekci umělé inteligence nástroj *Vision* a dále *AutoML Image Classification*. Protože je webové rozhraní přehledné a jednoduché na použití, není potřeba implementovat vlastní rozhraní. To by se muselo, v případě výraznějších změn při vývoji služby, přepracovávat.



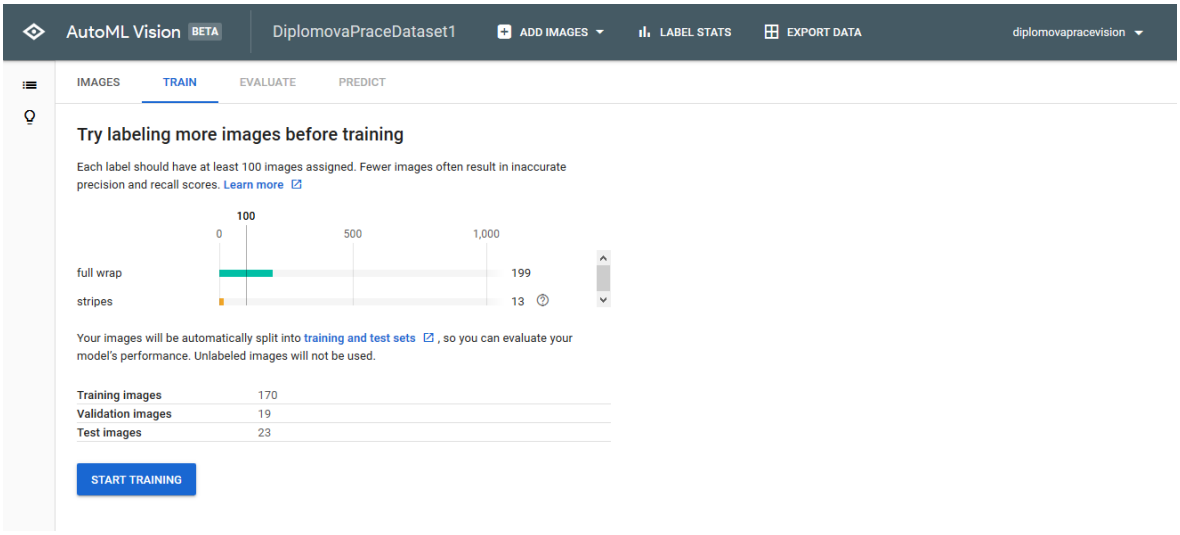
Obrázek 11 Ukázka vytvořeného datasetu v prostředí AutoML Vision

Prvním krokem je vytvoření nového datasetu. K tomu se využije příslušného tlačítka v sekci *Datasets*. Tím se zobrazí průvodce vytvoření datasetu. Nejprve se zvolí libovolný název datasetu. Dále se nahrají obrázky, ze kterých se má právě vytvářený dataset skládat. Nahrání obrázků má několik možností. První je nahrát obrázky přímo, a to jako jednotlivé .jpg, .png obrázky, nebo ideálně zvolit .zip archiv obsahující všechny obrázky. Při zvolení nahrávání jednotlivých obrázků musí být následně tyto obrázky klasifikovány ručně. Pokud jsou ale obrázky obsaženy v .zip archivu v různých adresářích, budou tyto obrázky klasifikovány podle názvu adresáře, ve kterém se nachází. Další možností vytvoření datasetu je použít existující .csv soubor, uložený v cloudovém úložišti, který obsahuje cesty a požadované klasifikační třídy ve tvaru: *gs://cesta-storage/img/img1.png, trida1*. Poslední možností při vytváření datasetu je zvolit možnost nahrát a klasifikovat obrázky později.



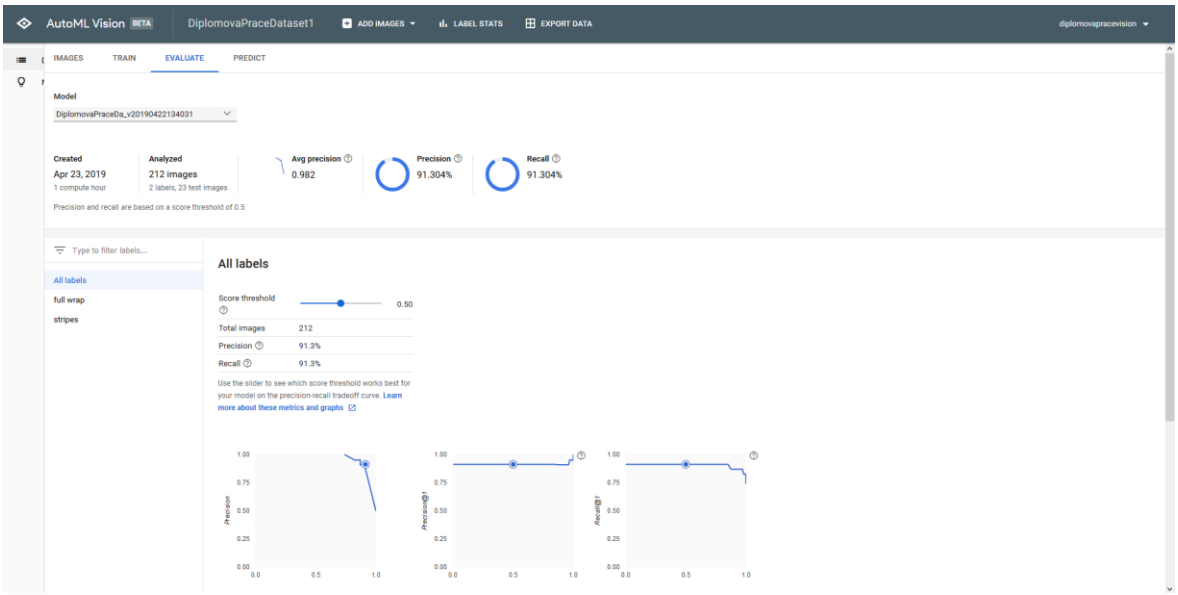
Obrázek 12 Tvorba nového datasetu

Po nahrání a vytvoření nového datasetu následuje spuštění a nastavení vlastního učení. Po zvolení datasetu v sekci Datasets se zobrazí nejprve informační okno s informacemi o zvoleném datasetu. Jsou zde zobrazeny četnosti jednotlivých klasifikačních tříd, a také počet obrázků rozdělených na trénovací, validační a testovací množinu. Minimální počet obrázků na jednu klasifikační třídu je stanoven na 100. Trénování je ale možno spustit i s menším počtem, ale ne menším než 10. Pro správnou klasifikaci se ale doporučuje mnohem vyšších počtů na jednotlivé třídy. Po spuštění učení se musí dále nastavit časový limit učení v jednotkách hodin. Jedná se o maximální možný počet hodin, po které se má model učit. Pokud algoritmus učení dosáhne uspokojivých výsledků v kratším čase, účtován bude pouze aktivní čas a ne celkový možný. Na obrázku níže je zobrazen stav učení modelu v rámci této práce. V požadavcích je uvedeno, že má model rozpoznávat tři různé třídy. V poskytnutých datech byl ale počet třetí třídy menší než 10 obrázků, proto musela být tato třída v tomto případě vyloučena.



Obrázek 13 AutoML Vision – spouštění učení

Jakmile je model naučen nebo uplynul maximální povolený čas, zobrazí se informační okno oznamující procentuální úspěšnost a přesnost naučeného modelu. Na následujícím obrázku je vidět výsledek naučení modelu z poskytnutých dat.



Obrázek 14 Vyhodnocení naučeného modelu AutoML Vision

6.5 Implementace do aplikace

Protože bylo řešení realizováno formou balíčku, je jeho implementace do libovolné aplikace značně usnadněna. Balíček byl strukturován a připraven tak, aby jej bylo možné nainstalovat pomocí nástroje Composer.

6.5.1 Instalace balíčku

Aby bylo možné balíček instalovat z veřejného repozitáře, musí se balíček do některého repozitáře nahrát. V rámci této práce ale byl balíček instalován z lokálního disku. Aby nástroj Composer balíček nalezl, musí se přidat cesta k balíčku na lokálním disku do projektového souboru *Composer.json*. Zde se musí do sekce *repositories* vložit další záznam typu *path*, kterému se následně předá cesta k adresáři s balíčkem. Pokud v projektu nebyly využity žádné vlastní repozitáře, a v souboru není uvedená sekce, je třeba ji vytvořit. V následující ukázce je zobrazen příklad sekce *repositories*.

```
"repositories": [
  {
      "type": "path",
      "url": "C:\\Diplomova_prace\\packages\\jedmak\\dpGoogleVision",
      "options": {
          "symlink": true
      }
  }
]
```

Dále aby bylo balíček možno nainstalovat, je třeba přidat záznam do požadovaných balíčků stejně, jako v případě jakéhokoliv jiného balíčku. Dále je třeba přidat záznam do sekce *autoload* a do jeho podsekce *psr-4*. Níže je ukázána sekce *autoload* se záznamem instalovaného balíčku. Aby instalace proběhla bez komplikací, je vhodné ve všech případech, kde figuruje cesta k balíčku, uvádět tuto cestu jako absolutní.

```
"autoload": {
  "classmap": [
      "database"
  ],
  "psr-4": {
      "App\\": "app/",
      "jedmak\\dpgooglevision\\": "C:\\Diplomova_prace\\packages\\jedmak\\dpGoogleVision
  }
}
```

Po připravení souboru *composer.json* stačí zavolat ve složce projektu příkaz: *composer update jedmak/dpgooglevision --prefer-source*. Tímto se balíček nainstaluje z cesty uvedené ve vytvořeném lokálním repozitáři. Dále je třeba zavést konfigurační soubor balíčku do aplikace. K tomu je nejprve nutné zavést třídu ServiceProvider. Třída se zavede přidáním následujícího řádku do konfiguračního souboru *config/app.php*:

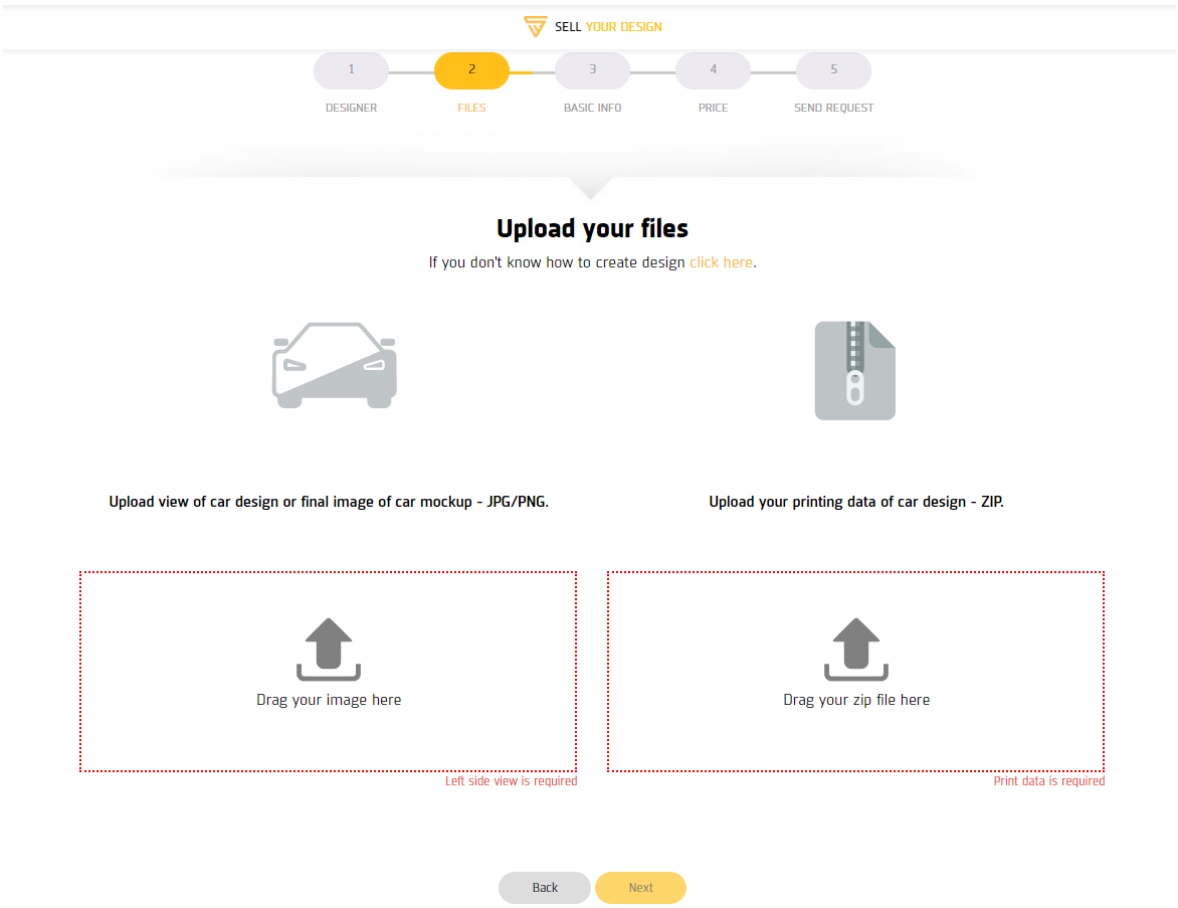
```
jedmak\dpgooglevision\DPGoogleVisionServiceProvider::class
```

Jakmile je přidán ServiceProvider balíčku do aplikace a balíček již byl nainstalován, vytvoří se konfigurační soubor balíčku i v aplikaci. Soubor se vytvoří po zavolání příkazu *php artisan vendor:publish*. Poté bude konfigurační soubor k nalezení mezi ostatními kon-

figuračními soubory ve složce *config*. Následně se v takto vytvořeném konfiguračním souboru nastaví všechny požadované hodnoty, případně se přidají některé proměnné do souboru nastavení prostředí serveru *.env* pod klíči uvedenými právě ve vytvořeném konfiguračním souboru.

6.5.2 Implementace do aplikace Wrapstock

V aplikaci Wrapstock je balíček použit pro usnadnění činnosti zákazníků aplikace. Jedná se o usnadnění procesu nahrávání nového designu od návrhářů, a také o usnadnění procesu požadování vlastního designu. V obou uvedených případech se v aplikaci jedná o vícekrokové formuláře, které musí zákazník vyplnit. Ve formulářích se nachází krok, ve kterém se uvádí specifikace vozidla, pro nějž má být design určen, ať už v případě vytváření nového nebo poptávání po vlastním designu. Součástí tohoto kroku je i možnost nahrát fotku nebo obrázek uváděného automobilu. Této skutečnosti bude využito pro použití vytvořeného balíčku.



Obrázek 15 Formulářový krok pro nahrání nového designu

Protože se jedná o JavaScriptové formulářové pole na straně klienta, jejichž obsah se odesílá na server, balíček je implementován do aplikace formou API. Je to také z důvodu, že oba formuláře požadují vyplnění stejných údajů, které lze získat automaticky s použitím Google Cloud Vision, tedy nainstalovaného balíčku. Vzhledem k tomu, že se vytváří nové API, je třeba pro něj vytvořit i adresu, ze které bude přístupné. Pro vytvoření adresy se využije směrovacího souboru Laravelu *routes/api.php*.

```
Route::post('/v1/predictImage', 'VisionAPIController@predictImage');
```

Vytvořené API je realizováno třídou *VisionAPIController*, která obsahuje pouze jednu funkci *predictImage* implementující funkce balíčku. V požadavku na toto API je obrázek s vozidlem, který je již kódován do base64. Jak je vidět z routy uvedené výše, je dotaz na API realizován jako požadavek POST. Požadavek obsahuje ve svém těle JSON objekt s jedním parametrem *image*, což je právě base64 obrázek.

API implementuje z vytvořeného balíčku pouze třídu *VisionFunctions* a její funkce pro zpracování odpovědi z Google Vision API. Pro potřeby formuláře je potřeba získat vedle výrobce, modelu a karosérie vozidla i jeho barvu, proto je použita příslušná funkce balíčku s nastaveným počtem třiceti výsledků. Dále je získaná odpověď zpracována tak, aby byly nalezeny všechny dostupné kategorie, barvy jsou přiřazeny k barvám z databáze, a je také vrácena nejvíce zastoupená barva. Hodnoty, které mají být z API vráceny, jsou také inicializovány na prázdné hodnoty. Protože balíček vrací pouze indexy záznamů z databáze, jsou za pomoci těchto indexů záznamy nalezeny a pro potřebu vyplnění formuláře jsou získány i názvy nalezených tříd.

```
public function predictImage(Request $request)
{
    $response = VisionFunctions::getImageLabelsAndProps($request->image,30);
    [$manufacturersModels, $foundBodyworks] = VisionFunctions::processLabels($response);
    $colors = VisionFunctions::matchAllColors($response);
    $bestRawColor = VisionFunctions::getBestRawColor($response);

    $manufacturer = '';
    $model = '';
    $color = '';
    $bodywork = '';
    $modelID = null;
    $manufacturerID = key($manufacturersModels);

    if($manufacturerID){
        $manufacturer = CarManufacturer::find($manufacturerID)->name;
        $modelID = current($manufacturersModels[$manufacturerID]);
        if($modelID){
            $model = CarModel::find($modelID)->name;
        }
    }
    $colorID = key($colors);
```

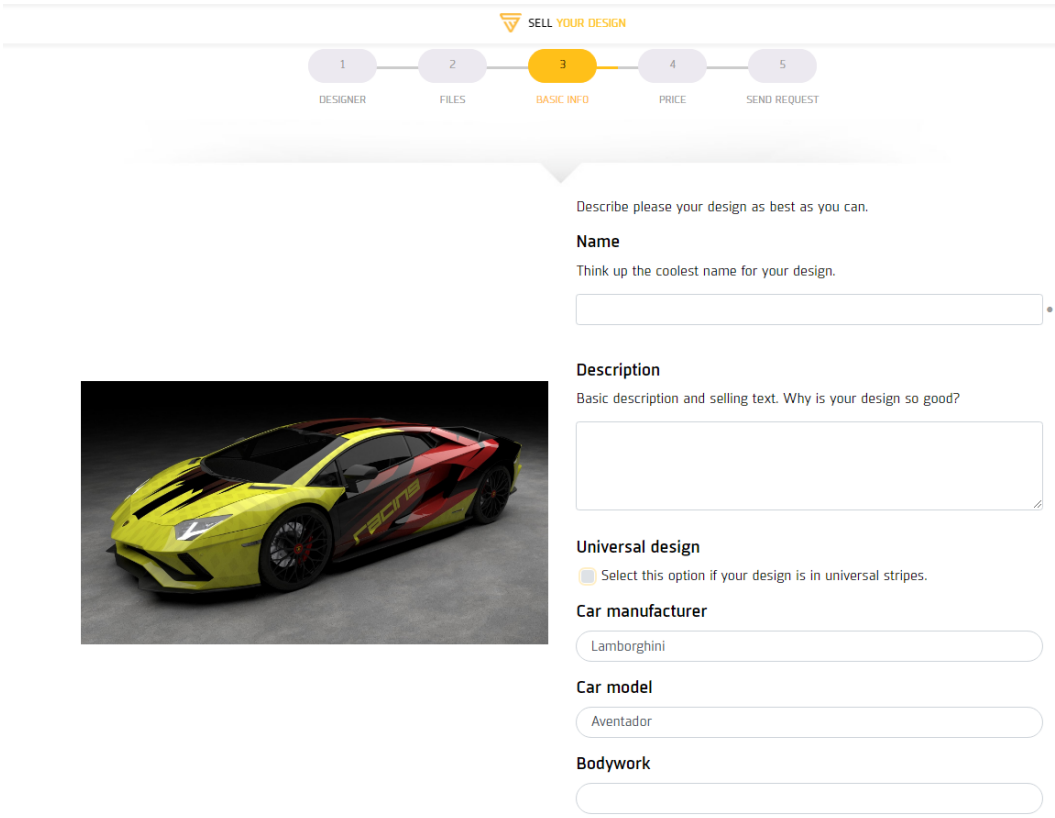
```
    if($colorID){
        $color = DesignColor::find($colorID)->hex_value;
    }
    $bodyworkID = key($foundBodyworks);
    if($bodyworkID){
        $bodywork = DesignBodywork::find($bodyworkID)->name;
    }

    return \Response::json([
        'manufacturer' =>  $manufacturer,
        'manufacturerID' => $manufacturerID,
        'model' => $model,
        'modelID' => $modelID,
        'color' => $color,
        'colorID' => $colorID,
        'colorsMatchingIDs' => array_keys($colors),
        'bestRawColor' => $bestRawColor,
        'bodywork' => $bodywork,
        'bodyworkID' => $bodyworkID
    ]);
}
```

Jak je z výše uvedené funkce patrné, výstupem z API je objekt typu JSON obsahující vždy dvojici ID a název pro všechny požadované třídy vrácené službou Google Cloud Vision.

Ve Wrapstock aplikaci jsou vícekrokové formuláře vedeny jako Vue.JS komponenty na straně klienta. Volání API implementující vytvořený balíček je tedy uskutečněno pomocí jQuery.ajax požadavku. Jakmile je získána odpověď ze serveru, jsou výsledky přiřazeny příslušným formulářovým polím. V případě vytváření nového designu se vybírají výrobce, model a karosérie vozidla pomocí výběrového pole. V případě podávání požadavku na vlastní design se ale výrobce a model vozidla zadávají formou textového řetězce a karosérie taktéž pomocí výběrového pole.

```
$.ajax({
  type: "POST",
  url: "/api/v1/predictImage",
  data: {
    image: fileBase64
  },
  dataType: "json",
  success: data => {
    this.formState.designInfo.manufacturer = data.manufacturerID;
    this.formState.designInfo.model = data.modelID;
    this.formState.designInfo.bodywork = data.bodyworkID;
    this.formState.designAttributes.colors = data.colorsMatchingIDs;
  }
});
```

Výsledkem implementace balíčku do aplikace je ničím nenarušený postup formulářem, kdy lze využít možnosti automatického vyplnění vybraných formulářových polí.



Obrázek 16 Krok formuláře pro nový design s výsledkem klasifikace obrázku

7 TESTOVÁNÍ

Součástí vývoje software je i testování vyvinutého řešení. V rámci této práce bylo provedeno testování jak samotného vytvořeného balíčku, tak i následná implementace do aplikace Wrapstock.

7.1.1 Testování balíčku

K otestování vytvořeného balíčku byla vytvořena jednoduchá testovací aplikace, jejímž cílem bylo otestovat správnou funkčnost balíčku. Aplikace byla vytvořena tak, aby obsahovala nezbytné minimum prvků potřebných pro otestování balíčku. Jako data v databázi byla využita poskytnutá data z aplikace Wrapstock.

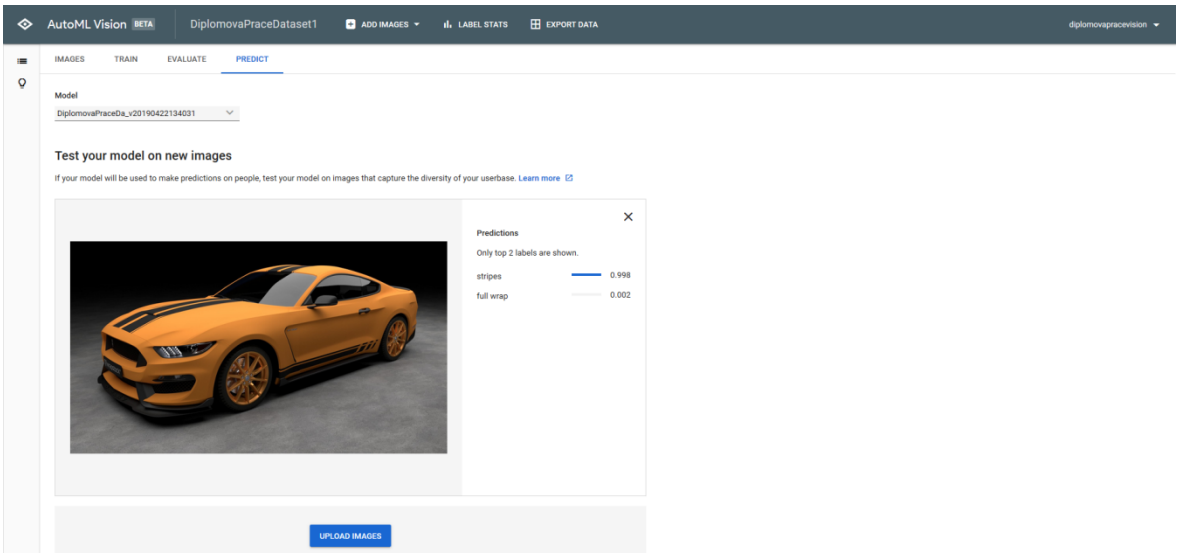
Testovalo se samotné volání Google Cloud Vision API a následně i zpracování odpovědi z tohoto API. Zpracování odpovědi na úrovni výrobců, modelů a karosérií vozidel probíhalo bez komplikací. Bylo potvrzeno, že Google Cloud Vision API vrací ve své odpovědi různé klasifikační třídy, které jsou z větší části nesouvisející s řešeným problémem. Protože nahrávanými obrázky jsou vozidla, byla v nezpracované odpovědi vrácena jako nejpravděpodobnější třída „pozemní vozidlo“. Tato informace je ale pro zadaný problém zcela zbytečná, proto následovalo zpracovávání jednotlivých tříd na základě dat z databáze, stejně jak bylo navrhnuto v návrhu řešení.

7.1.2 Testování v aplikaci Wrapstock

Po nainstalování balíčku do aplikace bylo třeba ověřit, zdali se balíček nainstaloval správně. Dle zadání měl být balíček implementován ve dvou různých formulářích. Nejprve se zpracovávalo řešení podání požadavku na vlastní design, kde bylo otestováno i to, jestli se balíček správně nainstaloval. Tento formulář byl zvolen, protože jsou jeho formulářové pole řešeny formou vkládání textových řetězců, a také nahrávání obrázku je pouze doporučeno a není vyžadováno. V této části bylo prováděno testování přesnosti klasifikace. Na základě odpovědí z Google Cloud Vision API bylo zjištěno, že ve většině případů nahraných obrázků vozidel, bylo API schopno vrátit všechny požadované klasifikační třídy, pokud byl nastaven počet klasifikačních tříd na třicet. V některých případech ale nebyla službou identifikována karoserie vozidla a případně jeho model. Toto může být způsobeno tím, že z nahraného obrázku není model příliš patrný, proto je pravděpodobnost odhadu natolik nízká, že je výsledek zastíněn například tím, že se na obrázku vyskytuje kolo od vozidla.

V rámci testování odpovědí bylo také zjištěno, že databáze poskytnutá k testování balíčku obsahuje data ve formátu, který souhlasí s odpovědí vrácenou API. Nevýhodou řešení je, že se spoléhá na existující databázi, která nemusí být pro správnou funkci klasifikace dostačující. Nedostatečná databáze by se projevila v případě, že klasifikační API vrátí ve své odpovědi všechny požadované informace, které ale nebudou aplikací rozpoznány. Výhodou ale je možnost využití stávající databáze bez narušení její struktury. Toto bylo potvrzeno využitím konfiguračního souboru balíčku, kde se nastavily všechny požadované názvy tabulek a sloupců podle stávající struktury. Balíček na základě tohoto nastavení přistupoval k databázi bez komplikací.

Součástí testování bylo i otestování možností AutoML Vision. Ačkoliv ve webovém rozhraní této služby vycházela velká přesnost naučeného modelu, výsledky jsou pro praktické využití nevhodné. Velká procentuální přesnost je způsobena tím, že v poskytnutém datasetu několikanásobně převyšovala počtem jedna ze dvou možných klasifikačních tříd. Toto má za následek, že klasifikátor přiřazuje většinu vstupních obrázků do převažující třídy. Proto je pro praktické využití třeba dodržet alespoň minimální doporučený počet dat na třídu (doporučeno 1000, ale jako minimum je uvedeno 100), i přes to, že některé obrázky z druhé klasifikační třídy klasifikoval model správně.



Obrázek 17 AutoML Vision – ukázka správné klasifikace vlastní třídy ve webovém rozhraní

ZÁVĚR

Cílem práce bylo vybrat a použít některou z dostupných cloudových služeb poskytujících počítačové vidění na úrovni klasifikace obrázků. Práce byla rozdělena do několika částí, kde v teoretické bylo popsáno strojové učení, jednotlivé služby poskytující ve své nabídce možnost klasifikace obrázků, a také zde byly popsány technologie použité při vytváření programové části práce.

Praktická část byla rovněž rozdělena na několik částí. Nejprve byla stručně popsána aplikace, do které byl vytvořený program implementován. Dále se práce zabývá návrhem řešení programové části. Zde byly rozebrány požadavky na vytvářený program a rozvržení výsledného programu. Na základě sestavených požadavků bylo rozhodnuto o způsobu realizace formou balíčku pro framework Laravel. Dále bylo na základě požadavků a zadání práce zvolena služba Google Cloud Vision, která poskytuje veřejné API. Tato byla zvolena hlavně kvůli přehledné dokumentaci, a také možnosti využití přidružené služby Google Cloud AutoML Vision. Vytvořený balíček tedy implementuje funkce poskytnutého API. Balíček byl uzpůsoben tak, aby jej bylo možné nainstalovat pomocí nástroje Composer.

Vytvořené řešení bylo úspěšně implementováno do aplikace Wrapstock. Na základě poskytnutých testovacích dat byla otestována přesnost klasifikace požadovaných tříd. Bylo zjištěno, že API vrací v převážné většině případů všechny požadované výsledky. V případě, že chyběla v odpovědi některá z klasifikačních tříd, jednalo se nejčastěji pouze o špatně identifikovatelný či neidentifikovatelný model vozidla.

Na závěr byla otestována možnost služby Google Cloud AutoML Vision. Bylo zjištěno, že pro vytvoření klasifikátoru tato služba poskytuje přehledné webové rozhraní, proto není třeba vytvářet vlastní rozhraní pro tyto účely. K natrénování modelu byl poskytnut balík dat, který obsahoval tři klasifikační třídy. Počty obrázků v jednotlivých kategoriích nebyly rovnoměrně rozloženy a převažovala jedna klasifikační třída v počtu 199 obrázků. Druhá třída obsahovala pouze 13 obrázků a poslední, která musela být z procesu učení vypuštěna, obsahovala pouze 2 obrázky. Dle předpokladů se klasifikátor nebyl schopen z tak malého a nevyváženého počtu dat správně naučit. V rámci testování přiřazoval vlastní klasifikátor vstupní obrázky ve většině případů do první třídy, která byla zastoupena největším počtem dat. Pro praktické použití vlastního klasifikátoru je tedy doporučeno mít počet dat rovnoměrně rozložen ve všech třídách a v absolutně minimálním počtu 100 obrázků na třídu.

Výsledkem této práce je balíček, který na základě výsledků z testování úspěšně implementuje klasifikační funkce Google Cloud Vision API. Balíček byl strukturován tak, aby jej šlo využít nejen pro klasifikaci tříd uvedených v zadání, ale i jako obecné rozhraní s Google Cloud Vision API. Součástí balíčku je i využití klasifikace pomocí služby Google Cloud AutoML Vision, kde se v rámci testování ukázal problém strojového učení s poskytnutým malým počtem dat.

SEZNAM POUŽITÉ LITERATURY

[1] MILLS, Terence. Machine Learning Vs. Artificial Intelligence: How Are They Different?. *Forbes* [online]. 2018 [cit. 2019-03-23]. Dostupné z: https://www.forbes.com/sites/forbestechcouncil/2018/07/11/machine-learning-vs-artificial-intelligence-how-are-they-different/

[2] Machine Learning: What it is and why it matters. *SAS The Power To Know* [online]. SAS INstitute, 2019 [cit. 2019-03-23]. Dostupné z: https://www.sas.com/en_hk/insights/analytics/machine-learning.html

[3] VOLNÁ, Eva. *ZÁKLADY SOFTCOMPUTINGU* [online]. Ostrava: Ostravská univerzita v Ostravě, 2012 [cit. 2019-03-23]. Dostupné z: http://www1.osu.cz/~volna/Zaklady_softcomputingu_skripta.pdf

[4] BISHOP, Christopher M. *Pattern recognition and machine learning*. New York: Springer, c2006. Information science and statistics. ISBN 978-0387-31073-2.

[5] Amazon Machine Learning Developer GuideTypes of ML ModelsTraining ML Models. *Amazon Machine Learning* [online]. 2016, 2. 8. 2016 [cit. 2019-03-23]. Dostupné z: https://docs.aws.amazon.com/machine-learning/latest/dg/machinelearning-dg.pdf#training-ml-models

[6] , Jason Brownlee. What is the Difference Between Test and Validation Datasets?. *Machine Learning Mastery* [online]. Machine Learning Mastery, 2017, 14. 7. 2017 [cit. 2019-03-23]. Dostupné z: https://machinelearningmastery.com/difference-test-validation-datasets/

[7] SHAH, Tarang. About Train, Validation and Test Sets in Machine Learning. *Towards Data Science* [online]. 2017, 6. 12. 2017 [cit. 2019-03-23]. Dostupné z: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[8] BROWNLEE, Jason. Overfitting and Underfitting With Machine Learning Algorithms. *Machine Learning Mastery* [online]. Machine Learning Mastery, 2016, 21. 3. 2016 [cit. 2019-03-23]. Dostupné z: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[9] BROWNLEE, Jason. What is a Confusion Matrix in Machine Learning. *Machine Learning Mastery* [online]. Machine Learning Mastery, 2016, 18. 11. 2016 [cit.

2019-03-23]. Dostupné z: https://machinelearningmastery.com/confusion-matrix-machine-learning/

[10] NARKHEDE, Sarang. Understanding Confusion Matrix. *Towards Data Science* [online]. 2018, 9. 5. 2018 [cit. 2019-03-23]. Dostupné z: ://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

[11] What Is Machine Learning?. *MathWorks* [online]. The MathWorks, ©1994-2019 [cit. 2019-03-23]. Dostupné z: https://www.mathworks.com/discovery/machine-learning.html

[12] COVER, T. M. a P. E. HART. *Nearest Neighbor Pattern Classification* [online]. 2018, 15. 5. 2018 [cit. 2019-03-23]. Dostupné z: https://www.cs.bgu.ac.il/~adsmb182/wiki.files/borak-lecture%20notes.pdf

[13] FRIEDMAN, Nir, Dan GEIGER a Moises GOLDSZMIDT. Bayesian Network Classifiers. *Machine Learning* [online]. Kluwer Academic Publishers, 1997, **29**(2/3), 131-163 [cit. 2019-03-23]. DOI: 10.1023/A:1007465528199. ISSN 08856125. Dostupné z: http://link.springer.com/10.1023/A:1007465528199.pdf

[14] LEWIS, David D. Naive (Bayes) at forty: The independence assumption in information retrieval. *Machine Learning: ECML-98* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, , 4-15 [cit. 2019-03-23]. Lecture Notes in Computer Science. DOI: 10.1007/BFb0026666. ISBN 978-3-540-64417-0. Dostupné z: http://link.springer.com/10.1007/BFb0026666.pdf

[15] WAGSTAFF, Kiri, Claire CARDIE, Seth ROGERS a Stefan SCHROEDL. Constrained K-means Clustering with Background Knowledge. *Proceedings of the Eighteenth International Conference on Machine Learning* [online]. 2001, , 577-584 [cit. 2019-03-23]. Dostupné z: https://web.cse.msu.edu/~cse802/notes/ConstrainedKmeans.pdf

[16] PIECH, Chris. K Means. *Stanford CS221* [online]. Stanford, 2013 [cit. 2019-03-23]. Dostupné z: http://stanford.edu/~cpiech/cs221/handouts/kmeans.html

[17] KAISER, Adrien. What is Computer Vision?. *Hayo* [online]. 2017, 12. 1. 2017 [cit. 2019-03-23]. Dostupné z: https://hayo.io/computer-vision/

[18] What Is The Difference Between Computer Vision And Image Processing?. *Analytics India Magazine* [online]. Analytics India Magazine, 2018, 26. 12. 2018 [cit.

2019-03-23]. Dostupné z: https://www.analyticsindiamag.com/what-is-the-difference-between-computer-vision-and-image-processing/

[19] Cloud Vision. *Google Cloud* [online]. Google, [2019] [cit. 2019-03-23]. Dostupné z: https://cloud.google.com/vision/

[20] AutoML Vision Beginner's guide. *Google Cloud* [online]. Google, [2019] [cit. 2019-03-23]. Dostupné z: https://cloud.google.com/vision/automl/docs/beginners-guide

[21] Pricing: Cloud Vision API. *Google Cloud* [online]. Google, [2019] [cit. 2019-03-23]. Dostupné z: https://cloud.google.com/vision/pricing

[22] Pricing: Cloud AutoML Vision. *Google Cloud* [online]. Google, [2019] [cit. 2019-03-23]. Dostupné z: https://cloud.google.com/vision/automl/pricing

[23] What is Azure. *Microsoft Azure* [online]. Microsoft, 2019 [cit. 2019-03-23]. Dostupné z: https://azure.microsoft.com/en-us/overview/what-is-azure/

[24] Cognitive Services. *Microsoft Azure* [online]. Microsoft, 2019 [cit. 2019-03-23]. Dostupné z: https://azure.microsoft.com/en-us/services/cognitive-services/

[25] Computer Vision. *Microsoft Azure* [online]. Microsoft, 2019 [cit. 2019-03-23]. Dostupné z: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/

[26] What is Computer Vision?. *Microsoft Azure* [online]. Microsoft, 2019, 4. 3. 2019 [cit. 2019-03-23]. Dostupné z: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home

[27] Cognitive Services pricing. *Microsoft Azure* [online]. Microsoft [cit. 2019-03-23]. Dostupné z: https://azure.microsoft.com/en-us/pricing/details/cognitive-services/

[28] Amazon Rekognition. *Amazon Web Services* [online]. Amazon Web Services, c2019 [cit. 2019-03-24]. Dostupné z: https://aws.amazon.com/rekognition/

[29] Amazon Rekognition Image. *Amazon Web Services* [online]. Amazon Web Services, c2019 [cit. 2019-03-24]. Dostupné z: https://aws.amazon.com/rekognition/image-features/

[30] Working with Images. *Amazon Machine Learning* [online]. 2016, 2. 8. 2016 [cit. 2019-03-23]. Dostupné z: https://docs.aws.amazon.com/rekognition/latest/dg/rekognition-dg.pdf#images

[31] Amazon Rekognition Video. *Amazon Web Services* [online]. Amazon Web Services, c2019 [cit. 2019-03-24]. Dostupné z: https://aws.amazon.com/rekognition/video-features/

[32] Amazon Rekognition pricing. *Amazon Web Services* [online]. Amazon Web Services, c2019 [cit. 2019-03-24]. Dostupné z: https://aws.amazon.com/rekognition/pricing/

[33] What is PHP?. *PHP* [online]. The PHP Group, c2001-2019 [cit. 2019-03-23]. Dostupné z: https://secure.php.net/manual/en/intro-whatis.php

[34] What can PHP do?. *PHP* [online]. The PHP Group, c2001-2019 [cit. 2019-03-23]. Dostupné z: https://secure.php.net/manual/en/intro-whatcando.php

[35] Laravel. *GitHub* [online]. 2019 [cit. 2019-03-23]. Dostupné z: https://github.com/laravel/laravel

[36] Laravel - Overview. *Tutorials Point* [online]. 2019 [cit. 2019-03-23]. Dostupné z: https://www.tutorialspoint.com/laravel/laravel_overview.htm

[37] Introduction. *Composer: Dependency Manager for PHP* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://getcomposer.org/doc/00-intro.md

[38] Basic usage. *Composer: Dependency Manager for PHP* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://getcomposer.org/doc/01-basic-usage.md

[39] ASP.NET MVC Overview. *Microsoft Docs* [online]. Microsoft, c2019, 27. 01. 2009 [cit. 2019-03-23]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

[40] MVC Architecture. *Chrome* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://developer.chrome.com/apps/app_frameworks

[41] MVC Architecture. *Tutorials Teacher* [online]. TutorialsTeacher.com, c2019 [cit. 2019-03-23]. Dostupné z: https://www.tutorialsteacher.com/Content/images/mvc/mvc-architecture.png

[42] CHACON, Scott. *Pro Git: [everything you need to know about the Git distibuted source control tool]*. 2nd ed. New York, NY: Apress, 2014. ISBN 978-1484200773.

[43] Git. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/

[44] About: Free and Open Source. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/about/free-and-open-source

[45] Git. *GitHub* [online]. 2019 [cit. 2019-03-23]. Dostupné z: https://github.com/git/git

[46] About: Branching and Merging. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/about/branching-and-merging

[47] Branches. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/images/about/branches@2x.png

[48] About: Distributed. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/about/distributed

[49] About: Data Assurance. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/about/info-assurance

[50] About: Staging Area. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/about/staging-area

[51] Staging Area. *Git* [online]. [2019] [cit. 2019-03-23]. Dostupné z: https://git-scm.com/images/about/index1@2x.png

[52] SUDHAKARAN, Sareesh. What is the difference between CIE LAB, CIE RGB, CIE xyY and CIE XYZ?. *Wolfcrow* [online]. 2013, 3. 1. 2013 [cit. 2019-05-10]. Dostupné z: https://wolfcrow.com/what-is-the-difference-between-cie-lab-cie-rgb-cie-xyy-and-cie-xyz/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Program Interface
BMP	Bitmap
CURL	Client URL Request Library
Dataset	Soubor dat
GIF	Graphic Interchange Format
GNU	GNU's not Unix!
GPL	General Public License
HTTP	Hypertext Transfer Protocol
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
JWT	JSON Web Token
KNN	K-Nearest Neighbours
LAB	Barevný prostor LAB
MB	Mega Byte
MVC	Model View Controller
OCR	Optical Character Recognition
ORM	Object Relational Mapper
PDF	Portable Document Format
PDO	PHP Data Object
PHP	PHP: Hypertext Preprocessor
PNG	Portable Network Graphics
REST	Representational State Transfer
RGB	Barevný prostor RGB
SDK	Software Development Kit

URL Uniform Resource Locator

XYZ Barevný prostor XYZ

SEZNAM OBRÁZKŮ

Obrázek 1 Confusion Matrix [10] 12
Obrázek 2 MVC architektura [41] 24
Obrázek 3 Ukázka větvení [47] 26
Obrázek 4 Zobrazení abstraktních vrstev Gitu [51] 26
Obrázek 5 Úvodní stránka Wrapstocku 28
Obrázek 6 Ukázka formuláře pro požadavek na vlastní design 29
Obrázek 7 Diagram případu užití 33
Obrázek 8 Sekvenční diagram – volání Google Vision API 33
Obrázek 9 Diagram tříd 34
Obrázek 10 Orientační struktura databáze 34
Obrázek 11 Ukázka vytvořeného datasetu v prostředí AutoML Vision 47
Obrázek 12 Tvorba nového datasetu 48
Obrázek 13 AutoML Vision – spouštění učení 49
Obrázek 14 Vyhodnocení naučeného modelu AutoML Vision 49
Obrázek 15 Formulářový krok pro nahrání nového designu 51
Obrázek 16 Krok formuláře pro nový design s výsledkem klasifikace obrázku 54
Obrázek 17 AutoML Vision – ukázka správné klasifikace vlastní třídy ve webovém rozhraní 56

SEZNAM TABULEK

Tabulka 1 Ceník služby Cloud Vision [21] 17
Tabulka 2 Ceník služby ruční kategorizace [22] 18
Tabulka 3 Ceník klasifikace naučeného vlastního modelu [22] 18
Tabulka 4 Ceník služby Computer Vision – Microsoft Azure [27] 20
Tabulka 5 Ceník služby Amazon Rekognition Image pro oblast EU [32] 21