STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

IsletSwipe:
Systém sběru dat pro trénink neuronové sítě

Adam Koza
Hlavní město Praha

Praha, 2022

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

IsletSwipe:
Systém sběru dat pro trénink neuronové sítě

IsletSwipe:
A data collection system for neural network training

Autoři: Adam Koza
Škola: Gymnázium Nový PORG, Pod Krčským lesem 25, 142 00 Praha 4
Kraj: Hlavní město Praha
Konzultant: Mgr. Zuzana Hübnerová, PhD

Praha, 2022

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 10.03.2022 Adam Koza

Poděkování

Děkuji především paní Mgr. Zuzaně Hübnerové, PhD za odborné vedení práce a za skvělou možnost poznat komunitu diabetologického centra v IKEM. Velké poděkování také patří panu MUDr. Davidu Habartovi, PhD, který mi umožnil IsletSwipe vyvíjet a poskytnul fotografie Langerhansových ostrůvků, které se v této práci objevují.

Za testování aplikace a zpětnou vazbu při vývoji děkuji také všem z diabetologického centra IKEM.

Anotace

Aplikace IsletSwipe byla vytvořena za účelem souhrnného sběru a analýzy hodnocení snímků vzorků Langerhansových ostrůvků pro trénink neuronové sítě. Užití neuronové sítě značně zrychluje proces výběru vzorků pro transplantaci Langerhansových ostrůvků, což je důležité pro transplantaci ostrůvků u pacientů s Diabetem typu 1. Diabetes typu 1 je chronické onemocnění postihující buňky produkující inzulin ve slinivce, které ovšem lze v některých případech léčit transplantací. Při automatizaci vyhodnocování vzorků pro transplantaci je problematické nalézt konsenzus správnosti ohraničení ostrůvků mezi odborníky, což má dopad na aproximace velikosti a váhy ostrůvků. Vytvořili jsme proto kompletní, flexibilní systém schopný automatizované dodávky, analýzy a prezentace výsledků pro určení správnosti ohraničení ostrůvků od odborníků. Systém stojí na centrální webové aplikaci a API, která komunikuje s mobilními klienty, které obsahují interaktivní prvky pro širokospektrální sběr dat. Analýzu dat provádí externí aplikace v jazyce Java. Systém se ověřil v praktickém užití pro tvorbu tréninkového materiálu pro neuronovou síť IsletNet a další vývoj bude směřovat k širším možnostem kvantitativní analýzy uživatelských hodnocení.

Klíčová slova

Langerhansovy ostrůvky; analýza; mobilní aplikace; trénink neuronové sítě

Annotation

The system IsletSwipe was created to enable the comprehensive collection, analysis, and presentation of user evaluations of pancreatic islet images to aid the training of a neural network. The usage of a neural network speeds up the selection of islets suitable for transplantation for Type 1 Diabetes. Type 1 Diabetes is a chronic disease that causes the dysfunction of insulin producing beta cells in pancreatic islets, which can be treated, in some cases, with transplantation. The approximation of islet mass and volume is hindered by disagreement between users on the correct islet borders. We therefore created a complete and flexible system capable of automatically delivering, analyzing, and presenting user evaluations of pancreatic islets. The system is built upon a central web application and an API that communicates with mobile clients. The mobile application contains interactive elements enabling the collection of a wide range of data on the examined islets. Analysis of the results is performed in a separate Java application. The system has proven itself to be capable of curating training material for the neural network IsletNet, and subsequent development will be directed towards broader options of quantitative analysis of user evaluations.

Keywords

Pancreatic islets; analysis; mobile application; neural network training

Obsah

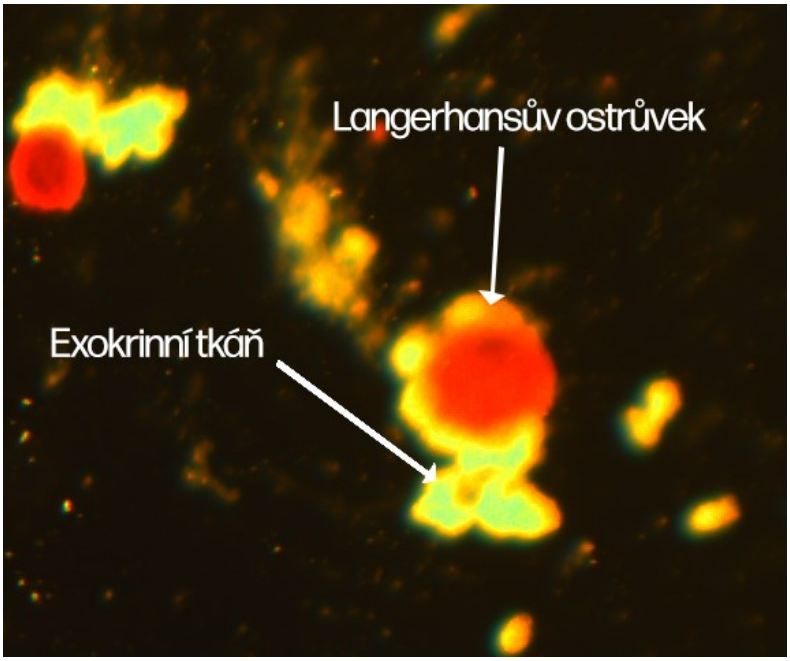
1. Úvod

Od roku 2020 jsem součástí projektu diabetologické laboratoře pražského Institutu Klinické a Experimentální Medicíny (dále IKEM) pro výzkum možností léčby Diabetu typu 1 pomocí transplantace buněk produkujících inzulin. Mimo jiné se projekt zaměřuje na digitalizaci a zefektivnění některých procesů při výběru vzorků pro transplantaci, do čehož zapadá i tato práce, IsletSwipe.

Diabetes Typu 1 je chronické onemocnění postihující celosvětově až 425 milionů jedinců[1]. Vzniká autoimunní reakcí klíčových buněk imunitního systému (mj. CD4+ a CD8+ T-lymfocytů)[2], což vede k postupnému zničení tzv. β-buněk ve slinivce břišní, které produkují hormon inzulin. Jelikož inzulin je důležitým faktorem např. v mechanismu příjmu sacharidů do buněk a provádění buněčného dýchání, nastávají při jeho nedostatku značné komplikace. Mezi ně se člení také hyperglykemie, stav příliš vysoké hladiny cukru v krvi, a chronické stavy jako diabetická neuropatie[1,3]. Dosud ale neexistuje jednoznačný lék pro toto onemocnění, pouze např. symptomatická léčba injekcemi inzulinu[2,4]. Jako efektivní možnost dlouhodobější léčby se ukázala transplantace β-buněk, neboť umožňuje nastolení nezávislosti na ručním dávkování inzulinu u většiny pacientů[5]. Do roku 2018 ve světě vzniklo přes 30 center pro provádění transplantaci β-buněk do pacientů diabetu typu 1, díky vzrůstající popularitě této experimentální metody[5].

Transplantace β-buněk je ale složitá procedura, která vyžaduje například i odhad objemu buněk v rámci transplantovaného vzorku[6]. K tomu jsou β-buňky součástí komplexů tzv. Langerhansových ostrůvků, které v sobě obsahují i jiné buňky s odlišnými funkcemi, např. α-buňky[5]. V obrázku níže (Obrázek 1) se nacházejí endokrinní β-buňky a exokrinní buňky v komplexech Langerhansových ostrůvků.

Pro transplantaci je potřeba jednak Langerhansovy ostrůvky vyjmout ze zdravé slinivky v procesu „izolace“, a také vypočítat jejich hmotnost a objem. Dosud tento proces prováděli lékaři manuálně, ovšem čas odborníků je velmi cenný a manuálně ohraničovat části vzorků je zdlouhavá a náročná práce, a tak vznikla v IKEM snaha tento proces automatizovat[7]. Jako první článek v automatizovaném procesu byla vytvořena neuronová síť IsletNet, která ohraničuje endokrinní β-buňky na digitálních snímcích vzorků Langerhansových ostrůvků[8]. Toto umožňuje výpočet jejich objemu pomocí globulárního modelu, a tudíž rychlý přehled o kvalitě a vydatnosti izolovaného vzorku.



Obrázek 1: Endokrinní a exokrinní tkáň v Langerhansových ostrůvcích. Obrázek pochází z databáze IsletSwipe. Foto: diabetologická laboratoř Institutu klinické a experimentální medicíny (IKEM), Praha

Pro správnou funkci neuronové sítě IsletNet je nutné vytvořit vhodný tréninkový materiál: manuálně označené vzorky, které neuronové síti slouží jako předloha při automatizovaném hodnocení. V procesu přípravy tréninkového materiálu může dojít k neshodě mezi více odborníky hodnotícími daný vzorek, a tak vznikla pro řešení této neshody iniciativa pro projekt popisovaný v této práci, IsletSwipe. Projekt má sloužit jako mezistupeň mezi odborníky a neuronovou síti: má shromažďovat odborníky ohodnocené vzorky izolací Langerhansových ostrůvků a provádět jejich statistickou analýzu. Provedení analýzy a shromáždění hodnocení má umožnit administrátorům sítě IsletNet efektivně vybrat správný ukázkový materiál pro spolehlivý trénink. Při tvorbě mobilní aplikace systému IsletSwipe zastáváme filozofii postupných kroků, tj. aby uživatel svá hodnocení posílal vždy po kouscích a nemusel tak trávit hodiny nad mnoha izolacemi. Dále chceme uživatele v rámci aplikace pozitivně motivovat pomocí trofejí, odměn a soutěží s ostatními, aby každý den dávali našemu systému kus svých dat, který tímto postupným úsilím může vytvořit ekvivalent mnoha zdlouhavých hodin manuálního kreslení ohraničení Langerhansových ostrůvků.

2. Analytický systém

Jako základní pilíř pro IsletSwipe (dále IS) je model analytického systému, který má umožnit globální sběr anonymizovaných dat a jejich následovnou analýzu pro vyvození posudku o kvalitě vzorku a vhodnosti pro trénink neuronové sítě[9]. Proto jsou v aplikaci důležité tři části: sběrová část, zpracovávací část a výstupní část.

Sběrová část je ve formě mobilní aplikace, kterou si mohou uživatelé IS nainstalovat a používat každodenně. Sběr probíhá na 3 základních úrovních: hodnocení celkové kvality a rozlišení izolace, hodnocení správnosti manuálních ohraničení jednotlivých ostrůvků a dokreslování vlastních manuálních ohraničení. Po dokončení hodnocení vzorku na těchto 3 úrovních jsou všechna data odeslána jednorázově z mobilní aplikace na centrální server. Uživatelská data se ale i před odesíláním ukládají na zařízení, což umožňuje vzorek hodnotit kousek po kousku, například i během několika dnů. Toto podporuje naší dříve stanovenou filozofii menších krůčků pro větší pokrok v rámci IS. Uživatelé si tak mohou aplikaci zapnout např. při jízdě v MHD a vyhodnotit několik obrázků, aby si vyplnili dlouhou chvíli.

Navazující analytická část je vykonávána serveru IS, na který jsou všechna uživatelská data odesílána. Data jsou převedena do číselné formy, zejména poměr souhlasů ku nesouhlasům s předlohou manuálního ohraničení jednotlivých ostrůvků. Pomocí matematického modelu se nám podařilo i kvantifikovat odchylku mezi uživatelsky nakresleným ohraničením a vzorovým ohraničením.

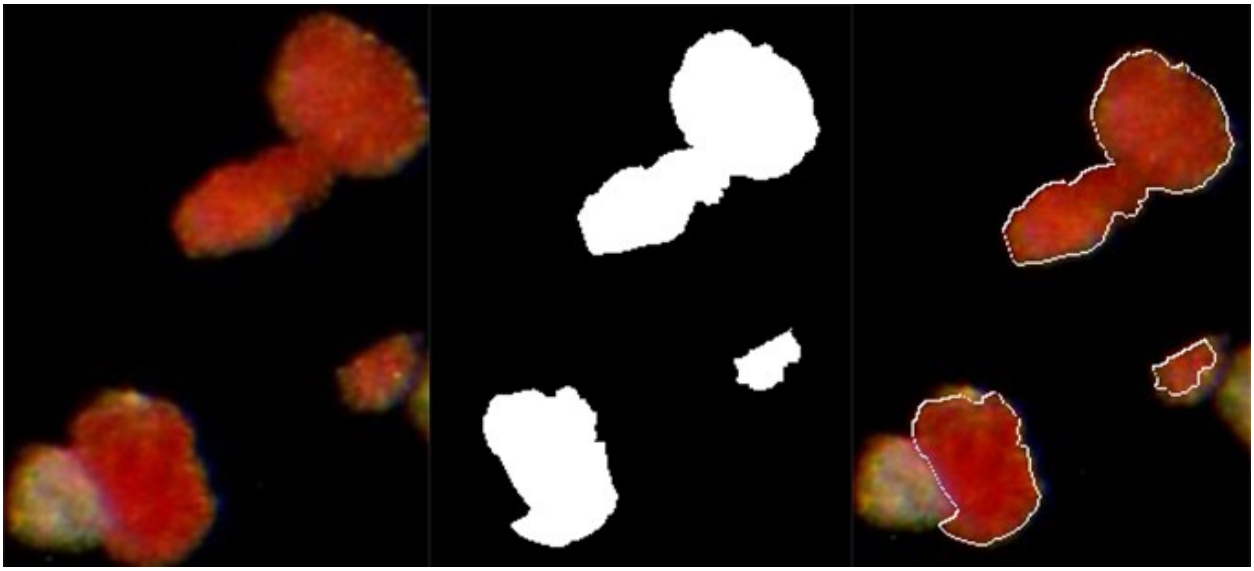
Následovně jsou tyto výsledky prezentovány ve výstupní části, kterou jsme v rámci IS zpracovali ve formě webové aplikace. Statistická analýza hodnocení jednotlivých izolací na 3 úrovních jsou administrátorovi předložena tak, aby mohl relativně rychle posoudit vhodnost obrázku pro trénink na IsletNet. Pro možnost navazující analýzy výsledků nabízí systém také funkci stažení výsledků ve formátu CSV, pomocí které může administrátor výsledky z IS uložit a přenést do jiného systému či aplikace. Tímto může administrátor dostat celistvý přehled o kvalitě vzorku pro trénink neuronové sítě IsletNet.

2.1 Obecná nomenklatura aplikace

Vzhledem ke komplexitě relací mezi datovými jednotkami IS je důležité pochopit roli jednotlivých souborů a jednotek ve statisticko-analytickém procesu. V IS existují 3 druhy obrázků:

- *Base image* (základní obrázek): obrázek obsahuje čistý vzorek snímku Langerhansových ostrůvků bez anotací.
- *Mask* (maska): maska je černobílý obrázek, který definuje regiony, které neuronová síť IsletNet označila za ostrůvky (bílé regiony) a které za okolní prostor (černé pozadí).
- *Contour* (kontura): Kontura je propojení masky a základního obrázku – maska je použita pro vykreslení ohraničení ostrůvků na základním obrázku. Tímto vzniknou barevné vzorky ostrůvků s tenkými bílými hranicemi kolem nich.

Rozdíly mezi těmito obrázky jsou znázorněny na následující koláži:



Obrázek 2: Příklad základního obrázku (vlevo), masky (uprostřed) a vytvořené kontury (vpravo), které přináleží stejnému vzorku. Foto: diabetologická laboratoř IKEM, Praha

3. Software a konfigurace vývojového prostředí

Jelikož je aplikace rozdělena na 3 komponenty – webovou stránku, mobilní aplikaci a generátor kontur – jsou její jednotlivé části naprogramované v několika jazycích. API (*Application Programming Interface*) tyto komponenty propojuje a zastřešuje komunikaci s databází.

3.1 Flutter

Mobilní aplikace, tedy sběrová část analytického systému, stojí na frameworku Flutter od společnosti Google, který je naprogramovaný v jazyce Dart. Flutter je schopen automaticky převádět univerzální kód Dart do systémově-specifických programovacích jazyků, např. do Java nebo Swift, čímž velice usnadňuje vývoj aplikací pro zařízení IOS a Android zároveň[10]. Rovněž disponuje framework Flutter širokou škálou vizuálních prvků pro náročnější funkcionality aplikace, jako například dokreslování uživatelských kontur (sekce **5.5.3 Tvorba vlastních kontur**).

3.2 Symfony

Zobrazovací část aplikace ve formě webové stránky je postavena na frameworku Symfony v jazyce PHP (*Hypertext preprocessor*). Symfony se vyznačuje svojí modularitou a staví na známém modelu MVC (Model-View-Controller)[11]. Framework byl zvolen také kvůli své kompatibilitě s PHP knihovnou *ApiPlatform*, která zastřešuje základní funkce pro API[12] (toto je blíže rozvedeno v sekci **4.3 Komunikace pomocí API).** Tvorba vlastního API systému byla v časovém horizontu projektu nemožná.

Se Symfony spolupracuje i jazyk Twig, který renderuje PHP příkazy do HTML (*Hypertext markup Language*) formy a umožňuje tak tvoření jednoduchých, statických stránek s pokročilejší

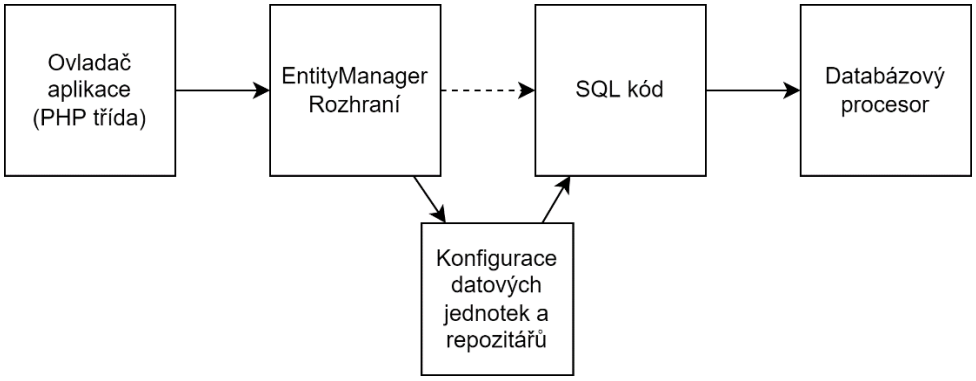
funkcionalitou jako formuláři či vyhledávacími boxy[13]. Pro dodání dynamické funkcionality skrze JavaScript, např. na stránce pro nahrávání obrázku (sekce **7.3.1 Manuální nahrávání ostrůvků**), byl využit integrovaný software Webpack Encore.

3.3 Java

Pro analytické funkce aplikace byl zvolen programovací jazyk Java. Java je systémově nezávislá díky operaci na virtuálním operačním systému JVM (Java virtual Machine)[14]. Také obsahuje mnoho užitečných zabudovaných funkcí pro manipulaci s obrázky, např. *ImageIO*[15]. Dalším důvodem zvolení Javy pro vývoj analytické aplikace je možnost program zabalit se všemi *dependencies* do jednoho .JAR souboru pro rychlý přenos. Externí knihovny jsou připojeny k aplikaci za pomocí správního systému Maven, který zajišťuje aktuálnost verze každé knihovny a jejich integraci do tříd projektu[16]. Analytická aplikace nemá vizuální prvky, protože je volána přímo z webové stránky skrze třídu *Process*.

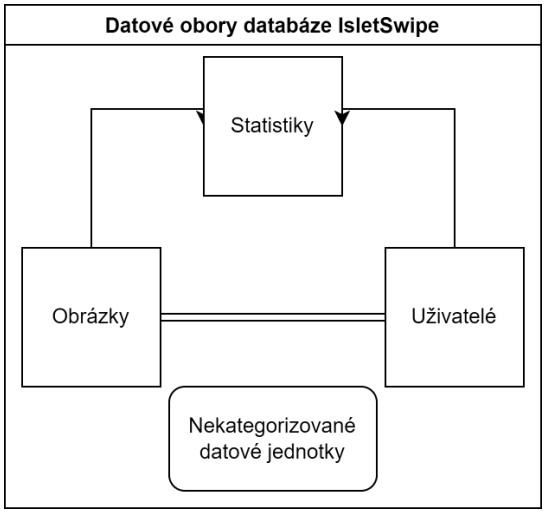
4. Databáze a API

Tato sekce se zabývá logickým propojením jednotlivých datových jednotek v databázi IS a jejich využití v systému API. Databáze IS se opírá o princip relační databáze, ve kterém jsou jednotlivé datové jednotky (tzv. *entities*) navzájem propojeny a ve sloupcích své databázové tabulky mohou odkazovat jedna na druhou. Toto umožňuje aplikaci zejména přehlednější ukládání dat a určování složitějších vztahů mezi nimi (např. ManyToOne, ManyToMany apod.). V IS je systém relační databáze zprostředkován pomocí frameworku Doctrine, který je výchozí součástí Symfony. Doctrine dodává nadstavbu databázi MySQL, jež umožňuje komplexní definice datových jednotek a jejich vztahů ve třídách PHP. Data z databáze jsou poté v reálném čase poskytována serveru pomocí Doctrine třídy *EntityManager*, která extrahuje obsah databáze do formy PHP tříd.



Obrázek 3: Ukazuje proces transformace schémat datových jednotek z PHP tříd do datového kódu. Foto: autor, vytvořeno v software DrawIO, dostupné z https://app.diagrams.net/

Data v rámci IS jsou dělena do tzv. oborů, což jsou logické celky obsahující datové jednotky spojené podobnou funkcionalitou. Mezi základní datové obory patří obrázky a uživatelé, kteří jsou propojeni oborem statistiky. Následující struktura vyznačuje obory databáze IS a jejich vzájemné vztahy



Obrázek 4: Označuje uspořádání datových oborů v databázi IS. Uživatelé a obrázky jsou logicky propojeny pomocí statistiky, které mají relační vztahy s oběma obory. Vedle toho existují datové jednotky, které nejsou kategorizované do oborů, ovšem ty slouží pro méně

4.1 Struktura datových jednotek

Centrem celé databáze je obor obrázků, který v sobě ukládá informace o vzorcích Langerhansových ostrůvků. Pro každý vzorek skladuje databáze údaje o jeho základním obrázku, masce a obrázku kontury. Obrázky jsou poté napojeny na jednotky ostrůvků a uživatelské statistiky. Struktura propojení těchto datových jednotek je znázorněna v **příloze A.**

Dále propojuje aplikace všechna nahraná hodnocení s jejich autory (uživateli), ovšem při stahování statistik dochází k anonymizaci uživatelských jmen pro objektivitu ve vyhodnocování vzorků. Uživatelé jsou připojeni k uživatelským skupinám, které často označují např. transplantační centra nebo týmy. Diagram, který znázorňuje propojení uživatelů v systému, se nachází v **příloze B**.

4.3 Komunikace pomocí API

Datové obory IS jsou propojeny v API, která zprostředkovává jednotné komunikační médium mezi mobilními klienty a webovou stránkou. Konfigurace pro zabezpečení a zpřístupnění API endpointů je uložena v souboru *api_platform/resources.yaml*, který konfiguruje jednotlivé Symfony *Entities* a zajišťuje správné zpřístupnění dat na API. Klienti API buď stahují data z webu nebo nahrávají své výsledky do databáze. Proto API omezuje přístup klientů k metodám *DELETE* a *PATCH*, které nejsou použitelné v mobilní aplikaci, a proto jsou jen bezpečnostním rizikem, pokud nejsou v API vypnuty.

Konfigurační soubor *resources.yaml* obsahuje pro každý endpoint parametr *security*, který určuje restrikce přístupu klientů podle jejich uživatelské role. Dále definuje tento soubor tzv. „normalizační“ a „denormalizační“ metody, čímž limituje hodnoty datové jednotky (*entity*), které jsou v API uveřejněny pro psaní a čtení. Dodatečně může konfigurace u některých jednotek upravovat přístupové *URL* či *GET* parametry pro *HTTPS* požadavky.

```yaml
App\Entity\UserEdit:
  shortName: "edits"
  attributes:
    security: 'is_granted("ROLE_ADMIN") or object.user == user'
    normalization_context:
      groups: ["edit:read"]
    denormalization_context:
      groups: ["edit:write"]
  itemOperations:
    get: ~
  collectionOperations:
    post:
      security: 'is_granted("ROLE_USER")'
      controller: App\Controller\API\Selections\CreateImageEditController
```

Obrázek 5: Příklad konfigurace datové jednotky pro uživatelské kontury (*User Edit*). V atributu *security* je definován obecný požadavek autorizace: buďto je uživatel vlastníkem dané kontury nebo má roli *ADMIN*

5. Mobilní aplikace

Mobilní aplikace IS je centrem sběrové části celého systému. Jedná se o propojení mezi uživatelem, analytickým systémem a administrací IS. Mobilní aplikace má především za úkol sbírat uživatelská data o kvalitě celkových obrázků a jednotlivých Langerhansových ostrůvků. Obsahuje ale také zábavné prvky, které mají uživatele pozitivně motivovat v další účasti v programu IS. Pro přenos potřebných dat z mobilní aplikace na web byl proto při vývoji kladen důraz na spolehlivé spojení pomocí API, aby se mobilní klienti mohli synchronizovat se současným stavem databáze na webu.

Vedle toho jsme si při vývoji uvědomili potřebu fungování aplikace bez připojení k internetu (*offline* funkcionality) pro použití např. v MHD. Tudíž je mobilní aplikace IS kromě propojení s API zaopatřena také vlastní *Cache* databází, vytvořené v systému SQLite, která umožňuje dočasné uložení dat z API do mobilu pro *offline* použití. Aplikace si jednorázově stáhne všechna potřebná data z API a aktualizuje jimi lokální *Cache* databázi, pokud se může připojit k internetu.

5.1 Datový model mobilní aplikace

Pro bezproblémovou komunikaci mezi mobilní aplikací a API je potřeba vytvořit flexibilní systém ukládání a odesílání uživatelských dat. Kvůli *offline* potřebám aplikace má proces odesílání dat z mobilní aplikace na web několik kroků: aplikace nejprve data od uživatele přijme, poté je uloží do lokální *Cache* databáze, aniž by kontaktovala API, a až po dokončení většího celku hodnocení data odešle na API. Tímto způsobem předcházíme v aplikaci zbytečně složitému kódu spojenému s odesíláním dat na několika místech a problémům při ztracení připojení k internetu. Souhrnně je datová architektura aplikace postavena tak, aby se minimalizovalo kontaktu mezi aplikací a API a potřebná data byla odesílána jednorázově.

Kvůli tomu existuje v aplikaci rozsáhlý systém *Cache* databáze, postavený na architektuře SQLite, který obsahuje především následující tabulky:

1. *Islet Image* – obsahuje data k lokálně staženým obrázkům a jejich parametrům, vč. obrázků kontur vzorků
2. *Islet Selection* – ukládá data jednotlivých Langerhansových ostrůvků a jejich pozic na obrázcích
3. *User Feedback* – sestává z uživatelských hodnocení obrázků a ostrůvků, které jsou následovně odesílány na API
4. *Contour Edit* – jsou uživatelem nakreslené úpravy existujících ohraničení Langerhansových ostrůvků (kontur), které jsou spolu s *UserFeedback* odesílány na API.

Nad *Cache* databází, kterou lze volat pouze asynchronní metodou s klíčovým slovem *await*, v aplikaci existuje také třída *Session*. *Session* je přístupná z celé aplikace a její stav se po restartu aplikace neukládá. V *Session* jsou uložena často používaná data, ke kterým je potřeba co nejrychlejší přístup. Jedná se proto například o parametry jednotlivých obrázků, současná sada apod. Kvůli náročnosti na paměť bylo rozhodnuto v Session neukládat data jednotlivých obrázků. Následující souhrn obsahuje všechny aktuálně používané prvky, které se ukládají v Session aplikace:

1. *Images* (obrázky) – Obrázky, které aplikace načetla z API či lokální *Cache* databáze. V *Session* jsou pomocí relací připojeny ke svým třídám *ImageHolder* a *IsletSelection*.
2. *Image Bundles* (balíčky obrázků) – Rozřazení obrázků do balíčků po 3, včetně indikace současného balíčku obrázků (sekce **5.2 Přístup k hodnocení obrázků**).
3. *Statistics* (statistiky) – Souhrnný seznam všech uživatelských hodnocení obrázků.

5.2 Přístup k hodnocení obrázků

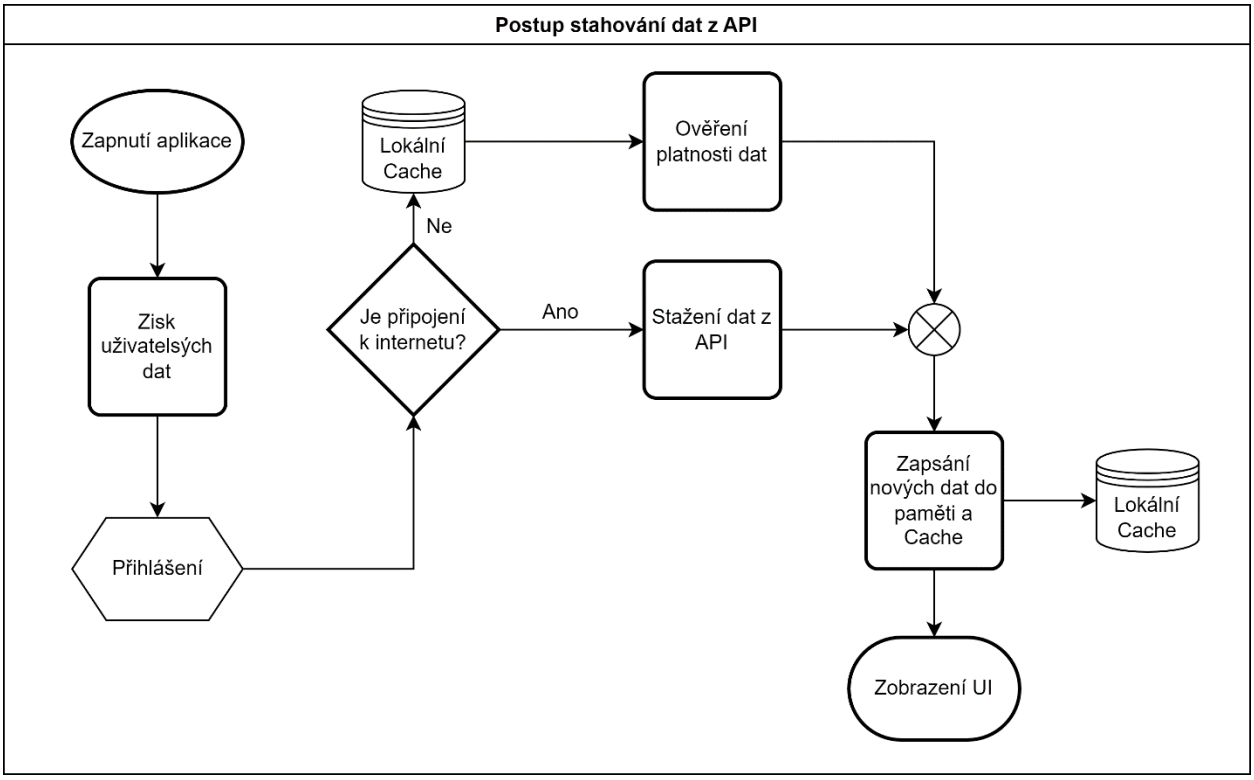
Při tvorbě designu aplikace jsme věnovali značnou pozornost způsobu, jakým uživatelům budou obrázky „servírovány“. Především jsme se zaměřovali na dodržení celosystémového přístupu k postupnému sběru uživatelských dat, a tak jsme se rozhodli pro rozdělení obrázků na přehledné „balíčky“ po 3, mezi kterými by byl uživatel odměněn animací a povzbudivým textem. Tímto způsobem dává aplikace uživateli najevo poděkování za věnování chvilky času pro tvorbu konsenzu dat umělé inteligence, a tak jej pozitivně motivuje pro další zapojení do systému.

Poděkování existuje v aplikaci ve formě animovaného GIF souboru a zvukového doprovodu, nazývaného *Award*. *Awards* jsou nahrávány na webové stránce (popsáno stručně v sekci **7.6 Správa uživatelských odměn**) a po dokončení balíčku 3 obrázků si vyžádá mobilní aplikace od serveru náhodný *Award*. Příkladem takové odměny je například kreslená animace ohňostroje.

5.3 Stahování obrázků

Po přihlášení do systému a ověření platnosti uživatelských dat musí IS aktualizovat svá data z API a případně stáhnout nejnovější sadu obrázků s patřičnými parametry. Aplikace se primárně

pokouší data stáhnout z API a vytvořit tak jejich nejaktuálnější schéma v lokální *Cache* databázi. Ovšem pokud není přístup k internetu, využije starší lokální kopie těchto schémat.



Obrázek 6: FlowChart znázorňující jednotlivé fáze stahování dat z API IS. Důležitý je zde krok vytvoření/přepsání dat lokální *Cache* databáze, která slouží pro případnou kopii schémat aktuální sady obrázků, když zařízení ztratí přístup k internetu. Foto: autor, vytvořeno v software DrawIO, dostupné z https://app.diagrams.net/

Aplikace data stahuje ve většině případů Flutter nativní metodou *Url.https* s headerem *Cookie* pro autentizaci každého požadavku. Pro stahování obrázků aplikace však přímo streamuje odpověď od API endpointu */api/image/download*, který místo JSON kódu předkládá obsah požadovaného obrázku ve formátu *application/octet-stream*. Streamovaná data jsou postupně zapisována do lokálního souboru, jehož existence je následovně potvrzena uložením do *Cache* databáze. Tento způsob umožňuje značnou flexibilitu při sledování průběhu stahování pomocí Flutter *StreamBuilderu*. Metodu pro stahování lze také přeložit na jiné procesorové vlákno pomocí tzv. Dart Izolátů (*isolates*)[17]. Následující ukázka kódu demonstruje proces stahování obrázku IsletImage:

```
Future<File> downloadImage(String url, String storagePath, {Function progressCallback, IsletImage  image, noCache = false}) async {

  … tvorba potřebných tříd, kontrola, že soubor již neexistuje apod.

  try {
    var request = await client.getUrl(new Uri.https(ApiUrls.API_URL, url));
    request.headers.add('Cookie', Session.sessionCookie);
    var response = await request.close();
    if (response.statusCode == 200) {
      // Seskupení streamovaných dat
      Uint8List streamContent = await consolidateHttpClientResponseBytes(response);
      downloaded = new File(storagePath);

      // Zapsání streamovaných dat z Uint8List do souboru
      await downloaded.writeAsBytes(streamContent);

      if (!noCache) {
        await ImageDownload.newCache(downloaded, image);
      }

      … aktualizace statusu stahování přes funkci progressCallback

      return downloaded;
    } else {
      … Zachycení chyb apod.
  }
```

Obrázek 7: Sekce kódu, která umožňuje streamování odpovědi API a stažení obrázku ze specifikované URL. Prvně dochází ke „konsolidaci“, tedy seskupení streamovaných bajtů obrázku do proměnné *Uint8List*. Tato proměnná je poté zapsána do souboru zařízení přes metodu *File.writeAsBytes.* Pro zjednodušení je zde vynechána část, která aktualizuje průběh stahování ve *StreamBuilderu*

Vždy po stažení dat jsou data připojena do Session, kde jsou asociována dle potřeby s dceřinými a mateřskými třídami. Po kompletní synchronizaci lokální databáze s API aplikace vygeneruje pomocné třídy a soubory, které optimalizují její běh a uživatelskou přívětivost. Mezi nejvýznamnější prvky optimalizace patří tvorba tzv. *Thumbnails*, tedy zmenšených obrázků zastupujících jednotlivé obrázky vzorků ostrůvků. Jelikož každý reálný obrázek má většinou přes 3 MB ($N = 149$, $\mu = 3.1$ MB, $\sigma = 1.3$ MB), je jejich zobrazování náročné na procesorovou sílu a paměť, což se ukázalo po testování zvláště na starších zařízeních. Proto IS volá metodu *generateThumbnails*, která obrázek zmenší pro rychlejší zobrazení. *GenerateThumbnails* používá funkci *FlutterNativeImage.compressImage* na novém procesorovém vlákně. Tato funkce snižuje kvalitu barev na obrázku, čímž efektivně zhoršuje jeho rozlišení, ale také zmenšuje jeho velikost. Protože *Thumbnails* se používají jen pro orientační identifikaci obrázku, a ne pro jeho samotné hodnocení, je nižší kvalita obrázku dostačující.

Jakmile je celý proces stahování dat hotový, zobrazí aplikace uživateli aktuální balíček obrázků pomocí widgetu *IsletHomeDisplay*.

5.4 Uživatelské rozhraní aplikace

Po dokončení stahování a procesů v pozadí dochází v aplikaci k interakcím s uživatelem, které mají vyústit sběrem potřebných dat a jejich odeslání na API. Pro zjednodušení procesu hodnocení obrázků a ostrůvků jsme v IS přidali několik prvků, které slouží jako nápověda nebo svojí intuitivností zjednodušují proces tvorby a sběru dat.

5.4.1 Main menu

První obrazovka, která se uživateli zobrazí po načtení aplikace, je hlavní menu, které existuje pro rychlou orientaci v aktuálních úkolech a pro přehled o celkovém postupu v dané sadě obrázků. Tuto obrazovku zprostředkovává prvek *IsletHomeWidget*, který obsahuje 3 *ImageTile* widgety. Widgety *ImageTile* zobrazují jednotlivé obrázky a zastřešují potřebná tlačítka a statistiky týkající se daného obrázku. Pro co největší jednoduchost aplikace zabírají obrázky většinu plochy *Main menu*, a tudíž není pozornost uživatelů odvrácena od hlavní funkce aplikace.

Obrázek 8: Screenshot příkladu vzhledu *Main menu*. Zde je vidět důraz na jednoduchost uživatelského rozhraní (*UI*), především díky velikosti obrázků relativně k ostatním prvkům aplikace. Obrázek úplně nahoře (označen 1) byl uživatelem již ohodnocen a čeká na odeslání na API. Tudíž se na něm zobrazují statistiky dávající uživateli přehled o vlastním hodnocení obrázku a možnost toto hodnocení zrevidovat. Foto: autor

5.4.2 Návod

Vedle přehlednosti uživatelského rozhraní (*UI*) aplikace jsme při vývoji IS viděli význam v představení poměrně složitého systému hodnocení ostrůvků uživatelům v interaktivní formě. Tudíž jsme zvolili možnost posouvacího obrázkového návodu, jenž obsahuje informace o roli aplikace ve vylepšování tréninkového materiálu pro neuronové sítě, přehledné ilustrace pro použití aplikace a návod pro případné řešení problému s aplikací.

Návod je rozdělen do 35 obrázků, které jsou uloženy staticky v sekci *assets* v mobilní aplikaci IS. Při zapnutí aplikace jsou obrázku návodu přečteny a jejich dimenze (šířka a výška) jsou zapsány do *Session*. Pomocí těchto údajů widget *Tutorial* správně vypočítá zmenšení obrázku, aby pokryl maximální plochu obrazovky, ale zároveň se na ni vešel.

5.4.3 Stupeň vítězů

Mezi důležitý element vytvoření atmosféry hry oproti zdlouhavé práci se v IS řadí obrazovka stupně vítězů, *Leaderboard. Leaderboard* dává uživatelům možnost zjistit, kolik obrázků ohodnotili ve srovnání s ostatními uživateli, kterým byla přidělena stejná sada. Vypočítání pozic jednotlivých uživatelů probíhá kompletně na straně API a aplikace si je pouze vyžádá pomocí endpointu */api/leaderboard*. Odpověď ve tvaru JSON obsahuje data o pozicích 3 uživatelů na stupních vítězů a statistiky aktuálně přihlášeného uživatele. Aby aplikace nemusela stahovat data nejlepších uživatelů při každé iteraci životního cyklu widgetu *Leaderboard*, uloží se stažená data stupně vítězů do *Session*. Pro aktualizaci dat může uživatel využít speciální tlačítko, které pomocí asynchronní metody data v *Session* přepíše a widget *Leaderboard* donutí k obnovení pomocí metody *setState*.

5.5 Zobrazování obrázků

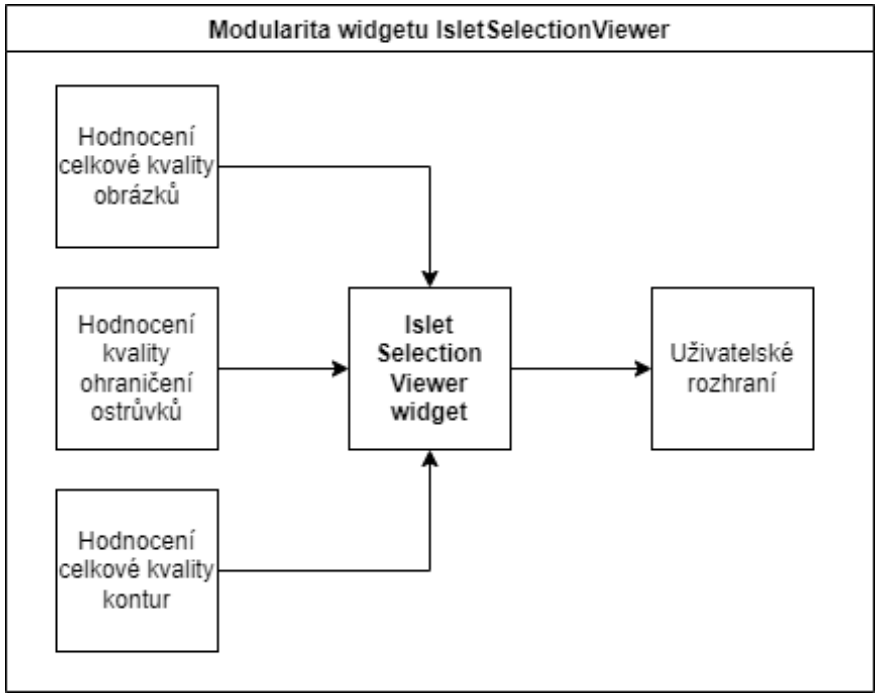
Možnost interaktivního prohlížení obrázků je nejdůležitější funkcionalitou mobilní aplikace IS, jelikož skrze ni lze od uživatele získat přesná data o jejich názoru na dané ohraničení vzorku. Při vývoji IS bylo proto rozhodnuto umožnit uživateli co největší volnost v prohlížení obrázků. Toho bylo dosaženo z velké části pomocí knihovny *PhotoView*[10], která navazuje na sadu funkcí Flutteru pro snímání gest na displeji a dovoluje uživateli si prohlížet, přibližovat či oddalovat obrázky. Knihovna tyto funkce obaluje vlastním Widgetem, který dovoluje značné přizpůsobení specifickým situacím, jako např. i ovládací třídu pro streamování hodnot pozice daného obrázku[10]. Celý modul zobrazování obrázků v aplikaci zastřešuje widget *IsletSelectionViewer*, který provádí proces zobrazování a posouvání obrázku.

IS na jednom obrázku provádí sběr 3 typů dat: kvality celkového základního obrázku, kvality celkového ohraničeného vzorku (kontury) a kvality ohraničení vybraných Langerhansových ostrůvků. Aby aplikace mohla rozlišit mezi jednotlivými druhy sběru dat, je hodnocení obrázku rozděleno do úkolů, tzv. *Tasks*. Úkoly pro jednotlivé obrázky jsou předgenerované při zapnutí aplikace a po otevření obrázku vybere IS první úkol, který uživatel ještě nesplnil. Díky tomu lze

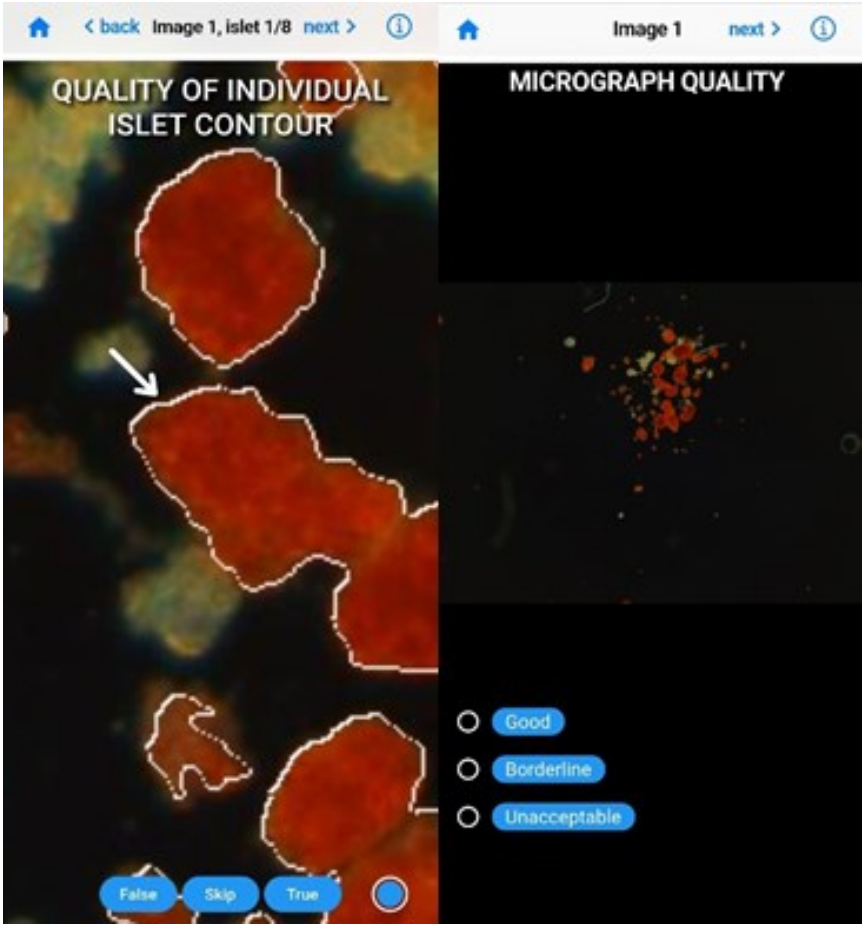
v aplikaci implementovat navigační lištu, pomocí které se uživatel může pohybovat mezi úkoly na jednom obrázku. Všechny instance třídy *Task* jsou totiž napojeny na své mateřské třídy *IsletImage* v *Session* (která je popsaná v sekci

5.1 Datový model mobilní aplikace) a tudíž jsou jednoduše přístupné při *renderování* lišty.

Zobrazovat obrázek a umožnit uživateli se na něm pohybovat je v aplikaci potřeba na několika místech (Obrázek 9). Tudíž je widget *IsletSelectionViewer* vytvořen modulárně, aby jako argument akceptoval instanci *Task*, která obsahuje obrázek pro zobrazení a parametr určující prvky ovládání, které mají obrázek překrývat. Tímto může aplikace jednoduše rozlišit mezi zobrazováním hodnocení jednoho ostrůvku a hodnocení celkového obrázku, jelikož *IsletSelectionViewer* dokáže podle odlišného druhu *Task* na obrazovce zobrazit příslušná tlačítka pro získání hodnocení.



Obrázek 9: Obrázek ukazuje různá propojení funkcionalit, které všechny využívají widget *IsletSelectionViewer*. Tímto způsobem se v aplikaci značně snižuje množství duplicitního kódu (přes 2000 řádek). Každý komponent hodnocení zavolá *IsletSelectionViewer* se specifickým typem třídy *Task*, který widgetu přesně řekne, které ovládací prvky jsou na displeji potřeba. Foto: autor, vytvořeno v software DrawIO, dostupné z https://app.diagrams.net/



Obrázek 10: Srovnání screenshotů 2 obrazovek, na každé z nich je sbírán jiný druh dat o stejném obrázku. Obě obrazovky překrývají jednotný widget *IsletSelectionViewer* vlastními prvky: vlevo seskupením radiových tlačítek, vpravo textem a tlačítky pro hodnocení individuálního ohraničení Langerhansova ostrůvku. Foto: autor

5.5.1 Polohování ostrůvků

Při hodnocení kvality jednotlivých Langerhansových ostrůvků je potřeba načíst *PhotoView* knihovnu ve widgetu *IsletSelectionViewer* se specifickou defaultní pozicí na displeji. Pro polohování widgetu *PhotoView* byla využita knihovna *AlignPositioned*, jelikož defaultní parametr *basePosition*[18] ve *PhotoView* se v praxi neosvědčil. *AlignPositioned* umožňuje posunutí PhotoView relativně k displeji mobilu tak, aby se na centru obrazovky zobrazil požadovaný ostrůvek[18].

Výpočet pozice obrázku proto probíhá ve dvou krocích. Nejprve jsou souřadnice středu ostrůvku, který má být v centru obrazovky, převedeny do soustavy souřadnic relativní k velikosti displeje. Následovně je vypočten potřebný posun v této soustavě souřadnic pro widget *AlignPositioned*, aby se dosáhlo potřebného centra obrazovky. Toto lze sumarizovat do následujícího vzorce:

$$x_{displej} = \frac{\text{šířka widgetu}}{\text{šířka obrázku}} * x_{obrázek} * scale \quad (1)$$

Vedle pozice na ostrůvku vypočítává aplikace i přiblížení obrazovky (*scale*), aby ostrůvek pokrýval přibližně celý displej a byl tak pro uživatele znatelný. Zoom hodnoty *n* je ve knihovně *PhotoView* definovaný jako pokrytí celé obrazovky 1/*n*-tinou celého obrázku[18]. Vypočtením zlomku velikosti ostrůvku vzhledem k velikosti celého obrázku lze proto vypočítat koeficient přiblížení obrázku na daný ostrůvek. Jelikož ale velikost ostrůvku není z důvodu úspory úložiště na API přístupná, aproximuje IS ostrůvek ke kruhu. Tato aproximace používá pouze dvě hodnoty u každého ostrůvků: souřadnice centra ostrůvku a šipky k ostrůvku. Vzdálenost mezi centrem ostrůvku a jeho okrajem, který je definovaný pozicí šipky, je tím pádem poloměr kružnice popisující daný ostrůvek. Z toho lze vyvodit následující vzoreček:

$$scale = \frac{\sqrt{(edge_x - center_x)^2 + (edge_y - center_y)^2}}{\sqrt{height^2 + width^2}} \quad (2)$$

Na rovnici 2 *edge* a *center* označují souřadnice k šipce, resp. ke středu ostrůvku. *Height* a *Width* jsou výška, resp. šířka obrázku.

5.5.2 Životní cyklus widgetu *IsletSelectionViewer*

Pokud uživatel hodnotí kvalitu individuálních ostrůvků, má možnost dokreslit vlastní označení, tzv. *Contour edit*. V IS je funkcionalita dokreslování vlastní kontury integrovaná do widgetu *IsletSelectionViewer*, který umožňuje uživatelům překreslovat přes předloženou konturu. Uživatel nejprve hodnotí Langerhansův ostrůvek, a pokud vyjádří nesouhlas s předloženým ohraničením ostrůvku, má za úkol vlastní konturu dokreslit (tento proces je popsán v sekci **5.5.3 Tvorba vlastních kontur**). Tudíž je rozdělen běh widgetu *IsletSelectionViewer* do 3 stavů:

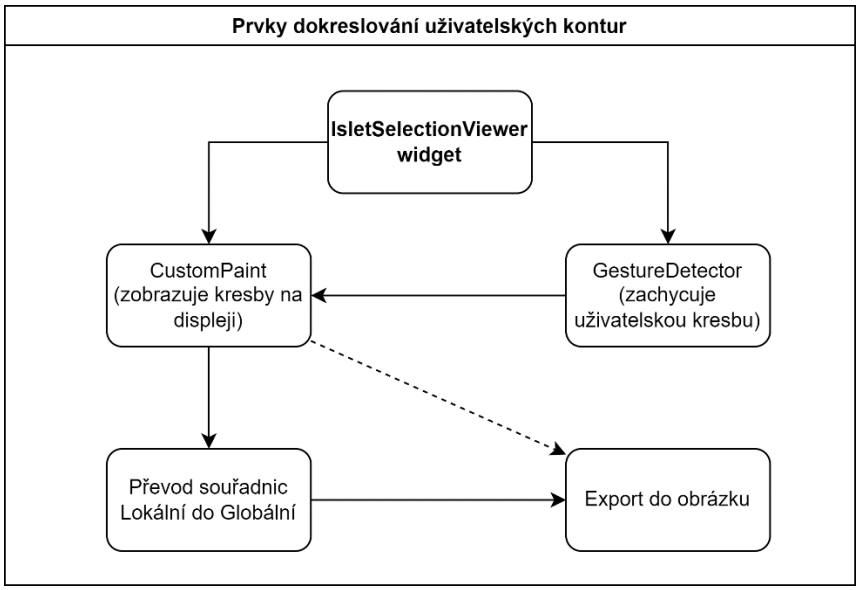
1. *Rating* (hodnocení): V této fázi uživatel pouze hodnotí předložené ohraničení ostrůvku pomocí tlačítek *True*, *False* či *Pass*. Obrázkem může posouvat pomocí gest, nemůže na něj ale vlastní ohraničení dokreslovat.
2. *Zooming* (přibližování): V této fázi uživatel projevil zájem dokreslit vlastní ohraničení ostrůvku, ale s obrázkem nejprve pohybuje, aby jej nastavil do správné pozice pro kreslení.
3. *Editing* (upravování): Nyní si uživatel našel správnou pozici pro dokreslení vlastního ohraničení ostrůvku a sám dokresluje novou konturu pro ostrůvek.

Mezi fázemi přepíná funkce *undo*. Pokud uživatel zvolí hodnocení *False* ve fázi *rating*, je automaticky přesunut do fáze *editing*, ze které může přepnout pomocí tlačítka do fáze *zooming*. Fáze při kreslení kontur byly přidány kvůli převodu souřadnic uživatelských kontur. Pokud by uživatel pohnul s obrázkem mezi kreslením více kontur, změnila by se relativní pozice widgetu *PhotoView* k obrázku. Tím pádem by konverze souřadnic při exportu z lokálních do globálních fungovala pouze pro jednu uživatelskou konturu, nikoli pro všechny.

5.5.3 Tvorba vlastních kontur

Při zvolení hodnocení *False* pro předlohu kontury Langerhansova ostrůvku je uživatel povinen nakreslit vlastní konturu. Jakmile se dostane do fáze *editing* (v sekci **5.5.2 Životní cyklus**

widgetu *IsletSelectionViewer*), dojde k překrytí widgetu *IsletSelectionViewer* widgetem *SelectionPainter*. *SelectionPainter* je založený na Flutter knihovně *CustomPaint*, která zprostředkovává volné kreslení na obrazovku v reálném čase. Tento widget spolupracuje s prvkem zachytávajícím uživatelská gesta na obrazovce, tzv. *GestureDetector*, jehož data využívá pro kresbu čáry na displeji. Propojení těchto widgetů je znázorněno na následujícím diagramu:



Obrázek 11: Diagram znázorňující jednotlivé úrovně, které spolupracují pro vytvoření systému snímání, zobrazování a exportování uživatelských kontur (*Contour edits*). Foto: autor, vytvořeno v software DrawIO, dostupné z https://app.diagrams.net/

Jakmile je uživatel s vlastní konturou spokojen, potvrdí její uložení do databáze. Při uložení je exportován obrázek pouze s nově nakreslenou uživatelskou konturou, který má stejné dimenze jako jeho mateřský snímek vzorku ostrůvku, V tomto procesu jsou lokální souřadnice uživatelské kontury na displeji převedeny do globálních souřadnic relativních k celému obrázku. Proces je inverzní funkcí převodu z globálních na lokální souřadnice, rovnice 1.

5.6 Odesílání uživatelských dat na API

Kvůli co nejlepší offline kompatibilitě se uživatelská data odesílají najednou po dokončení jednoho obrázku. Uživatel dostane možnost stisknout tlačítko „Submit“ na widgetu *IsletTile*, což spustí sekvcenci odesílání dat na API. Následující data jsou přenášena při odesílání na API:

1. Hodnocení individuálních Langerhansových ostrůvků (*True*/*False*/*Pass*)
2. Celkové hodnocení obrázku a konturového snímku
3. Individuální uživatelské kontury u vybraných ostrůvků

Jelikož proces může trvat až několik desítek sekund, probíhá plně asynchronně a hlavní vlákno aplikace volá pouze po dokončení procesu, nebo když narazí na chybu.

6. Aplikace pro označování vzorků – Contour Generator

Před tím, než mohou uživatelé hodnotit označení ostrůvků (kontury), je potřeba patřičná označení vytvořit. Z laboratoří přicházejí zejména dva druhy obrázků pro jeden snímek: základní obrázky (*base images*) a černobílé masky (*masks*) – jejichž role jsou popsány v sekci **2.1 Obecná nomenklatura aplikace.** Pro to je použita aplikace pro tvoření kontur (pracovní jméno *Contour generator*, česky Generátor kontur), která tyto obrázky sloučí a vytvoří z nich obrázek kontury. Obrázek kontury se uživatelům zobrazuje v mobilní aplikaci.

Mimo to slouží aplikace i k nacházení pozic jednotlivých ostrůvků na obrázcích a k jejich přepisu do formy čitelné pro databázi IS. Pro tyto účely byla vytvořena v jazyce Java, který umožňuje systémovou nezávislost na virtualizační platformě JVM (*Java Virtual Machine*)[14] a má mnoho pomocných funkcí pro manipulaci obrázků, zejména třídu *ImageIO*[15].

6.1 Nacházení ohraničení ostrůvků

První funkcí této aplikace je tvorba obrázku kontury, kdy je překryta maska na základní obrázek. Pouhé překrytí obou obrázků ale nemůže fungovat, neboť na masce jsou obrázky znázorněny nejen jako ohraničení, ale i jako celá plocha jimi pokrytá (Obrázek 2). Proto je potřeba tuto vyplňující plochu před překrytím odstranit, čímž zbydou na masce 1px široké hranice Langerhansových ostrůvků.

Pro vymezení hranice iteruje aplikace všemi pixely obrázku. Za pixel ostrůvku je považován ten, který je vybarven bíle, přičemž ostatní jsou vybarveny černě. Hraniční pixely ostrůvků, oproti pixelům jejich vnitřní plochy, mají alespoň 1 sousední pixel, který není bílý (tudíž neoznačuje ostrůvek). Proto aplikace u každého pixelu kontroluje jeho 4 sousední pixely a do dočasného úložiště zapíše data o pouze těch pixelech, které mají méně než 4 bílé sousední pixely. Tím pádem se v úložišti ocitnou jen hraniční pixely ostrůvků, jejichž pozice lze nabarvit bíle na základním obrázku pro získání kontury.

```
int hitCounter = 0; // Registr, který zaznamenává počet sousedních ostrůvků, které jsou bílé
if (i != 0 && j != 0) {
        // Kontrola všech sousedních pixelů. Při zjištění bílého pixelu dojde k inkrementaci hod-
        noty hitCounter
        hitCounter = (mask.getRGB(i + 1, j + 1) == markedColor.getRGB()) ? hitCounter + 1 :
        hitCounter;
        hitCounter = (mask.getRGB(i + 1, j) == markedColor.getRGB()) ? hitCounter + 1 :
        hitCounter;
        hitCounter = (mask.getRGB(i, j + 1) == markedColor.getRGB()) ? hitCounter + 1 :
        hitCounter;
        hitCounter = (mask.getRGB(i - 1, j - 1) == markedColor.getRGB()) ? hitCounter + 1 :
        hitCounter;
}
if (hitCounter > 1 && hitCounter < 4) { // Pokud pixel není kompletně obklopený bílými pixely,
je překryt na základní obrázek
            base.setRGB(i, j, markedColor.getRGB());
}
```

Obrázek 12: Kód, který zajišťuje ukládání pouze hraničních pixelů. Tyto pixely jsou následně překryty na základní obrázek, čímž vznikne obrázek kontury.

6.2 Automatické vybírání ostrůvků

Ve webové aplikaci probíhá výběr pozic ostrůvků a šipek k nim buď manuálně (sekce **7.3.1 Manuální nahrávání ostrůvků**), nebo automaticky – bez interakce ze strany uživatele. K automatickému vybírání ostrůvků se používá generátor kontur, který data o vybraných ostrůvcích odsílá do databáze IS. Generátoru kontur je přidělena složka obsahující základní obrázky (*base images*), jejich odpovídající masky a CSV soubor, který odkazuje na jednotlivé páry maskek a základních obrázků.

Contour Generator nejprve načte podle CSV souboru všechny páry obrázků a pro každý vygeneruje odpovídající soubor kontury (dle procesu ze sekce **6.1 Nacházení ohraničení ostrůvků**). Poté na obrázcích identifikuje všechny Langerhansovy ostrůvky a vypočítá jejich centrální a okrajové body. Následně generátor kontur vybere vzorek ostrůvků pomocí semi-randomizované metody. Jakmile je tento proces dokončen, data o vybraných ostrůvcích na každém obrázku jsou shromážděna do JSON souboru, který je odeslán do webové aplikace pro načtení do databáze.

6.2.1 Vstup pomocí CSV souboru

CSV soubor spojuje polohy základních obrázků (*base images*) s maskami (*masks*). Jelikož tento soubor je manuálně dodáván administrátorem IS, užití formátu CSV, který lze tvořit v software jako Microsoft Excel®, se ukázalo vhodnější než spíše programátorský formát JSON. Kromě dat o obrázcích obsahuje CSV soubor i sloupec *pixel_size*, který určuje délku a šířku 1 pixelu na daném obrázku v mikrometrech. Tato hodnota se mění v závislosti na původu vzorku a způsobu jeho zobrazování. Přestože IS hodnotu *pixel_size* nevyužívá, je stejně zapisována pro orientaci administrátorů a případné statistické účely. *Pixel_size* totiž hraje důležitou roli v odborném

posuzování vzorků, neboť umožňuje převod abstraktní jednotky „čtverečních pixelů“ na reálnou jednotku čtverečních mikrometrů.

6.2.2 Ukládání obrázků do paměti

Jelikož aplikace musí pro nacházení ostrůvků, tvoření kontur apod. porovnávat obrázky s jejich maskami, bylo třeba vytvořit metodu, která by umožnila rychlé čtení a zapisování dat pixelů na obrázcích masek v uspokojivé algoritmické obtížnosti.

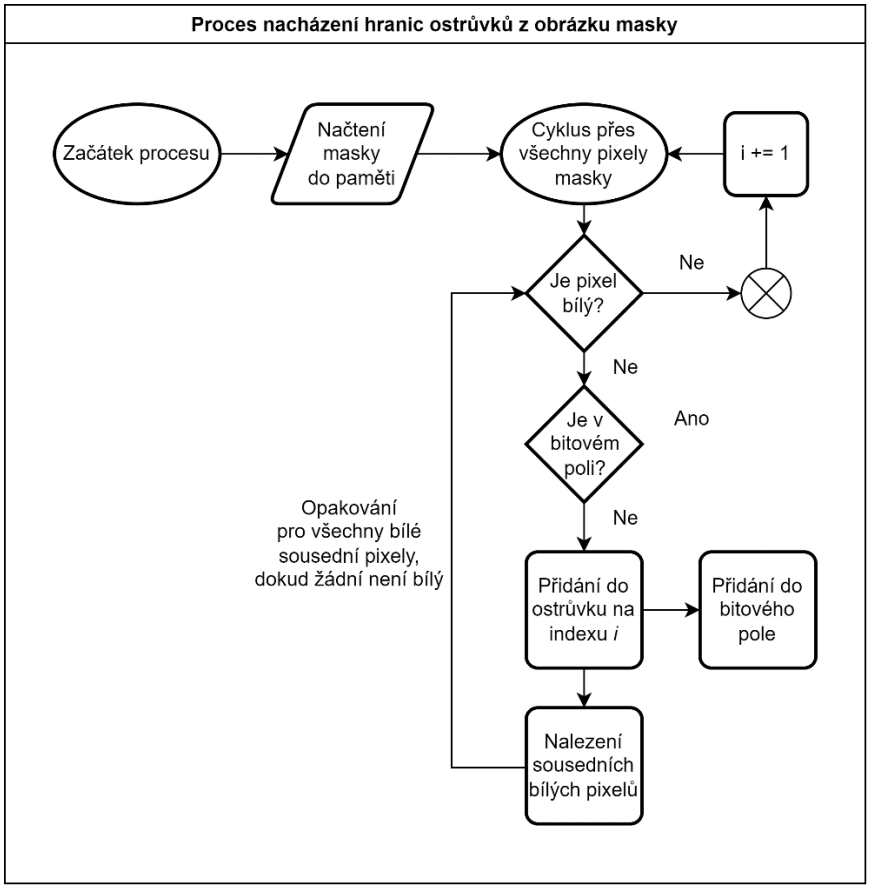
To jsme vyřešili pomocí bitových polí, které mají počet indexů $n = šířka\ obrázku * výška\ obrázku$. Pole mají datový typ *boolean[]* a velikost každého z jejich prvků by měla odpovídat 1 bitu[20]. Do tohoto pole jsou postupnou iterací obrázku masky zapsána data o bílých a černých pixelech, kde pixelu se souřadnicemi x a y je přiřazen v poli index $i = (y * šířka\ obrázku) + x$. Celé pole je inicializováno s hodnotami 0 (*false*), tedy indikujícími černé pixely. Zapsání hodnoty 1 (*true*) v poli reprezentuje bílý pixel. Tímto způsobem lze data každého pixelu zapisovat i číst s konstantní algoritmickou obtížností $\mathcal{O}(1)$[21].

6.2.3 Identifikace ostrůvků na obrázku masky

Obrázky masky jsou černobílé, kde bílá indikuje ostrůvek a černá tkáň mezi ostrůvky, která je pro účely IS nepodstatná. Jeden ostrůvek je vždy spojitá plocha bílých pixelů, která je kompletně ohraničena černými pixely. Tudíž izolace jednotlivých bílých regionů z obrázku umožní identifikaci všech ostrůvků. Pro sledování ostrůvků je vytvořen Java seznam (*ArrayList*). Do něj zapisuje aplikace všechny body přináležející k ostrůvkům, každý z nich vymezený v seznamu jiným indexem.

Toho je docíleno pomocí algoritmu založeném na postupné expanzi z jednoho bílého pixelu na všechny sousední. Jakmile algoritmus identifikuje na obrázku pixel ostrůvku (tedy bíle zbarvený), bude se snažit najít všechny jeho sousedy, bílé pixely (fáze expanze). Pixely jsou přidávány vždy do jednotného indexu seznamu ostrůvků. Jakmile nastane iterace, kdy nenajde algoritmus žádný nový sousední pixel, který by byl bílý, přejde cyklus na jiný pixel na obrázku a inkrementuje index aktuálního ostrůvku.

Jelikož ale algoritmus může ve dvou oddělených iteracích narazit na různé pixely stejného ostrůvku, docházelo by k opakovanému přidávání stejného ostrůvku do registru. Pro prevenci tohoto problému je před spuštěním procesu vytvořeno bitové pole o velikosti $n = šířka\ obrázku * výška\ obrázku$ (podobně jako v sekci **6.2.2 Ukládání obrázků do paměti**), ovšem kompletně prázdné. Jakmile je bílý pixel na obrázku masky při expanzním algoritmu nalezen, je bit na indexu odpovídající pozici pixelu nastaven na hodnotu 1. Před tím, než expanzní algoritmus přidá další pixel do ostrůvku, zkontroluje v bitovém poli reprezentujícím masku index $i = (y * šířka\ obrázku) + x$, jestli již nemá hodnotu 1, a tudíž neoznačuje pixel, který nenáleží již přidanému ostrůvku. Jelikož přístup a psaní do pole má konstantní algoritmickou obtížnost, je tato operace nenáročná na procesor[21].



Obrázek 13: Diagram, který popisuje mechanismus identifikace ostrůvků. Po načtení masky do paměti přechází aplikace na expanzní algoritmus („cyklus přes všechny pixely masky“). Foto: autor, vytvořeno v software DrawIO, dostupné z https://app.diagrams.net/

Jakmile skončí iterace přes všechny pixely masky v cyklu *for*, odešle metoda nalezené ostrůvky ve formě *ArrayList*.

6.2.4 Výpočet center a šipek k ostrůvkům

Po identifikaci ostrůvků musí aplikace na každém z nich vypočítat pozice centra ostrůvku a šipky k ostrůvku pro jeho označení v mobilní aplikaci IS. Každý ostrůvek je po fázi identifikace (sekce **6.2.3 Identifikace ostrůvků na obrázku masky**) převeden do třídy typu *IsletSelection* a data o jeho centru a šipce jsou do této třídy poté připsána.

Výpočet centra ostrůvku probíhá aproximací rozdílu maximálních a minimálních souřadnic dceřiných bodů ostrůvku. Při detekci ostrůvků aplikace iteruje přes každý pixel daného ostrůvku, a při tom zapisuje maximální a minimální hodnotu souřadnic x a y do paměti. Z toho je vypočteno centrum ostrůvku (rovnice 3):

$$centrum = \frac{max - min}{2} \tag{3}$$

Přestože bod šipky může být *de facto* jakýkoli okrajový bod ostrůvku, pro univerzalitu aplikace používá body, které se vyskytují v levém horním rohu každého ostrůvku. Aplikace v cyklu *while* postupně odečítá od souřadnic centra ostrůvku *x* a *y* hodnotu 1, dokud nenarazí na pixel, který leží mimo ostrůvek. Tímto postupuje diagonálně ve směru do levého horního rohu ostrůvku. Jakmile narazí na pixel, který leží mimo ostrůvek, označí poslední pixel ostrůvku za pozici pro šipku k ostrůvku.

6.2.5 Vybírání reprezentativního vzorku ostrůvků

Před posláním dat o ostrůvcích do databáze IS se vybírají jen některé ostrůvky pro hodnocení, protože by bylo nerealistické chtít po uživatelích hodnotit více než 30 ostrůvků na každém obrázku. K tomuto účelu používá IS polo-randomizační techniku. Výstupem je uživatelem zvolený počet ostrůvků, které odpovídají ve stejných proporcích různým velikostním kategoriím na obrázku.

Ostrůvky jsou nejprve rozděleny do kategorií podle velikosti, čímž jsou přiděleny třídám *IsletCategory.* Třída *IsletCategory* obsahuje data o intervalu velikostí ostrůvků, které do ní mají být vloženy, například rozsah velikostí 50µm-60µm. *IsletCategory* jsou poté seřazeny a ostrůvky v nich přeházeny pomocí randomizační funkce *Collections.shuffle*. Aplikace dále distribuuje počet ostrůvků co nejrovnoměrněji po všech velikostních kategoriích. Počet ostrůvků pro každou kategorii probíhá následovně (rovnice 4; n_i – počet ostrůvků, n_k – počet kategorií):

$$n = \left\lfloor \frac{n_i}{n_k} \right\rfloor \tag{4}$$

Po provedení distribuce dle rovnice 4 nemusí být dosaženo požadovaného počtu ostrůvků, pokud neplatí následující (rovnice 5; n_i – počet ostrůvků, n_k – počet kategorií):

$$n_i \bmod n_k = 0 \tag{5}$$

Proto aplikace u některých kategorií vybere o 1 ostrůvek více při randomizačním procesu, čímž dosáhne požadovaného počtu ostrůvků n_i.

Celý proces vybírání je zprostředkováván Java strukturami *streams*, které dovolují filtrování na seznamech a polích, což dodává IS schopnost dynamicky rozdělovat seznam všech ostrůvků podle jejich velikostí. Následující výtažek z kódu ukazuje, jak aplikace distribuuje ostrůvky do kategorií:

```
for (IsletImage image : images) {
        List<IsletCategory> imageCats = categories;
        List<IsletSelection> imageSelections = new ArrayList<>();
        for (IsletCategory cat : imageCats) { // Iterace přes všechny kategorie ostrůvků
          List<IsletSelection> selections = cat.childSelections;
          Collections.shuffle(selections); // Randomizace: ostrůvky jsou v kategorii proházeny

          // Odstranění ostrůvků za kategorie, které nepatří tomuto obrázku
          List<IsletSelection> matched = selections.stream()
                .distinct()
                .filter(image.getSelections()::contains)
                .collect(Collectors.toList());

          // Limitování počtu přidaných ostrůvků. Ty jsou poté přiděleny instanci IsletImage
          imageSelections.addAll(matched.subList(0, categorySizes[sizeCounter]));
          sizeCounter++;
        }
        sizeCounter = 0;

       // Přidělení ostrůvků k instanci IsletImage
        image.setSelections(imageSelections);
}
```

Obrázek 14: Kód ukazující proces výběru ostrůvků z kategorií pomocí randomizační metody *Collections.shuffle*.

6.3 Komunikace mezi generační aplikací a webovým serverem

Webová aplikace komunikuje s generátorem kontur pomocí příkazů, tedy *Commands*. Jelikož webová aplikace posílá dlouhé příkazy, nelze jejich obsah vždy přenášet pomocí konzolových argumentů, a tak jsou zapisovány do JSON souborů, které generátor kontur čte.

Aplikace pro každý požadavek vyžadující užití generátoru kontur vytváří soubor JSON s unikátním názvem, tzv. *command ID*. Generátor kontur toto *command ID* použije pro označení souboru s výsledky analytické operace, který nazve *export_commandID.json*. *Command ID* obsahuje mimo jiné unikátní identifikátor z PHP funkce *uniqid()*, což zamezí potenciální situaci, kdy dva nebo více uživatelů zavolá generátor kontur najednou a aplikace neví, který příkazový soubor použít pro vstup dat. Formát JSON byl zvolen kvůli své univerzalitě a schopnosti převodu do datových typů polí a objektů jak v PHP, tak v Javě.

7. Webová aplikace IsletSwipe

Webová aplikace založená na PHP frameworku Symfony je bodem průniku všech prvků systému IS. Dochází zde k propojení mobilních klientů s databází skrze API a s administrátory skrze uživatelské rozhraní webové aplikace. Webová aplikace má také oprávnění volat generátor kontur (**sekce 6. Aplikace pro označování vzorků – Contour Generator**) pro případnou manipulaci s obrázky v databázi. API funguje paralelně s administrací na jednom webovém serveru a přistupuje k jednotnému databázovému serveru. Administrace ovšem pro přístup do databáze API nevolá, nýbrž pomocí rozhraní *EntityManager* se do databáze připojuje přímo. API

tudíž slouží pouze jako limitované zpřístupnění vybraných operací na databázi pro určené klienty, tedy uživatele mobilní aplikace.

Celkově obsahuje webová administrace následující oddíly:

1. Správa uživatelských účtů: Zabývá se zvaním nových uživatelů a odstraňování těch stávajících, také přiřazováním či odebíráním uživatelů od uživatelských skupin.
2. Správa obrázků a sad: Zde dochází k přidávání či odebírání obrázků ze systému IS a jejich připojování k sadám.
3. Statistický přehled: Slouží k zobrazování statistik spojených s uživatelským hodnocením obrázků a ohraničení ostrůvků.
4. Správa odměn: Tato sekce je užívána k přidávání/odebírání unikátních GIF souborů, které se zobrazují uživatelům mobilní aplikace jako odměny (sekce 5.2 Přístup k hodnocení obrázků)

7.1 Zabezpečení, autentizace a autorizace

Symfony poskytuje v PHP pro účely IS dostačující nadstavbu pro autentizaci uživatelů v reálném čase. IS používá PHP *Session cookies* v hlavičce *Set-cookie* pro unikátní identifikaci uživatelů a přidělení oprávnění. Klient API či webová aplikace nejprve zavolají stránku pro přihlášení, která je pro API endpoint */api/login* a pro webové frontend rozhraní */login*. Po úspěšném zadání údajů je mu vrácena v hlavičce *Session cookies*, kterou používá při všech ostatních požadavcích pro prokázání své identity, dokud platnost *cookies* nevyprší. Procesy autentizace jsou definovány v konfigurační třídě *UserAuthenticator*, která dědí funkcionality třídy *AbstractFormLoginAuthenticator*. Tímto způsobem těží aplikace z defaultního Symfony autentizačního systému, a zároveň definuje vlastní parametry pro autentizaci.

Pomocí *Session cookie* lze při běhu webové aplikace kdykoli extrahovat osobní data uživatele pro možnou funkcionalitu webu. Jelikož logika autorizace, tedy udělování či zamítání přístupu na jednotlivé stránky podle oprávnění uživatele, je v IS relativně jednoduchá, nebylo potřeba zavézt systém Symfony *Voters*, které se pro komplexnější autorizaci často používají[12]. Webová aplikace je přístupná, až na výjimku uživatelského profilu a stránky pro přihlášení, pouze pro administrátory, a tudíž ji lze zabezpečit přímo v konfiguračním souboru *security.yaml*. V *security.yaml* totiž IS omezuje všechen přístup na URL s prefixem *admin* pouze pro administrátory.

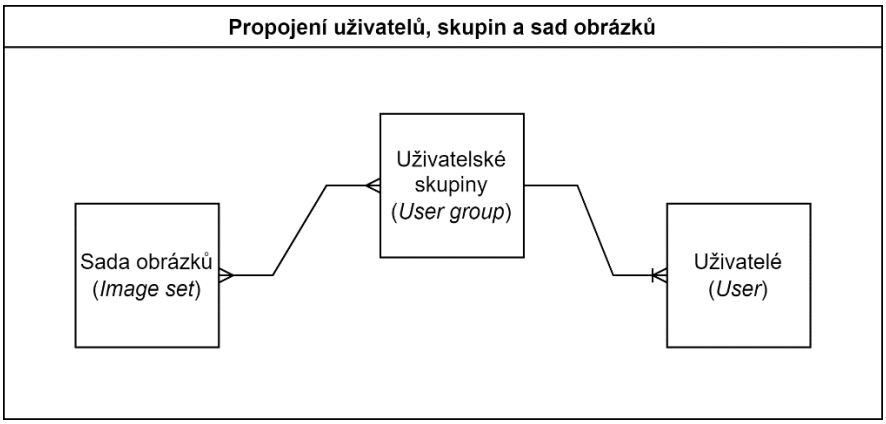
7.2 Správa uživatelských účtů a skupin

Administrátoři mají ve webové aplikaci IS možnost vytvářet či odstraňovat uživatele a přidělovat jim role buď administrátora (*ROLE_ADMIN*) nebo běžného uživatele (*ROLE_USER*). Při vytváření uživatele administrátor zadává pouze jejich jméno a email, na který aplikace pomocí systému *SwiftMailer* pošle email s pozvánkou. V pozvánce jsou patřičné přihlašovací údaje pro uživatele, včetně náhodně vygenerovaného hesla, a odkazy na stažení mobilní aplikace IS.

Jelikož ekosystém IS je uzavřený pouze pro pozvané uživatele, jsou pozvánky praktičtější než otevřená registrace, neboť tímto způsobem mají administrátoři plnou kontrolu nad zvaním nových uživatelů ke spolupráci v IS.

Po zadání uživatelských dat renderuje webová aplikace Twig šablonu s emailem do HTML souboru, který je čitelný pro emailové klienty. Knihovna *SwiftMailer* se následovně připojí na SMTP server IS, pomocí kterého odešle email na patřičnou emailovou adresu uživatele. Pro lepší organizaci šablony zprávy využívá IS šablonového frameworku *Inky*[20], který je nadstavbou jazyka Twig.

Uživatelé jsou při přidávání také připsáni do uživatelské skupiny. Tyto skupiny identifikují nejčastěji jednotlivé laboratoře a slouží k rozdělení aktivace sad (sekce **7.4 Správa sad**). Skupiny jsou uloženy v databázi podle datové jednotky *UserGroup*, která definuje relaci mezi uživateli a skupinami jako ManyToOne a přiděluje skupinám jméno a zkratku.



Obrázek 15: Diagram propojení uživatelů, uživatelských skupin a sad obrázků. Sada obrázků může být aktivována buď globálně pro všechny uživatele, nebo jen pro cílové skupiny uživatelů. Foto: autor, vytvořeno v software DrawIO, dostupné z https://app.diagrams.net/

7.3 Správa obrázků

Schopnost efektivního přidávání a manipulace s obrázky je zásadní pro plynulou funkčnost celého systému IS. Proto existují dvě možnosti pro přidávání obrázků do systému: manuální nebo automatické. Automatické přidávání využívá generátoru kontur, kterému specifikuje pouze obrázky, na kterých mají být ostrůvky vybrány. Tento proces byl popsán v sekci **6.1 Nacházení ohraničení ostrůvků**. Při manuálním přidávání naopak musí administrátor sám na obrázku vybrat potřebné ostrůvky ke zhodnocení.

7.3.1 Manuální nahrávání ostrůvků

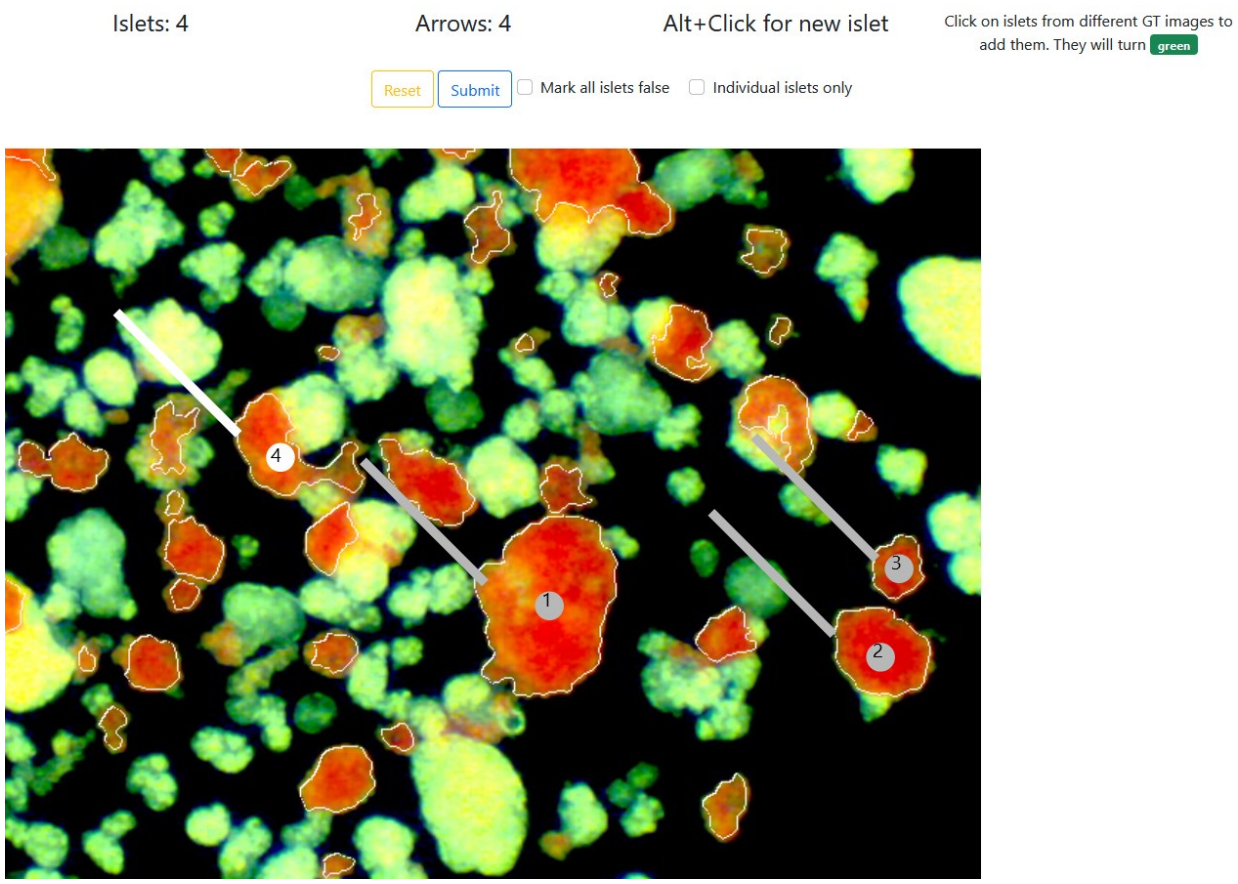
Velká část uživatelské interakce při nahrávání obrázků a vybírání ostrůvků je naprogramovaná ve skriptovacím jazyce JavaScript. Aplikace nejprve využívá knihovny *Dropzone.JS* pro zprostředkování odesílání základního obrázku a masky na server. Tam jsou pro ně vytvořeny nové datové jednotky (*Entities*), které jsou zapsány do databáze a napojeny na společnou

jednotku *ImageHolder*, která všechny obrázky daného vzorku spojuje. Po obdržení základního obrázku i masky webová aplikace automaticky zavolá generátor kontur, který vytvoří obrázek kontury. Kontura je přiřazena do datové jednotky *IsletContour* a spolu s ostatními obrázky je nahrána na server. Po dokončení tohoto kroku má již aplikace všechny potřebné obrázky pro vytvoření kompletního přehledu o daném vzorku.

Následuje tudíž vybírání Langerhansových ostrůvků pro šetření v rámci mobilní aplikace IS. Administrátoři vybírají co nejméně ostrůvků na každém obrázku, a proto musí mít možnost si přiblížit na jednotlivé části obrázku a identifikovat tak nejpravděpodobnější body sporů, které by IS mohl vyřešit. U každého ostrůvku jsou důležité dvě hodnoty: pozice centra ostrůvku a pozice šipky k ostrůvku. Nejprve je obrázek kontury načten v HTML pro zobrazování v prohlížeči a je poslán jako argument do knihovny *Panzoom*, která umožňuje interakci s obrázkem pomocí myši. Administrátor tak může obrázek přibližovat či oddalovat a posouvat na něm po libovolných ostrůvcích.

Administrátor vybírá šipky a ostrůvky pomocí kliknutí myši. EventHandler *onClick* v knihovně *panzoom* s každou událostí kliknutí zasílá i pozici myši relativně k zobrazovanému obrázku, což umožňuje výpočet pozice kliknutí relativně k originálnímu souboru obrázku, nikoli relativně k prohlížeči. Tato pozice je zapsána do pole v JavaScriptu jako objekt ostrůvku (*Islet*) nebo šipky (*Arrow*). Jak šipky, tak ostrůvky samotné jsou identifikovatelné pomocí číselného označení *ID*. Při každém kliknutí se střídá právě přidávaný objekt: po přidání pozice centra ostrůvku se přidává šipka, atd.

Pro přehled při vybírání ostrůvků jsou vybrané ostrůvky také znázorněny na obrazovce. K tomu užívá IS segmentu SVG (*Scalable vector graphics*), který je pomocí CSS (*Cascading style sheets*) parametru *position: absolute* překryt přes obrázek kontury. SVG segment obsahuje kolečka reprezentující centra ostrůvků a přímky znázorňující šipky k ostrůvkům.



Obrázek 16: Screenshot webové aplikace ve fázi volení ostrůvků na novém obrázku. Přímky korespondují k šipkám a kruhy k centrům ostrůvku. Každý ostrůvek je pro přehlednost očíslovaný pomocí vlastního ID, které je uloženo ve třídě *Islet* či *Arrow*. Šedivé objekty ukazují na ostrůvky přidané na obrázku se stejným základním obrázek, ale odlišnou maskou. Foto: autor

Po potvrzení výběru ostrůvků pomocí tlačítka submit (Obrázek 16, nahoře uprostřed) jsou data o ostrůvcích odeslána spolu s *ID* upravovaného obrázku na URL */admin/selections/new*. Data jsou přenášena ve formátu JSON pomocí JQuery metody AJAX (*Asynchronous JavaScript and XML*). V PHP ovladači jsou data o ostrůvcích převedena do instancí datové jednotky *IsletSelection* a asociována se svým mateřským obrázkem. Tyto změny reflektuje aplikace v databázi a odešle JavaScript programu potvrzení o úspěchu požadované akce.

7.3.2 Přejímání ostrůvků z jiných obrázků

Možné využití IS je i porovnání správnosti různých masek na stejném základním obrázku. Tyto obrázky mají stejné jméno, ale liší se v identifikátoru tzv. GT (*Ground truth*), který označuje druh masky. Pro rychlé srovnání stejných ostrůvků na různých maskách byla tudíž přidána možnost rychle zkopírovat ostrůvky mezi obrázky se stejným *base image*. Před renderováním obrázku zavolá PHP ovladač databázi a vyžádá pomocí Doctrine *EntityManager* všechny ostrůvky, které jsou na obrázcích se stejným jménem, ale odlišným GT, jako je právě upravovaný obrázek. Ostrůvky jsou protříděny PHP funkcí *array_unique*, která z nich odstraní duplicitní prvky.

Při renderování obrázku postupuje JS script s těmito ostrůvky stejně jako s uživatelsky přidanými, tedy vygeneruje pro ně patřičné SVG prvky a přidá je do paměti. Tyto „defaultní“ ostrůvky jsou ukládány ale v dočasném poli, které není po potvrzení výběru odesílané na server. Pokud uživatel na defaultní ostrůvek klikne, *EventHandler* determinuje, že se jedná o defaultní ostrůvek, a přesune jej z pole neaktivních ostrůvků na ostrůvky, které mají být přidány na právě upravovaný obrázek.

Tímto způsobem mohou administrátoři jednoduše vytvořit kopii jednoho obrázku s odlišným GT pro determinaci rozdílu mezi správností jednotlivých masek. Jakmile jsou ostrůvky na obrázku vybrány, je tvorba informací o obrázku dokončena a lze jej poslat na mobilní aplikaci IS.

7.4 Správa sad

Obrázky mohou vznikat během oddělených separací ostrůvků, tzv. izolací, a tudíž se ukázalo logické je seskupovat do sad. Datová jednotka *ImageSet* má relaci *OneToMany* k jednotkám *IsletImage*, čímž je propojena se svými dceřinými obrázky. *ImageSet* může nabývat několika stavů: může být tzv. současný („*current*“), aktivovaný („*activated*“) nebo archivovaný („*archived*“). *Current*, *activated* a *archived* jsou sloupce v databázové tabulce *ImageSet* a podle jejich hodnot může Doctrine *EntityManager* filtrovat sady dle aktuálního stavu.

Pokud uživatel přidává nový obrázek do aplikace, je tento obrázek automaticky přiřazen do sady ve stavu „current“. Současnou sadu aplikace extrahuje z databáze pomocí specifického DQL (*Doctrine Query Language*) příkazu.

Jakmile je sada připravená pro ohodnocení uživateli v mobilní aplikaci, administrátor ji může aktivovat, načež systém aktualizuje její záznam v databázi. Aktivace sady je ale vázaná i na specifické uživatelské skupiny, neboť některé laboratoře mohou např. požadovat separátní hodnocení vlastními odborníky než všemi uživateli IS. Proto má datová jednotka pro *ImageSet* relaci ManyToMany s *UserGroups*, aby mohla být aktivace sady specifikována dle potřeby pro některé uživatelské skupiny.

Po dokončení hodnocení uživateli je sada přesunuta administrátory manuálně do stavu „archivace“, kdy jsou její statistická data na serveru ponechána, ale uživatelům mobilní aplikace se již nezobrazuje.

Pokud je sada ve stavu „current“ nebo „activated“, administrátor ji nemůže odstranit. Pokud je sada aktivovaná, nelze ani modifikovat její obsah. Toto slouží jako preventivní prostředek před nevyžádanými změnami na aktuálně zkoumaných sadách. Sadu lze odstranit jen tehdy, kdy je ve stavu „archivace“.

7.5 Automatické nahrávání ostrůvků

Alternativní metodou k manuálnímu vybírání ostrůvků na obrázku je jejich automatická identifikace a polo-automatické vybírání pomocí softwarové metody. Toto je zprostředkováno aplikací generátoru kontur, což bylo již popsáno v sekci **6.2 Automatické vybírání ostrůvků**. Z pohledu uživatele dochází k automatickému výběru kontur na jediné obrazovce ve webové aplikaci. Uživatel na této obrazovce nahrává ZIP archiv s přesně danou strukturou, který obsahuje vše potřebné pro generátor kontur. Struktura ZIP souboru je zmíněna v informačním okénku na stránce a je následující:

- Složka *images*/ - v ní jsou uloženy základní obrázky
- Složka *masks*/ - v ní jsou uloženy masky
- CSV soubor s libovolným názvem – tabulka shromažďující informace o obrázcích v archivu, která propojuje jednotlivé masky s jejich základními obrázky a udává u nich parametr *pixel_size* (více rozvedeno v sekci **6.2.1 Vstup pomocí CSV souboru**).

Okno pro nahrávání souboru je vytvořeno a stylizováno s pomocí knihovny *Dropzone.JS*, která zavolá po nahrání souboru JQuery AJAX, při kterém dojde k přenosu dat nahraného souboru na ovladač na serveru. V ovladači *ImageAutomaticUploadController* dochází nejprve k rozbalení ZIP souboru pomocí modulu *ext-zip*. Cesta k extrahovanému souboru je poté poslána ve třídě *Process* jako argument generátoru kontur, který vytvoří kontury pro všechny páry masek a základních obrázků a vybere u každé z nich ostrůvky.

```
$open = $archive->open($dest . $zip, \ZipArchive::CREATE);
    if ($open === TRUE) { // Operace pokračuje, pokud je archiv úspěšně otevřen
        $archive->extractTo($dest . $name); // Extrakce obsahu archivu
        @$archive->close();
    } else {
        throw new NotFoundHttpException($open);
    }
```

Obrázek 17: Část kódu, která rozbaluje ZIP soubor na serveru IS. Soubor je nejprve otevřen, jeho formát je ověřen interně pomocí funkce *$archive->open()* a data uchovaná v něm aplikace poté extrahuje.

Jednotlivé kroky webová aplikace volá jako separátní AJAX požadavky na server, čímž může pomocí *ProgressBaru* sledovat průběh celkového výběru ostrůvků. Jakmile je proces dokončen, data ostrůvků a obrázků jsou konvertována do patřičných datových jednotek a propojena s novou sadou obrázků.

7.6 Správa uživatelských odměn

Nejmenší sekcí webové aplikace je administrace odměn pro uživatele. Tyto odměny si mobilní aplikace vyžádá po dokončení balíčku tři obrázků přes API. Odměny mají dvě části, které jsou nahrány na server a propojeny v databázi přes datovou jednotku *Award*: soubor obrázku (animované GIF či PNG nebo JPG) a audio soubor (ve formátu MP3).

Jednotka odměn má v tři endpointy v API: */api/award*, */api/award/audio* a */api/award/image*. První endpoint dodává mobilní aplikaci obecné informace jako motivační text nebo formát obrázku, zatímco druhý a třetí odesílá data samotného obrázku, resp. audio souboru pro stažení na mobilní zařízení.

8. Závěr

Systém IS je souhrnný nástroj pro zisk, zpracovávání a zobrazování dat o uživatelském konsenzu nad snímky Langerhansových ostrůvků. V každém článku IS je kladen důraz na metodičnost a interaktivitu jednotlivých prvků pro maximalizaci způsobů, jak uživatelé mohou projevit svůj názor na daný snímek. V rámci systému jsme proto dosáhli bilance mezi rozmanitostí metod získávání dat a schopností kvantifikovat uživatelská data pro automatizovanou statistickou analýzu. Kombinací toho je sada aplikací jednoduchých na použití, ale schopných extrahovat různorodá data pro evaluaci vzorků Langerhansových ostrůvků. Při setkání s některými uživateli aplikace vyvstala často jednoduchost použití jako stěžejní bod pro úspěch aplikace, čehož bylo dosaženo především pomocí mnoha interaktivních prvků v mobilní aplikaci za použití frameworku Flutter.

Pro tvorbu spolehlivého systému pro sběr a analýzu dat je ovšem potřeba data sbírat s co nejvyšší přesností. V současnosti v rámci IS pracujeme na vývoji speciální statisticko-analytické metody pro kvantifikaci odchylky uživatelské kresby od předlohy. Tímto způsobem lze podložit či případně vyvrátit tvrzení o přesnosti mobilní aplikace IS při tvorbě vlastních uživatelských kontur. Jelikož označování Langerhansových ostrůvků je precizní práce, která netoleruje vysokou míru chybovosti, vidíme jako nutné ověřit schopnosti aplikace v dokreslování vlastních ohraničení před tím, než by tato ohraničení byla využita pro síť IsletNet.

Testování ukázalo, že IS je schopen plnit úkoly, které povedou k vylepšení kvality pravdy pro neuronovou síť IsletNet. Během vývoje bylo odstraněno mnoho chyb a nedostatků a nyní se při běhu aplikace jedná u chyb spíše o anomálie než systematické chyby. Přesto aplikaci schází důležité systémy jako kompletní analýza uživatelských kontur či notifikace pro efektivnější komunikaci mezi administrátory a uživateli. Proto je IS pořád otevřený projekt, přestože hlavní část jeho funkcionalit je již dokončená.

Dopad IS nemusí být limitován jen na neuronovou síť IsletNet. Metoda transplantace Langerhansových ostrůvků pro léčbu Diabetu typu 1 totiž postupně proniká do více světových transplantačních center[2]. Tato procedura s sebou ale nese řadu problémů, které omezují její aplikovatelnost na širokou škálu diabetologických pacientů[2]. IS zde stojí především jako element v řadě nástrojů pro pomoc při společném vylepšování transplantační technologie. Jelikož je systém založen na uživatelské součinnosti a chtění pomoci při společném technologickém postupu, má potenciál i ve stmelování globální „ostrůvkářské“ komunity. IS slouží totiž jako

ověřovací prvek v dlouhém řetězu zobrazovacích technologií a analytických technik, a jeho širší použití pravděpodobně vzbudí debaty nad kvalitou současných technologií, které se podílejí na procesu transplantace Langerhansových ostrůvků. Proces vedoucí ke snímku Langerhansových ostrůvků totiž obsahuje mnoho kroků a výsledná kvalita snímku proto záleží např. na rozlišení mikroskopu nebo barvení tkáně. A právě IS uživatelům dává možnost tyto kroky a technologie srovnávat, čímž dostává aspekt sociální sítě. Jelikož transplantace Langerhansových ostrůvků je globální snahou k efektivnější léčbě Diabetu typu 1, vidíme tento aspekt srovnávání jako vhodný přídavek do „ostrůvkářské“ komunity, který může vézt k vývoji efektivnějších a lepších nástrojů.

Výsledkem projektu je kompletní systém s potenciálem do budoucnosti. Vývoj na IS pokračuje a směřuje jak k vylepšení současných nedostatků, tak ke přidání žádaných funkcionalit jako notifikací nebo převodu obrázků do formátu SVG. Systém je v současnosti schopen přijímat a zpracovávat uživatelská data a propojovat mnoho diabetologických center dohromady. Navazující vývoj systému je tudíž směřovaný primárně na uhlazování uživatelského rozhraní a vylepšování funkcionalit podle poptávky administrátorů nebo expertů v oblasti diabetologie. Výsledky projektu IS a evaluace jeho funkčnosti pro účely sítě IsletNet jsou plánovány pro publikaci na jaře 2022.

9. Seznam obrázků

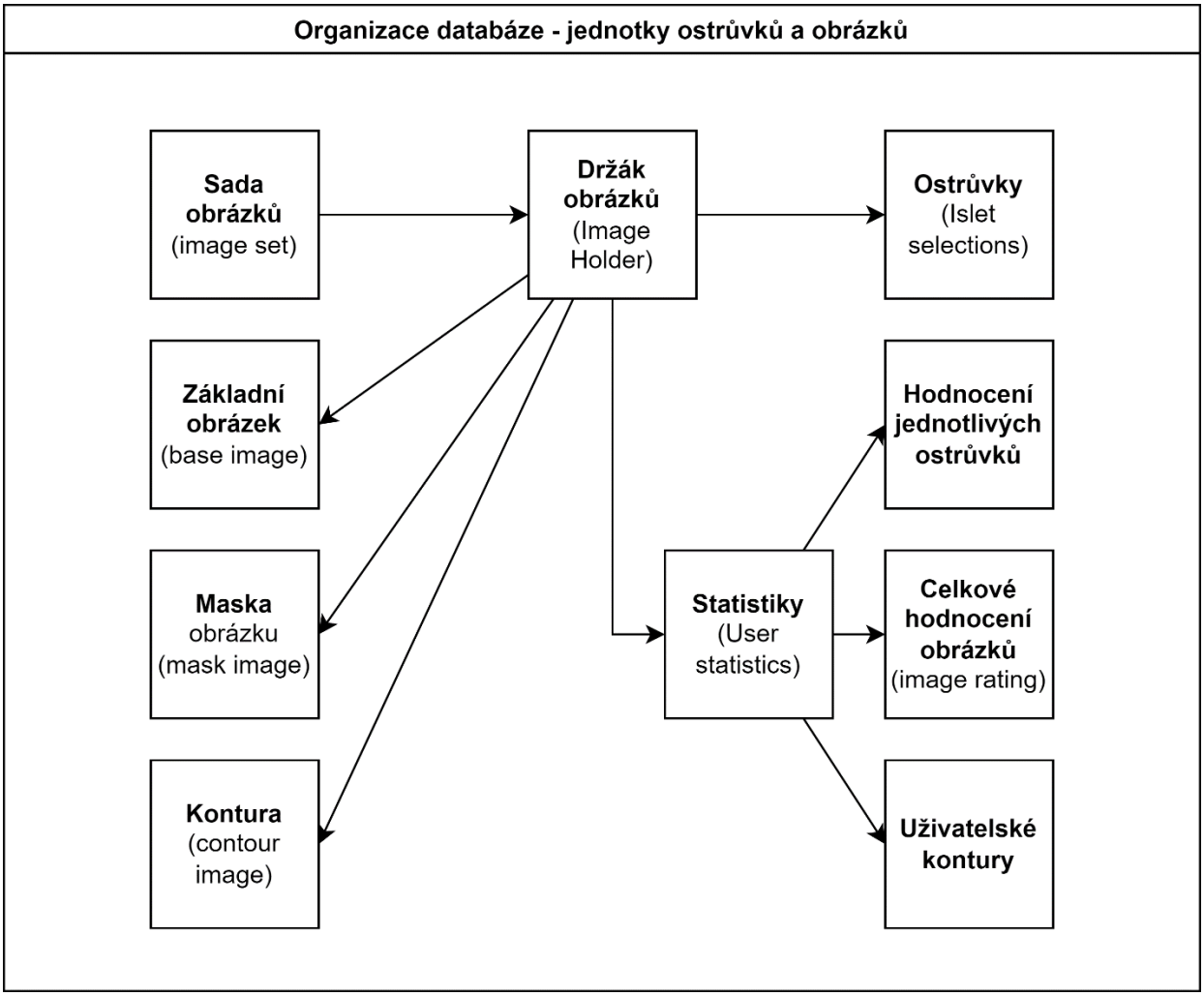
10. Použitá literatura

1. WEISWASSER, Jonathan M., Subodh ARORA, Charles SHUMAN, Dwight KELLICUT a Anton N. SIDAWY. Diabetic neuropathy. *Seminars in Vascular Surgery* [online]. **16**(1), 27-35 [cit. 2022-02-05]. ISSN 08957967. Dostupné z: doi:10.1053/svas.2003.50004
2. GILLESPIE, K. M. Type 1 diabetes: pathogenesis and prevention. *Canadian Medical Association Journal* [online]. 2006, **175**(2), 165-170 [cit. 2022-02-05]. ISSN 0820-3946. Dostupné z: doi:10.1503/cmaj.060244
3. SKYLER, Jay S., George L. BAKRIS, Ezio BONIFACIO, et al. Differentiation of Diabetes by Pathophysiology, Natural History, and Prognosis. Diabetes [online]. 2017, 66(2), 241-255 [cit. 2022-02-15]. ISSN 0012-1797. Dostupné z: doi:10.2337/db16-0806
4. BINGLEY, Polly J., Jeffrey L MAHON a Edwin A.M. GALE. Insulin Resistance and Progression to Type 1 Diabetes in the European Nicotinamide Diabetes Intervention Trial (ENDIT). Diabetes Care [online]. 2008, 31(1), 146-150 [cit. 2022-02-05]. ISSN 0149-5992. Dostupné z: doi:10.2337/dc07-0103
5. GAMBLE, Anissa, Andrew R. PEPPER, Antonio BRUNI a A. M. James SHAPIRO. The journey of islet cell transplantation and future development. Islets [online]. 2018, 10(2), 80-94 [cit. 2022-02-05]. ISSN 1938-2014. Dostupné z: doi:10.1080/19382014.2018.1428511
6. KISSLER, H.J., J.C. NILAND, B. OLACK, et al. Validation of methodologies for quantifying isolated human islets: an islet cell resources study. Clinical Transplantation [online]. 2010, 24(2), 236-242 [cit. 2022-02-05]. ISSN 09020063. Dostupné z: doi:10.1111/j.1399-0012.2009.01052.x
7. HABART, David, Jan ŠVIHLÍK, Jan SCHIER, et al. Automated Analysis of Microscopic Images of Isolated Pancreatic Islets. Cell Transplantation [online]. 2016, 25(12), 2145-2156 [cit. 2022-02-05]. ISSN 0963-6897. Dostupné z: doi:10.3727/096368916X692005
8. Iterait, a.s., 2022, https://isletnet.com [cit. 2022-02-15]
9. HABART, David, Adam KOZA a Frantisek SAUDEK. P.125: IsletSwipe. Transplantation [online]. 2021, 105(12S1), S49-S50 [cit. 2022-02-15]. ISSN 0041-1337. Dostupné z: doi:10.1097/01.tp.0000804576.76690.bf
10. Flutter. Flutter [online]. [cit. 2022-03-13]. Dostupné z: https://flutter.dev/
11. ZANINOTTO, Francois a Fabien POTENCIER. The Definitive Guide to Symfony. Apress, 2007. ISBN 9781430203797.
12. API Platfomr: REST and GraphQL Framework on top of Symfony and React. API Platform [online]. [cit. 2022-03-13]. Dostupné z: https://api-platform.com/
13. Twig: The flexible, fast, and secure template engine for PHP. Twig [online]. [cit. 2022-03-13]. Dostupné z: https://twig.symfony.com/
14. LINDHOLM, Tim. Java Virtual Machine. The Java Virtual Machine Specification. 3rd ed. Upper Saddle River (New Jersey): Addison-Wesley, 2013, s. 2-2. ISBN 978-0133260441.

15. ImageIO. *Java Platform SE 7* [online]. [cit. 2022-03-10]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html
16. Apache Maven Project. Maven [online]. [cit. 2022-03-13]. Dostupné z: https://maven.apache.org/
17. Isolates. *Concurrency in Dart* [online]. [cit. 2022-03-10]. Dostupné z: https://dart.dev/guides/language/concurrency
18. *Photo View* [online]. [cit. 2022-03-17]. Dostupné z: https://github.com/bluefireteam/photo_view
19. *Align Positioned* [online]. [cit. 2022-03-17]. Dostupné z: https://pub.dev/packages/align_positioned
20. Primitive Data Types. *Java Tutorials* [online]. [cit. 2022-03-10]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
21. JAYANTI, Siddhartha a Julian SHUN. Fast Arrays: Atomic Arrays with Constant Time Initialization. 35th International Symposium on Distributed Computing (DISC 2021). 2021, 35(209), 25:1-25:1. Dostupné z: doi:10.4230/LIPIcs.DISC.2021.25
22. Symfony documentation – voters [online]. [cit. 2022-03-17] https://symfony.com/doc/current/security/voters.html
23. *Inky dokumentation* [online]. [cit. 2022-03-18]. Dostupné z: https://get.foundation/emails/docs/inky.html

11. Přílohy

Příloha A: Organizace datových jednotek obrázků v databázi.



Příloha B: Organizace datových jednotek uživatelů v databázi