Příloha č. 1b

Obecná technická specifikace

Obsah

1 Úvod a obecná architektura

1.1 Popis problému a motivace

Univerzita Karlova využívá studijní informační systém (dále jen „SIS“) vyvinutý firmou ERUDIO s.r.o. Jádro tohoto systému pochází z 90. let 20. století a je již technologicky zastaralé. Systém má velké množství modulů, které však od sebe nejsou odděleny, což (mimo jiné) významně ztěžuje možnost škálovat výkon systému a také realizovat rozvoj systému více dodavateli. Výkonové problémy se pak projevují zejména při hromadných elektronických zápisech do předmětů a přihlašování studentů na zkoušky. Část modulů má podobu tzv. těžkých klientů, většina z nich má podobu webové aplikace. Jednou z velkých nevýhod stávajících webových modulů SISu je však jejich velmi problematické až nemožné využívání z mobilních zařízení.

V roce 2021 uzavřela UK dodatky smluv s firmou ERUDIO s.r.o., které umožňují SIS dále rozvíjet vlastními silami nebo s využitím třetích stran. To otevřelo univerzitě cestu k tomu, aby se vyvázala ze stávající tzv. vendor lock-in pozice a otevřela vývoj SISu směrem k většímu počtu dodavatelů, mezi nimiž by plnila úlohu integrátora.

Záměrem je převést SIS do podoby moderního informačního systému, který má modulární a tzv. servisně orientovanou architekturu, kde jednotlivé moduly spolu komunikují formou volání webových služeb, a ne sdílením dat ve společné databázi. Tento architektonický model umožňuje lépe škálovat výkon systému a usnadňuje jeho vývoj více různými dodavateli, kteří realizují jednotlivé moduly nebo jejich části, které spolu komunikují prostřednictvím těchto webových služeb.

Současně je cílem formálně popsat procesy (provést tzv. byznys analýzu), pro které SIS poskytuje podporu, a funkcionality obsažené v systému tak, aby tento (průběžně aktualizovaný) popis mohl sloužit jako podklad pro další rozvoj systému – pro komunikaci mezi univerzitou a jednotlivými dodavateli a také mezi dodavateli navzájem.

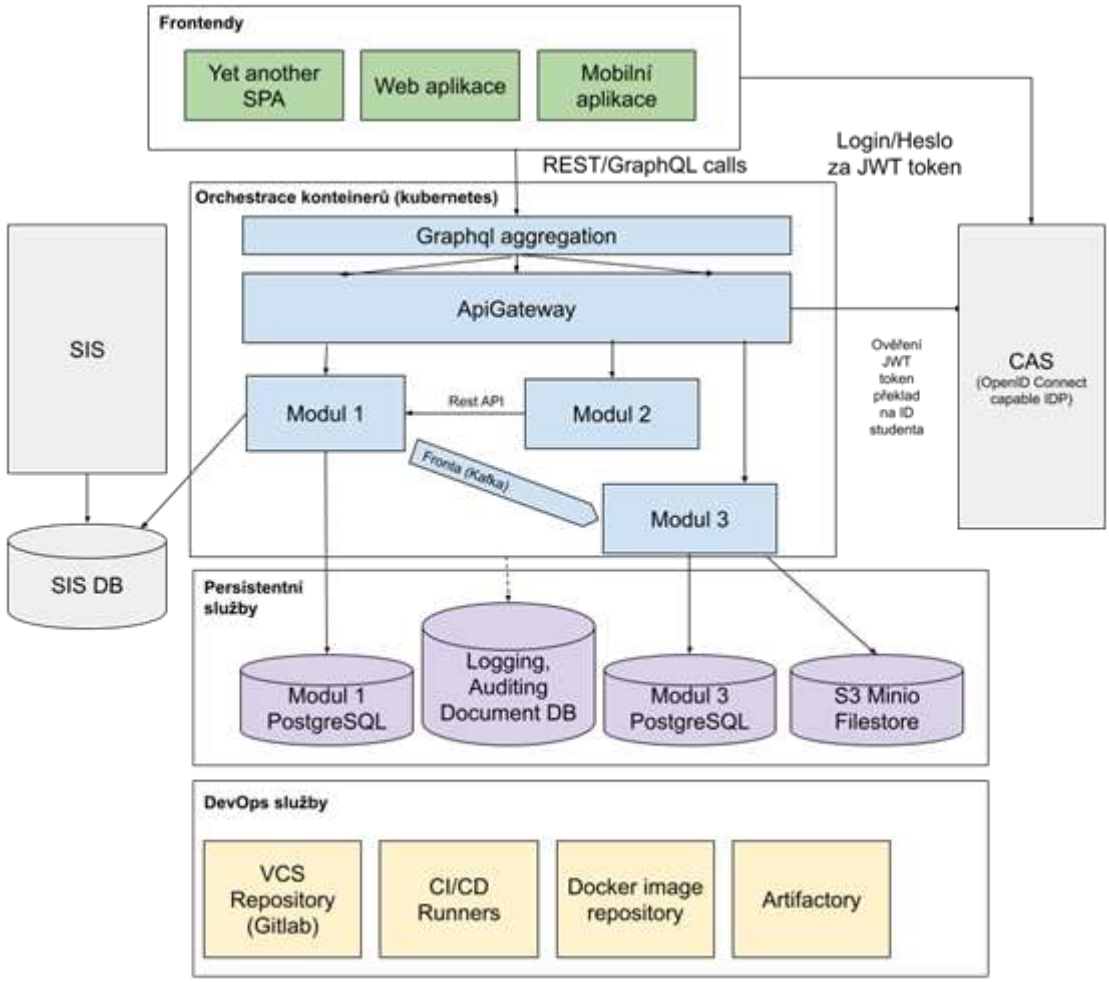
1.2 Obecné infrastrukturní požadavky

1.2.1 Koncept nové architektury

Nová architektura se vyznačuje svojí modulárností. Infrastruktura poskytuje sdílené služby aplikačním modulům, jako je autentizace, persistence, logování, monitorování, audit, atd.

Cílem této architektury je umožnit zejména:

- Paralelní vývoj několika dodavateli
- Postupnou reimplementaci Studentského Informačního Systému (SIS)
- Škálování výkonu při zátěži
- Continuous deployment
- Rozsáhlé možnosti QA na různých úrovních systému
- V případě zastarání jednoho modulu není třeba přepisovat celý systém



1.2.2 Prostředí, kontejnerizace

Veškeré aplikace budou nasazovány pomocí kontejnerů. Zvolenou technologií pro orchestraci je Kubernetes. Správa Kubernetes probíhá pomocí platformy Rancher. Kubernetes je napojený na Centrální Autentizační Službu (CAS) UK.

Jsou připravená tři prostředí:

1. **DEV (Development)**
    - Vývojářské prostředí

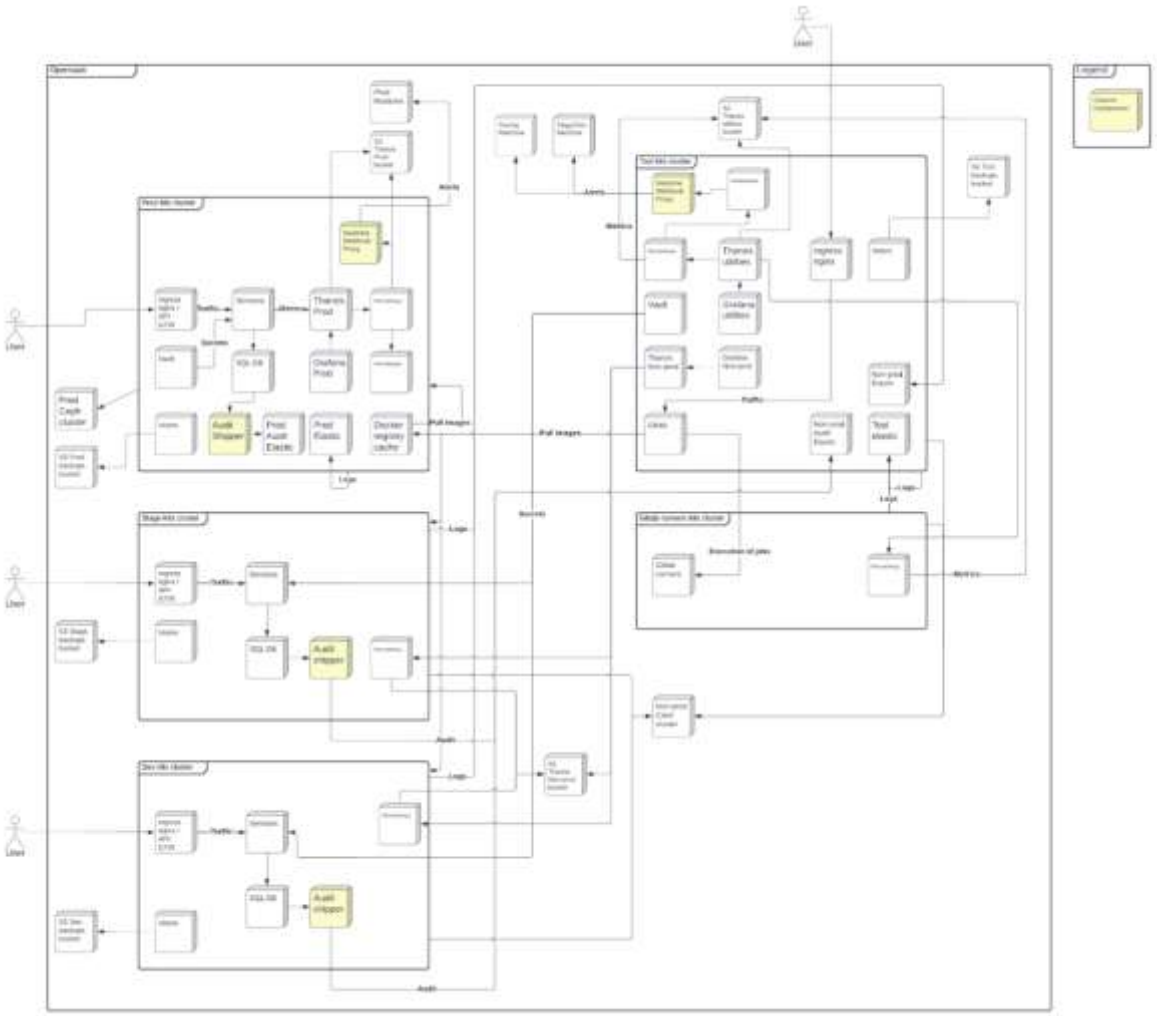
- Toto prostředí bude dostupné dodavatelům, vývojáři zde mohou nasazovat své moduly bez asistence pomocí předpřipravených CI/CD pipelines
- Možnost testovat integraci s ostatními moduly při vývoji

2. **STAGE (Staging)**
    - Testovací prostředí pro UK
    - Možnost spouštět zátěžové testy
    - Nasazování modulů je spravováno UK
3. **PROD (Production)**
    - Prostředí dostupné cílovým uživatelům

Každé z výše uvedených prostředí má vlastní Kubernetes cluster. Pro zjednodušení správy prostředí STAGE a DEV používají tato prostředí také sdílený TOOLS cluster, který obsahuje zejména sdílený Ceph cluster a sdílené nástroje pro monitoring a logování. Mezi další neprodukční systémové clustery patří také GitLab Runners cluster a Rancher management cluster.

Jedním z cílů vytvořeného prostředí je nasazení modulů v režimu vysoké dostupnosti (high availability) společně s automatickým škálováním dle aktuální zátěže.

Následující diagram znázorňuje přiřazení jednotlivých systémových komponent do zmiňovaných clusterů. Vlastní komponenty (vytvářeny interně na UK) jsou zvýrazněny žlutě. Moduly vyvíjené dodavateli jsou shrnuty pod komponentu “Services”.



Legenda

- Kubernetes – nástroj pro orchestraci kontejnerů nad clusterem serverů
- NGINX Ingress – obecná reverzní proxy
- Kong API Gateway – specializovaná rozšiřitelná reverzní proxy s pokročilými funkcemi
- PostgreSQL – relační databáze
- Kafka – asynchronní fronta pro komunikaci mezi moduly
- Ceph – platforma spravující datová úložiště
- S3 – běžné rozhraní pro úložiště souborů/objektů
- GitLab – nástroj pro správu zdrojových kódů a automatizaci
- GitLab Runner – komponenta GitLabu, která zprostředkovává automatizaci
- Docker Registry – úložiště obrazů kontejnerů
- ArgoCD – nástroj pro synchronizaci nasazování aplikací a infrastruktury
- Velero – zálohovací nástroj pro Kubernetes
- Prometheus – nástroj pro sběr a krátkodobé úložiště metrik
- Alertmanager – nástroj pro správu automatizovaných upozornění
- Thanos – nástroj pro dlouhodobé ukládání metrik
- Grafana – nástroj pro zobrazování metrik a grafů
- Redmine – wiki a správa incidentů
- Redmine Webhook Proxy – vlastní komponenta pro napojení Alertmanageru a Redmine
- Elasticsearch – úložiště logů a audit záznamů
- Logstash – nástroj pro transformaci logů a audit záznamů
- Filebeat – nástroj pro sběr logů z jednotlivých komponent
- Audit Shipper – vlastní komponenta pro přesun audit záznamů z modulových DB schémat do centrálního úložiště auditních záznamů, napr. Elasticsearch
- Kibana – nástroj pro zobrazení a vyhledávání logů a audit záznamů
- Gatekeeper – nástroj pro vynucování pravidel pro všechny součásti Kubernetes

1.2.3 Git repositář kódu, CI/CD

Pro správu zdrojového kódu jednotlivých modulů je používána platforma GitLab, provozována v rámci výše popsaného kontejnerového prostředí. GitLab je sdílený pro všechna tři běhová prostředí. Slouží také zároveň jako Container Repository (nicméně produkční prostředí má vlastní Docker Registry Cache).

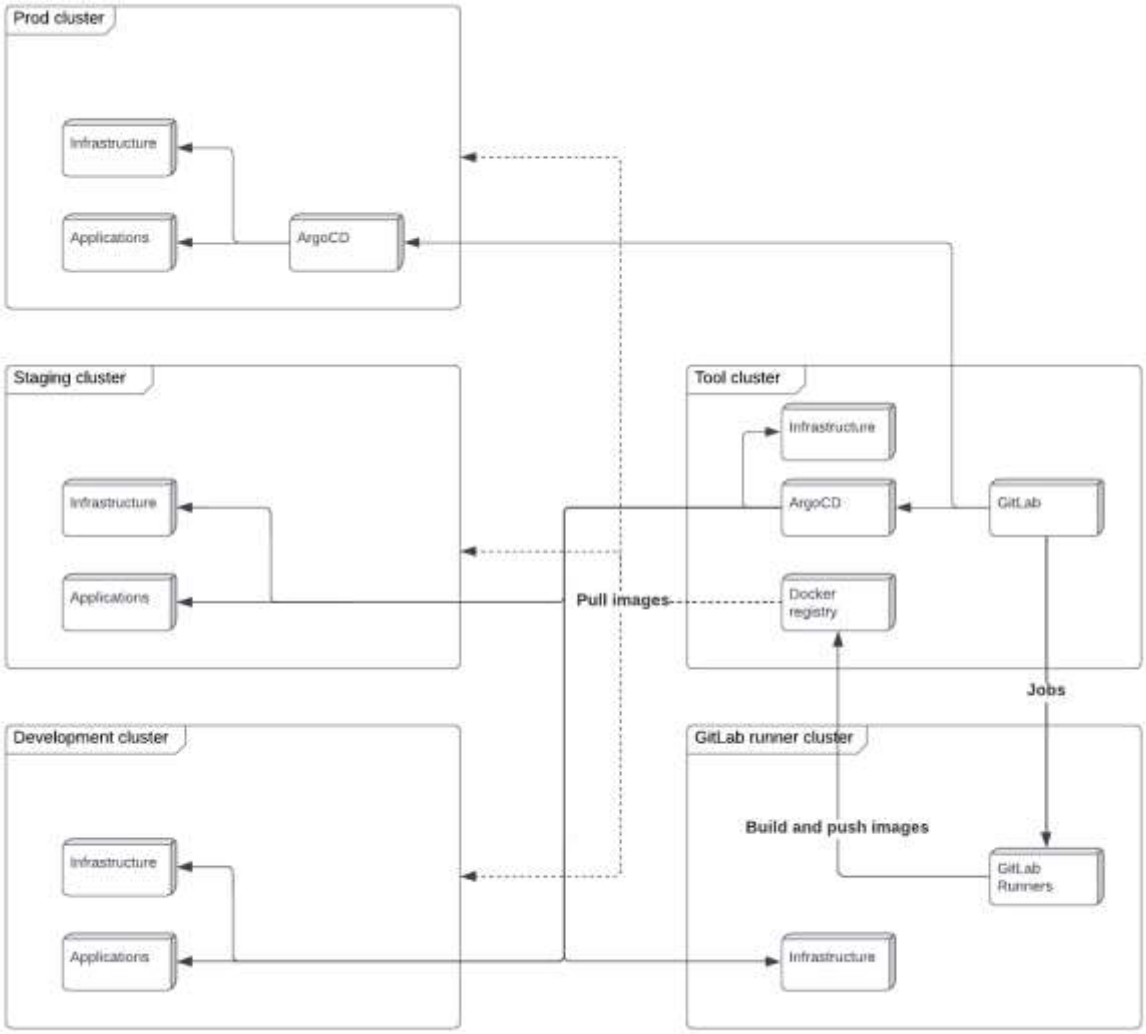
Pro každý modul bude v rámci prostředí GitLab připraven separátní projekt (skupina projektů), ke kterému obdrží dodavatel daného modulu přístup. Zadavatel také vytvoří základní CI/CD pipelines pomocí nástrojů GitLab, ArgoCD a Helm Charts pro automatizované nasazování modulu v prostředí DEV. Správa a následný vývoj těchto CI/CD pipelines bude následně probíhat ve spolupráci s dodavatelem. Finální CI/CD pipelines budou zadavatelem aplikovány také do STAGE a PROD prostředí.

Každý modul musí být reprezentovaný kontejnerem s následujícími požadavky:

- Kód musí být kompatibilní s rootless Docker image
- Každý modul používá standardní šablony pro GitLab CI a standardní Helm chart
- Base images kontejnerů: poslední stable Alpine, Debian nebo Ubuntu

Kromě samotného vyvíjeného modulu budou dodavateli zpřístupněné v rámci prostředí GitLab i vzorové moduly.

Následující CI/CD diagram znázorňuje 2 ArgoCD instance, které obsluhují všechny clustery a nasazení infrastruktury a modulů. Produkční instance je kompletně oddělená od neprodukční. ArgoCD je napojené na GitLab a synchronizuje stav několika repositářů s nastavením aplikací do jednotlivých clusterů.



1.2.4 Správa hesel a přístupových údajů (Secrets)

Veškeré přístupové údaje (URL služeb, hesla, tokeny atd.) budou každému modulu poskytnuty pomoci GitLab proměnných v odpovídajícím GitLab projektu a poté pomocí CI/CD pipelines dostupná jako proměnné běhového prostředí (OS environment variables). Každý modul tedy musí tyto informace načítat při startu z proměnných prostředí. **Je explicitně zakázáno ukládat podobné informace, zejména hesla a podobné přístupové údaje v jakýchkoliv konfiguračních souborech modulu.**

1.2.5 Principy komunikace mezi službami/moduly

Pro komunikaci mezi jednotlivými moduly budou použité následující technologie

- REST(-like) HTTP API
- GraphQL
- RPC
- Fronta
- Webhooks

1.2.5.1 Synchronní komunikace

Jako primární technologie určená pro synchronní komunikaci mezi moduly bude použito REST-like HTTP API. V některých případech v budoucnu může být také předepsána implementace GraphQL nebo RPC rozhraní. Konkrétní specifikace typu API (REST vs. GraphQL vs. RPC), včetně specifikace požadovaných operací, datových formátů, vstupních a výstupních dat bude detailně určena v zadávací dokumentaci každého modulu.

Kromě předpisu API rozhraní, které má modul vystavit pro ostatní moduly bude součástí zadávací dokumentace každého modulu také seznam a popis API rozhraní jiných modulů, které bude požadovaný modul potřebovat pro implementaci svých funkcionalit.

Pro zamezení snižování výsledné spolehlivosti modulů by moduly mezi sebou neměly v rámci zpracovávání jednoho požadavku od uživatele provést víc než dvě synchronní volání. Moduly by navíc neměly mezi sebou mít cyklickou závislost synchronních volání.

1.2.5.2 Asynchronní komunikace

Jako primární technologie určenou pro asynchronní komunikaci mezi moduly bude použita fronta, implementována pomocí platformy Apache Kafka. V rámci jednoho prostředí bude poskytnuta jedna sdílená instance Kafky. Přístupové informace modul obdrží v rámci sady odevzdávaných přístupových údajů pomocí CI/CD, jak bylo popsáno výše.

1.2.6 Webhooks

Další z dostupných alternativ pro komunikaci mezi moduly je také využití Webhooks u případů, kde charakter komunikace implementovaní touto technologii bude výhodnější než použití standardního REST API rozhraní. Požadavek na použití Webhooks bude pro daný modul specifikován v rámci zadávací dokumentace modulu.

1.2.7 Síťový provoz a napojení, reverzní proxy, API brána

Veškerý síťový provoz bude směrován na vstupní komunikační body (endpointy) modulu pomocí reverzní proxy a API brány. Tyto prvky budou poskytovat terminaci příchozího TLS spojení, load-balancing, základní autorizaci a směrování datového provozu na odpovídající moduly. Pro propagaci zdrojových IP adres klientů příchozích HTTP spojení budou přidávané hlavičky **X-Forwarded-For**.

Použité technologie pro reverzní proxy i API bránu jsou zároveň kompatibilní s logovacím, monitorovacím a tracing řešením, takže transparentně probíhá např. zaznamenávání požadavků do

logu včetně cesty volání, status kódu a doby trvání, korelace logů, latence a statistických indikátorů s ostatními moduly.

Korelace jednotlivých auditních a logovacích zpráv je zajištěna pomocí přidávání identifikátoru požadavku Correlation-ID do všech logovacích a auditních zpráv. Tento identifikátor je automaticky vytvořen pro každý nový požadavek vstupující do systému přes API bránu, a to pomocí přidání HTTP hlavičky **X-Correlation-Id** s hodnotou vygenerovaného identifikátoru.

1.2.8 Autentizace a autorizace

Z pohledu autentizace a autorizace počítá architektura systému se dvěma hlavními případy užití:

- Uživatel přistupující k systému pomocí prohlížeče skrze webový Portál
- Jiný systém/uživatel přistupující přímo k veřejnému API systému

V prvním případě proběhne autentizace a autorizace uživatele pomocí odpovídajícího flow OpenID Connect/OAuth, s využitím stávající Centrální Autentizační Služby (CAS) UK. Z pohledů aplikačních modulů bude výsledkem přihlášení access token, který obdrží Portál SIS (jakožto vstupní uživatelské webová brána do SIS).

Na základě přijatého access tokenu (a z něj odvozených informací o přihlášeném uživateli) může Portál rozhodnout, ke kterým portálovým aplikacím bude uživateli poskytnutý přístup. Pro vstup do specifické portálové aplikace musí Portál následně požádat o poskytnutí odpovídajícího HTML UI specifický modul. V rámci tohoto volání Portál poskytne modulu také access token kontextu transakce. Jestliže tento modul potřebuje navíc kontaktovat další modul, musí mu v rámci odpovídajícího API volaní také přeposlat obdržený access token.

Podobně ve druhém případě, při použití odpovídajícího flow OpenID Connect/OAuth, modul obdrží v rámci přijatého požadavku také odpovídající access token, který v případě volání dalších modulů jim bude také přeposílat.

Jednotlivé moduly můžou na základě obdrženého access tokenu dále přesněji vyhodnocovat interní autorizační pravidla v závislosti na specifikách své aplikační logiky. Detaily autorizační logiky specifické pro daný modul budou upřesněné v zadávací dokumentaci.

1.2.9 Perzistentní datová úložiště

Pro perzistentní ukládání dat má každý modul k dispozici následující tři druhy úložišť:

- RDBMS – Relační databáze podporující transakce
- File/blob úložiště – S3
- Fulltext indexovaná dokumentová databáze (Document Store)

Pro všechny výše uvedené nástroje pro perzistenci dat zabezpečí zadavatel zároveň odpovídající řešení pro zálohu dat.

Všechny tyto nástroje jsou zároveň spravovány centrálně a přístupové údaje k nim jsou jednotlivým modulům poskytovány dle výše popsaného mechanismu sdílení přístupových údajů.

Veškeré změny nutné pro instalaci nebo upgrade se provádí automatizovanými migračními skripty poskytnutými dodavatelem. Úkolem těchto skriptů je zabezpečit reprodukovatelnost instalace na dalších prostředích.

Z důvodu použití kontejnerizace a automatického horizontálního škálování nesmí moduly ukládat žádná persistentní data na lokální disk. Jakákoliv potřeba ukládat data na disk dočasně musí být předem konzultována a odsouhlasena zadavatelem. Detaily takové případné implementaci pak musí být popsány v dodavatelem doložené technické dokumentaci.

1.2.9.1 Relační databáze

Standardní relační databáze bude v drtivé většině případů použitá jako primární nástroj pro perzistentní uložení dat. Jedním z hlavních důvodů je nutnost integrace nově vyvíjených modulů s původním Studijním informačním systémem na datové úrovni, protože integrace na úrovni databáze je jediný původním systémem podporovaný způsob napojení. Původní systém využívá databázi Oracle.

Pro každý modul tedy bude vytvořeno nové databázové schéma, v rámci kterého budou vytvořené pohledy (views) pro čtení dat sdílených s původním systémem, a také uložené databázové procedury (stored procedures) pro vykonávání zápisových transakcí. Detailní popis dostupných databázových prostředků je dodán jako součást zadávací dokumentace daného modulu.

Kromě práce se sdílenými (starými) daty budou některé moduly potřebovat také ukládat data odpovídající novým, doposud neevidovaným byznys objektům. Pro tyto datové entity bude zadavatelem předepsána datová struktura databázových tabulek, které modul v rámci jemu dostupného databázového schématu bude využívat pro perzistenci datových entit odpovídajícího typu.

Vedle specifikací předepsaných datových objektů budou moduly typicky potřebovat i další pomocné databázové tabulky pro ukládání implementačně specifických perzistentních dat. Strukturu potřebných tabulek sdělí dodavatel zadavateli v průběhu implementace, a to v podobě automaticky spustitelných SQL skriptů. Vytvoření potřebných databázových struktur poté zabezpečí zadavatel.

Každý modul pracující s relační databází musí tento zdroj využívat optimalizovaným způsobem, dle dobrých zvyklostí. Nutností je používání kvalitního Database Connection Pool nástroje (pro Javu např. c3p0) umožňujícího vysokou míru nastavitelnosti a sdílení databázových spojení. Nastavení Database Connection Poolu pak musí být dostupné v rámci konfigurace modulu, který ho využívá. Samozřejmostí je pak používání prepared statements a dalších standardních doporučení při práci s relační DB.

1.2.9.2 File/blob úložiště – S3

Pro potřeby ukládání souborů nebo blobů je pro každý modul k dispozici objektové úložiště dostupné pomocí S3 API, poskytované systémem Ceph. V případě potřeby je možné vytvořit pro daný modul i vícero S3 bucketů.

Vytváření S3 bucketů je možné na základě konzultace a schválení zadavatelem. Dodavatel musí kromě účelu a typu užití poskytnout také očekávaný objem ukládaných dat, včetně odhadované četnosti čtecích a zápisových operací.

1.2.9.3 Fulltext indexovaná dokumentová databáze

V odůvodněných případech je možné pro modul vytvořit separátní index v rámci dokumentové databáze Elasticsearch. Tento typ perzistence je vhodný zejména v případech, kdy je potřebná fulltextová indexace dokumentově orientovaných dat.

Vytváření indexů je možné na základě konzultace a schválení zadavatelem. Dodavatel musí kromě účelu a typu užití poskytnout také očekávaný objem ukládaných dat, včetně odhadované četnosti čtecích a zápisových operací.

1.2.10 Audit

Nedílnou součástí procesování většiny příchozích požadavků daným modulem je synchronní, transakčně korektní a dostatečně průkazné vytváření odpovídajících auditovacích záznamů popisujících důležité kontextuální podrobnosti procesovaných požadavků. Vytvářené auditní záznamy jsou pak neměnné a slouží primárně jako zdroj dat pro naplnění legislativních požadavků, nebo také pro účely reportingu.

Součástí zadávací dokumentace pro každý modul je také seznam auditních událostí, které nastávají v rámci daného modulu po dobu jeho aplikačního běhu. Příkladem takových událostí může být například již zmiňovaná obsluha příchozích požadavků, generování odchozích požadavků, ale také například start/stop modulu, runtime změna konfigurace, přihlášení uživatele, import dat, zpuštění důležitých automatizovaných procesů na pozadí, detekce alertu nebo chyby systému, zaslání e-mailu, atd.

Každý záznam o auditní události musí obsahovat:

- Časovou značku (timestamp) události
- Typ události
- ID služby (modulu)
- ID instance
- ID uživatele
- ID požadavku
- Correlation ID (viz sekce Logování)
- ID nebo jméno vykonávané akce
- Další položky specifické pro daný typ události specifikované v zadávací dokumentaci modulu (typicky vyžadované legislativou a platnými směrnicemi)

Přesný seznam všech pro modul předepsaných auditních událostí, které modul musí implementovat, včetně požadované struktury dat, je součástí zadávací dokumentace daného modulu.

Auditní záznam pro předepsanou událost musí být uložen perzistentním a transakčně korektním způsobem jako nedílná součást odpovídajícího výpočetního procesu. Například, nemělo by být možné vrátit data v rámci obsluhy požadavku ale přitom nevytvořit odpovídající auditní záznam, nebo naopak, neobsloužit požadavek ale přitom vytvořit auditní záznam o jeho obsluze. Tato vlastnost se typicky dosahuje pomocí vytváření auditních záznamů v rámci stejné databázové transakce, uvnitř které probíhá i obsluha procesovaného požadavku a zápis aplikačních dat. Jinými slovy, pro vytváření auditních záznamů bude použit tzv. Outbox Pattern, s outbox tabulkou v databázi modulu.

Kromě samotného sémantického významu jednotlivých auditních událostí je tyto možné dělit do dvou hlavních kategorií na a) párové, a b) nepárové. Příkladem nepárových událostí je např. start systému nebo pokus o přihlášení uživatele do systému. Párové události typicky popisují začátek a konec odpovídajícího procesu nebo transakce, přičemž pro daný proces platí, že buď má delší trvání,

nebo v jeho průběhu je zvýšená pravděpodobnost možného pádu nebo zaseknutí systému (např. z důvodu nedostatku systémových procesů, čekaní na externí zdroje atd.). Párové události jsou tedy typicky používané pro zaznamenání procesování uživatelských požadavků, přičemž jeden auditní záznam je vytvořen při přijetí požadavku (včetně detailů popisujících „kdo se ptá na co“), a druhý auditní záznam je vytvořen při zaslání odpovědi na daný požadavek (detaily zde zachycují „co bylo poskytnuto“, případně úspešnost (status) dané transakce, dobu trvání, a podobně.

Auditní záznamy vytvářené všemi aplikačními moduly v rámci jejich transakčních databázových schémat jsou kontinuálně přesouvány a agregovány do dlouhodobého centrálního perzistentního úložiště auditních záznamů pomocí interně vyvíjené komponenty Audit Shipper.

1.2.11 Logování

Všechny moduly zapisují logovací záznamy do standardních výstupů, tj. do **stdout** a **stderr**, přičemž do stderr by měly být zapisovány pouze chybové zprávy (zprávy úrovně ERROR).

Logy jsou agregovány, zpracovány a uchovávány pomocí platformy Elastic Stack (Filebeat, Logstash, Elasticsearch, Kibana) za účelem možnosti vzájemné korelace událostí ze všech modulů a pro zajištění dostupnosti logů i po kritickém selhání a odstranění kontejneru. Webové rozhraní aplikace Kibana v DEV prostředí bude dodavateli zpřístupněno pro umožnění vyhledávání událostí týkajících se konkrétního modulu, kontejneru, časového rozmezí, ID požadavku či úrovně atd. Do budoucna se plánuje vytvoření vlastní aplikace pro prohlížení logovacích záznamů, který by poskytoval ještě větší pohodlí a přehled pro náhled do logů objemných víceúrovňových transakcí.

Z výše uvedených důvodů musí moduly vytvářet logovací záznamy ve strojově čitelné strukturované formě. Jako výchozí formát bylo zvoleno Elastic Common Schema (ECS) ve formátu JSON (https://www.elastic.co/guide/en/ecs-logging/overview/current/intro.html). Tento formát byl dále rozšířen o několik přídavných atributů. Následující tabulka uvádí přehled povinných atributů (ECS fields), které musí obsahovat každý logovací záznam. Samozřejmě, je možné logovat i další metadata dle ECS standardu, nicméně níže uvedené položky jsou povinné.

Název atributu	**Popis**	**Příklad**
@timestamp	Časová značka logovacího záznamu	2022-10-11T20:48:18.054Z
custom.nano	Počet nanosekund v rámci aktuální sekundy z časové značky logovací události. Slouží pro rozlišení pořadí událostí/zpráv vytvořených v rámci stejné milisekundy.	54230932
log.level	Log level logovací zprávy. Možné hodnoty jsou ERROR, WARN, INFO, DEBUG, TRACE. Tabulka uvedená níže obsahuje popis logovacích zpráv, které má modul vytvářet pro jednotlivé logovací levely.	INFO
service.name	Jméno nebo identifikátor modulu, stanovený na základě domluvy se zadavatelem	portal
message	Obsah textu logovací zprávy	Received request: /portal/

Název atributu	Popis	Příklad
error.stack_trace	Stacktrace logovací zprávy (je-li dostupný), určený zejména pro chybové zprávy	java.lang. NullPointerException at java.base/ java.util.Objects. requireNonNull (Objects.java:208) at …
custom.correlation.id	Unikátní ID zpracovávaného požadavku, které bylo přirazeno pomocí platformy API Gateway při vstupu do systému. Všechny moduly v rámci architektury si při následných voláních tento identifikátor odevzdávají v HTTP hlavičkách.	2b7e26cc-fcdb-429f-b685-f5150c2bb95a#1765
custom.uuid	ID logovacího záznamu ve tvaru UUID, vygenerováno modulem při vzniku záznamu	cb9321ba-b30f-47ad-90b9-781fb310576a
custom.has_children	Boolean příznak indikující, zda-li daná logovací událost je počátkem logicky podřazeného výpočetního podstromu obsahujícího podřazené logovací záznamy (které můžou být vytvořené jiným modulem nebo volanou komponentou modulu). Při stromovém zobrazení logovacích záznamů si je možné logovací záznamy s hodnotou tohoto příznaku nastavenou na „true“ možné představit jako rozkliknutelný podstrom podřazených logovacích událostí.	false
custom.parent	ID přímo nadřazeného rodičovské logovací zprávy. Jestliže je aktuální požadavek vykonáván jako podřazená operace nejaké nadřazené části aplikační logiky, pošle nadřazený modul nebo komponenta modulu identifikátor rodičovské logovací zprávy (kořen logovacího podstromu) pomocí HTTP hlavičky **X-Parent-Id**, nebo pomocí interních komunikačních cest mezi interními komponenty moduly. Od podřazené aplikační logiky se pak očekává vyplňování tohoto identifikátoru do všech vytvářených zpráv až do momentu kdy nevznikne nový logovací podstrom. Způsob vytváření a užití parent id je také demonstrován pomocí zdrojového kódu vzorových modulů, které budou dodavateli zpřístupněny v rámci přístupu do GitLab.	b0da9c30-b7f4-4210-8605-0b7d797bcf20

Následující tabulka popisuje, jaký typ logovacích zpráv je vyžadováno vytvářet v rámci jednotlivých logovacích levelů.

Log level	Typ požadovaných logovacích zpráv
ERROR	• Zalogování chybových stavů, z nichž se modul (proces zpracování požadavku) nezotavil, včetně stacktrace
WARN	• Zalogování chybových a “deprecated” stavů, z nichž se modul (proces zpracování požadavku) zotavil
INFO	• Zalogování zahájení a ukončení každé transakce/requestu včetně ID uživatele a status kódu (transakcí se rozumí každá naplánovaná aktivita - job, aktivita iniciovaná uživatelem i aktivita iniciovaná jinou komponentou systému) • Modul zajistí dostupnost informací o činnosti modulu pro potřeby zpětné diagnostiky neočekávaných a náhodných stavů
DEBUG	• Zalogování zahájení a ukončení komunikace s ostatními moduly a systémy třetích stran s úrovní DEBUG • Zalogování interních kroků aplikační logiky z pohledu procesního flow takovým způsobem, aby bylo možné ověřit korektnost procesního algoritmu • Zalogování všech ostatních zpráv, které pomohou případné diagnostice neočekávaných stavů
TRACE	• Zalogování podrobností o průběhu vykonávání aplikační logiky, obsahujících zejména objemný obsah, např. payload transakcí, TLS handshake, velmi jemné krokování složitějších algoritmů, apod.

Následující ukázka kódu v jazyce Java demonstruje jeden z možných způsobu vytváření logovacího zázamu ve formátu JSON na základe dodaných strukturovaných hodnot jednotlivých atributů (fieldů).

Ukázka Java kódu – příklad serializace logovacího záznamu

```
public String toJson() {
cr
   JSONObject jsonLog = new JSONObject();

   jsonLog.put("@timestamp",
DateTimeFormatter.ISO_INSTANT.withZone(ZoneOffset.UTC).format(instant));
   jsonLog.put("custom.nano", Integer.toString(instant.getNano()));
   jsonLog.put("custom.correlation.id", correlationId);
   if (uuid != null) jsonLog.put("custom.uuid", uuid);
   if (parent != null) jsonLog.put("custom.parent", parent);
   if (stackTrace != null) jsonLog.put("error.stack_trace", stackTrace);
   jsonLog.put("log.level", level.name());
   jsonLog.put("message", message);
   jsonLog.put("custom.has_children", hasChildren ? "true" : "false");
   jsonLog.put("service.name", serviceName);

   return jsonLog.toString();
}
```

V některých případech bude nutné citlivé údaje vkládané do logovacích záznamů šifrovat. Každé šifrované zprávě musí předcházet stejná nešifrovaná zpráva bez citlivých údajů. Seznam citlivých údajů bude pro každý modul upřesněn v zadávací dokumentaci.

Defaultní log level je nastavitelný pomocí konfigurace modulu (v proměnné prostředí). Nicméně komponenty systému předřazené před volání samotného modulu (např. Portál, nebo Centrální API Modul) poskytují schopnost nastavovat požadovaný dynamický log level pro vykonávanou transakci (požadavek) na základě kontextuálních informací (např. potřeba logování v levelu DEBUG pouze pro uživatele XY). Dynamicky určený log level požadavku tedy jednotlivé aplikační moduly mohou pro danou transakci obdržet v HTTP hlavičce s názvem **X-Log-Level** daného volání. V případě podřazeného navazujícího volání dalších modulů musí modul tuto hlavičku dalším modulům také přeposlat.

Aplikační logika modulu musí zabezpečit, aby se pro logovací zprávy, pro které je potřeba náročnější příprava výstupních dat (např. serializace větších objektů do paměti atd.), tato příprava výstupních dat nevykonávala v případě, kdy odpovídající logovací zpráva nebude dle aktuálního log levelu zaslána na výstup (ať již z konfigurace nebo dle dynamicky vyžadované úrovně logování). Například, v úrovni DEBUG může v kódu probíhat logování celého obsahu HTTP transakcí. Je nežádoucí, aby se v případech, kdy je aktuální logovací úroveň nižší (ERROR až INFO), prováděla serializace obsahu HTTP transakcí do paměti, jelikož by to zcela zbytečně zabíralo jak procesorové, tak i paměťové prostředky systému při frekventovaném běhu takových transakcí.

Při použití některých externích knihoven může být komplikované zapisovat na výstup všechny zprávy v požadovaném strukturovaném formátu (knihovny můžou nějaké zprávy již generovat). Dodavatel by ale měl v součinnosti a na základě dohody se zadavatelem rozhodnout o odchytávání těchto zpráv a jejich zapisování na výstup v požadovaném formátu, pakliže to nebude představovat nepřiměřenou vývojářskou náročnost.

HTML/JS frontend aplikace standardně logují do vývojářské konzole a do bufferu, v produkci je log do konzole vypnutý.

1.2.12 Tracing

Mezi hlavní cíle nasazení tracingu patří zobrazení trvání dotazů a souvisejících metadat, korelace latence napříč všemi moduly, dostupnost těchto informací i po kritickém selhání a odstranění kontejneru, nebo také vyhledávání informací týkajících se konkrétního modulu, kontejneru, časového rozmezí či požadavku.

Jako hlavní technologie pro implementaci tracingu v této architektuře systému jsou použité:

- OpenTelemetry
- Elastic APM

Je doporučeno pro implementaci modulů využít dostupnou instrumentaci pomocí OpenTelemetry SDK. Pro důležité logické celky aplikační logiky může být v zadávací dokumentaci modulu předepsána vlastní implementace nových měřených úseků (spanů). Dodavatel může taktéž navrhnout zadavateli vytvoření vlastních nových měřených úseků.

Sbírané tracing záznamy jsou odesílané knihovnou OpenTelemetry SDK do Elastic APM, kde jsou uchovávany, analyzovány a zobrazovány pro cílové uživatele těchto údajů. Přístup do Elastic APM (Kibana) bude v rámci DEV prostředí poskytnutý také dodavateli.

V případě nemožnosti použít OpenTelemetry SDK je možné vytvářet vlastní datové zprávy ve standardním otevřeném formátu OpenTelemetry a zasílat je pomocí standardních protokolů (OTLP/gRPC) do Elastic APM.

Trace musí obsahovat Correlation ID (viz sekce Síťový provoz), časovou značku, ID instance, unikátní ID v rámci celého systému, ID uživatele, status code, a informace o požadavcích vyvolaných tímto požadavkem. Případné další požadavky na obsah traců může být upřesněn v zadávací dokumentaci modulu.

Tracing daného modulu musí být vypínatelný a nastavitelný pomocí proměnných prostředí (konfigurace modulu).

1.2.13 Metriky a monitoring

Mezi důležité vlastnosti zvolené architektury patří také sběr metrik ze všech provozovaných modulů, kde metrikou se rozumí číselná řada popisující nějakou vlastnost systému/modulu v čase. Příkladem mohou být počty přijatých a odeslaných transakcí, jejich průměrná/minimální/maximální délka trvání, počet otevřených uživatelských sezení, rychlost vyřizování jednotlivých databázových dotazů, četnost spouštění jobů na pozadí atd. Tyto metriky jsou sbírané (nebo odesílané modulem) v pravidelných časových intervalech (např. jednou za minutu). Průměrně složitý modul může poskytovat stovky různých metrik.

Mezi hlavní důvody sběru metrik patří:

- **diagnostika systému/modulů**
- rutinní sledování provozu systému pomocí administrátorských dashboardů zobrazujících aktuální (živý) stav systému a přehled o jeho aktivitě
- analýza a sledování selhání a neočekávaných stavů
- zobrazení statistických informací o četnosti selhaní v jednotlivých modulech
- poskytnutí informací, jež koncovému uživateli pomohou určit, zda chybný stav, jenž nastal, je způsoben chybou na straně systému nebo je externího původu
- vzájemná korelace statistických údajů všech modulů, instancí a HW prostředků
- **automatické notifikace v případě překročení povolených mezí vybraných metrik (alertování)**
- různé pohledy pro vlastníky systému, provozovatelé, správce infrastruktury a vývojáře
- **reporting**

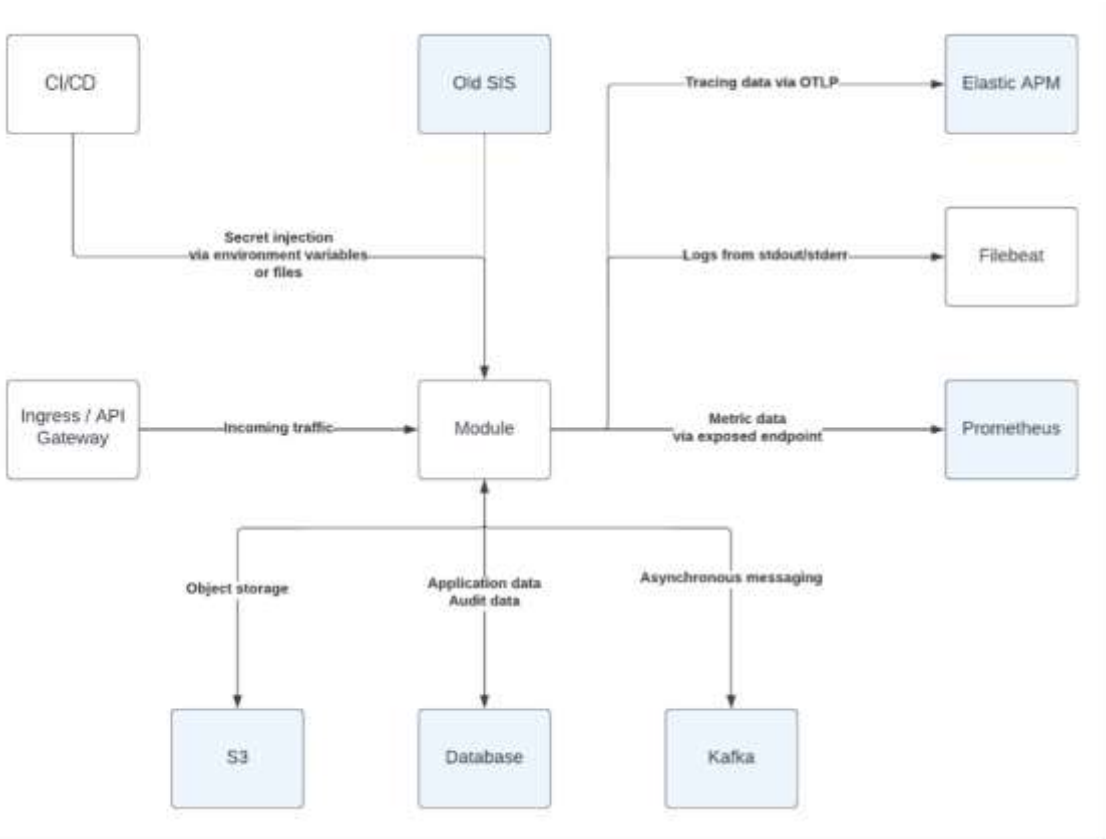
Tyto metriky jsou sbírané a ukládané ve specifické databázi určené pro ukládání numerických časových řad. Pro tento účel byla zvolena platforma Prometheus/Thanos. Agregované statistiky jsou automaticky analyzovány systémem Alertmanager, který v případě překročení očekávaných limitů automaticky vytvoří nový záznam (ticket) v systému Redmine a upozorní administrátory v aktuální on-call směně. Metriky je možné prohlížet pomocí aplikace Grafana, ke které v rámci DEV prostředí obdrží dodavatel přístup.

Pro automatický sběr musí každý modul vystavit endpoint s metrikami v Prometheus formátu. Alternativou je aktivní posílání metrik na Pushgateway. Forma odevzdávání metrik bude domluvena mezi zadavatelem a dodavatelem na začátku implementačního projektu daného modulu.

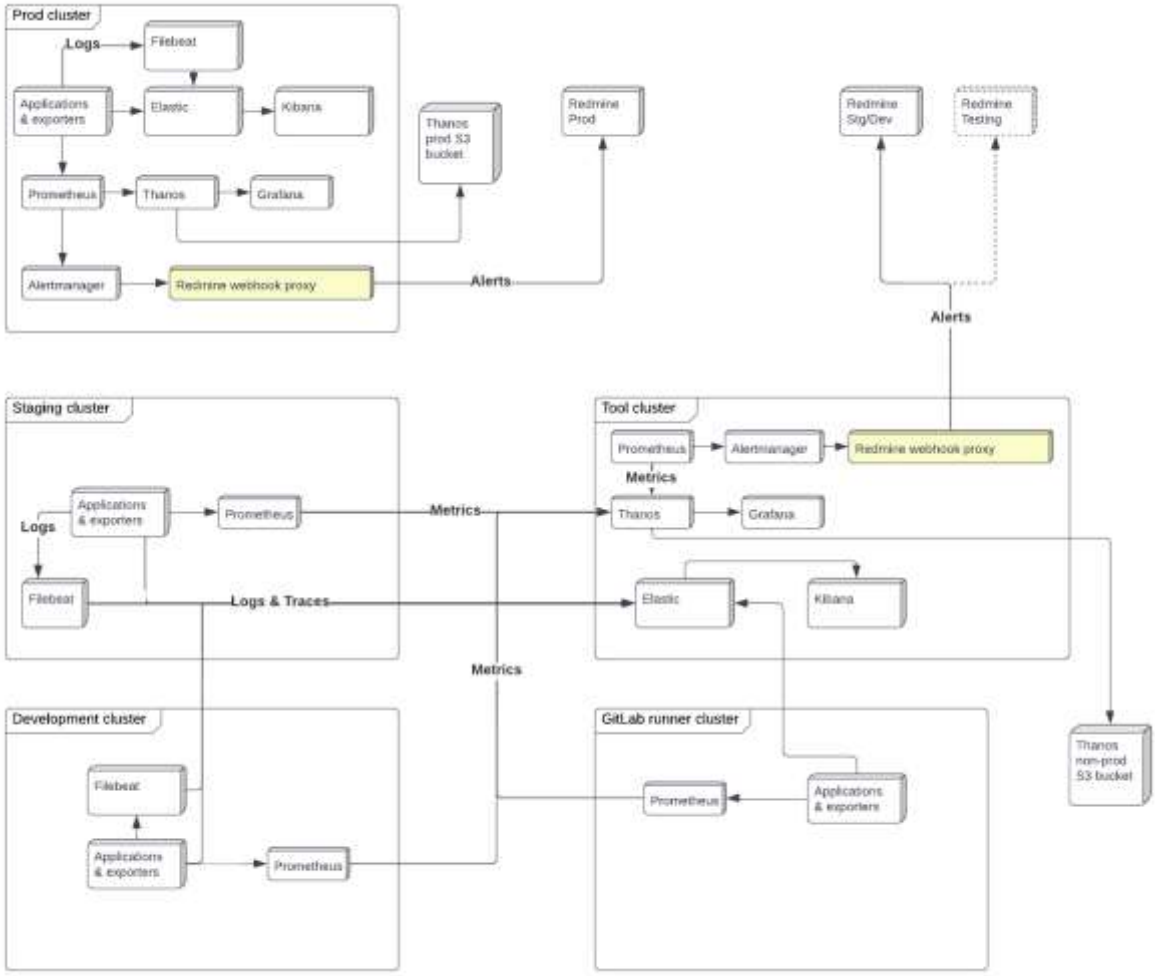
Mezi hlavní druh metrik budou patřit zejména informace o stavu, výkonu a vytížení modulu. Požadované metriky budou specifikovány v zadávací dokumentaci modulu. Dodavatel může po domluvě se zadavatelem dodat i další metriky související s implementací daného modulu pro informovanější monitoring a ladění systému. **Dodavatel musí dodat dokumentaci poskytovaných metrik (jednotky, způsob měření, přesná sémantika atd.) a doporučení pro nastavení alertů pomocí PromQL.**

1.2.14 Diagramy primárních interakcí modulu

Následující diagram shrnuje hlavní komponenty infrastruktury, se kterými budou aplikační moduly přímo interagovat (mimo komunikaci s ostatními aplikačními moduly). Modře zvýrazněné komponenty vyžadují přímou integraci do kódu modulu.



Následující diagram popisuje tok metrik, logů, traců a alertů celým systémem. Sdílený monitoring pro Development a Staging cluster se nachází v Tool clusteru. Produkční cluster obsahuje svůj oddělený monitoring. Komponenty vyvíjeny interně na UK jsou zvýrazněny žlutě. Jednotlivé vyvíjené moduly jsou na diagramu zahrnuty pod komponentou “Applications & exporters”.



1.2.15 Reporting

Každý aplikační modul musí poskytovat data pro sdílenou reporting infrastrukturu v takové míře, aby bylo možné splnit požadavky na reporting od stakeholderů – typicky se bude jednat o data z monitoringu a auditu, nicméně ve vybraných případech i o jiný způsob vhodného poskytnutí dat pro potřeby požadovaných reportů, např. ve formě API definovaného k tomuto účelu. Veškeré detaily budou upřesněny v zadávací dokumentaci modulu.

1.2.16 Dostupnost a spolehlivost (High Availability)

Všechny aplikační moduly by měly podporovat provoz ve vysoké dostupnosti (HA). Každý modul by tedy mělo být možné provozovat ve vícero instancích zároveň a umožnit tak i horizontální škálování systému. Přesnější výkonové požadavky a propustnost jednotlivých endpointů (množství požadavků za časové okno, rychlost odezvy, atd.) budou specifikovány v zadávací dokumentaci daného modulu. V ideálním případě by všechny moduly měly být implementovány bezstavově (stateless) z důvodu jednoduché škálovatelnosti.

V některých konkrétních případech lze uvažovat i o implementaci bez podpory vysoké dostupnosti (např. kvůli zvýšení celkové propustnosti pro danou transakci za předpokladu, že požadovaný výkon není technologicky možné jinak docílit), nebo o implementaci vysoké dostupnosti modulů

vyžadujících stavovost pomocí tzv. sticky sessions (nikoliv plně stateless). Každá taková výjimka však musí být schválena zadavatelem, a to pouze ve specifických a dobře odůvodnitelných případech.

Změnu konfigurace modulu a nasazení nové verze by mělo být ve většině případů možné bez vypnutí modulu (rolling update) pro zabezpečení co nejvyšší dostupnosti systému. Samozřejmě v odůvodněných případech je možné nasazení nové verze také po čas plánovaného výpadku (downtime), například v situacích vyžadujících náročnější změnu struktury relační databáze, migrace dat, atd.

1.3 Obecné technické požadavky na moduly

Každá aplikační doména (z pohledu koncového uživatele modul v Portálu SIS) bude implementována nejméně pomocí dvou modulů z pohledu runtime:

- **„Backendový" modul** – zabezpečující implementaci byznys logiky, přístup k perzistentním datům, logickou autorizaci specifickou pro danou doménu, poskytování strukturovaných dat pro „frontendový" modul generující uživatelské HTML rozhraní
- „**Frontendový" modul** – zabezpečující HTML reprezentaci uživatelského rozhraní vloženého do hlavního Portálu SIS

Všechny moduly poběží v kontejnerizovaném prostředí Kubernetes jako nezávislé kontejnery. Pro každý modul (kontejner) bude vytvořen separátní GitLab projekt a CI/CD pipeline.

1.3.1 Technické požadavky na backendové moduly

Pro implementaci backendových modulů jsou povoleny následující technologie:

- Java/Kotlin (+ Spring Boot)
- TypeScript + Next.js/Nest.js
- Python + WSGI/ASGI
- C# + ASP.NET Core

1.3.2 Technické požadavky na frontendové moduly

Stěžejní základ pro implementaci frontendu pro specifickou portálovou aplikaci je zadavatelem dodána UX/UI analýza, jakožto součást zadávací dokumentace modulu. Pro dosažení jednotného designu bude dodavateli poskytnutá také knihovna společných komponent, vybudovaná s použitím konceptu Atomic Design[1], která bude postupně rozšiřována o nové komponenty a šablony.

Pro server-side část implementace frontendových modulů jsou povoleny stejné technologie jako pro implementaci backendových modulů, uvedené výše.

Pro browser-side část implementace frontendových modulů jsou povoleny následující technologie:

- HTML/CSS + Vanilla JS
- TypeScript + React

Uživatelská rozhraní musí podporovat zobrazování v různých jazykových mutacích. Jazyk zobrazení bude vyplývat z nastavení uživatelského profilu v rámci centrálního modulu Portál, který informaci o požadovaném jazyku přepošle specifickému frontend modulu pomocí cookie s názvem **sis_lang**

v HTTP hlavičce požadavku. Není-li určeno jinak, modul musí poskytovat českou i anglickou mutaci uživatelského rozhraní. Případné další požadované jazykové mutace budou specifikovány v zadávací dokumentaci modulu.

Kromě jazyku zobrazení bude Portál posílat i následující HTTP hlavičky:

Jmého HTTP hlavičky	**Popis**
X-Correlation-Id	Unikátní ID zpracovávaného požadavku, které bylo přirazeno pomocí platformy API brány při vstupu do systému. Všechny moduly v rámci architektury si při následných voláních tento identifikátor odevzdávají v HTTP hlavičkách.
X-Base-Path	Každá portálová aplikace bude mít přidělený nejaký namespace pro vytváření unikátních cest pro context-path HTTP requestů. Prefixem namespacu aplikace bude vždy namespace centrálního Portálu, jehož hodnota je portálovým aplikacím odevzdávaná pomocí této HTTP hlavičky. Příklad zasílané basePath: „/portal“. Jestliže přiděleným identifikátorem portálové aplikace je například „app1“, bude finální namespace aplikace vypadat následovným způsobem: „/portal/app1/“. Veškeré další zdroje, které prohlížet bude potřebovat pro Aplikaci1 dotáhnout musí jako prefix context-path používat tento namespace, na základě kterého Portál ví, které portálové aplikaci má přeposlat požadavek na daný zdroj.
X-Log-Level	Dynamicky určený logovací level pomocí kterého má modul logovat aktuálně procesovaný požadavek. V případě podřazeného navazujícího volání dalších modulů musí modul tuto hlavičku dalším modulům také přeposlat. Možné hodnoty jsou ERROR, WARN, INFO, DEBUG, TRACE.
X-Parent-Id	ID přímo nadřazeného rodičovské logovací zprávy. Jestliže je aktuální požadavek vykonáván jako podřazená operace nejaké nadřazené části aplikační logiky, pošle nadřazený modul nebo komponenta modulu identifikátor rodičovské logovací zprávy (kořen logovacího podstromu) pomocí HTTP hlavičky **X-Parent-Id**, nebo pomocí interních komunikačních cest mezi interními komponenty moduly. Od podřazené aplikační logiky se pak očekává vyplňování tohoto identifikátoru do všech vytvářených zpráv až do momentu kdy nevznikne nový logovací podstrom. Způsob vytváření a užití parent id je také demonstrován pomocí zdrojového kódu vzorových modulů, které budou dodavateli zpřístupněny v rámci přístupu do GitLab.

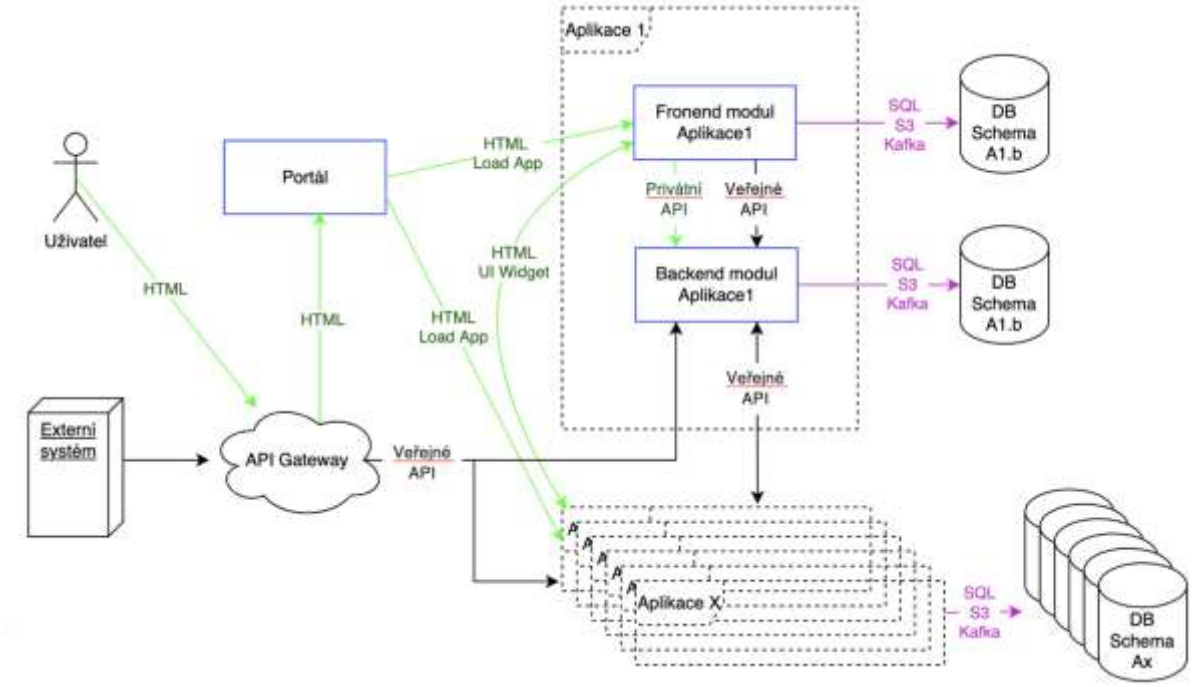
1.3.3 Interakce mezi moduly

Následující obrázek ukazuje schéma jednotlivých API volání pro dva různé případy užití.

1. Prvním je uživatel (osoba) přistupující na centrální Portál SIS s cílem vstoupit do portálové Aplikace1. Moduly Aplikace1 poskytnou Portálu HTML kód, který Portál vloží do celkového rámce HTML stránky vrácené prohlížeči uživatele. Úkolem frontendového modulu Aplikace1 je zabezpečit uživatelské HTML rozhraní pro Portál, a to primárně za

pomoci backendového modulu Aplikace1, implementujícího aplikační logiku Aplikace1, případně jiných modulů systému (ať již backendových poskytujících např. REST API, nebo frontendových poskytujících embedovatelné HTML widgety). Volání jejichž návratovou hodnotou je uživatelské rozhraní v HTML formátu je vyznačeno zelenou barvou. Na tomto místě je potřeba zmínit, že každý backendový modul může pro svůj frontendový modul vystavovat kromě zadávací specifikací předepsaného veřejného (aplikačního) API také privátní „frontendové“ API. Aplikační (veřejné) i neveřejné API pak společně poskytují frontendovému modulu potřebné funkce a data.

2. Druhým případem užití je přímé volání veřejného API systému externí aplikací. Volání veřejného (aplikačního) API jednotlivých modulů je vyznačno černou barvou.



Příklad jednoduché ukázkové portálové aplikace poskytující uživatelské HTML rozhraní pro integraci do centrálního Portálu bude dodavateli zpřístupněno v rámci přidělení přístupu do platformy GitLab.

2 Obecné požadavky na uživatelská rozhraní

2.1 Obecné principy uživatelského rozhraní

Součástí zadávací dokumentace pro uživatelské rozhraní jsou:

1. Graficky zpracované náhledy pro rozpracovaný modul.
2. Samostatně zpracované komponenty, které se v modulu vyskytují

Grafická podoba zpracovaných náhledů není finální grafikou, ale pouze pracovním grafickým návrhem UI. Rozpracování finalní grafiky není součástí zadání jednotlivých modulů. Přesto obrazovky a komponenty budou mít jednotný vzhled a společně budou tvořit jednotný design systému, jehož správná a důsledná implementace je potřebná pro rychlé a správné nasazení finální grafiky.

Komponenty UI, ze kterých se budou skládát moduly v jednotlivých zadáních budou tvořit společnou knihovnu, která se bude postupně doplňovat s každým nově zadaným modulem. Všechny komponenty UI, přitomné v návrzích UI budou mít svůj zdroj v knihovně komponent.

Součástí knihovny budou "placeholdery" prvků určené pro poskytování zpětné vazby UI v případě delší odezvy modulů. Finální podoba těchto prvků bude řešená v rámci grafického návrhů. Základní zpětná vazba (spinner) má být uživateli poskytnutá všude tam kde se očekává odezva delší než 1000 ms.

2.2 Předání grafických podkladů pro UI

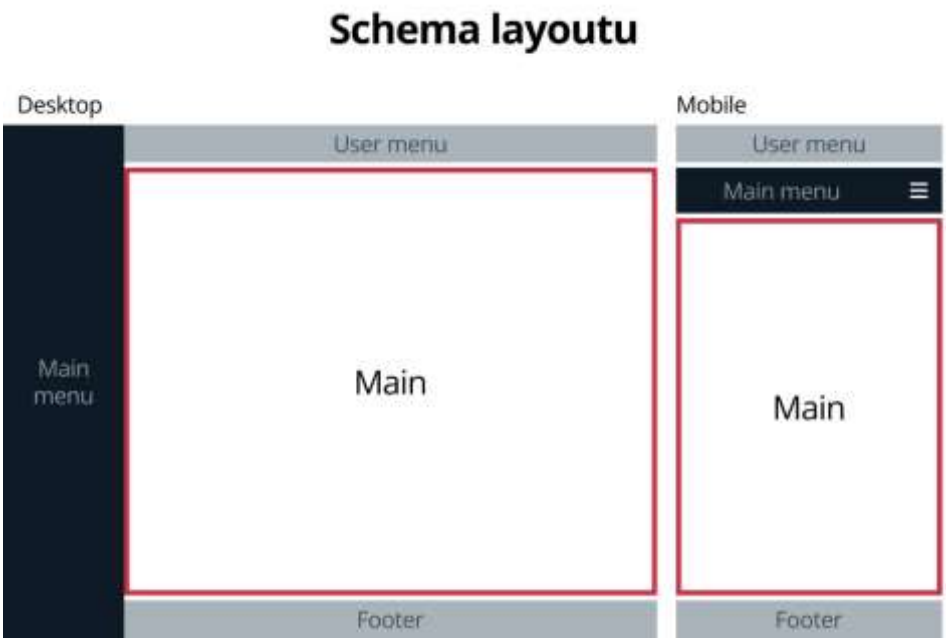
Zadávací dokmentace pro UI konkrétního modulu bude obsahovat:

1. Exporty návrhů ve dvou variantách - desktop (šířka 1512 px) a mobil (šířka 414 px) a ve dvou formátech JPG a PDF.
2. Přístup ke zdrojovým souborům návrhů (Figma)Odkaz na soubor s komponenty společnými pro všechny doposud zpracované moduly v rámci projektu (Figma).
3. Flowchart obrazovek
4. Interaktivní prototyp ve Figmě

2.3 Uživatelské rozhraní - popis základního layoutu

V této části jsou popsané vlastnosti layoutu z hlediska responzivního chování aplikace.

Uživatelské rozhraní modulů Studijního Informačního Systému vychází z modulární architektury systému a je navrženo tak, aby umožňovalo postupné začleňování nově vytvořených modulů do aplikace. Skládá se ze dvou ovládacích panelů – jeden pro volby uživatelského profilu a druhý pro navigaci v modulech, a ze sekce, určené pro zobrazení zvoleného modulu. Tyto prvky tvoří jednotný základní layout aplikace.



Pro účely zadání je podstatná definice responzivního chování sekce, do níž bude umístěn HTML kód modulu. Obsah zbývajících sekcí layoutu dodá centrální webový Portál.

2.3.1 Základní prvky uživatelského prostředí

- Hlavní navigační panel (MAIN MENU), který obsahuje branding a hlavní nabídku, která slouží k výběru jednotlivých modulů.
- Panel uživatelského menu (USER MENU) – obsahuje informace o uživateli, volby pro přepínání mezi rolemi uživatele a volby pro nastavení dalších preferencí na uživatelské úrovni
- Jednotná patička (FOOTER)
- Hlavní panel modulu (MAIN), ve kterém se zobrazuje aktuální modul

2.3.2 Responzivita a breakpoints

Uživatelské rozhraní je založené na fluidním layoutu a počítá s intenzivním využitím flex CSS komponenty.Breakpoints (BP) jsou minimalizované a jsou uplatněné pro každou ze základních komponent zvlášť. Nevztahují se nutně k velikostem konkrétních zařízení, ale vychází spíše z potřeby komponent. BP pro základní sekce layoutu jsou následovné:

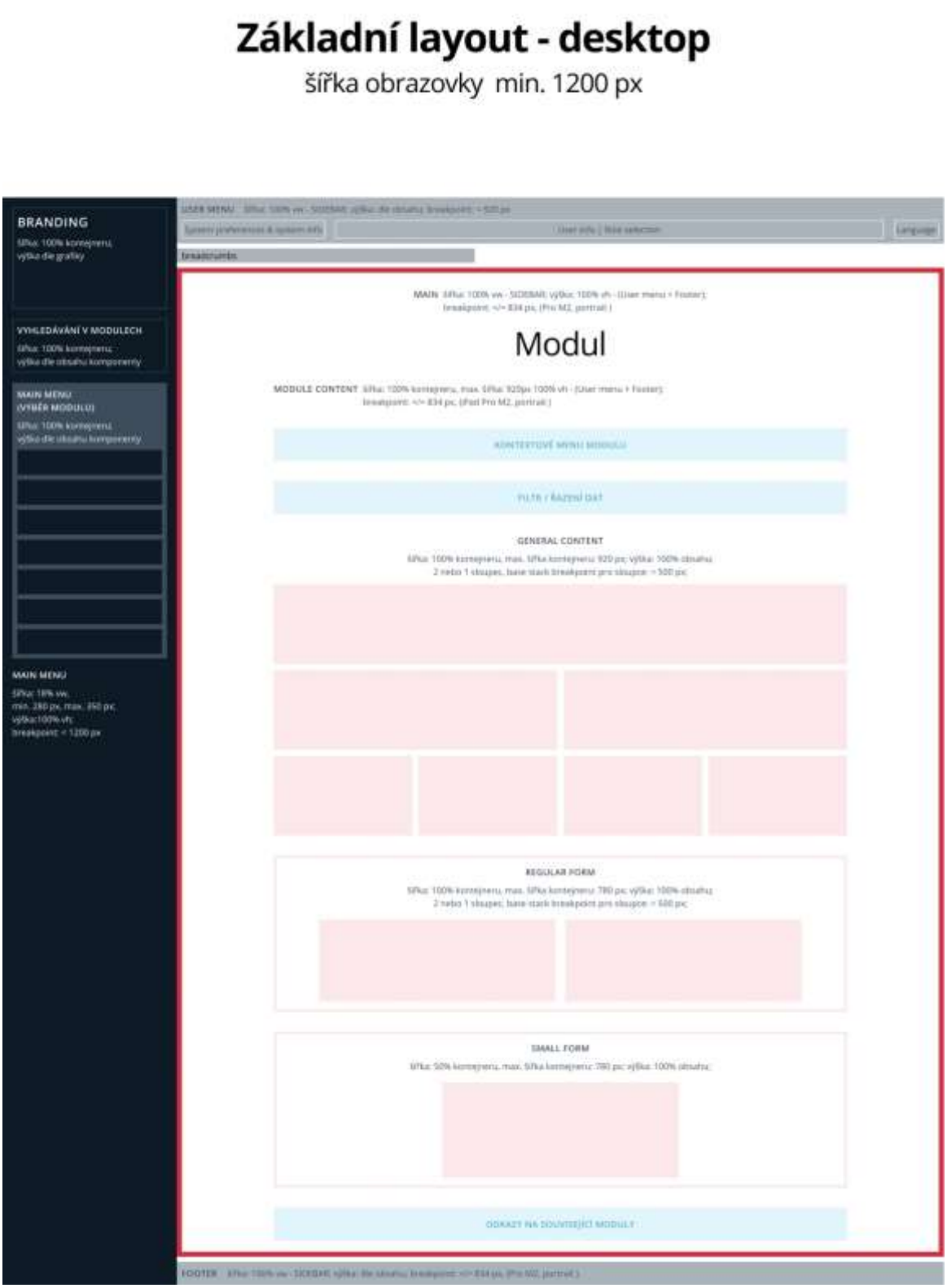
1. MAIN MENU BP: **max-width: 1199px**
    a. Pod tímto BP se panel zobrazuje nad hlavním obsahem a nabídka je schovaná pod ikonou.
    b. Nad tímto BP se panel zobrazuje vlevo od hlavního obsahu
2. USER MENU BP**: max-width: 799px**
    A. Pod tímto bodem jsou některé textové položky sdružené pod ikonku s dropdown menu
    B. Nad tímto bodem jsou tyto položky umístěné přímo v liště
3. MAIN BP: **max-width: 499px** – tento BP se týká především chování flex prvků. Od tohoto bodu dolů se ruší sloupce ve formulářích a maximální počet sloupců v ostatních komponentách se redukuje na 2.

Potřeba dalších BP může přibývat s rozpracováním konkrétních modulů, přesto snaha bude držet se co nejvíce jíž zmíněních.

2.3.3 Dimenze základních sekcí layoutu

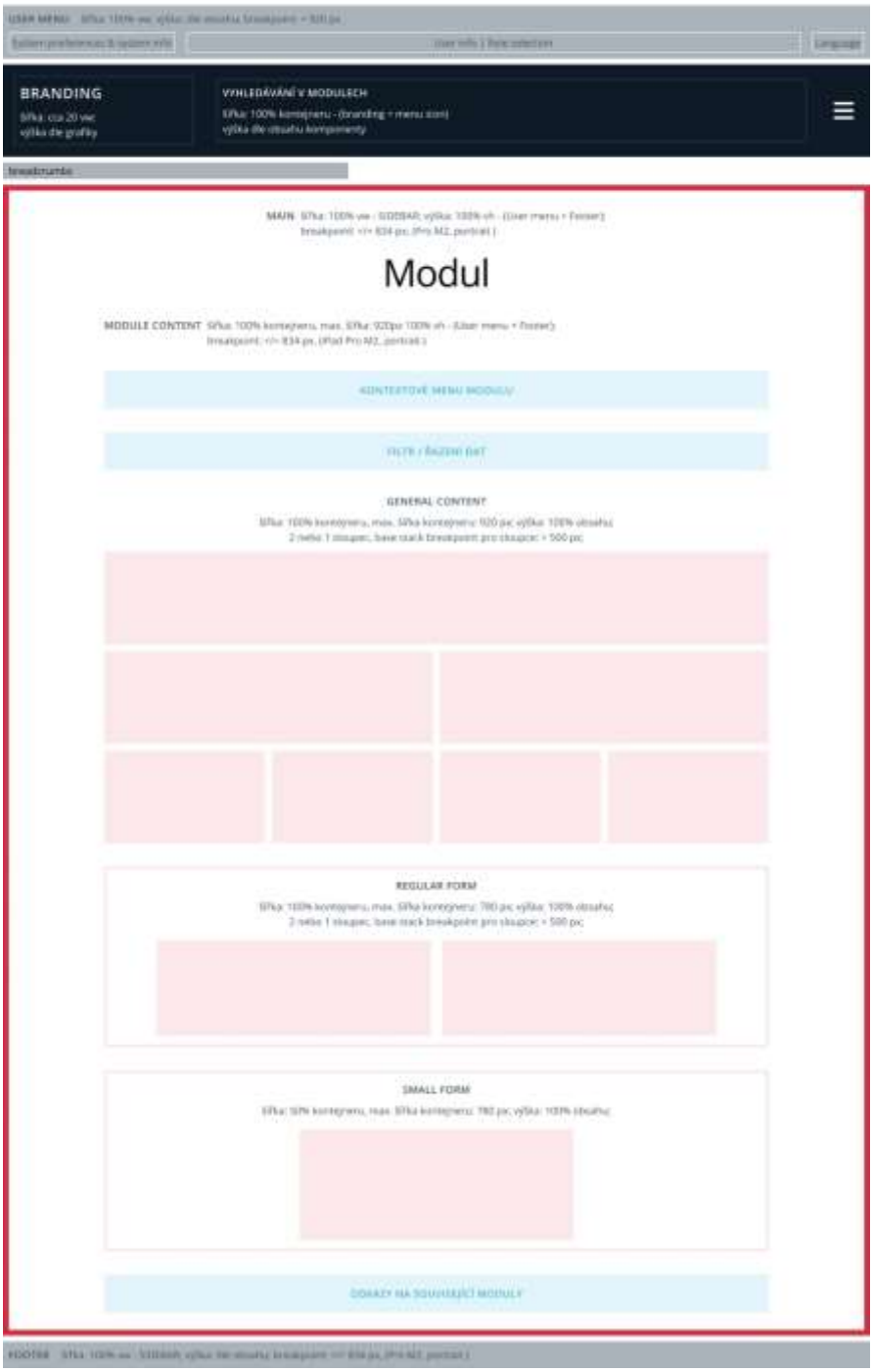
1. MAIN MENU **18% vw, max. šířka 280px**
2. Komponenty v sekci MAIN **100% nadřazeného prvku, max. šířka 920px**
3. Formuláře **100 % nadřazeného prvku, max. šířka 780px**
4. Malé formuláře **100 % nadřazeného prvku, max. šířka 460px**

2.3.4 Grafické náhledy



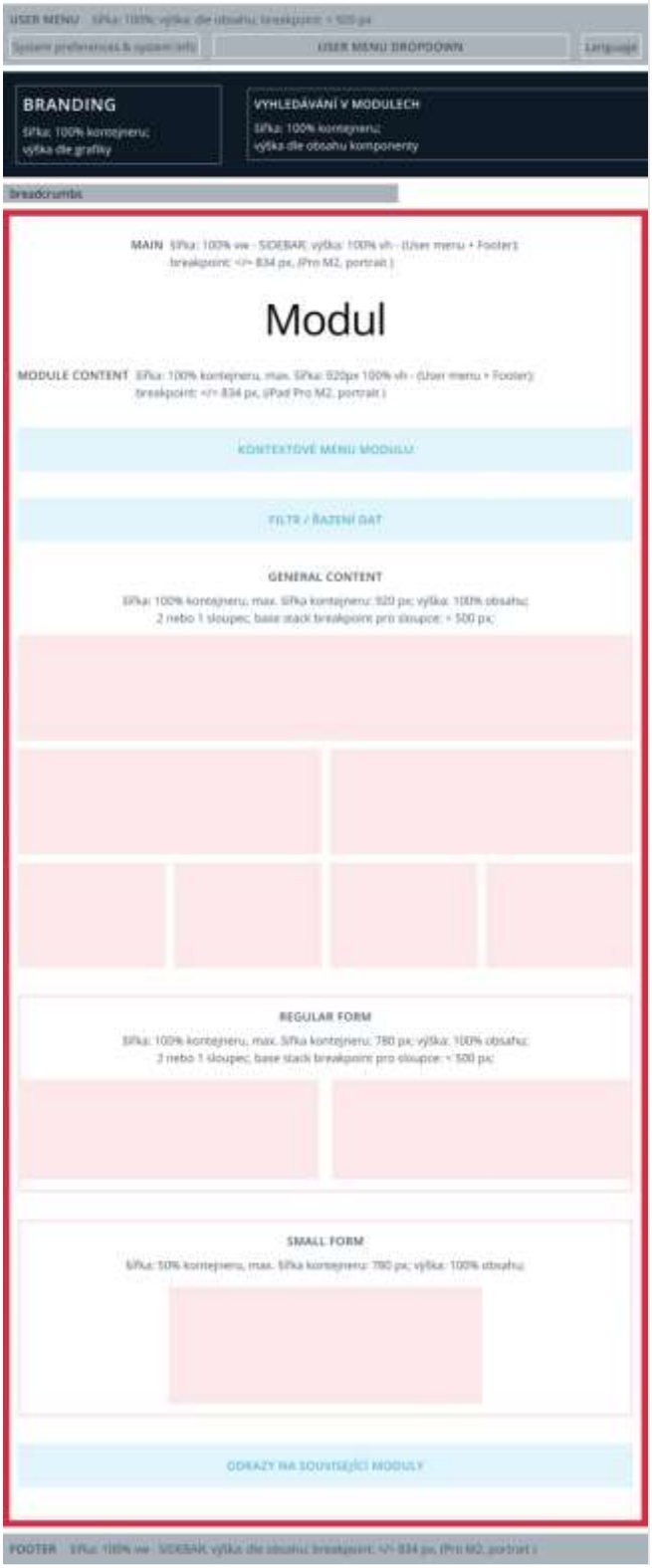
Základní layout - tablet

šířka obrazovky < 1200 px



Základní layout - small screen

šířka obrazovky < 800 px



Základní layout - mobile

šířka obrazovky
</= 500 px

2.4 Požadavky na optimalizaci

Návrh uživatelského rozhraní (UI) aplikace musí brát v potaz fakt, že aplikace bude sloužit uživatelům s velmi odlišnými požadavky na zařízení a prohlížeče (zaměstnanci školy vs. studenti), proto je optimalizace většiny případu užití pro mobilní zařízení i pro desktop počítač/notebook stejně důležitá.

1) Desktop
   a) Windows / Chrome
   b) Windows / Edge
   c) Windows / Firefox
   d) OS X / Safari 15+
   e) OS X / Chrome
   f) OS X / Firefox
2) Mobile
   a) Android 10+ / Chrome
   b) iOS 13+ / Safari
   c) Android 10+ / Samsung internet

Tento seznam se zakládá na veřejných datech o nejpoužívanějších OS/prohlížečů v ČR a na interních uživatelských statistikách UK.

Minimální velikost obrazovky, pro kterou je aplikace určená je 375 x 667 pro mobilní zařízení a 768 x 1024 pro tablet. Aplikace se má zobrazovat s v souladu s grafických návrhem jak na desktopovém zařízení, tak na obrazovkách těchto velikostí.

2.5 Požadavky na přístupnost a použitelnost

Aplikace musí splňovat úroveň shody AA dle WCAG 2.0WCAG 2.0. Splněné požadavky na přístupnost je potřeba doložit protokolem z automatické kontroly specializované on-line služby, která bude specifikována dodatečně s dodavatelem. V rámci protokolu jsou pro dodavatele závazné parametry, které jsou předmětem jeho kompetencí a součástí jemu určeného zadání.

Zde je minimální závazný seznam parametrů schody s WCAG 2.0, relevantní pro dodavatele sw:

1. Správná sémantická struktura HTML kódu
2. Správné zvětšení obsahu při zvětšení/zmenšení
3. Podpora navigace pomocí klávesnice
4. Správné typy formulářových polí dle HTML5
5. Pokud se obsah automaticky mění, uživatel musí mít možnost pozastavit automatické změny, případně přepnout na jejích manuální ovládání
6. Možnost procházet obsah pomocí čtečky obrazovky
7. Možnost přeskočit navigaci pomocí klávesnice
8. Automaticky přesouvat focus na modální okna, vyskakovací okna, upozornění atd.
9. Prevence automatického odhlášení uživatele.
10. Prevence automatického přehrávání zvuku a videa, pokud to není žádoucí chování.
11. Důsledné použití atributu HTML lang
12. Používat atributy ARIA tam, kde struktura HTML není jednoznačná
13. Alternativní způsoby konzumace multimediálních souborů (transkripce dialogů a popis obsahu)
14. Data pro grafy, tabulky, mapy, SVG atd. konzumovatelná prostřednictvím asistenčních technologií

3 Obecné požadavky na strojová rozhraní modulů

Z důvodu vysoké modularizace systému je nutné sjednotit podobu strojových (programovacích) rozhraní jednotlivých modulů. Pod pojmem strojové rozhraní modulu rozumíme rozhraní umožňující strojovou (programovanou) interakci s modulem. Strojovou interakcí s modulem rozumíme komunikaci modulu s jinými částmi systému (jiné moduly, front-end) za účelem výměny dat a volání operací. Rozlišujeme několik druhů strojových rozhraní popsaných v následující tabulce.

Popis druhu strojového rozhraní	Název druhu strojového rozhraní	Vlastník	Specifikace
Volání aplikačních operací poskytovaných modulem pro potřeby výkonu funkcionalit systému zahrnujících CRUD operace nad daty spravovanými modulem a operace implementující části byznys logiky zajišťované modulem.	Veřejné aplikační strojové rozhraní (veřejné API) – určené pro jiné moduly systému	UK	OpenAPI + JSON Schema + mapování na sémantický byznys slovník
	Privátní aplikační strojové rozhraní (privátní API) – určené pro front-end část modulu	Dodavatel	OpenAPI + JSON Schema
Šíření a příjem notifikací o událostech, které nastaly uvnitř modulu a mají být pozorovatelné vně modulu, zahrnující události s významem odpovídajícím událostem nastávajícím v rámci byznys logiky zajišťované modulem.	Notifikační rozhraní	UK	JSON Schema + mapování na sémantický byznys slovník
Přístup k datům spravovaných modulem pro potřeby datově řízené univerzity	Datové rozhraní	UK	Datové schéma (CSVW, JSON Schema, …) + mapování na sémantický byznys slovník

Modul musí poskytovat alespoň jedno veřejné API. Může obsahovat více různých veřejných API lišících se specifikací, soukromých API, notifikačních rozhraní a datových rozhraní.

Vlastník rozhraní je plně odpovědný za jeho specifikaci sestávající z definice struktury programovacího rozhraní (definice operací, definice datových struktur) a dokumentace programovacího rozhraní. Nikdo jiný než vlastník rozhraní není oprávněn jeho specifikaci změnit.

Následují konkrétní obecné požadavky na jednotlivé druhy programovacích rozhraní. Požadavky jsou doplněny příkladem hypotetického modulu Zábava (Amusement) umožňujícího studentům nakupovat vstupenky na koncerty. Jedná se čistě o hypotetický příklad modulu, který není součástí zadání.

3.1 Obecná pravidla

Nejprve uvedeme obecná pravidla, která platí pro všechny druhy rozhraní.

3.1.1 Jmenné konvence

- Všechna URL endpointů rozhraní jsou definována relativně vůči základnímu URL SIS UK (tzv. base URL). Toto základní URL se může lišit dle prostředí, a proto jej neuvádíme a není pro návrh rozhraní podstatné.
- Všechny složky relativních URL (kroky cest, parametry) jsou uváděny v angličtině v kebab-case-notaci.
- Jediným povoleným formátem používaným veřejnými, privátními a notifikačními rozhraními je formát JSON. Jediným povoleným jazykem pro definici datových struktur pro formát JSON je jazyk JSON Schema ve verzi 2020-12 nebo pozdější.
- Názvy prvků datových struktur (klíče JSON, případně elementy a atributy XML a sloupce CSV) jsou uváděny v angličtině v kebab-case-notaci.

3.1.2 Chování klientů rozhraní

- Všechna URL endpointů rozhraní slouží jako identifikátory i lokátory. Nejsou však nositeli aplikační/byznysové sémantiky. V žádném případě není možné je na klientovi parsovat a na výsledku parsování stavět aplikační logiku.

3.2 Obecné požadavky na veřejná API

Veřejné API nabízí sadu operací, které umožňují vykonávání byznys operací, ze kterých sestávají byznys procesy/případy užití definované v kapitole byznys analýzy.

Veřejné API je z technického hlediska RESTful API, tj.

- reprezentuje byznys entity jako zdroje
- umožňuje CRUD manipulaci se zdroji,
- používá JSON pro reprezentaci zdrojů,
- používá URL pro identifikaci zdrojů,
- je bezstavové,
- používá HTTP metody POST, GET, PUT, PATCH, DELETE, OPTIONS a HEAD pro CRUD manipulaci se zdroji,
- implementuje HATEOAS princip (Hypertext As The Engine of Application State).

Veřejná API jsou verzovaná. Verze číslujeme sémanticky (semantic versioning).

3.2.1 Zdroje a jejich typy

Zdroj je základním technickým konceptem REST. Zdrojem je konkrétní nebo abstraktní věc, která je byznys entitou vyplývající z byznys analýzy, o níž chceme prostřednictvím veřejného API zpřístupnit údaje a umožňovat s ní CRUD manipulaci.

Uvažujeme následující typy zdrojů:

- zdroj typu objekt
  - reprezentuje byznys entitu nebo její část
    - např. student, koncert, interpret, vstupenka
- zdroj typu kolekce objektů
  - reprezentuje kolekci všech byznys entity daného typu
    - např. kolekci všech studentů, kolekci všech koncertů, kolekci všech interpretů, kolekci všech vstupenek
  - pro objekty v kolekci obsahuje pouze základní údaje, nikoliv jejich kompletní detail. Mezi základními údaji je vždy:

        - URL objektu (jedná se o zdroj)
        - typ případně typy objektu
        - byznysový identifikátor objektu
- zdroj typu vztah
    - reprezentuje byznys vztah dané byznys entity s jinou byznys entitou
        - např. vstupenka studenta (vztah studenta s jeho vstupenkou), majitel vstupenky (vztah vstupenky s jejím majitelem-studentem) nebo interpret vystupující na koncertu (vztah koncertu s interpretem, který na koncertu vystupuje)
- zdroj typu kolekce vztahů
    - reprezentuje kolekci vztahů, které reprezentují pro danou byznys entitu všechny její byznys vztahy odpovídající dané asociaci v byznys modelu
        - např. kolekce všech vstupenek daného studenta (0..*), kolekce všech majitelů dané vstupenky (0..1) nebo kolekce všech interpretů vystupujících na koncertu (1..*)
- zdroj typu ovladač (controller)
    - reprezentuje specifickou byznys operaci (akci) se zdrojem typu objekt, kterou není možné realizovat jako CRUD operaci
        - např. zneplatnění vstupenky.

Zdroj typu kolekce vztahů je kolekcí, jejíž prvky nereprezentují byznys entity vztažené k dané byznys entitě, ale vztahy samotné.

Zdroje typu ovladač by měly být využívány co nejméně – pouze v případech, kdy není možné danou byznys operaci realizovat jako CRUD operaci nad zdrojem typu objekt nebo kolekce. Např. byznys operaci změny majitele vstupenky na koncert může být realizována jako operace Update (HTTP PUT) na zdroji typu objekt reprezentujícím majitele vstupenky. Nezavádíme proto zdroj typu ovladač pro změnu majitele vstupenky. Zdroj typu ovladač může být nutný pro změnu stavu byznys entity, která není přímo realizovatelná jako prostý Update nějaké její části. Např. se může jednat o změnu stavu byznys entity na stav zneplatněno, tj. operace zneplatnění vstupenky. Alternativou by bylo vytvoření zdroje typu objekt reprezentujícím platnost vstupenky. Pokud ale takový zdroj nemáme a nemůžeme jej z nějakého důvodu vytvořit, musíme vytvořit zdroj typu operace. Vždy ale preferujeme variantu vytvoření potřebných zdrojů, je-li to možné.

3.2.2 Pravidla pro určování zdrojů a tvorbu jejich URL

Každý zdroj je vystaven prostřednictvím endpointu na URL, které je veřejné v rámci systému, tj. lze na něj přistoupit z jiných částí systému.

URL zdroje je identifikátorem zdroje v rámci celého systému a zároveň jeho lokátorem. URL zdroje má následující tvar:

```
[základní URL]/[relativní URL modulu]/public-api/[relativní URL veřejného
        API]/[verze API specifikace]/[relativní URL v rámci API]
```

kde

- `[základní URL]` je HTTP URL (tzv. base URL) společné pro všechny endpointy všech služeb a je určeno technickou infrastrukturou systému
- `[relativní URL modulu]` je relativní URL modulu, obvykle v podobě názvu modulu
- `[relativní URL veřejného API]` je relativní URL veřejného API v rámci modulu, obvykle v podobě názvu veřejného API, který musí být v rámci modulu unikátní. Ve

výjimečných případech může být vynecháno, ale je to považováno za antipattern. Pokud v budoucnu dojde k rozhodnutí vytvořit druhé veřejné API, bude toto druhé muset být pojmenováno, čímž dojde k nekonzistenci v konvenci tvorby URL
- `[verze API specifikace]` dle semantic versioning. Změna verze znamená změnu veřejného API. Změna modulu při zachování API nemění její verzi.
- `[relativní URL v rámci API]` je relativní URL daného zdroje, s nímž je možné prostřednictvím veřejného API manipulovat. Je unikátní v rámci veřejného API.

Základními pravidly při určování zdrojů jsou následující pravidla.

- Pro typ byznys entity popsaný v byznys analýze určíme
  - zdroj typu kolekce objektů, který reprezentuje kolekci všech instancí tohoto typu. Platí, že `[relativní URL v rámci API] = [název typu v množném čísle]`, např.:
    - `students`
    - `interprets`
    - `tickets`
    - `concerts`
  - zdroje typu objekt, které reprezentují jednotlivé instance typu (tj. jednotlivé byznys entity). Platí, že `[relativní URL v rámci API] = [název typu v množném čísle]/{byznysový identifikátor pro tento typ byznys entity}`, např.:
    - `students/12345678`, kde 12345678 je číslo osoby (UKČO) studenta
    - `interprets/pokac`, kde pokac vyplývá z názvu interpreta, který je unikátní
    - `tickets/v-xzfhrd`, kde v-xzfhrd je kód vstupenky
    - `concerts/4758`, kde 4758 je číslo koncertu
- Pro asociaci mezi dvěma typy byznys entit popsanou v byznys analýze a její navigovatelný konec (v UML diagramech označeno šipkou, T je typ navigovatelného konce, S je druhý typ v asociaci) určíme
  - zdroj typu kolekce vztahů, který reprezentuje kolekci všech vztahů modelovaných asociací mezi danou instancí typu S a instancemi typu T. Platí, že `[relativní URL v rámci API] = [relativní URL v rámci API zdroje typu objekt reprezentujícího danou instanci typu S]/[název role asociačního konce nebo název asociace jako podstatné jméno]`. V případě maximální kardinality asociačního konce == 1 je název v relativním URL uveden v jednotném čísle. V případě > 1 je uveden v množném čísle. Např.:
    - `students/12345678/tickets`
    - `tickets/k-xzfhrd/owner`
    - `concerts/4758/interprets`
  - zdroje typu vztah, které reprezentují jednotlivé vztahy modelované asociací ve směru navigovatelného konce. Platí, že `[relativní URL v rámci API] = [relativní URL v rámci API zdroje typu objekt reprezentujícího danou instanci typu S]/[název role asociačního konce nebo název asociace jako podstatné jméno]/{byznysový identifikátor pro typ T}`. V případě maximální kardinality asociačního konce == 1 je název role v relativním URL uveden v jednotném čísle. V případě > 1 je uveden v množném čísle, např.:
    - `students/12345678/tickets/k-xzfhrd`
    - `tickets/k-xzfhrd/owner/12345678`
    - `concerts/4758/interprets/pokac`

- Pro specifickou byznys operaci/akci se zdrojem typu objekt, kterou není možné realizovat jako CRUD operaci, určíme zdroj typu ovladač. Platí, že `[relativní URL v rámci API] = [relativní URL v rámci API zdroje typu objekt]/[název byznys operace/akce jako podstatné jméno v jednotném čísle]`, např.:
    - `tickets/k-xzfhrd/invalidation`

Výše uvedené příklady jsou relativní URL v rámci API. Jejich plné relativní URL v rámci systému je pak tvořeno dle výše uvedeného předpisu relativním URL modulu, relativním URL API v rámci modulu, verzí API a relativním URL v rámci API. Náš příklad je v kontextu modulu Zábava, tj. plná relativní URL jsou např.:

- `amusement/public-api/basic/1.0.0/students`
- `amusement/public-api/basic/1.0.0/students/12345678`
- `amusement/public-api/basic/1.0.0/students/12345678/tickets`

Výše uvedené příklady ukazují, že byznysové identifikátory v relativních URL mohou být nečíselné. Pokud je to možné, preferujeme číselné byznysové identifikátory. Jejich (ne)existence však vyplývá z byznys analýzy. Pokud není v byznys analýze identifikován žádný číselný identifikátor, volíme nečíselný.

Použití byznysových identifikátorů je nutné. Pokud bychom použili umělé databázové primární klíče, hrozilo by pronikání interní databázové logiky modulu do systému, což je zakázáno.

Výše uvedená pravidla neaplikujeme maximalisticky. Není nutné vytvářet zdroje pro každý typ byznys entity a pro každou byznys asociaci. Pravidla aplikujeme dle potřeby. Není např. nutné vytvářet zdroje pro každý navigovatelný konec každé asociace. V případě jednodušších API můžeme mít dokonce jen jeden zdroj typu kolekce objektů (např. kolekce studentů) a pro každý objekt v kolekci zdroj (např. jednotlivé studenty). Prvky vnitřní struktury zdrojů (např. kontaktní údaje, bankovní účty) nemusíme nutně reprezentovat jako samostatné zdroje. Vždy vycházíme z byznys analýzy a v ní identifikovaných byznys operací.

Pokud ale nějaký zdroj určíme, musíme ho určit podle jednoho z výše uvedených pravidel. Určování jiných zdrojů podle jiných pravidel není povoleno.

3.2.3 Pravidla pro návrh operací pro manipulaci se zdroji

Definice CRUD operací musí vyplývat z byznys analýzy. Každá určená CRUD operace je mapována na odpovídající byznys operaci/akci popsané v byznys analýze. Při určování CRUD operací pro manipulaci s jednotlivými typy zdrojů se řídíme následující tabulkou.

	Kolekce objektů	**Objekt**	**Kolekce vztahů**	**Vztah**	**Ovladač**
POST (Create)	Vytvoření nového objektu	N/A	N/A	N/A	Vykonání ne-idempotentní operace na daném objektu
GET (Read)	Čtení kolekce objektů	Čtení detailu objektu	Čtení kolekce se základními údaji o vztažených objektech	N/A	N/A
PUT (Update, příp. idempotentní Create)	N/A	Aktualizace objektu poskytnutím jeho nové reprezentace	Vložení vztahu s objektem do kolekce. Pro kardinality ..1 to znamená zároveň odstranění existujícího vztahu, pokud existuje.	N/A	Vykonání idempotentní operace na daném objektu
PATCH (Update)	N/A	Aktualizace části údajů o objektu poskytnutím nových hodnot vybraných údajů	N/A	N/A	Vykonání idempotentní operace na daném objektu
DELETE (Delete)	N/A	Smazání objektu	N/A	Smazání vztahu	N/A

3.2.4 Pravidla pro stránkování a filtrování kolekcí pomocí parametrů v URL

Operace pro čtení kolekce (GET) může bez dalšího znamenat načtení dat o velkém množství byznys entit. Pokud je potřeba množství omezit, je možné URL zdroje typu kolekce doplnit stránkovacími a filtrovacími parametry.

Stránkovacími parametry jsou

- `_offset`, jehož hodnotou je celé číslo >= 0, určuje pozici prvního prvku kolekce, který bude vypsán ve výsledku,
- `_limit`, jehož hodnotou je celé číslo > 0 které je shora omezeno (každou službou zvlášť), určuje počet prvků, které budou vypsány ve výsledku,
- `_order-by`, jehož hodnotou je název vlastnosti prvků kolekce s primitivní hodnotou, podle které má být kolekce seřazena pro účely stránkování
    - pokud není parametr uveden, vyplývá seřazení z interní reprezentace dat v modulu, přičemž je takové seřazení zdokumentováno v dokumentaci modulu
    - seznam požadovaných atributů dle kterých je možné řadit bude explicitně stanoven v rámci definice API

Dokumentace zdroje typu kolekce uvádí, zda podporuje stránkování a stránkování s explicitní specifikací seřazení vč. seznamu vlastností, podle kterých je možné třídit. Musí se jednat o vlastnosti definované ve specifikaci datové struktury pro reprezentaci kolekce.

Filtrovacími parametry jsou

- `_type`, jehož hodnotou je seznam řetězců, které reprezentují typy prvků, které se mohou vyskytovat v kolekci a dle kterých je možné kolekci filtrovat
- další parametry odpovídající vlastnostem prvků kolekce s primitivní hodnotou, názvy parametrů nesmí začínat znakem _, který je jako prefix rezervovaný

Dokumentace zdroje typu kolekce uvádí, zda podporuje filtrování podle typu, včetně výčtu možných typů, jejich popisu a vazby na typy byznys entit. Dále uvádí, zda podporuje filtrování podle dalších parametrů, včetně výčtu vlastností, které lze použít jako filtrovací parametry. Musí se jednat o vlastnosti definované ve specifikaci datové struktury pro reprezentaci kolekce.

3.2.5 Pravidla pro definici zdrojů a operací pro manipulaci se zdroji

Veřejné API pro manipulaci se zdroji je formálně definováno s pomocí OpenAPI verze 3.0.3 nebo vyšší (dále jen Open API). Definice je vyjádřena jako YAML dokument nebo sada YAML dokumentů, která popisuje tvar URL zdrojů a možné CRUD operace nad těmito zdroji tak, jak byly navrženy dle výše uvedeného postupu. Vstupní parametry, které jsou součástí URL jsou definovány také jako součást specifikace. Vstupní parametry, které jsou předávány v těle požadavků, a výstupní reprezentace zdrojů, které jsou předávány v těle odpovědi, jsou reprezentovány jako JSON dokumenty, jejichž JSON struktura je popsána dle následující kapitoly a je obsažena v definici API nebo v samostatných souborech a je odkazována.

3.2.6 Pravidla pro návrh a popis JSON datových struktur

Zdroje a vstupní parametry CRUD operací předávané v tělech požadavků jsou reprezentovány v datovém formátu JSON. Datová struktura pro reprezentaci zdroje je navržena odvozením z odpovídající části konceptuálního modelu, jehož struktuře odpovídá. Může být doplněna o technické datové prvky bez sémantického významu zachyceného v konceptuálním modelu.

Datová struktura je popsána datovým schématem vyjádřeným v jazyku JSON Schema. Datové prvky definované ve schématu, které nejsou technickými datovými prvky, jsou napojeny na prvky konceptuálního modelu. Datová schémata tak nepopisují pouze syntax ale také sémantiku pomocí pojmů definovaných v konceptuálním modelu z byznys analýzy.

Existuje katalog datových struktur, které je povinné používat při definici nových datových struktur. Katalog definuje základní datové struktury, např. pro reprezentaci specifikace časových údajů, množství, adres, apod.

Operace na zdrojích musí na vstupu (v rámci těla HTTP požadavku) i na výstupu (v rámci těla HTTP odpovědi) data o zdrojích reprezentovat v k tomu navržených datových strukturách dle odpovídajících JSON schémat. Každá operace je tak doplněna o definici vstupní a výstupní datové struktury. Vstupy a výstupy jednotlivých volání operací musí být validní vůči JSON schématům definujících tyto datové struktury. Požadavek na veřejné API s nevalidním vstupem musí být odmítnut s příslušným chybovým HTTP kódem.

3.2.7 Pravidla pro implementaci principu HATEOAS

Dle principu HATEOAS musí být pro daný zdroj specifikována volání možných operací nad daným zdrojem a možných operací nad souvisejícími zdroji způsobem, který nevyžaduje, aby byla byznys

logika volání operace implementována na straně klienta. K implementaci principu HATEOAS lze přistoupit staticky nebo dynamicky.

- Dynamický přístup znamená, že možné operace jsou uvedeny přímo, konkrétně a jednoznačně v reprezentacích zdrojů, které jsou vraceny v tělech odpovědí na volání operací. K tomu lze využít RDF 5988, ale konkrétní technický standard bohužel neexistuje.
- Statický přístup znamená, že možné operace jsou definovány v definici zdrojů. K tomu lze využít OpenAPI konstrukt zvaný Link. Pro získání konkrétních volání je nutné definici možných operací na klientovi interpretovat dle logiky konstruktu Link. Interpretační logika musí být tedy reprezentována na straně klienta, nicméně se nejedná o byznys logiku, takže není porušen princip HATEOAS.

V našem případě volíme následující kombinaci obou přístupů:

- Dynamický: V reprezentaci zdroje uvádíme URL zdroje a souvisejících zdrojů, které může klient použít pro volání operací čtení zdroje (GET). Pokud je URL uvedeno, operace HTTP GET je definována.
- Statický: Všechny ostatní možné operace nad daným zdrojem a nad souvisejícími zdroji definujeme jako součást OpenAPI definice veřejného API.

3.2.7.1 Dynamický HATEOAS pro operaci čtení

Do JSON datové struktury reprezentující zdroj jakéhokoliv typu vkládáme do místa, ve kterém referencujeme související zdroj typu objekt, jeho URL. URL je buď absolutní nebo relativní vůči URL veřejného API. Je uvedeno jako hodnota JSON klíče `_id`. Tento klíč pochází ze specifikace JSON-LD, kde je používán na identifikaci zdrojů reprezentujících konkrétní nebo abstraktní entity. V případě, že klient při zpracování JSON dokumentu s reprezentací zdroje narazí na klíč `_id`, interpretuje jej jako URL zdroje typu objekt. Pomocí HTTP GET požadavku na toto URL získá detailní reprezentaci odkazovaného objektu.

Pro zvýšení jednoznačnosti interpretace doplňujeme klíč `_id` klíčem `_type`. Ten také pochází ze specifikace JSON-LD. Jeho hodnotou je sémantický typ nebo pole sémantických typů identifikovaného objektu. Sémantické typy jsou katalogizovány v katalogu sémantických typů. Katalog sémantických typů uvádí pro každý sémantický typ jeho lidsky čitelný název a popis a propojuje jej na odpovídající prvek nebo prvky z byznys analýzy, typicky na odpovídající typ byznys entit.

Následující příklad demonstruje dynamický HATEOAS v reprezentaci vstupenky zdrojem `vstupenky/v-xzfhrd`. Klient získává detailní reprezentaci vstupenky, přičemž o měně ceny vstupenky a o jejím majiteli získává pouze základní informace. Detailní informace může získat voláním operace čtení (GET) na URL definovaných v klíči `_id`. Klíč `_type` může použít pro určení správné prezentační logiky pro uvedené sémantické typy.

```
{
    "_id": "tickets/v-xzfhrd",
    "_type": "ticket",
    "code": "v-xzfhrd",
    "price": {
        "value": 799,
        "currency": {
            "_id": "https://číselníky.cuni.cz/číselník/měny/položka/czk",
            "_type": "Měna",
            "notace": "CZK"
        }
```

```
    },
    "owner": {
        "_id": " studenti/12345678",
        "_type": ["student", "doctoral-student"],
        "personal-number": "12345678",
        "full-name": "Jan Evangelista Purkyně",
    }
}
```

Následující příklad demonstruje dynamický HATEOAS v reprezentaci studenta zdrojem `students/12345678`. Klient získává detailní reprezentaci studenta, přičemž o vstupenkách studenta získává pouze základní informace. Detailní informace o vstupence může získat voláním operace čtení (GET) na URL definovaných v klíči `_id`. Klíč `_type` může použít pro určení správné prezentační logiky pro uvedený sémantický typ.

```
{
    "_id": " studenti/12345678",
    "_type": ["student", "doctoral-student"],
    "personal-number": "12345678",
    "full-name": "Jan Evangelista Purkyně",
    "birth-date": "1787-12-17",
    "tickets": [
        {
            "_id": " tickets/v-xzfhrd",
            "_type": "ticket"
        },{
            "_id": " tickets/v-zuadre",
            "_type": "ticket"
        }
    ]
}
```

Následující příklad demonstruje dynamický HATEOAS v reprezentaci kolekce vstupenek zdrojem `vstupenky`. Klient získává základní informaci o jednotlivých vstupenkách. Detailní informace může získat voláním operace čtení (GET) na URL definovaných v klíči `_id`. Klíč `_type` může použít pro určení správné prezentační logiky pro uvedené sémantické typy.

```
[
    {
        "_id": "tickets/v-xzfhrd",
        "_type": "ticket",
        "kód": "v-xzfhrd"
    },{
        "_id": "tickets/v-zuadre",
        "_type": "ticket",
        "code": "v-xzfhrd"
    }
]
```

3.2.7.2 Statický HATEOAS pro CRUD operace

Výše uvedený přístup naplňuje princip HATEOAS pouze částečně, protože pokrývá pouze operace čtení zdrojů. Proto zavádíme ještě jeden přístup, kterým je statická specifikace volání dalších operací. V rámci OpenAPI definice veřejného API doplňujeme k definici každé operace nad zdrojem také statické definice toho, jaké další operace nad jakými dalšími souvisejícími zdroji můžeme volat poté, co obdržíme odpověď na její volání. Např.

- k definici operace čtení kolekce studentů doplníme statickou definici dalších možných operací

        - čtení detailů jednotlivých studentů
        - operace vložení nového studenta do kolekce
- k definici operace čtení studenta doplníme statickou definici dalších možných operací
    - úprava studenta
    - smazání studenta

Konkrétní sada možných operací je závislá na typu zdroje:

- pro zdroj typu objekt jsou definovány následující operace
    - pro operaci čtení objektu (GET)
        - aktualizace daného objektu (PUT), pokud je podporována
        - smazání daného objektu (DELETE), pokud je podporováno
        - pro každý typ byznys entit, kterého je objekt instancí a pro který byl určen zdroj typu kolekce objektů
            - čtení kolekce objektů, ve které jsou reprezentovány všechny instance tohoto typu byznys entit (GET)
        - pro každý konec byznys asociace, který má počátek v typu byznys entity, jehož je objekt instancí, a pro který byl určen zdroj typu kolekce vztahů
            - čtení kolekce vztahů, ve které jsou reprezentovány vztahy odpovídající byznys asociaci spojující objekt s jinými objekty (GET)
            - vytvoření nového vztahu v kolekci vztahů, pokud je podporováno (PUT)
        - pro každou specifickou byznys akci s objektem pro kterou byl určen zdroj typu ovladač
            - vykonání byznys akce (PUT/POST)
    - pro operaci aktualizace objektu (PUT/PATCH)
        - čtení objektu (GET)
- pro zdroj typu kolekce objektů reprezentujících množinu instancí daného typu byznys entit jsou definovány následující operace
    - pro operaci čtení kolekce (GET)
        - vytvoření nového objektu v kolekci (POST), pokud je podporováno
        - pro každý objekt v kolekci
            - čtení detailu objektu (GET)
    - pro operaci vytvoření nového objektu v kolekci (POST)
        - čtení objektu (GET)
- zdroj typu kolekce vztahů reprezentujících množinu vztahů odpovídajících byznys asociaci spojující objekt s jinými objekty
    - pro operaci čtení kolekce (GET)
        - vytvoření nového vztahu v kolekci vztahů, pokud je podporováno (PUT)
        - pro každý vztah v kolekci
            - smazání existujícího vztahu z kolekce vztahů, pokud je podporováno (DELETE)

Příklad specifikace veřejného API hypotetického modulu Zábava je uveden v příloze „Příklad specifikace veřejného API ve formátu OpenAPI 3.yaml“ tohoto dokumentu.

3.3 Obecné požadavky na privátní API

Privátní API je kolekcí operací, které naplňují specifické potřeby uživatelského front-end daného modulu. Privátní API je určeno pouze a jenom pro potřeby front-end daného modulu. Nesmí být

použito pro žádné jiné účely. Nesmí zajišťovat funkcionality, které jsou poskytovány veřejným API, tj. privátní API ani žádná jeho část nesmí striktně ani volně kopírovat funkcionality veřejného API nebo jeho části.

Protože není vystaveno veřejně do systému, nejsou na něj kladeny tak striktní požadavky jako na veřejné API.

Technicky se jedná o JSON API, které

- umožňuje čtení dat reprezentujícím byznys entity,
- umožňuje operace pro čtení i zapisování dalších dat, které nutně nevyplývají z byznys analýzy, ale jsou potřebné pro správné fungování uživatelského fronte-endu daného modulu,
- používá JSON pro reprezentaci dat,
- je bezstavové.

Privátní API musí být zdokumentováno a datové struktury pro reprezentaci dat v datovém formátu JSON musí být popsány v jazyku JSON Schema. Pokud nějaká datová struktura privátního API sémanticky odpovídá nějakému typu byznys entit popsaných v byznys analýze, musí být tato souvislost zaznamenána v dokumentaci.

Zadavatel si vyhrazuje právo z privátního API nebo jeho části vytvořit veřejné API. Zodpovědnost za privátní API nebo jeho část při využití tohoto práva převezme za API odpovědnost a zajistí, aby splňovalo výše uvedené podmínky na veřejné API. Dodavatel, který privátní API nebo část využíval, musí svůj kód potřebným způsobem upravit tak, aby pracoval s vytvořeným veřejným ekvivalentem.

4 Zajištění jakosti (QA) a dokumentace

4.1 Obecné požadavky na kvalitu kódu a bezpečnost

4.1.1 Kvalita kódu

Zdrojový kód každého modulu musí procházet kontrolou linteru a nástroje na statickou analýzu kódu s cílem odhalení chyb již v čase kompilace. Tento nástroj musí odpovídat zvolené technologii implementace daného modulu.

Zdrojový kód každého modulu musí procházet kontrolou nástroje na formátování odpovídající standardům a dobrým zvykům pro zvolenou technologii implementace.

4.1.2 Bezpečnost

Požadavky na implementaci modulů:

1. Modul musí být implementovaný tak, aby byl odolný vůči známým bezpečnostním hrozbám
2. Kód modulu prochází kontrolou **Static Vulnerability Scan**
    - Neobsahuje závažné/critical chyby
3. Kód prochází kontrolou **Dependency Vulnerability Check**
    - Neobsahuje závažné/critical chyby
4. Kód prochází kontrolou **Dynamic Vulnerability Analysis**
    - Neobsahuje závažné/critical chyby
5. Modul používá knihovny třetích stran, k nimž je poskytována LTS (long term support), nebo u nichž lze předpokládat střednědobá podpora (5-10 let) – například na základě popularity použití dané knihovny v rámci komunity

Pro prokázání splnění podmínek uvedených v bodech 2, 3, 4 dodavatel dodá pro každou novou verzi reporty z úspěšného ověření bezpečnosti pro jednotlivé položky. U false-positive případů uvedených v dodaných reportech bude poskytnuto písemné odůvodnění proč dodavatel považuje daný bod reportu za false-positive. Zadavatel bude namátkově a v některých případech automaticky pravidelně provádět tuto kontrolu také.

Následující tabulka uvádí přehled typických nástrojů pro provádění požadovaných skenů a reportů.

Druh skenu	**Typické nástroje**
Static Vulnerability Scan	IBM AppScan, Fortify, Veracode, FindSecBugs, Brakeman, Bandit
Dependency Vulnerability	OWASP Dependency Check, Bundler Audit, Safety
Dynamic Vulnerability Analysis	Burp Suite, OWASP ZAP, HP WebInSpect

4.2 Obecné požadavky na dokumentaci

Pro každý modul musí být dodavatelem odevzdána následující dokumentace:

1. Instalační a konfigurační příručka
    - Znalost systémových požadavků
    - Znalost způsobu konfigurace modulu
    - Znalost postupu instalace na lokální stanici a ve vývojovém prostředí

2. Dokumentace funkčních a nefunkčních požadavků
    - Znalost happy paths i unhappy paths
    - Napomáhá odhalení regresních závad na úrovni modulu

3. Dokumentace privátního (frontendového) i veřejného API standardizovaným strojově čitelným formátem
    - Verzované API v gitu
    - Znalost API zachycená pro vývojáře ostatních modulů
    - Možnost generovat klienty a mock servery pro dané API automaticky

4. Dokumentace umístění dat používaných modulem a možnosti jejich anonymizace (tam kde je to aplikovatelné)
    - Pro potřeby zálohování a správy dat využívaných modulem
    - Splnění požadavků dle GDPR před provedením kopií, záloh atd.

5. Dokumentace technické architektury modulu/aplikace
    - Interní architektura modulu
    - Komunikace s jinými moduly
    - Možnosti provozu modulu v HA
    - Popis autorizačních rozhodnutí prováděných modulem

6. Dokumentace generovaných auditních událostí

7. Dokumentace generovaných metrik
    - Popis všech metrik (jednotky, způsob měření, přesná sémantika atd.)

- Doporučení pro nastavení alertů pomocí PromQL

8. Dokumentace vlastních přidaných datových polí (fieldů) do strukturovaných logovacích záznamů

9. Popis všech použitých knihoven třetích stran a zdůvodnění jejich použití

10. Technická dokumentace netriviálních algoritmů

11. Popis SQL skriptů pro vytvoření potřebné databázové struktury a iniciální naplnění dat/číselníků

12. Popis datových migračních skriptů

13. Dokumentace testovací strategie, testovacích scénářů, výkonnostních testů a vytvořených testovacích, simulačních a mockovacích nástrojů

14. Doporučená systémová (hardwarová) konfigurace pro nasazení do testovacího (Stage) i produkčního (Prod) prostředí, odpovídající požadovaným výkonnostním parametrům

15. Uživatelská příručka pro administraci pomocí administračního UI rozhraní (jestli nějaké existuje)

16. Uživatelské příručka pro běžné uživatele

Očekávané technologie a standardy:

- Gherkin – pro popis testovacích scénářů
- OpenAPI 3.0 / GraphQL Schema – dokumentace API
- Markdown – formátovaný text
- UML diagramy
- README.md – informace pro vývojáře a operations tým, co daný modul dělá, jak ho rychle spustit ve vývojovém prostředí a jak pro něj vyvíjet

4.3 Obecné požadavky na QA

4.3.1 Obecné požadavky na testování

Pro potřeby ověření kvality dodaného software (Quality Assurance - QA) je kromě dodání samotné aplikace očekáváno také dodání několika druhů automatických, poloautomatických a manuálních testů, a to v následujících kategoriích:

- **Unit a integrační testy**
  - Cíl: zvýšená šance odhalení regresních závad na úrovni modulu, a to zejména při úpravách a refactoringu
  - Zejména netriviální procesy a algoritmy by měly být pokryté pomocí automatizovaných (unit) testů
  - Bezchybná komunikace mezi jednotlivými komponentami uvnitř aplikace, ale také mezi rozhraním různých systému by měly být pokryté pomocí automatizovaných (integračních) testů

    - Technická implementace testů bude závislá na zvolené technologii pro implementaci jednotlivých modulů
    - Dodavatel musí poskytnout zadavateli veškeré potřebné nástroje a dokumentaci pro automatické spouštění testů jako součásti CI/CD pipeline (build modulu)
    - Procento pokrytí testy a míra automatizace bude specifikována pro každý modul v zadávací dokumentaci modulu

- **Systémové „end-to-end“ testy**
    - Cíl: zvýšená šance odhalení regresních závad na úrovni vzájemné integrace modulů při úpravách a refactoringu
    - Cíl: dokumentace technického postupu jednotlivých uživatelských cest (user journeys) pro vzájemnou validaci korespondence s byznys analýzou modulu
    - Všechny uživatelské cesty (user journeys) a automatizované procesy by měly být pokryté pomocí end-to-end testů (testovací scénáře)
    - Testovací scénáře mohou být automatizované, poloautomatizované nebo zcela manuální, s maximální možnou (rozumnou) mírou plné automatizace co největšího počtu testovacích scénářů, s cílem zapojení testů do pravidelného (nočního) spouštění všech plně automatizovaných scénářů v testovacím prostředí zadavatele
    - Testovací scénáře budou vytvářeny ve formátu Gherkin
    - Seznam minimální požadované sady dodaných testovacích scénářů bude specifikován pro každý modul v zadávací dokumentaci modulu

- **Výkonnostní testy**
    - Cíl: zvýšená šance odhalení výkonnostních nedostatků při úpravách a refactoringu
    - Pokrytí všech výkonnostně citlivých částí systému výkonnostními testy
    - Výkonnostní testy musí být automatizované v maximální možné míře
    - Společně s testy poskytne dodavatel zadavateli i detailní technickou dokumentaci popisující fungování, přípravu dat a způsob jejich spouštění
    - Výkonnostní požadavky pro jednotlivé funkce systému, případně míra automatizace testů budou specifikovány v zadávací dokumentaci modulu

4.3.2 Mock, simulační nástroje a nástroje na generování dat

Testy ze všech kategorií, zejména pak testy funkčních požadavků je potřebné umět spouštět jak v rámci plně nasazeného prostředí, tak i proti lokální instanci daného modulu (pro možnost otestování před nasazením do prostředí). Pro tuto potřebu poskytne dodavatel zadavateli také odpovídající nástroje:

- Mock nástroje (nástroje simulující funkci třetích komponent pomocí definovaného rozhraní komponenty)
- Nástroje a generátory pro automatizované vytváření testovacích datových sad
- Simulační nástroje (generátory klientského provozu pomoci API rozhraní modulu, nástroje simulující uživatelský provoz atd.)

Požadovaný seznam nástrojů relevantní k funkcím daného modulu bude upřesněn v zadávací dokumentaci.

4.3.3 Doporučená systémová konfigurace

Jako součást dokumentace poskytne dodavatel zadavateli také popis doporučené systémové (hardwarové) konfigurace pro nasazení do jak do testovacího (Stage) tak i do produkčního (Prod) prostředí, odpovídající požadovaným výkonnostním parametrům systému/modulu.