DELTA – Střední škola informatiky a ekonomie, s.r.o.
Ke Kamenci 151, PARDUBICE

MATURITNÍ PROJEKT

Integrace služeb třetích stran (ThirdParty-Integrations)

Příjmení, jméno: Pořický David

Třída: 4.B

Studijní obor: Informační technologie 18-20-M/01

Školní rok: 2024/2025

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení:	David Pořický
Pro školní rok:	2024/2025
Třída:	4. B
Obor:	Informační technologie 18-20-M/01
Téma práce:	Integrace služeb třetích stran (ThirdPartyIntegrations)
Vedoucí práce:	*RNDr. Jan Koupil, Ph.D.*

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem tohoto projektu je vytvořit modul pro integraci služeb třetích stran do stávající webové aplikace EOS. První implementovanou integrací bude služba **make.com**, která bude potřebovat přístup k naší API a možnost načítat data pomocí specifického **accessTokenu**. Tento token bude fungovat pouze na dedikovaných API endpointech vytvořených pro danou integraci. Modul zároveň poskytne podporu pro **first-party** služby, jako je náš frontend nebo hypoteticky servisní aplikace.

Specifikace projektu:

1. **Analýza a návrh:**
    - Budou definovány požadavky na API pro komunikaci s externími službami (konkrétně make.com).
    - Bude vytvořen návrh struktury modulu **ThirdPartyIntegrations** a navrženy dedikované API endpointy pro třetí strany i **first-party** služby.
2. **Integrace služeb třetích stran:**
    - Implementace modulu **ThirdPartyIntegrations**.
    - Služba make.com bude moci přistupovat k specifickým endpointům, které budou navrženy pro danou integraci.
    - API bude obsahovat mechanizmy pro bezpečnou autorizaci a autentizaci přístupů ze strany třetích stran.

3. **First-party API:**
    - Implementace dedikovaných API endpointů pro **first-party** služby (např. náš frontend či servisní aplikace).
    - Podpora více typů přístupových práv a tokenů pro různé služby, včetně **first-party** integrací.
4. **AccessToken a zabezpečení:**
    - Bude vytvořen bezpečnostní mechanismus pro generování a správu **accessTokenů**.
    - **AccessToken** bude omezen na dedikované endpointy, nicméně pro službu **make.com** nebude mít omezenou platnost.
    - Implementace zabezpečení API komunikace proti zneužití a neoprávněným přístupům.
5. **API endpointy pro integraci:**
    - Implementace dedikovaných endpointů, které budou komunikovat se službami třetích stran i **first-party** aplikacemi.
    - Podpora více typů přístupových práv pro různé typy integrací.
6. **Technické požadavky:**
    - Implementace API bude dodržovat standardy REST API.
    - Bude zajištěna vysoká úroveň bezpečnosti dat při komunikaci s externími i **first-party** službami.
    - Struktura kódu bude navržena tak, aby bylo možné snadno přidávat nové metody přístupu a rozšiřovat stávající funkcionalitu.
7. **Prezentace a obhajoba:**
    - Bude připravena prezentace projektu včetně technických detailů a praktického využití modulu **ThirdPartyIntegrations**.
    - Kód nebude veřejně přístupný, ale bude dostupný pro potřeby obhajoby u maturity.

Požadované výstupy:

- Funkční modul **ThirdPartyIntegrations** pro integraci služeb třetích stran i **first-party** služeb.
- Podpora načítání dat z naší API pomocí **accessTokenu**.
- API endpointy dedikované pro třetí strany s bezpečnostními mechanismy.
- Dokumentace pro uživatele.
- Prezentace projektu.

Hodnocení:

- Kvalita a funkčnost integrace se službou **make.com**.
- Bezpečnost a správná správa **accessTokenů** a API endpointů.
- Kvalita dokumentace a prezentace projektu.

Stručný časový harmonogram:

- **Září: Seznámení s platformou make.com, připomenutí fungování systému.**
- **Říjen–listopad: Vytváření Modulů, databáze, PEST testů, first-party endpointů.**
- **Prosinec: Implementace JWT tokenu.**
- **Leden-Únor: Implementace third-party endpointu a napojení na JWT token.**
- **Březen: Práce na dokumentaci.**

Prohlášení

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

V Pardubicích dne _____ _____

David Pořický

Poděkování

Děkuji vedoucímu mého projektu, RNDr. Janu Koupilovi, Ph.D., za odborné vedení, cenné rady, a trpělivost během celé práce. Velké poděkování patří také hlavnímu backend developerovi Patrikovi Václavkovi za zaučení do systému a vedení mé práce, celému týmu EOS za přátelský kolektiv a rady, a hlavně CEO EOS media Ondřeji Syrovému za možnost podílet se na vývoji jejich systému.

Anotace

Tento projekt se zaměřuje na vývoj modulu **ThirdPartyIntegrations**, který umožní integraci služeb třetích stran do stávající webové aplikace **EOS**. Hlavním cílem je vytvoření bezpečného a flexibilního API, které bude podporovat komunikaci s externími i first-party službami. První implementací je integrace platformy **make.com**, která má přístup ke specifickým dedikovaným API endpointům prostřednictvím accessTokenu. Modul dále poskytuje dedikované endpointy pro **first-party** služby, jako je náš frontend či servisní aplikace. Projekt klade důraz na vysokou úroveň zabezpečení, snadnou rozšiřitelnost a dodržování standardů REST API. Výstupem je plně implementovaný modul připravený na integraci dalších služeb.

Klíčová slova

Laravel, PHP, Backend, JWT, first-party, third-party, EOS, Make.com, Endpoint

Obsah

1. Úvod 9
2. EOS media s.r.o. 9
3. MAKE.COM 9
4. Technologie 11
   4.1 Laravel 11
   4.2 MariaDB 13
5. Endpoints 13
   5.1 First-party 13
   5.2 Third-party 15
6. Zabezpečení API 15
   6.1 Zabezpečení first-party endpointů 15
   6.2 JSON Web Token (JWT) 16
   6.3 Šifrování tokenů a práce s nimi 17
7. Databáze 18
   7.1 Schéma databáze 18
8. Ukázka 19
9. Závěr 21

Seznam obrázků 22

Reference 22

Úvod

V minulém roce jsem měl možnost absolvovat odbornou praxi ve firmě EOS, kde jsem se začal seznamovat s jejich systémem a způsobem práce. Do té doby jsem neměl žádné praktické zkušenosti s frameworkem Laravel, ale během praxe jsem získal základní znalosti a dovednosti potřebné pro práci s tímto systémem.

Po skončení praxe jsme se domluvili, že bych mohl na projektu pokračovat a současně jej použít jako maturitní práci. Během práce na projektu mi vedoucí zadával úkoly rozčleněné do jednotlivých bodů, které jsem postupně plnil. Každý úkol měl svůj konkrétní cíl a zahrnoval různé aspekty vývoje, od back-endové logiky po implementaci nových funkcionalit. Pokud jsem narazil na něco, co jsem neznal, vedoucí mi často pomáhal a poskytoval mi konzultace, díky čemu jsem se rychleji učil a zdokonaloval se ve vývoji aplikací v Laravelu. Samotný projekt je zaměřen na poskytování možnosti majitelům sportovních klubů spravovat a pracovat s daty svých členů na platformě Make.com. Tato platforma slouží k automatizaci procesů a usnadnění administrativní práce, což je pro kluby velmi důležité. Tento projekt nejen velmi rozšířil moje technické znalosti, ale také mi poskytl užitečnou zkušenost s reálným vývojem softwaru pro konkrétní potřeby klientů.

1. EOS media s.r.o.

EOS media s.r.o. je digitální kreativní studio se sídlem v Ústí nad Orlicí, které se specializuje na vývoj webových a mobilních aplikací, správu digitálního obsahu a marketingové služby. Firma byla založena v roce 2016, ale její historie sahá až do sezóny 2011/2012, kdy začala s vývojem platformy EOS pro digitalizaci administrativy a komunikace sportovních klubů.

Hlavním produktem EOS media je eos club zone, komplexní platforma pro správu sportovních oddílů a organizací. Tato aplikace umožňuje efektivní vedení členské základny, komunikaci mezi trenéry, hráči a rodiči, správu plateb a plánování sportovních akcí. Díky své modularitě a široké jazykové podpoře je eos club zone využívána téměř 600 kluby ve třech evropských zemích

Vedle tohoto systému se EOS media věnuje také tvorbě webových stráne, vývoji e-shopů, grafickému designu a copywritingu. [1]

2. MAKE.COM

MAKE.com je vizuální nástroj pro automatizaci procesů a integraci aplikací, který umožňuje propojení různých služeb bez nutnosti složitého programování. Funguje jako prostředník mezi aplikacemi, které běžně používáme, a umožňuje automatizovat opakující se úkoly efektivně a intuitivně.

Jak MAKE.com funguje?

Platforma pracuje na principu scénářů, které se skládají z jednotlivých modulů propojujících aplikace a služby. Každý scénář představuje automatizovaný pracovní postup, který může být spuštěn manuálně, v pravidelných intervalech, nebo při určité události. Tyto scénáře mohou být

jednoduché (např. automatické odeslání e-mailu po vyplnění formuláře) nebo složité (např. synchronizace dat mezi různými databázemi, zpracování objednávek a správa zákaznických informací).

Každý modul v MAKE.com vykonává konkrétní akci, například:

- **Získání dat** – Načtení informací z API, databáze nebo aplikace (např. přijetí nové objednávky z e-shopu, načtení nového kontaktu z Google Sheets).
- **Zpracování dat** – Úprava, filtrování, transformace nebo výpočty s daty (např. změna formátu času, převod měny, výpočet cen).
- **Odeslání dat** – Přenos informací do jiné aplikace nebo systému (např. zapsání nového zákazníka do CRM, automatické zaslání faktury, přidání úkolu do Trella).

Všechny tyto operace probíhají ve vizuálním rozhraní, kde si uživatelé mohou jednotlivé moduly jednoduše propojit pomocí drag-and-drop editoru. Na obrázku 1 je jednoduchý příklad automatizace Gmailu:



Obrázek 1 Ukázka modulů na platformě make.com

Gmail lze v Make.com integrovat jako automatizovaný spouštěč pro různé akce. Připojení se provádí přímo přes oficiální Gmail API, kde je nutná autorizace uživatele. Po autorizaci lze nastavit spouštěč sledující nové e-maily, který následně zpracovává přijatá data.

V rámci scénáře lze tato data předávat dalším službám, například API ChatGPT, a výslednou odpověď automaticky odeslat zpět odesílateli e-mailu. Celý proces probíhá bez nutnosti manuálního zásahu, a umožňuje tak automatizaci odpovědí.

Tento konkrétní modul byl vytvořen přímo společností make, za pomoci oficiální API Gmailu, stejně jako pár dalších modulu, jako Trello, Dropbox nebo Slack. Naprostá většina modulu je ale vytvářena konkrétními společnostmi stojícími za daným produktem [2]

3. Technologie

3.1 Laravel

Laravel je moderní PHP framework postavený na principu MVC (Model-View-Controller) architektury, která rozděluje aplikaci do tří hlavních vrstev: Model (datová vrstva), View (prezentační vrstva) a Controller (řídicí vrstva). Tento přístup zajišťuje lepší organizaci kódu, snadnější údržbu a rozšiřitelnost aplikace. Model se stará o práci s daty a logiku spojenou s databází, View zobrazuje uživatelské rozhraní a Controller zpracovává požadavky uživatele a propojuje Model s View. V tomto projektu používáme Laravel pouze pro backend, takže view není potřeba.

Routování v Laravelu

Jednou z klíčových vlastností Laravelu je systém routování, který mapuje HTTP požadavky na konkrétní metody v kontrolerech. Routy jsou definovány v souboru basic.php a umožňují snadno nastavit, jaké akce se mají provést při návštěvě dané URL. Například při přístupu na /third-party-integration/1 může být zavolána metoda index v ThirdPartyIntegragrationApiController, která následně vrátí příslušný záznam. Routování je flexibilní a podporuje různé typy požadavků (GET, POST, PUT, DELETE), což je ideální pro tvorbu RESTful API. [3]

Pro práci s databází Laravel nabízí nástroj Eloquent ORM, který umožňuje pracovat s databázovými tabulkami jako s objekty. Každá tabulka je reprezentována modelem, který zjednodušuje operace jako vytváření, čtení, aktualizace a mazání záznamů (CRUD). Místo psaní složitých SQL dotazů lze použít jednoduché metody, například ThirdPartyIntegration::find(1) vrátí požadovaný záznam. Laravel také podporuje migrace, které slouží k verzování a správě databázových struktur. Migrace umožňují vytvářet, upravovat a mazat tabulky pomocí kódu, což zajišťuje konzistenci databáze napříč různými prostředími. [4]

Další důležitou součástí Laravelu je Service Container, což je nástroj pro správu závislostí a provádění dependency injection. Service Container automaticky řeší závislosti mezi třídami, což usnadňuje testování a modularitu aplikace. Například pokud kontroler potřebuje službu pro odesílání podepisování JWT tokenu, Laravel tuto službu automaticky předá do konstruktoru kontroleru. Tento mechanismus také umožňuje snadné nahrazování tříd jejich implementacemi, což je užitečné při testování nebo rozšiřování funkcionality. [5]

3.1.1 PEST testy

Pest je moderní testovací framework pro PHP, který je postavený na PHPUnitu, ale přináší řadu vylepšení, která výrazně zjednodušují psaní testů. Pest je navržen tak, aby vývojáři mohli psát testy rychleji a efektivněji, přičemž zachovává plnou kompatibilitu s PHPUnitem.

Testy v Pestu se soustředí na psaní jednoduchých testů pro kontrolu správnosti funkcí a metod v aplikaci. Při testování API endpointů Pest umožňuje provádět HTTP požadavky, jako jsou GET, POST, PUT nebo DELETE, což usnadňuje interakci s API. Je možné ověřit nejen stavové kódy odpovědi, ale také obsah odpovědi. Tento způsob testování je obzvlášť užitečný při testování složitějších scénářů, kde je potřeba ověřit nejen správnost dat, ale i to, jak API reaguje na různé kombinace vstupních dat.

Při testování API endpointů často využívají model factories pro generování testovacích dat, což umožňuje rychle vytvořit realistické objekty modelů bez nutnosti ručního zadávání hodnot pro každý test. Model factories umožňují vývojářům snadno vytvořit potřebné množství testovacích dat, například pro testování API endpointu, který vrací seznam uživatelů. V tomto případě může vývojář využít factory k vygenerování deseti uživatelů, které následně použije pro ověření, že endpoint vrací správná data a odpovídá očekávané struktuře.

Použití Pest testů v Laravelu přináší i další výhodu v podobě snadné detekce potenciálních bezpečnostních problémů. Testování API může odhalit zranitelnosti, jako jsou problémy s autentizací nebo autorizací.

Zde je test pro endpoint */third-party-integrations/{thirdPartyIntegrationId}/access*

```
20  test(
21      description: 'GET /third-party-integrations/{thirdPartyIntegrationId}/access 200 OK -> index',
22      function (TestApiDataset $dataset): void {
23          $thirdPartyIntegrationAccess = modelFactory( modelClass: ThirdPartyIntegrationAccessModelFactory::class)->create();
24          $thirdPartyIntegrationId = $thirdPartyIntegrationAccess::guardType($dataset->getData()->first())->getThirdPartyIntegrationId();
25          $thirdPartyIntegrationAccessId = $thirdPartyIntegrationAccess::guardType($dataset->getData()->first())->getId();
26
27          expect($this->apiGet("/third-party-integrations/{$thirdPartyIntegrationId}/access/{$thirdPartyIntegrationAccessId}"))
28              ->toBeApiItem(
29                  structure: $dataset->getStructure(),
30                  presenter: ThirdPartyIntegrationAccessApiTransformer::class,
31                  item: $thirdPartyIntegrationAccess,
32                  status: 200,
33              );
34      }
35  )->with([
36      apiDataset(
37          auth: authorizeAsClub()->standardClub()->addAllTeamsRole(
38              read: [ModuleRegistry::settings()],
39          ),
40          dataset: static fn (): Collection => modelFactory( modelClass: ThirdPartyIntegrationAccessModelFactory::class)
41              ->createMany( count: 2, [
42                  ThirdPartyIntegrationAccess::ATTR_THIRD_PARTY_INTEGRATION_ID =>
43                      \modelFactory( modelClass: ThirdPartyIntegrationModelFactory::class)->create()->getId(),
44              ]),
45          structure: ThirdPartyIntegrationAccessApiStructure::full(),
46      ),
47  ]);
```

Obrázek 2 Ukázka kódu pro PEST testy

23-25. Vytvoření instance thirdPartyIntegrationAccess pomocí modelFactory a získání thirdPartyIntegrationId a thirdPartyIntegrationAccessId z tohoto modelu pro použití v API požadavku.

27-33. Zde se odesílá **GET požadavek** na API endpoint */third-party-integrations/{thirdParty-IntegrationId}/access/{thirdPartyIntegrationAccessId}*, kde jsou dynamicky vloženy hodnoty thirdPartyIntegrationId a thirdPartyIntegrationAccessId.

- **Ověření**:
  - **struktuře** obsahuje strukturu očekávané odpovědi, která je získána z $dataset->getStructure().
  - **presenter** definuje transformer, který by měl být použit k transformaci dat.
  - **item** je model, který se očekává, že bude vrácen v odpovědi API.
  - **status** určuje, že odpovědí očekáváme statusový kód **200 OK**.

35-47. Tato část používá funkci with(), která je specifická pro Pest a Laravel, a slouží k předání **testovacích dat** do testu.

- **auth**: Zde je použita metoda authorizeAsClub() pro definici uživatele, který bude autorizován k přístupu. Tato metoda ověřuje, že uživatel má potřebná oprávnění pro testovaný endpoint.
- **dataset**: Tento callback vytváří dvě instance modelu ThirdPartyIntegrationAccess s náhodnými daty (pomocí modelFactory).
- **structure**: Určuje, jaká struktura dat se očekává v odpovědi API. [6] [7]

3.2 MariaDB

MariaDB je open-source relační databázový systém, který je široce používán pro správu dat v různých aplikacích, včetně webových aplikací a e-commerce platforem. Je kompatibilní s MySQL, což usnadňuje přechod z MySQL bez nutnosti změn v aplikacích. MariaDB je navržen pro rychlé zpracování dotazů a může zvládat vysoké zatížení, což je důležité pro aplikace, které vyžadují rychlé zpracování dat.

MariaDB funguje jako relační databáze, což znamená, že data jsou organizována do tabulek, sloupců a řádků. Každý řádek v tabulce může obsahovat primární klíč, který umožňuje vytvářet vztahy mezi daty. [8]

4. Endpoints

4.1 First-party

First-party endpointy jsou část API navržená pro přímou komunikaci mezi interními aplikacemi a serverem. V kontextu maturitního projektu slouží k propojení frontendu a případných servisních aplikací s backendem systému EOS. Tyto endpointy jsou optimalizovány pro interní potřeby organizace a poskytují bezpečný přístup k datům a funkcím, které jsou klíčové pro provoz aplikace.

First-party endpointy fungují jako vstupní body, kde aplikace zasílají požadavky na server pomocí standardních HTTP metod, jako jsou **GET**, **POST**, **PUT**, **DELETE** nebo **PATCH**. Každý endpoint je navržen tak, aby reprezentoval konkrétní zdroj nebo operaci v systému,

například načítání dat, jejich úpravu nebo mazání. URL struktura těchto endpointů často obsahuje dynamické parametry, které umožňují přístup k specifickým datům, například /api/v2/third-party-integrations/{integrationId}.

```
twikin
public function show(
    ThirdPartyIntegrationAccessRepositoryInterface $thirdPartyIntegrationAccessRepository,
    ThirdPartyIntegrationAccessPolicy $thirdPartyIntegrationAccessPolicy,
    int|string $thirdPartyIntegrationId,
    int|string $thirdPartyIntegrationAccessId
): ApiResponse {
    $thirdPartyIntegrationAccess = $thirdPartyIntegrationAccessRepository
        ->query()
        ->whereThirdPartyIntegrationId((int) $thirdPartyIntegrationId)
        ->getById((int) $thirdPartyIntegrationAccessId);
    $thirdPartyIntegrationAccessPolicy->authorizeRead($thirdPartyIntegrationAccess);

    return $this->itemResponse(
        $thirdPartyIntegrationAccess,
        transformerClass: ThirdPartyIntegrationAccessApiTransformer::class
    );
}
```

Obrázek 3 Ukázka kódu controlleru

1. **Injektování závislostí**:
   Na začátku metody jsou injektovány potřebné závislosti, jako je ThirdPartyIntegrationAccessRepositoryInterface a ThirdPartyIntegrationAccessPolicy.

2. **Parametry metody**:
   Metoda přijímá dva důležité parametry:

   - $thirdPartyIntegrationId
   - $thirdPartyIntegrationAccessId

   Tyto parametry jsou dynamicky získávány z URL požadavku.

3. **Dotaz do databáze**:
   Pomocí repository ($this->thirdPartyIntegrationAccessRepository) je vytvořen dotaz, který vyhledává konkrétní záznam na základě poskytnutých identifikátorů ($thirdPartyIntegrationId a $thirdPartyIntegrationAccessId).

4. **Autorizace přístupu**:
   Po načtení záznamu je volána metoda authorizeRead(), která ověřuje, zda má uživatel oprávnění k přístupu k tomuto zdroji.

5. **Transformace dat**:
   Na konci metody je použit transformátor ($this->thirdPartyIntegrationAccessApiTransformer), který převede získaná data do požadovaného formátu pro odpověď API.

6. **Návratová hodnota**:
   Metoda vrací odpověď ve formátu JSON. Tato odpověď je připravena tak, aby byla snadno čitelná pro klientskou aplikaci.

Seznam endpointů:

GET	/api/v2/third-party-integrations
GET	/api/v2/third-party-integrations/{thirdPartyIntegrationId}
POST	/api/v2/third-party-integrations
PUT	/api/v2/third-party-integrations/{thirdPartyIntegrationId}
GET	/api/v2/third-party-integrations/{thirdPartyIntegrationId}/access
GET	/api/v2/third-party-integrations/{thirdPartyIntegrationId}/access/{thirdPartyIntegrationAccessId}
POST	/api/v2/third-party-integrations/{thirdPartyIntegrationId}/access
PUT	/api/v2/third-party-integrations/{thirdPartyIntegrationId}/access/{thirdPartyIntegrationAccessId}
DELETE	/api/v2/third-party-integrations/{thirdPartyIntegrationId}/access/{thirdPartyIntegrationAccessId}
PATCH	/api/v2/trash/third-party-integrations/{thirdPartyIntegrationAccessId}

4.2 Third-party

Third-party endpointy jsou dedikované API body navržené pro komunikaci s externími službami. V rámci maturitního projektu slouží k integraci platformy make.com, která prostřednictvím těchto endpointů načítá data ze systému EOS.

Third-party endpointy umožňují externím službám přístup k datům prostřednictvím HTTP požadavků. Pro zajištění bezpečnosti je každý požadavek doprovázen podepsaným JWT tokenem, který je předáván jako parametr v požadavku a ověřován na straně serveru.

Platforma make.com využívá tyto endpointy k načítání dat o členech, potřebných pro automatizaci procesů.

5. Zabezpečení API

5.1 Zabezpečení first-party endpointů

V rámci maturitního projektu bylo zabezpečení first-party endpointů implementováno pomocí speciálního autorizačního mechanismu. Tento mechanismus využívá autorizační endpoint, který generuje ověřovací token s platností 30 minut. Tento token je následně přidáván do autorizační hlavičky každého požadavku na first-party endpointy a slouží k ověření oprávněnosti přístupu. V praxi bude ale ověřování fungovat na principu autorizačních práv uživatele.

5.2 JSON Web Token (JWT)

JSON Web Token (JWT) je otevřený standard, který umožňuje bezpečný přenos informací mezi dvěma stranami ve formátu JSON. Tyto tokeny jsou kompaktní, samostatné a digitálně podepsané, což zaručuje jejich důvěryhodnost a integritu. JWT se používá především pro autentizaci a autorizaci v moderních webových aplikacích, ale mohou být využity i pro bezpečnou výměnu dat.

JWT se skládá ze tří částí oddělených tečkou:

1. Header (Hlavička)

Hlavička obsahuje informace o typu tokenu a algoritmu použitém k jeho podpisu. Typicky vypadá takto:

{

"alg": "HS256",

"typ": "JWT"

}

Hlavička je zakódována pomocí Base64Url.

2. Payload (Obsah)

Payload obsahuje tvrzení (*claims*), což jsou informace o uživateli nebo entitě, která token používá. Tvrzení mohou být:

- **Registrované**: standardní deklarace registrované u Internet Assigned Numbers Authority (IANA) a definované specifikací JWT pro zajištění interoperability s aplikacemi třetích stran nebo externími aplikacemi. Standardní deklarace OIDC jsou vyhrazené deklarace. To mohou být například:
- iss (issuer): Issuer of the JWT
- sub (subject): Subject of the JWT (the user)
- aud (audience): Recipient for which the JWT is intended
- exp (expiration time): Time after which the JWT expires

Vlastní: Definované uživatelem, například jméno nebo role uživatele

Příklad payloadu:

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

Payload je také zakódován pomocí Base64Url.

3. Signature (Podpis)

Podpis zajišťuje integritu tokenu a ověřuje jeho původ. Vytváří se kombinací zakódované hlavičky a payloadu, které jsou podepsány pomocí tajného klíče. [9]

5.3 Šifrování tokenů a práce s nimi

V rámci projektu je kladen důraz na bezpečnou práci s autentizačními tokeny. Tento systém využívá šifrování klíčů a podpisování JWT tokenů k ochraně endpointu. Celý proces správy tokenů je navržen tak, aby zajistil bezpečnost a minimalizoval riziko neoprávněného přístupu k citlivým datům.

Generování a uložení šifrovacího klíče

Při prvním nasazení systému je manuálně spuštěním scriptu vygenerován šifrovací klíč, který slouží jako hlavní prvek pro šifrování citlivých informací. Tento klíč má délku 32 bytů a je zakódován do formátu Base64, aby byl jednoduše čitelný a snadno manipulovatelný v konfiguračních souborech. Ukládá se do souboru .env, který obsahuje veškeré citlivé konfigurace aplikace a není součástí verzovacího systému, čímž je minimalizováno riziko úniku těchto informací.

Ukládání a šifrování Signing Secret

Při vytváření nové integrace třetí strany je generován tzv. Signing Secret. Tento klíč slouží k podepisování a ověřování integrity tokenů používaných v rámci konkrétní integrace. Signing Secret je opět tvořen 32bytovým náhodným řetězcem, který je před uložením do databáze zašifrován pomocí šifrovacího klíče uloženého v .env. Tento přístup zajišťuje, že samotný Signing Secret není nikdy uložen v databázi v nešifrované podobě a není možné jej přímo přečíst bez přístupu k šifrovacímu klíči.

Generování a podepisování JWT tokenů

Autorizace přístupu k endpointu je realizována pomocí JWT tokenů. Při vytvoření přístupu třetí strany (ThirdPartyIntegrationAccess) je automaticky vygenerován JWT token, který je zatím ponechán v nepodepsané podobě a uložen do samostatné tabulky v databázi. V hlavní tabulce third_party_integration_access je uložen pouze cizí klíč na tento token, což zajišťuje vyšší úroveň bezpečnosti a minimalizuje možnost úniku citlivých informací.

V okamžiku, kdy oprávněný uživatel (administrátor nebo tvůrce přístupu) požádá o přístupový token prostřednictvím rozhraní embedu, probíhá několik bezpečnostních kroků. Nejprve se z databáze načte zašifrovaný Signing Secret odpovídající dané integraci. Tento klíč je následně dešifrován pomocí hlavního šifrovacího klíče ze souboru .env. Jakmile je Signing Secret v čitelné podobě, použije se k podepsání JWT tokenu. Tento proces zajišťuje, že podepsaný token obsahuje platný kryptografický podpis, který ověřuje jeho pravost a integritu.

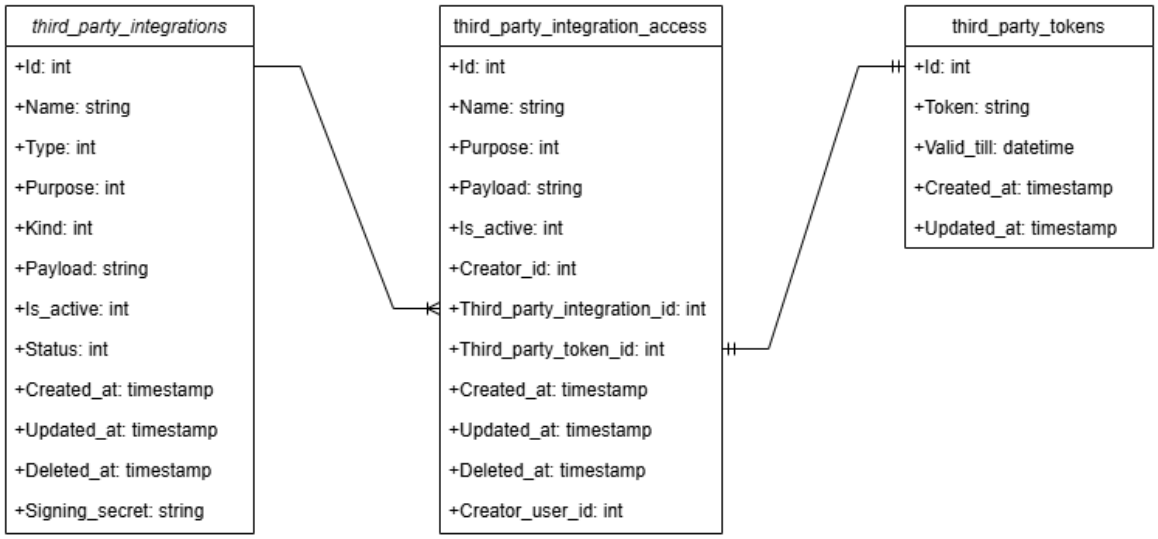
Zajištění bezpečnosti a ochrana přístupových údajů

Díky tomuto přístupu jsou veškeré citlivé údaje bezpečně spravovány a ukládány tak, aby nebylo možné je zneužít ani v případě kompromitace databáze. Jelikož je Signing Secret vždy uložen v šifrované podobě a přístupový token není nikdy uložen přímo v databázi, je minimalizováno riziko neoprávněného přístupu k těmto informacím. Tento model správy autentizačních údajů umožňuje bezpečnou integraci externích aplikací a služeb, aniž by bylo nutné vystavovat přihlašovací údaje nebo přístupové tokeny přímo.

Navržený systém poskytuje vysokou úroveň ochrany, a zároveň umožňuje flexibilní správu přístupů k jednotlivým integracím. Díky tomu je možné bezpečně spravovat přístupy třetích stran bez obav z úniku citlivých dat nebo zneužití přístupových klíčů.

6. Databáze

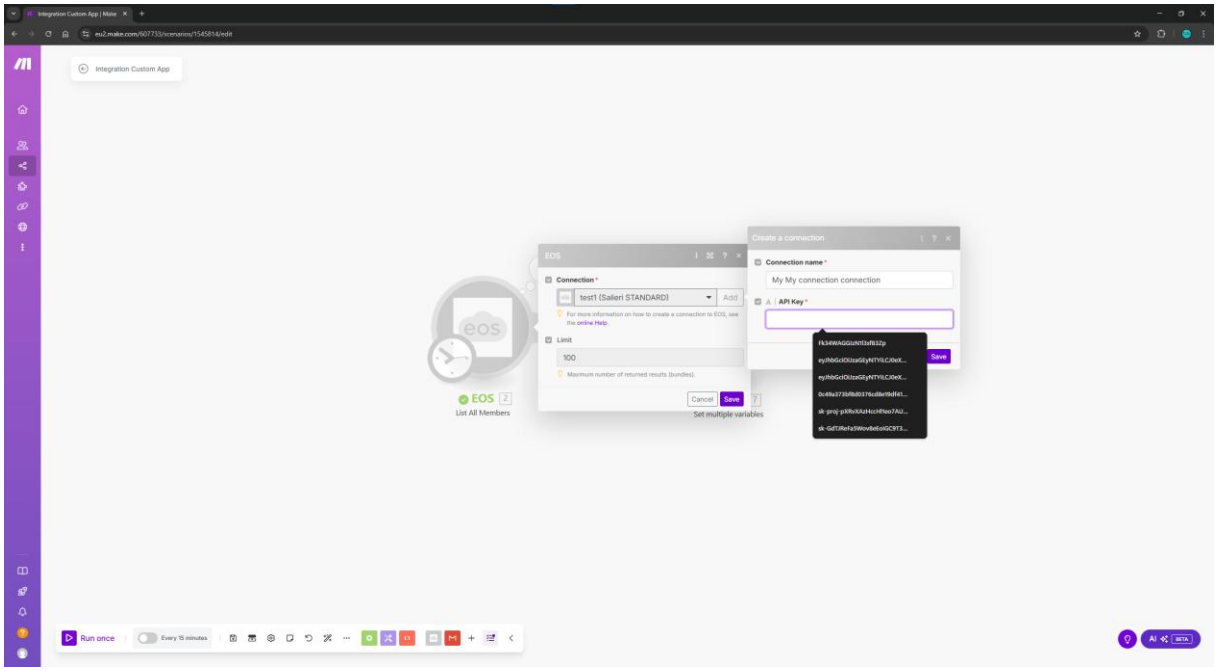
6.1 Schéma databáze



Obrázek 4 Class diagram

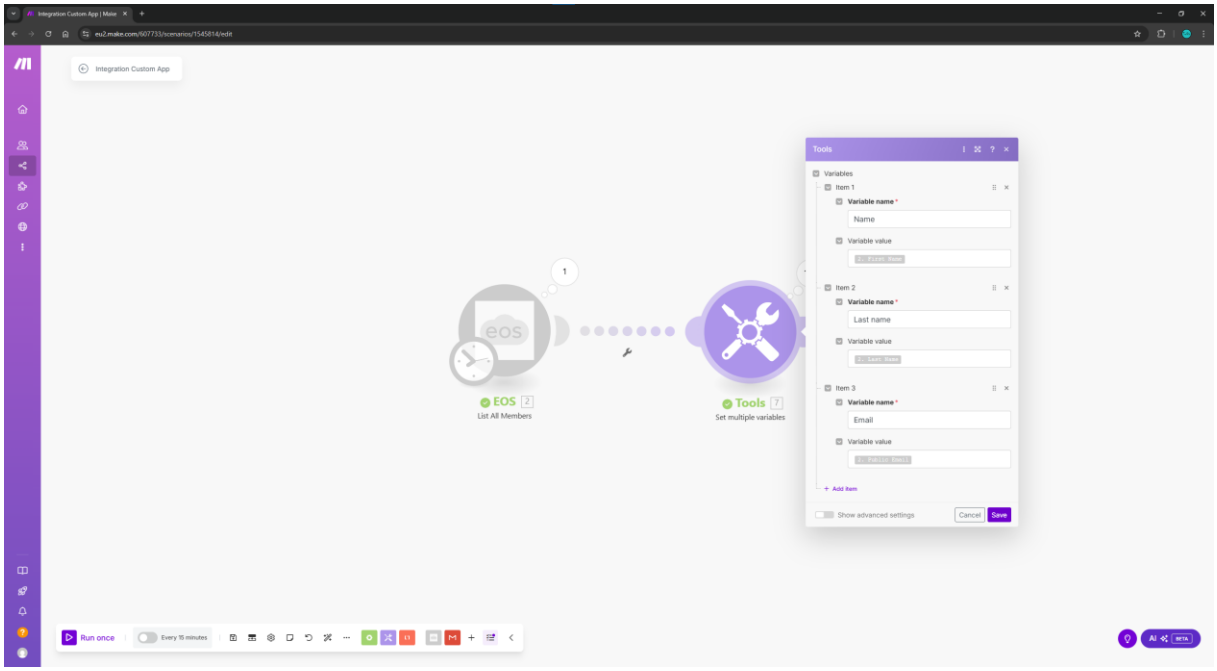
7. Ukázka

Na obrázku 5 je vidět již hotový modul na platformě make.com, v kolonce connections lze přidávat nová připojení, a to zadáním názvu a vložením podepsaného JWT tokenu do kolonky API Key. Modul jako takový je zatím privátní, před publikací bude muset projít ověřením.

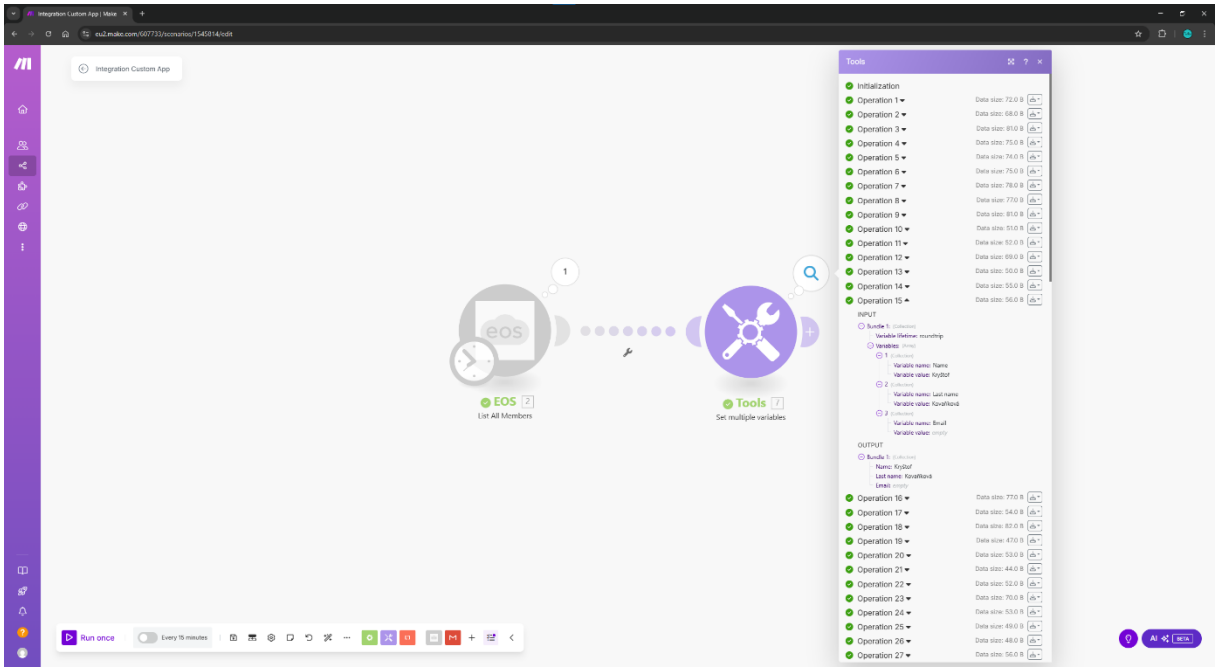


Obrázek 5 Ukázka hotového modulu

Získaná data následně putují do modulu pro vytváření proměnných (nativní). Na obrázku 6 je vidět nastavení tohoto modulu.



Obrázek 6 Ukázka hotového modulu



Obrázek 7 Ukázka hotového modulu

Na obrázku 7 je již vidět jak vypadají samotná data, každá operace je jeden záznam (člen), a základní údaje jsou uloženy do proměnných, se kterými by se mohlo pracovat dál.

8. Závěr

V rámci tohoto projektu byl vytvořen modul, který umožňuje integraci služeb třetích stran do systému EOS. První implementací je propojení se službou make.com, která získává data prostřednictvím dedikovaných API endpointů. Byl navržen a implementován systém autorizace a autentizace, založený na JWT tokenech, který zajišťuje bezpečný přístup k API. Tokeny jsou generovány a šifrovány. Dále byly vytvořeny first-party endpointy, určené pro přímou komunikaci mezi backendem systému EOS a interními aplikacemi. Byly implementovány testy pomocí Pest frameworku pro ověření funkčnosti API. Projekt splňuje požadavky na bezpečnost, rozšiřitelnost a kompatibilitu s REST API standardy a je připraven na budoucí integrace dalších služeb.

Seznam obrázků

Obrázek 1 Ukázka modulů na platformě make.com 10
Obrázek 2 Ukázka kódu pro PEST testy 12
Obrázek 3 Ukázka kódu controlleru 14
Obrázek 4 Class diagram 18
Obrázek 5 Ukázka hotového modulu 19
Obrázek 6 Ukázka hotového modulu 19
Obrázek 7 Ukázka hotového modulu 20

Reference

[1] „eos.cz,“ 26 3 2025. [Online]. Available: https://www.eos.cz/nas-pribeh.

[2] „academy.make.com,“ 26 3 2025. [Online]. Available: https://academy.make.com/.

[3] „medium.com routing,“ 26 3 2025. [Online]. Available: https://medium.com/@sirajul.anik/understanding-route-router-routing-in-laravel-how-is-it-registered-8c3244631af3.

[4] „medium.com orm,“ 26 3 2025. [Online]. Available: https://manoj-shu100.medium.com/eloquent-orm-in-laravel-simplifying-database-interactions-b0269942f190.

[5] „laravel.com,“ 26 3 2025. [Online]. Available: https://laravel.com/docs/12.x.

[6] „pestphp.com,“ 26 3 2025. [Online]. Available: https://pestphp.com/docs/installation.

[7] „dev.to,“ 12 6 2023. [Online]. Available: https://dev.to/alphaolomi/step-by-step-to-pest-php-testing-framework-in-laravel-10-6e1.

[8] „mariadb.org,“ 26 3 2025. [Online]. Available: https://mariadb.org/documentation/#entry-header.

[9] „auth0.com,“ 3 26 2025. [Online]. Available: https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-claims.