# STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

**Obor SOČ: 18. Informatika**

# Automatický Webový Magazín

Vypracoval: Petr Steinbauer

Třída: IT4/A

Škola: Střední škola spojů a informatiky, Tábor, Bydlinského 2474

Kraj: Jihočeský

Konzultant: Ing. Dana Almášiová

# Čestné prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil (a)jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci SOČ. Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

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

V……dne……..

Podpis……..

# Obsah

# Anotace

Tato práce popisuje kompletní vývoj webového magazínu pomocí frameworků. Cílem je vyrobit funkční web, na kterém jsou články generovány pomocí WikiAPI a umělé inteligence. Web bude běžet na lokálním stroji. Téma bylo vybráno, jelikož se autor zajímal o programování, tvorbu webu a umělou inteligenci.

Výsledkem je nejen plně funkční web, který zobrazuje data z databáze, ale funkční skripty pro administrátora, které slouží ke generování obsahu. Celý zdrojový kód je dostupný na stránce GitHub: https://github.com/Mafiosoweb1/synthetic-magazine.

# Klíčová slova

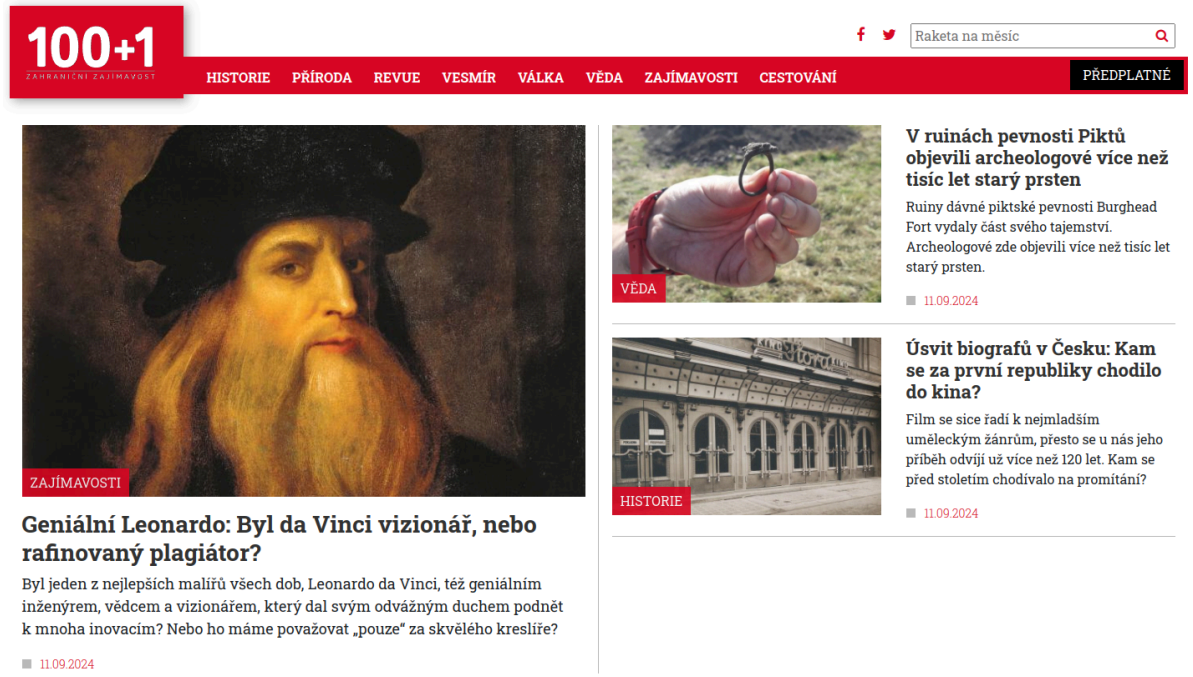
Webový Magazín, Umělá Inteligence, PhP, Nette, Docker

# Annotation

This work describes the complete development of a web magazine using frameworks. The goal is to create a functional website where articles are generated using the WikiAPI and artificial intelligence. The website will run on a local machine. The topic was chosen because the author is interested in programming, web development, and artificial intelligence. The result is not only a fully functional website that displays data from a database but also working scripts for the administrator, which are used to generate content. The entire source code is available on GitHub: https://github.com/Mafiosoweb1/synthetic-magazine.

# 1) Průzkum trhu

Před zahájením vlastního projektu je klíčové analyzovat existující projekty, které se zaměřují na podobný obsah a mohou sloužit jako inspirace. V současnosti existuje řada webových stránek a sociálních profilů, které publikují krátké články na různorodá témata, často rozdělena do kategorií, jako jsou příroda, historie, věda či kultura. Tento formát se ukázal jako populární nejen na klasických webech, ale i na sociálních sítích, kde krátký a poutavý obsah získává široké publikum. Například na platformě X existuje populární uživatel, který pravidelně sdílí 3–5 krátkých článků denně a dokázal si vybudovat komunitu více než 15 000 českých sledujících. Jeho úspěch ukazuje, že existuje výrazná poptávka po rychle konzumovatelném obsahu, který lidé mohou snadno číst během dne. Kromě digitálního prostoru mají některé podobné projekty i tištěnou formu, například časopisy zaměřené na historické události či vědecké zajímavosti. Tyto články bývají různě dlouhé – některé stručné a úderné, jiné obsáhlejší a podrobnější. Obecně platí, že jakýkoliv kvalitní obsah věnovaný historii, přírodě, vědeckým objevům či zajímavosti ze světa technologií může být cenným zdrojem inspirace. Analýza těchto úspěšných modelů pomůže lépe pochopit preference cílové skupiny a přizpůsobit strategii publikace tak, aby měla co největší dosah a efekt.




Zde je ukázka webu 100+1 zahraniční zajímavost (https://www.stoplusjednicka.cz/)

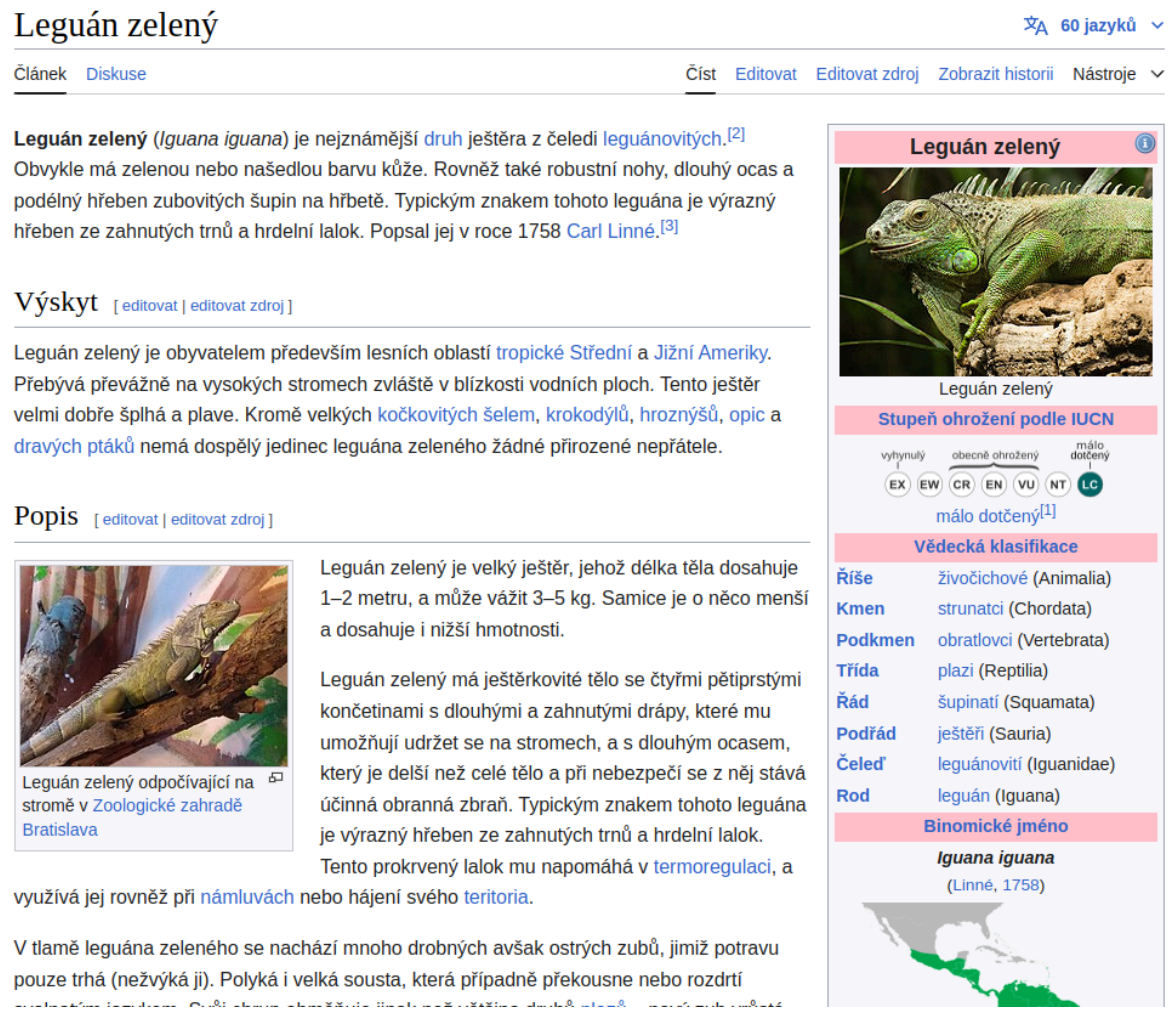
## 1.1) Analýza požadavků a výzkum podkladů

Aby umělá inteligence – konkrétně velký jazykový model (LLM) – mohla generovat články, potřebuje jasně definovaný zdroj informací, který bude analyzovat a interpretovat. Vzhledem k ochraně osobních údajů a etickým omezením se nemůže sama rozhodovat o tom, o čem bude psát. Je proto nezbytné poskytnout jí relevantní podklady, ze kterých bude schopna čerpat fakta a vytvářet obsah. Jako nejdostupnější zdroj informací se nabízí česká Wikipedie (https://cs.wikipedia.org/), která obsahuje rozsáhlou databázi článků na témata související s historií, přírodou i vědou. Pro vědecké a přírodovědné články lze využít i zahraniční jazykové mutace Wikipedie, protože tyto oblasti jsou obvykle zpracovány objektivně a podloženy univerzálně uznávanými fakty. Historické články je však vhodné omezit na českou verzi Wikipedie, jelikož výklad dějin se může v různých zemích výrazně lišit. Aby bylo možné zajistit dostatek kvalitních dat pro generování článků, bude nutné prověřit, jakým způsobem lze efektivně získávat relevantní podklady a zda je tento přístup realistický.

## 1.2) První experiment - test funkčnosti

Prvním krokem bude praktický test – vložení několika ručně vybraných článků z Wikipedie do populárního jazykového modelu a následné vyhodnocení výsledků. Cílem je zjistit, zda umělá inteligence dokáže na základě poskytnutých zdrojů vytvářet kvalitní a čtivé články. Tento experiment pomůže určit, jak dobře model rozumí vstupním datům, zda generuje přesný a smysluplný obsah a jaké jsou případné limity této metody.

Zde je náhodně vybraný článek z Wikipedie:



Leguán zelený

60 jazyků

Článek Diskuse | Číst Editovat Editovat zdroj Zobrazit historii Nástroje

**Leguán zelený** (*Iguana iguana*) je nejznámější druh ještěra z čeledi leguánovitých.[2] Obvykle má zelenou nebo našedlou barvu kůže. Rovněž také robustní nohy, dlouhý ocas a podélný hřeben zubovitých šupin na hřbetě. Typickým znakem tohoto leguána je výrazný hřeben ze zahnutých trnů a hrdelní lalok. Popsal jej v roce 1758 Carl Linné.[3]

### Výskyt [ editovat | editovat zdroj ]

Leguán zelený je obyvatelem především lesních oblastí tropické Střední a Jižní Ameriky. Přebývá převážně na vysokých stromech zvláště v blízkosti vodních ploch. Tento ještěr velmi dobře šplhá a plave. Kromě velkých kočkovitých šelem, krokodýlů, hroznýšů, opic a dravých ptáků nemá dospělý jedinec leguána zeleného žádné přirozené nepřátele.

### Popis [ editovat | editovat zdroj ]



Leguán zelený odpočívající na stromě v Zoologické zahradě Bratislava

Leguán zelený je velký ještěr, jehož délka těla dosahuje 1–2 metru, a může vážit 3–5 kg. Samice je o něco menší a dosahuje i nižší hmotnosti.

Leguán zelený má ještěrkovité tělo se čtyřmi pětiprstými končetinami s dlouhými a zahnutými drápy, které mu umožňují udržet se na stromech, a s dlouhým ocasem, který je delší než celé tělo a při nebezpečí se z něj stává účinná obranná zbraň. Typickým znakem tohoto leguána je výrazný hřeben ze zahnutých trnů a hrdelní lalok. Tento prokrvený lalok mu napomáhá v termoregulaci, a využívá jej rovněž při námluvách nebo hájení svého teritoria.

V tlamě leguána zeleného se nachází mnoho drobných avšak ostrých zubů, jimiž potravu pouze trhá (nežvýká ji). Polyká i velká sousta, která případně překousne nebo rozdrtí

**Leguán zelený**



Leguán zelený

**Stupeň ohrožení podle IUCN**

vyhynulý | obecně ohrožený | málo dotčený

EX EW CR EN VU NT LC

málo dotčený[1]

**Vědecká klasifikace**

| | |
|---|---|
| Říše | živočichové (Animalia) |
| Kmen | strunatci (Chordata) |
| Podkmen | obratlovci (Vertebrata) |
| Třída | plazi (Reptilia) |
| Řád | šupinatí (Squamata) |
| Podřád | ještěři (Sauria) |
| Čeleď | leguánovití (Iguanidae) |
| Rod | leguán (Iguana) |

**Binomické jméno**

***Iguana iguana***

(Linné, 1758)



Nyní bylo nutné navrhnout vhodný **prompt** (instrukce pro AI), který zajistí, že umělá inteligence správně pochopila zadání a vygeneruje kvalitní článek. Je vhodné, aby prompt byl co nejpřesnější, aby obsahoval jasné instrukce ohledně odpovědi. Tím, že odpověď vynutíme ve formátu JSON s jasně definovanými klíči, zajistíme snadné další zpracování.

Ahoj, poslu ti clanek z wikipedie, potreboval bych abys ho upravil jako popularni kratky clanek - neco mezi 3-5 odstavci. Bude to v magazinu 1001 zajimavosti a pak na socialni site.

odpoved chci jako json, s kliči: header, text, imagePath, imagePath, imageTitle

V tomto promptu je vidět:

- oslovení - nepraktické, spíše ze zvyku
- zdroj - informaci o tom odkud jsou data (z wikipedie)
- zadání - instrukce, že z dat je požadován článek o délce 3-5 odstavců
- forma - informací o tom, že jde o zábavný časopis a sociální sítě se nastaví AI i forma jak má článek generovat
- výstupní formát - informací že výstup má být ve formátu json a jaké mají být klíče se AI sdělil nejen výstupní formát, ale rovnou (vhodně pojmenovanými klíči) i hodnoty které mají být vyplněny. Například vhodně pojmenovaný klíč “header” je natolik jasný, že lze předpokládat že do něj inteligence nevyplní například datum.

Formát JSON byl zvolen především z praktických důvodů. Rovněž mohl být zvolen i formát XML, ale v dalším kódu by se musela řešit konverze dat, která je ve zvoleném jazyku (php) obtížnější. Některé formáty jsou naopak nevhodné, například csv (excel) - například kvůli odsazením (entrům) v buňkách.

Zde je obsah, který nám umělá inteligence vytvořila.

```
{
  "header": "Leguán zelený: Král tropických pralesů",
  "text": [
    "Leguán zelený (Iguana iguana) je fascinující druh ještěra, který se proslavil svou zelenou barvou a majestátním vzhledem. Tito velcí plazi, kteří mohou dorůstat až dvou metrů, žijí především v tropických lesích Střední a Jižní Ameriky. Jsou mistři v lezení po stromech a v plavání, což jim umožňuje přežívat v blízkosti vodních ploch.",
    "Leguán zelený má robustní tělo s dlouhým ocasem, který používá jako zbraň, a výrazný hřeben z ostnů na zádech. Díky své schopnosti měnit barvu kůže a regulovat tělesnou teplotu se dokonale přizpůsobil životu v horkých tropických podmínkách. Kromě několika predátorů, jako jsou kočkovité šelmy nebo krokodýli, nemají dospělí leguáni mnoho nepřátel.",
    "Tento býložravý ještěr je oblíbeným zvířetem i mezi chovateli. Jeho nenáročný jídelníček, skládající se z listí a ovoce, a dlouhá životnost činí z leguána skvělého společníka. Avšak jeho chov vyžaduje velké terárium a speciální péči, aby mohl naplno rozvinout své přirozené chování."
  ],
  "imagePath": [
    "MC GruenerLeguan.jpg",
    "Iguana_on_tree.jpg"
  ],
  "imageTitle": [
    "Leguán zelený ve svém přirozeném prostředí",
    "Leguán odpočívající na stromě"
  ]
```


Tato odpověď byla uspokojivá, nicméně příliš akademická a doslovná, proto byl prvotní návrh rozšířen o další zadání. Po umělé inteligenci se nově požaduje aby - pokud ho zná - doplnila do článku nějaký zábavný fakt o probíraném tématu. Šlo především o to dosáhnout hravějšího a přirozenějšího stylu psaní.

V experimentu byla použita přirozená řeč, protože na výsledek budoucího promptu nemá tato forma vliv. V ukázce je vidět dotaz a odpověď AI.

hele jo dobrý, vis nejakou zajimavost o leguanovi? pokud ano, tak ji vepiš do textu

Zajímavost o leguánovi zeleném je, že tento plaz má třetí oko, tzv. temenní oko, které se nachází na vrcholu jeho hlavy. Toto oko neslouží k běžnému vidění, ale pomáhá leguánovi vnímat změny světla a stínu, což ho chrání před predátory útočícími zhora. Teď to vepíšu do textu:

json Zkopírovat kód


```
{
  "header": "Leguán zelený: Král tropických pralesů",
  "text": [
    "Leguán zelený (Iguana iguana) je fascinující druh ještěra, který se proslavil
    "Jednou z nejzajímavějších vlastností leguána zeleného je jeho třetí oko, známé
    "Leguán zelený má robustní tělo s dlouhým ocasem, který používá jako zbraň, a v
    "Tento býložravý ještěr je oblíbeným zvířetem i mezi chovateli. Jeho nenáročný
  ],
  "imagePath": [
    "MC GruenerLeguan.jpg",
    "Iguana_on_tree.jpg"
  ],
  "imageTitle": [
    "Leguán zelený ve svém přirozeném prostředí",
    "Leguán odpočívající na stromě"
  ]
}
```

## 1.3) Druhý experiment - zábavnější text

Z uvedené ukázky je zřejmé, že odpověď bude odpovídat požadavkům. Proto byl sestaven prompt, který zadání vysvětluje podrobně, aby generátor textu věděl, co je přesně požadováno.

V níže uvedeném screenshotu je zobrazen výsledný prompt v kódu. V proměnné $categoryText jsou vypsány kategorie, jako “číslo” a “hodnota” kde číslo je například 1, a hodnota například “příroda, ekologie, zvířata, rostliny”. Kompletní seznam bude vysvětlen dále v kódu.

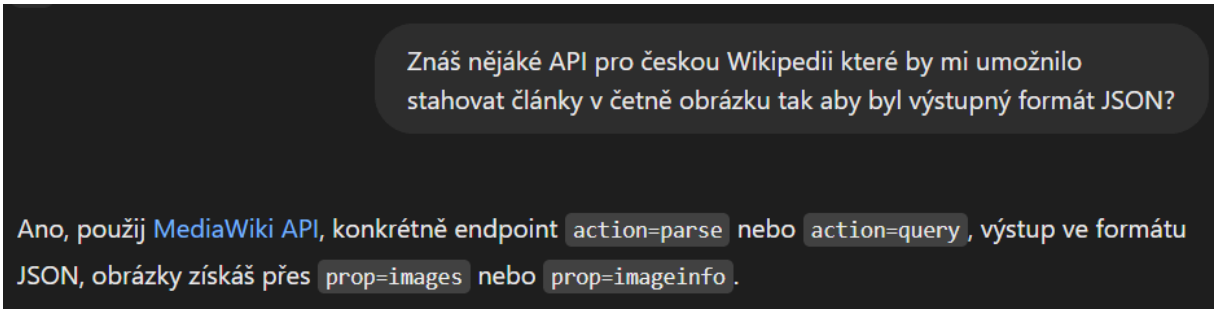
V nadpisu je použit nadpis požadovaného článku z databáze a v textu je stažený text.

```
                    "content" =>
"Vygeneruj článek pro magazín se zajímavostmi, konkrétně jeho nadpis a obsah.
Poskytnu ti obsah na toto téma z Wikipedie a ty mi vrať odpověd
jako JSON soubor, s klíči heading, content a categoryId.
Json musi byt validni a neměl by obsahovat žádné anotace ```json {' a podobně.
Do categoryId vyplň číslo kategorie, která odpovídá obsahu: " . $categoryText . "
a to dle tvého vygenerovaného obsahu.
Potřebuji aby ta kategorie seděla co nejpřesněji podle toho obsahu.
Pokud budeš vědět nějakou zajímavost, tak ji můžeš přidat do obsahu.
Cílem je aby článek byl pro lidi zajímavý a přitažlivý, může být i lehce vtipný pokud to dané téma dovolí.
Do hlavního textu můžeš použít smajlíky a emotikony, ale ne příliš, jen jako zajímavý element pro oko.
----------
Nadpis: " . $article->getSourceHeading() . "
Text: " . $article->getSourceContent(),
```

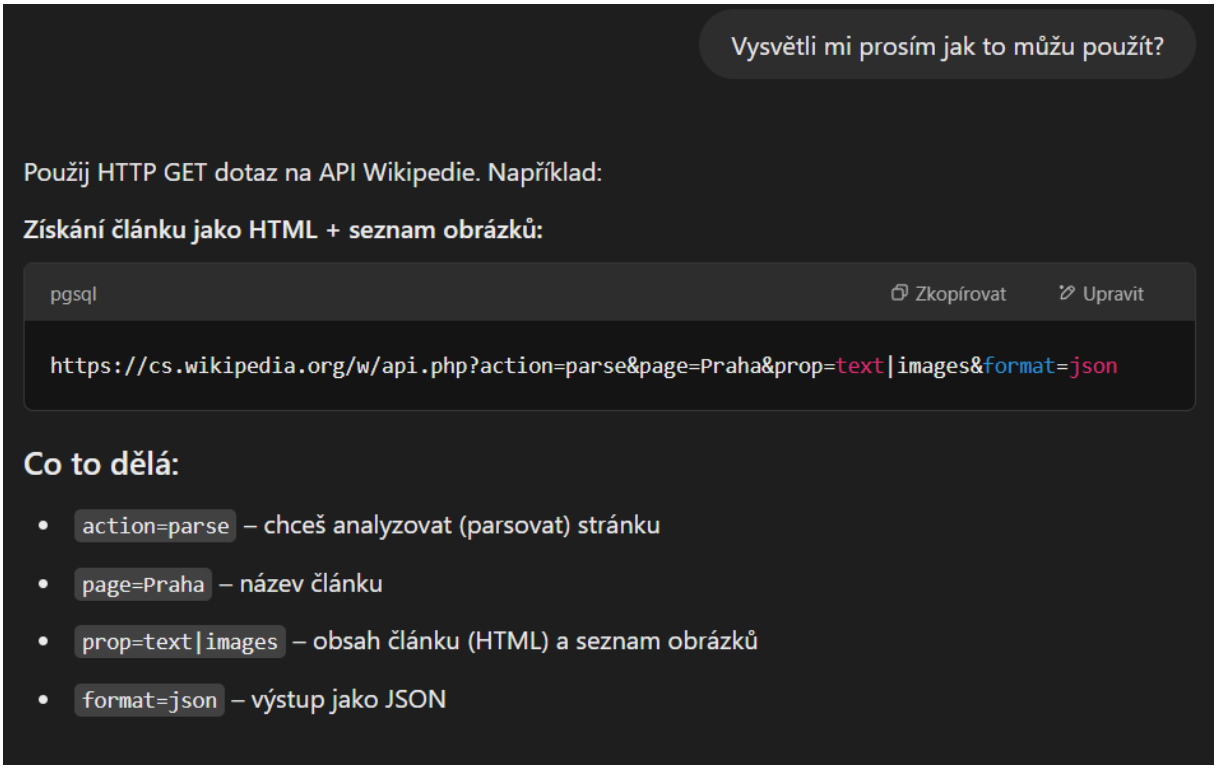
Toto nám zajistí, že prompt je použitelný pro libovolné téma z Wikipedie.

# 1.4) Třetí experiment - automatické získávání dat

Nebylo žádoucí si články dohledávat ručně na wikipedii, je vhodné využívat nějaké volně dostupné api, které wikipedie poskytuje. Než prohledávat internet postaru vyhledávačem, je vhodnější se na to rovnou zeptat umělé inteligence.




Bylo prověřeno API a na první pohled nebylo jasné, jak se s API pracuje, jednoduchá otázka pro LLM potvrdí domněnku a rovnou ukáže jak o data žádat.




Po pár experimentech a porad s LLM je jasné, jaké bude finální url pro náhodné články z české wikipedie a bude zobrazeno dále v kódu.

# 2) Výběr technického stacku

## 2.1) Kritéria

Byl hledán programovací jazyk vhodný pro generování internetových stránek. Z povahy věci bude potřeba nějaká databáze, do které se budou moci ukládat stažená data. Rovněž je vhodné nezačínat projekt od prázdného projektu, abychom nemuseli řešit bezpečnostní problémy a další výzvy a mohli se soustředit na efektivitu psaní kódu.

## 2.2) Vybrané nástroje

Jako programovací jazyk byl zvolen nejpopulárnější programovací jazyk, který i přes mnohaletou historii v roce 2014 obsluhoval 77% veškerého webového provozu. [1]. - **PHP** [2]

Protože není žádoucí řešit vývoj projektu od nuly, byl vybrán i webový framework pro PHP. Zde volba padla na **Nette Framework** [3], a to z důvodů české lokalizace dokumentace.

Významným plusem byla existence již hotového sandboxu Contributte [4], který stačí spustit a rovnou máte k dispozici celý framework. Contributte/Nette poskytuje rovnou (mimo jiné) tyto rozhraní:

- **Routy** [5] (hezké url adresy)
- **Entity** [6] (elegantní způsob pro ukládání a čtení dat z databáze, (z projektu Doctrine [7]))
- **Commandy** [8] (možnost volat příkazy z commandu, čímž odpadá nutnost psát administraci)
- **Utils** [9] (metody pro časté operace, například Strings pro modifikaci textů nebo Url pro generování validních url adres)
- **Latte** [10] (šablonovací nástroj pro psaní bezpečného html)

Jako grafická framework (hotové css předpisy) byl zvolen populární Bootstrap 5.3. [11] Ten umožňuje psát html a neřešit design, protože obsahuje předpřipravené designové prvky (například menu, tlačítka, rozmístění prvků)

Nebylo rovněž žádoucí řešit instalaci, konfiguraci, nebo problémy mezi serverovou a lokální verzí, a tak byl použit nástroj pro virtualizaci **Docker**[15]. Ten umožní aplikaci rovnou spustit a provozovat.

## 2.3) Instalace

Instalace nástrojů docker a následně Contributte (Nette Framework) je vcelku přímočará záležitost. Docker se nainstaluje do operačního systému, a hotový sandbox Contributte se v něm pouze založí přes IDE editor Phpstorm, kterým bude programován kód programu. Tím se v kontejnerech dockeru spustí databáze, webový server nginx, php a nástroj pro případnou editaci databáze Adminer. Web je rovnou přístupný na webové adrese http://127.0.0.1

## 2.4) Očištění ukázkového sandboxu

Protože výchozí sandbox obsahuje části, které nepotřebujeme (například administraci), tak je z projektu odebereme. Projekt je poskytován pod volnou licencí, takže je v pořádku vyjmout všechny zmínky o sandboxu a připravit si je pro požadovaný účel.
Nakonec projekt bude zveřejněn na GIT uložišti na adrese [12], kam budou postupně commitovány (zasílány) změny v kódu.

# 3) Tvorba backendu

Jako forma backendu byly zvoleny takzvané Commands - příkazy. Jsou dostupné pouze v příkazové řádce na serveru, kam se nikdo kromě administrátora nedostane. Odpadá tak nutnost řešit autorizaci uživatelů v nějaké webové administraci.

# 3.1) Příprava struktury dat

Pro uložení dat v Nette slouží takzvané Entity - mocný nástroj který, odstíní programátora od nutnosti se učit jazyk SQL. Jen se určí, jaké entity (tabulky) se mají vytvořit a jaké data (sloupce) a jejich forma (číslo, text, datum) se má použít. Díky tomuto nastavení se zajistí nejen kvalita dat (do čísla například nelze uložit text), ale i snadný způsob, jak data číst a zapisovat bez znalosti SQL.

```php
<?php declare(strict_types=1);

namespace App\Model\Database\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Model\Database\Repository\ArticleRepository")
 * @ORM\HasLifecycleCallbacks
 */
 Petr Steinbauer *
class Article extends AbstractEntity
{
    use TId;
    use TCreatedAt;
    use TUpdatedAt;

    /** @ORM\Column(type="integer", unique=true) */
    private int $wikiId;  3 usages

    /** @ORM\Column(type="integer", unique=false, nullable=TRUE) */
    private ?int $categoryId=null;  2 usages

    /** @ORM\Column(type="string", length=255, nullable=FALSE, unique=false) */
    private string $heading;  2 usages

    /** @ORM\Column(type="text", nullable=FALSE, unique=false) */
    private string $content;  2 usages

    /** @ORM\Column(type="string", length=255, nullable=FALSE, unique=false) */
    private string $sourceHeading;  2 usages

    /** @ORM\Column(type="text", nullable=FALSE, unique=false) */
    private string $sourceContent;  2 usages

    /** @ORM\Column(type="string", nullable=TRUE, unique=false) */
    private ?string $picture=null;  2 usages

    /** @ORM\Column(type="integer", nullable=FALSE, unique=false, options={"default":0}) */
    private int $status = 0;  2 usages
```

Na screenshotu je vidět Entita Article. Jediná tabulka, která obslouží všechny požadavky, které na projekt klademe.

Vytvořením této třídy je Nette nařízeno že bude existovat tabulka Article, a bude obsahovat tyto atributy (sloupce) a to včetně jejich formy (int, string, date, atd…)

| Název | Typ | Vysvětlení | Vlastnosti |
|---|---|---|---|
| TId | int | Přidává entitě primární klíč id. | Automatické generování identifikátoru. |
| TCreatedAt | Datetime | Zajišťuje ukládání data a času vytvoření záznamu. | Hodnota se nastaví při vytvoření objektu. |
| TUpdatedAt | Datetime | Zajišťuje ukládání data a času poslední aktualizace. | Automatická aktualizace při změně dat. |
| wikiId | int | ID článku z externího systému (např. Wikidata). | unique=true, nelze NULL. |
| categoryId | ?int | ID kategorie, pod kterou článek spadá. | Může být NULL, není unikátní. |
| heading | string | Nadpis článku. | Max. 255 znaků, povinné pole. |
| content | string | Hlavní text článku. | Typ text, povinné pole. |
| sourceHeading | string | Původní nadpis, ze kterého byl článek vygenerován. | Max. 255 znaků, povinné pole. |
| sourceContent | string | Text původního zdrojového obsahu. | Typ text, povinné pole. |
| picture | ?string | Odkaz nebo název obrázku doprovázejícího článek. | Max. 255 znaků, může být NULL. |
| status | int | Stav článku (např. koncept, schváleno, publikováno). | Výchozí hodnota 0, povinné pole. |

Z tabulky je zřejmé, které sloupce obsahují který údaj. O datovou strukturu se tedy dále není nutno starat.

## 3.2) První command - stažení dat z wikipedie a uložení

Poté co je definována datová struktura, tak se bude programovat script, který reálně získá data z wikipedie. Přesněji stáhne data z API a uloží je.

### 3.2.1) Registrace nového Commandu

Dle specifikace Commandů [8] je nutné zaregistrovat command do Nette, a to tak že vytvoříme ve správné namespace vhodně pojmenovanou Classu - byl zvolen název ArticleCreateCommand. Název jasně napovídá, že v tomto commandu se vytváří nové články.

ArticleCreateCommand.php

```php
<?php declare(strict_types=1);

namespace App\Console;

use ...

#[AsCommand(name: 'article:create')] no usages Petr Steinbauer
class ArticleCreateCommand extends Command
{
    //entityManager slouží pro práci s databází
    private EntityManagerDecorator $entityManager; 3 usages

    // Konfigurace příkazu/commandu
    protected function configure(): void no usages Petr Steinbauer
    {
        $this->setDescription( description: 'Creates article from wiki.');
        $this->addArgument(
            name: "count",
            mode: InputArgument::OPTIONAL,
            description: "Number of articles to create",
            default: 1
        );
    }

    // Konstruktor třídy - Naplní EntityManager do třídy
    public function __construct(EntityManagerDecorator $entityManager)
    {
        parent::__construct();
        $this->entityManager = $entityManager;
    }
```

V kódu je vidět založení nového commandu, pod volacím názvem **article:create**. Aby command byl funkční je dle dokumentace nezbytné, aby dědil (extends) od připravené classy Command. Dědění classy Command nám udělí povinnost mít metody **configure** a **execute**. První si probereme zde a druhou v další části.

**Privátní proměnná $entityManager** - slouží k práci s databází bez znalosti SQL.

**Metoda configure** je nastavení tohoto commandu, definuje se mu jak popisek, tak možnost mít nepovinný argument (pojmenovaný "count"), který je nepovinný, z popisku říká, že se jedná o počet stažených článků a rovněž diktuje výchozí počet stažených článků (jeden).

**Metoda __construct:** v php se při jakémkoliv volání jakékoliv metody volá metoda __construct. Pokud má parametry, Nette za Vás zajistí, že se parametry předají. V této metodě plníme privátní proměnnou $entityManager.

## 3.2.2) Funkční kód - stažení wikipedie článků

Na obrázku je vidět jak se commandy volají - s parametrem **list** se vypíší všechny

```
php /var/www/html/bin/console list
Console Tool

Usage:
  command [options] [arguments]

Available commands:
  completion                      Dump the shell completion script
  hello                           Hello world!
  help                            Display help for a command
  list                            List commands
 article
  article:create                  Creates article from wiki.
  article:generate                Generates article from sourceContent.
```

A takto náš konkrétní **article:create** s parametrem 3 - s požadavkem na stažení třech článků

```
php /var/www/html/bin/console article:create 3
Article created.
Article created.
Article created.
```

rovněž je možno použít zkratku a:c 3 - kombinace a/c je jedinečná a tak program ví co spustit.

```php
public function getWikiData(): mixed  1 usage  new *
{
    //nacteme clanky z wiki API
    $url = new Url( url: 'https://cs.wikipedia.org/w/api.php'); //sestavení url
    $url->setQueryParameter( name: 'action', value: 'query');
    $url->setQueryParameter( name: 'generator', value: 'random');
    $url->setQueryParameter( name: 'grnnamespace', value: '0');
    $url->setQueryParameter( name: 'grnlimit', value: '1');
    $url->setQueryParameter( name: 'prop', value: 'extracts|pageimages');
    $url->setQueryParameter( name: 'exintro', value: '');
    $url->setQueryParameter( name: 'explaintext', value: '');
    $url->setQueryParameter( name: 'piprop', value: 'original');
    $url->setQueryParameter( name: 'format', value: 'json');

    //načtení dat z url(absolutní url)
    $wikiData = FileSystem::read($url->getAbsoluteUrl());
    return json_decode($wikiData, associative: true); //dekódování jsonu
}
```

Metoda **execute** pro vytvoření článku bude potřebovat dvě metody, první z nich metoda **getWikiData** sestaví správnou url pro stažení jsonu a pak data stáhne pomocí Utility Filesystem [9], a nahraje je do proměnné $wikiData. Tuto proměnnou pak převede pomocí json_decode [9] na objekt. Tento objekt pak metoda getWikiData vrátí.

```php
public function findImage(mixed $page): ?string  1 usage  new *
{
    $exceptions = ['.svg','.gif','.jpeg','.webp','.png','.jpg']; //koncovka musí být obrázek
    $pictureUrl = null;  //url obrázku, defaultně null
    if (isset($page['original']) && isset($page['original']['source'])) { //pokud existuje obrázek
        $foundValidImage = false; //obrázku nevěříme
        foreach ($exceptions as $exception) {
            if (str_ends_with($page['original']['source'], $exception)) {
                $foundValidImage = true; //obrázek je validní
                break; //ukončení foreach
            }
        } //konec break
        if ($foundValidImage) { //pokud je obrázek validní
            $pictureUrl = $page['original']['source']; //nastavení obrázku
        }
    }
    return $pictureUrl;
}
```

Druhou metodu, kterou metoda execute potřebuje je metoda **findImage**. Ta má za úkol najít obrázek ve stažených datech a pokud neexistuje nebo se nejedná o obrázek (ale například pdf), tak vrátit null.

**Samotná metoda Execute**

```php
protected function execute(InputInterface $input, OutputInterface $output): int  no usages
{
    $count = $input->getArgument( name: 'count');
    for ($i = 1; $i <= $count; $i++) { //cyklus pro vytvoření článků
        $wikiList = $this->getWikiData(); //získání dat z wiki
        foreach ($wikiList['query']['pages'] as $page) { // Iterace přes články

            // test 1 Má článek z wiki méně než 100 znaků - pokračuj na další iteraci
            if (strlen($page['extract']) < 100) { continue; }
            // test 2 obsahuje v názvu slovo Rozcestník - pokračuj na další iteraci
            if (str_contains($page['title'], 'Rozcestník')) { continue; }

            $pictureUrl = $this->findImage($page); //nalezení obrázku

            $article = new Article(); //vytvoření nového článku
            $article->setCreatedAt();
            $article->setHeading( heading: '');
            $article->setContent( content: '');
            $article->setWikiId($page['pageid']);
            $article->setPicture($pictureUrl); //nastavení obrázku
            $article->setSourceHeading($page['title']);
            $article->setSourceContent($page['extract']);
            $article->setStatus( status: Article::STATUS_CONCEPT);
            $this->entityManager->persist($article); //uložení článku

            try {
                $this->entityManager->flush(); //uložení do databáze
            } catch (Exception $e) { //chyba při ukládání
                $output->writeln( messages: 'Error: ' . $e->getMessage());
                continue;
            }
        } //konec foreach,když je continue tak se přeskočí na další iteraci
        $output->writeln( messages: 'Article created.');
    }
    $output->writeln( messages: 'All articles created.');
    return 0;
}
```

Metoda je volaná commandem a vykonává v cyklu stažení článků. Pokud je volána například s parametrem 3 tak se 3x vykoná její vnitřní část.

Ve vnitřní části (ve for cyklu) se zavolá již popsaná metoda getWikiData která vrátí data a předá je do proměnné $wikiList. I když je v reálu přítomen jen jeden článek, tak API wikipedie vrací data jako pole v rozměru [“query”][“pages”] - proto je tento rozměr procházen a vrácen do proměnné $pages.

Provedou se dva testy na validitu získaných dat (krátký text a rozcestníky), při neúspěchu se jde na další článek. Článků je mnoho, je neefektivní řešit nevalidní články, prostě se bez náhrady přeskočí.

Teprve potom se ověří existence obrázku pomocí již popsané metody findImage.

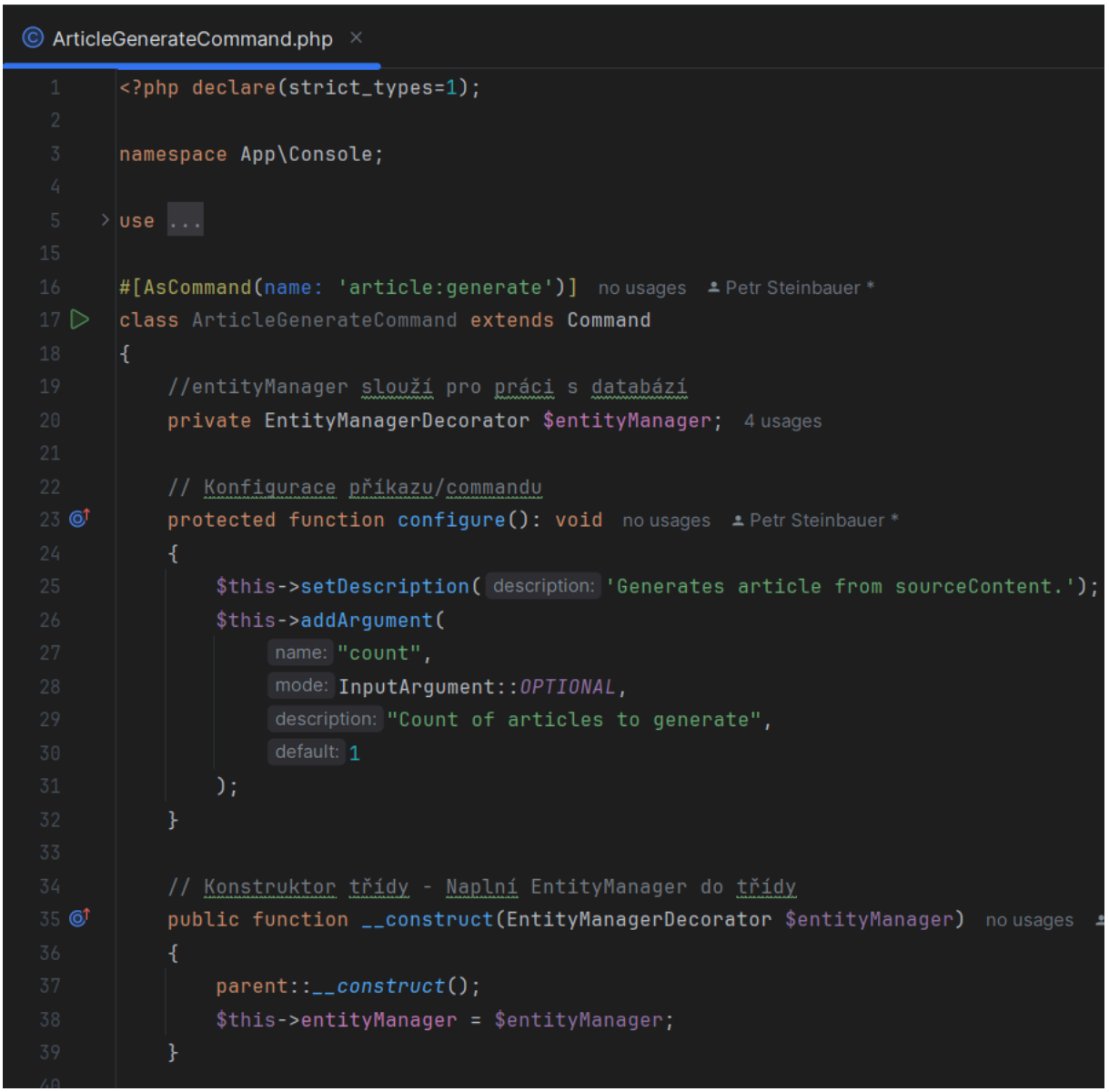
Jako poslední krok se obrázek uloží. Zavolá se entita Article, která již byla definována v přípravě, naplní se data a entita se uloží (nejdříve v php (persist) a pak do databáze (flush)). Kdyby se článek nepovedlo uložit, vypíše se chyba a jde se na další článek.

Na konci cyklu se vypíše text o tom, že byl článek stažen a potom jako poslední se napíše hláška o dokončení.

## 3.4) Druhý command - vygenerování článku přes Chat GPT a uložení dat

### 3.4.1) Registrace commandu

Command registrujeme identickým způsobem jako první command.




Rozdíl je prakticky jen ve jméně commandu a popisku toho, co command dělá.

## 3.4.2) Funkční kód

Stejně jako v předchozím commandu bude třeba pomocná metoda - v tomto případě je to metoda **callChatGPT**.

```php
public function callChatGPT(mixed $article): mixed  1 usage  Petr Steinbauer *
{
    //vytvoření textu pro kategorie z Article::CATEGORIES_NAMES_GPT
    //kde je pole s kategoriemi a jejich čísly
    $categoryText = "";
    foreach (Article::CATEGORIES_NAMES_GPT as $key => $value) {
        $categoryText .= '"' . $value . '": ' . $key . ", ";
    }

    $openAi = new OpenAi(getenv( name: 'CHAT_GPT_API_KEY'));
    $model = 'gpt-4o-mini';
    $complete = $openAi->chat([
        'model' => $model,
        'messages' => [
            [
                "role" => "user",
                "content" =>
                    "Vygeneruj článek pro magazín se zajímavostmi, konkrétně jeho nadpis a obsah.
Poskytnu ti obsah na toto téma z Wikipedie a ty mi vrať odpověd
jako JSON soubor, s klíči heading, content a categoryId.
Json musi byt validni a neměl by obsahovat žádné anotace '```json {' a podobně.
Do categoryId vyplň číslo kategorie, která odpovídá obsahu: " . $categoryText . "
a to dle tvého vygenerovaného obsahu.
Potřebuji aby ta kategorie seděla co nejpřesnější podle toho obsahu.
Pokud budeš vědět nějakou zajímavost, tak ji můžeš přidat do obsahu.
Cílem je aby článek byl pro lidi zajímavý a přitažlivý, může být i lehce vtipný pokud to dané téma dovolí.
Do hlavního textu můžeš použít smajlíky a emotikony, ale ne příliš, jen jako zajímavý element pro oko.
----------
Nadpis: " . $article->getSourceHeading() . "
Text: " . $article->getSourceContent(),
            ],
        ],
    ]);
    $data = json_decode($complete,  associative: true);
    //v $result['choices'][0]['message']['content'] je odpověď z chatGPT
    return json_decode($data['choices'][0]['message']['content'],  associative: true);
}
```

Tato metoda se stará o zavolání API ChatGPT[13] se správným klíčem (GET_GPT_API_KEY) a navrženým promptem. Součástí promptu je seznam kategorií, o který se tato metoda taky stará.

ChatGPT API samo o sobě vrací odpověď jako json, proto je třeba pomoci json_decode “rozbalit” na objekt. A teprve v tomto objektu je reálná odpověď ChatGPT, která byla promptem požadována také jako json, takže ji znovu rozbalíme pomocí json_decode a vrátíme na výstup.

**Metoda execute**

```php
protected function execute(InputInterface $input, OutputInterface $output): int  no usages
{
    //vytáhne všechny články, které mají prázdný nadpis, obsah a mají obrázek
    $articles = $this->entityManager->createQueryBuilder()
        ->from( from: Article::class,  alias: 'a')
        ->select( select: 'a')
        ->where( predicates: 'a.heading = :heading')->setParameter( key: 'heading',  value: '')
        ->andWhere('a.content = :content')->setParameter( key: 'content',  value: '')
        ->andWhere('a.picture IS NOT null')
        ->setMaxResults($input->getArgument( name: 'count')) //počet článků
        ->getQuery()
        ->getResult();

    if (count($articles) == 0) {  //pokud nejsou žádné články, končí
        $output->writeln( messages: 'All articles are generated.');
        return 0;
    }

    foreach ($articles as $article) {
        $result = $this->callChatGPT($article);
        // test if $resultText['heading'] is null, continue
        if ($result['heading'] == null) {
            continue;
        }
        // uložení hodnot do článku
        $article->setHeading($result['heading']);
        $article->setContent($result['content']);
        $article->setCategoryId($result['categoryId']);
        $article->setStatus(Article::STATUS_PUBLISHED);
        $article->setUpdatedAt();
        //uložení článku, nejdříve do php, poté do databáze
        $this->entityManager->persist($article);
        $this->entityManager->flush();

        $output->writeln( messages: 'Article generated.');
    }
    $output->writeln( messages: 'All articles generated.');
    return 0;
}
```

Metoda execute se stejně jako v ArticleCreateCommand pouští automaticky, použije $entityManager pro stažení dat z databáze dle předem určených podmínek (bez nadpisu, bez textu ale s fotkou) a to v počtu který určil argument commandu.

Pokud nenajde žádné články, tak se po vypsání příslušné hlášky ukončí, v opačném případě po jednom prochází jednotlivé články (které obsahují jen zdrojové wiki data) a nad každým článkem volá metodu callChatGPT.

Pokud je výsledek validní (testuje se existence nadpisu), tak se získaná data uloží do databáze. Pak následují jen hlášky, které vypisují že byl článek modifikován.

## 3.5) Jak se tedy plní databáze?

Použitím kombinace dvou příkazů: **console a:c 10** a **console a:g 10**. První stahuje náhodné články z Wikipedie a ukládá je do databáze, druhý si najde články, které ještě nebyly vygenerovány a mají fotografii, a pomocí ChatGPT je vygeneruje a uloží.

# 4) Tvorba frontendu

Teď, když jsou data v databázi, je možno se věnovat jejich vykreslení na web pro webový prohlížeč a konzumenty obsahu.

## 4.1) Jak funguje routování a procházení webu

V Nette funguje routování tak, že když uživatel zadá URL adresu (např. example.cz/kontakt), systém pomocí tzv. routeru zjistí, která část aplikace má tuto žádost zpracovat.
Router tedy přeloží URL na "adresu v kódu", předá ji danému presenteru, který si připraví potřebná data (např. z databáze) a předá je šabloně. Výsledkem je HTML stránka, která se odešle zpět do prohlížeče. Vše funguje automaticky podle definovaných pravidel. Stačí tedy vyrobit akci v presenteru a příslušné šablony. Jednu společnou pro všechny obsahy a jednu pro každý obsah (výpis článků, detail)

# 4.2) Hlavní šablona, menu

```
{block #main}
    <div class="container-fluid d-flex flex-column"> {* Začátek hlavního kontejneru *}
        <header class="masthead"> {* Začátek hlavicky *}
            <div class="inner">
                <h3 class="masthead-brand" title="Synthetic Magazine">
                    <a n:href=":Front:Home:default">
                        <img src="/favicon-32x32.png" alt=""> {* logo webu *}
                        <span class="d-none d-lg-inline">  {* Název webu *}
                            Synthetic Magazine
                        </span>
                    </a>
                </h3>
                <nav class="nav nav-masthead justify-content-center"> {* Navigace *}
                    <a n:class="$presenter->isLinkCurrent(':Front:Home:default') ? active, nav-link"
                       n:href=":Front:Home:default">
                            {* Odkaz na domovskou stránku *}
                            {* n: = NETTE udělej odkaz do Front:Home:default *}
                            Home
                    </a>
                    {* Výpis kategorií *}
                    {foreach \App\Model\Database\Entity\Article::CATEGORIES_ENABLED as $categoryId}
                        <a n:class="$presenter->isLinkCurrent(':Front:Home:category', $categoryId) ? active, nav-link"
                           n:href=":Front:Home:category, $categoryId">  {* Odkaz na kategorii *}
                                {\App\Model\Database\Entity\Article::CATEGORIES_NAMES[$categoryId]}  {* Název kat. *}
                        </a>
                    {/foreach} {* Konec výpisu kategorií *}
                </nav> {* Konec navigace *}
            </div>
        </header> {* Konec hlavicky *}

        <div class="container"> {* Začátek obsahu *}
            <main role="main" class="inner">
                {include content} {* Vložení obsahu *}
            </main>
        </div> {* Konec obsahu *}

        <footer class="mastfoot p-3 mt-auto text-center"> {* Začátek patičky *}
            <div class="inner">
                © {=date('Y')} Synthetic Magazine, All rights reserved, Mafiosoweb1
            </div>
        </footer> {* Konec patičky *}
    </div>
{/block}
```

Hlavní šablona webu v souboru @layout.latte obsahuje: hlavičku, patičku i blok pro vypsání obsahu. Šablonovací jazyk Latte ma svojí syntaxi. |Výsledné html je napsané pro bootstrap 5.3, takže má rovnou pěkný defaultní vzhled. V hlavičce se vykresluje logo (je použitý obrázek z favicon), nadpis webu (jen pro PC variantu, na mobilu vidět není) a pak odkaz na hlavní stranu a pomocí foreach se vygenerují i odkazy do jednotlivých sekcí.

Lze si povšimnout, že některé tagy obsahuji attributty s prefixem n: - to znamená, že je zpracovává nette a dělá nad nimi logiku. Například n:href=”Front:Home:default” říká: “Nette, sem vygeneruj odkaz na hlavní stranu”, nebo n:href=”Front:Home:category, 1” říká: “Nette, sem vygeneruj odkaz na sekci s id 1”. Stejná logika lze pak použít i pro ostatní attributty,

například class. Tento zápis podporuje i podmínky if, takže lze například zjišťovat která sekce je aktivní a dle toho přidávat nebo odebírat z class některé části, například “active”.

## 4.3) Obsah navštívené stránky

Pokud Vás Nette Framework přesměruje na nějakou platnou část - například do kategorie - tak naplní požadované proměnné a zavolá příslušnou šablonu. Její obsah se pak vypíše v hlavním @layout.latte.

```
{block #content}
    <div class="mb-3 jumbotron p-3">
        <div class="card-body">
            <div class="row">
                <div class="col-12 col-md-9">
                    <h1 class="card-title" n:block="#title">{$article->getHeading()}</h1>
                    <p class="card-text">{$article->getContent()|breaklines}</p>
                    <p class="card-text">
                        <span class="badge bg-info">
                            <a n:href=":Front:Home:category, categoryId: $article->getCategoryId()">
                                {\App\Model\Database\Entity\Article::CATEGORIES_NAMES[$article->getCategoryId()]}
                            </a>
                        </span>
                        <span class="badge bg-secondary">{$article->getUpdatedAt()->format('d.m.Y H:i')}</span>
                        <span class="badge bg-success">
                            <a href="https://cs.wikipedia.org/?curid={$article->getWikiId()}" target="_blank">
                                Odkaz na zdroj
                            </a>
                        </span>
                    </p>
                </div>
                <div class="col-12 col-md-3">
                    <div class="img-thumbnail">
                        <a href="{$article->getPicture()}" data-fslightbox target="_blank" rel="noopener noreferrer">
                            <img src="{$article->getPicture()}" class="card-img-top rounded" alt="">
                        </a>
                        <div class="text-muted text-center mt-1">
                            <small>Obrázek: {$article->getSourceHeading()}</small>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <h2 class="cover-heading">Související články</h2>
    <div class="row m-0 mb-3 p-0">
        {foreach $relatedArticles as $relatedArticle}
            {include 'parts/articleSmall.latte', article => $relatedArticle}
        {/foreach}
    </div>
{/block}
```

Na obrázku je vidět jedna ze šablon, kontrétně detail článku. Zde je v horní části kód, který vypisuje detail článku. Jmenovitě vypíše do tagu H1 nadpis a do tagu p text vygenerovaný umělou inteligencí (filter |breaklines zajistí, že se odřádkování v textu vypíše jako tag <br >). Pod textem článku se vypisuje prokliknutelná kategorie, a odkaz na původní článek (zdroj).

V další části (která je vidět na větším rozlišení vpravo) je pak obrázek z wikipedie, který lze rozkliknout přes javascriptovou metodu fslightbox. Pod obrázkem je za textem “Obrázek: ” zobrazen původní nadpis z wikipedie.

Jako poslední se pod článkem vykreslují související články. Vykreslují se pomocí podšablony parts/articleSmall.latte, která je funkcionálně stejná jako stávající šablona.

# Závěr

Tato práce si klade za cíl ověřit možnosti automatického generování článků pomocí programovacího jazyka PHP a rozhraní API modelu ChatGPT. Výsledkem je funkční aplikace, která dokáže na základě jednoduchého vstupu (nadpis a text z wikipedie) vygenerovat unikátní, srozumitelný, stylisticky ucelený text. Hlavním přínosem práce je praktické propojení moderní technologie umělé inteligence s běžně používanými webovými nástroji. Ukázal jsem, že i pomocí relativně jednoduchého řešení lze automatizovat proces tvorby textového obsahu, což může být užitečné například v oblasti blogování, správy webových stránek, nebo při návrhu tzv. "virtuální redakce".

Můj projekt zároveň slouží jako základ pro další rozvoj — lze ho snadno rozšířit o pokročilejší funkce, jako je výběr stylu psaní, automatická kontrola faktické správnosti, nebo generování doprovodných obrázků. Tato práce mi umožnila lépe pochopit možnosti a limity současné umělé inteligence, prohloubit znalosti v oblasti API integrací a naučit se pracovat s technologiemi, které mají potenciál zásadně ovlivnit budoucnost digitálního obsahu.

# Zdroje dat, citace


[1] PHP - Wikipedie. Online. Wikipedie. 2025. Dostupné z: https://cs.wikipedia.org/wiki/PHP. [cit. 2025-03-27].

[2] PHP - Stránka. Online. PHP. 2024. Dostupné z: https://www.php.net/. [cit. 2025-03-27].

[3] Nette - Framework. Online. Nette. 2024. Dostupné z: https://nette.org/cs/. [cit. 2025-03-27].

[4] Contributte. Online. Contributte. 2025. Dostupné z: https://contributte.org/. [cit. 2025-03-27].

[5] Nette - Routy. Online. Nette. 2025. Dostupné z: https://doc.nette.org/cs/application/routing. [cit. 2025-03-27].

[6] Doctrine - Entity. Online. Doctrine. 2025. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/3.3/reference/basic-mapping.html. [cit. 2025-03-27].

[7] Doctrine. Online. Doctrine. 2024. Dostupné z: https://www.doctrine-project.org/. [cit. 2025-03-27].

[8] Symfony - Commands. Online. Symfony. 2024. Dostupné z: https://symfony.com/doc/current/console.html. [cit. 2025-03-27].

[9] Nette - Utilities. Online. Nette. 2024. Dostupné z: https://doc.nette.org/en/utils. [cit. 2025-03-27].

[10] Latte. Online. Latte. 2025. Dostupné z: https://latte.nette.org/cs/. [cit. 2025-03-27].

[11 ]Bootstrap. Online. Latte. 2025. Dostupné z: https://getbootstrap.com/docs/5.3/getting-started/introduction/. [cit. 2025-03-27].

[12] GIT. Online. GIT. 2024. Dostupné z: https://git-scm.com/. [cit. 2025-03-27].

[13] ChatGPT - API Playground. Online. ChatGPT. 2025. Dostupné z: https://platform.openai.com/docs/overview. [cit. 2025-03-28].

[14] ChatGPT - Webový Model. Online. ChatGPT. 2025 Dostupné z: https://chatgpt.com [cit. 2025-03-28].

[15] Docker. Online. Docker. 2025. Dostupné z: https://www.docker.com/. [cit. 2025-03-28].