



# WEB KE HŘE PIXEL DREAD

Maturitní práce

Autor	<b>Lukáš Pop</b>
Obor	<b>Informační technologie</b>
Vedoucí práce	<b>Ing. Tomáš Kazda, DiS</b>
Školní rok	<b>2024/2025</b>
Počet stran	<b>33</b>
Počet slov	<b>5380</b>

# Zadání maturitní práce

<b>Název</b>	<b>Předmět</b>
Web ke hře PixelDread	WEB

**Téma**  
Práce se zabývá vývojem webu s proprietárním redakčním systémem pro mobilní hru PixelDread. Web obsahuje blog, seznam aktualizací, přihlašovací systém, rozhraní pro správu obsahu, stránku s ochranou osobních údajů a FAQ (často kladené otázky).

**Použité prostředky**  
React, typescript, asp.net core, figma, docker

<b>Cíle práce</b>	
1	Vytvoření návrhu ve figmě
2	Vytvoření micro headless CMS
3	Vytvoření webového frontendu
4	Deployment docker image na VPS

<b>Osnova práce</b>	
2	Návrh endpointů
4	Nastavení přihlašovacího systému
5	Vytvoření administrativního rozhraní
6	Vývoj blogu a seznamu aktualizací
7	Vytvoření stránky na ochranu osobních údajů a FAQ
9	Spuštění kontejneru na hostingu
1	Katalog funkčních a nefunkčních požadavků
3	Návrh GUI
8	Tvorba Docker image



## Přihláška k maturitní práci

## Jméno a příjmení studenta

Pop, Lukáš

## Název práce

Web ke hře PixelDread

## Třída

P4A

## Školní rok

MP2024/25

## Přidělené role

## Vedoucí práce

## Oponent

Podpis

Kazda, Tomáš

Podpis

Stehlík, Michal

Obecná ustanovení	Vypracování a odevzdání práce proběhne v souladu s platnými normami (vyhláška 177/2009 Sb.) a aktuálním dokumentem "Pokyny k vypracování prací" vydaným školou.
	Práce bude hodnocena z hlediska jejího praktického využití, zvládnutí dokumentace po věcné i formální stránce a obhajoby celé práce. Student byl seznámen s kritérii hodnocení maturitní práce.
	Práce bude odevzdána ve dvou stejnopisech vázaných pevnou nebo kroužkovou vazbou.
	Veškeré náklady na MP včetně vyhotovení obou tištěných kopií si student hradí sám.
Licenční ujednání	Ve smyslu § 60 (Školní dílo) autorského zákona č. 121/2000 Sb. poskytují SPŠSE a VOŠ Liberec výhradní a neomezená práva k využití této mé maturitní práce.
	Bez svolení školy se zdržím jakéhokoliv komerčního využití mé práce.
	Pro výukové účely a prezentaci školy se vzdávám nároku na odměnu za užití díla.

## Finanční rozvaha - odhad celkových nákladů

V Kč	Náklady celkem	Hrazené školou
Výrobní	0	0
Na služby	0	0

Jedná se o MP, jejíž vypracování si škola vyžádala? Ano – Ne

## Podpis studenta (vyjadřuje souhlas s uvedenými údaji a ujednáními)

V Liberci 04.10.2024

Podpis

## Konzultant

Práci podporuji

Podpis

## Předmětová komise

Práci doporučuji

Podpis

## Třídní učitel

Práci doporučuji

Podpis

## Garant oboru

Práci doporučuji

Podpis

## Ředitel školy

Práci doporučuji

Podpis

---

## Anotace

Práce se zabývá vývojem webu s proprietárním redakčním systémem pro mobilní hru PixelDread. Web obsahuje blog, seznam aktualizací, přihlašovací systém, rozhraní pro správu obsahu, stránku s ochranou osobních údajů a FAQ (často kladené otázky).

## Summary

The project focuses on the development of a website with a proprietary content management system for the mobile game PixelDread. The website includes a blog, update list, login system, content management interface, privacy policy page, and FAQ (frequently asked questions) section.

## Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní práci vypracoval sám a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne 14.03.2025

.....  
Lukáš Pop

# Obsah

Úvod.....	1
1 Postupy a technologie .....	2
1.1 Systémy pro správu obsahu (CMS) .....	2
1.1.1 Typy CMS.....	2
1.2 Frontend technologie .....	3
1.2.1 React.....	3
1.2.2 React-Router .....	3
1.2.3 React-Select.....	4
1.2.4 React-Helmet-Async.....	4
1.2.5 TypeScript.....	4
1.2.6 Quill .....	4
1.2.7 DOMPurify.....	4
1.3 Backend technologie.....	5
1.3.1 ASP.NET Core.....	5
1.3.2 Entity Framework Core .....	5
1.3.3 Architektura systému .....	5
1.3.4 Databáze SQLite .....	6
1.4 Docker.....	6
1.5 REST API.....	6
1.5.1 Principy REST .....	6
1.5.2 Struktura a standardizace endpointů.....	7
2 Implementace .....	8
2.1 Databázový model, entity a jejich vazby .....	8
2.1.1 ArticleText .....	8
2.1.2 ArticleLink.....	8
2.1.3 ArticleFAQ.....	8
2.1.4 ArticleMedia.....	8
2.2 Veřejné stránky a jejich využití.....	9
2.2.1 Blog.....	10
2.2.2 FAQ.....	10
2.2.3 Patch Notes .....	11
2.2.4 Stránka pro ochranu osobních údajů .....	11

---

2.3	Administrativní rozhraní .....	11
2.3.1	Správa Blogu .....	11
2.3.2	Správa FAQ a Patch Notes .....	12
3	Nasazení.....	14
3.1	Příprava aplikace pro nasazení.....	14
3.2	Vytvoření image.....	14
3.2.1	DockerFile pro React.....	14
3.2.2	DockerFile pro ASP.....	16
3.2.3	Docker Compose.....	17
3.3	Nasazení na VPS.....	18
3.4	Konfigurace DNS.....	19
	Závěr.....	20
	Seznam zkratk a odborných výrazů.....	21
	Seznam obrázků.....	22
	Použité zdroje .....	23
A.	Seznam příložených souborů .....	I

# Úvod

Tato práce se zaměřuje na vývoj webové stránky pro mobilní hru Pixel Dread s využitím malého headless CMS. Webová aplikace poskytuje hráčům aktuální informace o vývoji hry, zahrnuje blog, seznam aktualizací, FAQ (často kladené otázky) a administrativní rozhraní pro správu obsahu. Cílem je vytvořit stránku, která má snadno spravovatelný obsah, drží se jednoduchosti a pixelové tematiky hry a bude nasazena na doménu pixeldread.com.

Jedním z klíčových prvků tohoto webu je stránka s ochranou osobních údajů, která je nezbytná pro splnění požadavků Google Play. Tato stránka informuje uživatele o způsobu zpracování jejich osobních údajů a zajišťuje transparentnost v souladu s legislativními požadavky.

Web rovněž slouží jako marketingový nástroj, který návštěvníky motivuje ke stažení hry. Obsahuje prezentaci hry, pravidelné aktualizace a přehled funkcí, čímž podporuje budování aktivní komunity hráčů.

Technologie využité v projektu jsou: React, TypeScript, ASP.NET Core, Figma a Docker, což umožňuje efektivní vývoj, škálovatelnost a snadné nasazení na VPS. Primárním cílem je zajistit stabilní provoz webové aplikace na doméně pixeldread.com, kde bude hráčům poskytován aktuální obsah a možnost interakce s komunitou.

---

# 1 Postupy a technologie

## 1.1 Systémy pro správu obsahu (CMS)

Systémy pro správu obsahu (Content Management Systems – CMS) jsou softwarové aplikace, které jsou využívány pro tvorbu, správu a publikaci digitálního obsahu bez potřeby znalosti programování nebo pokročilých technologií. Tyto systémy jsou vhodné zejména pro webové stránky, jejichž obsah se často aktualizuje nebo mění, jako například blogy, e-shopy nebo informační stránky.

Hlavní myšlenkou CMS je oddělení obsahu od jeho vizuální prezentace. Tato vlastnost umožňuje provádění změn vzhledu webových stránek bez nutnosti zásahu do samotných dat. Další významnou výhodou je uživatelsky přívětivé administrační rozhraní, díky kterému je možné obsah snadno a efektivně spravovat. Důležitou funkcí CMS je rovněž možnost spolupráce více administrátorů zároveň, přičemž každý může mít nastavena různá práva pro přístup a úpravu obsahu. (1)

### 1.1.1 Typy CMS

#### 1.1.1.1 Tradiční CMS

Tradiční CMS, někdy označované jako monolitické systémy, jsou založené na principu, kdy je backendová část (tedy správa obsahu) přímo propojena s frontendem, který obsah zobrazuje uživatelům. V praxi to znamená, že správce obsahu vytváří nebo upravuje články a další prvky přímo v administračním rozhraní, odkud se poté obsah automaticky zobrazuje na webové stránce podle předpřipravených šablon.

Mezi hlavní výhody tradičních CMS patří jejich jednoduchost a přívětivost k uživatelům, kteří nemají technické znalosti. Administrátoři tedy mohou provádět úpravy a aktualizace obsahu jednoduše a rychle. Kromě toho existuje široká nabídka různých rozšíření nebo doplňků, kterými lze jednoduše přidávat nové funkce nebo změnit vzhled celého webu.

Jako nevýhoda je často uváděna nižší flexibilita, protože frontend a backend jsou pevně propojené, což může omezovat možnosti přizpůsobení se specifickým požadavkům nebo moderním technologiím. Navíc tento typ CMS může být méně efektivní v případech, kdy web obsahuje velké množství obsahu, což se projevuje například pomalejším načítáním.

Typickými příklady tradičních CMS jsou např. WordPress a Joomla (2)



### 1.1.1.2 Headless CMS

Headless CMS je označení modernějšího přístupu ke správě obsahu. Jeho hlavním principem je úplné oddělení backendové části systému od prezentace obsahu, která je řešená samostatně. Obsah je spravován prostřednictvím API, což umožňuje jej jednoduše distribuovat na libovolná zařízení či platformy, například webové stránky, mobilní aplikace nebo další digitální média.

Největší výhodou tohoto typu systému je možnost nezávisle zobrazovat obsah napříč různými platformami. Díky tomu je umožněna vysoká flexibilita, protože tvůrce obsahu není omezen jednou specifickou technologií nebo designem. Tento způsob práce také nabízí lepší výkon a snadnou škálovatelnost, protože systém může být optimalizován pro rychlejší načítání a efektivnější přenos dat. (3)

## 1.2 Frontend technologie

### 1.2.1 React

React je populární open-source JavaScriptová knihovna, která byla vytvořena společností Meta. Slouží primárně k vytváření interaktivních a dynamických uživatelských rozhraní. přičemž klade důraz na opakovanou použitelnost jednotlivých komponent. React umožňuje programátorům rozdělit uživatelské rozhraní na menší, izolované a snadno spravovatelné části nazývané komponenty. Tyto komponenty jsou nezávislé a lze je využívat opakovaně napříč aplikací, což usnadňuje vývoj a údržbu rozsáhlých projektů.

Jedním z klíčových prvků technologie React je tzv. virtual DOM (Document Object Model). DOM je standardizované rozhraní, které reprezentuje strukturu HTML dokumentů a umožňuje jejich dynamickou manipulaci pomocí JavaScriptu. Virtual DOM v Reactu je odlehčená kopie skutečného DOM, která existuje v paměti aplikace. React nejprve provede změny na tomto virtual DOMu a až poté porovná rozdíly s reálným DOMem. Tento přístup výrazně zvyšuje výkon, protože minimalizuje počet potřebných operací a změn v reálném DOMu, které jsou často časově náročné. (4)

### 1.2.2 React-Router

React-Router je standardní knihovna pro směrování (routing) v aplikacích postavených na technologii React. Směrování představuje proces, kdy se na základě URL adresy mění zobrazovaný obsah, aniž by bylo nutné znovu načítat celou stránku. To je klíčové pro tzv. single-page aplikace (SPA), kde veškerá navigace probíhá uvnitř jedné HTML stránky. (5)

---

### 1.2.3 React-Select

React-Select je populární knihovna pro vytváření rozšířených výběrových polí v aplikacích postavených na Reactu. Poskytuje široké možnosti přizpůsobení, jako je asynchronní načítání dat, podpora více výběrů, vyhledávání a vlastní stylování. Výhodou oproti nativnímu HTML `<select>` je lepší uživatelský zážitek a snadnější práce s dynamickými daty. (6)

### 1.2.4 React-Helmet-Async

Pro správu metadat v aplikaci postavené na Reactu se často využívá knihovna React-Helmet-Async. Tato knihovna umožňuje dynamickou úpravu `<title>`, `<meta>` tagů a dalších prvků v `<head>` sekci HTML dokumentu, což je zásadní pro SEO a sdílení obsahu na sociálních sítích. (7)

### 1.2.5 TypeScript

TypeScript je nadstavba JavaScriptu vyvinutá společností Microsoft, která zavádí do JavaScriptu statické typování a další moderní prvky známé z objektově orientovaných jazyků. Statické typování umožňuje vývojářům definovat datové typy proměnných, funkcí a objektů již při psaní kódu. To vede ke snazšímu odhalování chyb již ve fázi vývoje, před spuštěním aplikace. (8)

### 1.2.6 Quill

Quill je moderní open-source WYSIWYG editor, který umožňuje bohaté formátování textu, vkládání obrázků, odkazů a dalších prvků, avšak vše je to uloženo v HTML, což pro řešení obrázků není vhodné, proto jsou použity v projektu jen potřebné editační nástroje v nástrojové liště. (9)

### 1.2.7 DOMPurify

DOMPurify je bezpečnostní knihovna, která slouží k čištění HTML kódu před škodlivými skripty, čímž pomáhá zabránit XSS útokům. Při zpracování obsahu vkládaného uživatelem je nezbytné zajistit jeho dezinfekci, zejména pokud se používají WYSIWYG editory. (10)

## 1.3 Backend technologie

### 1.3.1 ASP.NET Core

ASP.NET Core je moderní open-source webový framework vyvinutý společností Microsoft, který slouží k tvorbě webových aplikací a REST API. Je oblíbený zejména díky multiplatformní podpoře, vysokému výkonu a rozšířeným možnostem zabezpečení. (11)

Důležitou součástí ASP.NET Core je knihovna ASP.NET Core Identity, která vývojářům usnadňuje správu uživatelů, autentizaci a autorizaci. Identity poskytuje komplexní systém pro správu uživatelských účtů včetně jejich registrace, přihlášení, správy hesel a zabezpečení dat uživatelů. Identity lze snadno integrovat do aplikací pomocí vestavěných funkcí a umožňuje flexibilní správu oprávnění uživatelů. (12)

V rámci autentizace je v ASP.NET Core Identity často využíván standard JWT (JSON Web Token). JWT umožňuje bezstavové ověření identity, kdy server vystaví klientovi unikátní token obsahující informace o uživateli a jeho oprávněních. Klient pak tento token používá v každém požadavku pro ověření své identity. Výhodou tohoto přístupu je zvýšení výkonu aplikace, protože není nutné uchovávat stavové informace o relacích přímo na serveru. (13)

Kombinace JWT a ASP.NET Core Identity představuje robustní a škálovatelné řešení autentizace a autorizace, které je vhodné pro zabezpečení moderních webových aplikací.

### 1.3.2 Entity Framework Core

Entity Framework Core je Object-Relational Mapping (ORM) framework vyvinutý společností Microsoft, který slouží k jednodušší správě databází. Tento framework umožňuje vývojářům pracovat s databázemi prostřednictvím objektově orientovaného přístupu, aniž by bylo nutné psát ručně SQL dotazy. (14)

### 1.3.3 Architektura systému

Projekt je založen na dvouvrstvé architektuře, kde frontend a backend fungují jako oddělené komponenty, které spolu komunikují prostřednictvím REST API. Frontend je vytvořen pomocí Reactu s TypeScriptem a pro komunikaci používají Axios instance, které volají http požadavky. Backend je postaven na ASP.NET CORE a využívá Entity Framework Core pro práci se SQLite databází.

---

### 1.3.4 Databáze SQLite

SQLite je open-source databázový systém založený na relačním modelu dat. Jedná se o lehkou, souborově orientovanou databázi, která nevyžaduje samostatný databázový server ani náročnou konfiguraci. Data jsou v případě SQLite uchovávána v jednom souboru, což umožňuje jednoduchou správu a snadnou přenositelnost databáze mezi různými prostředími či zařízeními. (15)

Jednou z hlavních předností databáze SQLite je její nízká náročnost na systémové zdroje. Díky tomu je vhodná zejména pro aplikace, které nepotřebují obsluhovat velké množství uživatelů současně, například menší webové aplikace, mobilní aplikace nebo vestavěné systémy. (15)

## 1.4 Docker

Docker open-source software, který má za cíl poskytnout izolaci jednotlivých částí či celých aplikací do kontejnerů, čímž odebere z aplikace nepotřebné soubory a virtualizovaný operační systém. Tímto procesem se sníží velikost a náklady na provoz. Nevýhodou však je vázanost na hostitelský operační systém, který je využíván pro běh aplikace. (16)

## 1.5 REST API

Representational State Transfer (REST) je architektonický styl určený pro návrh webových služeb, který umožňuje komunikaci mezi klientem a serverem pomocí standardních HTTP požadavků. V tomto projektu backend poskytuje RESTful služby, se kterými komunikuje Reactový frontend. Tento způsob komunikace umožňuje spojení obou částí projektu (frontendu a backendu). (17)

### 1.5.1 Principy REST

REST (Representational State Transfer) je architektonický styl pro návrh webových služeb, který definuje pravidla komunikace mezi klientem a serverem. REST API je navrženo tak, aby bylo jednoduché, škálovatelné a efektivní, přičemž se řídí šesti základními principy.

- Klient-server – Klient a server jsou oddělené entity, které spolu komunikují prostřednictvím API. Tím je zajištěna modularita systému a možnost nezávislého vývoje obou částí.
- Bezstavovost – Každý požadavek musí obsahovat všechny potřebné informace, protože server mezi požadavky neuchovává žádný stav. Tento přístup zjednodušuje škálování aplikací.

- Mezipaměť (Cacheability) – Umožňuje ukládání odpovědí do mezipaměti (cache), což snižuje zatížení serveru a zrychluje odezvu API.
- Jednotné rozhraní (Uniform Interface) – API využívá standardizované HTTP metody (GET, POST, PUT, DELETE) a data vrací v jednotném formátu, například JSON.
- Vícevrstvá architektura – Oddělení systému do více vrstev umožňuje například lepší správu zabezpečení a logiky aplikace.
- Kód na požádání – API může na požádání poskytnout klientovi kód, například skripty pro zpracování dat, i když tento princip není povinný. (18)

### 1.5.2 Struktura a standardizace endpointů

Správně navržené REST API endpointy zajišťují konzistenci, přehlednost a efektivní komunikaci mezi klientem a serverem. Klíčem k dobré architektuře je jasná struktura URL, správné použití HTTP metod a jednotný návratový formát). (19)

Endpointy by měly být srozumitelné. Například místo `/api/getArticles` se správně používá `/api/articles`, protože operaci určuje HTTP metoda. Vztahy mezi entitami se pak vyjadřují hierarchicky, například `/api/posts/{postId}/articles` vrací články konkrétního příspěvku, zatímco `/api/users/{userId}/posts` zobrazí příspěvky daného uživatele. (20)

Pro komunikaci mezi klientem a serverem se využívají standardní HTTP metody. GET slouží k získání dat, POST k vytvoření nových záznamů, PUT k jejich aktualizaci a DELETE k odstranění. Díky tomu není nutné do URL přidávat slova jako „create“ nebo „delete“, což vede k přehlednější struktuře. (21)

---

## 2 Implementace

### 2.1 Databázový model, entity a jejich vazby

Struktura databáze je vytvořena z několika tabulek, které mají mezi sebou různé vazby a každá má různý význam pro projekt. Hlavní entitou je zde tabulka Post složená ze článků Articles. Post funguje jako univerzální typ příspěvku, protože každý příspěvek má jen jednu kategorii, díky které lze příspěvky rozdělit na blogové, seznam aktualizací a často kladené otázky (FAQ). Další tabulkou je Tag spojený spojovací tabulkou PostTag, aby mohla vzniknout vazba N:M neboli každý Tag může být využitý v každém Postu. Další tabulkou jsou OG Data, která jsou nepovinná, protože jsou zde připravené pouze pro blogové příspěvky, které lze otevírat a z OG Data se vypisují meta data pro stránku.

Nejdůležitější vazbou, která z diagramu nemusí být snadno pochopitelná, je vazba mezi Post a Article, kdy je opět použita spojovací tabulka pod jménem PostArticle. V této spojovací tabulce je napsaný typ článku, jež je na příspěvek vázaný a pořadí, podle kterého se články budou řadit v příspěvku. Articles je abstraktní třída, která sice zařizuje vztah mezi entitami, avšak využívají se její podtřídy, kterými jsou již jednotlivé články obsahující potřebný obsah.

#### 2.1.1 ArticleText

Nejjednodušším typem je ArticleText, obsahující pouze string Content připravený na ukládání HTML v textovém řetězci (tzv. string).

#### 2.1.2 ArticleLink

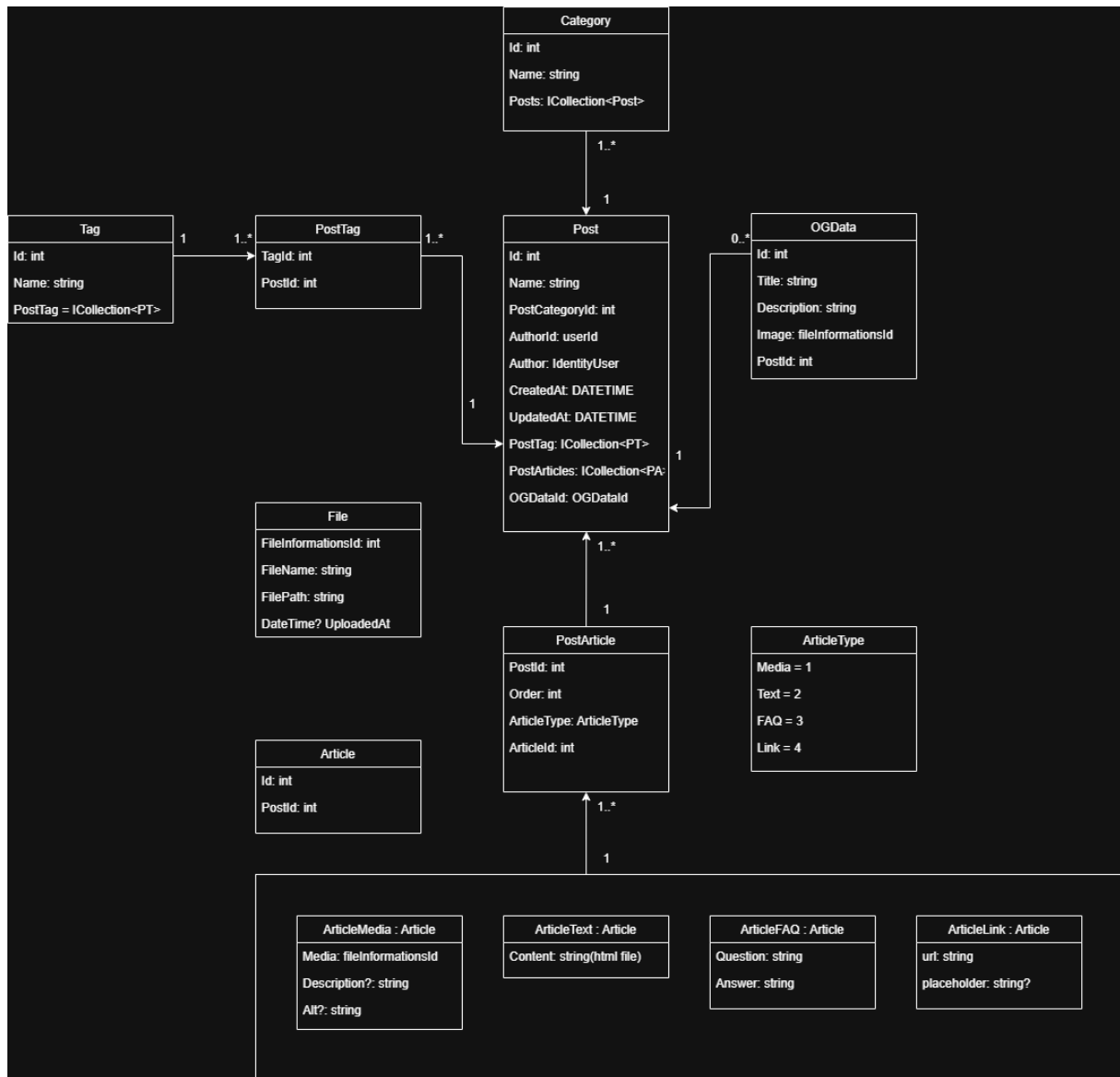
Druhým podobně fungujícím článkem je ArticleLink obsahující dvě stringové vlastnosti. Potřebnou URL adresu na stránku a nepovinný placeholder, pod kterým bude odkaz zobrazený.

#### 2.1.3 ArticleFAQ

Dalším typem je ArticleFAQ, vyrobený specificky kvůli sekci často kladených otázek, uchovává otázku a odpověď obojí typu string.

#### 2.1.4 ArticleMedia

Posledním typem článku je ArticleMedia, sloužící k navigaci na uložený obrázek. Není vhodné řešení ukládání obrázků za pomoci jejich rozložení do base64 stringu, kvůli pomalé rychlosti a nadbytečné zátěži při větších obrázcích. (22) Proto je zde vlastnost



## 2.2 Veřejné stránky a jejich využití

9

---

tak směřování je nasimulováno pomocí React-Routeru a provádí se změny v adrese URL podle stránky, na které se nachází návštěvník.

### 2.2.1 Blog

Tato stránka používá komponentu `PostList`, která má za úkol vypisovat příspěvky dle jejich kategorií pomocí připravené služby ve frontendové složce `api`. Konkrétně volá `axios` instanci `getPostsByCategory`, která provede HTTP request na GET a navrátí list příspěvků a jejich OG Data, pokud nějaké mají. Také díky kolekci `PostTag` lze pomocí funkce `map` získat Tagy od jednotlivých příspěvků. Blogy se vykreslují v 16:9 obdelníkové podobě v různém počtu sloupců dle zařízení (od jednoho na mobilních zařízeních po tři na počítači). Každý blogový příspěvek, který má v OG Datech uložené informace o obrázku využije tento obrázek jako náhledový, v případě absence obrázku je zde výplňový obrázek s textem „no image available“. Pod obrázkem se nachází název příspěvku, také jen v případě, že se v získaném objektu `Post` nějaký nachází, to samé platí pro Tagy, které se vypisují pod název. V `PostListu` se také nachází `React-Select` na filtr podle Tagů.

Jednotlivé příspěvky se při otevření získají pomocí služby `getPostBySlug`, protože každý blogový příspěvek má při své výrobě povinné vyplnění slugu. Slug je část URL adresy, kterou lze vidět za „/blog“ u každého blogového příspěvku. Slug je jedním z OG Dat, ostatní vlastnosti OG Dat se pomocí knihovny `React-Helmet-Async` vyplní Title, Description a Image v HTML hlavičce.

Při získání Postu pomocí Slugu se odešle další request `getArticlesByPostId`, který získá články dle `PostId` a zobrazí jejich obsah pomocí komponent, které jsou používány napříč různými obsahovými stránkami. Tyto komponenty jsou: `ArticleTextComponent`, `ArticleLinkComponent`, `ArticleFAQComponent` a `ArticleMediaComponent`. Každá z nich má nadefinovaný styl a vzhled, jak má vypadat. `ArticleTextComponet` před vypsáním HTML pomocí `dangerouslySetInnerHTML` použije funkci `sanitize` z knihovny `DOMPurify` pro bezpečnost před XSS útokem.

### 2.2.2 FAQ

Pro tuto stránku byl vytvořen jeden příspěvek, který používá stejné vypisování příspěvků jako blog po rozkliknutí konkrétního, avšak získá `Post` podle předdefinovaného `id`, které se nikdy nezmění, jelikož příspěvek nelze smazat. Používají se zde zmíněné komponenty pro vypisování otázek a odpovědí a zbylého obsahu, který si zde vytvoří administrátor. Stránka má za úkol odpovědět na obvyklé otázky ohledně mobilní hry.



### 2.2.3 Patch Notes

Patch Notes stránka (v překladu seznam aktualizací) slouží k oznámení nových změn, oprav či přidáných věcí. Tato stránka používá upravený PostList přidáním vstupního parametru HasDetails, čímž se určuje, jestli příspěvek půjde otevřít pro čtení příspěvků nebo ne. Pro tento případ je HasDetails nastavené na false a je každý příspěvek mapován a následně místo na náhled se vykreslí všechny články a případně název příspěvku komponentou ArticlesFromPost.

### 2.2.4 Stránka pro ochranu osobních údajů

Stránka pro ochranu osobních údajů je nezbytným požadavkem pro publikaci mobilní hry na Google Play. Tato stránka informuje uživatele o způsobu zpracování jejich osobních údajů, přičemž splňuje legislativní požadavky.

## 2.3 Administrativní rozhraní

Do administrativního prostředí se lze dostat pomocí URL adresy „/login“, nebo při pokusu dostání se přímo na adminské stránky „/admin/\*“. Při načítání se spustí api volání požadavku GET Admin/Me, která určuje, zda je uživatel přihlášený či nikoliv. Pokud není, zobrazí se stránka s textem „Access Denied“ a tlačítkem, které odkazuje na přihlašovací formulář. Využívám zde modaly na veškeré akce či potvrzení a oznámení, protože dle mého osobního názoru není vhodné užívat alert funkci a mít rozdílně udělané oznámení apod.

### 2.3.1 Správa Blogu

Nachází se zde tlačítko pro správu Tagů, které otevře modal, ve kterém lze vytvořit nový Tag, editovat jejich názvy či je mazat. Při přejmenování či vytvoření Tagu se kontroluje, zda už tento Tag se mezi nimi nenachází a blokuje tak tvorbu duplicitních Tagů.

Další možnou akcí na této stránce je samotné vytváření obsahu, kdy se při kliknutí na tlačítko Create post otevře modal se dvěma kroky obsahující inputy pro vyplnění údajů o příspěvku. Těmito údaji jsou: jméno, povinný slug a tlačítko pro přidání OG Dat, které otevře další modal. Dále je zde Creaable React-Select, ve kterém je možnost přímo vyrábět nové Tagy či zvolit již existující a přidat je tak do příspěvku. Toto byl obsah pro první krok. Ve druhém kroku se nachází možnosti pro výběr, jaký typ článku chce administrátor vytvořit. U blogu to jsou: text, odkaz a media články. Při výrobě textového článku se otevře Quill editor, ve kterém lze snadno psát článek. Při výrobě odkazu se ukážou dva inputy, jeden pro URL adresu a druhý pro placeholder, který se pak bude

---

zobrazovat na místě odkazu a funguje jako živý odkaz. Dalším typem je Media, kde se nahrává obrázek a lze k němu vyplnit input pro jeho popis a alt vlastnost, která je v tagu `<img>`. Poslední typ zde není nabízený, avšak objeví se v následující správcovské stránce. Jde o typ FAQ, který má jako odkaz také dva inputy. První je pro otázku a druhý je pro její odpověď.

Po vytvoření minimálně jednoho článku se článek zobrazí v preview, kde lze každému článku měnit pořadí díky nastavenému drag and dropu, nebo článek ještě změnit či smazat. Po určení pořadí, jak jdou články za sebou, lze příspěvek vytvořit a odeslat tak api požadavek na výrobu postu, kde se využívají DTOs (data transfer objects) první, jménem PostDTO, jehož vlastnosti odpovídají všemu, co je potřeba mít před rozdělením dat do tabulek. Neboli odešle CategoryId (přiřazené dle toho na jaké stránce jsme např. nyní jde o blog) kolekci vybraných TagIds, kolekci ArticleDTO (článků), která má nastavené všechny specifické údaje obsahu článků za nepovinné (např. Content, URL a Otázka), čímž umožní tento objekt používat všem typům článků. Pouze vlastnosti ArticleType (typ příspěvku) a Order (pořadí) jsou povinné. Poslední nepovinnou částí PostDTO je OGDataDTO obsahující pouze informace o OG Datech. Tato data se v controlleru zpracují a vytvoří se Post entita, Articles a všechny spojovací tabulky pro spojení. To samé platí pro užití tagy.

Další věci jsou zde vypsané příspěvky opět dle kategorií a nabízí se zde administrátorovi možnost zobrazit příspěvek, čímž jej odkáže do otevřeného příspěvku na stránce blogu. Další je možnost smazání, která otevře ověřovací modal a ověří, zda akce byla chtěná a potvrdí tak odstranění příspěvku, kterým se díky cascade delete odstraní i spojovací tabulky a články tohoto příspěvku. Poslední a nejzajímavější možností je možnost úpravy, která administrátora odkáže na novou stránku určenou pro editace. Tato stránka umožňuje upravit veškerý obsah příspěvku či pomocí drag and drop funkce a následného uložení prohazování jejich pořadí. V blogových příspěvcích lze i měnit OG Data, čímž lze změnit náhledový obrázek blogu.

### 2.3.1.1 Nahrávání do složky Uploads

Jelikož projekt vyžaduje místo, kam ukládat efektivně data souborů, konkrétně obrázků, tak prostřednictvím FileControlleru a HTTP požadavku na POST zpracovává soubor a následně jej ukládá do složky Uploads. V databázi je tak uložen pouze záznam o FileInformations, které obsahují jméno souboru, velikost, datum výroby a cestu k souboru. Z Reactu se tento soubor předává pomocí FormData a axios.post.

### 2.3.2 Správa FAQ a Patch Notes

Tyto dvě následující stránky mají stejný princip jako tvorba blogových příspěvků, jen mají méně možností, které lze při výrobě udělat. FAQ ani Patch Notes nemají

nastavitelná OG Data, jelikož se jejich obsah na prezentující stránce zobrazuje jen na jedné stránce díky komponentě ArticlesFromPost. Správa FAQ je obzvlášť jednoduchá, jelikož funguje jako úprava jednoho příspěvku, protože FAQ je pouze jeden příspěvek. Pokud neexistuje FAQ příspěvek, systém zobrazí tlačítko pro jeho výrobu. Výroba obnáší stejný proces, co u blogového příspěvku, jen se místo CategoryId pro blog, odešle CategoryId pro FAQ a nejsou zde žádné tagy ani název příspěvku. FAQ články neumožňují vkládání obrázků. U Patch Notes to funguje naprosto stejně jako u tvoření a úpravy blogových příspěvků, jen zde nejsou tagy a není možné si zobrazit konkrétní příspěvek Patche, protože jsou již vypsané celé příspěvky pod sebou i s jejich články.

---

## 3 Nasazení

Aby tento projekt naplnil svůj účel, musí tato aplikace být nasazena a zpřístupněna uživatelům. Pro tento účel lze využít Docker, který pomocí kontejnerizace připraví aplikaci do produkčního prostředí. Díky své modularitě jde aplikace spustit s minimálními úpravami konfigurace, což proces nasazení usnadní.

### 3.1 Příprava aplikace pro nasazení

Před samotným nasazením aplikace je nutné zkontrolovat a otestovat komunikaci mezi frontendem a API, ověřit funkčnost databáze a prověřit zpracování HTTP požadavků. Následně je důležité provést důkladnou kontrolu chyb, odstranit nadbytečné importy a nevyužité funkce. Pro simulaci běhu aplikace byl použit příkaz `npm run build`, který v konzoli zobrazoval případné upozornění na chyby a optimalizační doporučení. (23)

### 3.2 Vytvoření image

Každá část aplikace má vlastní Dockerfile, který definuje způsob sestavení kontejneru. Pro správu více služeb a jejich komunikaci se používá `docker-compose.yml`, který zajišťuje společné spuštění všech potřebných kontejnerů. (16)

#### 3.2.1 DockerFile pro React

Frontendová část aplikace je nasazena v kontejneru a spouštěna pomocí Nginxu. Následující Dockerfile definuje postup sestavení a spuštění React aplikace:

```
# Sestavení React aplikace
FROM node:18-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
```

```
# Nastavení Nginxu pro produkční prostředí
FROM nginx:stable-alpine AS production
```

```

WORKDIR /usr/share/nginx/html
RUN rm -rf ./*
COPY --from=build /app/dist ./
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

React je SPA, což znamená, že většina zpracování probíhá na straně klienta. Aby bylo zajištěno správné načítání stránek, je nutné upravit konfiguraci Nginxu:

```

server {
    listen 80;
    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri /index.html;
    }

    # Statické soubory - lepší kešování
    location ~*
\.(?:ico|css|js|gif|jpe?g|png|woff2?|eot|ttf|svg|map)$ {
        expires 6M;
        access_log off;
        add_header Cache-Control "public, max-age=15778463";
    }

    # Proxy pro API
    location /api {
        proxy_pass http://api:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

```

```
# Gzip komprese pro lepší výkon
gzip on;
gzip_types    text/plain    text/css    application/json
application/javascript    text/xml    application/xml
application/xml+rss text/javascript;
gzip_vary on;
}
```

Kešování statických souborů bylo nastaveno na 6 měsíců, což zrychluje načítání aplikace při opětovných návštěvách uživatelů. (23)

### 3.2.2 DockerFile pro ASP

```
# Base image pro runtime aplikace
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 8080

# Build image
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["PixelDread.csproj", "."]
RUN dotnet restore "./PixelDread.csproj"
COPY . .
RUN dotnet build "./PixelDread.csproj" -c Release -o
/app/build

# Publikování aplikace
FROM build AS publish
RUN dotnet publish "./PixelDread.csproj" -c Release -o
/app/publish /p:UseAppHost=false

# Finální image
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
```

```
ENTRYPOINT ["dotnet", "PixelDread.dll"]
```

Pro ukládání databázového souboru jsme zvolili volume db-data, protože se nám to hodí v případě, kdybychom vytvářeli novou verzi nebo by nám aplikace spadla.

### 3.2.3 Docker Compose

Aby bylo možné efektivně spravovat více kontejnerů současně, je využít Docker Compose, což je nástroj umožňující definovat a spravovat více kontejnerů v jednom souboru pomocí docker-compose.yml. Tento přístup zjednodušuje nasazení, protože všechny služby jsou nakonfigurovány na jednom místě a mohou být spuštěny jediným příkazem, níže je uvedena ukázka souboru docker-compose.yml, který je využitý v projektu. (24)

```
version: '3.8'

services:
  api:
    build:
      context: ./Backend/PixelDread
      dockerfile: Dockerfile
    networks:
      - app-network
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - ASPNETCORE_FORWARDEDHEADERS_ENABLED=true
      - ConnectionStrings__DefaultConnection=Data
        Source=/app/data/pixeldread.db
    volumes:
      - db-data:/app/data

  frontend:
    build:
      context: ./Frontend
      dockerfile: Dockerfile
    ports:
      - "3000:80"
```

```
networks:
  - app-network
depends_on:
  - api

networks:
  app-network:
    driver: bridge

volumes:
  db-data:
```

Mezi kontejnery je vytvořena privátní síť, která zajišťuje bezpečnou komunikaci mezi službami bez nutnosti vystavovat backend přímo na veřejné síti, což zvyšuje bezpečnost. Kromě toho je databázový soubor uložen v persistentním svazku (volume db-data), což zajišťuje jeho zachování i po restartu kontejnerů. (16)

Spuštění všech služeb je následně provedeno pomocí příkazu:

```
docker-compose up -d --build
```

### 3.3 Nasazení na VPS

Po vytvoření a otestování kontejnerů bylo nutné aplikaci nasadit do produkčního prostředí na VPS. Proces nasazení zahrnoval několik kroků, které zajistily správné přenesení projektu a jeho spuštění v produkčním prostředí.

Nejprve byl vytvořen archiv projektu, aby se zamezilo přenosu zbytečných souborů a zrychlil se proces nahrávání na server:

```
tar --exclude=node_modules --exclude=bin --exclude=obj --  
exclude=.git --exclude=dist --exclude=.vscode -czf  
projekt.tar.gz .
```

Archivovaný projekt byl následně přenesen na VPS pomocí protokolu SCP:

```
scp projekt.tar.gz user@server:/opt/projekt/
```

Na serveru bylo nejprve nutné odstranit staré soubory, aby se předešlo konfliktům mezi novými a starými verzemi aplikace:



```
rm -rf /opt/projekt/*
```




Po vyčištění byla nová verze aplikace rozbalena do příslušného adresáře:

```
tar -xzf projekt.tar.gz -C /opt/projekt/
```

Po dokončení přenosu bylo nutné aplikaci spustit. Vzhledem k tomu, že byla nasazena pomocí Docker Compose, bylo možné spustit všechny potřebné kontejnery jediným příkazem, zmíněným v předešlé podkapitole.

### 3.4 Konfigurace DNS

Aby byla aplikace přístupná pod svou doménou, musely být nakonfigurovány záznamy DNS. Tato konfigurace probíhala prostředím Cloudflare, kde se záznamy jednoduše vyplňovaly do tabulky. Záznamy a vzhled této tabulky jsou znázorněné v příloženém obrázku.

<input type="checkbox"/>	Type ▲	Name	Content	Proxy status	TTL	Actions
<input type="checkbox"/>	A	pixeldrea...	194.163.191....		Auto	<a href="#">Edit ▶</a>
<input type="checkbox"/>	AAAA	pixeldrea...	2a02:c206:22...		Auto	<a href="#">Edit ▶</a>
<input type="checkbox"/>	CNAME	www	pixeldread.c...		Auto	<a href="#">Edit ▶</a>
<input type="checkbox"/>	TXT	pixeldrea...	"google-site...	DNS only	5 min	<a href="#">Edit ▶</a>

Obrázek 2 – Cloudflare tabulka záznamů

Záznam typu A slouží k nasměrování domény na IPv4 adresu serveru a záznam AAAA je použitý pro IPv6 adresu serveru. Dalším záznamem je CNAME, který se používá pro přesměrování uživatele ze subdomény www na hlavní doménu neboli je stránka přístupná pod oběma názvy, jak s www, tak i bez něj. (25)

---

## Závěr

Vytvoření a publikace zmiňované webové aplikace proběhla úspěšně, projekt naplnil své cíle a má prostor i na případné vylepšení či přidání nových funkcí. Největším problémem byla slepá cesta, která byla ve špatném rozhodnutí a tvrdohlavém vytvoření aplikace se špatně rozšiřitelným a fungujícím databázovým modelem, čímž se uprostřed vývoje práce musela vytvářet od úplných základů znovu. Dalším problémem bylo nasazení na VPS, jelikož REST api nechtělo komunikovat s frontendem. Avšak i přes náročnou cestu byly chyby vyřešeny a práce úspěšně dokončena.

## Seznam zkratek a odborných výrazů

**CMS** – Content Management System (Systém pro správu obsahu)

**FAQ** – Frequently Asked Questions (Často kladené otázky)

**API** – Application Programming Interface (Rozhraní pro programování aplikací)

**VPS** – Virtual Private Server (Virtuální privátní server)

**SPA** – Single-Page Application (Jednostránková aplikace)

**JWT** – JSON Web Token (Standard pro bezpečnou výměnu informací)

**ORM** – Object-Relational Mapping (Mapování objektově-relačních databází)

**SEO** – Search Engine Optimization (Optimalizace pro vyhledávače)

**URL** – Uniform Resource Locator (Jednotný lokátor zdrojů)

**OG Data** – Open Graph Data (Metadata pro sociální sítě)

**DTO** – Data Transfer Object (Objekt pro přenos dat)

**HTML** – HyperText Markup Language (Značkovací jazyk pro tvorbu webových stránek)

**CSS** – Cascading Style Sheets (Kaskádové styly)

**XSS** – Cross-Site Scripting (Bezpečnostní zranitelnost webových aplikací)

**HTTP** – HyperText Transfer Protocol (Protokol pro přenos hypertextu)

**DNS** – Domain Name System (Systém doménových jmen)

**CDN** – Content Delivery Network (Sít' pro doručování obsahu)

**JSON** – JavaScript Object Notation (Formát pro výměnu dat)

**SCP** – Secure Copy Protocol (Bezpečný protokol pro přenos souborů)

**ASP.NET** – Active Server Pages .NET (Webový framework od Microsoftu)

**Nginx** – Webový server a reverzní proxy

**SQLite** – Lightweight SQL Database (Lehká relační databáze)

**DOM** – Document Object Model (Objektový model dokumentu)

---

## Seznam obrázků

Obrázek 1–Databázový model.....	9
Obrázek 2 – Cloudflare tabulka záznamů.....	19

## Použité zdroje

1. **Wikipedia contributors.** Content management system. *wikipedia.org*. [Online] Wikipedia contributors, 7. 2 2025. [Citace: 23. 2 2025.] [https://en.wikipedia.org/w/index.php?title=Content\\_management\\_system&oldid=1274533486](https://en.wikipedia.org/w/index.php?title=Content_management_system&oldid=1274533486).
2. **Wikipedia Contributors.** Content Management System. *Wikipedia, The Free Encyclopedia*. [Online] 2025. [Citace: 1. 3 2025.] [https://en.wikipedia.org/wiki/Content\\_management\\_system](https://en.wikipedia.org/wiki/Content_management_system).
3. **FourCrows.** Co je to Headless CMS? [Online] 2023 . [Citace: 1. 3 2025.] <https://www.fourcrows.cz/slovník-pojmu/headless-cms/>.
4. **Meta Platforms, Inc.** React Documentation. *React.dev*. [Online] 2025. [Citace: 23. 2 2025.] <https://react.dev/>.
5. **REMIX SOFTWARE.** Getting Started with React Router. *React Router Documentation*. [Online] 2025. [Citace: 6. 3 2025.] <https://reactrouter.com/6.28.0/start/overview>.
6. **Watson, Jed.** React-Select – The Select Component for React. *React Select*. [Online] 2022. [Citace: 6. 3 2025.] <https://react-select.com/home#welcome>.
7. **NPM.** React Helmet Async – Lightweight Async SSR Helmet. *React Helmet Async*. [Online] 2018. [Citace: 7. 3 2025.]
8. **Microsoft Comporation.** TypeScript – JavaScript With Syntax for Types. *TypeScript*. [Online] 2025. [Citace: 6. 3 2025.] <https://www.typescriptlang.org/>.
9. **Quill.** Your powerful rich text editor. *QuillJS*. [Online] 2025. [Citace: 7. 3 2025.] <https://quilljs.com/docs/why-quill>.
10. **Cure53.** DOMPurify – Fast and secure XSS sanitizer. *Github*. [Online] 2025. [Citace: 7. 3 2025.] <https://github.com/cure53/DOMPurify>.
11. **Microsoft.** ASP.NET Core Documentation. *Microsoft Learn*. [Online] 2024. [Citace: 23. 2 2025.] <https://learn.microsoft.com/en-us/aspnet/core/>.
12. **Microsoft Comporation.** Introduction to Identity on ASP.NET Core. *Learn Microsoft*. [Online] 2024. [Citace: 6. 3 2025.] <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-9.0&tabs=visual-studio>.
13. **Auth0.** Introduction to JSON Web Tokens. *JWT Debugger*. [Online] 2025. [Citace: 6. 3 2025.] <https://jwt.io/introduction/>.
14. **Microsoft Corporation.** Microsoft Learn. *Entity Framework Core*. [Online] 2024. [Citace: 6. 3 2025.] <https://learn.microsoft.com/en-us/ef/core/>.
15. **SQLite.** About SQLite. *SQLite*. [Online] 2024 . [Citace: 6. 3 2025.] <https://www.sqlite.org/about.html>.

- 
16. **Docker, Inc.** Docker Compose Documentation. *Docker Docs*. [Online] 2025. [Citace: 1. 3 2025.] <https://docs.docker.com/compose/>.
  17. **Corporation, Microsoft.** Overview of REST API. *Microsoft Learn*. [Online] 2024. [Citace: 6. 3 2025.] <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>.
  18. **Fielding, Roy.** Architectural Styles and the Design of Network-based Software Architectures. *University of California*. [Online] 2000. [Citace: 6. 3 2025.] [https://ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm).
  19. **Microsoft Corporation.** Best practices for REST APIs. *Microsoft Learn*. [Online] 2024. [Citace: 6. 3 2025.] <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>.
  20. **Google.** API design guide. *Google Cloud*. [Online] 2025. [Citace: 6. 3 2025.] <https://cloud.google.com/apis/design>.
  21. **Mozilla Developer Network.** HTTP request methods. *Mozilla developer*. [Online] 2025. [Citace: 6. 3 2025.] <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>.
  22. **Why "optimizing" your images with Base64 is almost always a bad idea.** *Bunny*. [Online] 2019. [Citace: 6. 3 2025.] <https://bunny.net/blog/why-optimizing-your-images-with-base64-is-almost-always-a-bad-idea/#:~:text=Due%20to%20how%20Base64%20works,as%20increases%20your%20bandwidth%20bill..>
  23. **Nginx, Inc.** Nginx Documentation. *Nginx Docs*. [Online] 2025. [Citace: 1. 3 2025.] <https://nginx.org/en/docs/>.
  24. **Docker, Inc.** Docker Compose Documentation. *dockerdocs*. [Online] 2025. [Citace: 1. 3 2025.] <https://docs.docker.com/compose/>.
  25. **DigitalOcean.** How to Deploy Applications on a VPS. *DigitalOcean Docs*. [Online] 2025. [Citace: 1. 3 2025.] <https://www.digitalocean.com/docs/>.
  26. **Contentful.** What is Headless CMS? *Contentful*. [Online] 2024. [Citace: 23. 2 2025.] <https://www.contentful.com/headless-cms/>.
  27. **Stack Overflow.** What is the difference between npm install and npm run build? *Stack Overflow*. [Online] 2025. [Citace: 2. 3 2025.] <https://stackoverflow.com/questions/43664200/what-is-the-difference-between-npm-install-and-npm-run-build>.
  28. **TheServerSide.** Dockerfile vs docker-compose: What's the difference? *TheServerSide.com*. [Online] 2025. [Citace: 2. 3 2025.]
  29. **OpenJS Foundation.** npm Documentation - CLI Commands. *npm Docs*. [Online] 2025. [Citace: ] <https://docs.npmjs.com/cli/v9/commands/npm>.
  30. **Figma, Inc.** What is Figma? *Figma Help Center*. [Online] 2025. [Citace: 1. 3 2025.] <https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma>.

31. Microsoft. TypeScript Handbook. *TypeScriptLang.org*. [Online] 2024. [Citace: 23. 2 2025.] <https://www.typescriptlang.org/docs/>.
32. Microsoft. Docker and ASP.NET Core. *learn.microsoft*. [Online] 2025. [Citace: ] <https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/?view=aspnetcore-9.0>.





## A. Seznam příložených souborů

- **MP2025-Pop-Lukas-Web-kre-hre-Pixel-Dread.docx** – editovatelná verze dokumentace maturitní práce
- **MP2025-Pop-Lukas-Web-kre-hre-Pixel-Dread.pdf** – tisknutelná verze dokumentace maturitní práce
- **Aplikace** – zdrojové kódy
- **Návrh ve Figmě** – Pop-Lukas-Pixel-Dread-Navrh.fig

Soubory jsou k dispozici v internal repozitáři organizace pslib-cz, který naleznete prostřednictvím přiloženého QR kódu:



[https://github.com/pslib-cz/MP2024-25\\_Pop-Lukas\\_Web-ke-hre-PixelDread](https://github.com/pslib-cz/MP2024-25_Pop-Lukas_Web-ke-hre-PixelDread)