Webový správce hesel s možností rozdělení do skupin a rolí

Jindřich Caletka

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jindřich Caletka**
Osobní číslo:	**A21527**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Webový správce hesel s možností rozdělení do skupin a rolí**
Téma práce anglicky:	**A Web-Based Password Manager with the Ability to Divide into Groups and Roles**

Zásady pro vypracování

1. Nastudujte a rozveďte problematiku v kontextu tématu práce.
2. Navrhněte webovou aplikaci pro správu hesel, která bude umožňovat skupinové sdílení.
3. Návrh realizujte za využití tokenů.
4. Zvolte vhodné prostředky a technologie pro implementaci.
5. Implementujte vlastní řešení a otestujte.
6. Výsledky vhodně prezentujte a popište.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SULLIVAN, Bryan a Vincent LIU. *Web Application Security, A Beginner's Guide*. McGraw Hill, 2011. ISBN 978--0071776165.
2. BURNETT, Mark. *Perfect Password: Selection, Protection, Authentication*. Syngress, 2005. ISBN 978-1597490412.
3. PINE, David. *Learning Blazor: Build Single-Page Apps with WebAssembly and C#*. O'Reilly Media, 2022. ISBN 978--1098113247.
4. SMITH, Jon. *Entity Framework Core in Action, Second Edition*. 2nd edition. Manning, 2021. ISBN 978-1617298363.
5. PEYROTT, Sebastián. *The JWT Handbook* [online]. Auth0, 2018 [cit. 2023-11-11]. Dostupné z: https://auth0.com/resources/ebooks/jwt-handbook

Vedoucí bakalářské práce: **Ing. Petr Žáček, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor;
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

Vývoj správce hesel jako webové aplikace je tím, čím se tato bakalářská práce zabývá. Principy autentizace a ověřování identity byly zkoumány v teoretické části, včetně zkoumání důležitosti silných hesel, dvoufaktorové autentizace a bezpečnostních aktualizací. Byla popsána odolná, avšak zapamatovatelná hesla spolu s technikami jejich vytváření, jako je například hashování a solení.

Byla vysvětlena potřeba správců hesel na základě limitací tradičního přístupu k heslům. Bylo definováno, co je správce hesel, jeho vývoj, bezpečnostní aspekty a různé typy. Závěr byl zaměřen na budoucnost správy hesel s využití AI, blockchainu, kvantových výpočtů, důležitost kybernetické bezpečnosti a nové technologie jako Passkeys.

V praktické části bylo popisováno plánování vývoje aplikace, včetně výběru technologií. Byly probírány i přípravné kroky. Dále byly zdůrazněny implementační detaily jako propojení s databází, vývoj frontendu a backendu v ASP.NET Core Blazor, rovněž správa repozitářů a služeb. Nakonec bylo zmíněno nasazení, konfigurace a budoucí údržba aplikace.

Klíčová slova: Blazor, ASP.NET Core, Správce hesel, Webová aplikace

ABSTRACT

The development of a password manager as a web application is the focus of this bachelor thesis. The principles of authentication and identity verification were examined in the theoretical part, including an exploration of the importance of strong passwords, two-factor

authentication, and security updates. Resilient yet memorable passwords were described along with techniques for their creation, such as hashing and salting.

The need for password managers was explained based on the limitations of the traditional approach to passwords. The role of a password manager, its development, security aspects, and various types were defined. The conclusion was focused on the future of password management utilizing AI, blockchain, quantum computing, the importance of cybersecurity, and new technologies like Passkeys.

In the practical part, the planning of application development, including the selection of technologies, was described. Preparatory steps were also discussed. Furthermore, implementation details such as database integration, frontend and backend development in ASP.NET Core Blazor, as well as repository and service management, were emphasized. Deployment, configuration, and future maintenance of the application were mentioned in the end.

Keywords: Blazor, ASP.NET Core, Password manager, Web application

Chtěl bych poděkovat panu Ing. Petru Žáčkovi, PhD. za podporu a znalosti, které mi daly jasnou představu, jak by měla moje bakalářské práce vypadat.

OBSAH

ÚVOD

V době, kdy digitální identita získává na stále větším významu, nám otázky bezpečnosti účtů a hesel připomínají, jak je důležité chránit naše digitální stopy. Existuje spousta aplikací, do kterých se denně registrují a přihlašují miliony lidí. Někteří pak mají několik účtů se stejnými přihlašovacími údaji, nebo používají externího poskytovatele identity, např. přihlášení pomocí Google účtu. Jistě, je to snadný způsob a člověk si nemusí pamatovat heslo, ale o to jednoduší je to pak pro člověka, který chce zneužít vašeho účtu.

Z toho důvodu se obecně doporučuje používat unikátní a složitá hesla. Složitá v tom smyslu, že jsou dlouhá, skládající se z malých a velkých písmen, čísel a speciálních znaků. Pamatovat si taková hesla může být složité a určitě nikoho nepotěší, když zadá své složité heslo a po kliknutí na přihlásit dostane hlášku, že zadané heslo není správné.

Jako řešení se nabízí správce hesel, který uživateli umožní vygenerovat náhodná a dlouhá hesla, které si do aplikace uloží a může si ho jednoduše zkopírovat nebo mu aplikace přímo navrhne, že sama vyplní přihlašovací formulář. Právě tohle je tématem téhle bakalářské práce s cílem takový správce hesel vytvořit. Další předností bude rozdělení do skupin, kde administrátoři budou moci přidávat uživatele a poskytovat jim sdílené přihlašovací údaje.

I. EORETICKÁ ČÁST

1 AUTENTIZACE

Autentizace je pojem, který označuje proces dokazování pravosti nějaké skutečnosti nebo dokumentu. V informatice je tento termín spojován s prokazováním identity uživatele. Obvykle uživatel prokazuje svou totožnost poskytnutím svých přihlašovacích údajů, tj. dohodnuté informace sdílené mezi uživatelem a systémem.

1.1 Přihlašovací údaje

Kombinace uživatelského jména a hesla je nejoblíbenějším mechanismem ověřování a je také známá jako ověřování heslem.

Známým příkladem je přístup k uživatelskému účtu na webových stránkách nebo u poskytovatele služeb, jako je Facebook nebo Gmail. Před přístupem k účtu je nutné prokázat, že uživatel vlastní správné přihlašovací údaje. Služby obvykle zobrazí obrazovku, která požaduje zadání uživatelského jména spolu s heslem. Poté porovnají údaje vložené uživatelem s hodnotami dříve uloženými v interním úložišti.

Po zadání platné kombinace těchto údajů, poskytovatel služby umožní pokračovat
a umožní přístup k uživatelskému účtu.

Zatímco uživatelské jméno může být veřejné, jako například e-mailová adresa, heslo musí být důvěrné. Kvůli své důvěrnosti musí být hesla chráněna před krádežemi ze strany kyberzločinců. Ačkoli jsou totiž uživatelská jména a hesla na internetu široce používána, jsou notoricky známá jako slabý bezpečnostní mechanismus, který hackeři pravidelně zneužívají. [1]

1.2 Faktory ověřování

Určitá kategorie pověření, jako je uživatelské jméno a heslo, se obvykle označuje jako autentizační faktor. I když je ověřování pomocí hesla nejznámějším typem ověřování, existují i další ověřovací faktory. Existují tři typy autentizačních faktorů, které se obvykle klasifikují následovně:

- Něco, co je vám známo, například heslo.

- Něco, co je vámi vlastněno, například chytrý telefon.
- Něco, co je ověřováno na vás, například biometrické ověřování. [1]

1.3 Dvoufaktorová autentizace a Biometrie

Dvoufázové ověřování (2FA), též známé jako dvoufaktorové ověřování, je bezpečnostní proces, během něhož jsou při přihlašování k online účtu vyžadovány dvě formy identifikace. Místo jednoduchého přihlášení pomocí uživatelského jména a hesla je přístup k účtu v rámci 2FA podroben opětovnému ověření totožnosti. [2]

1.3.1 Autentikační metody pro 2FA

Při využívání dvoufaktorového ověřování jsou dostupné různé metody ověření identity. Zde je uveden seznam nejčastěji preferovaných možností:

- **Hardwarové tokeny:**

Hardwarové tokeny mohou být poskytovány ve formě klíčenky, která generuje kódy každých několik sekund až minut. Jedná se o jednu z nejstarších forem dvoufaktorového ověřování.

- **Push notifications:**

Metoda dvoufaktorového ověřování pomocí push nevyžaduje žádné heslo. Tento typ 2FA odesílá signály na telefon, který vás buď vyzve k schválení/odmítnutí přístupu na webovou stránku nebo do aplikace za účelem ověření vaší totožnosti.

- **Ověření pomocí SMS:**

Tedy ověření pomocí textových zpráv, lze využít jako formu dvoufaktorového ověřování, když je zpráva odesílána na důvěryhodné telefonní číslo. Uživatel je následně vyzván k interakci s textem nebo k použití jednorázového kódu pro ověření své totožnosti na webu nebo v aplikaci.

- **Hlasové ověřování:**

Ověřování hlasem funguje podobně jako push oznámení, s tím rozdílem, že identita je potvrzena prostřednictvím automatizace. Hlas vás vyzve k stisknutí klávesy nebo k poskytnutí svého jména a identifikaci. [3]

- **Biometrické ověřování:**

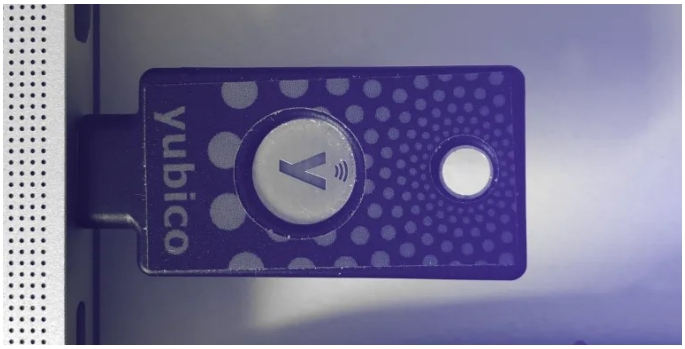
Biometrické 2FA se stále častěji využívá, kdy jste vy sami bezpečnostním tokenem. Pro prokázání, že jste to vy, kdo se přihlašuje, lze využít otisky prstů, sken obličeje, duhovky či sítnice.

V mnoha aplikacích je například umožněno využít Touch ID v iPhonu nebo Fingerprint Unlock v systému Android pro přístup k do aplikace po zadání přihlašovacích údajů.

Biometrické ověření často slouží jako třetí způsob ověření totožnosti pro vícefaktorové ověřování (MFA).

Vzhledem k obtížné možnosti padělání biometrických údajů jsou stále častěji využívány i jako přístupové metody, které zcela eliminují používání hesel, známé též jako ověřování bez hesla.

Biometrické ověřování však není bez rizika. Pokud někdo pronikne do biometrického systému 2FA a dojde k úniku nebo replikaci otisku prstu nebo skenu obličeje, nelze je změnit. [2]



Obrázek 1: Hardware Token YubiKey od Yubico

1.4 Aktualizace softwaru a Security patches

Pokud se vyhýbáte aktualizacím softwaru nebo odkládáte jejich provedení, můžete být více vystaveni hrozbám kybernetické bezpečnosti. Věnování času aktualizaci softwaru a opravě

bezpečnostních chyb se z dlouhodobého hlediska vyplatí, protože chrání vaše soukromé informace. [4]

1.4.1 Aktualizace softwaru

Zlepšení uživatelské zkušenosti je dosaženo prostřednictvím aktualizací softwaru. Připomínky k aktualizaci softwaru mohou být nepříjemné, avšak mají svůj důvod. Aktualizace softwaru označují soubor změn, které opravují nebo vylepšují software. Tato změna může zlepšit výkon a použitelnost nebo opravit kritické bezpečnostní chyby. V závislosti na softwaru mohou aktualizace probíhat automaticky. Aktualizace lze provádět na veškerém softwaru, ale aktualizace operačního systému jsou nejdůležitější, neboť na něm závisí další software, jako jsou aplikace nebo ovladače. [4]

1.4.2 Security patches

„Software updates include security patches that fix software vulnerabilities and keep malware out of your system.“

Bezpečnostní nedostatky jsou využívány kybernetickými zločinci. Slabiny nebo díry v softwarových programech, které umožňují proniknutí škodlivého softwaru do zařízení, jsou označovány jako softwarové zranitelnosti. Tyto zranitelnosti následně umožňují hackerům získat přístup k datům, ovládnout počítač nebo zpomalit systém. Bezpečnostní záplaty, které opravují slabá místa softwaru a brání průniku škodlivého softwaru do systému, jsou obsaženy v aktualizacích softwaru. Aktualizace softwaru a operačních systémů by měly být považovány za jednu z nejdůležitějších součástí rutinní kybernetické hygieny, což jsou pravidelné kroky, které mohou být podniknuty ke zlepšení online bezpečnosti. [4]

2 HESLA

V online světě je nutností používat hesla. Jsou považovány za jeden z nejdůležitějších prostředků pro zabezpečení digitálního života. Jsou používána k zabránění komukoli jinému v přístupu k cizím bankovním účtům, e-mailům, účtům na sociálních sítích a všemu ostatnímu, čemu se lidé věnují online.

Proto je důležité používat silná hesla, která nejsou uhodnutelná nikým jiným. Důležité je také, aby hesla nebyla používána opakovaně na různých webových stránkách a službách, protože pokud jsou údaje získány protivníkem z jedné stránky (například v důsledku úniku dat), mohou být tyto údaje použity k přístupu ke všem dalším službám, které sdílejí stejné přihlašovací údaje.

Pro skutečně bezpečné heslo je vyžadováno použití kombinace velkých a malých písmen, číslic a symbolů a mělo by mít alespoň osm znaků (čím delší, tím lepší). Bohužel, si lidský mozek nezapamatuje dlouhé a složité řetězce typu *g1y2<gU+9fnq;smg0:;+*, natož na zapamatování několika takových řetězců pro každou webovou službu. [5]

2.1 Složité, ale zapamatovatelné heslo

Existují různé způsoby, jak si vymyslet složité a zároveň zapamatovatelné heslo.

2.1.1 Použití mnemotechnického prostředku

Hesla, jako je h9!fdjhGH68%J@, jsou bohužel bezpečná, avšak jejich zapamatování není snadné pro člověka. Jedním způsobem, jak to vyřešit, je vytvořit frázi nebo větu, kterou lze snadno zapamatovat. Například: "Skákal pes přes oves".

Vaši frázi lze následně přeměnit na heslo tím, že se využije první písmeno každého slova a přidají se čísla a symboly. Z uvedeného příkladu by heslo mohlo být "SkakalP3s PresOves*".

Posledním krokem při vytváření úspěšné mnemotechnické pomůcky je spojení hesla s mentálním obrazem, který vám pomůže si jej zapamatovat. Například vzpomínka na lidovou píseň pro děti, kterou jistě většina lidí zná, může vyvolat vzpomínku na heslo.

2.1.2 Použití náhodné heslové fráze

Další možností je použití řetězce náhodných, avšak zapamatovatelných slov. Například může být vytvořen řetězec "Blue Tiger Pizza Rainbow" (mezi slovy ponechané mezery, které zvyšují složitost). Tento druh náhodné heslové fráze lze snadno vytvořit pomocí skvělého, nenáročného nástroje, kterým je Diceware, nebo můžete využít generátor ve správci hesel. [6]

2.2 Odolnost hesel

Navíc, silná hesla používaná k ověřování mohou být odolná vůči útokům hrubou silou na jednu stranu, ale na druhou stranu jsou nepoužitelná proti útokům typu phishing a key-logger software nebo password stuffing. Tyto typy útoků nejsou zaměřeny na odhadnutí hesla uživatele, ale přímo na jeho krádež. [1]

2.3 Prolamování hesel

Kdysi specializovaná dovednost je nyní dostupné téměř všem. Kdokoli pomocí široce dostupných nástrojů, jako jsou John the Ripper, Medusa nebo Cain & Abel, může zkoušet prolomit heslo. [7]

Prolamování hesel je proces, při kterém se zašifrované nebo zahashované heslo - formy, v níž jsou hesla v systémech obvykle uložena - přemění zpět do původní podoby otevřeného textu, kterou vytvořil uživatel. Tento druh dešifrování hesla se nazývá prolomení pouze tehdy, pokud se provádí mimo proces ověřování. [8]

„*Hashování není šifrování.*“ – pan Ing. Petr Žáček, Ph.D.

Záměr prolamování hesel může být jak benevolentní, tak i zlomyslný. Správci systémů tak mohou využívat prolamování hesel ke kontrole síly hesel, aby eliminovali ta slabá. Mezitím mohou hackeři pomocí nástrojů pro prolamování hesel získat neoprávněný přístup k různým účtům a systémům a vykrást citlivé údaje. [8]

Dobré je taky znát způsoby, jakými systémy hesla ukládají.

2.3.1 Otevřený text, šifrování a hash

Tři základní metody, které mohou být použity systémem k uložení hesla. Pokaždé, co uživatel zadá heslo, musí být systémem určena metoda, jak zjistit, zda je heslo správně zadané. Musí být *„něco“* uloženo. První a nejzřejmější metodou je prosté uložení hesla přesně tak, jak bylo zadáno. Tato metoda otevřeného textu ukládá data bez jakéhokoli zastírání, šifrování nebo kódování.

Další metodou je šifrování každého hesla před jeho uložením např. do databáze. Šifrování kombinuje otevřený text s dalším tajným klíčem, čímž vzniká zkomolený řetězec, který lze získat pouze pomocí stejného klíče. Jinými slovy, šifrování je pouze uložení hesla chráněného heslem. Opět, kdokoli s tímto hlavním heslem by měl přístup k celé databázi, takže je jen o něco málo bezpečnější než prostý text.

Šifrování hesel je obecně pro mnoho účelů nepřijatelné, ale je to rozhodně lepší než otevřený text. Někdy aplikace musí uložit heslo jako slovo a načíst otevřený text pro pozdější použití, a to nelze obejít. Například systém Windows šifruje a ukládá různá hesla, aby mohl spouštět systémové služby a připojovat se k různým zdrojům. Často se s tím setkáte, když se objeví přihlašovací dialogové okno a vaše heslo je již zadáno, reprezentováno řetězcem hvězdiček.

Při ukládání hesel je běžnou praxí použití hashovacích funkcí. Hash je výsledkem algoritmu, který zpracovává vstupní heslo a vytváří z něj specifický řetězec znaků, který ho reprezentuje. Tyto algoritmy jsou jednosměrné, což znamená, že není možné z hash hodnoty zpětně získat původní heslo. Pro ověření hesla systém použije stejný hashovací algoritmus na zadaný vstup a porovná výsledek s údaji uloženými v hashovací databázi. Shodují-li se, systém ví, že obě hesla musela být totožná, aby vytvořila stejný hash. [7]

2.3.2 Hash a Solnička

I když hash nelze přímo dešifrovat, hackeři mají technickou možnost provést zpětné dešifrování hashe, což jim umožňuje poměrně přesně odhadnout původní vstup. S přístupem k hashům webových stránek a trochou času mohou být schopni zjistit heslo uživatele v prostém textu.

Právě v takových situacích může být solení hesel užitečné. Solení hesla zahrnuje přidání dalších znaků do otevřeného textu hesla před zahashováním.

2.4 Limitace tradičního přístupu k heslům a Potřeba správce hesel

Navzdory snaze o využívání silných hesel k posílení bezpečnosti existují stále hrozby, kterým je tento tradiční přístup neodolný. S rostoucím komplexním prostředím kybernetických hrozeb je zapotřebí inovativního přístupu k ochraně hesel. Moderní nástroje, známé jako správci hesel, nabízejí efektivní řešení těchto výzev a umožňují správcům systémů lépe řídit a chránit přístupové údaje.

3 DEFINICE SPRÁVCE HESEL

Správce hesel je počítačový program (aplikace), která je schopna generovat bezpečná hesla a bezpečně je ukládat. Také ukládá hesla, vytvořená uživatelem. Většina moderních správců hesel umožňuje snadné zadávání těchto uložených hesel (a dalších přihlašovacích údajů, například uživatelského jména) při přihlašování na webové stránky a do aplikací.

Protože jsou však klíče uchovávány k digitálnímu životu uživatele, je velmi důležité, aby byl samotný správce hesel bezpečný.

Používají tedy jisté hlavní heslo, aby zabránili neoprávněnému přístupu. To znamená, že uživatelé takové aplikace si musí pamatovat pouze jejich hlavní heslo, místo aby si pamatovali hesla ke všem svým účtům. [1][5]

3.1 Vývoj

Vývoj správců hesel odráží neustálou snahu o zvýšení bezpečnosti od počátků hesel v prostém textu až po dnešní sofistikované šifrovací algoritmy. Sledování historie správců hesel odhaluje příběh inovací, odolnosti a přizpůsobování se neustále se měnícímu prostředí hrozeb. [9]

3.2 Bezpečnost

Správci hesel jsou koncipováni s cílem zajistit bezpečnost vašich hesel; avšak otázka, jak jsou sami o sobě bezpeční, přetrvává. Ve hře je mnoho faktorů. Obecně lze konstatovat, že kvalitní správce hesel je navržen s ohledem na ochranu před kyberzločinci a zajištění soukromí vašich údajů. Úspěch této ochrany však značně závisí na konkrétní službě.

Existuje několik společných prvků, které nejlepší správci hesel sdílejí. Prvním z nich je inteligentní používání šifrování. Kvalitní správce hesel šifruje trezory, kde ukládáte svá hesla, pomocí nejmodernějšího šifrovacího algoritmu, jako je například AES-256.

Pokud však služba skutečně dbá na soukromí uživatelů, využívá také end-to-end šifrování. Při běžném šifrování jsou šifrovací klíče sdíleny jak uživatelem, tak službou, při odesílání dat z vašeho počítače na servery správce hesel.

Naopak end-to-end šifrování vám poskytuje klíč pouze vám. Tím je eliminována možnost sledování ze strany služby při přenosu i ukládání dat. I když je implementace této technologie technicky obtížná, někteří správci hesel ji nenabízejí.

V případě, že služby nevyužívají end-to-end šifrování, mohou mít následky katastrofální důsledky. Největší únik dat v historii cloudového úložiště, k němuž došlo v roce 2012 u služby Dropbox, byl způsoben opakovaným používáním zaměstnance této služby jeho hesel, což umožnilo hackerům proniknout do systému. Hackeři poté dešifrovali databázi a ukradli hesla 70 milionů uživatelů. V případě využívání end-to-end šifrování by k takovému incidentu nemohlo dojít.

Používání bezpečnostních opatření, jako je end-to-end šifrování, minimalizuje riziko lidské chyby a s ní spojenou pravděpodobnost narušení. [10]

3.2.1 End-to-end šifrování

E2EE šifrování je procesem transformace dat, která jsou normálně čitelná pro lidi (například e-mail s otevřeným textem), do nečitelného šifrovaného textu, jenž může být rozluštěn pouze legitimními stranami prostřednictvím správného kryptografického klíče.

Termín "šifrování end-to-end" popisuje metodu šifrování, při níž jsou data zakódována v každém kroku své cesty z jednoho zařízení na druhé. "End-to-end" označuje počáteční a konečný bod trasy dat. Například při odesílání e-mailu je počátečním bodem vaše zařízení a cílovým bodem je zařízení příjemce.

Takovým způsobem E2EE reprezentuje bezpečný způsob komunikace, který brání všem třetím stranám v přístupu k obsahu vašich zpráv, a to jak během přenosu z jednoho zařízení na druhé, tak i "v klidu" na serveru. Když někomu pošlete e-mail pomocí E2EE, obsah vaší zprávy zůstává nedostupný pro každého kromě příjemce, který má jediný klíč umožňující dešifrování obsahu. [11]

Obrázek 2: Proces E2EE

3.3 Typy

Optimální výběr správce hesel pro vaše potřeby závisí na několika faktorech. Je preferováno využívání vestavěného správce hesel v zařízení nebo prohlížeči před absencí takového. Je třeba však mít na paměti, že jsou omezené a nabízejí podstatně menší hodnotu ve srovnání s bezplatným cloudovým správcem hesel. [12]

3.3.1 Cloudový správci hesel

Patří mezi oblíbené možnosti pro jednotlivce i firmy. Hesla a citlivé údaje jsou šifrovány a chráněny na vlastních serverech. Hlavní výhodou je snadný přístup odkudkoli pomocí jakéhokoli počítačového zařízení. Služba je centrálně hostována a udržována poskytovatelem služby. [12]

3.3.2 On-premise (lokální) správci hesel

I když jsou dostupné mnohé cloudové aplikace pro správu hesel v mobilních zařízeních, k dispozici jsou i nativní správci hesel v systémech iOS a Android, jako jsou Apple Keychain a Google Password Manager, přičemž umožňují bezpečné ukládání hesel do mobilního zařízení a automatické vyplňování hesel na webových stránkách a v aplikacích. [12]

3.3.3 Open-source správci hesel

Transparentnost a možnost přizpůsobení činí open-source správce hesel oblíbenými. Jsou poskytovány zdarma, a jejich veřejně dostupný zdrojový kód umožňuje komunitě vývojářů provádět revize bezpečnosti. [13]

3.3.4 Komerční

Na druhé straně, prémiové funkce, jako je podpora zákazníků, týmové sdílení a další pokročilé možnosti, jsou nabízeny komerčními správci hesel. Tyto správce hesel jsou obvykle zpoplatněny, i když některé mohou poskytovat omezené verze zdarma. [13]

3.4 Budoucnost

Budoucnost správců hesel se předpokládá v inovacích, které přinesou novou úroveň zabezpečení a zvládnou výzvy budoucnosti, od biometrického ověřování po decentralizovaná řešení identity. [13]

Hesla prošla vývojem od svých skromných počátků v podobě základních alfanumerických kódů až po složité řetězce vyžadující kombinaci znaků, čísel a symbolů. Tento vývoj odráží zvýšené povědomí o hrozbách kybernetické bezpečnosti.

Svět správy hesel se rychle vyvíjí pod vlivem nových technologií a měnících se potřeb uživatelů. Podívejme se na současný stav vývoje v oblasti správy hesel a zkoumejme, jak tyto trendy formují náš přístup k digitálnímu zabezpečení.

1. Častější používání správců hesel

S rostoucí složitostí požadavků na hesla roste oblíbenost správců hesel. Kromě pamatování hesel umožňují generování silných a jedinečných hesel pro každý účet. Další významnou výhodou je jejich schopnost automatického vyplňování přihlašovacích údajů, což snižuje riziko phishingu tím, že zajišťuje zadávání přihlašovacích údajů pouze na legitimních stránkách.

2. Vzestup vícefaktorového ověřování (MFA)

Dalším trendem, který nabírá na síle, je vícefaktorové ověřování. MFA přidává další vrstvu zabezpečení tím, že pro získání přístupu k účtu vyžaduje dva nebo více ověřovacích faktorů. Např. heslo, PIN, otisk prstu nebo rozpoznání obličeje.

3. Biometrické ověřování

Tahle oblast se rychle stává klíčovým hráčem v oblasti správy hesel. Tato metoda využívá k ověření identity jedinečné biologické charakteristiky, jako jsou otisky prstů, rozpoznávání obličeje, skenování duhovky nebo rozpoznávání hlasu.

- Rozpoznávání otisků prstů a obličeje

Tyto technologie nabízejí pohodlný a bezpečný způsob odemykání zařízení a ověřování totožnosti uživatelů. Novější modely jsou schopny 3D mapování a pokročilých rozpoznávacích algoritmů, které zabraňují falšování a falešným poplachům.

- Skenování duhovky a rozpoznávání hlasu

Tyto technologie, méně běžné, ale v prostředí s vysokým stupněm zabezpečení se stále více prosazují. Skeny duhovky poskytují vysokou úroveň přesnosti díky jedinečným vzorům v duhovce každého člověka. Rozpoznávání hlasu, i když pohodlné, se stále potýká s problémy při přesném zachycení hlasových vzorů, zejména v hlučném prostředí.

Biometrické ověřování představuje významný posun oproti tradičním systémům hesel a nabízí kombinaci vyšší bezpečnosti a uživatelského pohodlí. Vyvolává však také důležité obavy o ochranu soukromí, protože biometrické údaje jsou vysoce citlivé a v případě kompromitace je nelze změnit jako heslo.

S tím, jak se pouštíme do oblasti nových technologií, je jasné, že inovativní řešení budou určovat budoucnost správy hesel. Převratné pokroky, které jsou připraveny v příštích letech, budou nově definovat zabezpečení hesel. [14]

3.4.1 Umělá inteligence ve správě hesel

Revoluci ve správě hesel přináší umělá inteligence prostřednictvím prediktivní analýzy a automatického generování silných hesel. Algoritmy umělé inteligence jsou schopny analyzovat chování uživatelů, předvídat potenciální bezpečnostní hrozby a navrhovat změny k zvýšení bezpečnosti. Správci hesel s umělou inteligencí navíc mohou generovat složitá hesla, která jsou pro hackery téměř neuhádnutelná, ale pro uživatele je správa jednodušší. [14]

3.4.2 Technologie blockchain pro bezpečnou správu hesel

Získávající pozornost technologie blockchain přináší decentralizaci a zvyšuje bezpečnost systémů pro správu hesel. Ukládáním hesel v decentralizované síti se výrazně snižuje rizi-

ko narušení centralizované databáze. Vlastnosti blockchainu, jako je neměnnost a transparentnost, poskytují robustní rámec pro bezpečné procesy ověřování. [14]

3.4.3 Kvantová výpočetní technika a zabezpečení hesel

Nástup kvantové výpočetní techniky představuje výzvy i příležitosti pro zabezpečení hesel. Kvantové počítače by potenciálně mohly prolomit současné metody šifrování, ale také otevírají cestu k vývoji šifrovacích algoritmů odolných vůči kvantům. Tato nově vznikající oblast je připravena radikálně změnit koncepci a implementaci zabezpečení hesel. [14]

3.4.4 Kybernetická bezpečnost a vzdělávání uživatelů

V oblasti správy hesel je nejdůležitější vzdělávání uživatelů. Informovanost o osvědčených postupech kybernetické bezpečnosti může výrazně snížit riziko narušení bezpečnosti. To zahrnuje pochopení významu používání silných a jedinečných hesel, rozpoznávání pokusů o phishing a bezpečné používání správců hesel a metod ověřování.

Jednotliví uživatelé i organizace musí přijmout osvědčené postupy pro správu hesel. Uživatelé by měli používat renomované správce hesel, zapínat funkci MFA a být ostražití v oblasti online bezpečnosti. Organizace by měly prosazovat přísné zásady používání hesel, provádět pravidelné bezpečnostní audity a poskytovat zaměstnancům školení o rizicích kybernetické bezpečnosti a preventivních opatřeních. [14]

3.4.5 Předpovědi do budoucna a názory odborníků

Názory a předpovědi odborníků nabízejí cenné pohledy na směřování správy hesel do budoucna. Předpokládá se větší integrace umělé inteligence a biometrie do autentizačních procesů a potenciální posun k decentralizovanějším systémům, jako je blockchain. Dalším velkým skokem ve správě hesel by mohlo být široké přijetí ověřování bez hesla. [14]

Správci hesel také mohou nabídnout užitek z poskytování informací o kreditních kartách, adresách a dalších osobních údajích uživatelů. Tím umožňují pohodlnější a bezpečnější nakupování online. [15]

"*The future of password management is likely to be a combination of passwordless authentication and password managers. Passwordless authentication is undoubtedly the future,*

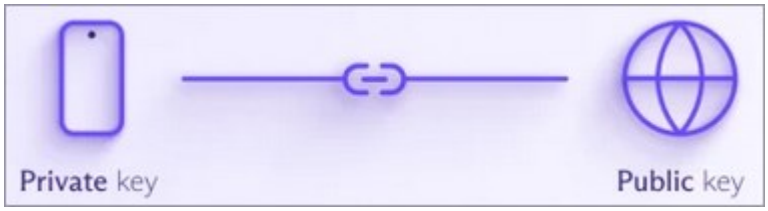
as it offers significant advantages over traditional password-based authentication. However, it's unlikely to completely replace password managers, as they still offer value in managing other sensitive data and syncing passwords across multiple devices," - Kevin Garce, Vedoucí marketingu ve společnosti iwoolfelt [15]

3.4.6 Passkeys

Passkeys jsou formou identifikace, která může být použita k získání přístupu k účtu. Nahrazují potřebu hesla a dvoufaktorové ověřování (2FA) tím, že k identifikaci uživatele využívají zařízení. Díky tomu, že není potřeba zadávat heslo, které by mohlo být ukradeno, passkeys chrání před phishingem. Jsou také méně náchylné k útokům hrubou silou, protože fungují na základě vytváření kryptografických klíčů.

Obvykle získáte přístup k účtu zadáním pověření, které jste uvedli při jeho vytváření: uživatelské jméno (často e-mailová adresa) a heslo. Ne však v případě passkeys. Když si vytvoříte účet u služby, která podporuje přístupové klíče, vygeneruje se sada šifrovacích klíčů správcem hesel. Při příštím pokusu o přístup k webu rozpozná klíče, které máte,
a přihlásí vás bez nutnosti zadávat heslo.

Využívají princip asymetrické kryptografie neboli kryptografie s veřejným klíčem. Jde o to, že při vytvoření passkeys se vygenerují dva matematicky propojené číselné klíče: jeden veřejný a druhý soukromý.



Obrázek 3: Soukromý a veřejný klíč

Veřejný klíč má služba, ke které se přihlašujete, zatímco vy jako uživatel, máte soukromý klíč, který je uložen ve správci hesel. Při přihlašování ke službě odešle veřejný klíč do vašeho zařízení výzvu, na kterou může správně odpovědět pouze váš soukromý klíč, čímž vás identifikuje jako vlastníka účtu.

Systém je velmi bezpečně navržen a je prakticky odolný proti útokům hrubou silou. Prolomení čísel, která se používají v kryptografii s veřejným klíčem, by vyžadovalo kombinaci kvantových počítačů a velkou spoustu času. [16]

4 VHODNÉ TECHNOLOGICKÉ PROSTŘEDKY

Pro úspěšnou implementaci projektu je nutné vybrat technologické prostředky, se kterými bude možné dosáhnout odpovídajícím požadavkům a cílům. S těmito prostředky je možné efektivně a spolehlivě realizovat navržené funkcionality a konečné aplikace.

4.1 Technogie a Framework pro Webovou Aplikaci

Framework pro vývoj webových aplikací je souborem knihoven, nástrojů a standardizovaných postupů, které poskytují strukturu pro vytváření aplikací.

Proces vývoje je zjednodušen tím, že nabízí opakovaně použitelné komponenty, předpřipravený kód a sadu pokynů, které umožňují vývojářům rychle a efektivně vytvářet webové stránky. Frameworky odstraňují potřebu začínat od nuly, což šetří čas a úsilí a podporuje osvědčené postupy kódování.

Frameworky podporují tvorbu spolehlivých, bezpečných a škálovatelných aplikací s využitím osvědčených postupů v oboru. Vnáší do procesu vývoje řád a strukturu, což umožňuje efektivní spolupráci. Kromě toho frameworky podporují přijetí standardizovaných kódovacích postupů, čímž zvyšují čitelnost kódu, udržovatelnost a celkovou kvalitu výsledného produktu. [17]

4.1.1 ASP.NET Core 8

ASP.NET Core je multiplatformní open-source architektura postavená na vývojářské platformě .NET 8 vytvořený specificky pro vývoj webových aplikací.

ASP.NET Core nabízí následující výhody:

- AntiForgeryToken, což je funkce, která se používá k prevenci útoků typu Cross-Site Request Forgery (XSRF/CSRF). Tyto útoky jsou možné, protože webové prohlížeče automaticky odesílají některé typy autentizačních tokenů s každým požadavkem na webovou stránku.

- Razor Pages usnadňují a zefektivňují kódování scénářů zaměřených na stránky oproti používání controllers a views.

- Blazor umožňuje pracovat s jazykem C# v prohlížeči spolu s JavaScriptem. Je tedy možné sdílet logiku aplikace na straně serveru a na straně klienta, která je napsaná pomocí .NET.

- Podpora poskytování služeb vzdáleného volání procedur (RPC) pomocí gRPC.

- Konfigurační mechanismus založený na prostředí, který je připravený pro cloud.

- Možnost nasazení na následujících zařízeních:

    - Nginx

    - Apache

    - Docker [18]

4.1.2 ASP.NET Core 8 Blazor

Blazorje front-endová webový framework postavený na ASP.NET Core, který podporuje vykreslování na straně serveru i interaktivitu klienta v jednom programovacím modelu:

- Je možné vytvářet bohatá interaktivní uživatelská rozhraní pomocí jazyka C#.
- Logika aplikace na straně serveru i klienta.
- Uživatelská rozhraní jsou vykreslována jako HTML a CSS pro širokou podporu prohlížečů.

Architektura založená na komponentách s vykreslováním na straně serveru a úplnou interaktivitou na straně klienta v jednom řešení je poskytována Blazor Web Apps službou, kde můžete být schopni přepínat mezi režimy vykreslování na straně serveru a na straně klienta a dokonce je kombinovat na stejné stránce.

Uživatelské rozhraní může být rychle dodáváno do prohlížeče statickým vykreslováním obsahu HTML ze serveru v reakci na požadavky Blazor Web Apps. Stránka se načítá svižně, protože vykreslování uživatelského rozhraní je prováděno urychleně na serveru bez nutnosti stáhnout objemnou sadu JavaScriptu. Uživatelské prostředí může být dále vylepšováno různými progresivními vylepšeními vykreslování serveru Blazorem, jako je zlepšená navigace s formulářovými příspěvky a streamování asynchronně generovaného obsahu vyobrazování.

Interaktivní vykreslování na straně serveru (interaktivní SSR), kde jsou interakce uživatelského rozhraní zpracovávány ze serveru přes připojení v reálném čase s prohlížečem, je podporováno Blazorem. Bohaté uživatelské prostředí, které by se očekávalo od klientské aplikace, ale bez nutnosti vytvářet koncové body rozhraní API pro přístup k zdrojům serveru, je umožněno interaktivním rozhraním SSR. Obsah stránky pro interaktivní stránky je předem generován, kde je obsah na serveru původně vygenerován a odeslán klientovi bez povolení obslužných rutin událostí pro vyobrazené ovládací prvky. Server vykresí uživatelské rozhraní HTML stránky co nejdříve v reakci na počáteční požadavek, což umožňuje aplikaci odpovídat na uživatele.

Interaktivita při vykreslování na straně klienta (CSR), která využívá modul runtime .NET vytvořený pomocí WebAssembly, který si můžete stáhnout s aplikací, je podporována Blazor Web Apps službou. Při spuštění Blazoru na WebAssembly má váš kód .NET přístup k plné funkčnosti prohlížeče a spolupráci s JavaScriptem. Váš kód .NET běží v sandboxu zabezpečení prohlížeče s ochrannou, kterou sandbox poskytuje proti škodlivým akcím na klientském počítači. Blazor aplikace mohou zcela namířit na webAssembly v prohlížeči bez zapojení serveru. U samostatné Blazor WebAssembly aplikace jsou zdroje nasazeny jako statické soubory na webovém serveru nebo službě, které mohou klientům obsluhovat statický obsah. Po stažení je možné samostatné Blazor WebAssembly aplikace ukládat do mezipaměti a spouštět offline jako progresivní webové aplikace (PWA). [19]

Na straně Blazor serveru je stavová aplikační architektura udržována. Trvalé připojení k serveru je aplikací zachováváno a stav uživatele je v paměti serveru uložen v okruhu. Jedním z příkladů stavu uživatele jsou data uchovávaná v instancích služby injektáže závislostí (DI), které jsou na okruh omezeny. Speciální přístup k používání Entity Framework Core je jedinečným aplikačním modelem, který Blazor poskytuje, vyžadován. [20]

4.1.3 Entity Framework ORM

Relační část je systémem pro řízení relačních databází (tedy databází). Existují i jiné druhy databází, ale oblíbenější je relační (tabulky, sloupce, primární klíč, cizí klíč atd. jsou známé, např. Oracle, MySQL, MSSQL).

A konečně mapovací sekce je částí, kde se uskutečňuje propojení mezi objekty a tabulkami.

Entity Framework Core je odlehčenou, rozšiřitelnou a multiplatformní verzí oblíbené technologie pro přístup k datům Entity Framework.

EF Core může fungovat jako objektově relační mapovač (ORM), což:

- Dovoluje vývojářům pracovat s databází pomocí objektů.
- Odstraňuje potřebu většiny kódu pro přístup k datům, který je obvykle nutno sepsat.

EF Core podporuje mnoho databázových enginů. [21]

4.2 Vývojářské prostředí

Integrované vývojové prostředí (IDE) je softwarová sada, ve které jsou všechny vývojové nástroje spojeny do jediného grafického uživatelského rozhraní. Díky tomu může být proces vývoje efektivněji a rychleji realizován. Příklady oblíbených integrovaných vývojových prostředí jsou Microsoft Visual Studio nebo produkty od JetBrains. [22]

4.2.1 Typy integrovaných vývojových prostředí

Je důležité, aby byly zváženy různé aspekty každého projektu, protože může být vyžadováno jiné IDE. Aspekty jako programovací jazyk, snadnost použití, spolehlivost a typ musí být zváženy. Existuje mnoho typů IDE, které nabízejí různé funkce, a tyto funkce umožňují efektivní vytváření kvalitních aplikací. [22]

4.2.2 JetBrains Rider

Pro vývojáře v jazycích C# a .NET je určeno multiplatformní IDE Rider, přičemž podporovány jsou i další jazyky. Díky své inteligentní architektuře je Rider oproti mnoha jiným IDE výrazně rychlejší. Za účelem zefektivnění procesu vývoje je poskytována rozsáhlá sada funkcí. Od inteligentního dokončování kódu a detekce chyb za běhu až po robustní nástroje pro refaktorizaci, je vše k dispozici k rozšíření možností kódování.

Uživatelské rozhraní aplikace Rider je přehledné a přizpůsobitelné. Postranní panel nástrojů, okno editoru, struktura projektu a další, git branche, spouštěcí profily, commit do cloudové služby, jako je například Azure, projektové okno při startu a další součásti jsou navrženy tak, aby zlepšily a usnadnily práci. [23]

4.2.3 Docker

Docker je platformou pro vývoj, distribuci a spouštění aplikací. Umožňuje oddělit aplikace od infrastruktury, aby bylo možné rychle dodávat software. Infrastruktura je spravována stejným způsobem jako aplikace s využitím nástroje Docker. Využitím metodik nástroje pro distribuci, testování a nasazování kódu může být výrazně zkrácena prodleva mezi napsáním kódu a jeho spuštěním v produkci.

Možností, kterou Docker poskytuje, je zabalit a spustit aplikaci ve volně izolovaném prostředí zvaném kontejner. Izolace a zabezpečení umožňují spustit na jednom hostiteli mnoho kontejnerů současně. Kontejnery jsou odlehčené a obsahují vše potřebné ke spuštění aplikace, takže se nemusí spoléhat na to, co je nainstalováno na hostiteli. Kontejnery lze snadno sdílet a mít jistotu, že každý, s kým jsou sdíleny, dostane stejný kontejner, který funguje stejně.

Dále poskytuje nástroje a platformu pro správu životního cyklu kontejnerů:

- Vývoj aplikace a jejích podpůrných komponent pomocí kontejnerů.
- Kontejner se stává jednotkou pro distribuci a testování aplikace.
- Až bude připravenost dosažena, aplikace může být nasazena do produkčního prostředí jako kontejner nebo orchestrovaná služba. Tento postup je stejný, bez ohledu na to, zda je produkčním prostředím lokální datové centrum, poskytovatel cloudu nebo jejich hybrid.

Docker se používá pro rychlé a konzistentní dodávání aplikací. Životní cyklus vývoje je efektivnější díky Dockeru, který umožňuje vývojářům pracovat ve standardizovaných prostředích pomocí místních kontejnerů. Ty jsou skvělé pro kontinuální integraci a kontinuální dodávání (CI/CD).

- Vývojáři píší kód lokálně a sdílejí svou práci s kolegy pomocí kontejnerů Docker.
- Pomocí Dockeru přesouvají své aplikace do testovacího prostředí a spouštějí automatizované a manuální testy.
- Když vývojáři najdou chyby, mohou je opravit ve vývojovém prostředí a znovu je nasadit do testovacího prostředí k testování a ověření.

- Po dokončení testování je předání opravy zákazníkovi stejně jednoduché jako odeslání aktualizovaného kontejneru do produkčního prostředí.

Kontejnery mohou být spuštěny na lokálním notebooku vývojáře, na fyzických nebo virtuálních strojích v datovém centru, u poskytovatelů cloudových služeb nebo v kombinaci různých prostředí.

Přenositelnost a odlehčená povaha kontejnerů usnadňuje také dynamická správa pracovních zátěží, škálování nebo rušení aplikací a služeb podle potřeb podniku, a to téměř v reálném čase.

Docker využívá architekturu klient-server. Klientem Docker je komunikováno s daemonem Docker, který se stará o sestavování, spouštění a distribuci kontejnerů. Klient a daemon mohou běžet na stejném systému, nebo klient může být připojen k vzdálenému daemonu. Mezi nimi je komunikováno pomocí rozhraní REST API, přes sockety UNIX nebo síťové rozhraní. Dalším klientem je Docker Compose, který umožňuje práci s aplikacemi složenými ze sady kontejnerů.

Kontejnery jsou spustitelnými instancemi image. Mohou být vytvářeny, spouštěny, zastavovány, přesouvány nebo odstraňovány pomocí rozhraní Docker API nebo CLI. Kontejner může být připojen k jedné nebo více sítím, může být k němu připojeno úložiště nebo může být dokonce vytvořen nový image na základě jeho aktuálního stavu.

Ve výchozím nastavení je kontejner relativně dobře izolován od ostatních kontejnerů a svého hostitelského počítače. Úroveň izolace sítě, úložiště nebo jiných základních subsystémů kontejneru od ostatních kontejnerů nebo od hostitelského počítače může být ovlivněna.

Kontejner je definován svou bitovou kopií a všemi konfiguračními možnostmi, které mu jsou poskytnuty při jeho vytváření nebo spuštění. Při odstranění kontejneru jsou ztraceny veškeré změny jeho stavu, které nejsou uloženy v trvalém úložišti.

Image je šablonou pouze pro čtení s pokyny pro vytvoření kontejneru. Často je založena na jiném image s určitými dalšími úpravami. Například může být vytvořen image, který je založen na image ubuntu, ale nainstaluje webový server Apache a vaši aplikaci, spolu s konfiguračními údaji potřebnými k jejímu spuštění.

Vlastní images mohou být vytvářeny nebo mohou být používány ty, které byly vytvořeny jinými a jsou zveřejněny v registru. Chcete-li vytvořit vlastní image, musí být vytvořen soubor Dockerfile s jednoduchou syntaxí pro definování kroků potřebných k vytvoření obrazu a jeho spuštění. Každým pokynem v souboru Dockerfile je vytvářena vrstva image. Když je soubor Dockerfile změněn a image je obnoven, jsou obnoveny pouze ty vrstvy, které se změnily. Tím se image stávají lehkými, malými a rychlými ve srovnání s jinými virtualizačními technologiemi. [24]

4.2.4 Docker Compose

Docker Compose je nástrojem pro definování a spouštění aplikací s více kontejnery.

Compose je nástrojem, který zjednodušuje kontrola celého vašeho aplikačního stacku a usnadňuje správa služeb, sítí a svazků v jednom srozumitelném konfiguračním souboru YAML. Poté jedním příkazem jsou vytvořeny a spuštěny všechny služby z vašeho konfiguračního souboru.

Compose je používán ve všech prostředích; produkci, stagingu, vývoji, testování, a také pracovních postupech CI. Má také příkazy pro správu celého životního cyklu vaší aplikace:

- Služby jsou spouštěny, zastavovány a přebudovávány
- Stav běžících služeb je zobrazován
- Výstup protokolu běžících služeb je streamován
- Jednorázový příkaz je spouštěn na službě [25]

4.2.5 MSSQL Server a Databáze

Microsoft SQL Server je relačním systémem pro správu databází (RDBMS). K instanci nebo databázi SQL Serveru je připojováno aplikacemi a nástroji a je s nimi komunikováno pomocí jazyka Transact-SQL (T-SQL).

Server SQL Server může být nainstalován do systému Windows nebo Linux, nasazen v linuxovém kontejneru nebo nasazen na virtuálním stroji Azure či jiné platformě virtuálního stroje. [26]

SQL Server se skládá ze dvou hlavních součástí:

- Databázový engine
- SQLOS

Základní komponentou SQL Serveru je Database Engine, který se skládá z relačního enginu, který zpracovává dotazy, a úložného enginu, který spravuje databázové soubory, stránky, indexy atd. Navíc vytváří databázové objekty, jako jsou uložené procedury, pohledy a spouštěče.

Relační engine obsahuje součásti, které určují optimální způsob provedení dotazu. Je také známý jako procesor dotazů. Relační engine si na základě vstupního dotazu vyžádá data z úložného enginu a zpracuje výsledky. Mezi jeho úkoly patří zpracování dotazů, správa paměti, správa vláken a úloh, správa vyrovnávací paměti a distribuované zpracování dotazů.

Úložný engine je zodpovědný za ukládání a načítání dat z úložných systémů, jako jsou disky a SAN.

Pod relačním enginem a úložným enginem se nachází operační systém SQL Server Operating System neboli SQLOS. Poskytuje různé služby operačního systému, jako je správa paměti a vstupně-výstupních operací, stejně jako zpracování výjimek a synchronizační služby. [27]

SQL server je možné rychle nainstalovat na Windows nebo Linux, ale pohodlnější možnost je rozběhnout SQL server v Docker kontejneru. Lépe řečeno, není potřeba žádná instalace, pokud již máte nainstalovaný Docker. Stačí si stáhnout image z Docker hubu, nebo využít desktop aplikace Docker desktop, kde je možné prohledávat Docker hub podle názvu image, bez potřeby použít prohlížeč. Po stažení image stačí zadat 3 env proměnné (typ serveru, heslo a přijmout EULA) do spouštění dockeru a je to, výchozí uživatel je pak sa, a typ serveru je výchozí Developer, tudíž tahle env proměnná je volitelná.

SQL je možné spustit i pomocí příkazu v terminálu, není tedy potřeba interaktivní proces uživatele. Např.:

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=yourStrong(!)Password" -p 1433:1433 -d mcr.microsoft.com/mssql/server:2022-latest
```

[28]

4.2.6 Výhody SQL Serveru v Docker kontejneru před lokální instalací

- Cloud-ready

Je možné snadno spustit řešení na VPS, nebo GCP, AKS, nebo AWS. Vaše kontejnery se dají spustit na jakémkoli místě.

- Lepší ceny

Testování řešení na Linux runtime je možné, což vám ušetří peníze. To je proto, že virtuální servery na Windows jsou více nákladné než virtuální servery na Linux.

- Testování proti různým serverům/verzi

Je možné testovat řešení s různými verzemi SQL serveru, stačí změnit variantu image nebo typ serveru v prostředí proměnné.

- Izolace

Lze snadno vytvořit samostatné mosty sítě s SQL serverem, kontrolovat přístup. Můžete spustit několik instancí na jednom počítači najednou, snadno, pouze oddělováním sítí pomocí prostředků Docker.

- Resetování

Testování vyžaduje, aby všechny změny mohly být resetovány a všechny testy mohly být spuštěny od začátku (od stejného počátečního bodu). S kontejnery a svazky toho dosáhnete jedním příkazem.

- Transparentní konfigurace

Poskytujete Dockerfile a compose.yaml, kde jsou všechny kroky explicitně napsány jasně. Nepotřebujete poskytnout další readme's o tom, jak nastavit váš server.

- Cross-platform

Konfigurace Dockeru běží na jakémkoli operačním systému bez změn. Možná, že designéři používají MacOS a také chtějí spustit řešení lokálně? Je to snadné s Dockerem. [28]

4.3 Model

Manipulace s daty je prováděna pomocí modelu v EF Core. Model je konstruován třídami entit a kontextovým objektem, který je reprezentací spojení s databází. Dotazování a ukládání dat je umožněno kontextovým objektem.

EF nabízí následující metody k vývoji modelů:

- Model je vytvořen z existující databáze.
- Kód modelu je ručně sepsán tak, aby odpovídal databázi.
- Databáze je vytvořena z modelu pomocí migrací EF po jeho vytvoření. Evoluce databáze je umožněna migracemi při změnách modelu.

Dotazování je proces, během kterého jsou instance tříd entit načítány z databáze pomocí dotazů LINQ (Language Integrated Query). [29]

4.4 Přístup k databázi a update UI

Na nastavení DbContext se EF Core opírá pro přístup k databázi a je využíván jako jednotka práce. Rozhraní pro aplikace ASP.NET Core jsou nabízeny EF Core AddDbContext, které ve výchozím nastavení registrují kontext jako službu jako scoped. V aplikacích na straně Blazor serveru může být registrace vymezených služeb považována za problematickou, protože instance jsou sdíleny mezi komponentami v okruhu uživatele. DbContext není považován za bezpečný pro vlákna a není určen pro souběžné použití.

Stávající životnosti jsou považovány za nevhodné z těchto důvodů:

- Singletonem je sdílen stav napříč všemi uživateli aplikace a vede k nevhodnému souběžnému použití.
- Scoped (výchozí) představuje podobný problém mezi komponentami pro stejného uživatele.
- Transientem je vytvořena nová instance na požadavek, ale vzhledem k tomu, že komponenty mohou být dlouhodobé, výsledkem je delší kontext, než bylo zamýšleno.

Následující doporučení jsou navržena tak, aby poskytovala konzistentní přístup k používání EF Core v aplikacích na straně Blazor serveru. Ve výchozím nastavení je doporučeno použití jednoho kontextu pro každou operaci. Kontextem je navrženo rychlé vytváření instancí s nízkou režií:

```
using var context = new MyContext();
return await context.MyEntities.ToListAsync();
```

Pokud chcete zabránit více souběžným operacím, je doporučeno použití příznaku:

```
if (Loading)
{
    return;
}

try
{
    Loading = true;
}
finally
{
    Loading = false;
}
```

Operace jsou umístěny za řádek `Loading = true;` v bloku `try`. Logika načítání nevyžaduje uzamčení záznamů databáze, protože zabezpečení vláken není považováno za problém. Logika načítání je používána k zakázání ovládacích prvků uživatelského rozhraní, aby uživatelé neúmyslně nevybírali tlačítka nebo aktualizovali pole při načítání dat. Pokud existuje nějaká šance, že více vláken může přistupovat ke stejnému bloku kódu, je doporučeno vložit továrnu a vytvořit novou instanci na operaci. V opačném případě je vkládání a používání kontextu obvykle považováno za dostatečné. U délších operací EF Core, které využívají sledování změn nebo řízení souběžnosti, je doporučeno rozsah kontextu na dobu životnosti komponenty. [30]

Co se týče aktualizace uživatelského rozhraní, je důležité zdůraznit, že aktualizace UI by měly být prováděny s ohledem na aktuální stav načítání dat. To znamená, že ovládací prvky uživatelského rozhraní by měly být zakázány nebo povoleny v závislosti na tom, zda jsou data právě načítána. Toto je klíčové pro zabránění neúmyslného výběru tlačítek nebo aktualizace polí uživatelem během načítání dat.

Pokud jde o CSS, může být také užitečné využít dynamické styly nebo třídy, které reagují na stav načítání dat. Například, můžete mít CSS třídu, která zakáže určité ovládací prvky nebo je udělá méně viditelnými během načítání dat. Tímto způsobem můžete zajistit, že

uživatelské rozhraní je vždy v souladu se stavem načítání dat a poskytuje uživatelům jasné vizuální zpětné vazby.

4.4.1 Tailwind CSS

Tailwind CSS je utility-first CSS framework, který je navržen tak, aby umožnil vývojářům rychle a efektivně vytvářet uživatelské rozhraní. Místo toho, aby byly psány vlastní styly pro každou komponentu, je poskytnuta sada malých, znovupoužitelných tříd, které mohou být kombinovány přímo v HTML kódu.

```
<button class="w-10 h-9 flex lg:hidden justify-center
        items-center rounded-md transition-all
        hover:bg-white hover:bg-opacity-10 duration-150">
```

Obrázek 4: Tailwind classes

Fungování Tailwind CSS spočívá v tom, že jsou skenovány všechny soubory, které musí být nastaveny v konfiguračním souboru `tailwind.config.js` v sekci `content`. Následně jsou generovány odpovídající styly a ty jsou zapsány do statického CSS souboru. To znamená, že je možné vytvářet komplexní designy bez nutnosti psát velké množství vlastního CSS kódu.

Pro instalaci Tailwind CSS je nejjednodušší a nejrychlejší způsob použití nástroje Tailwind CLI s použitím Node.js a npm. Po spuštění příkazů `npm install -D tailwindcss` a `npx tailwindcss init` je Tailwind nainstalován do projektu a je vytvořen konfigurační soubor.

```
module.exports = {
    content: ["./src/**/*.{html,js}"],
    theme: {
      extend: {},
    },
    plugins: [],
  }
```

Obrázek 5: Tailwind konfigurační soubor

Tailwind lze také využít v Blazor webové aplikaci, ale v tomto případě se styly nesestaví automaticky. Je nutné to udělat pomocí příkazu, který sleduje změny v souborech a podle toho sestaví všechny třídy do jednoho CSS souboru. To znamená, že při nasazení aplikace není nutné mít Node.js na straně serveru.

5 HOSTING

Webhosting je online služba, díky které je obsah vašich webových stránek publikován na internetu. Když je zakoupen hostingový plán, je pronajímán prostor na fyzickém serveru, kam jsou ukládány všechny soubory webových stránek a jejich obsah dat.

Hostingové technologie a zdroje potřebné pro efektivní a bezpečný provoz vašich webových stránek jsou poskytovány webovými hostiteli. Jsou odpovědní za udržování provozu serveru, provádění bezpečnostních opatření hostingu a zajištění úspěšného přenosu dat, jako jsou texty, fotografie a další soubory, do prohlížečů návštěvníků. [31]

Když je uvažováno o hostování webových stránek nebo aplikací, připadají v úvahu dvě hlavní možnosti: cloudový hosting a self-hosting. Cloud hostingem je využívána síť vzdálených serverů umístěných na internetu k ukládání, správě a zpracování dat, nikoliv na místním serveru nebo osobním počítači. Naproti tomu self-hostingem je myšleno hostování webových stránek nebo aplikací na vlastním serveru, který je vlastněn, spravován a udržován uživatelem nebo organizací.

Termínem "cloud hosting" je označován typ webhostingu, při kterém jsou webové stránky nebo aplikace umístěny v síti vzdálených serverů, běžně označované jako cloud. Na tyto servery je dohlíženo a jsou udržovány poskytovateli třetích stran, jako jsou Google Cloud, Microsoft Azure nebo AWS.

Vlastním hostingem je označováno hostování webové stránky nebo aplikace na serveru, který je vlastněn a spravován např. firmou. Může se jednat o vyhrazené servery nebo virtuální privátní servery (VPS). [32]

5.1 Hostování v cloudu

Podle společnosti IBM je cloud hostingem zpřístupňováno aplikací a webových stránek pomocí cloud zdrojů. Na rozdíl od tradičního hostingu nejsou řešení nasazována na jediném serveru. Místo toho je využívána síť propojených virtuálních a fyzických cloud serverů, která hostí aplikaci nebo webovou stránku, což zajišťuje větší efektivitu, flexibilitu a škálovatelnost. [32]

5.1.1 Výhody cloudového hostingu

Mezi výhody cloudového hostingu patří:

- Škálovatelnost: Cloudovým hostingem je umožňováno uživatelům rychle rozšiřovat nebo snižovat jejich zdroje podle potřeby, aniž by museli investovat do nového hardwaru nebo infrastruktury.

- Spolehlivost: Poskytovateli cloudového hostingu jsou často využívána více datových center a jsou používány pokročilé redundantní a zálohovací systémy, aby byla data vždy dostupná a přístupná.

- Nákladová efektivita: Cloudový hosting může být nákladově efektivnější než tradiční hostingové možnosti, protože uživateli jsou placeny pouze cloudové služby a zdroje, které využívají, a nemusí být investováno do drahého hardwaru nebo infrastruktury.

Mezi oblíbené poskytovatele cloudového hostingu patří například Amazon Web Services, Google Cloud, Microsoft Azure a DigitalOcean. [32]

5.2 Self-hosting

Self-hostováním je myšlen způsob, jak jsou provozovány vlastní webové stránky/aplikace zřízením vlastního serveru a sítě, obvykle doma, a jak jsou tím pádem hostována vlastní data a služby samotnými uživateli místo toho, aby bylo spoléháno výhradně na třetí strany. Těmi, kdo se rozhodnou pro self-hosting, místo využití platformy jako služby nebo poskytovatele veřejného cloudu bude provozována vlastní síť a kromě budování jejich webových stránek nebo aplikace bude pečováno i o údržbu a provozuschopnost.

Jednoduše řečeno se jedná o hostování webové stránky nebo aplikace na serveru, který je vlastněn a spravován jednotlivcem nebo firmou. Může se jednat o dedikované servery nebo virtuální privátní servery (VPS). [32]

5.2.1 Výhody vlastního hostování

Mezi výhody vlastního hostování patří:

- Kontrola: self-hostováním je poskytována uživatelům plná kontrola nad jejich serverem a umožňuje jim přizpůsobit a nakonfigurovat prostředí podle jejich konkrétních potřeb.

- Flexibilita: Hostitelský server je velmi flexibilní, protože může být využíván i na jiných serverech. Ve srovnání s cloudovým hostingem umožňuje vlastní hostování uživatelům vybrat si operační systém, software a hardware, což jim poskytuje větší flexibilitu. To znamená, že si uživatelé mohou vybrat ty nejlepší nástroje pro své konkrétní potřeby.

- Náklady: self-hostování může být levnější než cloud hosting pro podniky, které nepotřebují mnoho zdrojů, protože je placeno pouze za server a internetové připojení, nikoliv měsíční poplatek za zdroje.

Některé příklady možností vlastního hostování zahrnují:

- Dedikované servery
- Virtuální privátní servery (VPS)
- Domácí servery [32]

II. PRAKTICKÁ ČÁST

6 PLÁNOVÁNÍ

Před začátkem vývoje webové aplikace pro správu hesel bylo provedeno důkladné plánování celého projektu. Plánování zahrnovalo několik kroků, které pomohly strukturovat problém, stanovit požadavky, zvolit vhodné technologie a navrhnout architekturu celého řešení.

Nejprve bylo nutné rozdělit hlavní problém na menší podproblémy a identifikovat klíčové funkční celky aplikace. Tento krok umožnil lépe pochopit rozsah projektu a usnadnil řešení dílčích částí.

Původně měla být prováděno autentizace a autorizace pomocí JWT, ale z tohoto plánu se muselo sejít. Důvodem je nedostatečná kapacita znalostí pro implementaci takového systému. Není problém vygenerovat token, ale spravovat autorizaci, aby uživatele vůbec pustila do aplikace. Po velkém snažení a mnoha různých způsobech, bylo tedy rozhodnuto použít cookies.

6.1 Rozdělení podproblémů

Jako hlavní funkční celek byla identifikována správa uživatelů. Ta zahrnuje registraci nových uživatelů, při které bude potřeba zachytit údaje jako e-mail, heslo a jiné informace o uživateli. Dále pak samozřejmě přihlašování a odhlašování uživatelů, úpravu uživatelských profilů a možnost změny hesla v případě potřeby.

Nedílnou součástí správy uživatelů je autentizace a autorizace. Zde bude klíčová implementace ověřování identity uživatelů pomocí e-mailu a hesla. Autorizace by neměla dovolit přístup do aplikace kromě registračního a přihlašovacího formuláře.

Jako další hlavní funkční celek byla stanovena samotná správa hesel. Ta bude zahrnovat vytváření, úpravu a mazání přihlašovacích údajů, generování bezpečných hesel a ukládání hesel v zašifrované podobě v databázi. Nesmíme opomenout ani fulltextové vyhledávání a filtrování hesel.

Velmi důležitou částí aplikace bude sdílení hesel mezi uživateli. Zde půjde o vytváření skupin pro sdílení, správu členů skupin (přidávání, odebírání), určení vlastníka neboli

Ownera každé skupiny a následně samotné sdílení vybraných hesel mezi členy. Členské role budou moci pouze číst nebo sdílet vlastní přihlašovací údaje.

Nedílnou součástí je samozřejmě uživatelské rozhraní. To musí být responsivní pro různá zařízení, musí poskytovat přehledný výpis a tlačítka, které budou otevírat vyskakovací okna. Veškeré akce by měly být odděleny do modálních nebo vyskakovacích oken, celkově by UI mělo být intuitivní pro ovládání a snadnou navigaci.

Tímto rozdělením na menší podproblémy byl získán přehled o jejich vzájemných vazbách a závislostech. Bude možné systematicky postupovat implementací po částech, kdy některé budou stavět na jiných.

6.2 Výběr technologií

Pro vývoj webové aplikace pro správu hesel bylo rozhodnuto o použití ASP.NET Core 8 s Blazorem. V Blazoru jsou zahrnuty jak webový server pro zpracování HTTP požadavků a generování stránek, tak klientskou část pro vývoj uživatelského rozhraní.

Blazor, moderní webový framework postavený na ASP.NET Core, umožňuje vyvíjet interaktivní uživatelská rozhraní použitím jazyka C#, ale umožňuje taky práci s JavaScriptem. To bylo považováno za velkou výhodu, jelikož jsou mány bohatší zkušenosti s C# a .NET ekosystémem oproti jiným technologiím. Blazorem je tedy umožněno zužitkovat stávající znalosti a dovednosti.

Aplikace bude provozována na straně serveru, konkrétně tedy půjde o Blazor web app s renderováním na serveru. Pro Blazor přineslo ASP.NET Core 8 řadu vylepšení, která zjednodušují vývoj a nasazení těchto aplikací. Mezi klíčové novinky patří například interaktivní server-side rendering (SSR). Tím je poskytováno plnohodnotné interaktivní uživatelské rozhraní srovnatelné s klasickými klientskými JS aplikacemi SPA. Velkou výhodou ale je, že veškerá aplikační logika i zpracování uživatelských akcí běží na straně serveru. Není tedy potřeba implementovat samostatná REST API, vše je obstaráváno přímo serverovou částí aplikace.

Pro práci s daty a ukládáním hesel do databáze bude využíván objektově-relační mapovací nástroj Entity Framework Core. EF Core podporuje širokou škálu databázových enginů, jedním z nich je zvolený Microsoft SQL Server jako primární úložiště dat.

ASP.NET Core Identity je také součástí plánu pro vývoj aplikace. Jedná se o knihovnu, která poskytuje funkce pro autentizaci a autorizaci uživatelů. ASP.NET Core Identity umožňuje snadno implementovat standardní bezpečnostní funkce, jako je registrace, přihlašování, změna hesla a ověřování e-mailu.

Důležitou součástí ASP.NET Core Identity je, že využívá EF Core. To znamená, že všechny informace o uživatelích a jejich rolích jsou uloženy v databázi a spravovány pomocí EF Core.

Takže stručný popis technologií by mohl vypadat takto:

- **ASP.NET Core 8** - Jako primární webový framework pro vývoj celé aplikace.
- **Blazor Server** - Pro vývoj uživatelského rozhraní pomocí C# a občasného interaktivního SSR (per component).
- **Entity Framework Core** - Jako ORM pro přístup k databázi a správu dat.
- **ASP.NET Core Identity** - Pro správu autentizace a autorizace uživatelů.
- **Microsoft SQL Server** - Jako databázový engine pro ukládání dat. [33][34][35]

7 ARCHITEKTURA APLIKACE

Design patterns a techniky používané při návrhu a vytváření aplikace jsou popsány architekturou aplikace. Plán a osvědčené postupy, které je třeba dodržovat při vytváření aplikace, poskytuje architektura, aby byla nakonec vytvořena dobře strukturovaná aplikace.

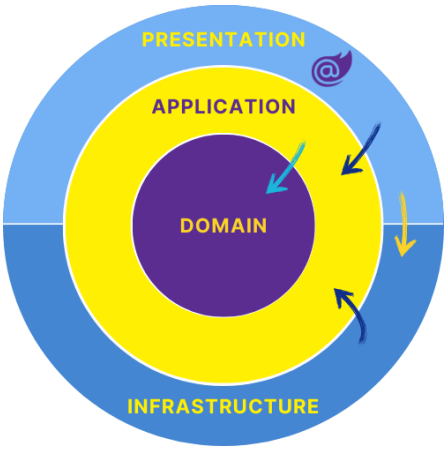
Při vytváření aplikace mohou být nápomocny design patterns softwaru, které jsou popsány vzory a návrhy řešení problému.

Služby front-endu i back-endu budou součástí architektury aplikace. Uživatelským prostředím aplikace se zabývá vývoj front-endu, zatímco přístup k datům, službám a dalším existujícím systémům, díky nimž aplikace funguje, je zaměřením vývoje back-endu. [36]

Pro vývoj webové aplikace pro správu hesel byl zvolen architektonický vzor Clean Architecture.

7.1 Clean architecture

Tento přístup klade důraz na oddělení součástí aplikace do logických vrstev podle jejich zodpovědností a minimalizaci vzájemných závislostí mezi vrstvami. U Clean Architecture je směřováním jednotlivých vrstev dovnitř zajišťováno oddělení zájmů. [37]



Obrázek 6: Koncept Clean Architecture

7.1.1 Prezentační vrstva (Web)

- Obsahuje Blazor komponenty pro vykreslení uživatelského rozhraní

- Tato vrstva má referenci na vrstvy Application a Infrastructure

7.1.2 Aplikační vrstva (Application)

- Implementuje business logiku aplikace jako služby (services)
- Má referenci pouze na vrstvu Domain

7.1.3 Doménová vrstva (Domain)

- Modeluje klíčové entity a koncepty aplikace (ApplicationUser, Vault apod.)
- Nemá žádné externí závislosti

7.1.4 Infrastrukturní vrstva (Infrastructure)

- Kód pro přístup k externím zdrojům (databáze, ověřování, externí služby atd.)
- Implementace repozitářů pro práci s daty
- Má referenci pouze na vrstvu Application

Clean Architecture je založena na striktním dodržování referenční integrity mezi vrstvami. Na vnějších vrstvách jsou vnitřní vrstvy, jako jsou doménová a aplikační vrstva, udržovány zcela nezávislé. [38]

Pouze prostřednictvím veřejných rozhraní, nikoli přímými referencemi, je umožněna komunikace mezi vrstvami. Services z aplikační vrstvy jsou volány prezentační vrstvou, které interně využívají repozitáře z infrastrukturní vrstvy k přístupu k datům. [39]

Vzor Dependency Injection je v aplikaci použit pro řízení životního cyklu objektů, vkládání závislostí a propojování různých komponent. ASP.NET Core obsahuje vestavěný kontejner IoC, který je konfigurován v programové části aplikace. Ale v rámci dodržení principů Clean Architecture jsou definovány třídy, které budou zaregistrovány v kontejneru, a také je specifikována jejich životnost (Singleton, Scoped, Transient). K tomu je ve vrst-

vách Infrastructure a Application, a jiných, když je potřeba, vytvořena třída DependencyInjection, která vrací IServiceCollection. [40]

```
public static class DependencyInjection
{
    public static IServiceCollection AddApplication(this IServiceCollection services)
    {
        services.AddScoped<IPasswordGeneratorService, PasswordGeneratorService>();
        services.AddScoped<IIdentityService, IdentityService>();

        return services;
    }
}
```

Obrázek 7: Registrace services v aplikační vrstvě

Tato třída obsahuje metodu AddApplication, která rozšiřuje IServiceCollection a registruje services v kontejneru. Metoda AddApplication umožňuje registraci služeb, jako jsou IPasswordGeneratorService a IIdentityService, a jejich implementace PasswordGeneratorService a IdentityService do kontejneru. Tyto služby jsou poté dostupné pro vkládání závislostí v rámci celé aplikace, když se zaregistrují v programové části aplikace:

```
builder.Services
    .AddApplication()
    .AddInfrastructure(builder.Configuration);
```

Obrázek 8: Registrace service collection z různých vrstev

Díky použití Dependency Injection mohou být v konstruktorech tříd deklarovány závislosti na jiných službách, které budou automaticky vloženy kontejnerem při vytváření instance. Toto usnadňuje testování, neboť v jednotkových testech mohou být závislé objekty nahrazeny falešnými implementacemi (Mock).

8 PŘÍPRAVA

Je nutné mít nainstalované .NET 8 SDK, ať už pro vytvoření samotné aplikace, nebo pro její spuštění, se kterou by se měla nainstalovat i sada ASP.NET Core a její templaty, včetně těch, týkajících se Blazoru. To je možné zjistit příkazem v terminálu dotnet –info, kde by pod sekcí .NET runtimes installed, měly být uvedeny AspNetCore.app a NETCore.App.

```
.NET runtimes installed:
  Microsoft.AspNetCore.App 8.0.4 [/usr/share/dotnet/shared/Microsoft.AspNetCore.App]
  Microsoft.NETCore.App 8.0.4 [/usr/share/dotnet/shared/Microsoft.NETCore.App]
```

Obrázek 9: Úspěšně nainstalovaný ASP.NET Core

Po instalaci SDK je zapotřebí nainstalovat nástroje Entity Frameworku, pomocí příkazu v terminálu dotnet tool install --global dotnet-ef se nainstaluje CLI nástroj pro vytváření migrací, jejich aplikování a generování kódu založené na modelech, pro vytvoření databáze a tabulek. Úspěšnou instalaci nástroje dotnet ef je možné ověřit opět v terminálu. Měl by se vykreslit obrazec jednorožce s dalšími textovými informacemi.

Obrázek 10: Úspěšně nainstalovaný nástroj dotnet ef

Dále bude zapotřebí SQL Server, který si je možné nainstalovat přímo lokálně na svůj počítač, ale pro tento projekt bude použita Docker image SQL Serveru, kvůli udržitelnosti, stabilitě a taky bez nutnosti další lokální instalace. Pro jednoduší manipulaci s image a kontejnerem je možné stáhnout desktop aplikaci Docker Desktop. V aplikaci lze vyhledat image mssql/server. Pullnutím image mcr.microsoft.com/mssql/server:2022-latest se image stáhne a po prvním spuštěním bude nutné zadat environmentální proměnné. Konkrétně jsou vyžadovány:

- ACCEPT_EULA s hodnotou Y, což je odsouhlasení licenčních podmínek
- MSSQL_SA_PASSWORD se zvoleným libovolným heslem pro administrátorský účet „sa“.

Až budou tyto proměnné nastaveny a kontejner bude běžet, bude možné se k SQL Serveru připojit.

8.1 Založení projektu

Celý projekt bude vytvářen v .NET IDE od JetBrains, a sice Rideru. Nicméně možnost použití Visual Studia je stále otevřená, s tím, že postup práce zůstává nezměněný.

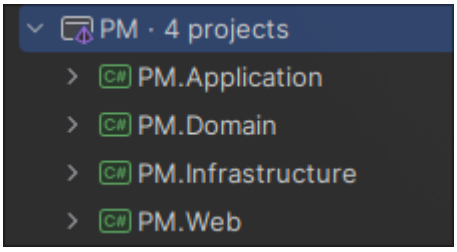
Na úvodní projektové obrazovce se vytvoří nový projekt typu Web. Po zadání názvu pro solution a projekt s příponou .Web, tak jako je uvedeno v konceptu Clean Architecture, se je potřeba ujistit, že není zakliklý checkbox s popiskem „Put solution and project in the same directory“. Jako Target framework se zvolí nainstalovaný net8.0 z SDK 8.0. Z templatů se nabízí dva konkrétní pro Blazor. Pokud se nenabízí žádný Blazor template, je možné tyhle templaty v Rideru doinstalovat.

Jedním z templatů je WebAssembly Standalone App. Taková Blazor aplikace funguje tak, že její závislosti a .NET runtime jsou staženy do prohlížeče a aplikace je spuštěna přímo na vlákně uživatelského rozhraní prohlížeče. Veškeré aktualizace uživatelského rozhraní a obsluha událostí probíhají v rámci stejného procesu. [42]

Ve zkratce, WebAssembly aplikace běží na straně klienta. Ale pro potřeby správce hesel bude mnohem vhodnější i jednodušší použít runtime na straně serveru. Tudíž zvolen bude template Blazor Web App. V dalším nastavení při vytváření projektu je nutné zvolit možnost Per page/component pro konfiguraci Interactivity location.

8.1.1 Architektura

Se založeným projektem mimo solution je možné přidat další projekty, které se propojí přidáním reference. V tuhle chvíli existuje pouze projekt .Web. Pravým kliknutím na solution je možné přidat další projekty typu Class Library. Dle konceptu se přidají projekty .Application, .Domain a .Infrastructure.



Obrázek 11: Clean Architecture v projektu

8.1.2 Reference

Po rozbalení projektu se naskytne možnost Dependencies. Kliknutím pravým tlačítkem myši vyskočí nabídka s možností Reference. Tímto způsobem je možné přidat reference z daného projektu na druhý daný projekt, opět dle konceptu. Ve zkratce, Web má referenci na Infrastructure, Infrastructure má referenci na Application a ten má referenci na Domain. Tímto způsobem je zvětší části docíleno Clean Architekture.

8.1.3 Nugety

V prostředí Rideru lze nugety snadno nainstalovat pomocí ikony v postranním panelu, kde stačí vyhledat požadované nugety a nainstalovat je do příslušných projektů. Nugety jsou v podstatě balíčky, které obsahují předkompilovaný kód a umožňují vývojářům snadno přidávat funkcionality do svých aplikací, aniž by museli tento kód psát sami. V tomto konkrétním případě budou potřeba nugety pro Entity Framework Core. Ty umožní pracovat s databázemi a implementovat správu identit. Konkrétně bude potřeba následujících nuget:

1. Microsoft.EntityFrameworkCore: S tímto balíčkem je poskytnuto Entity Frameworku rozhraní pro práci s databázemi.

2. Microsoft.EntityFrameworkCore.SqlServer: SQL Server se používá jako databázový engine, nebo driver, pro Entity Framework.

3. Microsoft.AspNetCore.Identity.EntityFrameworkCore: Služby pro správu identit jsou zpřístupněny pomocí této nugety v ASP.NET Core aplikacích.

4. Microsoft.AspNetCore.Identity.UI: S Identity baličkem je možné používat typy pro integrované uživatelské rozhraní Razor Pages a pro Identitu v ASP.NET Core.

5. Microsoft.EntityFrameworkCore.Design: Tahle nugeta je vyžadována pro nástroj dotnet ef. Umožňuje práci s migracemi databáze v Entity Frameworku.

První 4 nugety budou nainstalovány v projektu Infrastructure, protože tento projekt bude obsahovat kód pro práci s databází a správu identit. 3. a 4. nugety budou rovněž vyžadovány v projektu Domain, který bude obsahovat entity a modely pro správu identit. Poslední nuget (5) bude nainstalován v projektu Web, který reprezentuje samotnou webovou aplikaci, kde budou prováděny migrace databáze.

8.2 Návrh databáze

V C# lze udělat návrh databáze pomocí modelů, třídy s atributy představující tabulku a sloupce.

8.2.1 Entity a datové třídy

Objekt reálného světa, který je relevantní pro danou aplikační doménu a jehož data jsou žádoucí ukládat do databáze trvale, je představen jako entita. V objektově orientovaném návrhu aplikace jsou datové třídy přesně reprezentovány jako entity v kódu aplikace.

1. ApplicationUser

V aplikaci je entita ApplicationUser reprezentována pro uživatele. Vychází z třídy IdentityUser<Guid>, která je součástí balíčku Microsoft.AspNetCore.Identity a poskytuje základní funkcionality pro správu uživatelských účtů, ověřování, autorizaci atd. Hlavní výhodou je, že tato funkcionalita nemusí být implementována samostatně.

Atributy jako UserName, Email, PasswordHash a další, které jsou potřebné pro uživatelské účty, jsou obsaženy v třídě IdentityUser. Přetypováním na Guid specifikujeme, že jako primární klíč bude použito GUID, což je 128bitové celé číslo, které lze použít tam, kde je vyžadován jedinečný identifikátor. Takový identifikátor je charakterizován velmi nízkou pravděpodobností duplicity.

Kromě děděných atributů musí obsahovat entita ApplicationUser atribut UserName, podmínkou třídy IdentityUser je, že UserName nesmí být prázdným řetězcem, a dva navigační atributy:

- Vaults (List<Vault>): Představuje vztah 1:N k entitě Vault. To znamená, že jeden uživatel může mít přiřazeno více trezorů pro uložení citlivých dat.

- UserRoles (List<GroupUserRole>): Reprezentuje vztah M:N mezi uživatelem a skupinami (Group) přes vazební entitu GroupUserRole. Díky tomu může být jeden uživatel členem více skupin a jedna skupina může obsahovat více uživatelů.

```
public class ApplicationUser : IdentityUser<Guid>
{
    public override string? UserName { get; set; }

    public List<Vault> Vaults { get; set; } = [];
    public List<GroupUserRole> UserRoles { get; set; } = [];
}
```

Obrázek 12: Třída ApplicationUser představují uživatele

Toto umožňuje přiřazovat uživatelům různé role v rámci různých skupin, což je běžný přístup pro implementaci oprávnění a řízení přístupu v aplikacích.

2. Vault

Koncept trezoru pro bezpečné uložení citlivých informací, jako jsou uživatelská jména a hesla, je představován entitou Vault. Každá instance této třídy reprezentuje jeden konkrétní trezor.
Vault je identifikován svým unikátním identifikačním číslem Id typu Guid. Dále jsou obsaženy další atributy popisující obsah trezoru - Username, Password, Title a Websites (seznam webových stránek, ke kterým se přihlašovací údaje vztahují). Atribut IsDeleted je využívána k označení, zda je trezor označen ke smazání. Data poslední modifikace a vytvoření jsou používány k zaznamenání časových údajů o změnách trezoru.

Důležité jsou také navigační atributy reprezentující vztahy k jiným entitám:

- ApplicationUserId a ApplicationUser reprezentují cizí klíč mezi trezorem a uživatelem aplikace (ApplicationUser).

- Groups pak představuje vztah 1:N mezi trezorem a skupinami (Group). Jeden trezor může být sdílen mezi více skupinami.

```
public class Vault
{
    public Guid Id { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
    public string Title { get; set; }
    public string? Websites { get; set; }
    public bool IsSelected { get; set; }
    public bool IsDeleted { get; set; }
    public DateTime LastModified { get; set; }
    public DateTime CreatedAt { get; set; }

    public Guid ApplicationUserId { get; set; }
    public ApplicationUser ApplicationUser { get; set; }

    public List<Group> Groups { get; set; } = [];
}
```

Obrázek 13: Třída Vault reprezentující trezor

Tato entita je tedy modelována jako klíčový koncept pro uchovávání a správu citlivých přihlašovacích údajů v aplikaci, přičemž propojuje tyto údaje s uživateli aplikace a umožňuje sdílení mezi skupinami uživatelů.

3. Group

Entita Group je reprezentována jako koncept skupiny uživatelů v rámci aplikace. Skupina je identifikována svým unikátním identifikačním číslem Id typu Guid, které ji jednoznačně identifikuje. Dále je obsažen atribut Title, která určuje název dané skupiny. Klíčové jsou však navigační atributy reprezentující vztahy k jiným entitám: Atribut Members je typu List<GroupUserRole> a reprezentuje vztah M:N mezi skupinou a uživateli aplikace (ApplicationUser). Tento vztah je realizován přes vazební entitu GroupUserRole. To umožňuje, aby jedna skupina mohla mít více členů (uživatelů) a zároveň aby jeden uživatel mohl být členem více skupin. Atribut Vaults je typu List<Vault> a reprezentuje vztah 1:N mezi skupinou a trezory (Vault). To znamená, že ke každé skupině může být přiřazeno více trezorů, ale každý konkrétní trezor patří vždy právě jedné skupině.

```
public class Group
{
    public Guid Id { get; set; }
    public string Title { get; set; }

    public List<GroupUserRole> Members { get; set; } = [];
    public List<Vault> Vaults { get; set; } = [];
}
```

Obrázek 14: Třida Group reprezentující skupinu

Tato entita je tedy modelována jako koncept uživatelských skupin, členství uživatelů v těchto skupinách a také přiřazení trezorů ke skupinám. Je klíčová pro implementaci sdílení přístupu k citlivým datům mezi uživateli patřícími do stejné skupiny.

4. GroupUserRole

Třída GroupUserRole je reprezentována jako vazební entita, která umožňuje realizovat vztah M:N mezi entitami Group (skupina) a ApplicationUser (uživatel aplikace). Její hlavní úlohou je propojit uživatele se skupinami a definovat role, které uživatel v dané skupině zastává.

Vazební entita je obsazena unikátním identifikátorem Id typu Guid. Dále jsou obsaženy atributy ApplicationUserId typu Guid a ApplicationUser reprezentující vztah k entitě ApplicationUser (uživatelům). Tyto atributy určují, ke kterému konkrétnímu uživateli se záznam vazby vztahuje.

Atribut Role typu Roles určuje roli, kterou daný uživatel ve skupině zastává. Roles je typem enum, nebo vyjmenovaným typem, definovaným ve stejné třídě, který obsahuje dvě možné hodnoty: Owner a Member. Tím je vyjádřeno, zda je uživatel vlastníkem dané skupiny nebo jen jejím běžným členem.

Poslední navigační atribut Group typu Group určuje, ke které konkrétní skupině se vazba vztahuje.

```
public class GroupUserRole
{
    public Guid Id { get; set; }

    public Guid ApplicationUserId { get; set; }
    public ApplicationUser? ApplicationUser { get; set; }
    public Roles Role { get; set; }

    public Group Group { get; set; }
}

public enum Roles
{
    Owner,
    Member
}
```

Obrázek 15: Třída GroupUserRole reprezentující vazební entitu a enum Roles reprezentující uživatelské role ve skupině

Vazební entita tedy v podstatě ukládá záznamy, které dávají do souvislosti uživatele, skupiny a role uživatele v dané skupině. Díky tomu může jeden uživatel patřit do více skupin s různými rolemi, a naopak jedna skupina může mít více členů s odlišnými rolemi.

Tato vazební entita je klíčovým stavebním kamenem pro implementaci členství uživatelů ve skupinách, přiřazování rolí a realizaci vztahu M:N mezi skupinami a uživateli aplikace.

8.3 Návrh UI

Aplikací bude nabízena široká škála funkcí, proto je nezbytné, aby bylo uživatelské rozhraní interaktivní, umožňující plynulou práci s různými prvky.

8.3.1 Interaktivní design

Toho bude dosaženo využitím vyskakovacích oken (modal) a vysouvacích panelů, které se budou otevírat po kliknutí na příslušná tlačítka reprezentována popisky nebo ikonami, které budou jasně popsány nebo vizuálně odlišeny podle své funkce, což zajistí intuitivní ovládání celé aplikace. Tímhle způsobem bude dosažena celková jednoduchost a přehlednost, což vede k dobré uživatelské zkušenosti (UX) a zároveň bude klást důraz na design, UI. Jednotlivé prvky budou rozmístěny a navrhovány tak, aby bylo rozhraní vzdušné, ale zároveň poskytovalo uživateli snadný přístup ke všem potřebným funkcím a informacím.

8.3.2 Stánky a komponenty

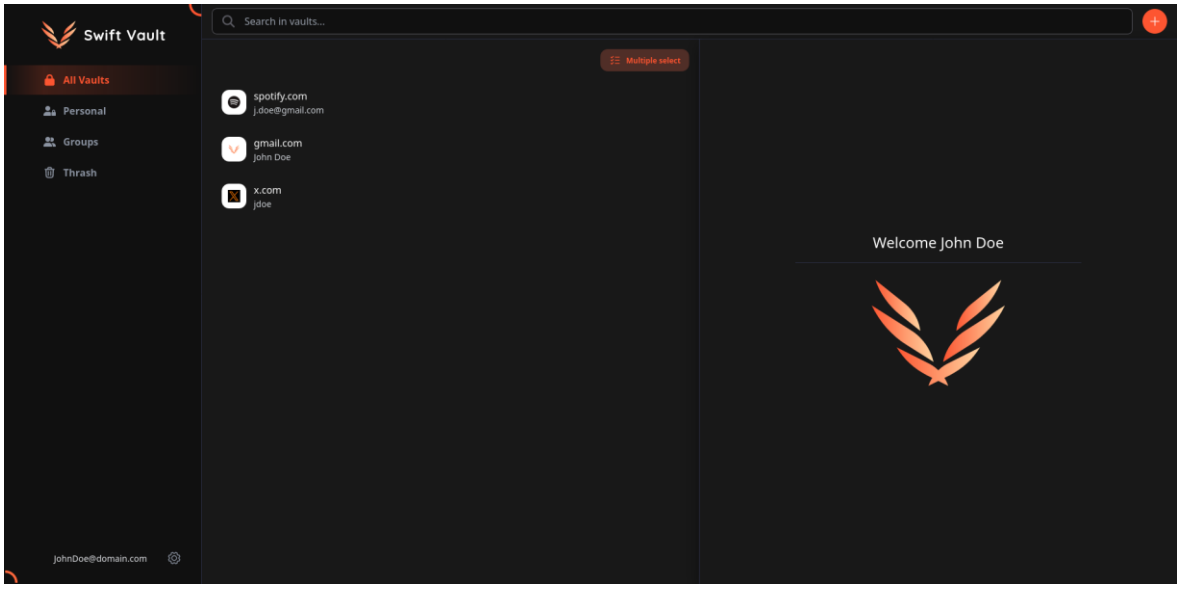
Aplikace bude rozdělena do čtyř hlavních stránek a dvou přihlašovacích, přičemž na každé z nich budou specifické komponenty umožňující provádění různých akcí a interakcí s aplikací. Tyto komponenty budou navrženy s ohledem na uživatelskou přívětivost a srozumitelnost.

8.3.3 Stylování

Stylování aplikace nebude realizováno prostřednictvím klasických CSS stylů, ale bude využit moderní CSS framework Tailwind CSS. Ten usnadňuje vývoj díky svému utility-first přístupu, kdy jsou styly přiřazovány přímo k jednotlivým HTML elementům pomocí předpřipravených utilit namísto vytváření složitých kaskádových stylů.

8.3.4 Barevný motiv

Za pozornost taky stojí výběr vhodné barevné palety, která bude ladit s celkovým designem aplikace a zároveň bude příjemná pro oči uživatelů. Barvy budou účelně využity pro odlišení různých prvků rozhraní, jako jsou tlačítka nebo textová pole, čímž se opět zvýší uživatelská přívětivost a zážitek. Moderní webové aplikace často umožňují přizpůsobení motivu, světlého nebo tmavého, podle systémových nastavení uživatele. V případě této aplikace však bude použito pouze jedno barevné schéma - tmavý motiv. Důvodem je lepší čitelnost a nižší úroveň oslnění při delším používání obrazovky počítačů nebo mobilních zařízení. Tmavé motivy obecně méně zatěžují zrak uživatele a snižují únavy očí. Pro případ, že by uživatel používal aplikaci za přímého slunečního svitu, kdy by standardní jas nemusel stačit pro dostatečnou čitelnost obsahu, tak bude možné zvýšit jas displeje k plné čitelnosti.



Obrázek 16: Design úvodní stránky

9 IMPLEMENTACE

S nainstalovanými vývojovými nástroji, požadovanými sadami SDK a dokončeným návrhem aplikace může začít samotná fáze implementace. Ta zahrnuje několik klíčových součástí - vývoj frontendu včetně uživatelského rozhraní, implementaci backendu pro zpracování aplikační logiky, vytvoření a propojení s databází pro ukládání dat a v neposlední řadě také pochopení základních principů a fungování využitých technologií a knihoven.

9.1 Propojení s MSSQL Databází a Entity Framework Core

Před samotným používáním databáze je nutné nastavit Connection String a související konfigurace databázového kontextu. DbContext je klíčovou součástí Entity Framework Core, která je zapouzdřována datovým kontextem aplikace a poskytuje rozhraní pro práci s databází. V této aplikaci je použit IdentityDbContext, který je rozšířením základního DbContextu o podporu správy uživatelských identit.

V projektu Infrastructure je vhodné vytvořit složku s názvem Database nebo Context, ve které bude umístěna třída s odpovídajícím názvem, reprezentující databázový kontext. Tato třída bude dědit z IdentityDbContext<TUser, TRole, TKey>. TUser je entita ApplicationUser reprezentující uživatele, TRole je výchozí role z Identity systému a TKey je datový typ použitý pro primární klíče. Vytvořená třída databázového kontextu vyžaduje parametr DbContextOptions, který je předáván konstruktoru třídy IdentityDbContext. V jazyce C# 12 bylo představeno zjednodušení syntaxe pro konstruktory, tzv. „primary constructors“. Díky tomu lze deklarovat konstruktor na úrovni definice třídy a parametry jsou dostupné kdekoliv v těle třídy. Každý model entity, na který je potřeba se dotazovat z databáze, musí být přidán do kontextu pomocí veřejné vlastnosti DbSet. Tím se entity začlení do databázového kontextu a budou moci být použity pro dotazování, vkládání, aktualizaci a mazání. Zároveň se tímto mapují odpovídající databázové tabulky.

```
public class MyDbContext(DbContextOptions<MyDbContext> options)
    : IdentityDbContext<ApplicationUser, IdentityRole<Guid>, Guid>(options)
{
    public DbSet<Group> Groups { get; set; }
    public DbSet<GroupUserRole> GroupUserRoles { get; set; }
    public DbSet<Vault> Vaults { get; set; }
}
```

Obrázek 17: Databázový kontext

V konfiguračním souboru appsettings.json jsou zapsána výchozí nastavení a Connection String k databázi. Je nutné mít na paměti, že tento soubor je veřejně přístupný, pokud je aplikace hostována. Proto by neměl obsahovat citlivé údaje, jako jsou hesla. Pro tyto účely je vhodnější využít nástroj Secret Manager. Nicméně v režimu vývoje na lokálním počítači není nutné vytvářet user secrets.

```
{
  "ConnectionStrings": {
    "Default": "Server=localhost;Database=PM;User Id=sa;Password=5Ab*cD9E;TrustServerCertificate=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Obrázek 18: Connection String pro připojení k databázovému serveru

Nakonec je nutné databázový kontext zaregistrovat jako service, aby bylo možné jej použít způsobem Dependency Injection v dalších services a repositories aplikace. Pro dodržení Clean Architecture se v projektu vytvoří třída DependencyInjection, která kontext zaregistruje.

V tuhle chvíli je nutné získat Connection String z konfiguračního souboru. Proto se do parametru metody přidá další argument, a sice IConfiguration, který se předá v programové části aplikace, jak je ukázáno na obrázku Registrace service collection**.**

```
public static class DependencyInjection
{
    public static IServiceCollection AddInfrastructure(this IServiceCollection services, IConfiguration configuration)
    {
        services.AddDbContextFactory<MyDbContext>(options =>
            options.UseSqlServer(configuration.GetConnectionString("Default")));

        return services;
    }
}
```

Obrázek 19: Registrace databázového kontextu

9.1.1 Vytvoření databáze

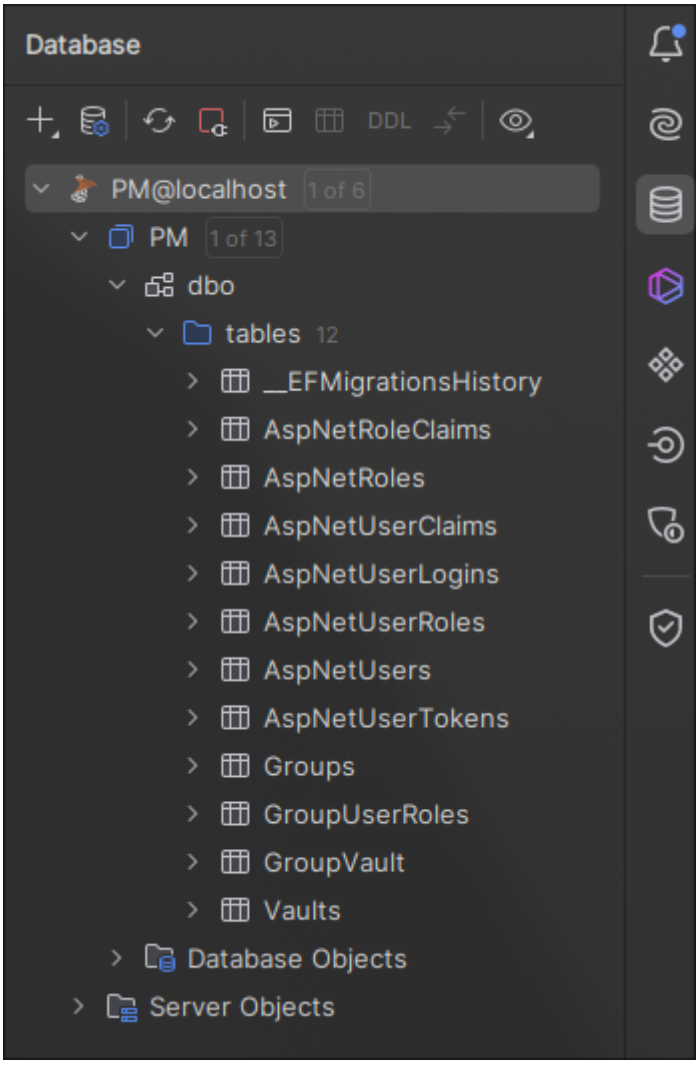
Vytváření a správa databáze v Entity Frameworku je značně zjednodušena. Databáze není nutné vytvářet ručně, o to se postará samotný EF Core. V projektu Domain jsou nadefinovány modely entit, reprezentující tabulky v databázi. Tyto modely jsou seskupeny v databázovém kontextu. Pro vytvoření databázových tabulek a dalších objektů podle definovaných modelů se používají migrace databáze, vytvořené nástrojem dotnet ef. Existují jistá pravidla, jak se tento příkaz spouští, ale nejjednodušší je se v terminálu přepnout do projektu Infrastructure a spustit dotnet ef odsud, jelikož kromě projektu s DbContextem je vyžadován také startup projekt, kterým je Web. Tím pádem konkrétní příkaz pro vygenerování migračního skriptu je:

```
dotnet ef migrations add "název migrace" -s ../NázevProjektu.Web
```

Po vygenerování migračního skriptu je třeba migraci aplikovat na cílovou databázi příkazem:

```
dotnet ef database update -s ../NázevProjektu.Web
```

Právě argument -s (--startup-project) specifikuje, že startup projekt je Web, ačkoliv migrace se provádějí v projektu Infrastructure. Po úspěšném průběhu obou příkazů je databáze vytvořena, včetně tabulek. Databáze, včetně vložených dat, je v tuhle chvíli prázdná a lze ji prohlédnout přímo v Database Exploreru v Rideru. Stačí si zvolit Microsoft SQL Server jako typ databáze a zkopírovat Connection String ze souboru appsettings.json do příslušného pole.



Obrázek 20: Prohlížeč databáse v Rideru

9.2 Integrace s Tailwind CSS

Existuje několik možností, jak přidat Tailwind do projektu. Jednou z tradičních a zároveň lepších metod je použití NodeJS a npm. V terminálu je nutno se přesunout do adresáře projektu Web a spustit příkaz npx tailwindcss init. Tímto se vytvoří konfigurační soubor tailwind.config.js, který bude na začátku prázdný, a proto Tailwind třídy nebudou funkční. V této chvíli závisí na struktuře složek projektu Web. Ve šabloně Blazor Web App jsou layouty, stránky i komponenty ukládány do složky Components. Tuto cestu je nutno nastavit v konfiguračním souboru, v sekci content.

```
module.exports = {
    content: [
        './Components/*.razor',
        './Components/**/*.razor',
        './Components/**/**/*.razor'
    ],
    theme: {
        extend: {},
    },
    plugins: [],
}
```

Obrázek 21: Tailwind config content

Dále je potřeba vytvořit soubor kaskádových stylů. Běžným způsobem je vytvořit složku Styles a soubor stylů v ní. Do tohoto souboru se pak vloží 3 řádky kódu:

```
@tailwind base;

@tailwind components;

@tailwind utilities;
```

Tyto direktivy umožňují vytvářet vlastní třídy a aplikovat na ně Tailwind třídy. Tímto způsobem lze zjednodušit vývoj, jelikož takové třídy jsou opakovatelné a budou mít stejný vzhled na každém elementu, kde se vlastní třída aplikuje.

Tailwind třídy musí být sestaveny do souboru umístěného ve složce wwwroot. Pro automatické sledování změn v Razor Pages je nutné spustit příkaz `npx tailwindcss -i ./Styles/app.css -o ./wwwroot/app.css –watch`, kde `-i` je vstupní soubor, tedy vlastní soubor se styly, a `-o` je výstupní soubor, tedy soubor, kam se Tailwind třídy sestaví. Tento soubor stylů je pak nutné připojit do hlavičky HTML v souboru App.razor.

9.3 Vývoj frontendu v Blazoru

Razor syntaxe je využívána Blazorem pro tvorbu stránek, layoutů a komponent. Klasické HTML a CSS umožňují vytvořit strukturu a design uživatelského rozhraní, ale Razor syntaxe obohacuje UI o reaktivní stav pomocí direktiv a vkládání kódu v jazyce C#. Reaktivní chování je dosahováno v direktivě `@code`, která umožňuje psát C# kód přímo v Razor souboru. Tímto způsobem jsou definovány atributy, metody a zachycovány události, které ovlivňují chování a stav komponenty. Pro sdílení logiky mezi komponentami je poskyto-

vána Blazorem vestavěná podpora pro Dependency Injection. To umožňuje vytvořit třídu reprezentující určitou funkcionalitu (service) a využít ji v libovolné Blazor komponentě pomocí direktivy `@inject`. Každá stránka v Blazoru obsahuje direktivu `@page`, která specifikuje vzor URL pro danou stránku. Tento vzor určuje, pro jaké cesty bude stránka renderována.

9.3.1 Implementace komponent

Namísto komplexních stránek a layoutů je aplikace sestavována z menších, znovu použitelných komponent. Tento přístup je upřednostňován kvůli řadě výhod, které přináší:

- Opakovatelnost: Komponenty jsou využívány na více místech v rámci aplikace, což vede k efektivnějšímu kódování a udržování konzistentního vzhledu a chování.

- Oddělená kódová báze: Kód komponent je oddělen od kódu stránek a layoutů, což umožňuje snadnější údržbu a lepší přehlednost celého projektu.

- Zlepšená čitelnost: Menší komponenty jsou snadněji čitelné a pochopitelné, což usnadňuje vývoj a budoucí rozšiřování aplikace.

V této aplikaci budou implementovány různé typy komponent sloužících k různým účelům:

- Layoutové komponenty: Header a SideNav jsou používány v rámci hlavního layoutu aplikace a sdíleny napříč stránkami.
- Prezentační komponenty: detaily záznamů jsou využívány ke zobrazování a úpravě dat.
- Akční komponenty: generátor hesla, vytvoření trezoru a skupiny jsou využívány k provádění specifických akcí v rámci aplikační logiky.

9.3.2 Komunikace mezi komponenty

Existují tři hlavní způsoby, jak mohou komponenty vzájemně komunikovat:

1. Parent to Child: Tento způsob je nejčastěji používán, když je žádoucí, aby byl dynamický obsah vykreslován v dceřiné komponentě. Na hlavní stránce se zobrazuje seznam přihlašovacích údajů, trezorů, pokud chceme upravit informace o některém

záznamu, kliknutím se nastaví proměnná. Dceřiná komponenta je pak volána a jsou jí předány potřebné parametry. Podle obdržených parametrů jsou pak vykreslována data.

```
[Parameter] public Vault? Vault { get; set; }
```

Obrázek 22: Parametr komponenty

```
<EditVault Vault="SelectedVault" />
```

Obrázek 23: Volání komponenty s parametrem

2. Child to Parent: Tento přístup může být využit v situaci, kdy je akce prováděna v dceřiné komponentě, která ovlivňuje uživatelské rozhraní rodičovské komponenty. K tomu je potřebný EventCallback jako parametr dceřiné komponenty. V aplikaci je tento přístup využíván, když je vybrán nějaký trezor a komponenta EditVault je otevřena. V ní je tlačítko používáno pro přesunutí aktuálního trezoru do koše.

```
[Parameter] public EventCallback Action { get; set; }
```

Obrázek 24: EventCallback parametr

Když se tento event vyvolá, stránka na něj musí reagovat a provést změny v seznamu trezorů a aktualizovat UI.

```
async Task MoveToTrash()
{
    var result = await VaultService.MoveToTrashAsync(SelectedVault!.Id);
    await Action.InvokeAsync(result);
}
```

Obrázek 25: Vyvolání EventCallbacku

```
<VaultDetails Action="() => HandleItemDeleted(_selectedItem!.Id)" />

void HandleItemDeleted(Guid itemId)
{
    _selectedItem = null;
    _vaults.RemoveAll(item => item.Id == itemId);
    StateHasChanged();
}
```

Obrázek 26: Event handler

3. State Container / Service: Tento způsob je používán, když mezi sebou potřebují komunikovat sourozenecké komponenty (dva nebo více komponent na stejné úrovni).

   V komponentě Header jsou volány další tři různé komponenty, které fungují na stejném principu. Jsou otevírány a zavírány po kliknutí na tlačítko. Je považováno za zbytečné vytvářet tři proměnné a tři funkce, které by udržovaly a měnily jejich stav. V takovém případě je vhodné využít centralizovaného správce stavu. Jednotlivé komponenty mohou měnit své stavy aktualizací sdíleného stavu ve správci.

```
public class AddComponentState<T>
{
    public bool IsComponentOpen { get; set; }

    public void SetComponentOpen(bool value)
    {
        IsComponentOpen = value;
        NotifyStateChanged();
    }

    public event Action? OnChange;

    private void NotifyStateChanged() => OnChange?.Invoke();
}
```

Obrázek 27: State Container

   Všechny komponenty, které musí reagovat na změny stavu, pak využívají události `OnChange`. Takový State Container je service, proto je nutné ho zaregistrovat v programové části:

```
builder.Services.AddScoped(typeof(AddComponentState<>));
```

   Ve všech komponentách je pak používána služba jako DI a je ještě přidán řádek `@implements IDisposable`. Tím je umožněno využití metody `Dispose()`.

```
@inject AddComponentState<AddGroup> GroupComponentState
@inject AddComponentState<AddPassword> PasswordComponentState
@inject AddComponentState<AddLogin> LoginComponentState
@implements IDisposable
```

Obrázek 28: DI AddComponentState service

Vhodný způsob komunikace mezi komponentami je klíčově důležitý pro zajištění požadovaného chování uživatelského rozhraní a umožňuje efektivní sdílení dat a koordinaci akcí napříč celou aplikací.

9.3.3 Update UI

V Blazoru je využíván koncept reaktivního programování, kde se daná část UI automaticky aktualizuje pokaždé, když dojde ke změně stavu nebo dat, na kterých je závislá. Pro aktualizaci stavu komponenty a tím i příslušné části UI může být v Blazoru využita metoda StateHasChanged() pro ruční oznámení změny stavu. Tahle metoda je využita při změně stavu komponent využívající AddComponentState service. I to je důvod, proč je nutné zavolat Dispose.

Životní cyklus komponent Blazor poskytuje také řadu životních metod komponent, jako jsou OnInitialized, OnParametersSet a OnAfterRender, které můžete využít pro načítání dat, nastavování počátečního stavu a provádění operací po vykreslení komponenty. Tyto metody umožňují efektivně spravovat aktualizace UI v různých fázích životního cyklu komponenty.
Pro aktualizaci stavu komponenty je nutné v těle metod OnInitialized a Dispose z IDosposable zavolat událost z třídy AddComponentState. Kdykoli se změní stav, událost StateHasChanged je vyvolána a komponenta je buď vykreslena nebo zmizí z obrazovky. Je důležité nezapomenout odhlásit metodu StateHasChanged komponenty z události OnChange, mohlo by to způsobit únik paměti. [43]

9.3.4 Formuláře a data binding

EditForm je zabudovanou součástí v Blazoru. Není to pouze samotná formulářová komponenta, ale komplexní řešení pro propojení dat, ověřování vstupů a odesílání formuláře. V případě použití Blazoru v režimu SSR není dosaženo úplné interaktivity pouhým přidáním submit tlačítka do komponenty a formulář se standardně neodesílá. Nicméně plná funkcio-

nalita EditForm komponenty v SSR režimu může být dosažena nastavením dalších atributů.

Jedním z atributů je FormName, který slouží k jednoduchému pojmenování formuláře. Dalším důležitým atributem je Model. Aby EditForm věděl, na jaká data se má vázat, je třeba vytvořit třídu ViewModel pro konkrétní formulář. ViewModel obsahuje pouze ty atributy, které se očekávají ve formuláři. Tím se liší od klasických modelových tříd entit, kde může být atributů více, protože mohou být například vyplňovány automaticky.

V code bloku Blazor komponenty je poté vytvořena a inicializována nová instance ViewModelu s atributem `[SupplyParameterFromForm(Name)]`. Parametr Name v tomto atributu je nutný pouze v případě, že v dané komponentě existuje více formulářů. V registračním formuláři v aplikaci je od uživatele vyžadováno zadání emailové adresy, hesla a jeho potvrzení. Tyto informace jsou nutné nadefinovat ve ViewModel třídě, která by mohla vypadat nějak takto:

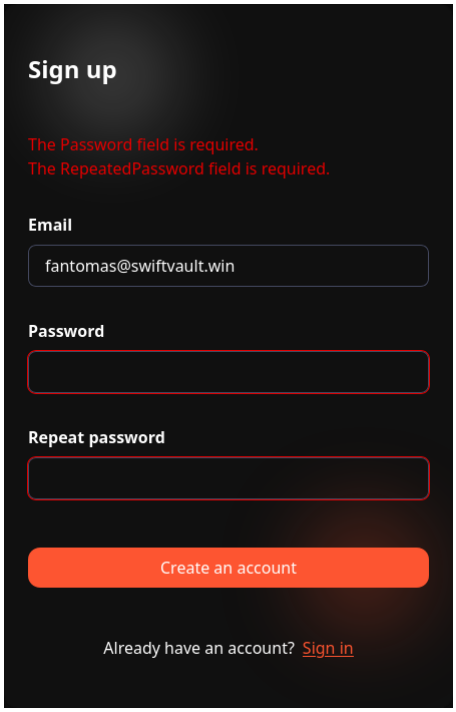
```
public class RegistrationViewModel
{
    [Required] [EmailAddress] public string Email { get; set; } = null!;

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.",
        MinimumLength = 6)]
    [DataType(DataType.Password)]
    public string Password { get; set; } = null!;

    [Required]
    [Compare(nameof(Password), ErrorMessage = "Passwords don't match!")]
    public string RepeatedPassword { get; set; } = null!;
}
```

Obrázek 29: Registrační View Model

Podle toho pak bude vypadat i registrační formulář. Pro validaci dat a informování uživatele o špatně vyplněném formuláři se do EditFormu volají komponenty `<DataAnnotationsValidator />` a `<ValidationSummary/>`.

```
<EditForm Model="ViewModel" method="post" OnValidSubmit="RegisterUser" FormName="register">
    <DataAnnotationsValidator/>
    <ValidationSummary/>
    ...
</EditForm>

@code {
    [SupplyParameterFromForm] private RegistrationViewModel ViewModel { get; set; } = new();
}
```

Obrázek 30: EditForm s atributy a inicializace ViewModelu



Obrázek 31: Registrační formulář

9.4 Vývoj backendu v ASP.NET Core 8

Backend je postaven na repositories a services, které jsou zaregistrovány jako metody, připravené k použití jako DI. Každá implementace bude mít svůj interface, který bude injectován v Blazor komponentech pomocí direktivy `@inject`. V kódovém bloku budou pak zpřístupněny všechny veřejné metody v interface.

Dalším důležitým bodem je, že repositories budou využívat DbContextFactory. To je kvůli životnímu cyklu DbContextu, který by ovlivňoval komponent v režimu interaktivního renderování. DbContext by stále existoval, tudíž by nenačítal nová data, kdybychom se třeba přepli na novou stránku. Ale právě tento problém řeší DbContextFactory. Začne nebo ukončí životní cyklus pokaždé, když se uživatel přepne na jinou stránku.

9.4.1 Repositories

Repositories jsou v podstatě služby, které se využívají pro manipulaci dat a komunikaci s databází. Mohou sem být zařazeny základní funkce modelu CRUD.

Veškeré metody budou implementovány v projektu Infrastructure, kde se nachází i DbContext. Kromě metod pro manipulaci dat bude vytvořena třída, která nám vrátí aktuálně přihlášeného uživatele. Tato třída se bude hodit v metodách pro komunikaci s databází.

```
public sealed class UserAccessor(
    IHttpContextAccessor httpContextAccessor,
    UserManager<ApplicationUser> userManager)
{
    public async Task<ApplicationUser> GetRequiredUserAsync()
    {
        var principal = httpContextAccessor.HttpContext?.User ?? throw new InvalidOperationException(
            $"{nameof(GetRequiredUserAsync)} requires access to an {nameof(HttpContext)}");

        var user = await userManager.GetUserAsync(principal);

        return user ?? null!;
    }
}
```

Obrázek 32: Třída UserAccessor s metodou GetRequiredUserAsync

Se správným Id uživatele je možné spravovat jeho data. Veškerá repositories budou sdružena do složky a pojmenována podle části aplikace, kterou ovládají. Například metody ovládající trezory budou umístěny ve třídě VaultService. Jako DI budou k dispozici naše třída UserAccessor a IDbContextFactory.

```
public class VaultService(
    UserAccessor userAccessor,
    IDbContextFactory<MyDbContext> dbContextFactory) : IVaultService
```

Obrázek 33: Hlavička třídy s DI jako primary constructor

Metody využijí userAccessor k získání uživatele a tím pádem jeho Id, které se v databázi používá jako primární a cizí klíč. Namísto ručních SQL dotazů budou použity metody EF Core, které pro práci s daty využívají LINQ, technologii založenou na dotazování. Ruční psaní rozhraní není nutné. V Rideru je možné rozhraní vygenerovat kliknutím pravým tlačítkem na třídu a v sekci Refactor vybrat možnost Extract Interface. Rozhraní lze vygenerovat do souboru s implementací, avšak lepším řešením je uložit jej do dalšího souboru v jiném adresáři, například IRepositories.

```
public async Task<List<Vault>> GetVaultsAsync()
{
    var user = await userAccessor.GetRequiredUserAsync();
    await using var context = await dbContextFactory.CreateDbContextAsync();
    return await context.Vaults.Where(c => c.ApplicationUserId == user.Id && c.IsDeleted == false).ToListAsync();
}
```

Obrázek 34: Metoda pro získání všech uživatelských trezorů

Formuláře sloužící pro vytvoření trezoru nebo skupiny vyžadují ViewModely.

```
public class AddLoginViewModel
{
    [Required] public string Title { get; set; } = null!;

    [Required] public string UserName { get; set; } = null!;

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; } = null!;

    public string Websites { get; set; } = null!;
}
```

Obrázek 35: ViewModel pro přidání trezoru

Pro vytvoření trezoru a uložení do databáze se využívá model Vault, do kterého jsou přiřazeny hodnoty získané z formuláře. Nejdříve je však nutné heslo zašifrovat. K tomu účelu byl použit algoritmus AES. Poté se vytvoří kontext a záznam se zapisuje do databáze.

```
public async Task<bool> CreateLoginAsync(AddLoginViewModel model)
{
    var user = await userAccessor.GetRequiredUserAsync();

    var key = keyManager.GenerateMasterKey();
    var enryptedKey = await keyManager.EncryptKey(key);
    var encryptedPassword = await aesEncryption.EncryptAsync(model.Password, key);

    var vault = new Vault
    {
        Title = model.Title,
        Username = model.UserName,
        Password = encryptedPassword,
        EncryptedKey = enryptedKey,
        Websites = model.Websites,
        ApplicationUserId = user.Id,
        CreatedAt = DateTime.Now
    };

    await using var context = await dbContextFactory.CreateDbContextAsync();
    context.Vaults.Add(vault);
    return await context.SaveChangesAsync() > 0;
}
```

Obrázek 36: Přidání nového trezoru do databáze

Heslo pak v databázi bude nečitelné.

Username	Password
april@april.me	MdZ0LORagxeFM0xrpD7rqlEcCQhY/IihN7nN9pnGWD8=
funguje	N+2i0IsfEShBfaT88Pkrvi8GYHASFiq08WLyT0gHcS0=

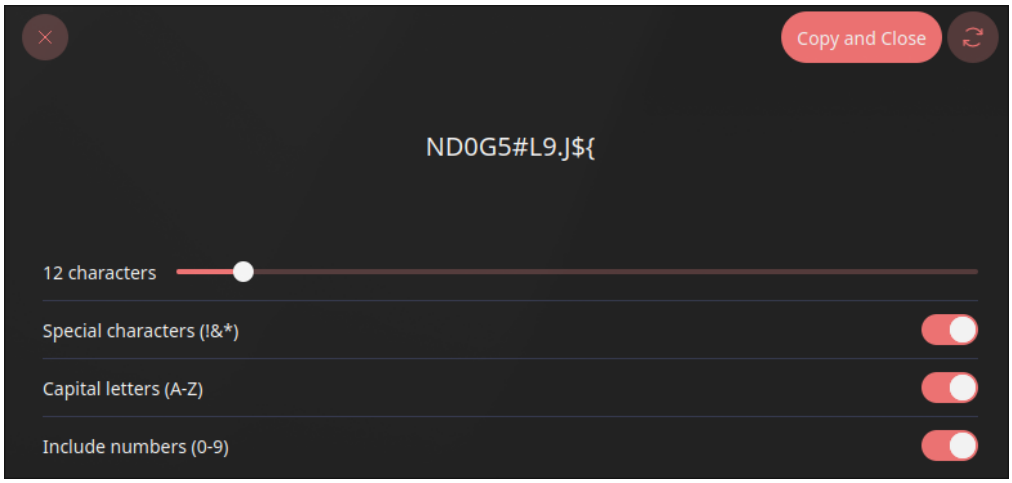
Obrázek 37: Hesla v databázi

9.4.2 Services

Metody Services jsou ty, které nepracují s daty, ale spíše pouze s logikou aplikace. Všechny takové metody budou napsány v projektu Application.

Jednou z logických metod může být generování hesla. Uživatel má možnost vygenerovat náhodné heslo podle počtu znaků a rozhodnout, zda chce zahrnout speciální znaky, velká

písmena a číslice. Po každé změně některé z těchto možností se heslo automaticky přegeneruje.



Obrázek 38: Generátor hesla

```
public string GeneratePassword(int passwordLength, bool includeSpecialCharacters, bool includeCapitalLetters,
    bool includeNumbers)
{
    var charSet = "abcdefghijklmnopqrstuvwxyz";
    var random = new Random();
    var passwordChars = new List<char>();

    if (includeSpecialCharacters)
    {
        charSet += "!@#$%^&*()_+-=[]{}|;:,.<>?";
        passwordChars.Add("!@#$%^&*()_+-=[]{}|;:,.<>? "[random.Next("!@#$%^&*()_+-=[]{}|;:,.<>? ".Length)]);
    }

    if (includeCapitalLetters)
    {
        charSet += "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        passwordChars.Add("ABCDEFGHIJKLMNOPQRSTUVWXYZ"[random.Next("ABCDEFGHIJKLMNOPQRSTUVWXYZ".Length)]);
    }

    if (includeNumbers)
    {
        charSet += "0123456789";
        passwordChars.Add("0123456789"[random.Next("0123456789".Length)]);
    }

    for (int i = passwordChars.Count; i < passwordLength; i++)
    {
        passwordChars.Add(charSet[random.Next(charSet.Length)]);
    }

    passwordChars = passwordChars.OrderBy(_ => random.Next()).ToList();
    return new string(passwordChars.ToArray());
}
```

Obrázek 39: Metoda pro vygenerování hesla

9.4.3 Autentizace a autorizace

Zjištěním, že implementace JWT je složitější než původně zdánlivě, a že se tokeny spíše využívají pro zabezpečení vzdálených služeb, jako je například Web API, které Blazor Server nepoužívá, došlo k použití cookies pro autentizaci a autorizaci.

Pro registraci a přihlášení uživatele byl využit Identity systém v ASP.NET Core, který výrazně usnadňuje celý proces pomocí tříd UserManager a SignInManager. Třída UserManager umožňuje spravovat uživatele, zatímco SignInManager ověřuje uživatele. Stránky, ke kterým by měl uživatel přístup, musí obsahovat direktivu `@attribute [Authorize]`. Pro zjednodušení celého procesu je tato direktiva napsána do souboru _Imports.razor. Tím pádem budou všechny stránky pod pravidlem Authorize, včetně stránek pro registraci a přihlášení. Aby byl umožněn přístup k těmto stránkám, je nutné přímo do nich přidat direktivu `@attribute [AllowAnonymous]`.

10 NASAZENÍ A ÚDRŽBA

Nasazení bude provedeno zapomocí Dockeru a Docker Compose.

10.1 Příprava pro nasazení

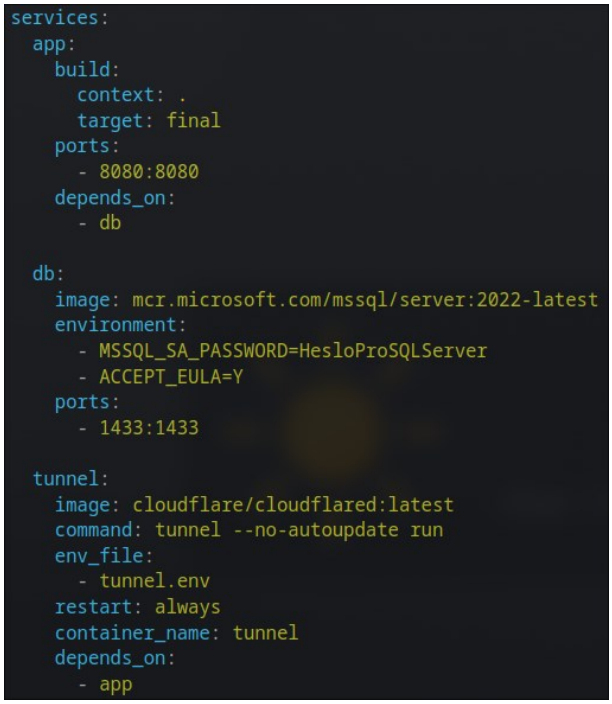
Příkazem `Docker init` v terminálu z adresáře Solution je možné vygenerovat Dockerfile tak, aby byl vidět všem projektům. Tento příkaz vygeneruje následující:

```
FROM --platform=$BUILDPLATFORM mcr.microsoft.com/dotnet/sdk:8.0-alpine AS build

COPY . /source

WORKDIR /source/PM.Web

ARG TARGETARCH

RUN --mount=type=cache,id=nuget,target=/root/.nuget/packages \
    dotnet publish -a ${TARGETARCH/amd64/x64} --use-current-runtime --self-contained false -o /app

FROM mcr.microsoft.com/dotnet/aspnet:8.0-alpine AS final
WORKDIR /app

COPY --from=build /app .

USER $APP_UID

ENTRYPOINT ["dotnet", "PM.Web.dll"]
```

Obrázek 40: Dockerfile

Komunikace s databází je nezbytná pro běh aplikace. V souboru compose.yaml jsou definovány služby a jejich vzájemné závislosti.

Pro možnost hostování na WAN je nezbytné vlastnit doménu. Pro tento projekt byla doména zaregistrována u poskytovatele Cloudflare. Dále byl vytvořen síťový tunel, který zajišťuje bezpečné spojení mezi Cloudflare a aplikací. Vygenerovaný token tunelu je třeba uložit do souboru s příponou .env.

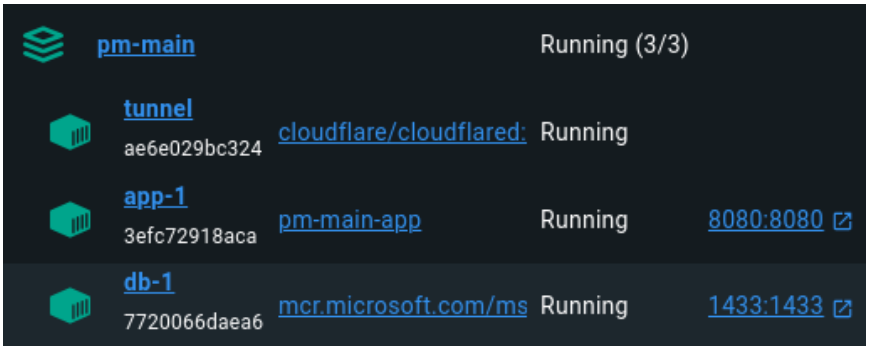


Obrázek 41: Compose.yaml

10.2 Konfigurace

Reverzní proxy jako Nginx nebo Apache jsou často používány pro většinu technologií. Nicméně, v případě, že technologie disponuje vlastním webovým serverem, to není nutné. Například v případě ASP.NET Core je používán Kestrel, který má vysoký výkon, podporuje HTTPS a je schopen fungovat na jakékoli platformě. Proto není nutné nastavovat reverzní proxy v aplikaci.

S použitím připraveného konfiguračního souboru compose.yaml je možné spustit všechny kontejnery jedním příkazem `docker compose up -d`. Veškeré potřebné images budou stáhnuty a kontejnery budou aktivovány. Z Docker compose pak vznikne Docker network.



Obrázek 42: Běžící Docker network

10.3 Aktualizace

K aktualizaci ať už některého z image nebo nové verze aplikace musí být network zastavena. Pro aktualizaci image je stačí přepsat v compose.yaml, pro nasazení nové aplikace je potřeba nahradit stávající nejnovější verzí a znovu spustit `docker compose up -d`. Pro úplné zastavení compose, něco jako reset, může být použit `docker compose down`.

ZÁVĚR

Do téhle práce jsem šel s tím, že jsem neměl tušení, jak pro mě budou některé části opravdu složité, a i dokonce neproveditelné. Výsledkem této bakalářské práce je velmi základní a omezená webová aplikace. Poskytuje jistou míru zabezpečení a základní akce. JSON Web Tokeny nebyli využity, jelikož pro Blazor Server aplikaci se velice obtížně implementují, tudíž byla použita náhrada v cookies. Uživatel se může registrovat i přihlásit a myslím si, že vzhledově stránky nemohl dopadnout lépe. Kombinace oranžové s odstíny tmavě šedé až černé barvy skvěle ladí.

Při vývoji nastaly mnohé problémy, jejichž vyřešení kolikrát vedlo ke způsobení dalších. Chtělo by si nad tím sednout a podívat se důkladně, co konkrétně způsobuje výjimky a zřejmě i předělat spoustu věcí. Jelikož i tohle mě velice brzdilo v dokončení aplikace. A taky samozřejmě čas, kterého bylo stráveno až přespříliš, nad některými implementacemi.

Nemyslím si, že je to k zahození, po refactoringu a zvážení jiného přístupu k některým akcím, by byla aplikace funkční a v budoucnu by nemusela sloužit jen jako správce hesel. Konzultace s někým, kdo se Blazorem zabývá, by byla jedině výhodou, ať už názorem, nebo radou.

SEZNAM POUŽITÉ LITERATURY

[1] OKTA. What is Authentication? Online. Auth0: Secure access for everyone. But not just anyone. C2024. Dostupné z: https://auth0.com/intro-to-iam/what-is-authentication. [cit. 2024-02-28].

[2] BONE, Harry. What is two-factor authentication (2FA)? Online. The Proton Blog. 2023, roč. 2023, č. 2, s. 1. Dostupné z: https://proton.me/blog/what-is-two-factor-authentication-2fa. [cit. 2024-05-12].

[3] What is two-factor authentication? Online. In: Microsoft Security. C2024. Dostupné z: https://www.microsoft.com/en-us/security/business/security-101/what-is-two-factor-authentication-2fa. [cit. 2024-05-12].

[4] The Importance of Computer Software Updates & Security Patches. Online. Microsoft 365 Life Hacks. 2022, roč. 2022, č. 1, s. 1. Dostupné z: https://www.microsoft.com/en-us/microsoft-365-life-hacks/privacy-and-safety/computer-software-update. [cit. 2024-05-12].

[5] CRAWFORD, Douglas. *What is a password manager and why do I need one?* Online. The Proton Blog. 2023, 18.8.2023. Dostupné z: https://proton.me/blog/what-is-a-password-manager. [cit. 2024-02-27].

[6] CRAWFORD, Douglas. How to create strong passwords you'll actually remember. Online. *The Proton Blog*. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://proton.me/blog/create-remember-strong-passwords. [cit. 2024-05-12].

[7] BURNETT, Mark. *Perfect Password: Selection, Protection, Authentication*. Syngress, 2005. ISBN 1597490415.

[8] GRIGUTYTĖ, Monika. What is password cracking and what techniques do hackers use. Online. *NordVPN*. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://nordvpn.com/blog/password-cracking/. [cit. 2024-05-12].

[9] BHATT, Paritosh. The Art and Science Behind Password Managers. Online. Medium. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://medium.com/@paritoshblogs/the-art-and-science-behind-password-managers-fbf5fb9c7f99. [cit. 2024-05-12].

[10] O'SULLIVAN, Fergus. Are password managers safe? Online. *The Proton Blog*. 2024, roč. 2024, č. 2, s. 1. Dostupné z: https://proton.me/blog/are-password-managers-safe. [cit. 2024-05-12].

[11] MARTINOLI, Marco. What is end-to-end encryption and how does it work? Online. *The Proton Blog*. 2023, roč. 2023, č. 2, s. 1. Dostupné z: https://proton.me/blog/what-is-end-to-end-encryption. [cit. 2024-05-12].

[12] *Introduction to different types of password managers*. Online. In: Zoho Vault. C2024. Dostupné z: https://www.zoho.com/vault/educational-content/different-types-of-password-managers.html. [cit. 2024-05-12].

[13] BHATT, Paritosh. The Art and Science Behind Password Managers. Online. Medium. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://medium.com/@paritoshblogs/the-art-and-science-behind-password-managers-fbf5fb9c7f99. [cit. 2024-05-12].

[14] *The Future of Password Management: Trends to Watch*. Online. In: Privacy Hawk. C2024. Dostupné z: https://privacyhawk.com/resources/the-future-of-password-management-trends-to-watch/. [cit. 2024-05-12].

[15] BISHOP, Edward. The Future of Password Management: Is it time to say goodbye to Password Managers? Online. *Market Business News*. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://marketbusinessnews.com/the-future-of-password-management-is-it-time-to-say-goodbye-to-password-managers/327495/. [cit. 2024-05-12].

[16] O'SULLIVAN, Fergus. What is a passkey? Online. *The Proton Blog*. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://proton.me/blog/what-is-a-passkey. [cit. 2024-05-12].

[17] WASIKE, Bravin. 30 Best Web Development Frameworks for 2023: A Comprehensive Guide. Online. *Dev Community*. 2023, roč. 2023, č. 2, s. 1. Dostupné z: https://dev.to/bravinsimiyu/30-best-web-development-frameworks-for-2023-a-comprehensive-guide-512i. [cit. 2024-05-12].

[18] Přehled ASP.NET Core. Online. *Microsoft Learn*. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0. [cit. 2024-05-12].

[19] ASP.NET Core Blazor. Online. Microsoft Learn. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/?view=aspnetcore-8.0. [cit. 2024-05-12].

[20] ASP.NET Core Blazor s Entity Framework Core (EF Core). Online. Microsoft Learn. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/blazor-ef-core?view=aspnetcore-8.0. [cit. 2024-05-12].

[21] Entity Framework Core. Online. *Microsoft Learn*. 2021, roč. 2021, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/en-us/ef/core/. [cit. 2024-05-12].

[22] M., Sopha. What Is a Development Environment? How Does It Differ From an Integrated Development Environment (IDE)? Online. *Hostinger Tutorials*. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://www.hostinger.com/tutorials/development-environment. [cit. 2024-05-12].

[23] OBREGON, Alexander. Embrace the World of Efficient Coding with JetBrains Rider. Online. *Medium*. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://medium.com/@AlexanderObregon/embrace-the-world-of-efficient-coding-with-jetbrains-rider-f67524000b5b. [cit. 2024-05-12].

[24] *Docker overview*. Online. Docker Docs. C2013-2024. Dostupné z: https://docs.docker.com/get-started/overview/. [cit. 2024-05-12].

[25] *Docker Compose overview*. Online. Docker Docs. C2013-2024. Dostupné z: https://docs.docker.com/compose/. [cit. 2024-05-12].

[26] What is SQL Server? Online. *Microsoft Learn*. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16. [cit. 2024-05-12].

[27] *What is SQL Server*. Online. In: SQL Server Tutorial. C2023. Dostupné z: https://www.sqlservertutorial.net/getting-started/what-is-sql-server/. [cit. 2024-05-12].

[28] grapes. What are the advantages of running SQL Server in a Docker container? Online. In: Stack Overflow. 2019. Dostupné z: https://stackoverflow.com/questions/54841211/what-are-the-advantages-of-running-sql-server-in-a-docker-container. [cit. 2024-05-12].

[29] Entity Framework Core. Online. *Microsoft Learn*. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/cs-cz/ef/core/. [cit. 2024-05-12].

[30] ASP.NET Core Blazor s Entity Framework Core (EF Core). Online. Microsoft Learn. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/blazor-ef-core?view=aspnetcore-8.0. [cit. 2024-05-12].

[31] G., Domantas. What Is Web Hosting – Web Hosting Explained for Beginners. Online. *Hostinger Tutorials*. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://www.hostinger.com/tutorials/what-is-web-hosting/. [cit. 2024-05-12].

[32] ROLAND, Chris Ebube. Cloud Hosting vs. Self-Hosting. Online. *Open Replay*. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://blog.openreplay.com/cloud-hosting-vs-self-hosting/. [cit. 2024-05-12].

[33] ASP.NET Core Blazor. Online. Microsoft Learn. 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/?view=aspnetcore-8.0. [cit. 2024-05-12].

[34] Identity model customization in ASP.NET Core. Online. Microsoft Learn. 2022, roč. 2022, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-8.0. [cit. 2024-05-12].

[35] PINE, David. Learning Blazor. O'Reilly Media, 2022. ISBN 978-1098113247.

[36] What is an application architecture? Online. *Red Hat*. 2020, roč. 2020, č. 1, s. 1. Dostupné z: https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture. [cit. 2024-05-12].

[37] GOD, Patrick. Clean Architecture w/ Blazor 🔥, Blazor in the Industry 🚀, .NET Web Academy Enrollment Closing Soon 🎓. Online. Patrick God Posts. March 2024, roč. 2024, č. 1, s. 1. Dostupné z: https://dotnet8.patrickgod.com/posts/clean-architecture-w-blazor-blazor-in-the-industry-net-web-academy-enrollment-closing-soon. [cit. 2024-05-12].

[38] BERGMAN, Per-Erik. Understanding Clean Architecture. Online. Medium. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://medium.com/@pererikbergman/understanding-clean-architecture-5f7a39a965fb. [cit. 2024-05-12].

[38] VASCONCELOS, Rubem. Clean Architecture: The concept behind the code. Online. Dev Community. 2022, roč. 2022, č. 1, s. 1. Dostupné z: https://dev.to/rubemfsv/clean-architecture-the-concept-behind-the-code-52do. [cit. 2024-05-12].

[39] Dependency injection in ASP.NET Core. Online. Microsoft Learn. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0. [cit. 2024-05-12].

[42] SUBRAMANIAN, Sundaram. What Is Blazor And How It Works. Online. *C# Corner*. 2023, roč. 2023, č. 1, s. 1. Dostupné z: https://www.c-sharpcorner.com/article/what-is-blazor-and-how-does-it-works/. [cit. 2024-05-12].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

2FA	Dvoufaktorová autentizace
MFA	Multifaktorová autentizace
PIN	Personal Identification Number
SMS	Short Message Service
AES	Advanced Encryption Standard
E2EEE	End-to-End Encryption
XSRF/CSRF	Cross-Site Request Forgery
RPC (gRPC)	Remote Procedure Call
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
CSR	Client-Side Rendering
PWA	Progressive Web Application
DI	Dependency Injection
MSSQL	Microsoft SQL Server
SQL	Structured Query Language
ORM	Object-Relational Mapping
EF	Entity Framework
IDE	Integrated Development Environment
CI/CD	Continuous Integration/Continuous Deployment
REST	REpresentational State Transfer
API	Application Programming Interface
UNIX	Uniplexed Information and Computing Service
CLI	Command Line Interface
RDBMS	Relational Database Management System
SQLOS	SQL Server Operating System
SAN	Storage Area Network
EULA	End User License Agreement

env	Environment
VPS	Virtual Private Server
GCP	Google Cloud Platform
AKS	Azure Kubernetes Service
AWS	Amazon Web Services
LINQ	Language Integrated Query
npm	Node Package Manager
SPA	Single Page Application
SSR	Server-Side Rendering
IoC	Inversion of Control
SDK	Software Development Kit
GUID	Globally Unique Identifier
UI	User Interface
UX	User Experience
CRUD	Create, Read, Update, Delete
HTTPS	HyperText Transfer Protocol Secure
JWT	JSON Web Token

SEZNAM OBRÁZKŮ