Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Vývoj webové aplikace pro decentralizované sdílení souborů

Bakalářská práce

Autor: Petr Schmidt
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Milan Košťák

Hradec Králové Srpen 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 15.8.2022 Petr Schmidt

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Milanu Košťákovi za metodické vedení práce, rady a cenné připomínky.

Anotace

Bakalářská práce se zabývá vývojem webové aplikace pro decentralizované sdílení souborů na internetu.

V teoretické části je popsána historie internetu, jeho struktura a dále problematika centralizace. Následuje představení metod umožňujících decentralizaci a distribuci obsahu v internetu, zejména pak technologie InterPlanetary File System (IPFS) a její široké možnosti uplatnění. V souvislosti s technologií IPFS je následně představena kryptoměna Filecoin figurující jako komplementární technologie pro fungování decentralizovaných a distribuovaných datových úložišť. Teoretická část dále pokračuje sekcí věnující se fungování webu a webových aplikací, souvisejícím skriptovacím a značkovacím jazykům, a knihovně React.

Praktická část se věnuje návrhu webové aplikace pro sdílení souborů uplatňující metody decentralizace vysvětlené v teoretické části. Následuje samotná implementace webové aplikace, popis jejího uživatelského prostředí a interních procesů. Výsledná aplikace ukazuje, že implementace popsaných principů umožňuje do jisté úrovně decentralizovat proces sdílení souborů přes internet, zejména jejich uložení a stahování. Mimo to výsledná aplikace nabízí volitelné funkcionality v podobě zabezpečení soukromých souborů, či optimalizace uložení souborů v IPFS pro jejich jednodušší stažení.

Annotation

Title: Building a Web-based Application for Decentralized File-sharing

The bachelor thesis presents the development of a web application for decentralized file sharing on the Internet.

The theoretical part describes the history of the Internet, its structure and centralization issues. This is followed by an introduction of methods enabling decentralization and distribution of content on the Internet, particularly the InterPlanetary File System (IPFS) technology and its wide range of applications. In the context of the IPFS technology, the Filecoin cryptocurrency is then introduced as a complementary technology for the functioning of decentralized and distributed data storages. Finally, the theoretical part concludes with a section on the functioning of the web and web applications, related scripting and markup languages, and the React library.

The practical part is devoted to the proposition of a web application for file sharing applying the decentralization methods explained in the theoretical part. This is followed by the actual implementation of the web application and a description of its user interface and internal processes. The resulting application shows that the implementation of the described principles allows to decentralize the process of file sharing over the Internet to a certain level, especially file storage and file downloading. Besides, the resulting application offers optional functionalities in the form of security of private files or optimization of file storage in IPFS for easier downloading.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Současný stav fungování sdílení obsahu na internetu je náchylný na řadu rizik – bezpečnost, cenzuru, integritu a tzv. body selhání („*points of failure*“). Důvodem je centralizace postavená na architektuře klient-server. Drtivá většina internetových služeb, které denně používáme, je centralizovaná.

V poslední době se však s rozmachem kryptoměn, a obecně Web 3.0, začalo mluvit o tzv. *decentralizaci*. Účelem decentralizace je ve své podstatě eliminace architektury klient-server nahrazením architekturou klient-klient (*peer-to-peer*). Postupné snahy open-source komunit o implementaci takových decentralizovaných sítí umožňují vznik velice robustních a odolných celosvětových systémů, jako například *InterPlanetary File System (IPFS)* – decentralizovaný internetový souborový systém.

Teoretická část práce se zprvu zaměřuje na popis architektury internetu, včetně jeho historie a dále problému centralizace služeb na něm fungujících. V návaznosti je následně představen princip decentralizace, definice peer-to-peer sítě, technologie IPFS a kryptoměna Filecoin. Zbývající kapitoly teoretické části práce se věnují problematice vývoje moderních webových stránek či aplikací, a knihovně *React*.

Na základě těchto poznatků byla v praktické části práce navržena a implementována webová React aplikace využívající technologii IPFS pro demonstraci možnosti sdílet soubory na internetu decentralizovaně.

2 Internet, centralizace a cenzura

Internet je téměř nedílnou součástí života mnoha lidí na planetě. Podle *Internet World Stats* je odhadováno, že k 30. červnu 2022 používá internet více než 66 % světové populace, s celkovým nárustem o 1355 % mezi roky 2000 až 2022. Využití internetu se neustále rozšiřuje do nejrůznějších odvětví. Věci, které dříve existovaly pouze fyzicky – „off-line“, jsou nyní digitalizované a dostupné přes internet. (Miniwatts Marketing Group 2022)

Kapitola stručně popisuje historii vzniku internetu a jeho struktury. Dále je rozebrán aktuální centralizovaný stav služeb na něm fungujících, přičemž je poukázáno na s tím spojené nevýhody a značná rizika.

2.1 Definice internetu

Slovo *internet* je ve své podstatě zkratkou složenou z anglických slov „*interconnected*“ (propojený) a „*network*“ (síť). Jak již název může napovídat, internet je velké množství navzájem propojených sítí a zařízení. Zařízeními jsou v tomto kontextu myšlena jakákoliv zařízení schopná v takové síti přijímat a odesílat data na základě stanovených standardů, například mobilní telefon, notebook, televize, stolní počítač nebo aktivní síťové prvky (router, směrovač apod.). (Machala 2007)

2.2 Historie internetu

Technologie internetu vychází z projektu *ARPANET*, vyvinutém americkou výzkumnou institucí *Advanced Research Projects Agency* (*ARPA*). ARPANET byla počítačová síť, spuštěná v roce 1969, která umožňovala komunikaci mezi počítači na velkou vzdálenost. (Machala 2007)

Hlavním důvodem vzniku takové technologie bylo v první řadě zajištění komunikace mezi základnami americké armády v případě, že by došlo k atomovému útoku, který v době studené války hrozil. Jednou z předních vlastností této sítě byla decentralizace – pokud by došlo k výpadku některého z uzlů v této síti, komunikace by byla zajištěna zbývajícími uzly. (Machala 2007)

Mimo armádní účely byl ARPANET využíván i na univerzitách, kde umožňoval například vzdálený přístup k tehdejším superpočítačům za účelem provádění složitých výpočtů. (Machala 2007)

Počínaje rokem 1972 se ARPANET zpřístupnil i běžným lidem a firmám, čímž se začal počet uzlů v síti exponenciálně zvyšovat. To mělo za následek masivní rozšíření této sítě, a to i za hranice USA. Postupně vznikl internet tak, jak jej známe dnes. (Machala 2007)

2.3 Struktura internetu a architektura klient-server

Jak již bylo zmíněno, internet definuje velké množství navzájem propojených sítí a zařízení. Základní struktura internetu je tak do jisté úrovně decentralizovaná. (Nast 2014)

I přes tuto strukturu však většina služeb na internetu funguje centralizovaně, na základě tzv. *klient-server* architektury. Centrálním bodem v této architektuře je právě server, ke kterému se následně připojuje množství klientů. Server je v této architektuře klíčovým uzlem, a může například umožňovat komunikaci mezi klienty. V případě, že ale dojde k výpadku tohoto uzlu, ztratí klienti možnost mezi sebou komunikovat – tento stav se nazývá „centrální bod selhání“.

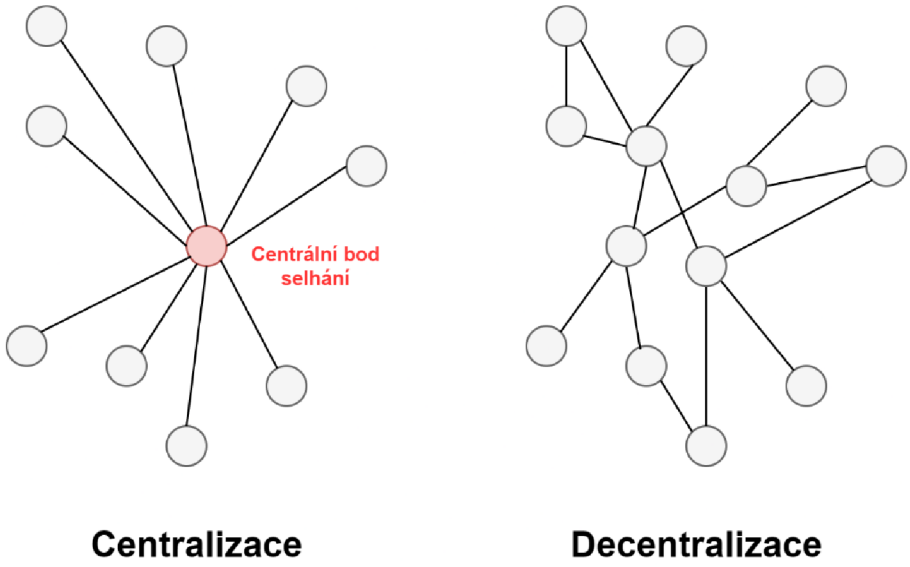
2.4 Cenzura a zabezpečení dat

Centrální bod selhání není jediná nevýhoda centralizované architektury klient-server. Pokud bychom se bavili například o službě, která umožňuje sdílení obsahu (textové příspěvky, obrázky, videa a jiné soubory), jsou data v plném vlastnictví této služby, uložená centralizovaně na serveru, a tak velice jednoduše modifikovatelná či kompletně cenzurovatelná. (Zarrin et al. 2021)

Uložení na centrálním serveru dále přináší i riziko kompletní ztráty uložených dat, pokud by například došlo k poškození serveru a data nebyla zálohována. Dále v případě soukromých dat pak téměř nikdy nemáme jistotu, zda jsou data skutečně chráněna proti neoprávněnému přístupu, tedy šifrována.

3 Decentralizace, IPFS a Filecoin

Decentralizace internetu ve všech jeho vrstvách je možnost, jak jej chránit před monopolizací, a zajistit jeho uživatelům vzájemnou rovnost a stejnou úroveň autority. Mimo to decentralizace umožňuje předcházet slabinám v podobě centrálních bodů selhání, které jsou vlastností centralizovaných architektur. (Zarrin et al. 2021; Raman et al. 2019)



Obrázek 1 Porovnání topologie centralizované a decentralizované architektury (práce autora)

3.1 Decentralizace prostřednictvím peer-to-peer sítě

Peer-to-peer (P2P) je typ sítě, kde mezi sebou komunikují přímo jednotliví klienti – tzv. *klient-klient* model. Jedná se o opak modelu klient-server, kdy je středem komunikace vždy nějaký server či servery. P2P sítě jsou decentralizované a odolné proti selhání, protože nejsou vázány na žádné centrální servery, nemají tedy centrální bod selhání. V P2P síti zároveň neexistuje žádná centrální autorita, a žádný klient (uzel) nemůže ovládat celou síť. (Wang et al. 2018) V současnosti mají P2P sítě široké využití, například:

- *BitTorrent* – nástroj pro masivní distribuci digitálního obsahu.

- Kryptoměny – digitální měny využívající asymetrické šifrování pro vytváření řetězů, tzv. *blockchainů*, digitálních podpisů jednotlivých „finančních“ transakcí.
- Decentralizovaná a distribuovaná datová úložiště, například IPFS a kryptoměnová síť *Filecoin*.
- Poskytování výpočetního výkonu pro těžbu kryptoměn (ověřování transakcí) nebo například pro dobrovolné výzkumné projekty (*BOINC*).

3.2 Technologie IPFS

InterPlanetary File System (*IPFS*) je distribuovaný P2P souborový systém (někdy též uváděný jako protokol), jehož cílem je propojení všech výpočetních zařízení neboli *uzlů*, v rámci stejného systému souborů. IPFS poskytuje obsahem adresovatelné blokové úložiště („*content-addressed block storage*“) s vysokou propustností. Z definice P2P sítě dále vyplývá, že IPFS nemá centrální bod selhání, a navíc jednotlivé uzly si nemusí navzájem důvěřovat, viz kapitola *Identity* (3.2.1). Jelikož jsou soubory distribuovaně uloženy v různém počtu uzlů a v různých částech světa, snižuje se riziko, že by soubory zmizely, pokud by nějaký uzel vypadl. (Benet 2019; Wolf 2022; Wang et al. 2018)

Cílem IPFS je primárně vytvořit globální decentralizovanou a distribuovanou infrastrukturu pro veřejné soubory, jako například webové stránky. *„Protokol má v open source světě dlouhodobou ambici nahradit HTTP protokol. Adresování ale u obou protokolů funguje přesně opačně. Zatímco HTTP vychází z toho, na jakém serveru se obsah nalézá, pro IPFS je důležitější, co je samotným obsahem.“* (Wolf 2022)

3.2.1 Identity

Jednotlivé uzly jsou identifikovány pomocí *NodeId*, což je kryptografický hash veřejného klíče daného uzlu. Každý uzel má svůj veřejný a privátní klíč. Při prvním spojení si uzly mezi sebou navzájem vymění veřejné klíče a zkontrolují, zda hash veřejného klíče protějšího uzlu je totožný s *NodeId* tohoto uzlu. Pokud jsou hash a *NodeId* odlišné, spojení mezi těmito uzly je ukončeno. (Benet 2019)

3.2.2 Síť

V IPFS běžně každý uzel komunikuje se stovkami jiných uzlů napříč celou sítí, respektive internetem. Pro komunikaci může být použit jakýkoliv transportní protokol, a IPFS může poskytovat lepší spolehlivost v sítích, které jsou nespolehlivé, například z důvodu vysoké latence či přetížení. (Benet 2019)

3.2.3 Soubory a content addressing

Soubor je při nahrání do IPFS rozdělen do menších kusů, tzv. *bloků*, přičemž tyto bloky jsou následně distribuovány do různého počtu uzlů. Společně s tím soubor získá tzv. „*content identifier*" (*CID*), což je identifikátor, který tvoří zvláštní adresu, kryptografický hash, založenou na samotných datech souboru. Tento princip se nazývá „*content addressing*". (Hayward 2021; Protocol Labs [b.r.])

```
bafkreih4hf5ai35zdnond7bgldopebnpssihffvbci3oe4amvte2xpjfse
```

Obrázek 2 Příklad identifikátoru CID v IPFS (práce autora)

S tím je potřeba zmínit dvě důležité vlastnosti CID: (Protocol Labs [b.r.])

- Totožný soubor, přidaný do dvou odlišných uzlů se stejnou konfigurací, bude mít totožné CID.
- Změnou obsahu souboru se změní i jeho CID.

Content-addressing nám umožňuje takzvaně „prokázat původ dat" právě tím, že CID ve své podstatě odkazuje na data a nejedná se o náhodnou adresu. (Nabben 2022) Pokud bychom například do IPFS nahráli určitý veřejný dokument, a následně odkaz na něj zveřejnili na webových stránkách pro širokou veřejnost, tak máme jistotu, že pod daným odkazem bude navždy tentýž dokument. Jak uvádí Hayward: *„IPFS není založen na blockchainu, ale je podobně neměnný: obsah nelze změnit, jinak by se změnil i samotný hash."* (Hayward 2021)

3.2.4 Směrování a protokol Bitswap

K lokalizaci uzlů a souborů (bloků) v síti používá IPFS distribuovanou tabulku hashů (*DHT*). Tato tabulka je distribuovaná po celé síti IPFS, a obsahuje data typu

klíč-hodnota („*key-value*“) potřebná ke zjištění síťových adres ostatních uzlů a také k nalezení uzlů, které obsahují hledané bloky. Zjednodušeně lze tuto tabulku definovat jako popis toho, *kdo* má *jaká data*. (Protocol Labs [b.r.])

Pro samotnou výměnu bloků dat pak IPFS používá protokol Bitswap. Dvěma hlavními úkoly protokolu Bitswap je získávání bloků vyžádaných klientem, a odesílání bloků (ve vlastnictví daného uzlu, ve kterém je protokol použit) ostatním uzlům, které tyto bloky vyžadují. (Protocol Labs [b.r.]; Rocha et al. 2021)

3.2.5 Pinning souborů

Soubory nahrané do IPFS nejsou automaticky trvale uložené. Persistence souborů v rámci IPFS uzlů totiž funguje podobně jako cache. Pokud soubor není na daném uzlu využíván (stahován), je automaticky po nějaké době vyřazen integrovaným „garbage collectorem“, aby se uvolnilo místo na disku. Soubory, které jsou často využívané, se můžou automaticky replikovat do více nezávislých uzlů, čímž se stanou víceméně trvale dostupnými. Nicméně, pokud se soubor přestane v daných uzlech využívat, bude garbage collectorem odstraněn. (Protocol Labs [b.r.])

Chceme-li mít soubor uložený v daném uzlu trvale, musíme jej *připnout* (angl. „*pin*“). Tím je zajištěno, že soubor nebude garbage collectorem odstraněn. (Politou et al. 2020) Tento krok však sám o sobě negarantuje trvalou dostupnost – ta je totiž přímo závislá na fungování daného uzlu. Za předpokladu, že by soubor nebyl replikován do dalších uzlů, tak by výpadek či záměrné vypnutí daného uzlu znamenalo zmizení souboru.

Pro skutečné zajištění dostupnosti souborů, i těch nevyužívaných, proto existují alternativní cesty, speciální „*pinning*“ služby, které můžeme rozdělit do dvou kategorií:

- **Centralizované:** Jedná se o různé soukromé subjekty, firmy, provozující IPFS uzly na svých serverech. Zdarma či za poplatek je možné soubory nahrát do jejich IPFS uzlů, a zajistit trvalé připnutí a dostupnost těchto souborů. Je však nezbytné zmínit, že ačkoliv můžou být soubory u daného subjektu uložené decentralizovaně, na více serverech a IPFS uzlech, z

obecného meřítka jsou uložené centralizovaně, výhradně na infrastruktuře tohoto subjektu.

- **Decentralizované:** Zpravidla se jedná o různé decentralizované systémy, například kryptoměnu *Filecoin*, kde jakýkoliv uživatel může provozovat IPFS uzly, a výměnou za jednotky této kryptoměny nabízet úložiště pro *připnuté* („*pinned*“) soubory. Tento princip je podrobněji rozebrán v podkapitole *Filecoin* (3.3.1).

3.2.6 Výhody IPFS

V praktickém využití se IPFS velmi podobá již existujícímu protokolu HTTP a na něm postavených systémech. Nicméně, ve svém jádře je IPFS velmi odlišné. IPFS funguje jako decentralizované a distribuované úložiště, což přináší tyto výhody:

- **Vysoká propustnost**
  Každý soubor nahraný do IPFS je rozdělen na více částí, a tyto části mohou být podle využití distribuovány do velkého množství uzlů napříč celou sítí IPFS. Při stahování souboru se klient připojí k mnoha uzlům obsahujícím části daného souboru a paralelně tyto části stahuje. To může přinést výhodu v podobě vysoké rychlosti, protože rychlost nezávisí na datové propustnosti jednoho bodu (jako tomu je zpravidla u modelu klient-server), nýbrž na celé skupině uzlů. Rychlost stahování však závisí na maximální rychlosti připojení daného klienta.
- **Decentralizace a distribuce**
  Absence centrálního bodu selhání, v kombinaci s distribucí dat napříč celou sítí a interní bezpečnostní politikou, dělá IPFS velmi odolné proti útokům, manipulaci s daty, výpadkům či snaze o cenzuru nebo zamezení fungování. (Doan et al. 2022)

3.2.7 Nevýhody IPFS

Vzhledem k poměrně nestandardnímu principu fungování, malému rozšíření a současnému stavu poskytování internetu, můžou být kladné vlastnosti i nevýhodami:

- **Vysoká propustnost – vysoká spotřeba dat**
  IPFS, stejně tak jako jiné P2P sítě, spotřebovává velké množství dat. To může být problém u uživatelů, kteří mají rychlostně či datově omezené připojení k internetu, například skrze tzv. *FUP* (*„Fair Use Policy"*) limit. Obecně se totiž předpokládá, že každý uživatel IPFS bude zároveň poskytovatelem prostoru (uzlů).
- **Malé rozšíření**
  Vzhledem k tomu, že technologie IPFS je v tuto chvíli poměrně nová, pro běžného uživatele zatím komplikovaná, nemá IPFS tak velké rozšíření. Tím pádem do sítě není připojeno tak velké množství uzlů. To představuje komplikaci pro distribuci dat a jejich hlubší decentralizaci. Můžou tedy nastat komplikace v podobě nižší rychlosti přenosu dat či vysoké odezvy před nalezením hledaného souboru v IPFS.
- **Nešifrovaná data souborů**
  IPFS šifruje přenos dat mezi uzly, nikoliv však data samotná. Kdokoliv, kdo má CID souboru, si může tento soubor stáhnout a prohlížet jeho data. Šifrování souborů je tedy potřeba provádět samostatně, před nahráním do IPFS. (Protocol Labs [b.r.])

3.2.8 Využití IPFS

Vzhledem k tomu, že IPFS je ve své podstatě souborový systém, jsou jeho možnosti využití velmi rozsáhlé, tak jako u standardních souborových systémů. Podle Beneta může IPFS sloužit:

1. *„Jako globální souborový systém, pod /ipfs a /ipns.*
2. *Jako osobní připojená sdílená složka, která automaticky verzuje, publikuje a zálohuje jakékoliv zápisy.*
3. *Jako systém pro sdílení šifrovaných souborů a dat.*
4. *Jako verzovaný správce balíčků pro veškerý software.*
5. *Jako kořenový souborový systém virtuálního stroje.*

6. *Jako bootovací souborový systém virtuálního stroje (pod hypervizorem).*
7. *Jako databáze: aplikace můžou zapisovat přímo do datového modelu Merkle DAG, a získat tím veškeré verzování, caching a distribuci, kterou IPFS nabízí.*
8. *Jako propojená (a šifrovaná) komunikační platforma.*
9. *Jako CDN pro velké soubory s kontrolou integrity (bez SSL).*
10. *Jako šifrovaná CDN.*
11. *Na webových stránkách, jako webová CDN.*
12. *Jako nový permanentní web, kde odkazy neumírají.“* (Benet 2019)

3.2.9 Přístup k souborům v IPFS

Kvůli v současnosti nízkému rozšíření a integrace IPFS v internetu je potřeba využít specifické cesty, jak soubory z IPFS získat, respektive stáhnout. Těmito cestami mohou být:

- Spuštění vlastního IPFS uzlu v počítači či s využitím volně dostupných standardizovaných implementací v různých programovacích jazycích. Uzel lze následně použít k mnoha operacím v síti IPFS, zvláště pak ke stažení libovolných souborů. Takový uzel je mimo jiné možné spustit i v rámci webové aplikace ve webovém prohlížeči.
- Použití webového prohlížeče, který již IPFS podporuje – například Brave nebo Opera. Tyto webové prohlížeče umožňují přístup k IPFS skrze protokol `ipfs://` .
- Použití tzv. *veřejné brány* („*public gateway*“), což jsou speciální domény fungující jako veřejné „brány“ do IPFS, které se navenek chovají stejně jako klasické webové servery. (Doan et al. 2022) Jednou z takových bran je https://cloudflare-ipfs.com/ipfs/. Tuto bránu je možné použít tak, že za lomítko uvedené adresy vložíme naše CID (`https://cloudflare-ipfs.com/ipfs/`***`<CID>`***) a výslednou adresu otevřeme.

Obrázek 3 Příklad získání souboru obrázku z IPFS pomocí veřejné brány (práce autora)

3.2.10 Popularita a rozšíření IPFS

Popularita IPFS se v posledních letech konstantě zvyšuje. Podle současných statistik veřejné brány https://ipfs.io/ipfs/ používá tuto bránu každý týden přibližně 230 000 uživatelů, respektive *peerů*, přičemž je přeneseno více než 125 terabajtů dat v 805 miliónech požadavcích. (Doan et al. 2022)

IPFS zároveň získává významnou podporu v rámci různých projektů poskytujících služby v odvětví webu. Například Cloudflare, jeden z největších globálních poskytovatelů *CDN* (*Content Delivery Network*) a DNS (*Domain Name System*), začal provozovat svoji vlastní veřejnou IPFS bránu v roce 2018. (Doan et al. 2022)

3.3 Blockchain

Blockchain je v současnosti jedna z nejvíce se rozvíjejících technologií z odvětví digitální decentralizace. Blockchain lze zjednodušeně definovat jako obrovskou „účetní knihu“ transakcí (databázi), jež je distribuovaná v rámci P2P sítě. *„Každá transakce v blockchainu je kryptograficky označena a ověřena ostatními těžebními subjekty, které drží kopii kompletního záznamu o všech transakcích.“* (Sharma et al. 2020) Těžebním subjektem je v tomto kontextu myšleno jakékoliv zařízení připojené do příslušné P2P sítě, které v daném blockchainu provádí kryptografické výpočty za účelem ověřování transakcí. Samotná „*těžba*“ (ověřování transakcí) je dále zpravidla incentivní, kdy těžební subjekty získávají na oplátku jednotky

kryptoměny fungující v daném blockchainu. Tyto procesy tak zcela odstraňují potřebu centrální autority, jež by monitorovala a regulovala jednotlivé transakce či interakce. (Sharma et al. 2020)

Kromě prostého prostředku směny (kryptoměn) může být blockchain aplikován v nejrůznějších odvětvích. Jedním z nich jsou distribuovaná datová úložiště či jejich incentivní vrstvy. (Zahed Benisi et al. 2020) V této práci se budeme věnovat kryptoměně *Filecoin*.

3.3.1 Filecoin

Filecoin je speciální kryptoměna navržená jako zprostředkovatelská a zároveň incentivní vrstva sítě IPFS. Tato vrstva umožňuje klientům za poplatek v kryptoměně Filecoin (*FIL*) ukládat jejich data, a zároveň „těžebním subjektům" (provozujícím IPFS uzly) poskytovat svoji výpočetní a diskovou kapacitu výměnou za FIL. Tento postup se nazývá „*storage deal*" (doslova „dohoda o skladování"). Po uzavření *storage deal* je skrze interní algoritmy *proof-of-replication* (důkaz o replikaci napříč uzly) a *proof-of-spacetime* (dohoda o uložení dat po uvedený časový úsek) zajištěno decentralizované uložení a dostupnost nahraných souborů. (Doan et al. 2022)

Jak již bylo uvedeno, účelem IPFS je primárně poskytovat otevřenou infrastrukturu pro veřejné, často využívané soubory. Pro zajištění uložení a dostupnosti méně využívaných či zcela soukromých souborů, včetně jejich distribuce a decentralizace, je třeba využít decentralizovaných *pinning* principů. K tomu je právě možné využít kryptoměnu *Filecoin*.

4 Webové aplikace a jejich vývoj

V této části jsou definovány pojmy „front-end“ a „back-end“, a dále jsou zde popsány jednotlivé součásti webové stránky či aplikace, včetně technologií a jazyků, které jsou v současnosti používané pro jejich vývoj.

4.1 Co je front-end?

Front-end (od anglického slova „*front*“, tedy „přední“) definuje část aplikace, která se spouští na straně klienta (zařízení uživatele), a zpravidla má nějaké uživatelské rozhraní. Tato část aplikace dále může komunikovat s back-endem prostřednictvím „*Application Programming Interface*“ (*API*), a využívá jeho různých služeb, například pro získávání a ukládání dat.

4.2 Co je back-end?

Back-end (od anglického slova „*back*“, tedy „zadní“) je ta část aplikace, která je spuštěna na vzdáleném serveru. Back-end poskytuje front-endu různé služby prostřednictvím API.

4.3 Používané jazyky

Pro vývoj webových stránek či aplikací se donedávna používala rozsáhlá řada programovacích, značkovacích a skriptovacích jazyků, a to jak na front-endu, tak i na back-endu. Nicméně, během posledních let se množství takových jazyků výrazně zredukovalo na menší množinu, která nyní odvětví dominuje.

4.3.1 HTML

Hypertext Markup Language (*HTML*) je značkovací jazyk používaný pro tvorbu webových stránek, respektive dokumentů. V HTML se používají tzv. *elementy* (*tagy*) pro sémantickou definici jednotlivých prvků stránky a jejich obsahu. Obecně princip tvorby webových stránek spočívá ve vytváření jednotlivých HTML dokumentů (*.html* souborů), v nichž je pomocí tagů definována struktura a obsah stránky, a případně i hypertextové odkazy na jiné stránky (*.html* soubory). V rámci HTML souboru je možné v případě potřeby používat i jazyky *CSS* a *JavaScript*.

4.3.2 CSS

Cascading Style Sheets (*CSS*) je speciální jazyk, pomocí něhož lze definovat, jak se mají jednotlivé HTML elementy, případně jejich množiny, zobrazovat. Jazyk CSS umožňuje aplikovat na HTML elementy velmi rozsáhlé množství vizuálních a interaktivních vlastností.

Mimo jiné lze pomocí jazyka CSS také dosáhnout tzv. responzivity – schopnosti webové stránky reagovat na rozměry okna webového prohlížeče či displeje za účelem přizpůsobení obsahu stránky pro dané rozměry. Díky tomu lze webové stránky implementovat tak, že si je můžeme bez problému zobrazit jak na velkém monitoru stolního počítače, tak i na malém displeji chytrého telefonu.

4.3.3 JavaScript

JavaScript (zkráceně také *JS*) je objektově orientovaný a událostmi řízený multiplatformní skriptovací jazyk. Je možné jej využít jak na straně klienta (front-end), tak i na straně serveru (back-end) prostřednictvím runtime platformy Node.js.

4.3.4 Využití JavaScriptu na straně klienta

Na straně klienta, webového prohlížeče, je JavaScript používán pro psaní skriptů, jež dělají webové stránky více dynamické a interaktivní. Pomocí JavaScriptu lze kompletně manipulovat s HTML elementy, jejich styly (CSS), a také reagovat na veškeré události (*eventy*), které uživatel na webové stránce vyvolá (např. kliknutím nebo pohybem myší), ale i události, které souvisí s interními procesy (např. načtení stránky).

Skripty je možné psát jak „manuálně“ (napsat skript a připojit jej přímo do webové stránky), tak i za pomoci nejrůznějších knihoven a frameworků. Jedním z takových frameworků může být například *Next.js* – nadstavba knihovny *React* sloužící pro psaní webových aplikací. Poté, co je webová aplikace za pomoci tohoto frameworku implementována, kód je v několika různých vrstvách zkompilován a optimalizován. Následně je vygenerována řada „zdrojových“ souborů (skriptů),

které webová aplikace využívá. Načítání těchto souborů si zajišťuje knihovna či framework zpravidla sám.

4.3.5 Využití JavaScriptu na straně serveru

S využitím speciální runtime platformy Node.js lze JavaScript spouštět i jako samotný proces, například na serveru. Tímto způsobem lze tedy naprogramovat kompletní serverovou aplikaci (např. API).

4.3.6 Typovaná nadstavba JavaScriptu – TypeScript

JavaScript není tzv. typovaný jazyk – veškeré proměnné jsou dynamicky typované, což v praxi znamená, že datový typ proměnné se může za běhu skriptu libovolně měnit. Tato vlastnost umožňuje vývojáři více komfortně programovat. Nicméně to přináší i značné nevýhody v podobě velké náchylnosti k chybám za běhu skriptu, a nepřehlednosti kódu, zvláště ve velkých a komplexních aplikacích. Z tohoto důvodu v roce 2012 představila společnost Microsoft nadstavbu JavaScriptu nazvanou *TypeScript*. TypeScript je téměř totožný s jazykem JavaScript, s tím rozdílem, že umožňuje (zpravidla dokonce vyžaduje) používat typy. TypeScript nabízí široké možnosti typování, včetně enumerací, generických typů a různých vestavěných operátorů a utilit. (Richards et al. 2015; Bierman et al. 2014)

Jelikož TypeScript je pouze nadstavba JavaScriptu – TypeScript sám o sobě není spustitelný, je potřeba jej zkompilovat (přeložit) do JavaScriptu. Z tohoto důvodu se používá pouze při vývoji, kde probíhá kontrola typů jak při psaní kódu, tak při jeho kompilaci. Po zkompilování kódu je výsledkem opět čistý JavaScript (spustitelný např. ve webovém prohlížeči), který je ale právě kvůli předcházející kontrole typů mnohem méně náchylnější na chyby. (Richards et al. 2015)

4.4 Document Object Model (DOM)

Document Object Model (*DOM*) je rozhraní, objektové vyjádření dokumentu HTML webové stránky, umožňující v JavaScriptu číst a manipulovat s obsahem dokumentu. Pomocí tohoto rozhraní je možné dynamicky upravovat obsah webové stránky, provádět akce a reagovat na různé události při běhu webové stránky. (VOXCAFE s.r.o. 2021)

4.5 NPM balíčky

Při vývoji aplikací na výše zmíněné platformě Node.js je možné využít i tzv. balíčky (*packages*), a to zejména prostřednictvím správce balíčků *NPM* (*Node Package Manager*). NPM je rozsáhlá on-line databáze open-source balíčků – utilit, skriptů, knihoven, frameworků a dalších. Velké množství těchto balíčků je napsáno právě v jazyce JavaScript či TypeScript, a je možné je libovolně využívat například při vývoji webových stránek či dalších aplikací.

5 React

React (alternativně nazýván také *ReactJS* nebo *React.js*) je open-source JavaScriptová knihovna pro vytváření webových aplikací – front-endů. React.js umožňuje rychle vytvořit robustní, modulární a reaktivní (interaktivní) webové aplikace. (VOXCAFE s.r.o. 2021)

5.1 Definice Single Page Application (SPA)

Základní myšlenka za fungováním webových stránek je ta, že mezi jednotlivými stránkami je navigováno pomocí hypertextových odkazů, kdy každý takový odkaz směřuje na konkrétní *.html* dokument. Každý z těchto dokumentů je pak v podstatě vlastní webovou stránkou. Kliknutím na odkaz se tak spustí načítání celé další webové stránky (webový prohlížeč indikuje načítání).

React tuto myšlenku však nahrazuje přístupem zvaným *Single Page Application* (SPA). SPA označuje webovou stránku, přesněji webovou aplikaci, která používá pouze jeden vstupní HTML dokument, typicky *index.html*. Veškerá navigace nebo aktualizace obsahu v takové aplikaci pak probíhá pouze v rámci tohoto jednoho HTML dokumentu a bez načítání stránky. O to se interně stará React, respektive framework na něm postavený, který manipuluje pouze s DOMem tohoto dokumentu. Samotné zdrojové soubory (skripty, CSS, obsah stránky) mohou být navíc v SPA stahovány až podle potřeby – například až poté, co uživatel přejde na stránku, kde jsou tyto soubory potřeba. (Solovei et al. 2018)

Velkou výhodou tohoto přístupu je zejména rychlost při navigaci mezi jednotlivými podstránkami webové aplikace, a to ze dvou následujících důvodů:

1. Navigace mezi podstránkami je rychlá, protože webový prohlížeč nenačítá celou stránku a ani neindikuje načítání.

2. Po prvotním načtení webové aplikace se načítají pouze dodatečné zdrojové soubory, které dané podstránky využívají. Tím je minimalizován přenos dat, což zvyšuje rychlost načítání podstránek, a zároveň například šetří mobilní data uživatelům přistupujícím k webové aplikaci z mobilního telefonu.

5.2 Virtual DOM

Práce s rozhraním DOM v prohlížeči je typicky z hlediska výkonu pomalá. *„Pokaždé, když se DOM změní, prohlížeč bude muset přepočítat CSS, spustit rozložení a překreslit webovou stránku.“* (VOXCAFE s.r.o. 2021) Z tohoto důvodu využívá React vlastní odlehčenou reprezentaci DOMu, tzv. *Virtual DOM*.

Účelem Virtual DOM je snížení dopadu na výkon při aktualizacích skutečného DOMu. Kdykoliv React detekuje změnu (například změnu stavu v komponentě), vytvoří na základě této změny nový Virtual DOM. Poté je procesem „DOM-diffing“ (alternativně „reconciliation“) porovnán současný Virtual DOM a nově vytvořený Virtual DOM za účelem nalezení rozdílů. Posledním krokem je aktualizace skutečného DOMu, kdy jsou aktualizována jenom ta místa, kde došlo ke změnám. (Persson 2020)

5.3 JavaScript XML (JSX)

JavaScript XML (*JSX*) je rozšíření standardního JavaScriptu, které nám umožňuje komponenty v Reactu psát, respektive strukturovat, stejně jako HTML. Pomocí JSX můžeme přímo v JavaScriptu využívat syntaxi v podobě XML (HTML) tagů. (Arancio 2021)

```
const elem = <div>Element</div>;
```

Obrázek 4 Zápis HTML elementu v JavaScriptu pomocí JSX (práce autora)

Při kompilaci je následně syntaxe JSX přeložena do čistého JavaScriptu – volání funkce pro vytvoření HTML elementu v rámci knihovny React.

```
const elem = React.createElement('div', null, "Element");
```

Obrázek 5 Výsledný JavaScript kód po kompilaci z JSX (práce autora)

JSX není nezbytnou součástí Reactu, nýbrž způsobem, jak výrazně zjednodušit syntaxi komponent pomocí strukturování kódu stejně jako HTML, namísto složitého vnořování funkcí. Porovnání stejné struktury zapsané jak v čistém JavaScriptu, tak pomocí JSX, můžeme vidět na příkladech níže.

```
React.createElement("div", { className: "wrapper" }, [
  React.createElement("h1", { className: "heading" }, "Nadpis"),
  React.createElement("p", { className: "content" }, "Text"),
]);
```

Obrázek 6 Zápis struktury elementů v čistém JavaScriptu (práce autora)

```
<div className="wrapper">
  <h1 className="heading">Nadpis</h1>
  <p className="content">Text</p>
</div>
```

Obrázek 7 Zápis struktury elementů pomocí JSX (práce autora)

5.4 Komponenty

Architektura webových aplikací implementovaných pomocí Reactu spočívá v rozdělení těchto aplikací do menších součástí – komponent. Tyto komponenty jsou od sebe izolované, mají své proměnné a případně i svůj vnitřní stav. Komponenty by měly být, pokud možno, znovupoužitelné („*reusable*“). Příkladem znovupoužitelné komponenty může být například tlačítko. Komponentu tlačítka implementujeme pouze jednou a následně tuto komponentu opakovaně používáme, kde potřebujeme. Jednou z výhod takového přístupu je, že pokud potřebujeme tlačítko upravit, stačí nám změnu provést v jeho komponentě. Tyto změny se následně projeví všude, kde je tato komponenta použita. (VOXCAFE s.r.o. 2021)

Komponenty se v Reactu donedávna standardně psaly jako JavaScriptové třídy neboli „*class komponenty*“. Funkcionální přístup (komponenty definované jako funkce) bylo možné také použít, nicméně pouze u bezstavových („*state-less*“) komponent. *„Dlouho platilo to, že pokud chceme, aby komponenta měla vnitřní stav či životní cyklus, musíme zvolit třídu.“* (Kříž 2019) Od verze Reactu 16.8 je však možné používat tzv. „*lifecycle*“ funkce i v rámci funkcionálních komponent – tyto komponenty tedy již mohou mít také vnitřní stav. Funkcionální přístup je dnes preferovaný způsob, jakým se v Reactu definují komponenty. Dělá totiž strukturu komponent přehlednější, a může být obecně jednodušší na implementaci.

Každá komponenta může mít libovolné vstupní parametry, které v kontextu Reactu nazýváme *props* (zkratka pro „*properties*“).

```
export const Button: React.FC = ({ children }) => {
  return <button className="ui-button">{children}</button>;
};
```

Obrázek 8 Ukázka kódu komponenty v Reactu (práce autora)

5.5 Životní cyklus (lifecycle)

Na úvod je nezbytné zmínit, že React využívá jednosměrného toku dat. To znamená, že rodičovské komponenty předávají data do komponent podřízených – podřízené komponenty se tak mění na základě těchto vstupních dat. Podřízené komponenty nemohou přímo měnit vstupní parametry rodičovských komponent.

Každá komponenta v Reactu má vlastní životní cyklus (lifecycle). Tento životní cyklus se dá rozdělit do tří obecných fází: (Baranowski 2022)

- **Mount** – počátek (zavedení) komponenty
- **Update** – změny komponenty v průběhu běhu aplikace
- **Unmount** – zánik (odebrání) komponenty

V první fázi *mount* dochází k prvotnímu zavedení komponenty (zavolání její funkce), s tím spojené inicializaci vnitřních stavů („*state*“) na základě výchozích vstupních parametrů, po které následuje *render* (vykreslení) do DOMu.

Fáze *update* zastřešuje aktualizace komponenty při změnách vstupních parametrů či interních stavů. React při těchto aktualizacích interně porovnává, zda došlo ke změnám ve výstupu komponenty, a podle toho překreslí („re-render“) komponentu v DOMu. Dokud je komponenta zavedena (mounted), může docházet k její aktualizaci. (Baranowski 2022)

Poslední fáze *unmount* je ukončením životního cyklu komponenty. Při této fázi je proveden úklid („clean-up“), jehož procesy jsou definované interně Reactem samotným, ale mohou být podle kontextu připravené i vývojářem, například za

účelem odebrání *event listenerů*, které by jinak způsobovaly problémy. Nakonec je komponenta odebrána z DOMu.

Z důvodu, že komponenty se během „svého života“ zpravidla několikrát aktualizují, nelze ukládat vnitřní stavy (hodnoty) do proměnných pouhým přiřazením. Při následné aktualizaci by totiž došlo k opětovnému přiřazení hodnoty, tedy v podstatě k resetování dané proměnné. Pro práci s vnitřními stavy a obecně životním cyklem komponenty je tak potřeba ve funkcionálních komponentách používat tzv. „*hooky*“.

5.6 Hooky

Hooky jsou speciální funkce v Reactu, které zpřístupňují správu vnitřního stavu a životního cyklu i mimo *class komponenty*. Tato funkcionalita byla přidána v Reactu verze 16.8 a poskytla tak vývojářům možnost psát plnohodnotné funkcionální komponenty, namísto původních *class komponent*. (Baid et al. 2022) Názvy hooků vždy začínají klíčovým slovem „*use*“, například *useState*. (Luojus 2021) Kromě vestavěných React hooků si může vývojář libovolně definovat hooky vlastní.

V současné době React obsahuje 15 vestavěných hooků. Některé, v praxi nejpoužívanější hooky, jsou charakterizovány v následujících podkapitolách.

5.6.1 useState

Tento hook je zásadní pro ukládání a spravování vnitřního stavu ve funkcionálních komponentách. Chceme-li nějakou hodnotu uchovat napříč aktualizacemi, či na základě změny této hodnoty vyvolat aktualizaci a re-render, musíme tuto hodnotu uložit použitím *useState*.

```
export const Button: React.FC = () => {
  const [value, setValue] = useState(0); // inicializace stavu „value”

  const click = () => {
    // zvýšení „value“ o 1 při kliknutí na tlačítko
    setValue(value + 1);
  };

  // výpis aktuální hodnoty při každé změně stavu „value“
  return <button onClick={click}>Clicked: {value}</button>;
};
```

Obrázek 9 Příklad použití hooku *useState* (práce autora)

5.6.2 useEffect

Pomocí tohoto hooku můžeme spravovat tzv. vedlejší účinky („*side effects*“), v reakci na životní cyklus komponenty nebo při změnách vstupních dat či stavů. Uvnitř tohoto hooku může být například zavoláno načítání dat z externí API nebo přidání a odebrání *event listenerů*. (Danthasinghe 2020)

Funkce hooku *useEffect* přijímá dva parametry:

- Funkci, která se má zavolat. V této funkci můžeme i podle potřeby vracet (*return*) funkci, která se má provést při odebírání (unmount) komponenty za účelem úklidu (clean-up).
- *Dependencies* neboli pole hodnot, jejichž změny mají vyústit v zavolání uvedené funkce. Pokud je toto pole prázdné, bude funkce zavolána jen jednou, při zavedení (mount) komponenty.

```
useEffect(() => {
  fetchTodos();

  // Tento useEffect bude zavolán pouze při zavedení komponenty,
  // jelikož pole 'dependencies' je prázdné (fáze mount)
}, []);

useEffect(() => {
  console.log("Value has changed!");

  // Při každé změne hodnoty 'value' bude zavolán tento useEffect
  // (fáze update)
}, [value]);

useEffect(() => {
  window.addEventListener("keydown", handleUserKeyPress);

  // Zde je navrácena tzv. clean-up funkce pro odstranění
  // event listeneru při zániku komponenty
  // (fáze unmount)
  return () => {
    window.removeEventListener("keydown", handleUserKeyPress);
  };
}, []);
```

Obrázek 10 Příklady použití hooku *useEffect* v kontextu fází životního cyklu (práce autora)

6 Návrh webové aplikace a výběr technologií

Tato kapitola se zaměřuje na návrh webové aplikace využívající technologii IPFS pro decentralizované sdílení souborů přes internet. Součástí je návrh základních parametrů a struktury webové aplikace, a dále jsou představeny technologie zvažované pro její implementaci.

6.1 Základní parametry webové aplikace

Cílem webové aplikace je mimo jiné zpřístupnění možnosti decentralizovaného sdílení digitálního obsahu pro široké spektrum uživatelů internetu, což můžou podpořit následující dva parametry popsané v jednotlivých podkapitolách.

6.1.1 Jednoduchý design a přístupnost

Příjemná interakce s webovou stránkou je zásadní součástí kvalitního uživatelského zážitku. Toho může být dosaženo jak vizuálním designem, tak i nepřímo pomocí optimálního návrhu uživatelského rozhraní, ve kterém se uživatel může jednoduše orientovat. (Jongmans et al. 2022) Design a rozhraní webové aplikace by zároveň mělo být responzivní. To znamená, že webovou aplikaci bude možné bez problému používat na široké škále zařízení s různou velikostí obrazovek – počítač s širokoúhlou obrazovkou, notebook, tablet nebo i mobilní telefon.

Kromě obecného designu je potřeba věnovat pozornost i správné implementaci z hlediska přístupnosti, zejména pro osoby s různými zdravotními postiženími. Přístupnost na webu lze implementovat do různé hloubky, nicméně mezi základní principy patří následující: (Ronen 2022)

1. Popisné „alt“ (alternative) texty, které popisují obsah na stránce, například obrázky či ikony.
2. Správná struktura nadpisů – tagů **H1** až **H6**. Webová stránka by typicky měla obsahovat pouze jeden H1 tag. Pro ostatní podnadpisy by se měli využívat poté tagy H2 až H6.

3. Používání sémantiky tzv. **ARIA** (Accessible Rich Internet Applications). ARIA je rozšíření HTML, které umožňuje jednotlivým elementům přidávat širokou škálu atributů pro zlepšení přístupnosti a kontextuální provázanosti mezi různými elementy.

6.1.2 Rychlost a optimalizace

Rychlost se v kontextu webových stránek a aplikací měří z více úhlů pohledů:

1. rychlost prvotního načtení stránky (také tzv. **PLT** – *Page Load Time*)
2. rychlost načítání obsahu stránky během prohlížení (obrázky, ikony apod.) či při navigaci mezi dalšími podstránkami
3. rychlost reakcí stránky (aplikace), tj. jak je webová stránka (aplikace) optimalizovaná na úrovni kódu

Rychlost prvotního načtení stránky (1) a rychlost načítání obsahu během prohlížení (2) závisí na dvou hlavních faktorech – rychlost serveru, který stránku provozuje, a velikost jednotlivých souborů stránky (HTML kód, CSS styly, JavaScript, obrázky).

Chceme-li docílit vysoké rychlosti na úrovni serveru (hostingu), můžeme využít tzv. *edge hosting*. Takový hosting spočívá v replikaci stránky do mnoha serverů v různých geografických lokacích za účelem zmenšení vzdálenosti mezi serverem a uživatelem. Menší vzdálenost znamená nižší latenci a tím pádem rychlejší načítání.

6.2 Funkce webové aplikace

Hlavním účelem webové aplikace je poskytnout uživatelům jednoduchý způsob, jak nahrávat a stahovat soubory do/ze sítě IPFS. Aby obsah nahraných souborů mohl být i skryt, aplikace by měla uživateli volitelně nabízet možnost šifrování, včetně automatického dešifrování při stahování souborů.

Po každém nahrání souboru by měl být vygenerován unikátní odkaz, volitelně obsahující dešifrovací klíč, který uživatel může následně zaslat komukoliv za účelem sdílení souboru. Při otevření tohoto odkazu by jej měla aplikace

automaticky zpracovat a předvyplnit do příslušného textového pole tak, aby uživatel mohl zahájit stahování.

Mimo zmíněnou funkcionalitu by měla aplikace obsahovat i modul, ve kterém bude uživateli poskytnut přehled o nahraných souborech, včetně možnosti jejich stažení či zkopírování odpovídajících odkazů.

6.3 Zvažované technologie a postupy

Webová aplikace je postavena na knihovně React, přičemž pro její vývoj je použit výhradně TypeScript a řada doplňujících open-source balíčků z NPM. Tato část některé zvažované technologie a postupy charakterizuje.

6.3.1 Node.js

Node.js je runtime platforma, která umožňuje spouštět JavaScript kód mimo webový prohlížeč. Platforma Node.js je využita primárně během vývoje, jakožto nezbytné vývojové prostředí.

6.3.2 Next.js

Next.js je framework postavený na knihovně React, umožňující vyvíjet kompletní webové aplikace v různých měřítkách – od jednoduchých front-end aplikací, až po kompletní full-stack aplikace, včetně back-endu. Kromě toho Next.js obohacuje React o řadu postupů a nástrojů, které zrychlují, zjednodušují a optimalizují vývoj a následné fungování webových aplikací. (Sasikumar et al. 2022)

6.3.3 Optimalizace CSS s využitím Tailwind CSS

Standardním způsobem, jakým se implementují kaskádové styly v rámci webových stránek, je připojení CSS souboru do této stránky. V tomto souboru se následně nachází různé selektory (typ elementu, třída, ID atd.) s definicemi stylů, které jednotlivé elementy na stránce stylují. V Reactu (Next.js) je z důvodu izolace žádoucí tyto CSS soubory definovat pro každou komponentu/část zvlášť. To však může vyústit v redundantní definici stejných nebo podobných stylů napříč komponentami, a tím i zbytečnému navýšení velikosti kódu. Jednou z metod, jak tomu předejít, je použití *Tailwind CSS*.

Tailwind CSS je poměrně nový CSS framework, který nám umožňuje psát tzv. *utility-first* CSS, s předdefinovanými CSS třídami. (Klimm 2021) Použitím Tailwind CSS lze tak teoreticky zcela nahradit explicitní CSS soubory, a aplikovat na HTML elementy pouze předdefinované CSS třídy. Tailwind CSS zároveň umí kód sledovat, a při kompilaci CSS zahrne pouze ty CSS třídy, které byly použity.

6.3.4 Web Crypto API

Web Crypto API je rozhraní, které je součástí JavaScriptových prostředí, včetně webového prohlížeče. Toto rozhraní zpřístupňuje různé metody pro přístup k nízko-úrovňovým kryptografickým primitivům, mimo jiné k šifrování. (Halpin 2014)

6.3.5 Estuary.tech

Estuary.tech je experimentální platforma provozující API pro nahrávání souborů do IPFS, včetně jejich distribuce do vícero IPFS uzlů, na základě automatizovaných dohod o skladování v rámci blockchainu Filecoin. K provozu této API využívá zmíněná služba open-source projekt zvaný *Estuary*, což je experimentální IPFS a Filecoin uzel, jež umožňuje jednodušší vytváření dohod o skladování a následný pinning souborů v síti IPFS. (Protocol Labs [b.r.])

Otevřený repozitář zmíněného projektu je k dispozici na adrese
https://github.com/application-research/estuary

7 Implementace webové aplikace

Kapitola se věnuje implementaci navržené aplikace, pod interním názvem „*Axiom*“, a to včetně jejího nasazení do produkčního prostředí. Základním předpokladem pro vývoj aplikace je předchozí instalace runtime platformy Node.js, NPM a Yarn.

7.1 Inicializace projektu

Prvním nezbytným krokem je inicializace projektu a pracovního prostředí. Aplikace je postavena na knihovně React, konkrétně na frameworku Next.js. Pro manipulaci s NPM balíčky byl zvolen správce balíčků PNPM. Základní inicializace projektu byla provedena následujícím příkazem v terminálu:

```
yarn create next-app --typescript
```

Obrázek 11 Příkaz pro prvotní inicializaci projektu (práce autora)

Tímto příkazem se v terminálu spustí interaktivní průvodce, který umožňuje upřesnit parametry aplikace a její název (včetně názvu adresáře). Po dokončení průvodce je na zadané adresářové cestě vygenerována výchozí adresářová struktura Next.js aplikace včetně potřebných NPM modulů.

Parametr `--typescript` zároveň uvádí, že chceme projekt vygenerovat s použitím TypeScriptu.

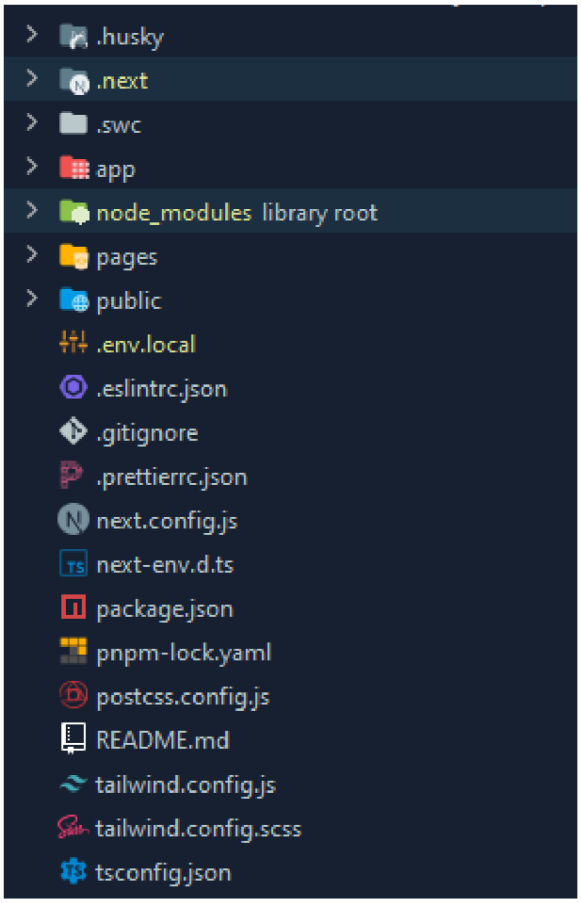
7.2 Verzování projektu

Verzování projektu je řešeno pomocí technologie *Git* na platformě *GitHub*. Verzování umožňuje mít přehled o veškerých změnách, které kdy v kódu proběhly, a odhalit tak případně chyby. Tímto způsobem se celý projekt zároveň i zálohuje. Tento krok není nezbytný, je však velice žádoucí.

Zdrojový kód aplikace je veřejně dostupný na adrese:
https://github.com/petrschmidt/axiom

7.3 Adresářová struktura aplikace

Celá aplikace je strukturována jako monolit v rámci jednoho adresáře. V kořenu tohoto adresáře se nachází různé konfigurační soubory, například pro *TypeScript*, *ESLint*, *Next.js*, *Prettier* a také tzv. *package.json* obsahující mimo jiné informace o nainstalovaných NPM balíčcích. Většina těchto konfiguračních souborů, kromě například konfiguračních souborů pro *Prettier* a *Tailwind*, je vygenerována automaticky při inicializaci projektu. Pokud nepotřebujeme nijak zvlášť upravovat chování jednotlivých odpovídajících nástrojů a knihoven, pak tyto soubory zpravidla upravovat nemusíme.



Obrázek 12 Adresářová struktura projektu (práce autora)

7.4 Vytváření stránek aplikace

Next.js nám umožňuje definovat jednotlivé cesty ve webové aplikaci již na úrovni adresářové struktury, v rámci adresáře `/pages`. Pokud bychom chtěli například na adrese `/car` zobrazit obsah stránky „Car“ (automobil), stačí nám vytvořit soubor

`car.tsx` a v něm tuto stránku implementovat. O vše ostatní se postará Next.js. Soubory a stránky můžeme i vnořovat – pokud bychom chtěli na adrese `/car/bmw` zobrazit stránku pro „BMW“ , stačí nám vytvořit podadresář `/car` a v něm soubor `bmw.tsx`.

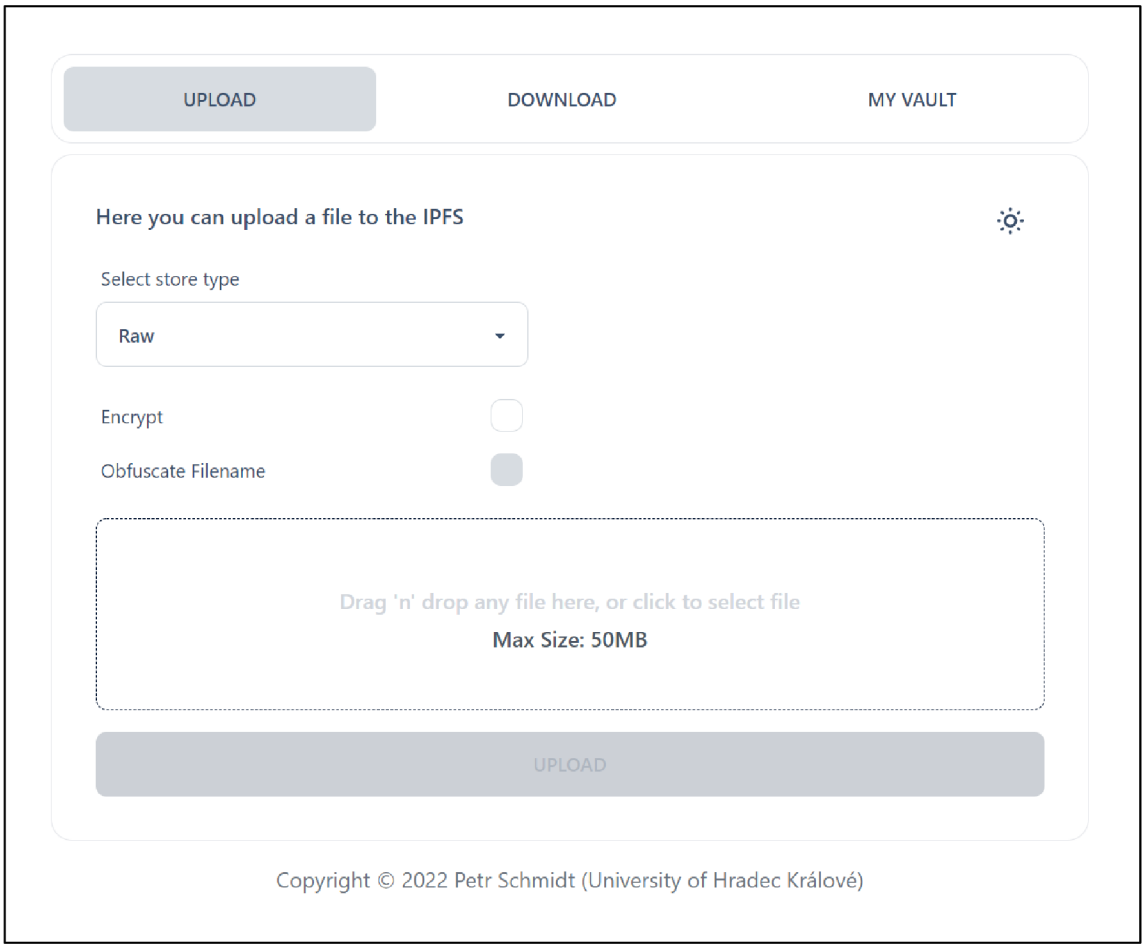
Kromě toho je možné použít i dynamické cesty, kdy název souboru definujeme pomocí hranatých závorek, a uvnitř těchto závorek poté zvolíme libovolný název „proměnné“. Takový název souboru může vypadat například takto: `[post].tsx`. Proměnnou je v tomto případě „post“. Tato proměnná je při běhu aplikace z URL extrahována a podle potřeby přístupná skrze Next.js hook zvaný *useRouter*.

V naší aplikaci jsou definované celkem tři stránky, respektive moduly:

- `/ (/index.tsx)` – výchozí modul sloužící pro nahrávání souborů
- `/download (/download.tsx)` – modul sloužící pro stahování souborů
- `/vault (/vault.tsx)` – seznam (trezor) uživatelem nahraných souborů

7.5 Základní struktura uživatelského rozhraní

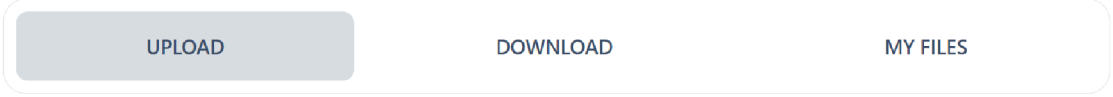
Celé uživatelské rozhraní je navrženo tak, aby bylo co nejjednodušší a minimalistické. K tomu byla mimo jiné využita knihovna *daisyUI* jakožto rozšíření Tailwind CSS. Jak lze vidět na obrázku níže, obsah všech modulů je zároveň situovaný do centrálního „okna“ uprostřed obrazovky za účelem jednodušší orientace v rozhraní. Aplikace dále nabízí možnost přepínat mezi světlým a tmavým motivem, včetně automatické detekce motivu nastaveného v zařízení uživatele.



Obrázek 13 Vzhled uživatelského rozhraní (práce autora)

7.5.1 Navigační lišta

Navigační lišta je zobrazena trvale, a umožňuje uživateli přepínat mezi jednotlivými moduly aplikace. Pro přehlednost je vždy zvýrazněn aktuální modul.



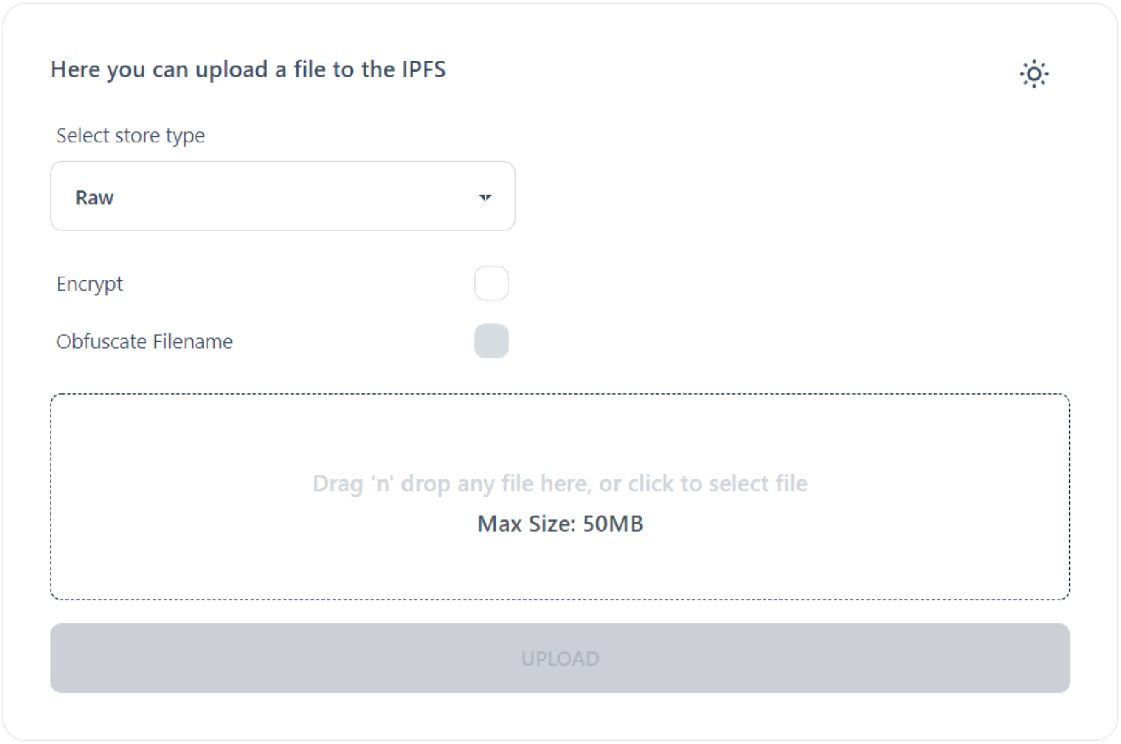
Obrázek 14 Navigační lišta (práce autora)

7.6 Moduly

Aplikace je rozdělena celkem do tří modulů (stránek): nahrávání, stahování a seznam souborů.

7.6.1 Nahrávání (upload)

Tento modul nabízí nejdůležitější funkcionalitu této aplikace. Slouží k nahrávání souborů do IPFS, a je zároveň úvodní stránkou.



Obrázek 15 Modul nahrávání souborů (práce autora)

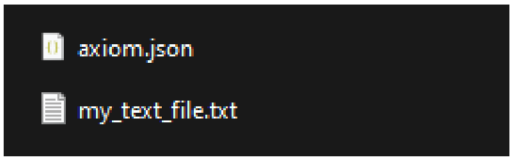
Volba způsobu uložení

V úvodu si může uživatel zvolit, jak budou soubory v IPFS uloženy. Aplikace nabízí dvě možnosti: *Raw* a *Wrapped File*.

Možnost *Raw* („surový soubor“) reprezentuje standardní způsob, jakým se soubory do IPFS běžně ukládají. Soubor je nahrán v nezměněné podobě a následně dostupný skrze vygenerovaný CID identifikátor. Může být tak rovnou stažen uživatelem (předpokládáme-li, že není šifrovaný). Značnou nevýhodou tohoto způsobu je, že neuchovává název a přípony souborů. Může se tak stát, že při

otevření tohoto souboru přes IPFS bránu se zobrazí „surový“ obsah souboru, a uživatel tak nebude vědět, v jakém formátu jej uložit.

Druhá možnost *Wrapped File* (obalený soubor) je speciálním způsobem nahrávání a stahování souborů do/z IPFS, který je proprietárně implementován v rámci této aplikace. Spočívá v obalení zvoleného souboru do ZIP formátu, společně s JSON souborem obsahujícím dodatečná metadata ke zvolenému souboru. Tento ZIP soubor – „*Wrapper*“ (obal) – je poté celý nahrán do IPFS. Metadata obsažená v tomto *Wrapperu* jsou následně při stahování z IPFS použita k „rekonstrukci“ původního souboru – obnovení jeho původního názvu, přípony a časového razítka poslední úpravy před nahráním. Obsah *Wrapperu*, metadata a vlastní soubor je možné vidět na obrázku níže.



Obrázek 16 Obsah souboru nahraného možností *Wrapped File* (práce autora)

Šifrování

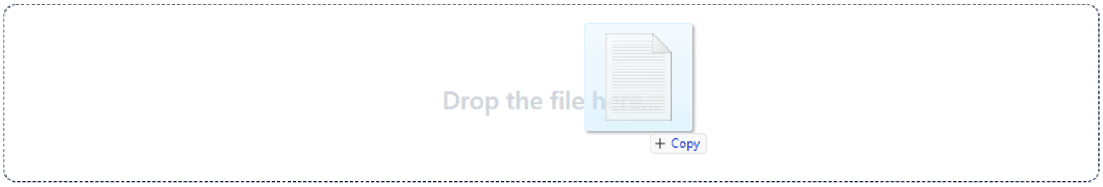
Pokud uživatel potřebuje sdílet soukromá data, může u obou zmíněných způsobů zároveň využít i *šifrování*. U obou způsobu probíhá šifrování odlišněji. U možnosti *Raw* je nahrán celý šifrovaný soubor (zcela nečitelný). Naopak u možnosti *Wrapped File* je nahrán nešifrovaný *Wrapper* (čitelná jsou pouze metadata) obsahující však šifrovaný zvolený soubor. Zde je nutné zmínit, že v případě ukládání způsobem *Wrapped File* zůstávají metadata souboru nešifrovaná, tím pádem je nešifrovaný i název souboru.

Obfuskace

Při nahrávání způsobem *Wrapped File* může uživatel chtít skrýt název souboru. K tomu slouží možnost obfuskace (znečitelnění). Při zaškrtnutí této možnosti je název souboru přepsán náhodným řetězcem znaků, a je zachována pouze jeho přípona.

Vybrání souboru

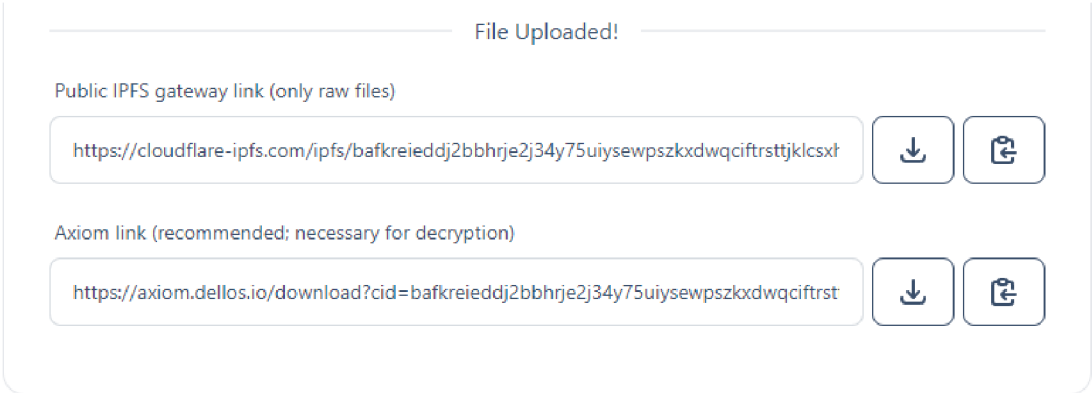
Soubor k nahrání je možné do vyhrazeného okna buď přetáhnout myší přímo z průzkumníku (tzv. „*drag 'n' drop*"), nebo kliknutím do tohoto okna, a následným zvolením souboru. Pro účely testování a prevenci zneužití této aplikace je maximální velikost souborů omezena na 50 megabajtů.



Obrázek 17 Zvolení souboru způsobem „drag 'n' drop" (práce autora)

Vygenerované odkazy po nahrání souboru

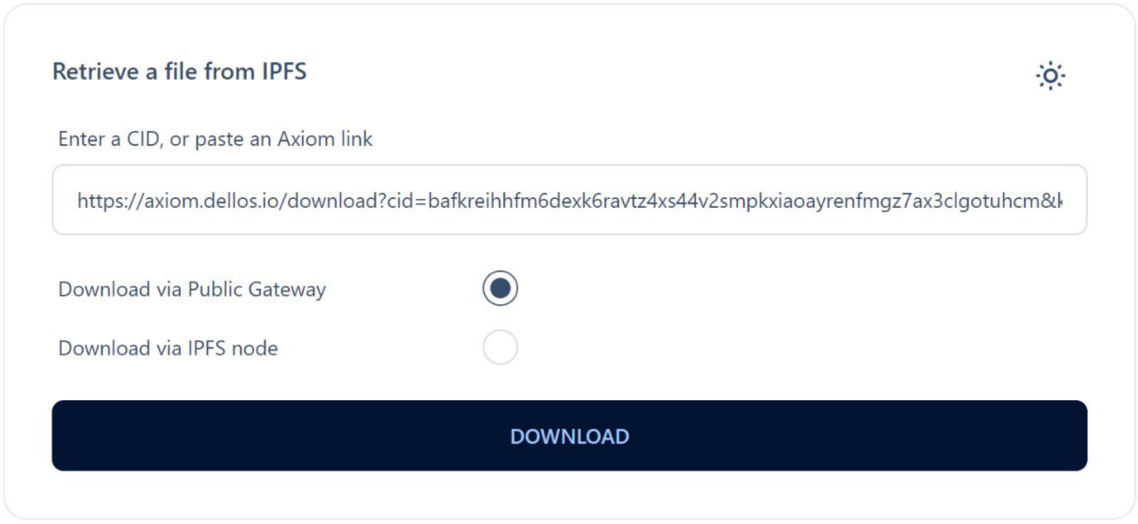
Jakmile je soubor úspěšně nahrán do IPFS, aplikace automaticky vygeneruje dva odkazy. První odkaz směřuje na veřejnou IPFS bránu a slouží výhradně pro přístup k nešifrovaným *Raw* souborům. Druhý odkaz směřuje zpět do aplikace Axiom a slouží jak k dešifrování souborů, tak k případné extrakci souboru z *Wrapperu*. Druhý odkaz je tedy potřeba použít, kdykoli je soubor šifrován, nebo nahrán způsobem *Wrapped File*. Po vygenerování mohou být oba odkazy zkopírovány a sdíleny s kýmkoliv, kdo má mít k souborům přístup.



Obrázek 18 Vygenerované odkazy k nahranému souboru (práce autora)

7.6.2 Stahování (download)

Modul stahování může být použit ke stažení veřejných souborů z IPFS, je však ale klíčový ke stahování šifrovaných souborů a souborů typu *Wrapped File*. Modul obsahuje jedno univerzální textové pole, do kterého je možné vyplnit jak samotné CID, tak i Axiom odkaz.



Obrázek 19 Modul stahování souborů (práce autora)

Stažení veřejného souboru přes identifikátor CID

Uživatel může v tomto modulu stáhnout jakýkoliv veřejně dostupný soubor z IPFS. K tomu stačí vyplnit CID identifikátor tohoto souboru do příslušného textového pole, a poté zahájit stahování kliknutím na tlačítko „Download“.

Stažení šifrovaného souboru nebo Wrapped File

Pokud byl soubor nahrán zašifrovaný nebo způsobem *Wrapped File*, musí být stažen na základě vygenerovaného Axiom odkazu. Ten totiž kromě CID obsahuje i parametry udávající způsob uložení daného souboru, a v případě šifrovaného souboru i data nezbytná pro jeho dešifrování. Tento Axiom odkaz musí být otevřen či ručně vložen do příslušného pole. Poté je možné zahájit stahování kliknutím na tlačítko „Download“.

Automatické předvyplnění

Modul stahování se může spustit ve dvou režimech:

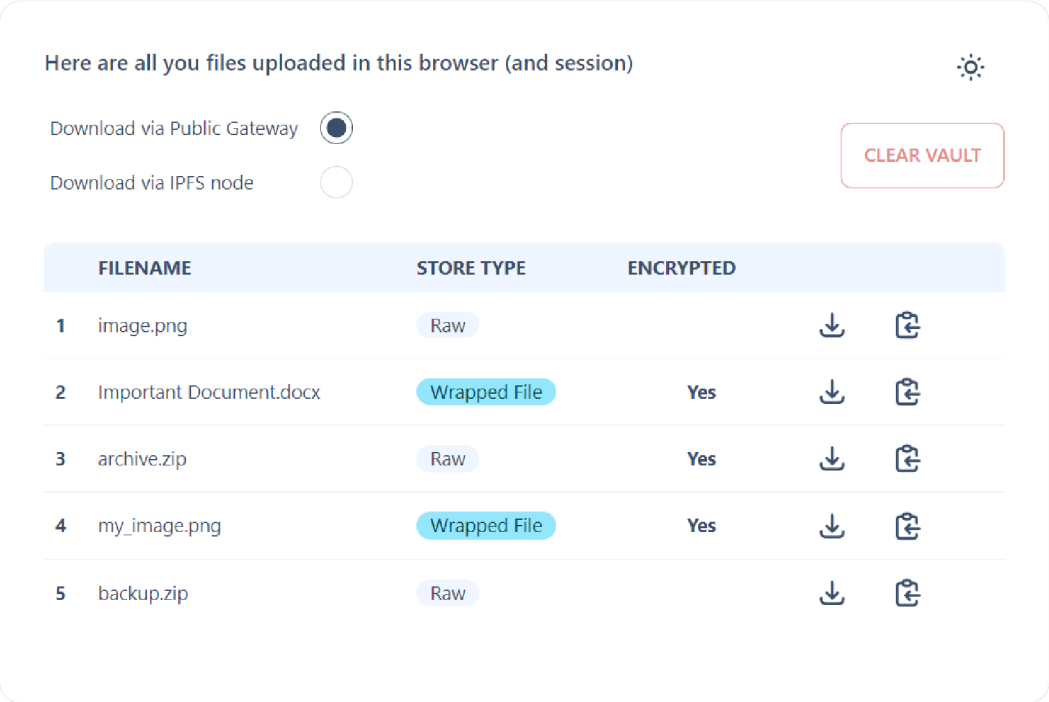
a) Uživatel naviguje na cestu `/download` bez doplňujících parametrů. Modul se v tomto režimu pouze načte, zbytek závisí na akcích uživatele, tj. manuální vložení CID nebo Axiom odkazu, a stažení příslušného souboru.

b) Uživatel naviguje na předem vygenerovaný Axiom odkaz pro stažení souboru obsahující cestu `/download` a k tomu doplňující parametry konkrétního souboru. Tento odkaz je následně aplikací automaticky rozpoznán a předvyplněn do příslušného pole. Uživatel může bezprostředně poté zahájit stahování kliknutím na tlačítko „Download“.

7.6.3 Seznam uživatelem nahraných souborů (vault)

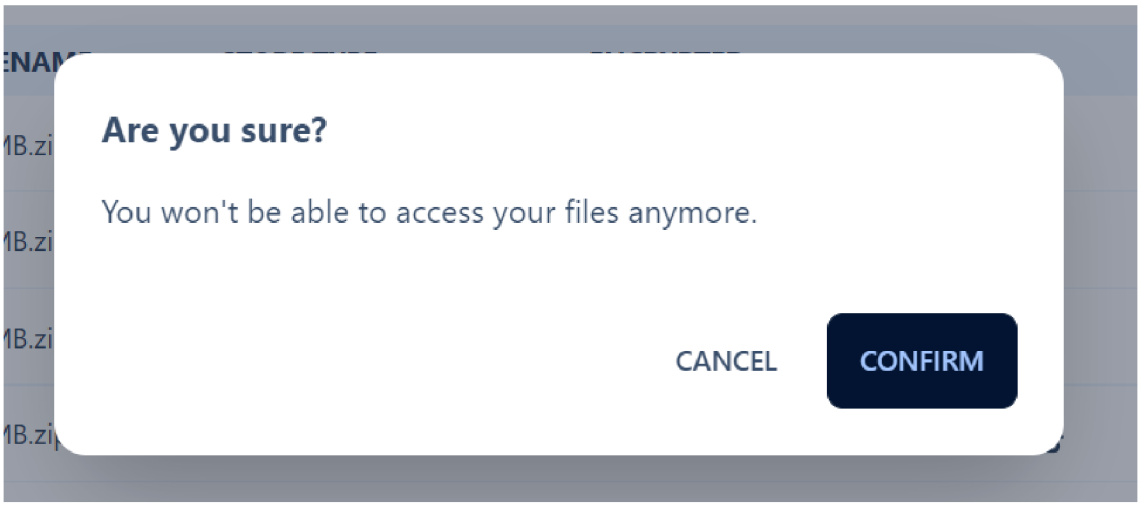
Všechny soubory, které uživatel prostřednictvím aplikace nahraje do IPFS, jsou zaznamenány v úložišti webového prohlížeče. Je tomu tak pro případ, kdyby se uživatel potřeboval vrátit k historicky nahraným souborům, či si jen zapomněl zkopírovat odkaz(y).

Seznam všech těchto souborů si může uživatel prohlédnout právě v tomto modulu. V modulu je vykreslena tabulka zobrazující názvy souborů, typ uložení, zvolené dodatečné možnosti (např. šifrování) a dále tlačítka pro stažení souborů či zkopírování příslušných odkazů.



Obrázek 20 Modul seznamu nahraných souborů (práce autora)

Pokud chce uživatel trezor vymazat, může k tomu použít tlačítko „Clear Vault“, viz předchozí obrázek. Vymazání trezoru však znamená ztrátu veškerých informací nahraných o souborech, včetně dat potřebných pro jejich dešifrování. Z tohoto důvodu se uživateli před vymazáním zobrazí potvrzovací modální okno.



Obrázek 21 Potvrzení vymazání trezoru (práce autora)

7.7 Implementace interních procesů

Veškeré akce v aplikaci doprovází řada komplexních interních procesů. V této kapitole jsou některé tyto procesy rozepsány a asociovány s jednotlivými moduly či prvky uživatelského rozhraní.

7.7.1 Trezor (vault)

Při každém načtení aplikace je provedena kontrola a případná inicializace tzv. trezoru (vault) sloužícího pro ukládání informací o uživatelem nahraných souborech. Trezor je ve webovém prohlížeči postaven okolo nativní *Web Storage API*, konkrétně *localStorage*. Toto úložiště je založeno na principu *key-value*, kdy *value* musí být textový řetězec. Proto je nutné veškerá data trezoru před uložením serializovat, a naopak při získávání deserializovat. To vše je implementováno v rámci vlastní *vault* „mikro" API, která zbytku aplikace zpřístupňuje tyto jednoduché metody:

- *initVault* – tuto funkci je nutné zavolat při spuštění aplikace ještě předtím, než se bude s trezorem jakkoliv manipulovat.
- *getVault* – tato funkce vrací kompletní objekt trezoru, který je získán z *localStorage* a následné deserializován.
- *updateVault* – tato funkce přijímá jakékoliv parametry trezoru, které potřebujeme aktualizovat, a následně je sloučí s daty v *localStorage*.
- *getFiles* – tato funkce vrací veškeré informace o nahraných souborech.
- *addFile* – tato funkce je zavolána po každém úspěšném nahrání souboru do IPFS, jejím účelem je přidat informace o daném souboru do trezoru.
- *clearFiles* – tato funkce slouží k odstranění veškerých informací o nahraných souborech z trezoru, a je provedena po kliknutí na tlačítko „Clear Vault" v modulu seznamu uživatelem nahraných souborů (vault).

7.7.2 Nahrávání souborů do IPFS

Soubory jsou do IPFS nahrávány prostřednictvím jednoduché bezplatné API, kterou poskytuje služba Estuary.tech. Aby bylo možné tuto API využívat, bylo potřeba provozovatele služby Estuary.tech kontaktovat za účelem vytvoření uživatelského účtu a následného vygenerování API klíčů.

7.7.3 Stahování souborů

V modulech, kde je možné stahovat soubory – modul stahování a modul trezoru, má uživatel možnost si vybrat, jakým způsobem budou soubory stahovány. Na výběr jsou možnosti dvě:

- **Přes veřejnou IPFS bránu**: Pokud je zvolena tato možnost, bude soubor z IPFS sítě stažen pomocí veřejné IPFS brány https://cloudflare-ipfs.com/ipfs/. Tento způsob nabízí zpravidla vysokou rychlost stahování a nízkou dobu potřebnou pro nalezení souboru v síti IPFS.
- **Přes interní IPFS uzel:** Pokud je zvolen tento způsob, bude přímo ve webovém prohlížeči spuštěn vlastní IPFS uzel, jež bude následně použit k získání daného souboru. Tento způsob je určen spíše pro testovací či záložní účely, protože má výrazně pomalejší rychlost stahování než veřejná IPFS brána.

7.7.4 Šifrování a dešifrování

Pro šifrování dat a souborů bylo v aplikaci implementováno zjednodušené API postavené na nativní prohlížečové technologii *Web Crypto API* (dále jen jako jmenný prostor „*crypto*“). Samotné šifrování je založeno na algoritmu AES-GCM s 256bitovým klíčem. Proces šifrování je v této vlastní API rozdělen do čtyř fází:

1. Nejprve je pomocí funkce *crypto.getRandomValues* vygenerován pseudonáhodný inicializační vektor.
2. Následuje funkce *crypto.subtle.generateKey*, jež na základě zvoleného šifrovacího algoritmu vygeneruje 256bitový šifrovací klíč.

3. Dalším krokem je samotné šifrování dat. K tomu je použita funkce *crypto.subtle.encrypt*, která s použitím vygenerovaného inicializačního vektoru a šifrovacího klíče zašifruje zvolená data (soubor).

4. Posledním velice důležitým krokem je vyextrahování šifrovacího, respektive dešifrovacího, klíče do podoby, abychom jej mohli zpětně použít pro dešifrování dat (souboru). To obstarává funkce *crypto.subtle.exportKey*.

Kromě zašifrovaných dat výše zmíněný proces API vrátí i inicializační vektor a vyextrahovaný šifrovací, respektive dešifrovací, klíč, který následně uložíme jako dodatečné informace k souboru.

Proces dešifrování dat je velmi podobný, potřebujeme k němu inicializační vektor a dešifrovací klíč. Hlavním rozdílem je, že dešifrovací klíč musíme nejprve importovat pomocí funkce *crypto.subtle.importKey*, a až poté jej použít pro dešifrování dat.

7.7.5 Obalení souboru (Wrapped File)

K obalení souboru při nahrávání způsobem *Wrapped File* je použit dodatečný NPM balíček *JSZip* sloužící k vytváření ZIP souborů přímo v prohlížeči. Proces obalení je jednoduchý a je rozdělen do dvou fází:

1. Nejprve jsou vygenerována metadata zvoleného souboru – název, typ souboru, datum poslední úpravy, časové razítko a informace, zda je obsažený soubor šifrovaný. Tato data jsou následně uložena ve formátu JSON do souboru „*axiom.json*“.

2. Vygenerovaná metadata a zvolený soubor jsou uloženy do souboru ZIP, a ten je následně předán do dalšího procesu – nahrání do IPFS.

7.7.6 Vlastní React hook pro komunikaci s externí API

Pro komunikaci s externí API – zejména nahrávání souboru přes Estuary.tech do IPFS – byl implementován vlastní React hook zvaný *useApi*. Ten umožňuje v Reactu využít *lifecycle* metody a stavový hook *useState* pro jednodušší implementaci reakcí na různé události během komunikace s API – načítání, chyba nebo samotné

získání dat. Tento hook je univerzální, a je možné jej použít pro různé koncové body API.

```
const [upload, { data, loading, error, progress, reset: resetFetch }] =
useApi(
  ApiEndpoint.ContentAdd
);
```

Obrázek 22 Příklad použití hooku *useApi* (práce autora)

Na obrázku výše je možné vidět použití tohoto hooku pro komunikaci s API za účelem nahrání souboru (*ApiEndpoint.ContentAdd*). Z návratových dat hooku je destrukturována řada parametrů. První parametr *upload* je funkce, která slouží pro spuštění procesu nahrávání. Zbývající parametry je pak možné použít pro renderování obsahu, respektive reakce různých komponent na změny hodnot těchto parametrů. Pokud například parametr *loading* bude mít hodnotu *true*, pak můžeme pomocí libovolné komponenty znázornit, že zrovna probíhá načítání (nahrávání souboru).

7.7.7 Pomocné API proxy

Autorizační API klíč potřebný pro zasílání požadavků na *Estuary.tech API* nemůžeme z bezpečnostního hlediska přímo zahrnout do front-endové části webové aplikace. Kdokoliv by totiž mohl tento klíč teoreticky odcizit. Z tohoto důvodu byl s pomocí frameworku Next.js vytvořen vlastní back-end poskytující speciální pomocné API – „proxy“, jež funguje jako most mezi aplikací a *Estuary.tech API*. Každý požadavek zaslaný na pomocné API je rozšířen o zmíněný API klíč, a požadavek je dále přesměrován na *Estuary.tech API*.

7.8 Nasazení hotové aplikace

Nasazení webové aplikace bylo rozděleno do dvou částí. Front-end část aplikace je vyexportována do statické podoby, a skrze platformu Fleek.co je hostována přímo v síti IPFS. Back-end část aplikace (pomocné API proxy) je provozována ve virtuálním serveru na platformě *DigitalOcean*.

Samotný proces nasazení aplikace je velice jednoduchý a zpravidla zcela automatický. Obě zmíněné platformy umožňují napojení vlastního Git repozitáře s aplikací. Po napojení repozitáře (typicky produkční větve „*main*“) je aplikace automaticky zkompilována a nasazena na příslušné platformě. Aplikace je poté dostupná na vygenerované URL adrese, případně na vlastní doméně. Pokud se v repozitáři aplikace v produkční větvi kódu provedou jakékoliv změny, dojde automaticky k opětovné kompilaci a nasazení nové verze aplikace.

8 Shrnutí výsledků

Výsledná webová aplikace úspěšně uplatňuje jeden ze způsobů, jak využít decentralizaci v odvětví sdílení souborů přes internet. Na rozdíl od některých tradičních centralizovaných služeb nabízí tato aplikace silnou ochranu před *body selhání*, neoprávněnými modifikacemi souborů a cenzurou. Decentralizace a distribuce nahraných souborů v IPFS je zajištěna skrze *dohody o skladování* uzavřené v blockchainové síti Filecoin. Zmíněný princip enkapsulace souborů – *Wrapped File* – pak přináší experimentální způsob, jak v síti IPFS sdílet soubory se zachováním jejich názvu a dalších parametrů. Mimo to je možné soubory před nahráním zašifrovat, a tím tak zajistit jejich soukromé sdílení. Samotná front-end část aplikace je dále také hostována přímo v síti IPFS, což zajišťuje její decentralizaci a stabilní dostupnost.

Webová aplikace je veřejně dostupná na adrese https://axiom.dellos.io/.

Vzhledem k tomu, že IPFS je P2P síť, a v současné době není stále dostatečně rozšířena, může odezva a rychlost při nahrávání či stahování souborů mírně kolísat. Dalším důvodem je to, že nahrané soubory jsou v síti IPFS zpravidla unikátní, zvláště pokud jsou šifrované nebo typu *Wrapped File*. To znamená, že bezprostředně po nahrání nemají téměř žádné IPFS uzly informace o těchto souborech, a po nějakou dobu „nevědí“, kde v síti se soubor nachází. Nicméně rychlost nahrávání a stahování souborů byla během testování vysoká a pro účely sdílení testovacích souborů více než dostatečná.

Pro znázornění rychlosti nahrávání a stahování bylo provedeno měření, které můžeme vidět níže v tabulce č. 1. Měření bylo provedeno se soubory typu *Wrapped File*, o velikosti 20 MB (160 Mb) a se zapnutým šifrováním pro zajištění unikátnosti souborů v rámci sítě IPFS. Jako režim stahování byla vybrána veřejná IPFS brána. Rychlost internetového připojení v době měření byla 33 Mbps (rel. odch. ±0 Mbps) v případě stahování a 45 Mbps (rel. odch. ±6 Mbps) v případě nahrávání.

Měření	Nahrávání	Odezva před zahájením stahování	Stahování
1	10,8 s (14,8 Mbps)	20,3 ms	7,4 s (21,6 Mbps)
2	6,6 s (24,2 Mbps)	90,3 ms	5,4 s (29,6 Mbps)
3	12,6 s (12,7 Mbps)	2,3 s	4,7 s (34 Mbps)
4	6,5 s (24,6 Mbps)	1,9 s	5,2 s (30,8 Mbps)
5	7,2 s (22,2 Mbps)	2,2 s	4,9 s (32,7 Mbps)
Průměr	8,7 s (18,3 Mbps)	1,3 s	5,5 s (29,1 Mbps)

Tabulka 1 Měření rychlost nahrávání a stahování souborů v aplikaci (práce autora)

9 Závěry a doporučení

Cílem práce bylo popsat čtenáři současnou architekturu internetu a centralizace služeb, představit alternativní decentralizované metody v podobě IPFS a Filecoin, a dále navrhnout a implementovat webovou aplikaci pro sdílení souborů uplatňující tyto alternativní metody.

Na začátku teoretické části práce byla zprvu vysvětlena stručná historie a architektura internetu. Následně byl definován problém centralizace a s tím spojená rizika. Poté byl podrobně popsán princip decentralizace, P2P sítě, distribuované datové úložiště IPFS a kryptoměna Filecoin. Dále byla představena problematika vývoje webových stránek a aplikací, související technologie a jazyky. Teoretická část práce byla zakončena popisem a rozborem fungování moderní JavaScriptové knihovny React, která je v současnosti velmi využívána pro vývoj moderních webových aplikací.

Účelem praktické části bylo implementovat webovou aplikaci, jež by uplatňovala možnosti decentralizace zmíněné v teoretické části pro sdílení souborů v internetu. Praktická část byla rozdělena do dvou sekcí. V první sekci proběhl souhrn základních požadavků na webovou aplikaci, návrh funkcí webové aplikace a výběr zvažovaných technologií. V druhé sekci byla popsána samotná implementace webové aplikace, její uživatelské rozhraní a interní procesy.

Zatímco soubory úspěšně nahrané do IPFS prostřednictvím této aplikace jsou v pravém slova smyslu decentralizované a distribuované, a tím i jejich stahování, aplikace samotná z obecného hlediska plně decentralizovaná není. Decentralizaci je v současnosti totiž možné v tomto kontextu a produkčním prostředí aplikovat pouze na front-end část aplikace, nikoli však na její back-end (API). Z tohoto důvodu aplikace obsahuje dva rizikové body, které se však vztahují pouze na proces nahrávání souborů. Těmito body jsou:

- Vzdálená Estuary.tech API.
- Vlastní virtuální server, na kterém je provozována pomocná API proxy.

Oba tyto body jsou však prostorem pro případný další rozvoj aplikace v kontextu hlubší decentralizace. Prvním krokem může být nahrazení služby Estuary.tech několika vlastními servery, na kterých by byly provozovány vlastní uzly *Estuary*. Druhým krokem, avšak více vzdáleným, může být nasazení celé aplikace, včetně back-endu a uzlů *Estuary*, v rámci některé budoucí decentralizované platformy, která by tyto procesy podporovala.

10 Seznam použité literatury

ARANCIO, Stephen, 2021. What is JSX? *Medium* [online]. [vid. 2022-07-22]. Dostupné z: https://medium.com/@sjarancio/what-is-jsx-e3dda0af3490

BAID, Ishika, Aryan RAWAT a Mr Jagrit KATHURIA, 2022. MODERN SOCIAL MEDIA WEBAPP (SOCIO). *Open Access* [online]. **04**(05), International Research Journal of Modernization in Engineering Technology and Science, 6. ISSN 2582-5208. Dostupné z: https://www.irjmets.com/uploadedfiles/paper/issue_5_may_2022/25252/final/fin_irjmets1654696352.pdf

BARANOWSKI, Wojciech, 2022. *React Lifecycle Methods* [online]. [vid. 2022-07-22]. Dostupné z: https://massivepixel.io/blog/react-lifecycle-methods/

BENET, Juan, 2019. IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3) [online]. 11 [vid. 2021-08-04]. Dostupné z: https://github.com/ipfs/papers/blob/f5fe0d880214a098851fb50983d7720ab5ff4213/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

BIERMAN, Gavin, Martín ABADI a Mads TORGERSEN, 2014. Understanding TypeScript. In: Richard JONES, ed. *ECOOP 2014 – Object-Oriented Programming* [online]. Berlin, Heidelberg: Springer, s. 257–281. Lecture Notes in Computer Science. ISBN 978-3-662-44202-9. Dostupné z: doi:10.1007/978-3-662-44202-9_11

DANTHASINGHE, Wathsala, 2020. Introduction to React Hooks — Basic Hooks. *Medium* [online] [vid. 2022-07-28]. Dostupné z: https://blog.devgenius.io/introduction-to-react-hooks-e49738432f54

DOAN, Trinh Viet, Yiannis PSARAS, Jörg OTT a Vaibhav BAJPAI, 2022. *Towards Decentralised Cloud Storage with IPFS: Opportunities, Challenges, and Future Directions* [online]. 2. duben 2022. B.m.: arXiv. [vid. 2022-07-17]. Dostupné z: http://arxiv.org/abs/2202.06315

HALPIN, Harry, 2014. The W3C web cryptography API: motivation and overview. In: *Proceedings of the 23rd International Conference on World Wide Web* [online]. New York, NY, USA: Association for Computing Machinery, s. 959–964 [vid. 2022-07-27]. WWW '14 Companion. ISBN 978-1-4503-2745-9. Dostupné z: doi:10.1145/2567948.2579224

HAYWARD, Decrypt / Andrew, 2021. How to Use IPFS: The Backbone of Web3. *Decrypt* [online] [vid. 2022-07-17]. Dostupné z: https://decrypt.co/resources/how-to-use-ipfs-the-backbone-of-web3

JONGMANS, Eline, Florence JEANNOT, Lan LIANG a Maud DAMPÉRAT, 2022. Impact of website visual design on user experience and website evaluation: the sequential mediating roles of usability and pleasure. *Journal of Marketing Management* [online]. **0**(0), 1–36 [vid. 2022-07-10]. ISSN 0267-257X. Dostupné z: doi:10.1080/0267257X.2022.2085315

KLIMM, Marvin Christian, 2021. *Design Systems for Micro Frontends - An Investigation into the Development of Framework-Agnostic Design Systems using Svelte and Tailwind CSS* [online]. B.m. [vid. 2022-07-27]. Hochschulbibliothek der Technischen Hochschule Köln. Dostupné z: https://epb.bibl.th-koeln.de/frontdoor/index/index/docId/1666

KŘÍŽ, Pavel, 2019. React Hooks, které potřebujete znát. *Zdroják* [online]. [vid. 2022-07-22]. Dostupné z: https://zdrojak.cz/clanky/react-hooks-ktere-potrebujete-znat/

LUOJUS, Tuomas, 2021. *Usability and Adaptation of React Hooks* [online]. B.m. [vid. 2022-07-28]. Tampere University. Dostupné z: https://trepo.tuni.fi/handle/10024/130986

MACHALA, Miroslav, 2007. *Historie internetu a jeho budoucí využití* [online]. B.m. [vid. 2022-07-07]. Masarykova univerzita, Pedagogická fakulta. Dostupné z: https://is.muni.cz/th/135874/pedf_b/Historie_lnternetu_aJeho_budouci_vyuziti.pdf

MINIWATTS MARKETING GROUP, 2022. *World Internet Users Statistics and 2022 World Population Stats* [online] [vid. 2022-07-10]. Dostupné z: https://www.internetworldstats.com/stats.htm

NABBEN, Kelsie, 2022. *Decentralized Technology in Practice: An analysis of socio-technical resilience in IPFS* [online]. SSRN Scholarly Paper. 31. březen 2022. [vid. 2022-07-17]. Dostupné z: doi:10.2139/ssrn.4082016

NAST, Condé, 2014. Tim Berners-Lee on the Web at 25: the past, present and future. *Wired UK* [online]. [vid. 2022-08-14]. ISSN 1357-0978. Dostupné z: https://www.wired.co.uk/article/tim-berners-lee

PERSSON, Morgan, 2020. *JavaScript DOM Manipulation Performance : Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks* [online] [vid. 2022-07-31]. Dostupné z: http://urn.kb.se/resolve?urn=urn:nbn:se:bth-19531

POLITOU, Eugenia, Efthimios ALEPIS, Constantinos PATSAKIS, Fran CASINO a Mamoun ALAZAB, 2020. Delegated content erasure in IPFS. *Future Generation Computer Systems* [online]. **112**, 956–964 [vid. 2022-08-15]. ISSN 0167-739X. Dostupné z: doi:10.1016/j.future.2020.06.037

PROTOCOL LABS, [b.r.]. *Estuary Documentation* [online] [vid. 2022-08-15]. Dostupné z: https://docs.estuary.tech

PROTOCOL LABS, [b.r.]. *IPFS Documentation | IPFS Docs* [online] [vid. 2022-07-17]. Dostupné z: https://docs.ipfs.io/

RAMAN, Aravindh, Sagar JOGLEKAR, Emiliano De CRISTOFARO, Nishanth SASTRY a Gareth TYSON, 2019. Challenges in the Decentralised Web: The Mastodon Case. In: *Proceedings of the Internet Measurement Conference* [online]. New York, NY, USA: Association for Computing Machinery, s. 217–229 [vid. 2022-07-15]. IMC '19. ISBN 978-1-4503-6948-0. Dostupné z: doi:10.1145/3355369.3355572

RICHARDS, Gregor, Francesco Zappa NARDELLI a Jan VITEK, 2015. Concrete Types for TypeScript. In: John Tang BOYLAND, ed. *29th European Conference on Object-Oriented Programming (ECOOP 2015)* [online]. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, s. 76–100 [vid. 2022-08-15]. Leibniz International Proceedings in Informatics (LIPIcs). ISBN 978-3-939897-86-6. Dostupné z: doi:10.4230/LIPIcs.ECOOP.2015.76

ROCHA, Alfonso, David DIAS a Yiannis PSARAS, 2021. Accelerating Content Routing With Bitswap: A Multi-Path File Transfer Protocol in IPFS and Filecoin. *Protocol Labs Research* [online] [vid. 2022-07-19]. Dostupné z: https://research.protocol.ai/publications/accelerating-content-routing-with-bitswap-a-multi-path-file-transfer-protocol-in-ipfs-and-filecoin/

RONEN, Ran, 2022. Council Post: Five Tips For Improving Web Accessibility And SEO. *Forbes* [online] [vid. 2022-07-10]. Dostupné z: https://www.forbes.com/sites/forbesbusinesscouncil/2022/06/30/five-tips-for-improving-web-accessibility-and-seo/

SASIKUMAR, S., S. PRABHA a Chandra MOHAN, 2022. *Improving Performance Of Next.Js App And Testing It While Building A Badminton Based Web App* [online]. SSRN Scholarly Paper. 27. květen 2022. [vid. 2022-07-27]. Dostupné z: doi:10.2139/ssrn.4121058

SHARMA, Pratima, Rajni JINDAL a Malaya Dutta BORAH, 2020. Blockchain Technology for Cloud Storage: A Systematic Literature Review. *ACM Computing Surveys* [online]. **53**(4), 1–32 [vid. 2021-10-15]. ISSN 0360-0300, 1557-7341. Dostupné z: doi:10.1145/3403954

SOLOVEI, V., O. OLSHEVSKA a Y. BORTSOVA, 2018. THE DIFFERENCE BETWEEN DEVELOPING SINGLE PAGE APPLICATION AND TRADITIONAL WEB APPLICATION BASED ON MECHATRONICS ROBOT LABORATORY ONAFT APPLICATION. *Automation of technological and business processes* [online]. **10**(1) [vid. 2022-08-07]. ISSN 2312-931X. Dostupné z: doi:10.15673/atbp.v10i1.874

VOXCAFE S.R.O., 2021. Lekce 1. - React - úvod. *MIND BLOG* [online] [vid. 2022-07-12]. Dostupné z: https://www.voxcafe.cz/index.php?article=/mindblog/clanky/programovani/lekce1.html

WANG, Shangping, Yinglong ZHANG a Yaling ZHANG, 2018. A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems. *IEEE Access* [online]. **6**, 38437–38450. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2018.2851611

WOLF, Karel, 2022. Meziplanetární P2P souborový systém zamíří skutečně do vesmíru. *Lupa.cz* [online] [vid. 2022-07-19]. Dostupné z: https://www.lupa.cz/clanky/meziplanetarni-souborovy-system-zamiri-skutecne-do-vesmiru/

ZAHED BENISI, Nazanin, Mehdi AMINIAN a Bahman JAVADI, 2020. Blockchain-based decentralized storage networks: A survey. *Journal of Network and Computer Applications* [online]. **162**, 102656 [vid. 2021-10-11]. ISSN 1084-8045. Dostupné z: doi:10.1016/j.jnca.2020.102656

ZARRIN, Javad, Hao WEN PHANG, Lakshmi BABU SAHEER a Bahram ZARRIN, 2021. Blockchain for decentralization of internet: prospects, trends, and challenges. *Cluster Computing* [online]. **24**(4), 2841–2866 [vid. 2022-07-15]. ISSN 1386-7857, 1573-7543. Dostupné z: doi:10.1007/s10586-021-03301-8

Zadání bakalářské práce

Autor:	**Petr Schmidt**
Studium:	I1900250
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Vývoj webové aplikace pro decentralizované sdílení souborů**
Název bakalářské práce AJ:	Building a Web-based Application for Decentralized File-sharing

Cíl, metody, literatura, předpoklady:

Cíl práce: Prozkoumat principy decentralizace obsahu v počítačových sítích a implementovat webovou aplikaci, která tyto principy uplatňuje

1. Prozkoumat principy decentralizace obsahu v počítačových sítích
2. Popsat vybrané decentralizované systémy
3. Prozkoumat webové technologie a frameworky
4. Implementovat webovou aplikaci pro decentralizované sdílení souborů za použití vybraných webových technologií a decentralizovaného systému
5. Zhodnotit dosažené výsledky

SHARMA, Pratima, Rajni JINDAL a Malaya Dutta BORAH, 2020. Blockchain Technology for Cloud Storage: A Systematic Literature Review. ACM Computing Surveys [online]. 53(4), 1–32 [vid. 2021-10-15]. ISSN 0360-0300, 1557-7341. Dostupné z: doi:10.1145/3403954

ZAHED BENISI, Nazanin, Mehdi AMINIAN a Bahman JAVADI, 2020. Blockchain-based decentralized storage networks: A survey. Journal of Network and Computer Applications [online]. 162, 102656 [vid. 2021-10-11]. ISSN 1084-8045. Dostupné z: doi:10.1016/j.jnca.2020.102656

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Milan Košťák
Oponent:	Ing. Pavel Kříž, Ph.D.

Datum zadání závěrečné práce: 15.10.2021