Mobilní mailový klient s možnostmi pokročilé archivace

Bc. Erik Faltynek

Diplomová 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 DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Erik Faltynek**
Osobní číslo:	**A22397**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Mobilní mailový klient s možnostmi pokročilé archivace**
Téma práce anglicky:	**Mobile Mail Client with Advanced Archiving Options**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Vyberte a popište použité technologie.
3. Navrhněte aplikaci včetně všech potřebných požadavků.
4. Implementujte mobilní aplikaci s možností pokročilé archivace.
5. Aplikaci vhodně otestujte a prezentujte výsledky.

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

Seznam doporučené literatury:

1. *React Native Learn once, write anywhere*. Online. Reactnative. 2023. Dostupné z: https://reactnative.dev. [cit. 2023--11-11].
2. *Express 4.18.1 Fast, unopinionated, minimalist web framework for Node.js*. Online. Expressjs. 2017. Dostupné z: https://expressjs.com. [cit. 2023-11-11].
3. *MongoDB Documentation*. Online. Mongodb. 2023. Dostupné z: https://www.mongodb.com/docs/. [cit. 2023-11--11].
4. MOSTEFAOUI, Ghita K. a TARIQ, Faisal (ed.). *Mobile Apps Engineering: Design, Development, Security, and Testing*. Chapman and Hall/CRC, 2018. ISBN 978-0367656898.
5. BOYD, Ryan. *Getting Started with OAuth 2.0: Programming Clients for Secure Web API Authorization and Authentication*. 1. O'Reilly, 2012. ISBN 1449311601.
6. LOSHIN, Pete. *Essential Email Standards: RFCs and Protocols Made Practical*. Har/Cdr. Wiley, 1999. ISBN 9780471345978.

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

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové 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 diplomové 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 diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou 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 – diplomovou 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 diplomové 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 diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové 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 diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 10.05.2024

Erik Faltynek, v.r.
podpis studenta

ABSTRAKT

Diplomová práce se zabývá vývojem mobilního mailového klienta s pokročilými možnostmi archivace. Hlavním výstupem práce je aplikace, která kombinuje standardní funkce mailového klienta s rozšířenými možnostmi pro archivaci emailů na základě specifických filtračních kritérií. Uživatelé mají možnost ukládat archivy jak v nezašifrované, tak v zašifrované formě přímo na zařízení nebo zašifrovaně v databázi s definovatelným datem expirace. Teoretická část práce se věnuje základním principům vývoje mobilních aplikací, klíčovým emailovým protokolům, a metodám autentizace a autorizace. Jsou rozebrány technologie šifrování, včetně symetrické šifry AES a bezpečných KDF funkcí pro odvození klíčů a uchování hesel. Rovněž je popsána integritní a autentizační metoda MAC pro ověření správnosti hesel. Praktická část detailně popisuje návrh a implementaci aplikace, počínaje definicí funkcionálních a nefunkcionálních požadavků, přes návrh informační architektury a uživatelského rozhraní, až po definici datového modelu a komunikačních rozhraní. Závěrečné kapitoly se věnují nasazení serverové části aplikace na cloudovou platformu a sestavení aplikace pro Android, doplněné o popis testovacího procesu během vývoje.

Klíčová slova: vývoj mobilních aplikací, React Native, Expo, Node, Express, MongoDB, OAuth 2.0, mailový klient

ABSTRACT

The thesis deals with the development of a mobile mail client with advanced archiving capabilities. The main output of the thesis is an application that combines standard email client functions with advanced options for archiving emails based on specific filtering criteria. Users have the option to store archives in both unencrypted and encrypted form directly on the device or encrypted in a database with a definable expiration date. The theoretical part of the thesis deals with the basic principles of mobile application development, key email protocols, and authentication and authorization methods. Encryption technologies are discussed, including the AES symmetric cipher and secure KDF functions for key derivation and password preservation. MAC integrity and authentication methods for verifying the

correctness of passwords are also described. The practical part describes in detail the design and implementation of the application, starting from the definition of functional and non-functional requirements, to the design of the information architecture and user interface, to the definition of the data model and communication interfaces. The final chapters deal with the deployment of the server side of the application on the cloud platform and the build of the Android application, complemented by a description of the testing process during development.

Keywords: mobile application development, React Native, Expo, Node, Express, MongoDB, OAuth 2.0, mail client

Tímto bych chtěl poděkovat svému vedoucímu práce Ing. Petru Žáčkovi, Ph.D. za cenné rady a konzultace ohledně diplomové práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **12**
I TEORETICKÁ ČÁST **14**
1 MOBILNÍ APLIKACE **15**
1.1 NATIVNÍ APLIKACE 15
1.2 MULTIPLATFORMNÍ APLIKACE 15
1.3 HYBRIDNÍ APLIKACE 16
1.4 PROGRESIVNÍ WEBOVÉ APLIKACE 17
1.5 MOBILNÍ MAILOVÉ APLIKACE 18
1.5.1 Některé mobilní mailové aplikace 19
1.5.1.1 Gmail 19
1.5.1.2 Outlook 20
2 EMAIL STANDARDY **21**
2.1 INTERNET MESSAGE FORMAT 21
2.2 SMTP 22
2.2.1 Simple Mail Transfer Protocol Secure 22
2.3 POST OFFICE PROTOCOL 23
2.4 INTERNET MESSAGE ACCESS PROTOCOL 24
3 AUTENTIZACE **25**
3.1 SESSION-BASED AUTHENTICATION 26
3.2 TOKEN-BASED AUTHENTICATION 26
3.3 OPENID CONNECT 28
4 AUTORIZACE **29**
4.1 OAUTH 2.0 29
4.1.1 Role 29
4.1.2 Interakce mezi rolemi 30
4.1.3 Authorization grant 31
4.1.3.1 Authorization Code grant 31
4.1.3.2 Proof Key for Code Exchange 33
4.1.3.3 Implicit grant 34
4.1.3.4 Resource Owner's Password Credentials grant 35
4.1.3.5 Client Credentials Grant 36
4.1.3.6 Device grant 37
4.1.3.7 Refresh token grant 38
5 UCHOVÁVÁNÍ TOKENŮ **40**
5.1 CLIENT-SIDE STORAGE 40
5.2 SERVER-SIDE STORAGE 40
5.3 UCHOVÁNÍ ID TOKENU 41
6 MOŽNOSTI POKROČILÉ ARCHIVACE EMAILŮ **42**
6.1 ADVANCED ENCRYPTION STANDARD 43
6.2 MAC & KDF 43
7 NERELAČNÍ DATABAZE **44**
8 POPIS VYBRANÝCH TECHNOLOGIÍ **45**

8.1 JavaScript a TypeScript 45
8.2 Klientská část 46
8.2.1 React Native 46
8.2.2 Expo 46
8.2.3 Použité knihovny na straně klienta 47
8.3 Serverová část 50
8.3.1 Node.js 50
8.3.2 Express.js 50
8.3.3 MongoDB 51
8.3.4 Použité knihovny na straně serveru 51
II PRAKTICKÁ ČÁST 55
9 SBĚR POŽADAVKU 56
9.1 Funkcionální požadavky 56
9.1.1 Funkcionální požadavky pro autentizaci pomocí tokenů 56
9.1.2 Funkcionální požadavky pro integraci emailových účtů 57
9.1.3 Funkcionální požadavky pro odesílání emailových zpráv 57
9.1.4 Funkcionální požadavky pro přijímání a zobrazování emailových zpráv 57
9.1.5 Funkcionální požadavky pro filtraci emailových zpráv 57
9.1.6 Funkcionální požadavky pro spravování emailových zpráv 58
9.1.7 Funkcionální požadavky pro pokročilou archivaci 58
9.1.8 Funkcionální požadavky pro obnovení archivovaných emailových zpráv 58
9.1.9 Funkcionální požadavky pro uživatelskou sekci 58
9.2 Nefunkcionální požadavky 59
10 NÁVRH APLIKACE 60
10.1 Informační architektura 60
10.2 Drátěný model aplikace 62
10.2.1 Přihlašovací obrazovka 62
10.2.2 Registrační obrazovka 63
10.2.3 Obrazovka pro zapomenuté heslo 64
10.2.4 Obrazovka pro resetování hesla 65
10.2.5 Obrazovka bez připojeného emailového účtu 66
10.2.6 Uživatelské menu 67
10.2.7 Obrazovka mailboxu 68
10.2.8 Obrazovka pro přečtení emailové zprávy 69
10.2.9 Obrazovka pro odeslání emailové zprávy 70
10.2.10 Filtrace 71
10.2.11 Panel nástrojů 72
10.2.12 Obrazovka pokročilé archivace 73
10.2.13 Obrazovka pro obnovu archívu 74
10.2.14 Obrazovka uživatelského nastavení 75

10.3 NÁVRH DATOVÉHO MODELU 76
10.4 KOMUNIKAČNÍ ČÁST 77
10.5 OPTIMALIZACE DATOVÉHO PROVOZU 77
10.6 DEFINICE ROZHRANÍ 78
10.6.1 Rozhraní pro autentizaci 78
10.6.2 Rozhraní pro správu hesel 79
10.6.3 Rozhraní pro protokol OAuth 2.0 80
10.6.4 Rozhraní pro SMTP protokol 81
10.6.5 Rozhraní pro protokol IMAP 81
10.6.6 Rozhraní pro archivaci 84
10.6.7 Rozhraní pro uživatelské nastavení 85
10.7 SEKVENČNÍ DIAGRAMY DŮLEŽITÝCH ČÁSTÍ APLIKACE 86
10.7.1 Proces přihlášení uživatele 86
10.7.2 Proces autentizace uživatele 87
10.7.3 Proces připojení emailového účtu 89
10.7.4 Proces obnovy přístupového tokenu emailového poskytovatele 93
10.7.5 Proces lokální archivace 94
10.7.6 Proces serverové archivace 96
10.7.7 Proces obnovy lokální archivace 97
10.7.8 Proces obnovy serverové archivace 98
11 VÝSLEDNÁ APLIKACE 101
11.1 ADRESÁŘOVÁ STRUKTURA APLIKACE 101
11.1.1 Struktura klientské části 101
11.1.2 Struktura serverové části 103
11.2 POPIS VÝSLEDNÉHO DATOVÉHO MODELU 105
11.3 IMPLEMENTACE PŘIHLÁŠENÍ A REGISTRACE 106
11.4 IMPLEMENTACE RESETU HESLA 107
11.5 IMPLEMENTACE AUTENTIZACE 109
11.6 IMPLEMENTACE OAUTH 2.0 110
11.7 IMPLEMENTACE IMAP PROTOKOLU 114
11.8 IMPLEMENTACE OBRAZOVKY MAILOVÉ SCHRÁNKY 115
11.9 IMPLEMENTACE FILTRACE 117
11.10 IMPLEMENTACE SPRÁVY EMAILOVÝCH ZPRÁV 118
11.11 IMPLEMENTACE POKROČILÉ ARCHIVACE 120
11.11.1 Klientská pokročilá archivace 121
11.11.2 Serverová pokročilá archivace 122
11.12 IMPLEMENTACE PROCESU ŠIFROVÁNÍ DAT 123
11.13 IMPLEMENTACE OBNOVY ARCHÍVŮ 123
11.13.1 Obnova lokální archivace 124
11.13.2 Obnova serverové archivace 125

11.14 Implementace procesu dešifrování dat 126
11.15 Implementace čtení emailové zprávy 127
11.16 Implementace odeslání emailové zprávy 129
11.17 Implementace uživatelského nastavení 130
11.18 Bezpečnost 132
12 NASAZENÍ APLIKACE 134
12.1 Nasazení serverové části 134
12.2 Sestavení mobilních aplikací 135
13 TESTOVÁNÍ 137
ZÁVĚR 139
SEZNAM POUŽITÉ LITERATURY 140
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 147
SEZNAM OBRÁZKŮ 150
SEZNAM PŘÍLOH 153

ÚVOD

Populární mailové aplikace disponují klasickou archivací, ale neumožňuji tyto archivace zašifrovat a uložit do mobilního zařízení. Proto cílem této diplomové práce je implementovat mobilního mailového klienta s možnostmi pokročilé archivace. Aplikace umožní tyto archívy bezpečně zašifrovat a následně obnovit s veškerými původními informacemi.

Práce se prvně v teoretické části zaměří na druhy vývoje mobilních aplikací s představením dvou nejpopulárnějších aplikací Google a Outlook. Další kapitola bude věnovaná emailovým protokolům, které jsou určené pro odesílání, příjímání a správu emailových dat. Následující kapitola se zabývá možnostmi autentizace uživatele, mezi které patří autentizace pomocí tokenů, session a OIDC. Následně je další kapitola věnovaná autorizačnímu protokolu OAuth 2.0, který je v dnešní době nezbytný pro bezpečné připojení emailových účtu k emailovému klientu. Následující kapitola je věnovaná způsobům bezpečného uchování tokenů získaných při procesu autorizace, kde se popisuje uchování na straně klienta a na straně serveru. Další kapitola popisuje kryptografický algoritmus AES pro možnosti pokročilé archivace, včetně algoritmů KDF pro odvození bezpečného privátního klíče z hesla. Součásti popisu KDF je i zmíněno jejich použitelnost v kontextu bezpečného uchování hesel v databázi a doporučení pro použití Argon2 a Scrypt. Pro zajištěni integrity a autentičnosti dat a také pro ověření správnost zadaného hesla pro účely dešifrování je v této části zmíněn MAC. Předposlední sekce je věnována představením nerelační databází s jejich rozdělením do čtyř typů. Poslední kapitola teoretické části se zabývá popisem technologií, jež byly využité při vypracování výstupu práce v podobě aplikace. Nejdříve bude popsán programovací jazyk, použitý pro obě části aplikace. Dále se popis rozdělí do dvou sekci, a to klientské a serverové části.

Praktická část práce bude zaměřená na návrh a prezentaci výsledné aplikace. První kapitola bude věnována návrhem funkcionálních a nefunkcionálních požadavků. V druhé kapitole pak následuje již samotný návrh aplikace. Zpočátku kapitoly bude navržena informační architektura, na kterou navazuje část s návrhem drátěných modelů obrazovek. Následně bude navržen datový model pro nerelační databázi. Další částí se budou věnovat návrhem komunikační části, optimalizaci datového provozu a definici rozhraní. Poslední části této kapitoly pak bude návrh sekvenčních diagramů pro důležité části aplikace. Třetí kapitola se bude věnovat prezentování výsledků s popisem funkcionality. Předposlední kapitola pak bude

popisovat nasazení serverové částí a sestavení mobilní aplikace. Poslední kapitola praktické práce bude věnovaná popisem způsobu testování aplikace.

I. TEORETICKÁ ČÁST

1 MOBILNÍ APLIKACE

V této kapitole bude představeno několik přístupu vývoje mobilních aplikací. Výběr jednoho z těchto typů je klíčovým aspektem v samotném vývoji, jenž se odvíjí od specifických požadavků a cílů projektu. Každý přístup představuje své výhody a nevýhody, které budou v této kapitole rozebrány.

Dále budou představeny požadavky na mobilní mailové klienty, včetně dvou nejpopulárnějších aplikací od poskytovatelů Google a Outlook. Budou popsaný silné stránky a klíčové funkce, které je řadí mezi nejlepší poskytovatele.

1.1 Nativní aplikace

Nativní vývoj aplikací zahrnuje tvorbu softwaru, který je specificky pro dané platformy. Tato tvorba zahrnuje programovací jazyky a frameworky, které poskytuje vlastník platformy. V současnosti jsou dvě dominantní platformy, a to iOS od společnosti Apple Inc. a Android od společnosti Google. Přestože mezi těmito platformami existují určité podobnosti, vývoj pro každou z nich vyžaduje použití odlišných SDKs (Software development kits). [1]

Pro vývoj aplikací určených pro platformu iOS se využívá vývojové prostředí XCode. Vývojáři mají možnost použít programovací jazyk Objective-C, anebo modernější alternativu Swift. Pro platformu Android je určené vývojové prostředí Android Studio s programovacím jazykem Java nebo Kotlin.

Mezi hlavní výhody nativních aplikací patří: vysoký výkon aplikací a přímý přístup k API (application programming interface) zařízení. Na druhou stranu, pokud je cílem vyvinout aplikaci dostupnou pro obě platformy, může se stát nativní vývoj nevýhodou. Vytváření a udržování dvou odlišných aplikací znamená nutnost spravovat dva různé kódy a angažovat dva odlišné vývojové týmy. To vede k vyšším finančním nákladům při samotném vývoji a následném udržování aplikace. [1]

1.2 Multiplatformní aplikace

Multiplatformní aplikace mohou být napsané v různých programovacích jazycích a frameworcích, kde jsou následně zkompilovány do nativního kódu pro iOS a Android. Díky této vlastnosti odpadá nevýhoda nativní aplikace, je zapotřebí pouze jeden tým, který se angažuje

ve vývoji a následné správě jednoho kódu. [1] Další výhodou je konzistentní UX a UI, čehož by bylo náročné dosáhnout pří vývoji dvou separátních aplikací [2].

Mezi nevýhody, oproti nativním aplikacím, patří limitace ohledně výkonu. A to zejména v aplikacích, které mají vysoké nároky na grafiku a zdroje mobilního zařízení. Frameworky pro toto odvětví vývoje sice nabízí širokou škálu přístupu k API zařízení, jako je kamera, GPS a notifikace. Ale v případě vydání nové funkce, mohou nastat limitace v jejím využití anebo dokonce zpoždění její podpory v daném frameworku. [2]

Při vývoji multiplatformních aplikací lze využít řadu frameworků, mezi které patří React Native, Flutter a Xamarin. Ačkoliv je pro vývoj možné použít libovolné vývojové prostředí, k přístupu k nativním API dané platformy je nezbytné aplikaci sestavit přímo na této platformě. K tomuto účelu je vyžadováno použití XCode pro sestavení aplikací pro iOS. Pro platformu Android je potřeba Android Studio anebo samostatný Android SDK.

1.3 Hybridní aplikace

Pro vývoj hybridních mobilních aplikací se používají webové technologie, jako jsou JavaScript, CSS a HTML. Tyto aplikace jsou následně zabalené do webového kontejneru, kterým je na platformě iOS *WKWebView* a na platformě Android *WebView*. Díky tomuto kontejneru je možno zobrazit na nativním zařízení aplikaci napsanou ve webových technologiích. Webový kontejner má však své limitace, nemůže přistupovat k nativním funkcím zařízení. Pro překročení limitací webového kontejneru, je potřeba zabalit webový kontejner do tzn. *Native application wrapper*. Na trhu existují dvě hlavní řešení tohoto typu: Apache Cordova a Ionic's Capacitor. Tyto dvě řešení nabízí pluginy, díky kterým lze přistupovat k nativnímu API zařízení např.: u iOS platformy TouchID pro přihlášení anebo využití technologie Bluetooth. Výslednou aplikaci lze publikovat v obchodech jednotlivých platforem, jako plnohodnotnou aplikaci. [3]

Hlavní výhodou hybridního přístupu je možnost využití webových technologií na vývoj. Díky tomu mohou vývojáři, se znalostmi těchto technologií, rychle a efektivně vyvinout mobilní aplikaci. [1] Tento přístup sdílí ostatní výhody s vývojem multiplatformních aplikací. Vývoj jednotného kódu pro platformy iOS a Android, což vede k nižším nákladům na vývoj a následnou údržbu. Tento přístup také vede k jednotnému uživatelskému zážitku a uživatelskému rozhraní.

Mezi nevýhody multiplatformního vývoje aplikací patří především nižší výkon v porovnání s aplikacemi vyvinutými nativním přístupem [1]. Kromě toho existuje také potřeba designovat vlastní uživatelské rozhraní, to však lze řešit využitím frameworku jako je Ionic. Framework Ionic poskytuje sadu předdefinovaných UI komponent usnadňující tento proces. [3]

1.4 Progresivní webové aplikace

Progresivní webové aplikace (PWA) představuje typ webové aplikace, který lze přidat na domovskou obrazovku zařízení, bez nutností jejich instalace z obchodu aplikací. Tyto aplikace fungují bez problému na různých zařízeních a platformách, ať už se jedná o tablety, osobní počítače nebo mobilní telefony. Jedná se o značnou výhodu ve srovnání s nativními aplikacemi, které jsou navrženy pro konkrétní platformu a vyžadují instalaci přímo z obchodu s aplikacemi. [4]

PWA nabízí řadu výhod, včetně rychlého vývoje pomocí webových technologií s následnou jednoduchou údržbou. Díky využití technologie *service worker* nabízí PWA rychlé načtení a plynulý uživatelský zážitek, včetně použití aplikace v offline režimu. Ve srovnání s nativními aplikacemi, PWA zabírají méně místa na zařízení a eliminují potřebu stahování aktualizací z obchodu s aplikacemi, jelikož aktualizace probíhají automaticky během přístupu k webové verzi aplikace. [4]

Jelikož jsou PWA založeny na webových technologiích, patří mezi jejich nevýhody omezený přístup k nativnímu API zařízení. Navíc představují vyšší bezpečnostní riziko, protože jsou náchylnější ke kybernetickým útokům, jenž si vyžaduje bezpečnostní opatření. U PWA se také snižuje možnost objevení potenciálními zákazníky, jelikož nejsou dostupné přímo v obchodech s aplikacemi. [4] Tento problém však lze řešit pomocí nástroje jako je *PWA builder*, který umožňuje z PWA vytvořit *package*. Tento *package* je pak možné publikovat v obchodech s aplikacemi. Mimo jiné tento nástroj zlehčuje celý proces vytváření a publikování aplikace. V případě nativní aplikace by se jednalo o vývoj v nativním kódu, jeho testovaní, konfigurování nastavení a vytváření ikon napříč různými zařízení a platformami. [5]

1.5 Mobilní mailové aplikace

Nejpopulárnější formou komunikace ve světě byznysu je bezpochyby email. V dnešní době tento způsob komunikace využívá až 4 miliardy uživatelů denně. Očekává se, že tohle číslo vzroste k roku 2026 až na 4.73 miliard uživatelů denně. Ne každý uživatel má tu možnost sedět u počítače celý den a vyřizovat elektronickou poštu. Pro tento problém je tu řešení v podobě emailových aplikací do chytrých zařízení. [6]

Před zmíněním klíčových vlastností emailových aplikací je důležité, aby byly představeny dva emailové subsystémy:

- **Poskytovatel *(provider)*:** Jedná se o server, který spravuje odesílání a přijímaní emailů. Například: Gmail, Outlook a Yahoo. [7]
- **Klient *(client)*:** Je to aplikace, se kterou uživatelé interagují. Například: Apple Mail a Thunderbird. [7]

Uživatelům je umožněno kombinovat různé emailové klienty s různými poskytovateli. Například může být použita aplikace Apple Mail a k ní mohou být připojeni poskytovatelé jako Outlook a Gmail. [7] Tato kombinace emailového klienta a poskytovatele tvoří celkovou strukturu emailové aplikace.

Emailové aplikace by měly disponovat následujícími klíčovými vlastnostmi:

- **Možnost spravovat více emailových účtů:** Aplikace by měla uživatelům umožnit připojení k známým a rozsáhlým emailovým poskytovatelům, jako jsou Gmail, Outlook a Yahoo. Tímto připojením se emailové účty sjednocují do jedné aplikace, což vede k lepší organizaci a efektivnější práci s emaily.
- **Synchronizace mezi zařízeními*:*** Tomu se rozumí tak, že by aplikace měla synchronizovat emaily napříč platformami a zařízeními. Tím se zajistí, že práce s elektronikou poštou bude konzistentní jak na iPadu, Windows počítači a Android telefonu. Synchronizace probíhá pomocí emailového serveru s využitím protokolů IMAP a POP. [6]
- **Vysoká míra doručitelnosti:** Tato míra označuje, kolik hromadných emailů není označeno za spam a je úspěšně doručen koncovým uživatelům. Rozmezí mezi 95 % až 99 % je považováno za excelentní. [6]
- **Zabezpečení:** Ochrana proti nevyžádané poště a škodlivému softwaru pomocí detektorů spamu a malware. [6]

1.5.1 Některé mobilní mailové aplikace

V této sekci budou podrobně popsány dvě nejpopulárnější mobilní emailové aplikace, a to na základě jejich klíčových vlastností a silných stránek, které je řadí na přední místa na celosvětovém trhu s emailovými aplikacemi.

1.5.1.1 Gmail

Gmail se řadí mezi nejlepších mobilních emailových aplikací díky svému modernímu a uživatelsky přívětivému rozhraní. Jeho popularita a vysoké hodnocení na Apple App Store jej staví na přední místo mezi emailovými klienty. Svou funkcionalitu rozšiřuje o organizování mailů, vyhledávání, integrace s aplikací pro chat a videohovory, a inteligentní nástroje jako je jazykový překladač. [8]

Pro osobní účely je aplikace zcela zdarma, avšak pro využití některých pokročilých funkcí je potřeba předplatné Google Workspace. Součástí těchto pokročilých funkcí jsou vlastní firemní emailové domény, zvýšené uložiště pro ukládaní souborů, zvýšená bezpečnost, zvýšený počet účastníků video hovorů a kancelářský software Dokumenty, Tabulky a Prezentace. Tyto pokročilé funkce jsou zvláště přínosné pro firmy, neboť jsou navrženy s ohledem na potřeby týmové spolupráce. [8]

Mezi klíčové vlastnosti aplikace Gmail patří [6] [8]:

- Automatická blokace více než 99.9 % spamů, phishingů, malwarů a škodlivých odkazů.
- Personalizovaná doporučení emailů.
- Možnosti pro organizaci emailů, včetně štítků, různých inboxů, vlajek, hvězdiček a dalších.
- Rozsáhlé vyhledávací funkce s pokročilými filtry.
- Automatický jazykový překlad.
- Možnost provádět ve vysoké kvalitě videohovory přes Google Meet a používat týmový chat přímo z aplikace.
- Schopnost odesílat a přijímat peníze prostřednictvím e-mailu.
- Nastavení pravidel a automatizace pro zpracování emailů.
- Široká paleta integrací s aplikacemi třetích stran.

1.5.1.2 Outlook

Microsoft Outlook je nejvíce využívanou aplikací ve velkých korporacích a podnicích. Je považován za jednu z nejpokročilejších emailových aplikací pro profesionální práci, díky svým schopnostem v oblasti správy úkolů a pokročilým bezpečnostním funkcím. Tento emailový klient nabízí nejen všechny profesionální nástroje, ale také umožňuje vytváření složitých pravidel pro zpracování pošty, automatizaci tvorby a správy úkolů a integraci tisíců nástrojů, od správy vztahů s klienty (CRM) až po fakturaci. Mimo těchto funkcionalit, také vyniká svou unikátní funkci "Přehrát mé emaily", která umožňuje audio procházení nových emailů s možností přeskakování, označování, archivace a mazání hlasem prostřednictvím virtuální asistentky Cortany. [8]

Pro osobní použití je Outlook dostupný zdarma, ale pro připojení vlastní domény a přístup k pokročilým funkcím je nutný plán Microsoft 365. Tyto pokročilé funkce zahrnují zabezpečený email pro podnikání, nejpokročilejší emailové funkce, zvýšené uložiště pro ukládání souborů, zvýšený počet účastníků video hovorů, týmový chat a kancelářský software Word, Excel, PowerPoint, Teams, OneDrive a další. [8]

Mezi klíčové vlastnosti aplikace se řadí [6] [8]:

- Spolupráce v reálném čase s ostatními Microsoft aplikacemi přes Microsoft Loop.
- Zefektivnění pracovních postupů, pomocí automatického vytváření a správy úkolů.
- Organizace emailů, včetně štítků, špendlíků a dalších pro lepší prioritizaci.
- Vytváření emailových šablon.
- Pokročilá pravidla pro správu mailů.
- Spojení více emailových účtů a kalendářů v jednom rozhraní.
- Vestavěné nástroje pro kontrolu gramatiky a pravopisu.
- Umožnění videohovorů s integrací Teams, Skype a Zoom.

2 EMAIL STANDARDY

Na konci první kapitoly byly představeny některé populární mobilní mailové aplikace. V této kapitole budou představený hlavní protokoly, díky kterým je možné samotné emaily odesílat, přijímat a spravovat.

2.1 Internet Message Format

Internet Message Format (IMF) je standardizována struktura emailových zpráv, která je vyžadována SMTP protokolem. Norma IMF je definována v RFC 5322 a vyžaduje, aby zprávy používaly výhradně znaky US-ASCII a byly rozděleny do řádků. Řádek je řetězec znaků, která je ukončena tzn. *carriage-return* (CR) ihned následovaným tzn. *line-feed* (LF), což je běžně zkráceno jako CRLF. [9] CR vrátí kurzor na začátek daného řádku bez změny textu, reprezentovaný řetězcem znaků *„\r“* (0x0D hex). LF pro změnu posune kurzor na nový a prázdný řádek, reprezentovaný řetězcem znaků *„\n“* (0x0A hex). [10] Každý řádek znaků je omezen na maximálně 998 znaků. Z důvodu efektivity je však doporučena délka maximálně 78 znaků. [9]

Emailová zpráva, která splňuje standard IMF, se skládá z hlavičky a těla. Tyto dvě sekce jsou odděleny prázdným řádkem. Tělo *(body)* jsou pouhé řádky US-ASCII znaků. Na druhou stranu hlavička *(header)* má definovanou strukturu, která obsahuje povinné a nepovinné parametry. [9]

Mezi povinné parametry patří [9]:

- ***Origination date*:** Jedná se o datum a čas jenž indikuje, že zpráva byla kompletní a připravená přejít do systému k odeslání. (Značí se *„Date:“*)
- ***From*:** Toto pole obsahuje seznam odesílatelů. (Značí se „*From:*“)
- ***Sender*:** Mail odesílatele. (Značí se *„Sender:“*)

Do nepovinných parametrů pak spadá: *reply-to*, *to*, cc, bcc, message-id, in-reply-to, references, subject, comments, keywords. [9]

2.2 SMTP

Simple Mail Transfer Protocol (SMTP) je standard definovaný v RFC 5321 [11]. Jeho hlavní úlohou je umožnit přenos emailů mezi zařízeními a servery nebo mezi samotnými servery. SMTP slouží pouze pro doručování emailových zpráv, a ne pro jejich načítání do emailového klienta. Pro načtení emailových zpráv ze serveru jsou určeny jiné standardizované protokoly. Při zasílání emailu, emailový klient zašle společně s emailem tzn. obálku (*envelope)*, která obsahuje informace o tom, odkud email pochází a kam putuje. Tato obálka je od hlavičky a těla emailu oddělena, a tak není pro koncového uživatele viditelná. [12] Emailová hlavička je tedy určená pouze pro zobrazení emailových dat uživateli a obálka na druhou stranu slouží pro server, který z ní vyčte informace o finální destinaci [13].

Ve stručnosti, komunikace pomocí SMTP protokolem funguje následovně [12]:

- **Otevření SMTP spojení:** S využitím protokolu TCP se mezi klientem a serverem naváže TCP spojení. Emailový klient zahájí proces odesílání emailu s příkazem *"Hello"*.
- **Přenos emailových dat***:* Klient pošle serveru sérii příkazů společně s hlavičkou a těla emailu.
- **Mail Transfer Agent***:* Na serveru běží program Mail Transfer Agent (MTA), který kontroluje doménu emailové adresy příjemce. Pokud se liší od domény odesílatele, MTA použije DNS k zjištění IP adresy příjemce.
- **Uzavření spojení:** Klient informuje server o dokončení přenosu dat a server spojení uzavře. Server nebude přijímat další data od klienta, pokud klient neotevře nové SMTP spojení.

2.2.1 Simple Mail Transfer Protocol Secure

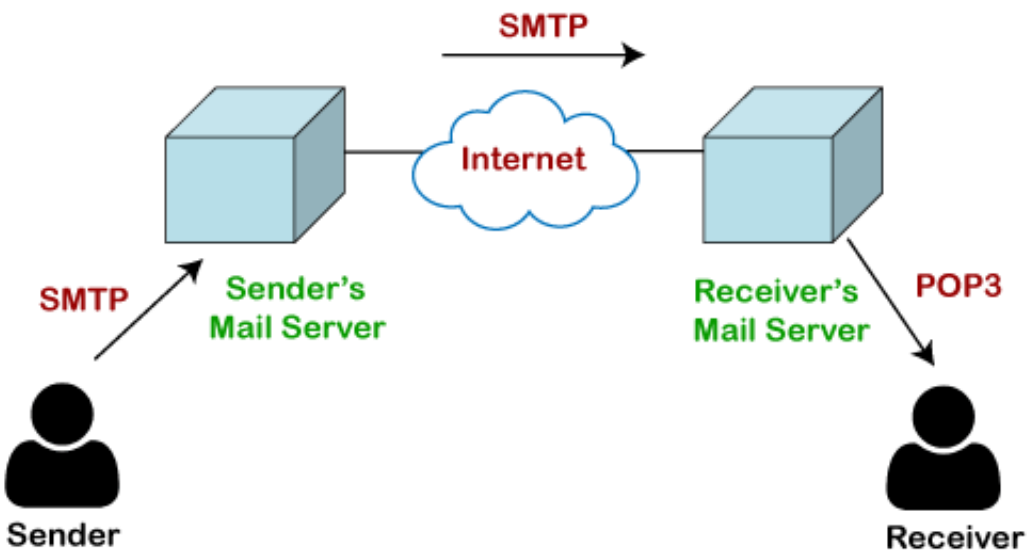
Simple Mail Transfer Protocol Secure (SMTPS), přináší do protokolu SMTP bezpečnostní vylepšení. Zavádí autentizaci účastníků, integritu dat a jejich důvěrnost s využitím SSL nebo TLS. Klient a server nadále komunikují přes standardní SMTP na aplikační vrstvě, celé připojení je však chráněno skrze SSL nebo TLS. [14]

2.3 Post Office Protocol

Post Office Protocol (POP) je protokol navržený pro přenos emailů z emailového serveru na lokální zařízení uživatele, a to po jejich odeslání a doručení skrze SMTP. POP je tzn. *pull*, jenž znamená, že emaily se stahují přímo ze serveru do zařízení uživatele. Po stažení je email smazaný ze serveru a stává se dostupným pouze na tomto zařízení. [15] Aktuální verze tohoto protokolu, POP3, je specifikována v RFC 1939 [11].

Hlavní výhodou protokolu POP je schopnost okamžitého přístupu k emailům i bez aktivního internetového připojení, což umožňuje čtení emailů offline. To přináší výhodu pro emailový server, protože tyto emaily neuchovává a jsou tak nižší paměťové požadavky. [15]

Na druhou stranu se nevýhodou stává nemožnost přistupovat k emailům na jiném zařízení, a to za předpokladu že není server nastaven pro uchování kopie příslušného emailu. Stahování přímo na lokální zařízení s sebou nese i riziko stáhnutí škodlivého softwarů. [15]

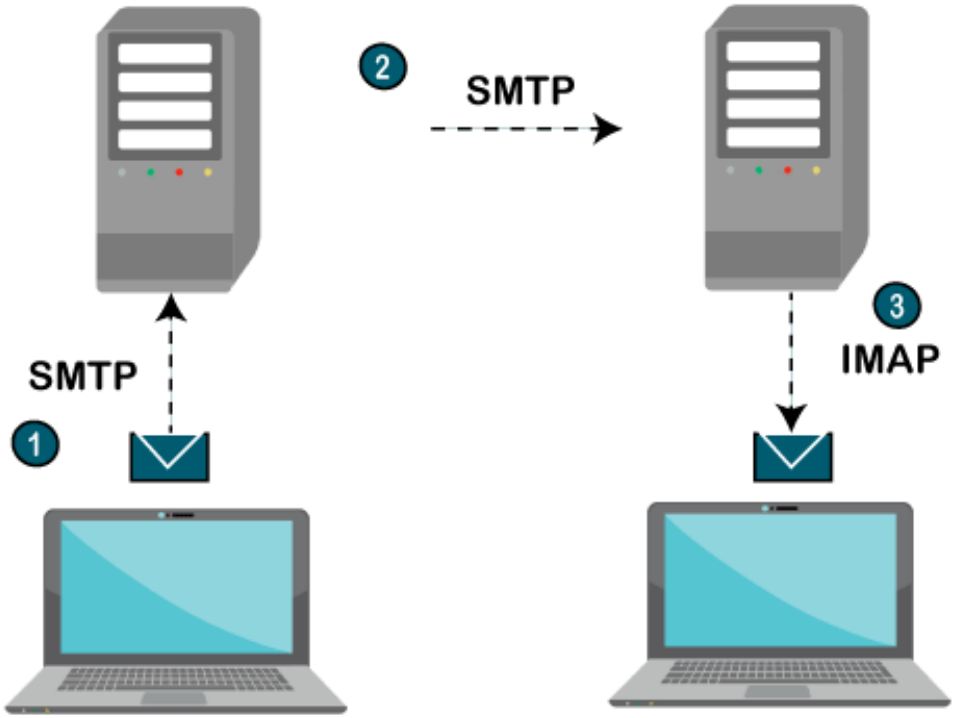


Obrázek 1: Proces odeslání a stažení emailu pomocí SMTP a POP3 [15]

2.4 Internet Message Access Protocol

Internet Message Access Protocol (IMAP) je navržený pro získávání emailů z emailového serveru. Tento protokol se skládá z klientské a serverové části, a pro připojení mezi nimi využívá protokolu TCP na transportní vrstvě. Pro nezašifrované připojení IMAP server poslouchá na portu 143. Kromě toho je k dispozici i možnost bezpečného zašifrovaného připojení prostřednictvím portu 993.Akturální verzí protokolu je IMAP4, která je definovaná v RFC 3501. [16]

V porovnání s protokolem POP, má IMAP mnohem flexibilnější řešení s širokou škálou funkcí. Uživatelé mohou s IMAP označovat emaily jako „přečtené“, vytvářet více mailboxů a třídit emaily do různých kategorií. Tento protokol také podporuje vyhledávání emailů a umožňuje stahovat jen části emailu, například pouze předmět a odesílatele, které se zobrazí v aplikaci před plným načtením obsahu emailu po jeho otevření. Díky tomu, že všechny emaily zůstávají na serveru, IMAP umožňuje synchronizaci mailboxů a emailů mezi různými zařízeními, což zahrnuje i jejich rozřazení do složek a odstraněné emaily, čímž zajišťuje konzistenci napříč všemi zařízeními. [16]



Obrázek 2: Proces odeslání a získání emailu pomocí SMTP a IMAP [16]

3 AUTENTIZACE

V předešlých kapitolách byly představeny některé mobilní mailové aplikace. Tyto aplikace umožňují uživatelům přihlášení, aby měli zabezpečená svá data a mohli spravovat své emailové účty. K zabezpečení přístupu k těmto údajům slouží autentizační procesy, které budou v této části práce popsány.

Autentizace je proces, ve kterém se uživatel ověří, že je tím, za koho se vydává. Typicky se tento proces využívá k zabezpečenému přístupu k datům anebo aplikaci. K autentizaci se využívá identifikátor v podobě uživatelského jména anebo emailu a také hesla. Následně jsou tyto identifikátory poslány na server, kde dojde k porovnání těchto údajů, pokud se přístupové údaje shodují s těmi v databázi, tak je uživateli přidělen přístup. [17]

Bezpečnostní systémy využívají různé mechanismy pro autentizaci uživatele. Tyto mechanismy se rozdělují do tří kategorií.

První z nich je *„something you know“*, kde spadají bezpečnostní otázky a hesla. To je z důvodu, že tyto informace zná pouze daný uživatel, který se chce autentizovat a získat přístup ke svým datům či aplikaci. [17]

Další typ se označuje jako *„something you have“*, do kterých patří zařízení, jako jsou USB s bezpečnostními tokeny a mobilní telefony. Mobilní telefony fungují při autentizaci tak, že když se uživatel pokouší přihlásit ke službě, tak mu dorazí jednorázový PIN kód v SMS anebo v aplikaci, který slouží k ověření totožnosti. [17]

Posledním typem je *„something you are“*, ve kterém hraje roli biometrická autentizace. Tato autentizace je v podobě naskenování rysu v obličeji anebo otisků prstů. Jelikož jsou fyzické charakteristiky každého jedince unikátní, tak je tento typ autorizace bezpečný. [17]

V procesu autentizace existuje zranitelnost, jestliže není stav autentizace neustále kontrolovaný na straně serveru. Tato zranitelnost se nachází zejména pokud se posílá stav v součástí požadavků, např. jako parametr *„authenticated = no“*. Zde by útočník mohl jednoduše změnit parametr na *„yes“* a tak jednoduše obejít autentizační proces. Pro zachování bezpečnosti je potřeba použít jednu z autentizačních přístupu, jako je *session-based* anebo *token-based*. [18]

3.1 Session-based authentication

Tento typ autentizace se také označuje jako *stateful authentication*, protože je na straně serveru uchováván stav autentizace. Při úspěšném přihlášení uživatele je vygenerován unikátní identifikátor relace (*Session ID*). Tento identifikátor je tzn. *opaque*, jenž znamená, že neobsahuje uživatelská data. Identifikátor má pouze jednu úlohu, a to ukazatel na relaci, která je uchována na serveru. [18]

Zde je nastínění, jak tento typ autentizace funguje v praxi [18]:

1. Uživatel zadá přihlašovací údaje, které aplikace pošle na server.
2. Server tyto údaje validuje a pokud jsou platné, tak vytvoří relaci s náhodně vygenerovaným identifikátorem.
3. Server pošle aplikaci identifikátor relace.
4. Aplikace společně s požadavky pak posílá i identifikátor, který server validuje. Pokud je validní, tak načte data z příslušné relace.
5. V případě, kdy se uživatel odhlásí, tak aplikace pošle požadavek na odhlášení a server vymaže relaci.

Pro maximalizaci bezpečnosti tohoto typu autentizace, je potřeba dodržet následující doporučení [18]:

- Identifikátory relací jsou náhodně generovány, a to s dostatečnou délkou a entropií.
- Identifikátory relací jsou zasílány pomocí HTTPS.
- Server verifikuje identifikátor vždy, když je požadavek na chráněné zdroje.
- Relace je odstraněná při odhlášení anebo vypršení platnosti relace.
- Nesnažit se implementovat celý tento autentizační proces sám. Namísto toho je vhodnější použít ověřené knihovny a frameworky.

3.2 Token-based authentication

Tento proces autentizace je znám také pod názvem *Stateless authentication*, jelikož je relace zakódovaná jako JSON objekt v tokenu, který se posílá do aplikace. Hlavní myšlenkou této autentizace je ta, že server nemusí ukládat žádný autentizační stav nebo data relace. Pro prevenci před kompromitaci autentizačních dat jsou tokeny chráněné pomoci

kryptografického elektronického podpisu. Nejpoužívanější formát tokenů je JSON Web Token (JWT). Který je definován v RFC 7519. [18]

JWT se skládá ze tří části, které jsou oddělené tečkou [18] [19]:

- ***header*:** Hlavička se následně skládá ze dvou částí v následujícím formátu:
  - `{"alg":"HS256","typ":"JWT"}`
  - Kde parametr *alg* definuje typ hash algoritmu, který se použil na podpis
  - A parametr *typ*, který udává typ tokenu.
- ***payload*:** Tato část obsahuje tzn. *„claims“* a má následující strukturu:
  - `{"sub":"1234567890","name":"John Doe","admin":true}`
  - První parametr spadá do *„Registered claims“*: Obsahuje nepovinné parametry, jako jsou například: *issuer*, *expiration time* a *subject*.
  - Druhý parametr spadá do *„Public claims“*: Obsahuje veřejné generické informace, sloužící pro spolupráci s aplikacemi třetích stran, jako je například: *name* a *email*.
  - Poslední parametr spadá do *„Private claims“*: Obsahuje specifické informace potřebné pro autentizace v aplikaci, například: *employee ID* nebo *admin*.
- ***signature*:** Poslední část již obsahuje samotný hash získány z elektronického podpisu.

Výsledný podpis se nadále vytváří pomocí algoritmu, definovaným v hlavičce, a tajného klíče. Tento podpis se pak kontroluje společně s datem expirace při přístupu ke chráněným zdrojům a zaručuje, že životnost tokenu není prošla a že se samotnými daty nebylo manipulováno. [18]

Nejběžnější způsob implementace tohoto autentizačního procesu je použití dvou tokenů. Jeden z nich je přístupový token (*access token),* který má formát JWT a byl popsán výše. Přístupový token má krátkou živostnost s nastavením *expiration time* a po jeho vypršení se použije tzn obnovovací token (*refresh token*). Obnovovací token má na druhou stranu delší životnost a používá se pro získání nového přístupového tokenu. [18] V případě nevalidního obnovovacího tokenu by mělo dojít ke smazání refresh tokenů na straně serveru a smazání obou tokenů uložených v aplikaci uživatele s následným odhlášením.

Doporučuje se dodržovat níže uvedená doporučení pro maximální zabezpečení při použití tohoto typu autentizace [18]:

- Při každém požadavku na přístup k chráněným zdrojům na serveru je nutné provádět kontrolu expirace a integrity tokenu.
- Privátní klíč, pro vytváření elektronického podpisu, musí být uchován v tajnosti.
- JWT by neměl obsahovat osobní údaje uživatele, v opačném případě by JWT měl být i zašifrován.
- Na mobilním zařazení, by tokeny měly být uložený v bezpečném uložišti.

3.3 OpenID Connect

OpenID Connect (OIDC) je autentizační vrstva, postavená na autorizačním protokolu OAuth 2.0. OIDC umožňuje aplikacím třetích stran verifikaci identity uživatele a k tomu také získání informací o uživateli. Další vlastností je umožnění registrace či přihlášení pomocí jednoho tlačítka. Například kdy se chce uživatel registrovat na webové stránce přes tlačítko „Google“ anebo „Facebook“. Po stisknutí tlačítka je uživatel přesměrován na stránku Google, kde se přihlásí a autorizuje sdílení informací s touto webovou stránkou. Po úspěšné autorizaci jsou zaslány stránce informace pomocí standardu JWT, které se označují jako *ID token*. V těchto informacích se může nacházet jméno, email, datum narození, pohlaví a další. [20]

4 AUTORIZACE

V současné době je důležité, aby emailový klient podporoval připojení více emailových účtů, což umožňuje spravovat všechny emaily v rámci jedné aplikace. Proces propojení účtů a ochranu osobních přihlašovacích údajů zajišťuje autorizace. Tento mechanismus chrání údaje uživatele před sdílením s aplikacemi třetích stran, což bude v této kapitole dále rozebráno

Autorizace je proces, který určuje uživateli anebo službě úroveň přístupu k datům, nebo k vykonání konkrétní operace. Příkladem může být Access Control List (ACL), který určuje přístupová práva uživatelům, kde mohou být klasičtí uživatelé a administrátoři. Pokud by uživatel chtěl provést změny na nějaké webové službě, které by ovlivnily bezpečnost, tak mu ACL odmítne přístup, ale pokud by tyto změny chtěl provést administrátor, tak mu to ACL povolí. [17]

4.1 OAuth 2.0

Open Authorization (OAuth) je standardizovaný autorizační framework, který umožňuje aplikacím třetích stran získat bezpečný přístup k datům uživatele, aniž by uživatel musel sdílet své přihlašovací údaje s touto aplikací třetí strany. [21]

Dříve se musely zadávat přihlašovací údaje přímo aplikaci třetí strany, aby mohla přistupovat například ke Gmail účtu, to však nebylo z bezpečnostního hlediska vůbec správné. Navíc tak aplikace získala plný přístup k uživatelskému účtu. Díky OAuth je možné určit, jaké práva by měla mít aplikace třetí strany, například pouze číst kontakty, a ne číst a posílat maily. [21]

4.1.1 Role

OAuth definuje čtyři role:

Resource owner:

- Rozumí se jako entita, která uděluje přístup k chráněným zdrojům. Pokud je *resource owner* člověk, tak je označován jakožto *end-user*. [22]

***Resource server*:**

- Označuje server, který hostuje chráněné zdroje. Tento server je schopný reagovat na požadavky na chráněné zdroje, pomocí *access tokens*. [22]

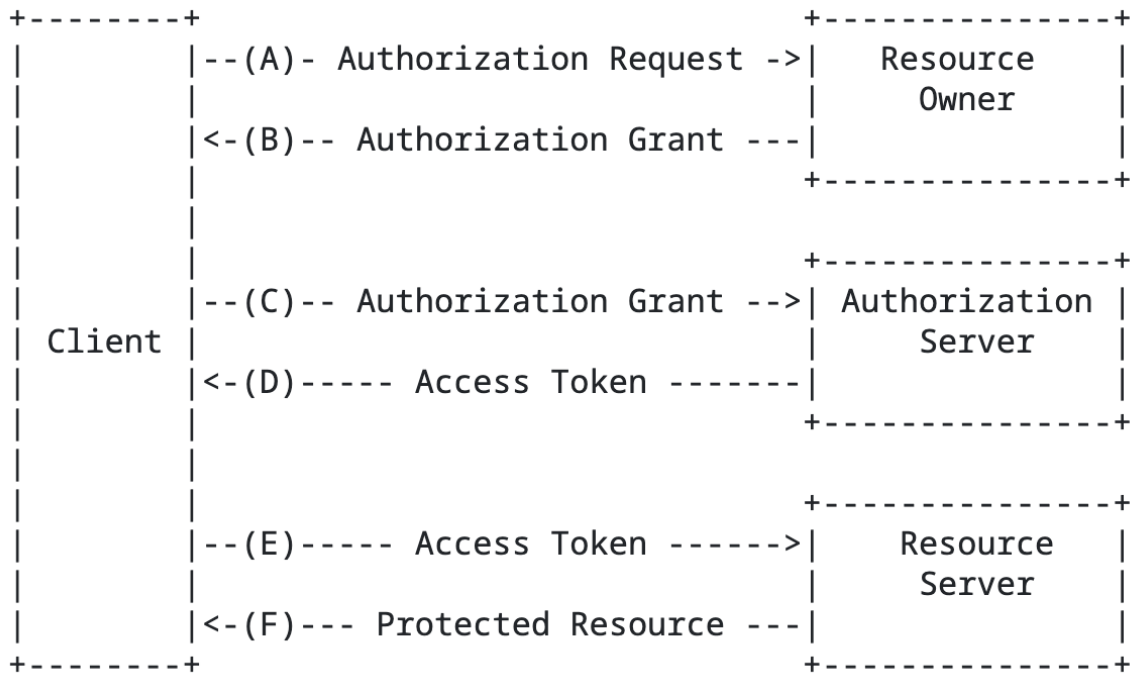
***Client*:**

- Chápe se jako aplikace, která vytváří požadavky na chráněné zdroje, pod vlastníkovým jménem a jeho oprávnění. Také je důležité zmínit, že označení *client* nespecifikuje konkrétní implementační charakteristiky. Aplikace tak může být spuštěna jak na serveru, počítači, mobilním zařízení a dalších zařízeních. [22]

***Authorization server*:**

- Jedná se o server, který uděluje *access tokens* roli *client*. Za podmínky, že *client* obdržel potřebnou autorizaci. [22]
- *Authorization server* může být jakožto samostatná entita anebo stejný server, jako *resource server*. [22]

4.1.2 Interakce mezi rolemi



Obrázek 3: Interakce mezi rolemi v OAuth 2.0 [22]

A) *Client* zasílá napřímo požadavek na *Resource owner*, kde požaduje od této role autorizaci. Preferovanější alternativou je zaslání požadavku nepřímo přes *Authorization server*, který by se choval jako prostředník. [22]

B) *Client* obdrží *Authorization grant*, který nese autorizační oprávnění role *resource owner*. [22] Existuje několik typů těchto grantů, které budou následně popsány v práci.

C) *Client* požaduje *Access Token*, k tomuto požadavku prezentuje udělený *Authorization grant*. [22]

D) Authorization server provede autentizaci klienta a ověří platnost uděleného grantu. Pokud je validace úspěšná, tak *client* obdrží *access token*. [22]

E) Se získaným tokenem *client* posílá požadavek o získání chráněných zdrojů na *Resource server*. [22]

F) *Resource server* tento token ověří a pokud je validní, tak zašle chráněné zdroje. [22]

4.1.3 Authorization grant

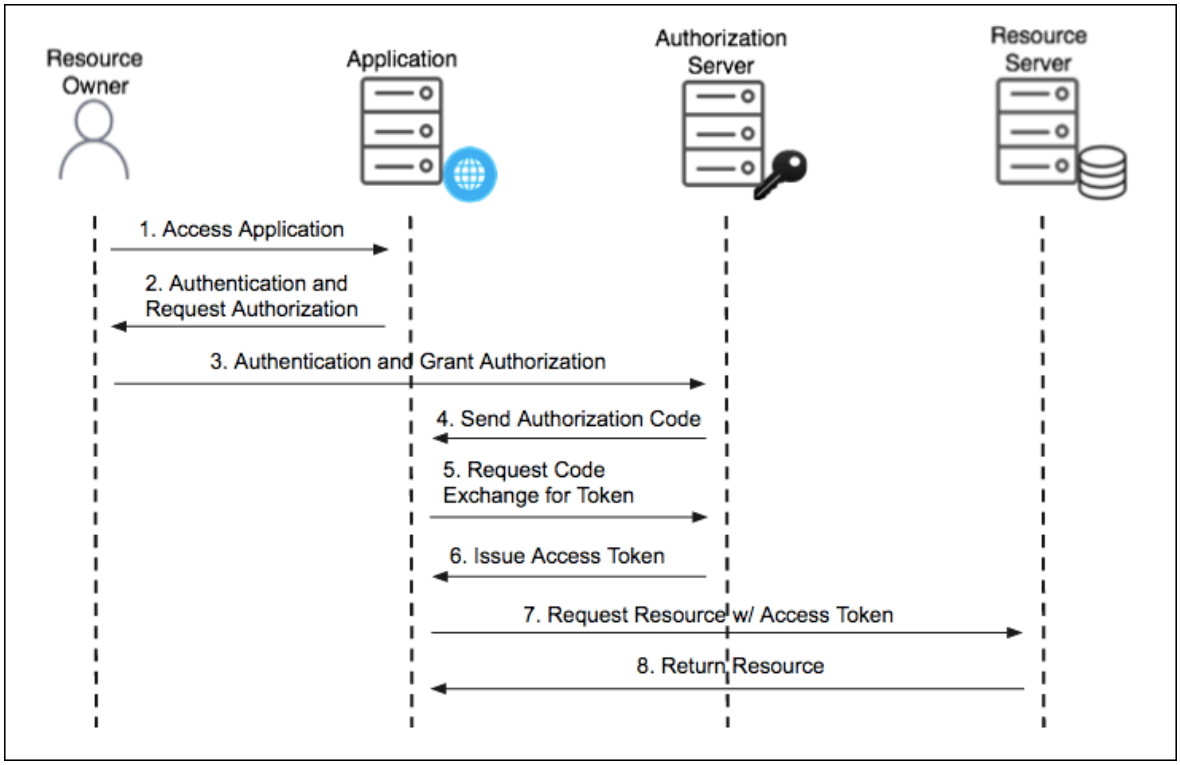
Authorization grant popisuje, jak se získávají přístupové tokeny (*access tokens)*. Jedná se o souhrn pravidel, tok dat a definování celého procesu autorizace pro obdržení tokenů. [21]

4.1.3.1 Authorization Code grant

Jedná se o nejběžnější a zároveň nejbezpečnější typ OAuth grantu, kdy je samotný uživatel zapojen. Hlavní složkou tohoto typu grantu je autorizační kód (*Authorization Code)*, jedná se o řetězec náhodných ASCII znaků, který je vyměněn za *access token*. [21]

Ten je získaný tak, že se autorizační server (*Authorization server*) stane prostředníkem mezi aplikací (*client*) a uživatelem (*resource owner*), kde aplikace přesměruje uživatele na autorizační server. Přesměrování je v podobě zobrazení přihlašovacího formuláře (v prohlížeči) pro službu, u které je třeba získat autorizaci. Po úspěšné autentizaci uživatele a získání jeho autorizace, přesměruje *Authorization Code* zpět na aplikaci. Přesměrování zpět na aplikaci je umožněno pomocí takzvaného „*redirect URI",* jenž se zasílá v těle *requestu* při autentizaci a autorizaci uživatele. *Redirect URI* v sobě obsahuje adresu URL pro již zmiňované přesměrovaní na aplikaci po úspěšné autorizaci a autentizaci uživatele. V tomhle momentu může aplikace vyměnit *Authorization Code* za *access token*, díky kterému může zasílat požadavky na server se zdroji (*Resource server*). [21] [22]

Jelikož se autentizace prováděla na autorizačním serveru, tak aplikace nezná přihlašovací údaje uživatele, což má velký bezpečnostní benefit. Další bezpečnostní prospěch je v podobě předávání *access token* přímo do aplikace bez nutnosti ho předávat skrze prohlížeč, kde by mohl být odcizen útočníkem. [21] [22]



Obrázek 4: Authorization Code Grant Flow [23]

Nyní bude popsán bodově postup s parametry tohoto typu grantu. Přiložený obrázek nekoresponduje s těmito body, slouží pouze pro vizualizaci fungování toho typu grantu. Následující body byly čerpány z těchto zdrojů: [21] [24].

1. *Client* přesměruje uživatele na *Authorization server* s těmito parametry:
    a. *response_type*: zde bude řetězec „*code*“.
    b. *client_id*: zde se zadává ID role *client,* jenž je vygenerovaný OAuth serverem.
    c. *client_secret:* jedná se o privátní klíč, který vygeneroval OAuth server. Tento klíč nesmí být veřejný a musí být uložen na serverové části aplikace.
    d. *redirect_uri:* URL, na kterou má být *client* přesměrován po úspěšné autentizaci a autorizaci uživatele.
    e. *scope*: jedná se o list pravomocí o které *client* žádá, jenž musí být autorizovaný uživatelem.
    f. *state*: volitelný parametr, do kterého se vkládá CSRF token.
2. Tyto parametry jsou ověřeny na *Authorization server*.
3. Uživatel je osloven, aby se přihlásil a autorizoval přístupy definované v listu *scope*
4. Pokud uživatel autorizoval všechny oprávnění, které *client* žádá, tak je přesměrován na URL (definovaný v parametru *redirect_uri*). Součásti tohoto přesměrování je také *query string*, jehož součástí jsou tyto parametry:

   a. *code*: Tady se nachází *Authorization Code.*
   b. *state*: tento vrácený token může *client* porovnat s tím odeslaným a ověřit, že vrácený *Authorization Code* souvisí s požadavkem, který odeslal tento *client,* a ne cizí *client.*
5. *Client* zašle *POST request* na autorizační server, tento požadavek musí obsahovat následující parametry:
   a. *client_id:* stejné ID jak v prvním bodě.
   b. *client_secret:* stejný privátní klíč, jak v prvním bodě.
   c. *grant_type*: zde se musí specifikovat typ grantu, v tomto případě je to *„authorization_code“*.
   d. *redirect_uri*: URL adresa, která se definovala v prvním kroku.
   e. *code*: tady se vkládá *Authorization Code*, získaný z předchozího kroku.
6. Autorizační server odpoví s JSON objektem, jenž obsahuje následující parametry:
   a. *token_type*: zde je řetězec *„Bearer“*.
   b. *expires_in*: číselná hodnota, která definuje dobu platnosti tokenu.
   c. *access_token*: přístupový token, pro přístup ke chráněným zdrojům.
   d. *refresh_token*: pokud je požadován, tak je vrácen a slouží pro obnovu přístupového tokenu.

4.1.3.2 Proof Key for Code Exchange

PKCE („*Pixy*“) je bezpečnostní vrstva, která ujišťuje, že *Authorization code* nemůže být odcizen útočníkem, který by zachytil komunikaci. [21]

Client tak vygeneruje privátní klíč, na který použije hash funkci SHA-256. Hash funkce jsou jedno cestné, což znamená, že z vygenerovaného hash řetězce není možné zjistit původní hodnotu. Následně při poslání požadavku pro získání autorizačního kódu (*Authorization code*), přidá navíc parametry *code_challenge*, ve kterém je samotný hash a *code_challenge_method*, ve kterém se specifikuje formát parametru *code_challenge*. Specifikace formátu je z důvodu, že se *code_challenge* může zaslat i v čisté podobě, ale to by mohlo mít negativní důsledky na bezpečnost. [21]

Při výměně autorizačního kódu za přístupový token (*access token*), *client* v POST požadavku přidá navíc parametr *code_verifie*r, do kterého vloží privátní klíč. Ten následně autorizační server projede hash funkcí a výsledný hash porovná se zaslanou hodnotu

code_challenge. Pokud se tyto hodnoty budou shodovat, tak autorizační server zašle přístupový token. [21]

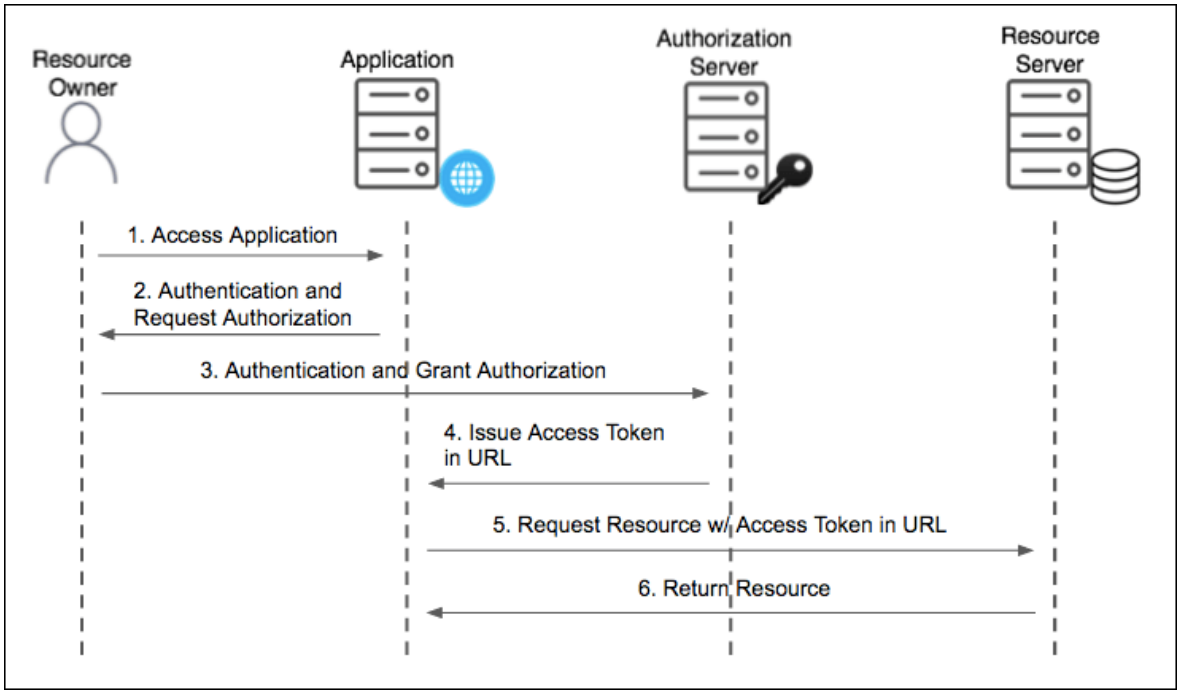
PKCE není vyloženě nutná, když *client* zasílá *client_id a* client_secret, ale je vhodné jí použít, protože dodává bezpečnostní nadstavbu, která chrání před CSRF a „*authorization code injection attacks“*. [21]

V momentě, kdy aplikace nemá vlastní backend a je veškerá logika aplikace na straně aplikace, jako je například mobilní aplikace anebo SPA, tak se označuje jakožto „*public client“*. To znamená, že *client secret* je možno vyčíst z kódu, takže k němu má případný útočník jednoduchý přístup. V tomto případě je nutné využít metodu PKCE. [21]

4.1.3.3 Implicit grant

Tento typ grantu by se neměl již používat, jelikož byl vyřazen z nejnovější verze specifikace OAuth 2.1. Důvodem vyřazení grantu je, že vynechává důležitý krok bezpečného získání přístupového tokenu (*access token*) z autorizačního serveru. [21]

U *Implicit grant* nedochází k přesměrování s autorizačním kódem (*Authorization Code)* na aplikaci (*Client*), tak jak je to u předchozího grantu *Authorization* Code *grant*. Namísto toho dochází k vrácení přístupového tokenu, jako část v URL adrese. Což znamená, že má k přístupovému tokenu každý přístup a je lehce odcizitelný útočníkem, což by vedlo ke kompromitaci dat uživatele. [21]



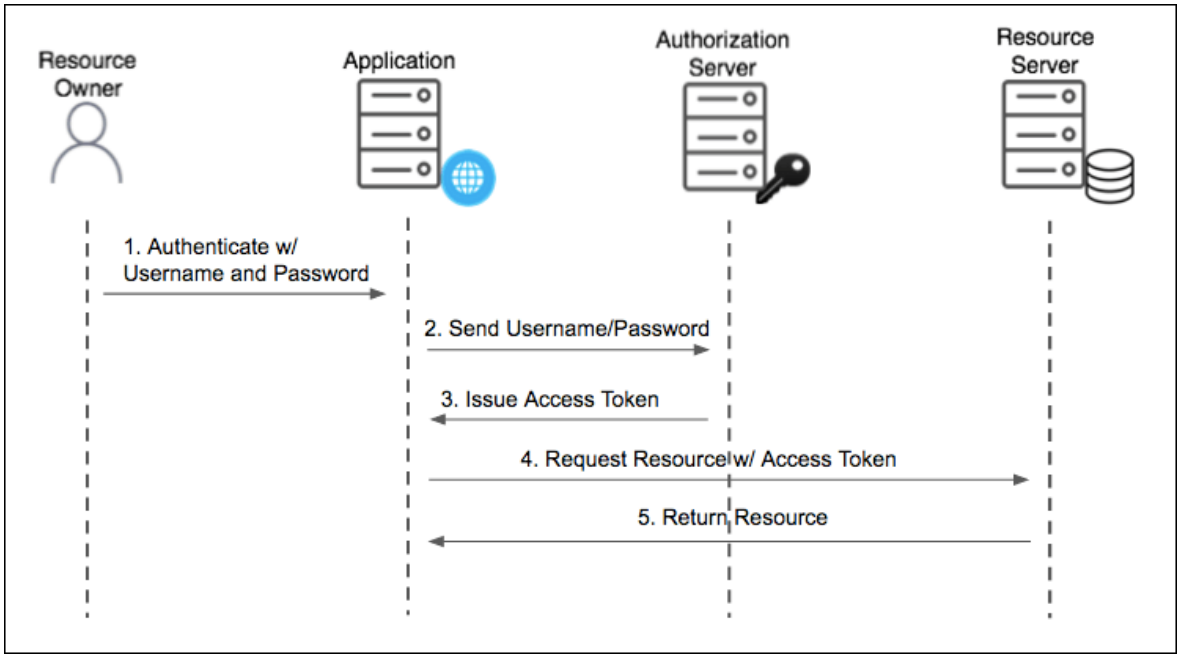
Obrázek 5: Implicit Grant Flow [23]

4.1.3.4 Resource Owner's Password Credentials grant

Také se jedná o zastaralý typ grantu a není doporučeno jej používat. Jeho využití je dobré v případě, kdy se provádí migrace ze zastaralé metody autentizace a autorizace na OAuth 2.0. Mnoho nativních mobilních aplikací a zastaralých webových aplikací využívají tuto metodu. Ve většině případů nechtějí tyto mobilní aplikace otevírat webový prohlížeč z důvodu, že chtějí uživateli prezentovat vlastní UI na přihlašovací formulář. [21]

Hlavní nevýhodou této metody je, že aplikace sbírají uživatelské přihlašovací údaje, které pak zasílají autorizačnímu serveru. Aplikace tak musí zajistit bezpečné ukládání těchto údajů. Což se výrazně liší od metody *Authorization Code grant*, kde se tyto údaje předávají přímo autorizačnímu serveru. Další hlavní nevýhodou je absence výhod, které nabízí samotný autorizační server, jako jsou: více faktorová autentizace, reset hesla, registrace, email verifikace a autentizace bez nutnosti použití hesla. [21]

Z těchto důvodu byla tato metoda vyřazena z nejnovější verze specifikace OAuth 2.1. [21]



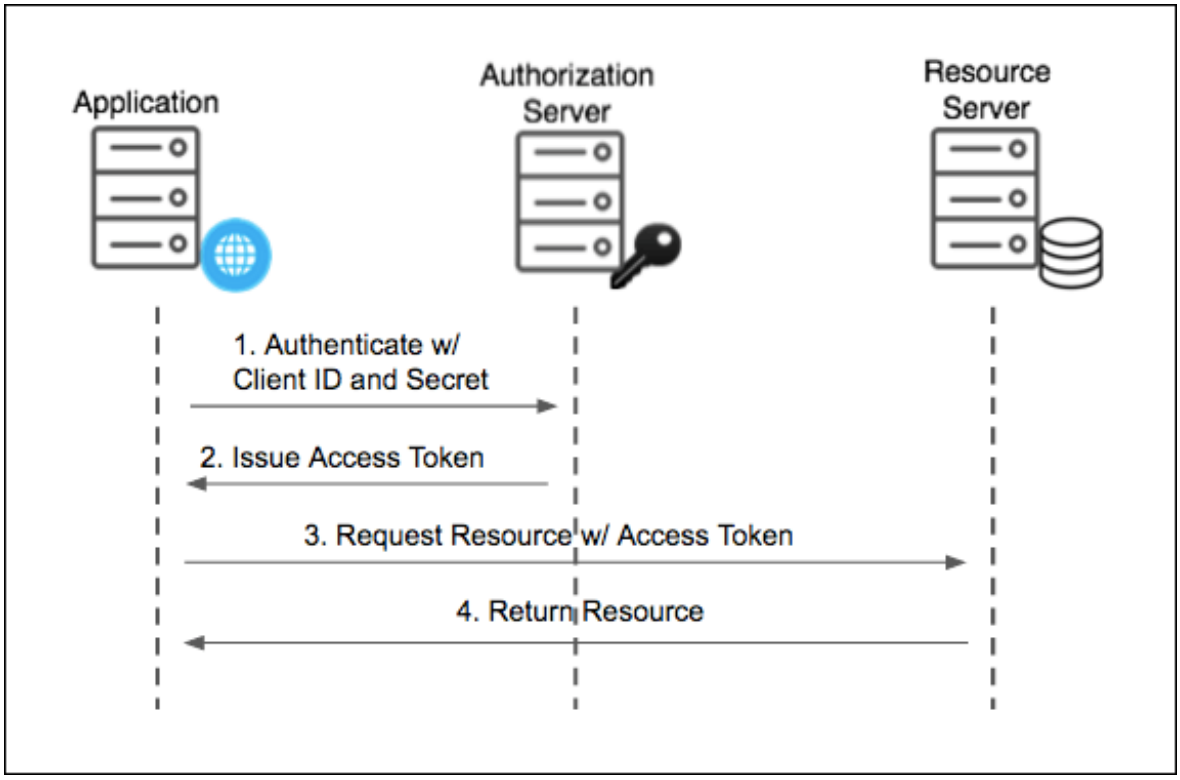
Obrázek 6: Resource Owner Password Grant Flow [23]

4.1.3.5 Client Credentials Grant

V této metodě není zahrnutý uživatel, jelikož je určená pro autorizaci mezi dvěma aplikacemi. Kde jeden *client* žádá autorizaci k prostředkům, který vlastní druhý *client*, na který následně zasílá požadavky (často ve formě vytvořeného API). Tak jako u ostatních metod, je zapotřebí mít *access token*, pro přistupovaní ke prostředkům jiné aplikace. [21]

Access token se získává pomocí zaslání požadavku na autorizační server, kde se do parametru zadají:

a) *client_id:* jenž je unikátní identifikátor aplikace. [21]
b) *client_secret:* jedná se o privátní klíč, který poskytuje autorizační server, který musí být držen v utajení. [21]
c) *grand_type:* tento parametr označuje, že se používá metoda *Client Credentials Grant.* [21]

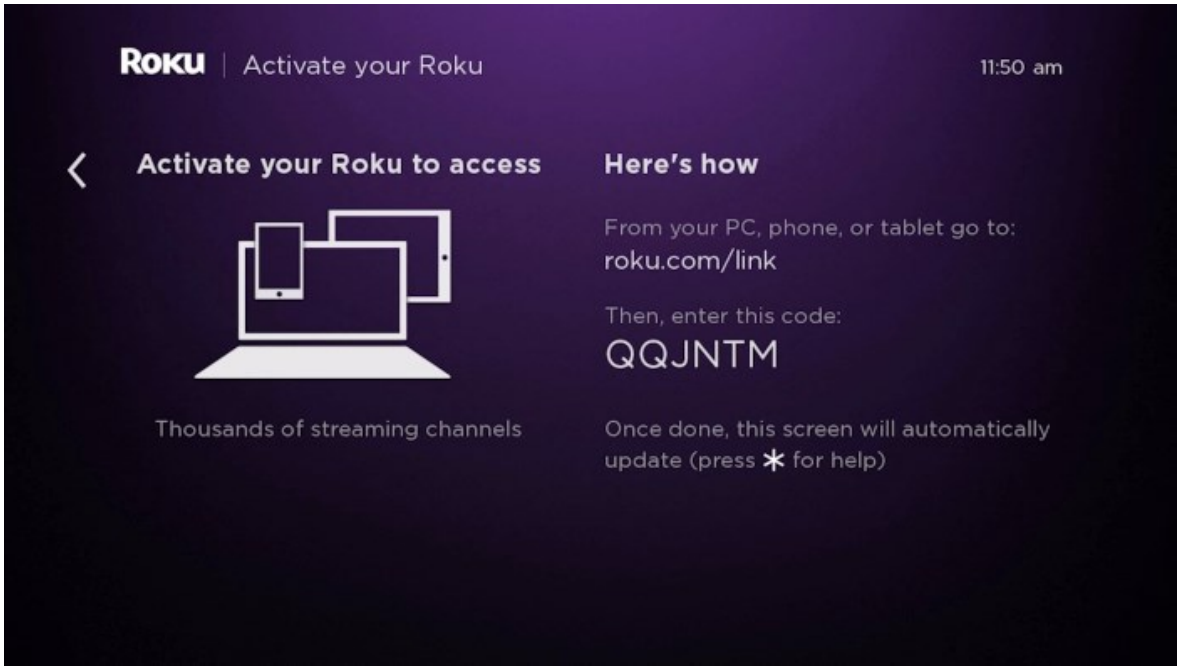


Obrázek 7: Client Credentials Flow [23]

4.1.3.6 Device grant

Tato metoda je využívaná na zařízeních, které nejsou uzpůsobené na zadávání přihlašovacích údajů, jelikož nemají klávesnici. Takové zařízení mohou být například herní konzole a televize, kde není příjemné psát zdlouhavé údaje na ovladači. Zvlášť kdy se uživatel může přepsat v heslu, a tak ho zadávat znovu. [25]

Proto byla zavedená tato standardizovaná metoda, kde stačí mít splněné tyto jednoduché požadavky. Je potřeba aby zařízení bylo připojeno k internetu, bylo schopné vytvářet HTTPS požadavky a také bylo schopné zobrazit uživateli kód, který se používá pro autorizaci. Posledním požadavkem je, aby uživatel vlastnil druhé zařízení, jako je mobilní telefon anebo počítač. [25]



Obrázek 8: Obrazovka na zařízení u Device grant [25]

4.1.3.7 Refresh token grant

Tato metoda je určená pro získání nového přístupového tokenu (*access token*), poté co jeho platnost vyprší. Každý přístupový token má určenou dobu platnosti a po uplynutí této doby je potřeba zaslat *refresh token* na autorizační server, který následně vrátí platný přístupový token s novou dobou platnosti. *Refresh token* není vydáván defaultně, ale je třeba si o něho zažádat. Stačí pouze do listu pravomocí *scope* zadat hodnotu „*offline*“. [24]

Nyní bude popsaný bodový postup s parametry tohoto typu grantu. Tyto body jsou čerpány ze zdroje [24].

1. *Client* zašle POST požadavek na autorizační server s následujícími parametry:
    a. *grant_type:* zde bude řetězec „*refresh_token“.*
    b. *refresh_token:* hodnota, kterou obdržel klient při inicializační autorizaci.
    c. *scope*: jedná se o list pravomocí o které *client* žádá, jenž musí být autorizovaný uživatelem. Pokud není stanoven, tak se použije originální list. Který byl definován při inicializační autorizaci.
2. Autorizační server odpoví s JSON objektem, jenž obsahuje následující parametry:
    a. *token_type*: zde je řetězec *„Bearer“.*
    b. *expires_in*: číselná hodnota, která definuje dobu platnosti tokenu.
    c. *access_token*: nový přístupový token.

d. refresh_token: nový *refresh token*.

5 UCHOVÁVÁNÍ TOKENŮ

Po dokončení některého typu grantu, jsou obdrženy tokeny, které je potřeba bezpečně uchovat. Mezi tyto tokeny patří *access token, refresh token* (pokud byl vyžadován) a *id token* (pokud bylo využité OpenID Connect). Jsou dva způsoby uchovaní těchto tokenů, první je *client-side storage* a druhý způsob je v *server-side session*. [26]

5.1 Client-side storage

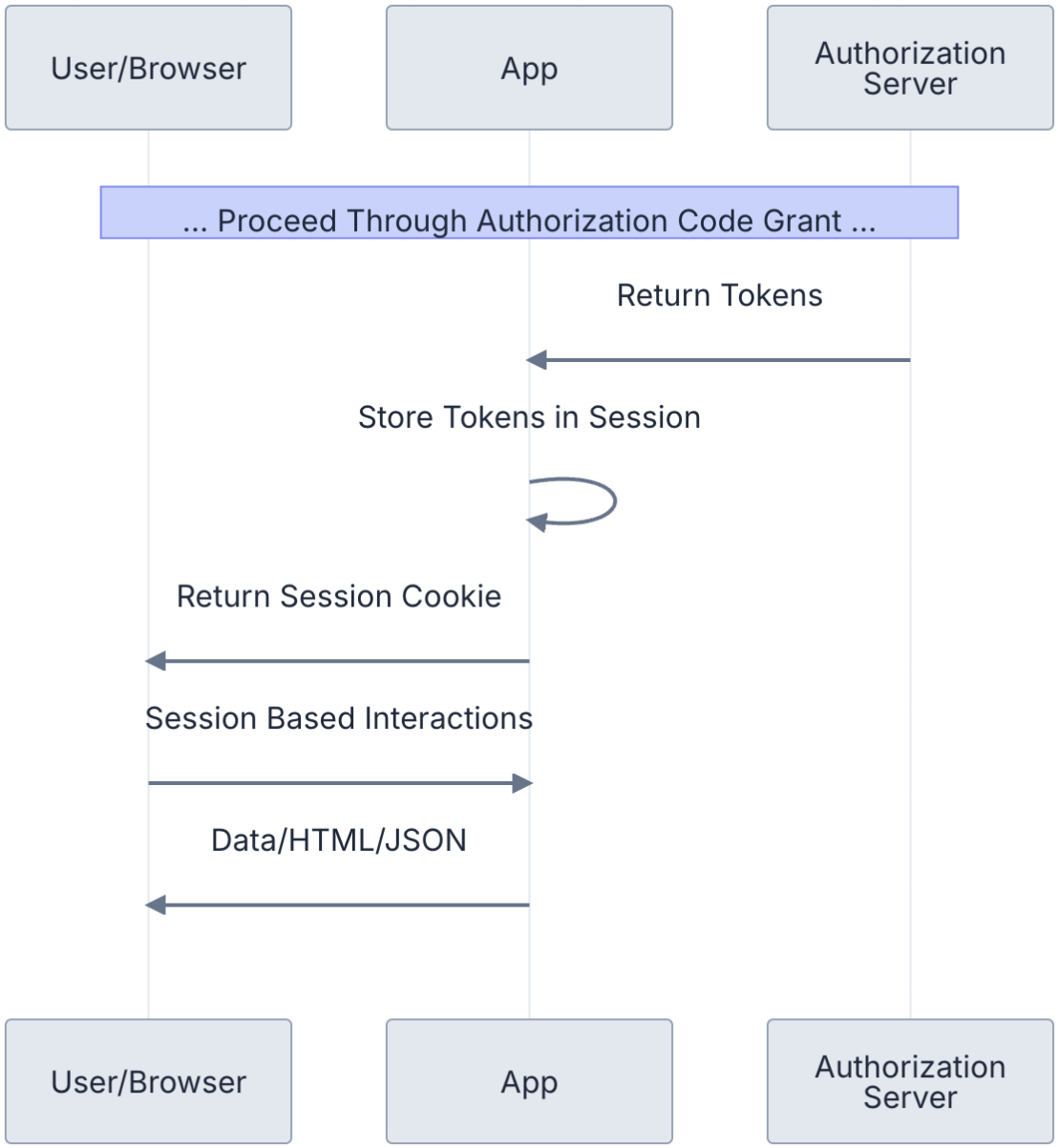
Tento způsob pro uchování *access* a *refresh* tokenů se provádí na mobilní a desktopové aplikaci anebo v prohlížeči. *Client-side storage* je vhodné použít u *microservices* aplikací, kde je několik API uzlů, protože umožňuje jednoduší škálovatelnost. [26]

U mobilních aplikací je potřeba uchovávat tyto tokeny na bezpečném místě, u operačních systému iOS to zajišťuje *Keychain* a operačního systému Android se o to stará *Android internal data.* [26]

V případě prohlížečů je bezpečné použít *HTTPOnly cookies* se *SameSite* nastavenou na hodnotu *„Lax“* anebo *„Strict“*. Tato metoda chrání tokeny před *cross-site scripting* (XSS) útoky, protože *HTTPOnly cookies* nejsou dostupné JavaScript kódu na straně klienta. [26]

5.2 Server-side storage

V této metodě se oba tokeny uchovávají v *server-side session*. Kde následně aplikace na straně klienta komunikuje se serverem pomocí *Session cookie*. Tato metoda je jednodušší na implementaci a je vhodná pro monolitické aplikace. [26]



Obrázek 9: Uchování tokenů v server-side session [26]

5.3 Uchování Id tokenu

Tento token je navrácen v případě použití OIDC. Je možné ho uchovávat v nezabezpečeném uložišti, jako je *localstorage*. Protože není využíván ke přístupu chráněným zdrojům, ale jen nese informace o uživateli (jako třeba jméno uživatele), které jsou zobrazovány v aplikaci. [26]

6 MOŽNOSTI POKROČILÉ ARCHIVACE EMAILŮ

Zmíněné populární mailové aplikace v první kapitole sice disponují funkci archivace, ale nenabízí možnost archivované emaily zašifrovat pomocí kryptografického algoritmu. V této kapitole bude obecně popsaná kryptografie a detailně vhodný algoritmus, pro účely šifrování archívů.

Moderní kryptografie se v rámci šifrování rozděluje do dvou základních kategorií. Toto rozdělení je stanovené počtem používaných klíčů v samotném šifrovacím procesu. První z této kategorie je symetrická kryptografie. Šifry symetrické kryptografie používají pouze jeden klíč pro zašifrování dat. Symetrické šifry jsou rychlé, avšak mají nevýhodu v momentě, kdy je potřeba sdílet zašifrovaná data s někým jiným. Pří sdílení je totiž potřeba i zaslat samotný tajný klíč, který by při předávání mohl být zachycen potencionálním útočníkem.

Do druhé kategorie spadá asymetrická kryptografie. Šifry v této kategorii pracují se dvěma klíči, a to veřejným a privátním klíčem. V případě šifrování dat, se pro zašifrování používá veřejný klíč, který získá odesílatel, jenž bude zasílat zašifrována data. Pro následné dešifrování se použije privátní klíč, který musí být držen v tajnosti. Avšak proces šifrování a dešifrování je pomalý, a tak není vhodný pro velký objem dat.

V praxi se často využívá kombinace obou přístupů, aby se eliminovali jejich nevýhody. Pro rychlé zašifrování velkého objemu dat se používají symetrické šifry. Aby bylo možné bezpečně sdílet symetrický klíč s příjemcem dat, tak se využívá asymetrická kryptografie. Tímto způsobem je symetrický klíč zašifrován veřejným klíčem příjemce a bezpečně odeslán. Příjemce pak použije svůj privátní klíč k dešifrování symetrického klíče, který následně využije k dešifrování dat. Tato metoda kombinuje rychlost symetrické kryptografie s bezpečnostními výhodami asymetrické kryptografie a je známá jako hybridní kryptografie.

Pro šifrovanou archivaci emailů a následném uložení na zařízení uživatele, či v databázi vystačí symetrická kryptografie.

6.1 Advanced Encryption Standard

Advanced Encryption Standard (AES), také známá pod názvem *„Rijndael“,* je považována za jednu z nejbezpečnějších symetrických blokových šifer. Momentálně není známy žádný útok na prolomení této šifry a je četně používaná zejména v TLS a SSL pro zabezpečenou komunikaci pomocí protokolu HTTPS. Díky její popularitě, jsou zahrnovaný AES instrukční sady v moderních procesorech. [27]

AES pracuje s různými délkami klíčů (128 bit – AES-128, 256 bit – AES-256), ale délka klíče neurčuje velikost jednoho bloku. Velikost jednoho bloku je totiž vždy fixní o velikosti 128 bitů. Pro maximalizovanou bezpečnost je doporučeno použít velikost klíče 256 bitů, a to zejména pro budoucí ochranu proti kvantovým počítačům. [27]

AES může pracovat s několika režimy činnosti blokových šifer (CBC, CFB, OFB, CTR, EAX, CCM a GCM) avšak za bezpečné a doporučené jsou považovaný CBC, CTR a GCM. Tyto tři režimy ke své práci potřebují při počátku šifrování a dešifrování náhodný řetězec znaků o stejné délce jako jeden blok (128 bit) zvaný jako *initialization vector* (IV) nebo jako *salt*. CBC vyžaduje přidat do posledního bloku takzvaný *padding* (odsazení), pro jeho zarovnání do stejné velikosti. A při dešifrování zprávy je potřeba toto odsazení odebrat. U režimu CTR není vyžadováno žádné odsazení, a navíc disponuje možností paralelního zpracování. Režim GCM rozšiřuje výhody CTR, a tak navíc disponuje kryptografickou známkou, která ověřuje validitu použitého klíče. [27]

V případě modů CBC a CTR je nutné použít Message authentication code (MAC), pro ověření platnosti použitého klíče při dešifrování. [27]

6.2 MAC & KDF

MAC může využít hash algoritmy, jako je například SHA-256. Tento přístup se označuje jako *hash-based message authentication code* (HMAC). MAC se může pro zachování integrity a autenticity využít v několika variantách: *„Encrypt-then-MAC“, „Encrypt-and-MAC“* a *„MAC-then-Encrypt“.* [27]

Jako kryptografický klíč lze použít i klasické heslo zvolené uživatelem, na které se použije Key Derivation Function (KDF). KDF jsou také vhodné pro bezpečné ukládání uživatelských hesel v databázi. Pro účely zabezpečení hesel jsou doporučené algoritmy Argon2 a Scrypt. V případě převedení hesla do binárního klíče je možné použít i PBKDF2. [27]

7 NERELAČNÍ DATABAZE

Nerelační databáze jsou známe pod pojmem *„Not Only SQL“* (NoSQL). NoSQL databáze jsou flexibilnější a oproti klasickým relačním databázím (SQL) ukládají data v netabulkové formě. Jejich flexibilita tkví ve snadné rozšiřitelnosti a rychlém provádění dotazů. To je docíleno díky tomu, že data nejsou uložena napříč tabulkami. [28]

NoSQL databáze jsou vhodné v případě, kdy je potřeba uložit záznamy s různými datovými typy v nestrukturované podobě. Jejich využití je také výhodné použít v případě, kdy se záznamy budou velmi často měnit. [28]

Typy nerelačních databází: [29]

- **Dokumentové databáze**
  - Tento typ databáze ukládá záznamy ve formátu, který je velice podobný formátu JSON
  - Každý dokument obsahuje páry záznamu ve struktuře „název a hodnota“ např.: „jméno: Erik“
  - Hodnoty těchto páru mohou být všelijakého typu (řetězec, číslo, pole, objekt)
- ***„Key-value“* databáze**
  - Tato typ databáze obsahuje pouze záznamy označené klíčem.
- ***„Wide-column stores“***
  - Zde se záznamy ukládají v tabulkách, řádcích a dynamických sloupcích.
- **Grafové databáze**
  - Zde se záznamy ukládají do uzlů a hrany, jenž tyto uzly spojují, popisují vztahy mezi uzly.

8 POPIS VYBRANÝCH TECHNOLOGIÍ

Tato kapitola bude věnovaná popisu použitých technologií při zpracování programovací části diplomové práce. Zpočátku budou popsaný programovací jazyky, použitý v celém projektu. Následně tato kapitola bude rozdělená do dvou podkapitol. Rozdělení bude zahrnovat klientskou část, kde se nachází frameworky a knihovny pro zhotovení mobilní aplikace. Druhá část obsahuje frameworky, knihovny a služby použité při vývoji backend části aplikace.

8.1 JavaScript a TypeScript

JavaScript (JS) je dynamicky, slabě typovaný a jedno vláknový programovací jazyk, který je nejpoužívanější na světě. Dříve byly webové stránky statické, to se ale změnilo s příchodem JS, což vyústilo ve vytváření dynamických webových stránek. JS totiž dokáže oživit webové stránky, protože je schopny manipulovat s HTML prvky a měnit CSS. Také je schopný reagovat na různé eventy, jako jsou stisknutí myši, změny ve formulářích a na navigaci mezi stránkami. [30]

JS není používaný pouze v rámci webových technologií, jeho využití lze také najít v jiných odvětví vývoje. Například je velmi populární i na straně serveru, kde může být spuštěn za pomocí runtime prostředí Node.js. JS se používá i při vytváření mobilních aplikací, které lze spustit na vícero platformách za pomocí různých frameworků, jako je například React Native. [30]

Největší nevýhodou tohoto jazyka je jeho dynamické typování, a tak se neprovádí kontrola chybných typů proměnných při kompilaci [30]. Tento neduh však řeší TypeScript.

TypeScript (TS) byl vyvinutý v roce 2012 společností Microsoft. TS vyřešil problémy jazyka JS, které nastávaly při rozsáhlých projektech, problémy v modularitě a obtížného debuggování. Tato silně typovaná nadstavba jazyka JS přinesla s sebou definování typů proměnných, *interfaces* a generické typy. TypeScript však nelze spustit ve webovém prohlížeči a ani v runtime Node.js. Proto je v rámci instalace TS do projektu také zavedený kompilátor, který zajišťuje překlad jazyka do JS. [31]

8.2 Klientská část

Tato sekce se bude věnovat představením JavaScript frameworkům a knihoven, které byly použity při implementaci mobilní aplikace.

8.2.1 React Native

React Native (RN) je jedním z nejpopulárnějším JavaScript frameworkem pro vývoj multi-platformních mobilních aplikací. RN je založený na frameworku React, který je zaměřený na tvorbu webových aplikací. Webový vývojáři, se znalostmi frameworku React, mohou jednoduše vyvíjet nativní mobilní aplikace s jednotným kódem pro obě platformy. [32]

Aplikace v RN jsou psané pomocí JSX, jenž je spojením jazyku JavaScript a značkovacího jazyku XML. JSX je následně překládán do nativních UI komponent pomocí takzvaného *bridge*. Ten spouští API pro vykreslení UI komponent v jazyku Objective-C pro platformu iOS anebo v jazyku Java pro platformu Android. *Bridge* také slouží jako oboustranný komunikační kanál mezi jazykem JavaScript a nativním jazykem platformy. Díky této komunikaci je možno přistupovat k nativním funkcím zařízení, jako je fotoaparát zařízení anebo GPS. [32] [33]

Ve frameworku React jsou UI komponenty tvořeny pomocí HTML prvků, jako jsou „*<h1>*“, „*<p>*“ a „* *“. Na rozdíl od toho React Native nepoužívá standardní HTML prvky. Místo toho se využívají speciální komponenty jako „*<View>*“, „*<Text>*“, „*<ScrollView>*“, „*<Image>*“, „*<Button>*“ a „*InputText*“, které jsou optimalizované pro mobilní prostředí a překládány do nativních UI elementů platformy. [33]

8.2.2 Expo

Pro vývoj aplikací pomocí frameworku React Native existují dva postupy. Jedním z nich je použití frameworku Expo a druhý je React Native CLI.

Expo je framework, který ulehčuje vývoj mobilních aplikací ve frameworku React Native. Díky tomuto frameworku mohou vývojáři rychle začít s vývojem, protože automatizuje proces konfigurace a napojení nativních závislostí specifických pro mobilní zařízení. Expo také poskytuje rozsáhlé knihovny pro využití nativních funkcí zařízení, jako jsou notifikace, platby a mapy, což usnadňuje snadnější integraci těchto funkcí do aplikací. Další výhodou je, že se Expo stará o konfiguraci nástrojů pro sestavení JSX kódu do nativního kódu. Pro sestavení a vydání produkční verze aplikace do obchodů s aplikacemi, nabízí Expo

cloudovou službu Expo Application Services (EAS). Kromě toho tento framework také nabízí aplikaci Expo Go, která lze nainstalovat na fyzické zařízení a následně se na ní připojit přes internet. To umožňuje otestovat napsaný kód na fyzických zařízeních, bez nutnosti jeho sestavování na daném zařízení. [34] Využití této možnosti lze ale v případě, kdy nejsou v projektu nakonfigurované nativně závislé knihovny. V tomto případě je nutné aplikaci sestavit a nainstalovat do fyzického zařízení. Tento proces však není omezen pouze na fyzické zařízení, lze vyvíjet a testovat i na virtualizovaných zařízeních v počítači.

Nevýhody u frameworku Expo jsou v podobě omezeného přístupu k některým nativním modulům, které mohou nastat při integraci knihoven třetí strany. Další nevýhoda tkví v omezené konfiguraci pro sestavení aplikace, v porovnání s přístupem pomocí React Native CLI. [34]

React Native CLI je tradiční přístup ve vývoji a nabízí flexibilní konfiguraci celého projektu. To však je za cenu obtížnějšího vývoje, jelikož je potřeba manuálně nastavit nativní závislosti. Taktéž v případě sestavení aplikace do nativního kódu je potřeba konfigurovat manuálně, což může být časově náročné, jelikož se jedná o komplexní proces. [34]

8.2.3 Použité knihovny na straně klienta

V této části budou zmíněný důležité knihovny, použité při vývoji mobilní aplikace na klientské straně.

Axios:

Axios je JavaScript knihovna používaná pro provádění HTTP požadavků v runtime prostředí Node.js anebo *XMLHttpRequests* z prohlížeče. Označuje se jako izomorfní knihovna, což znamená že může se stejným zdrojovým kódem běžet jak na straně serveru, tak i na straně klienta. [35]

Tato knihovna má řadu výhod: [35]

- Automaticky převádí data do formátu JSON v HTML formulářích, a i v odpovědi ze strany serveru.
- Na straně klienta má ochranu proti CSRF útokům.
- Axios obsahuje tzn. *interceptor* požadavků, který umožňuje spustit kód před odesláním HTTP požadavku nebo zpracováním odpověď ze serveru.
- Axios poskytuje mechanismus pro zrušení probíhajících HTTP požadavků.

- Také umožňuje nastavit časový limit pro HTTP požadavek. Pokud server neodpoví v předem definovaném časovém limitu, tak je požadavek zrušený.

Expo FileSystem:

Jedná se o modul poskytovaný v rámci Expo SDK, který umožňuje manipulaci se soubory a daty uloženými na zařízení. Modul také podporuje stahování souborů z internetu a jejich uložení přímo do zařízení. K tomu také umožňuje nahrávat soubory ze souborového systému zařízení na server. [36]

Expo SecureStore:

Expo SecureStore je modul součástí Expo SDK, který poskytuje bezpečný způsob ukládání malých dat. Je tak vhodný pro ukládání autentizačních tokenů, přihlašovacích údajů a dalších citlivých informací na zařízení. Data jsou ukládaný v páru typu klíč-hodnota, kde limit pro hodnotu je 2048 bytů. Tento modul využívá bezpečnostní mechanismy poskytované platformu, jako je *Keychain* služba na iOS a *SharedPreferences* na Androidu. [37]

React Native Date picker:

Jedná se knihovnu třetí strany, která umožňuje snadnou integraci UI komponenty pro výběr data a času. Tato komponenta nabízí širokou řadu vlastní customizace, včetně výběr definování modů pro výběr času, data a kombinaci datumu a času. Modul je podporovaný pro práci s React Native CLI a také s Expo. Přidanou hodnotu této knihovny je vícejazyčná podpora. [38]

React Native Document picker:

Tato knihovna je knihovnou třetí strany, která nabízí implementaci výběru souborů ze zařízení. Knihovna podporuje široký rozsah typů souborů, včetně dokumentů, obrázků, videí a dalších. Jedná se o tzn. *wrapper* okolo platformních funkcí pro výběr souborů, jako je *„UIDocumentPickerViewController“* pro iOS a *„Intent.ACTION_GET_CONTENT“* pro Android. Další funkcionalito je například definování výběru více souborů najednou anebo pouze jednoho. [39]

React Native MMKV:

Jedná se o knihovnu třetí strany, která slouží jako efektivní a výkonné uložiště typu kličhodnota. Je 30krát rychlejší než modul z Expo SDK AsyncStorage, protože je napsaná

v jazyce C++. Vysoká rychlost je docílená i díky použití JSI a C++ TurboModules, namísto tradičního RN *Bridge*. Poskytuje synchronní volání, bez použití *async/await* nebo *Promise*. Navíc umožňuje vytvářet více nezávislých instancí v zašifrované a nezašifrované podobě uložení. Data mohou být ukládána a načítána ve formátech *string, boolean, number a ArrayBuffer*. [40]

React Native Quick Crypto:

Tato knihovna třetí strany je implementací crypto modulu z runtime prostředí Node.js a je navržena pro použití kryptografických operací na mobilním zařízení. Je až 58krát rychlejší než ostatní kryptografické knihovny na bázi JS, protože je napsaná čistě v jazyce C++ a JSI. Bezpečnost je zajištěna použitím nativně kompilované kryptografie, která byla důkladně otestována v JS a C++ prostředí pomocí OpenSSL. Knihovna primárně vznikla pro aplikace zahrnující kryptoměny a peněženky, které vyžadují komplexní kryptografické operace. [41]

React Navigation:

React Navigation je knihovnou třetí strany, sloužící pro řízení navigace mezi obrazovkami. Umožňuje implementaci plynulých a intuitivních přechodu mezi obrazovkami s podporou gesty s prsty. React Navigation podporuje tzn. *deeplink*, který umožňuje přístup na konkrétní obrazovku pomocí externího odkazu. Tato knihovna umožňuje úpravu vzhledu, animací a chování jednotlivých obrazovek. Lze například upravit titulek, tlačítka a styly v záhlaví obrazovky. [42]

Knihovna nabízí tři různé typy navigaci, jako jsou *stack, tab* a *drawer*, které umožňují strukturovat obrazovky do logického pořadí [42]:

- ***Stack Navigation:***
    - V tomto typu navigace jsou jednotlivé obrazovky strukturované do zásobníkové datové struktury, kde nové obrazovky jsou přidávaný na vrchol zásobníku.
    - Obrazovka na vrcholu zásobníku je reprezentována jako aktuálně otevřená.
    - Navigace na původní obrazovku je možné provést tlačítka zpět anebo pomocí gestem.
- ***Tab Navigation:***
    - Poskytuje navigaci mezi různými obrazovkami pomocí záložek umístěných na spodku nebo vrchu obrazovky.
- ***Drawer Navigation:***

- Jedná se o populární navigační vzor ve vývoji mobilních aplikací.
- Zahrnuje skrytý navigační panel, který se vysune z boku obrazovky.
- Jednotlivé položky v tomto panelu pak odkazují na konkrétní obrazovky.

8.3 Serverová část

V této části budou popsané technologie využité při tvorbě serverové části aplikace.

8.3.1 Node.js

Node.js je open source a multiplatformní runtime prostředí, které umožňuje spouštět JavaScript kód mimo webový prohlížeč. Node využívá V8 JavaScript engine, který je napsaný v jazyce C++ a je možné ho spustit na různých operačních systémech, jako je macOS, Linux a Windows. V8 kompiluje JavaScript pomocí procesů just-in-time (JIT), což zrychluje jeho provedení. [43]

Node není vícevláknový, namísto toho běží pouze v jednom procesu. Díky tomu může zpracovat tisíce požadavků bez problému s režiemi více vláken, tak jak to může nastat u jiných vícevláknových modelů. Pro správu požadavků využívá smyčku události, kde se požadavky shromažďují do fronty. Node tyto požadavky postupně zpracovává, bez čekání na odpověď. Také neblokuje vstupně-výstupní operace, oproti ostatním modelům. To je dáno tím, že Node funkce neprovádějí I/O operace přímo. Blokování může nastat v případě některé synchronní operace ze standardní knihovny Node.js, to se však stává zřídka. [43]

Součásti tohoto runtime prostředí je i správce balíčku zvaný npm (Node Package Manager). Npm je největší softwarový registr na světě, umožňující sdílet open source software. Součásti npm je CLI (Command Line Interface), který lze použít pro stahování a instalaci softwaru. Taktéž spravuje závislosti všech knihoven v projektu, které jedním příkazem umožnuje nainstalovat. [44]

8.3.2 Express.js

Express.js je open source, rychlý, flexibilní a minimalistický serverový framework pro runtime prostředí Node.js. Express je navržen, aby byl flexibilní a přizpůsobivý, což znamená že nevyžaduje dodržování striktní architektury aplikace. Dokáže se přizpůsobit různým požadavkům a typům projektu, ať už jde pouze o RESTful API nebo komplexní aplikaci. Díky své jednoduchosti a minimalismu umožňuje Express.js rychle nastavit server, definovat trasy a zpracovávat HTTP požadavky.

Express usnadňuje organizaci aplikace prostřednictvím *middlewares* a *routes* (směrovače). *Middlewares* jsou funkce, které se provádí před zpracováním požadavků. Umožňují zpracovat různé operace, jako je autentizace uživatelů, čtení cookies, správu relací, logování informací a chyb. Směrovače pak zajišťují směrování požadavků na obslužné funkce (*controllers*), podle definování tras. [45]

8.3.3 MongoDB

MongoDB je open source dokumentová databáze, navržená pro uchování obrovského množství dat. Kategorizuje se jakožto NoSQL a podporuje širokou škálu programovacích jazyků, jako jsou C, C++, Java, PHP, Python, JavaScript, Ruby a další. [46]

MongoDB je strukturována následovně [46]:

- Databáze obsahuje kolekce, kde u klasických SQL databázi jsou ekvivalentem tabulek.
- Kolekce pak obsahují dokumenty, kde každý jeden může mít jinou strukturu.
- Dokumenty obsahují záznamy ve formátu klíč-hodnota. Typy hodnot mohou být různé, odpovídající datovým typům, které podporuje formát BSON.
- Dokumenty mohou mít v sobě vnořené další dokumenty, jenž umožňuje vytvářet složité vztahy mezi daty. Oproti klasické SQL databází, by bylo zapotřebí mít záznamy v několika tabulkách a dotazovat je pomocí příkazu *„join“*. Zatímco v MongoDB jsou záznamy v jednom dokumentu, které lze získat jedním jednoduchým dotazem.

Data jsou převáděná z formátu JSON do formátu BSON (Binary representation of JSON) a následně ukládána. Formát BSON tak umožňuje efektivnější dotazování databáze. Jedná se o *schema-less* databází, takže jedna kolekce může obsahovat různé typy dokumentů. Tyto dokumenty pak mohou mít různorodé záznamy a nemusí být navzájem podobné, například různý počet polí, obsah a velikost. Každý záznam v dokumentu má primární a sekundární index, což poskytuje efektivní vyhledávání dat a eliminuje tak potřebu prohledávat všechny dokumenty při každém dotazu. [46]

8.3.4 Použité knihovny na straně serveru

V této části budou popsané důležité knihovny, použité při implementaci serverové části aplikace.

Mongoose:

Mongoose je velice populární knihovnou třetí strany, pro modelování datové struktury a interakci s MongoDB. Umožňuje definovat schéma dokumentu v kódu aplikace, což znamená definování struktury jednotlivých záznamu v kolekci. Jednotlivé položky v definovaném schématu mají určené datové typy, mezi které patří například: pole, řetězec, číslo, datum a další. Dále mohou mít i určený některý typ validace, například: položky jsou povinné, musí být malým písmenem, musí být jedinečné a další. Po definování schématu, je jej potřeba aplikovat do kolekce. To je provádí pomocí modelu, který je pak zodpovědný za interakci s dokumentem pomocí CRUD operací. [47]

Compression

Jedná se o open source knihovnu třetí strany, která nabízí *middleware* umožňující kompresi HTTP odpovědí. Knihovna nabízí dva bezztrátové kompresní algoritmy deflate a gzip. Jsou zde široké možnosti nastavení, jako například míra komprese. Míra komprese se nastavuje pomocí parametru *„level“* s hodnotami od -1 do 9. Hodnota -1 je defaultní komprese, 0 je žádná, 1 je nejrychlejší a 9 je největší komprese. Čím větší je tedy hodnota parametru *„level“*, tím je větší komprese dat, ale zároveň je vyšší náročnost na výpočet. Dalším parametrem je *„memLevel“*, který udává množství alokované paměti pro proces komprese. [48]

Express Mongo Sanitize:

Tato knihovna třetí strany je *middleware* pro framework Express.js, který se stará o sanitaci dat poskytnutých uživatelem. Vyhledává v objektech klíčové hodnoty začínající na “$“ nebo obsahující tečku “.“, které se mohou nacházet v těle, query nebo parametrech požadavku. Po nalezení těchto hodnot mohou být nahrazeny jiným znakem anebo smazány. [49]

Tyto hodnoty jsou rezervované operátory pro MongoDB a bez sanitace dat by mohlo dojít k injekci kódu. Tento kód by pak mohl spustit škodlivý skript, který by změnil kontext databáze. [49]

Imapflow:

Jedná se o open source knihovnu třetí strany pro runtime prostředí Node.js. Hlavním přínosem je v jednoduché implementaci IMAP protokolu. Není potřeba mít dokonalé znalosti o funkčnosti protokolu IMAP, protože nabízí jednoduché API pro jeho implementaci. Nabízí širokou škálu funkci, jako třeba správu emailových schránek a získání jejich informací. Dále umožňuje správu emailových zpráv, jako je odstranění, získání jejich dat, přesunutí do jiné

schránky, přidání vlaječky a další. Autentizaci pro přístup k uživatelským mailovým zprávám nabízí v klasické formě pomocí emailu a hesla. Také umožňuje bezpečný a momentálně žádaný způsob pomocí přístupového tokenu, získaný z autorizačního procesu OAuth 2.0. [50]

Jsonwebtoken:

Jsonwebtoken je knihovna pro Node.js určená k implementaci JSON Web Token (JWT). Knihovna byla vyvinuta podle specifikace *„draft-ietf-oauth-json-web-token-08“* s využitím knihovny *node-jws*. Nabízí funkcionalitu pro podepisování a ověřování tokenů, s podporovou různých kryptografických algoritmů, jako jsou HMAC, RSA a ECDSA. Tokeny mohou obsahovat standardní nároky, jako expirace (*exp*), neplatnost před (*nbf*) a vydáno v (*iat*). Dále umožňuje nastavení specifických atributů, jako je vydavatel (*iss*) nebo publikum (*aud*). [51]

Knihovna nabízí synchronní a asynchronní operace a je vhodná pro autentizační a autorizační procesy, protože umožňuje uchovávání uživatelských informací přímo v tokenech. Redukuje tak potřebu uchovávat tyto informace v relaci na straně serveru. [51]

Nodemailer:

Jedná se o open source knihovnu bez závislostí na jiné knihovny, pro odesílání emailových zpráv pomocí protokolu SMTP. Je kladen důraz na bezpečnost, protože chrání před Remote Code Execution (RCE) útokům. Dále zajišťuje bezpečný přenos emailových zpráv pomocí TLS nebo STARTTLS. Umožňuje odesílat čistě textové nebo HTML zprávy s vloženými obrázky. Také podporuje veškeré znakové sady, včetně emotikonů. Dále knihovna umožňuje přikládání příloh k emailovým zprávám. [52]

Autentizace uživatele pro odesílání emailových zpráv je možné provést pomocí emailové adresy a hesla. [52] Knihovna také ale podporuje autentizaci pomocí přístupového tokenu získaným z autorizačního procesu OAuth 2.0. K tomu také umožňuje obsluhu automatické obnovy přístupového tokenu, při jeho expiraci. [53]

MailParser:

MailParser je knihovnou od stejného vydavatele, jako je Nodemailer. Jedná se o open source knihovnu, určená pro převod emailových zpráv ze surového formátu na čitelný textový formát. Knihovna nabízí dva separátní moduly, MailParser pro nízko úrovňový převod v bytovým formátu a simpleParser pro jednoduchý převod do čitelné podoby v podobě objektu.

Objekt mailové zprávy pak obsahuje informace jako jsou předmět zprávy, odesílatel, adresát, zpráva ve formátu HTML nebo čistý text, přílohy a další. [54]

II. PRAKTICKÁ ČÁST

9 SBĚR POŽADAVKU

Praktická část práce se bude zaobírat návrhem aplikace. Prvním stádiem návrhu aplikace je sběr požadavků. Analýza požadavků před samotným návrhem a vývojem je důležitou části, protože mohou zredukovat počet chyb při vývoji, a tak pomohou dostat produkt do úspěšné konečné fáze vývoje.

Požadavky se rozdělují do dvou kategorií:

- **Funkcionální požadavky:** definují konkrétní funkce dostupné pro uživatele. Mohou obsahovat položky jako datová manipulace, uživatelské operace a funkčnost systému.
- **Nefunkcionální požadavky:** definují chování systému a kritéria která musí splňovat. Mohou zahrnovat kritéria jako bezpečnost, uživatelskou přívětivost a výkon.

9.1 Funkcionální požadavky

FP1. Aplikace umožní autentizaci pomocí tokenů.
FP2. Aplikace umožní integraci různých emailových účtů.
FP3. Aplikace umožní odesílat emailové zprávy.
FP4. Aplikace umožní přijímat a zobrazovat emailové zprávy.
FP5. Aplikace umožní filtrovat emailové zprávy.
FP6. Aplikace umožni spravovat emailové zprávy.
FP7. Aplikace umožní pokročilou archivaci emailových zpráv.
FP8. Aplikace umožní obnovit archivované emailové zprávy.
FP9. Aplikace umožní mít uživatelskou sekci.

9.1.1 Funkcionální požadavky pro autentizaci pomocí tokenů

FP1a. Autentizace bude zahrnovat vydávání přístupových tokenů pro krátkodobý přístup a obnovovacích tokenů pro obnovu přístupových tokenů po jejich expiraci, bez nutnosti nového přihlášení.
FP1b. Aplikace umožní uživateli být autentizovaný současně na více mobilních zařízeních.

9.1.2 Funkcionální požadavky pro integraci emailových účtů

FP2a. Aplikace umožní připojení externích emailových účtu od emailových poskytovatelů Google a Outlook.

FP2b. Aplikace umožní toto připojení za pomocí standardizovaného autorizačního protokolu OAuth 2.0.

FP2c. Aplikace uchová přístupové a obnovovací tokeny externích emailových účtu.

FP2d. Aplikace bude spravovat výměnu exspirovaných přístupových tokenů za nové bez nutnosti nového přihlášení.

9.1.3 Funkcionální požadavky pro odesílání emailových zpráv

FP3a. Aplikace umožní odesílat emailové zprávy pod zvoleným emailovým účtem, prostřednictvím náležitého poskytovatele.

FP3b. Aplikace umožní odeslat emailovou zprávu vícero uživatelům ve formě kopie.

FP3c. Aplikace umožní odeslat emailovou zprávu vícero uživatelům ve formě skryté kopie.

FP3d. Aplikace umožní odesílat přílohy v různých formátech.

9.1.4 Funkcionální požadavky pro přijímání a zobrazování emailových zpráv

FP4a. Aplikace umožní přijímat zprávy pod zvoleným emailovým účtem.

FP4b. Aplikace umožní zobrazit pouze klíčové informace jednotlivých zpráv, pro redukci datového provozu.

FP4c. Aplikace umožní zobrazit pouze omezený počet zpráv pro redukci datového provozu.

FP4d. Aplikace umožní načíst další dávku zpráv, při dosažení konce obrazovky.

FP4e. Aplikace umožní načíst obsah zprávy, včetně příloh při rozkliknutí určeného emailu.

FP4f. Aplikace umožní uložit přílohy do souborového systému zařízení.

9.1.5 Funkcionální požadavky pro filtraci emailových zpráv

FP5a. Aplikace umožní filtrovat emailové zprávy podle předmětu anebo jména nebo emailové adresy odesílatele.

FP5b. Aplikace umožní aplikovat časový filtr v podobě od určitého data po určitý datum anebo filtrovat zprávy z konkrétního dne.

FP5c. Aplikace umožní filtrovat z pouze přečtených, nepřečtených anebo ze všech zpráv.

9.1.6 Funkcionální požadavky pro spravování emailových zpráv

FP6a. Aplikace umožní vybrané emailové zprávy označit jako přečtené nebo nepřečtené.

FP6b. Aplikace umožní vybrané emailové zprávy odstranit.

FP6c. Aplikace umožní vybrané emailové zprávy pokročile archivovat.

9.1.7 Funkcionální požadavky pro pokročilou archivaci

FP7a. Aplikace umožní archívovat emailové zprávy, na základě stanovených filtrovacích kritérií.

FP7b. Aplikace umožní pokročile archivovat emailové zprávy včetně všech příslušných informací a příloh.

FP7c. Aplikace umožní archivovat v nezašifrované a v zašifrované podobě na zařízení.

FP7d. Aplikace umožnit archívy uložené v aplikaci stáhnout do souborového systému zařízení.

FP7e. Aplikace umožní zašifrované archívy uložit do databáze na serveru.

FP7f. Aplikace umožní uživateli zvolit dobu ponechání archivace na serveru.

FP7g. Aplikace umožní uživateli si zvolit svůj šifrovací klíč, pro zašifrovanou archivaci.

FP7h. Aplikace umožní uživateli pojmenovat archív a přidat k němu popisek.

9.1.8 Funkcionální požadavky pro obnovení archivovaných emailových zpráv

FP8a. Aplikace umožní nahrát archívy ze souborového systému.

FP8b. Aplikace umožní obnovit archívy do jejich příslušných mailových složek.

FP8c. Aplikace umožní obnovit veškerá původní data.

9.1.9 Funkcionální požadavky pro uživatelskou sekci

FP9a. Aplikace umožní uživateli vytvořit účet.

FP9b. Aplikace umožní uživateli obnovit heslo, když ho zapomene.

FP9c. Aplikace umožní uživateli změnit uživatelské heslo v nastavení.

FP9d. Aplikace umožní smazat uživatelský účet.

9.2 Nefunkcionální požadavky

NFP1. Aplikace bude částečně funkční bez připojení k internetu.

NFP2. Aplikace bude kompatibilní na platformách iOS a Android.

NFP3. Aplikace bude uživatelsky přívětivá.

NFP4. Aplikace bude mít *token-based* autentizaci.

NFP5. Aplikace bude mít responsivní designe pro rozlišení mobilních zařízení.

NFP6. Aplikace bude optimalizovat datový provoz.

NFP7. Backend část bude implementována v technologii Node.js.

NFP8. Databáze bude zhotovená pomocí nerelační databáze MongoDB.

NFP9. Mobilní aplikace bude implementovaná v technologii React Native.

10 NÁVRH APLIKACE

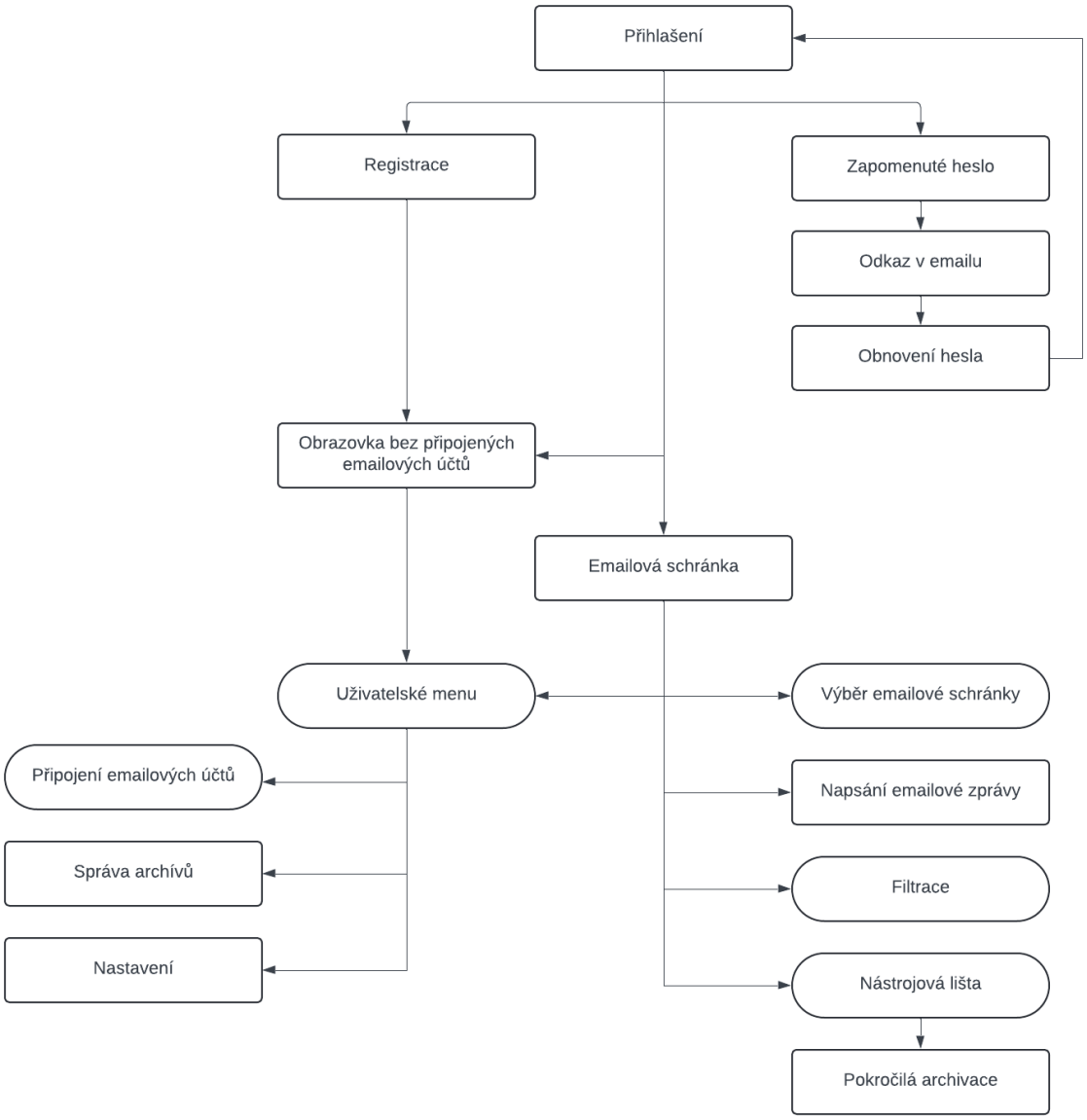
Návrh aplikace je proces přetvoření funkcionálních a nefunkcionálních požadavků do takové formy, aby pomohla vývojářům vytvořit požadovanou aplikaci. Z definovaných požadavků bude v této kapitole sestaven návrh mobilní mailové aplikace.

10.1 Informační architektura

Informační architektura (IA) je nápomocná při vizualizaci infrastruktury a hierarchii aplikace. IA poskytuje designerům a vývojovému týmu lepší perspektivu o produktu ve vývoji. Pomáhá tak při vývoji, přidáváním novým funkcí a úpravám již existujících funkcí. Množství detailů ve schématu IA není definované. Mohou být popsány různé prvky aplikace, jako jsou: navigace, funkce aplikace, chování aplikace a flow aplikace. Množství zahrnutých detailů ve schématu závisí na posudku designera. [55]

Při vytváření IA je důležité se dívat na aplikaci z pohledu samotného uživatele a zamýšlet se, jak se uživatel bude pohybovat mezi obrazovkami při používání aplikace. Takže je potřeba strukturovat informace do logické a lehce pochopitelné struktury, například aby nebylo možné přejít z přihlašovací obrazovky do obrazovky s uživatelským nastavením. [55]

V následujícím obrázku č. 10. je navrhnutá Informační architektura pro mobilní aplikaci MailClient. Obdélníky reprezentují obrazovky aplikace a ovály představují jednotlivé komponenty, ke kterým se uživatel může dostat z určité obrazovky. Tok uživatelské interakce je znázorněný pomocí šipky směřující buďto k obrazovce anebo komponentě.



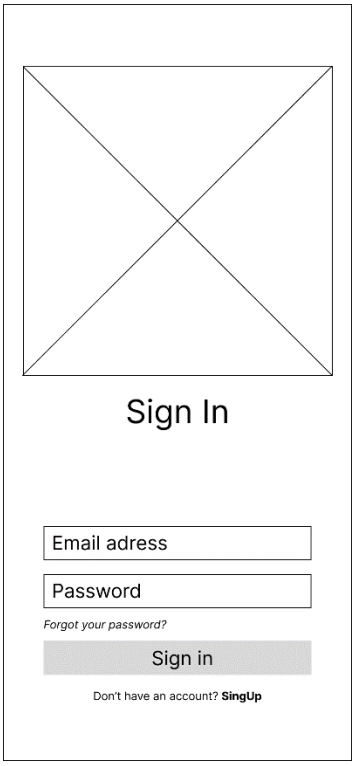
Obrázek 10: Informační architektura

10.2 Drátěný model aplikace

Po navržení informační architektury následuje proces navrhování prvotního designu. Tento prvotní návrh se nazývá „*wireframe*“ (drátěný model) a je základním vizuálním konceptem, který ukazuje rozložení obrazovek a funkční prvky aplikace. Drátěné modely neobsahují žádné designové prvky a jsou typicky černo bíle.

10.2.1 Přihlašovací obrazovka

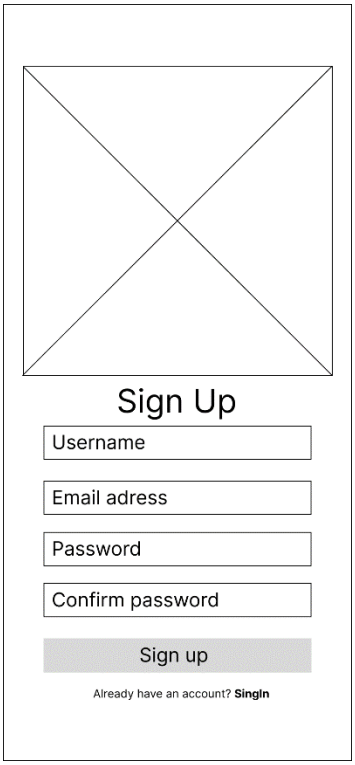
Přihlašovací obrazovka je prvotní obrazovka, která se zobrazí nepřihlášenému uživateli při otevření aplikace. Tato obrazovka obsahuje přihlašovací formulář a dva odkazy pro přesunutí na obrazovku „zapomenuté heslo“ a registrační obrazovku.



Obrázek 11: Wireframe přihlašovací obrazovky

10.2.2 Registrační obrazovka

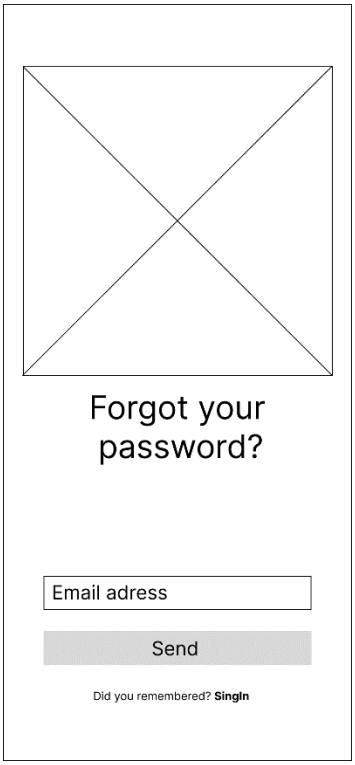
Registrační obrazovka obsahuje formulář pro registraci uživatele. Kromě formuláře obsahuje také odkaz na přesunutí na přihlašovací obrazovku.



Obrázek 12: Wireframe registrační obrazovky

10.2.3 Obrazovka pro zapomenuté heslo

Tato obrazovka obsahuje formulář, pro vyplnění uživatelské emailové adresy. Včetně formuláře, obsahuje i odkaz pro navigaci na přihlašovací obrazovku.



Obrázek 13: Wireframe obrazovky pro zapomenuté heslo

10.2.4 Obrazovka pro resetování hesla

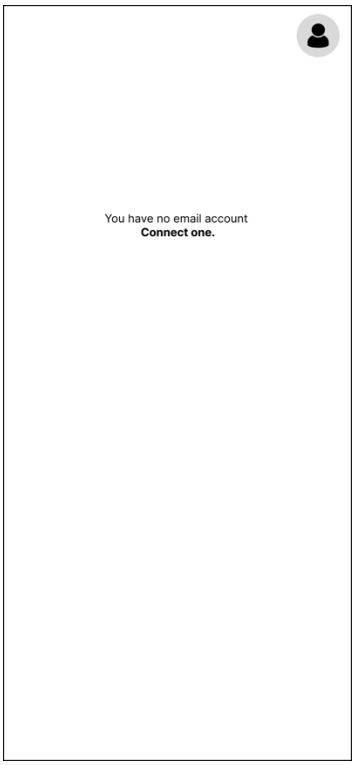
Na tuto obrazovku se uživatel dostane pouze z odkazu, který získá v emailové zprávě, po zaslání požadavku na obnovu hesla.

Obrazovka pro resetování hesla obsahuje odkaz pro navrácení na přihlašovací obrazovku a také formulář, pro nastavení nového hesla.

Obrázek 14: Wireframe obrazovky pro resetování hesla

10.2.5 Obrazovka bez připojeného emailového účtu

Tato obrazovka se zobrazí při úspěšné registraci. Obrazovka se ukáže i v případě, kdy uživatel odhlásí své připojené emailové účty. Uživatel může otevřít uživatelské menu pro připojení emailových účtu dvěma způsoby. Buď pomocí odkazu vybízející připojení emailového účtu, anebo pomocí ikony siluety uživatele v pravém horním rohu.

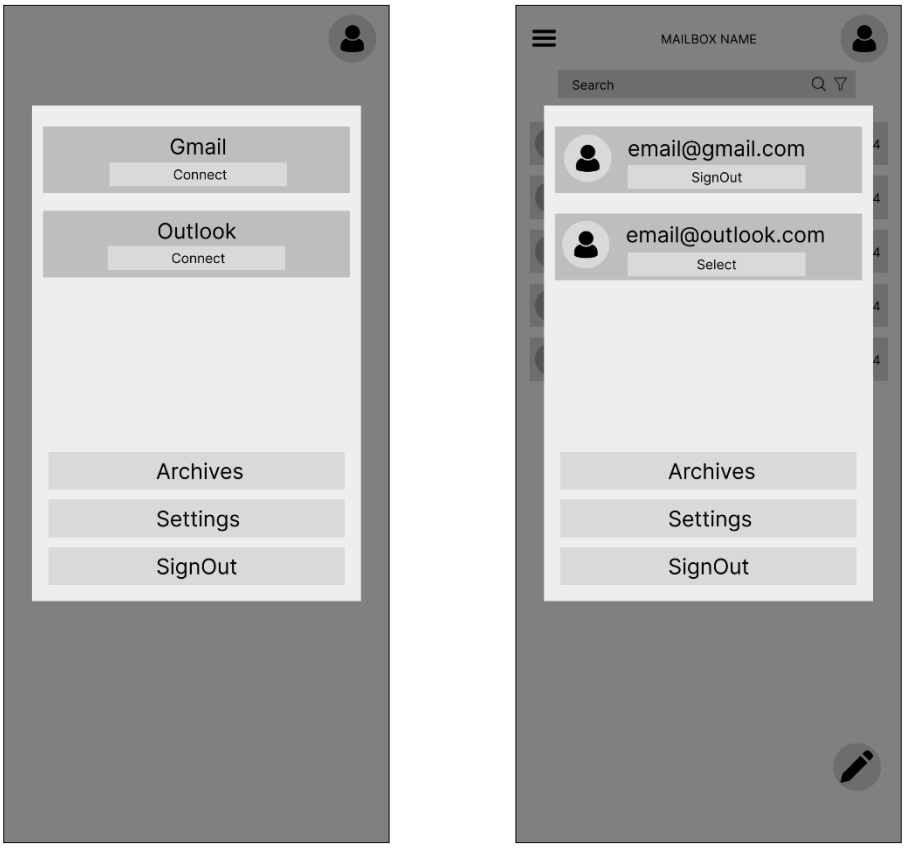


Obrázek 15: Wireframe obrazovky bez připojeného emailového účtu

10.2.6 Uživatelské menu

Uživatelské okno se zobrazí ve formě okénka, známy jako *„modal“*. Zde si uživatel může připojit emailové účty od poskytovatelů Google anebo Azure. Po připojení jednoho z těchto účtu, dojde na pozadí k načtení emailových zpráv a nahrazení obrazovky „bez připojeného účtu“, za obrazovku mailboxu. Včetně toho se také pod konkrétním tlačítkem zobrazí emailová adresa a tlačítko na odhlášení emailového účtu.

Kromě tlačítek pro připojení emailových účtu, jsou ve spodní části i další tři tlačítka. První z nich slouží pro otevření obrazovky pro správu archívů. Druhé tlačítko přesune uživatele na obrazovku s uživatelským nastavením. A poslední tlačítko plní funkci odhlášení se z aplikace.



Obrázek 16: Wireframe uživatelského menu

10.2.7 Obrazovka mailboxu

Při připojení emailového účtu, nebo jeho přepnutí, se zobrazí obsah prvního mailboxu vybraného účtu. Obrazovka se také zobrazí při přihlášení uživatele, s již napojenými emailovými účty. Obrazovka obsahuje v záhlaví tlačítko známé jako *„Hamburger Button“*, jméno otevřeného mailboxu a tlačítko pro otevření uživatelského menu. Po kliknutí na tlačítko *„Hamburger Button“*, vyjede z levé strany obrazovky menu, které obsahuje názvy mailboxů daného emailového účtu. Při zvolení některého z nich, se zobrazí seznam mailů, spjatých s vybraným mailboxem. Pod záhlavím je umístěno vyhledávací pole s tlačítkem, pro otevření pokročilé filtrace.

Hlavním obsahem této obrazovky jsou samotné emailové zprávy, které obsahují pouze základní informace, jako je jméno odesílatele, předmět zprávy a datum obdržení zprávy. Tyto zprávy jsou i obohacené ikonou, která obsahuje první písmeno jména odesílatele. Po kliknutí na některou z těchto zpráv je uživatel přesměrován na obrazovku pro přečtení emailové zprávy.

Poslední komponentou na této obrazovce je takzvaný *„Floating Action Button“*, který se nachází v pravém dolním rohu. Toto tlačítko je určeno pro přesměrování na obrazovku pro poslání emailové zprávy.

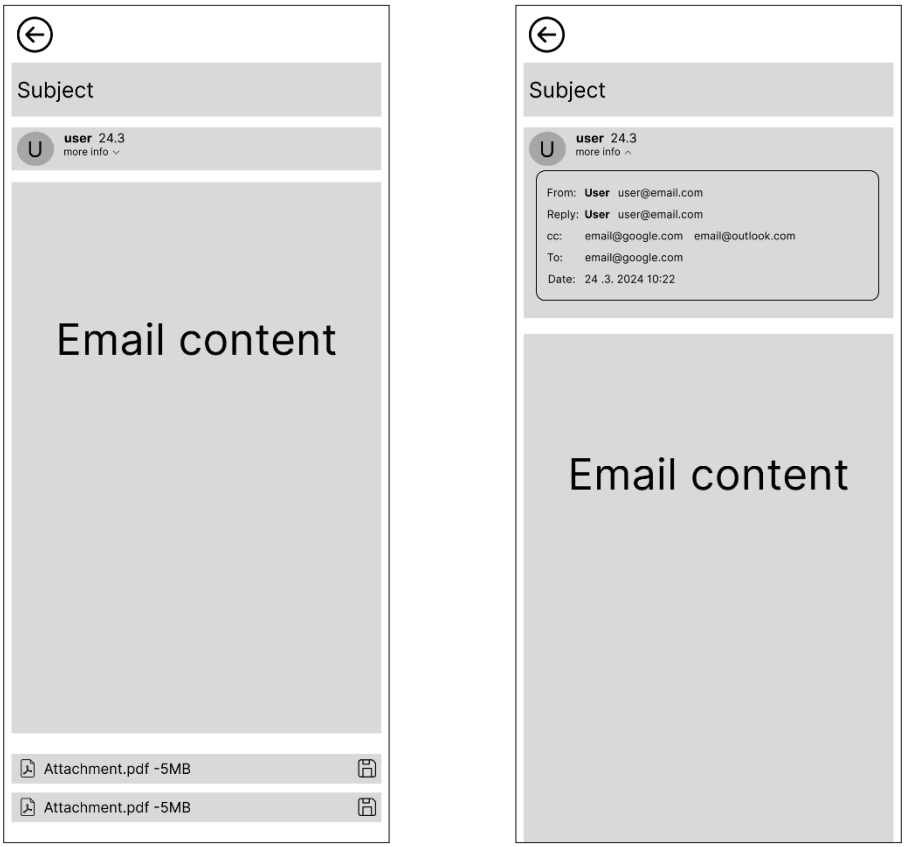
Obrázek 17: Wireframe obrazovky mailboxu

10.2.8 Obrazovka pro přečtení emailové zprávy

Obrazovka pro přečtení emailové zprávy obsahuje v levém záhlaví tlačítko pro navigaci zpět.

Dále se zde nachází předmět zprávy a informace o odesilateli. U informací o odesílateli se nachází i tlačítko, které při kliknutí rozbalí detailnější informace o zprávě. Mezi tyto informace patří jméno a adresa odesílatele, možnost odpovědi, seznam adresátu, kterým přišla kopie, adresa původního adresáta a přesný datum a čas přijetí zprávy.

Poté následuje samotný obsah zprávy a případný seznam příloh. Po kliknutí na položku přílohy, se uživateli zobrazí systémové okno, pro uložení přílohy do souborového systému zařízení.



Obrázek 18: Wireframe obrazovky pro přečtení emailové zprávy

10.2.9 Obrazovka pro odeslání emailové zprávy

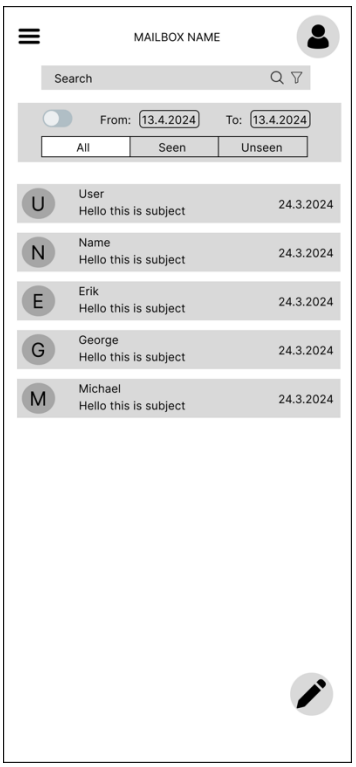
Obrazovka pro odeslání emailové zprávy obsahuje záhlaví s tlačítkem na navigaci zpět do obrazovky mailboxu. Kromě tohoto tlačítka se tam nachází i tlačítko pro odeslání zprávy a nahrání přílohy. Po kliknutí na tlačítko se zobrazí systémové okno, ve kterém lze zvolit přílohy k nahrání. Nahrané přílohy se objeví v zápatí obrazovky, kde je možnost odebrat přílohu pomocí tlačítka křížku.

Zbytek obrazovky již tvoří formulář pro napsání emailové zprávy. Obsahuje vstupní pole pro adresáta zprávy, předvyplněné pole odesílatele, předmět zprávy a samotná zpráva. Jsou tam i speciální vstupní pole pro kopii a skrytou kopii, ve kterých zle přidat více adresátu. Adresáti se přidají napsáním platné emailové adresy a následném stisknutí tlačítka plus. Odebrání adresáta se provede pomocí tlačítka křížku.

Obrázek 19: Wireframe pro odeslání emailové zprávy

10.2.10 Filtrace

Filtrace se nachází na obrazovce mailboxu. Její součástí je vyhledávací pole, které filtruje na základě jména a emailové adresy odesílatele a předmětu zprávy. Vedle tlačítka „lupy“ pro vyhledání, je i tlačítko „filtrace“ jenž je určeno pro rozbalení pokročilejšího vyhledávání. Jednou z kritérií pro filtraci je datum. Lze vyhledávat v datovém rozpětí od a do. V případě konkrétního dne, stačí nastavit obě pole na stejný datum. Při klinutí na pole datumu, se otevře systémový kalendář. Dalším filtračním kritériem je vybrání „všechny“, „přečtené“ a „nepřečtené“ zprávy.



Obrázek 20: Wireframe pokročilé filtrace

10.2.11 Panel nástrojů

Panel nástrojů se zobrazí při označení zprávy. Zpráva se označí podržením položky, kde následným klikáním na ostatní položky, se provádí další označování. Panel nástrojů obsahuje tlačítko zrušení selekce a indikátor množství zvolených položek. Na pravé straně ne nachází ikona obálky, která slouží pro označení vybraných zpráv jako „přečtené“ anebo „nepřečtené“. Pokud je jedna jediná položka nepřečtená, tak je možné celý výběr označit pouze jako „přečtený“. Druhé tlačítko „archivace“ slouží pro přesunutí na obrazovku „pokročilá archivace“, kde je možné s označenými emailovými zprávami provádět operace pokročilé archivace. Posledním tlačítkem je tlačítko „odstranění“, kde se po jeho kliknutí zobrazí potvrzovací formulář. Po potvrzení odstranění jsou označené zprávy trvalé odstraněny z mailboxu.

Veškeré tyto funkce z panelu nástrojů, zle provádět nad vyfiltrovanými zprávami z filtrace popsané v předchozí části.

Obrázek 21: Wireframe panelu nástrojů

10.2.12 Obrazovka pokročilé archivace

V záhlaví obrazovky se nachází pouze tlačítko pro zpětnou navigaci do obrazovky mailboxu. Nejdůležitější části je zde volba nastavení pro pokročilou archivaci.

Uživatel má možnost archivaci uložit lokálně do zařízení, a to v šifrované anebo v nezašifrované podobě. Po stisknutí tlačítka „archivace“ je uživateli zobrazeno systémové okno, pro uložení archívu do souborového systému zařízení.

Druhou možností je archív uchovat v databázi na serveru. Tlačítko pro vybrání „Server“ se odemkne v případě, kdy uživatel zapne přepínač „Encrypted“. Zároveň se i odemkne vstupní pole „Secret“, kde uživatel musí zadat tajné heslo pro zašifrování archívu. Při vybrání serverovém uchování archívu se rozbalí i přepínač pro zapnutí délky uložení archívu na serveru. Při kliknutí na pole „datum“ se uživateli zobrazí systémový kalendář.

Pro oba způsoby uložení archívu lze přidat jméno a popisek pro dodatečné informace. Vyplnění těchto polí není povinné

Obrázek 22: Wireframe obrazovky pokročilé archivace

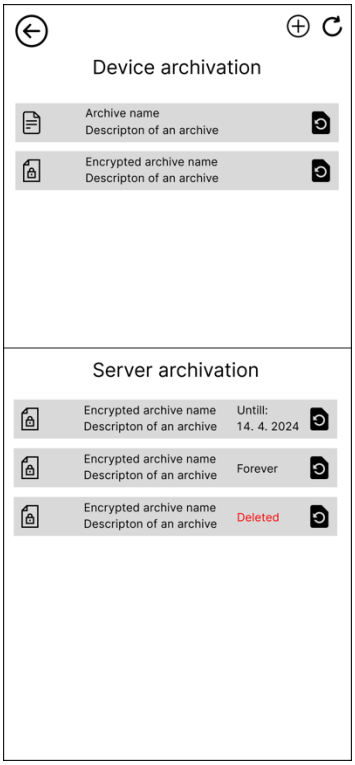
10.2.13 Obrazovka pro obnovu archívu

V záhlaví se nachází tlačítko pro navrácení do obrazovky mailboxu. V pravé části záhlaví jsou dvě tlačítka, první obsluhuje načtení archívů ze souborového systému zařízení, druhé tlačítko slouží pro získání archívů uložených na serveru v databázi.

Obrazovka je rozdělená do dvou částí. V první polovině se nachází archívy uložené v aplikaci. V levé části položky archívu se vyskytují ikony „soubor“ anebo „zamčený soubor“. Ikona souboru signalizuje, že je archív nezašifrovaný. Na druhou stranu ikona zamčeného souboru označuje archív jako zašifrovaný. V prostřední části položky jsou informace, jako je název a popis archívu, jenž byl definován uživatel při procesu archivace. Ikona „obnova souboru“ pak slouží pro samotný proces obnovy archívu. V případě zašifrovaného archívu, stisknutím tlačítka obnovy souboru je uživateli zobrazen formulář zadání hesla pro dešifrování.

V druhé částí se nachází pouze zašifrované archívy uložené na straně serveru. Zde má každá položka ještě dodatečné informace rozdělené do tří stavů: „Forever“ jenž označuje uložený archiv bez časového omezení, „Until:“ který udává do kdy bude archív uchován na serveru a poslední stav „Deleted“ jenž označuje archív jako odstraněný ze serveru.

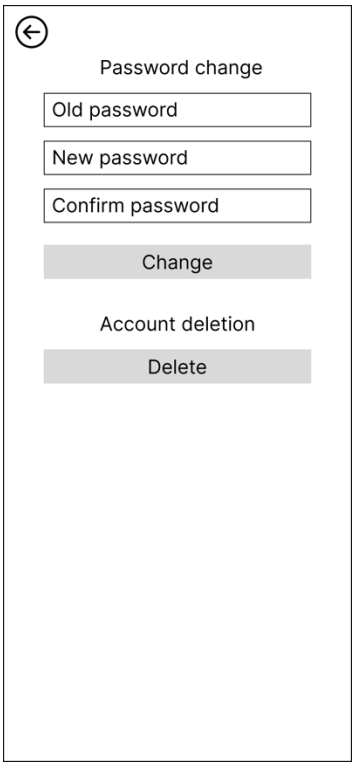
Tyto archívy lze kdykoliv obnovit bez ohledu na stav, ve kterém se nachází.



Obrázek 23: Wireframe obrazovky pro obnovu archívu

10.2.14 Obrazovka uživatelského nastavení

Jak u ostatních obrazovek, tak i záhlaví obsahuje pouze tlačítko zpět. Dále se zde nachází formulář pro změnu hesla k uživatelskému účtu. Poslední sekcí je zde odstranění účtu, po jeho kliknutí se zobrazí upozorňovací okno, pro potvrzení anebo zrušení akce pro smazání.



Obrázek 24: Wireframe obrazovky uživatelského nastavení

10.3 Návrh datového modelu

V této části bude navrhnuta struktura nerelační databáze. Databáze bude sloužit pouze pro uchování uživatelů a archívů. Níže budou popsané kolekce, které jsou zhotovené v návrhu datového modelu v obrázku č. 25.

Kolekce **users:**

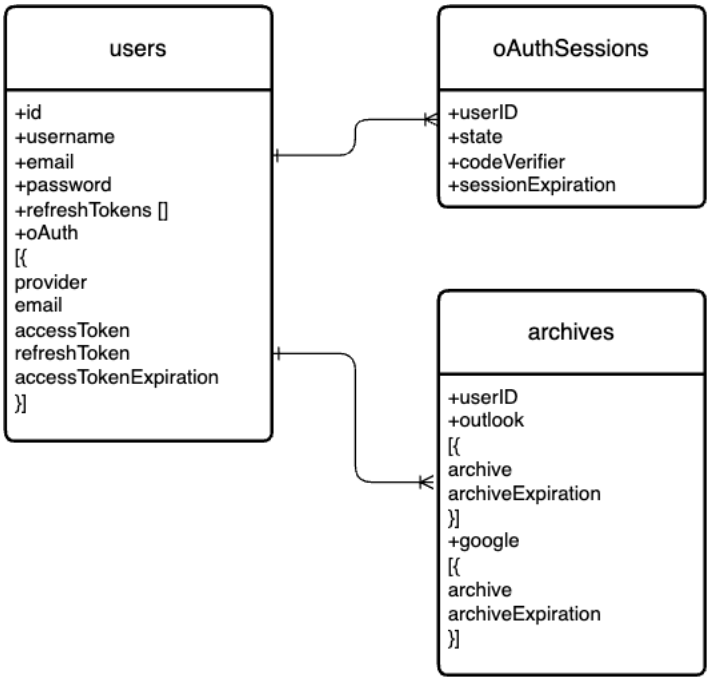
- **Popis:** kolekce sloužící pro ukládání dat o uživateli.
- **id:** unikátní identifikátor dokumentu (identifikátor uživatele).
- **username:** uživatelské jméno.
- **email:** uživatelský email.
- **password:** hash hesla.
- **refreshTokens:** pole obnovovacích tokenů.
- **oatuh:**
    - **provider:** název poskytovatele.
    - **email:** uživatelský email daného providera.
    - **accessToken:** přístupový token poskytovatele.
    - **refreshToken:** obnovovací token poskytovatele.
    - **accessTokenExpiration:** doba expirace přístupového tokenu.

Kolekce **oAuthSessions**:

- **Popis**: dočasná kolekce sloužící pro uložení relace při autorizačním procesu za využití protokolu oAuth 2.0.
- **userID:** identifikátor uživatele, který započal autorizační proces.
- **state:** CSRF token.
- **codeVerifier:** řetěz náhodných znaků (tajný klíč).
- **sessionExpiration:** časová doba, po které se dokument smaže (vypršení relace).

Kolekce **archives**:

- **Popis:** kolekce sloužící. Pro uchování archivací na straně serveru.
- **userID:** identifikátor uživatele.
- **outlook:** pole objektů pro poskytovatele Outlook, kde objekt obsahuje archívy společně s expiračním datem pro uchování (pokud si uživatel nastavil časové omezení).
- **google:** pole objektu s archívy pro poskytovatele Google.



Obrázek 25: Návrh datového modelu

10.4 Komunikační část

Před návrhem komunikačního rozhraní, je potřeba si definovat protokol využívávaný pro komunikaci. Je také důležité si definovat formu autentizace uživatele při přístupu na chráněné endpointy.

- **Protokol**:
  - RESTful API s komunikací přes HTTPS pro zajištění bezpečnosti.
- **Autentizace**:
  - Token-based autentizace (JWT) pro ověření uživatele při každém requestu.
- **Formát dat**:
  - Výměna dat ve formátu JSON.

10.5 Optimalizace datového provozu

Při návrhu komunikační části mezi aplikací a serverem, je potřeba i myslet na optimalizaci datového provozu. To je důležité zejména pro maximalizaci reakční rychlosti aplikace, což má pozitivní dopad na uživatelskou zkušenost (UX).

- **Komprese dat:** Data budou komprimována při přenosu mezi serverem a aplikací, což vede k rychlejšímu načítání a snížení spotřeby dat.

- **„Lazy-loading“ pro emailové zprávy a přílohy:** Tato technika zajistí, že se načítají pouze data potřebná pro zobrazení aktuálně vyžadovaného obsahu, čímž se snižuje spotřeba dat a zlepšuje se reakční rychlost aplikace.
- **„Infinite scrolling“ pro náhledy emailů:** Při posunu na konec sezamu emailů se automaticky načtou další dávky náhledových informací o emailových zprávách, jenž sníží množství načítaných dat a zlepší uživatelskou zkušenost.
- **„Caching“:** Ukládání emailových dat do lokální paměti zařízení umožní rychlejší přístup k emailovým zprávám a sníží potřebu opětovného načítání dat od poskytovatele. K tomu tato metoda ještě umožní částečnou funkčnost aplikace v offline režimu, kdy bude možné přistupovat k již načteným emailovým zprávám bez připojení k internetu.

10.6 Definice rozhraní

V této části budou navrhnuty API „*endpoints*“ (koncové body) pro komunikaci mezi mobilním mailovým klientem a backend částí.

10.6.1 Rozhraní pro autentizaci

V rámci tohoto rozhraní jsou API koncové body spjaté s autentizaci. Mezi ně se řadí obnova přístupového tokenu, registrace, přihlášení a odhlášení uživatele.

Endpoint pro registraci uživatele:

- **URL**: POST /auth/register
- **Popis**: Endpoint sloužící pro registraci nového uživatele.
- **Parametry**:
  - JSON: s uživatelským jménem, heslem, potvrzením hesla a emailovou adresou.
- **Návratová hodnota**:
  - JSON: přístupový a obnovovací token.

Endpoint pro přihlášení uživatele:

- **URL**: POST /auth/login
- **Popis**: Endpoint sloužící pro přihlášení uživatele.
- **Parametry**:
  - JSON: uživatelské jméno a heslo.

- **Návratová hodnota**:
    - JSON: přístupový a obnovovací token.

Endpoint pro obnovu přístupového tokenu:

- **URL**: POST /auth/refresh
- **Popis**: Endpoint sloužící pro obnovu přístupového tokenu.
- **Parametry**:
    - JSON: obnovovací token.
- **Návratová hodnota**:
    - JSON: přístupový a obnovovací token.

Endpoint pro odhlášení uživatele:

- **URL**: POST /auth/logout
- **Popis**: Endpoint sloužící pro odhlášení uživatele.
- **Parametry**:
    - JSON: obnovovací token.
- **Návratová hodnota**:
    - Potvrzení odhlášení.

10.6.2 Rozhraní pro správu hesel

Tato rozhraní budou sloužit pro proces zapomenutého hesla.

Endpoint pro požadavek na zapomenuté heslo:

- **URL**: POST /password/forgotPassword
- **Popis**: Endpoint sloužící pro započetí procesu resetování hesla.
- **Parametry**:
    - JSON: uživatelský email.
- **Návratová hodnota**:
    - Potvrzení o odeslání odkazu na zadanou emailovou adresu.

Endpoint pro přesměrování do aplikace:

- **URL**: GET /password/deeplink/:token
- **Popis**: Endpoint sloužící pro přesměrování do aplikace.
- **Parametry**:
    - Query: token pro resetování hesla.

- **Návratová hodnota**:
    - Deeplink do mobilní aplikace.

Endpoint pro požadavek na resetování hesla:

- **URL**: POST /password/resetPassword
- **Popis**: Endpoint sloužící pro nastavení nového uživatelského hesla.
- **Parametry**:
    - JSON: heslo a potvrzení hesla.
- **Návratová hodnota**:
    - Potvrzení o resetování hesla.

10.6.3 Rozhraní pro protokol OAuth 2.0

Zde jsou navrhnuté koncové body pro účely připojení emailového účtu třetí strany.

Endpoint pro požadavek na připojení emailové účtu:

- **URL**: GET /oauth/:loginProvider/login
- **Popis**: Endpoint sloužící pro připojení emailového účtu daného poskytovatele.
- **Parametry**:
    - Query: název emailového poskytovatele.
    - Header: přístupový token.
- **Návratová hodnota**:
    - Autorizační URL pro přihlášení a autorizování u daného poskytovatele.

Endpoint pro požadavek na zpracování redirect URI:

- **URL**: GET /oauth/:loginProvider/callback
- **Popis**: Endpoint sloužící pro zpracování redirect URI zaslaným od poskytovatele.
- **Parametry**:
    - Query: *„state“* a *„code verifier“*
- **Návratová hodnota**:
    - Redirect URL do aplikace s query parametry: emailová adresa připojeného účtu a název poskytovatele.

Endpoint pro požadavek na odhlášení emailového účtu:

- **URL**: POST /oauth/:loginProvider/logout
- **Popis**: Endpoint sloužící pro odhlášení připojeného emailového účtu.

- **Parametry**:
    - Query: název emailového poskytovatele.
    - Header: přístupový token.
- **Návratová hodnota**:
    - Potvrzení o úspěšném odhlášení.

10.6.4 Rozhraní pro SMTP protokol

Zde je navrhnut koncový bod pro odesílání emailové zprávy za pomocí protokolu SMTP.

Endpoint pro požadavek na poslání emailové zprávy:

- **URL**: POST /smtp/:loginProvider/sendEmail
- **Popis**: Endpoint sloužící pro poslání emailové zprávy.
- **Parametry**:
    - Query: název emailového poskytovatele.
    - Header: přístupový token.
    - JSON: adresát, předmět zprávy, adresáti pro kopii a skrytou kopii, zpráva a přílohy.
- **Návratová hodnota**:
    - Potvrzení o úspěšném odeslání.

10.6.5 Rozhraní pro protokol IMAP

Součástí tohoto navrženého rozhraní jsou veškeré koncové body spjaté s protokolem IMAP.

Endpoint pro požadavek na získání mailboxů:

- **URL**: GET /imap/:loginProvider/mailBoxes
- **Popis**: Endpoint sloužící pro získání seznam mailboxů zvoleného emailového účtu.
- **Parametry**:
    - Query: název emailového poskytovatele.
    - Header: přístupový token.
- **Návratová hodnota**:
    - JSON: seznam mailboxů.

Endpoint pro požadavek na získání seznam mailů:

- **URL**: POST /imap/:loginProvider/mailFromBox

- **Popis**: Endpoint sloužící pro získání seznamu náhledových informací ze zvoleného mailboxu.
- **Parametry**:
    - Query: název emailového poskytovatele.
    - Header: přístupový token.
    - JSON: jméno mailboxu, počet mailů, počet již načtených mailů a celkový počet mailů v mailboxu.
- **Návratová hodnota**:
    - JSON: seznam mailů.

Endpoint pro požadavek na získání aktuálního stavu mailboxu:

- **URL**: POST /imap/:loginProvider/mailBoxStatus
- **Popis**: Endpoint sloužící pro získání aktuálního stavu zvoleného mailboxu.
- **Parametry**:
    - Query: název emailového poskytovatele.
    - Header: přístupový token.
    - JSON: jméno mailboxu.
- **Návratová hodnota**: Status mailboxu.

Endpoint pro požadavek na získání detailu mailu:

- **URL**: POST /imap/:loginProvider/mailDetail
- **Popis**: Endpoint sloužící pro získání celé emailové zprávy včetně příloh.
- **Parametry**:
    - Query: název emailového poskytovatele.
    - Header: přístupový token.
    - JSON: jméno mailoboxu a id zprávy.
- **Návratová hodnota**:
    - JSON: Detail emailové zprávy.

Endpoint pro požadavek na smazání emailové zprávy:

- **URL**: POST /imap/:loginProvider/mailDelete
- **Popis**: Endpoint sloužící pro smazání jedné nebo více emailové zprávy.
- **Parametry**:
    - Query: název emailového poskytovatele.

  - Header: přístupový token.
  - JSON: jméno mailoboxu, seznam id emailů.
- **Návratová hodnota**:
  - Potvrzení o smazání.

Endpoint pro požadavek na označení zprávy vlaječkou:

- **URL**: POST /imap/:loginProvider/setFlag
- **Popis**: Endpoint sloužící pro nastavení vlaječky na jedné nebo více emailové zprávy.
- **Parametry**:
  - Query: název emailového poskytovatele.
  - Header: přístupový token.
  - JSON: jméno mailoboxu, seznam id emailů, vlaječka
- **Návratová hodnota**:
  - Potvrzení o označení.

Endpoint pro požadavek na získání surových emailových dat:

- **URL**: POST /imap/:loginProvider/getRaw
- **Popis**: Endpoint sloužící pro získání surových emailových dat, které budou komprimované pro účely pokročilé archivace.
- **Parametry**:
  - Query: název emailového poskytovatele.
  - Header: přístupový token.
  - JSON: jméno mailoboxu, seznam id emailů.
- **Návratová hodnota**:
  - Komprimovaná surová emailová data.

Endpoint pro požadavek na obnovu archivovaných emailových zpráv:

- **URL**: POST /imap/:loginProvider/restoreArchive
- **Popis**: Endpoint sloužící pro obnovu emailových zpráv, které byly archivované.
- **Parametry**:
  - Query: název emailového poskytovatele.
  - Header: přístupový token.
  - JSON: komprimovaná data.
- **Návratová hodnota**:

- Potvrzení o úspěšné obnově.

Endpoint pro požadavek na filtraci emailových zpráv:

- **URL**: POST /imap/:loginProvider/filteredMails
- **Popis**: Endpoint sloužící pro filtraci emailových zpráv, dle parametrů pro filtraci.
- **Parametry**:
  - Query: název emailového poskytovatele.
  - Header: přístupový token.
  - JSON: jméno mailoboxu, adresa odesílatele, název předmětu zprávy, časové rozpětí, druh zpráv (přečtené, nepřečtené nebo všechny).
- **Návratová hodnota**:
  - JSON: seznam emailových zpráv.

10.6.6 Rozhraní pro archivaci

Součásti tohoto rozhraní jsou koncové body pro účely archivace emailových zpráv.

Endpoint pro požadavek na nahrání archívu do databáze:

- **URL**: POST /archive/:loginProvider/upload
- **Popis**: Endpoint sloužící pro nahrání zašifrovaných archívů.
- **Parametry**:
  - Query: název emailového poskytovatele.
  - Header: přístupový token.
  - JSON: komprimovaná data.
- **Návratová hodnota**:
  - Potvrzení o úspěšném nahrání.

Endpoint pro požadavek na získání uložených archívů v databázi:

- **URL**: POST /archive/:loginProvider/getArchives
- **Popis**: Endpoint sloužící pro získání uložených zašifrovaných archívů z databáze.
- **Parametry**:
  - Query: název emailového poskytovatele.
  - Header: přístupový token.
- **Návratová hodnota**:
  - JSON: seznam zašifrovaných archívů.

10.6.7 Rozhraní pro uživatelské nastavení

Zde jsou navržené koncové body pro uživatelské nastavení.

Endpoint pro požadavek na změnění uživatelského hesla:

- **URL**: POST /user/changePswd
- **Popis**: Endpoint sloužící pro změnění uživatelského hesla.
- **Parametry**:
    - Header: přístupový token.
    - JSON: staré heslo, nové heslo, potvrzení nového hesla.
- **Návratová hodnota**:

Endpoint pro požadavek na odstranění uživatelského účtu:

- **URL**: POST /user/deleteAccount
- **Popis**: Endpoint sloužící pro odstranění uživatelského účtu.
- **Parametry**:
    - Header: přístupový token.
- **Návratová hodnota**:
    - Potvrzení o smazaní účtu.

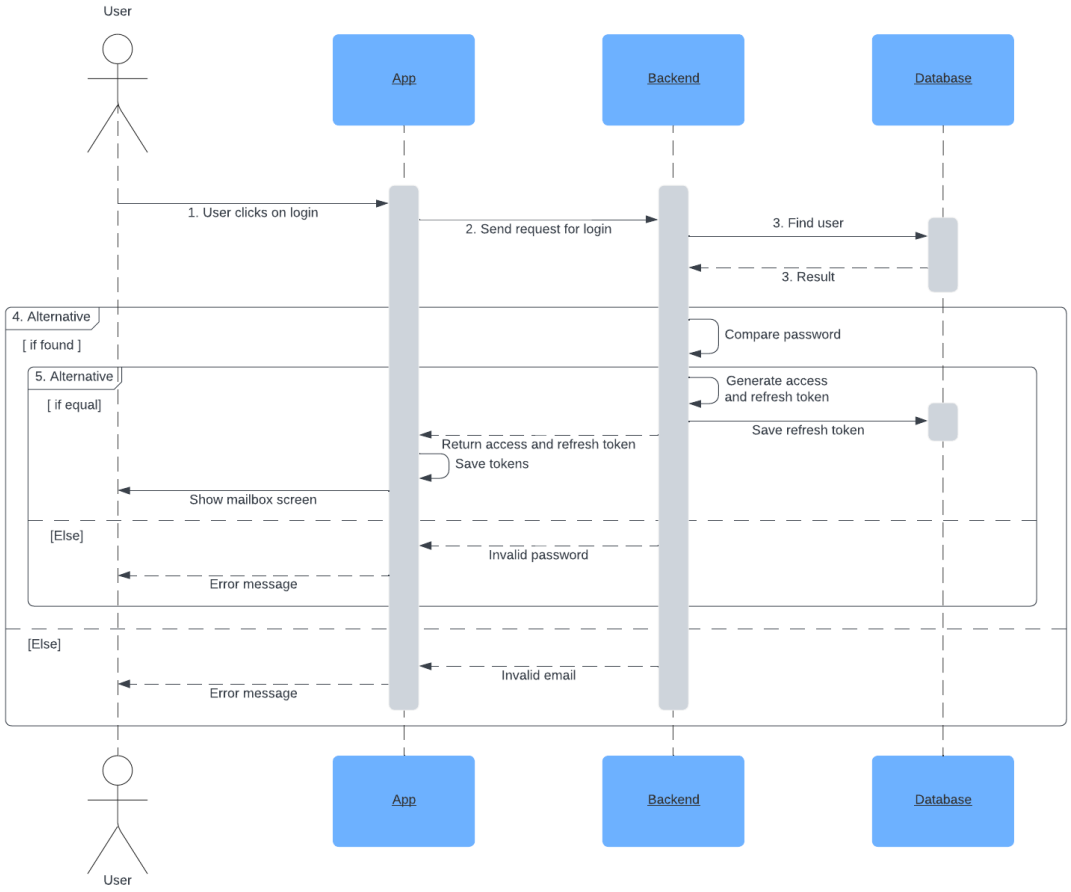
10.7 Sekvenční diagramy důležitých částí aplikace

V této části návrhu aplikace budou zhotoveny sekvenční diagramy pro vybrané důležité částí aplikace. Tyto diagramy umožňují lépe pochopit interakci mezi různými objekty a systémovými komnatami během provádění specifických funkcí. Toto pochopení pak usnadňuje vývoj a testování aplikace.

10.7.1 Proces přihlášení uživatele

Sekvenční diagram popisují proces přihlášení uživatele je prezentovaný na obrázku č. 26. Správný formát emailové adresy a kritéria pro formát hesla (minimální počet znaků, velké písmeno, speciální znak) budou kontrolovány v aplikaci a také na serveru. Z důvodu zredukování komplexnosti sekvenčního diagramu, nebude tato validace součásti diagramu. Níže jsou detailněji rozebrány jednotlivé kroky.

1. Uživatel po vyplnění emailové adresy a hesla stiskne tlačítko pro přihlášení.
2. Aplikace pošle požadavek na backend.
3. Backend provede dotaz na databázi s účelem nalezení uživatele.
4. Fragment s podmínkou:
    a. Pokud je uživatel nalezen:
       Backend provede hash hesla pomocí KDF.
    b. Jinak:
       Backend zašle chybu v podobě chybné emailové adresy
       Aplikace zobrazí chybu uživateli.
5. Vnořený Fragment s podmínkou:
    a. Pokud se hash hesla rovná s hodnotou hesla uloženou v databázi:
       Backend vygeneruje přístupový a obnovovací token.
       Backend uloží obnovovací token v databázi.
       Backend zašle odpověď s tokeny.
       Aplikace uloží tokeny do šifrovaného uložiště.
       Aplikace přesměruje uživatele do obrazovky mailboxu.
    b. Jinak: Backend zašle chybu v podobě chybného hesla.
       Aplikace zobrazí chybu uživateli.



Obrázek 26: Proces přihlášení uživatele

10.7.2 Proces autentizace uživatele

Autentizace je klíčová vrstva ochrany, která zajišťuje že pouze oprávnění uživatelé mají přístup k citlivým informacím a funkcím systému. Tato ochrana spočívá v kontrole přístupového tokenu. Kontrola se provádí před každým zpracováním požadavku na chráněné endpointy. Při expiraci přístupového tokenu je potřeba vyžádat nový token. Tento proces popisuje sekvenční diagram na obrázku č. 27. Bodový popis tohoto procesu je popsán níže.

1. Uživatel klikne na tlačítko, které reprezentuje požadavek na chráněný endpoint.
2. Aplikace zašle požadavek společně s přístupovým tokenem.
3. Backend provede validaci přístupového tokenu.
4. Fragment s podmínkou:

   a. Pokud je přístupový token validní (nebyl upraven nebo není exspirovaný): Backend zpracuje požadavek a odešle odpověď.
   b. Jinak:

      Backend pošle jako odpověď status kód 401.

      Aplikace jako odpověď na tento kód pošle požadavek na obnovu přístupového tokenu na endpoint „/auth/refresh“. Kde server provede validaci obnovovacího tokenu.

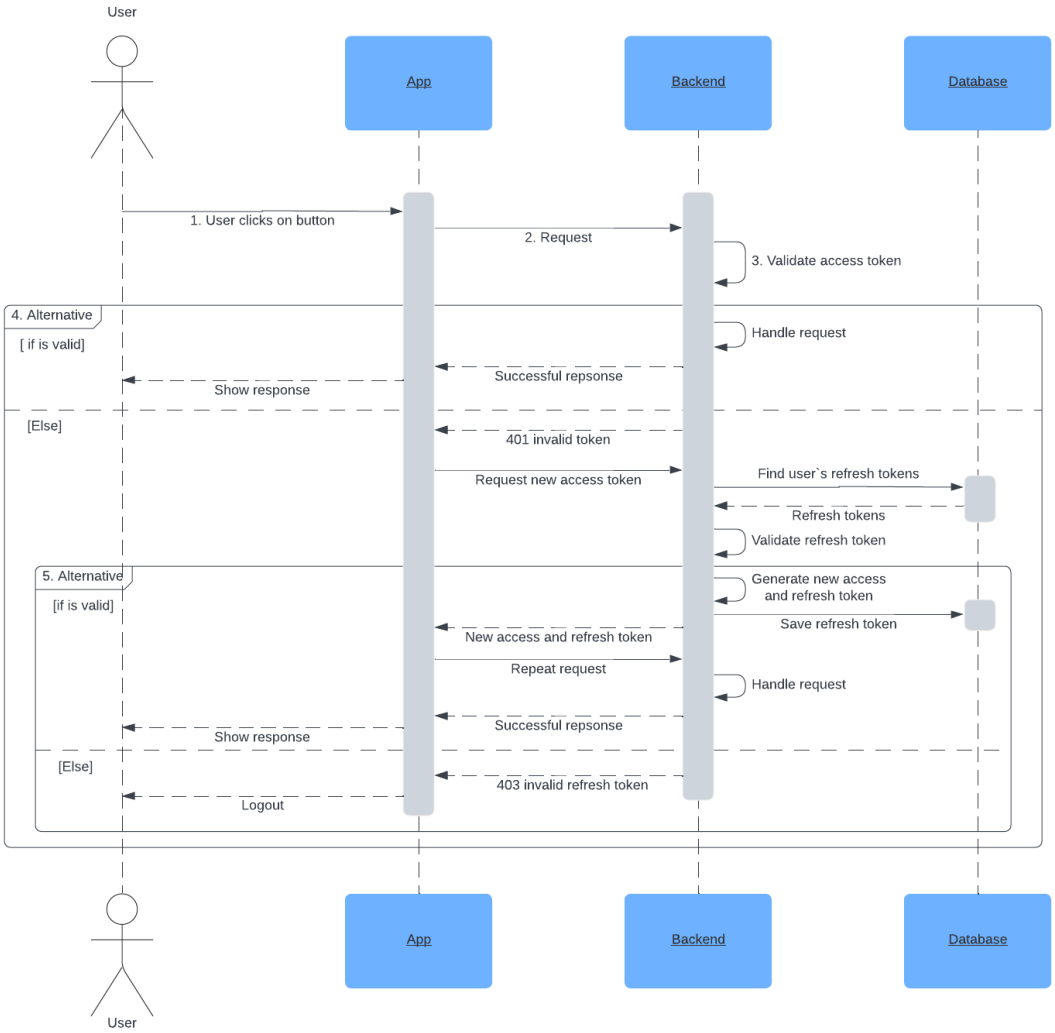
5. Vnořený Fragment s podmínkou:
   a. Pokud je obnovovací token validní a je uložený v poli obnovovacích tokenu kolekce users:

      Backend vygeneruje a odešle nový přístupový a obnovovací token.

      Následně je původní požadavek zopakovaný.
   b. Jinak:

      Backend pošle jako odpověď status kód 403.

      Aplikace při zachycení tohoto kódu provede odhlášení uživatele.



Obrázek 27: Proces autentizace uživatele

10.7.3 Proces připojení emailového účtu

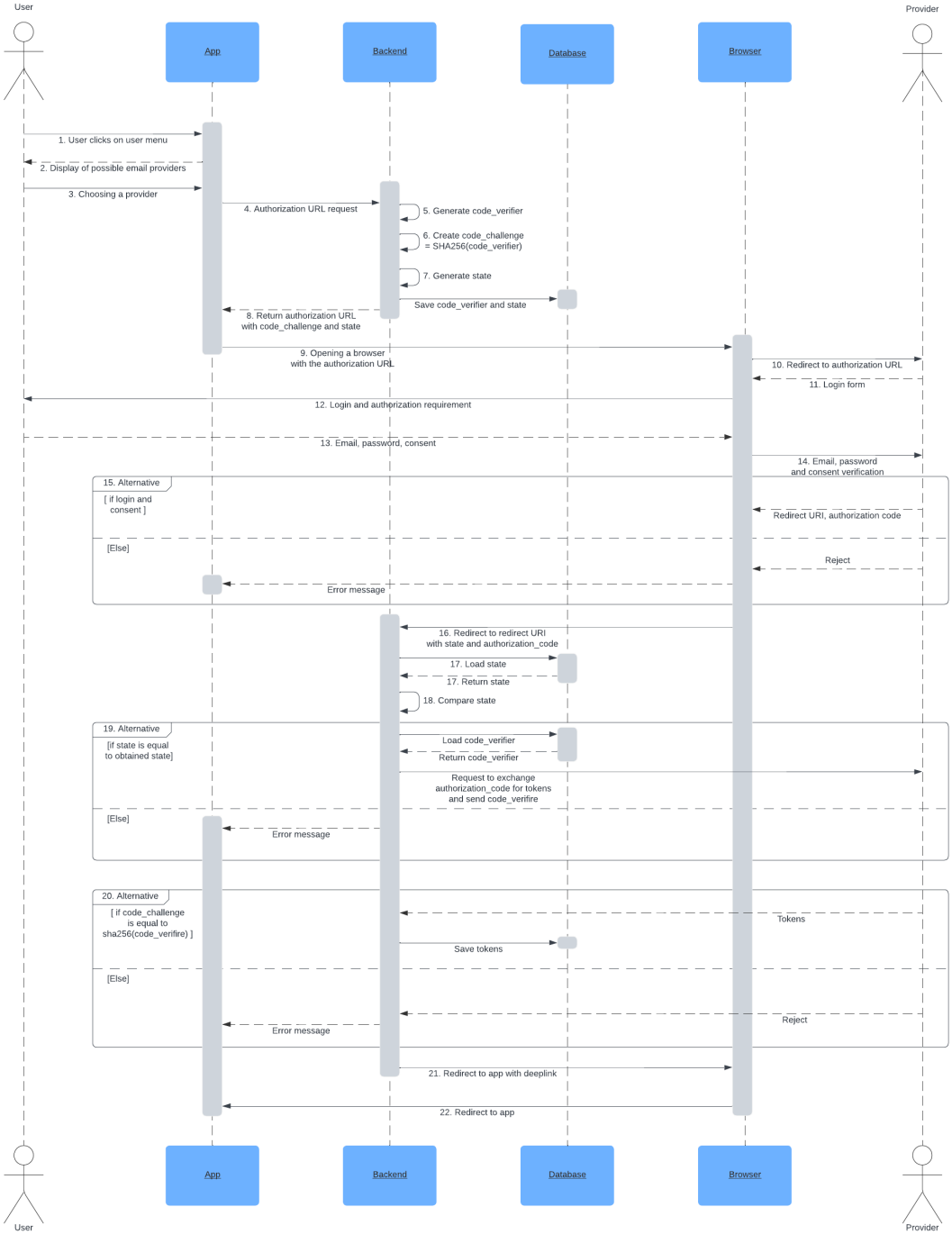
Na obrázků níže je prezentován sekvenční diagram procesu připojení emailového účtu za pomocí autorizačního protokolu OAuth 2.0. Protokol bude zhotoven ve formě *„Authorization Code grant“* s bezpečnostní vrstvou *„Proof Key for Code Exchange“*. Tento protokol je detailněji rozebraný v podkapitole 4.1.

Níže je rozepsán v krocích popis sekvenčního diagramu, který je prezentován na obrázku č. 28. V rámci toho procesu je počítáno s tím, že uživatel úspěšně prošel procesem kontroly autentizace ze sekce 10.7.2.

1. Uživatel rozbalí uživatelské menu.
2. Uživatelovi se zobrazí emailoví poskytovatelé Google a Outlook.

3. Uživatel vybere poskytovatele pro napojení svého emailového účtu.
4. Aplikace pošle požadavek pro získání autorizačního URL na endpoint „/oauth/:loginProvider/login“.
5. Backend vygeneruje *„code_verifier“* a zaznamená ho do kolekce oAuthSessions.
6. Backend zhotoví *„code_challenge“* z vygenerované hodnoty *„code_verifier“* za pomocí hash algoritmu SHA256.
7. Backend vygeneruje *„state“* a zaznamená ho do kolekce oAuthSessions.
8. Backend vrátí jako odpověď autorizační URL s hodnotami *„code_challenge“* a *„state*.
9. Aplikace otevře webový prohlížeč na zařízení.
10. Webový prohlížeč otevře autorizační URL, který odkazuje na emailového poskytovatele.
11. Ve webovém prohlížeči se otevře přihlašovací formulář poskytovatele.
12. Uživatelovi se zobrazí formulář a po přihlášení musí dát souhlas se sdílením jeho dat a k přístupu k psaní, čtení a trvalého odstranění emailových zpráv.
13. Uživatel zadá přihlašovací údaje a odsouhlasí požadované oprávnění pro přístup a správu svých dat.
14. Tato data se odešlou poskytovateli.
15. Fragment s podmínkou:
    a. Pokud se uživatel úspěšně autentizuje a odsouhlasí oprávnění:
       je na *„redirect uri“* (endpoint pro zpracování callbacku) zaslán *„authorization code“*.
    b. Jinak: Zobrazení chybové hlášky.
16. Backend na endpointu „/oauth/:loginProvider/callback“ přijímá od poskytovatele *„state“* (který byl přidán do query parametru v kroku č. 8) a *„authorization code“*.
17. Načtení hodnoty *„state“* z kolekce oAuthSessions.
18. Porovnání získané a načtené hodnoty *„state“*.
19. Fragment s podmínkou:
    a. Pokud se obě hodnoty rovnají:
       Načtení hodnoty *„code_verifier“ z* kolekce oAuthSessions a poslání společně s požadavkem na výměnu hodnoty *authorization code“* za přístupový a obnovovací token.
    b. Jinak: Zobrazení chybové hlášky.

20. Fragment s podmínkou:
    a. Pokud se odeslaná hodnota *code_challenge* (v kroku č. 8) rovná s tou vytvořenou u poskytovatele: [SHA256(*code_verifier*)]:
       Poskytovatel zašle přístupový a obnovovací token pro přístup ke chráněným zdrojům uživatele.
       Tokeny jsou následně uloženy do kolekce users.
    b. Jinak: Zobrazení chybové hlášky.
21. Backend zašle odpověď pro přesměrování uživatele zpět do aplikace za pomocí *„deeplink“* (URI sloužící pro otevření aplikace).
22. Přesměrování do aplikace s potvrzením o úspěšném připojení emailového účtu.



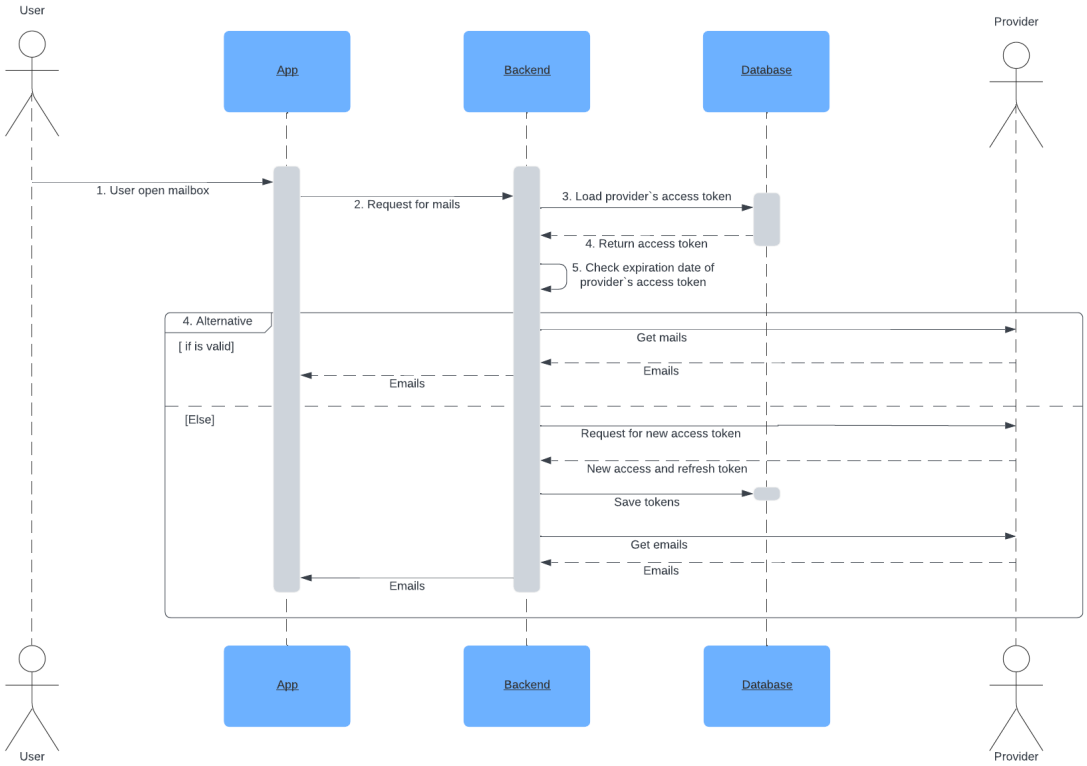
Obrázek 28: Proces připojení emailového účtu

10.7.4 Proces obnovy přístupového tokenu emailového poskytovatele

Při přístupu k emailovým službám poskytovatele, bude před každým požadavkem kontrolován přístupový token poskytovatele. Mezi tyto služby spadají všechny IMAP a SMTP koncové body definované v návrhu rozhraní (sekce 10.6.4 a 10.6.5).

Sekvenční diagram na obrázku č. 29 tento proces kontroly vizualizuje. Detailnější popis tohoto procesu je popsán níže. V rámci toho procesu je počítáno, že uživatel úspěšně prošel procesem kontroly autentizace ze sekce 10.7.2.

1. Uživatel otevře obrazovku schránky.
2. Aplikace pošle požadavek pro získání dávky náhledových informací dané emailové schránky.
3. Backend pošle požadavek na databázi pro načtení přístupového tokenu z kolekce users dle zvoleného emailového účtu.
4. Databáze vrátí přístupový token.
5. Backend zkontroluje datum expirace přístupového tokenu.
6. Fragment s podmínkou:
    a. Pokud je přístupový token validní:
       Backend provede IMAP dotaz pro získání dávky emailů.
       Backend data zpracuje a pošle dávku emailů do aplikace.
    b. Jinak:
       Backend požádá poskytovatele o nový přístupový token, jakmile ho získá tak jej uloží do databáze a provede IMAP dotaz pro získání dávky emailů.
       Následně data zpracuje a pošle emailová data do aplikace



Obrázek 29: Proces obnovy přístupového tokenu poskytovatele

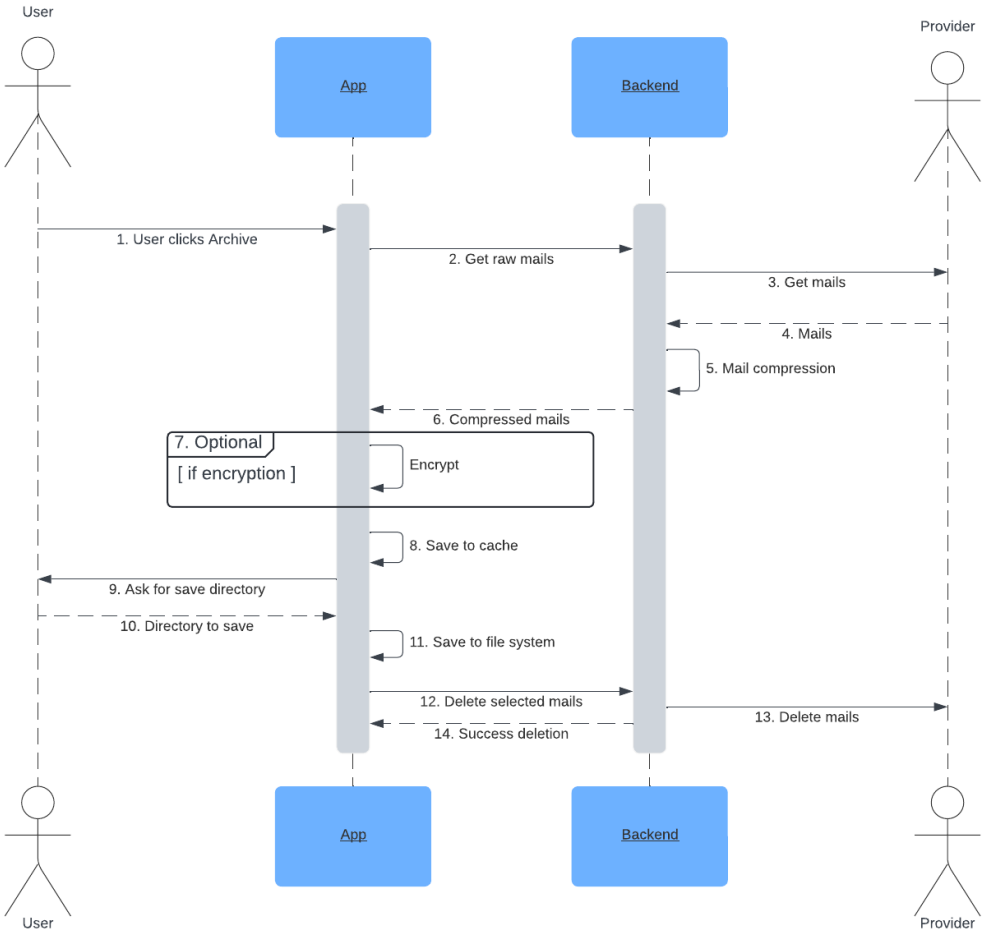
10.7.5 Proces lokální archivace

V této sekci bude navrhnut proces lokální archivace. Lokání archivace se v tomto kontextu rozumí, jako uložení archívu v mobilním zařízení. Tato archivace umožní uživateli uložit archív v zašifrované a nezašifrované formě.

Sekvenční diagram na obrázku č. 30 tento proces kontroly vizualizuje. Detailnější popis tohoto procesu je zhotovený níže. V rámci toho procesu je počítáno, že uživatel úspěšně prošel procesem kontroly autentizace ze sekce 10.7.1. Zároveň je bráno, že přístupový token poskytovatele je validní, a tak není potřeba jeho kontroly anebo obnovy ze sekce 10.7.4.

1. Uživatel již nad vybranými emailovými zprávami, stiskne tlačítko „archivovat“.
2. Aplikace pošle požadavek na získání surových emailových dat.
3. Backend pošle požadavek pro získání emailů na poskytovatele.
4. Poskytovatel vrátí emailová data.
5. Backend provede kompresi těchto dat.
6. Backend pošle odpověď v podobě komprimovaných dat.

7. Fragment volitelnosti:
    a. Pokud uživatel zvolí možnost zašifrované archivace:
    Aplikace provede šifrování archívu.
8. Archivace se uloží do cache.
9. Aplikace se zeptá uživatele na umístění pro uložení archivace do souborového systému.
10. Uživatel zvolí místo pro uložení.
11. Aplikace uloží archivaci do souborového systému.
12. Aplikace zašle požadavek pro odstranění archivovaných emailových zpráv ze schránky.
13. Backend zašle IMAP požadavek na poskytovatele.
14. Backend při úspěšné odstranění pošle potvrzení.



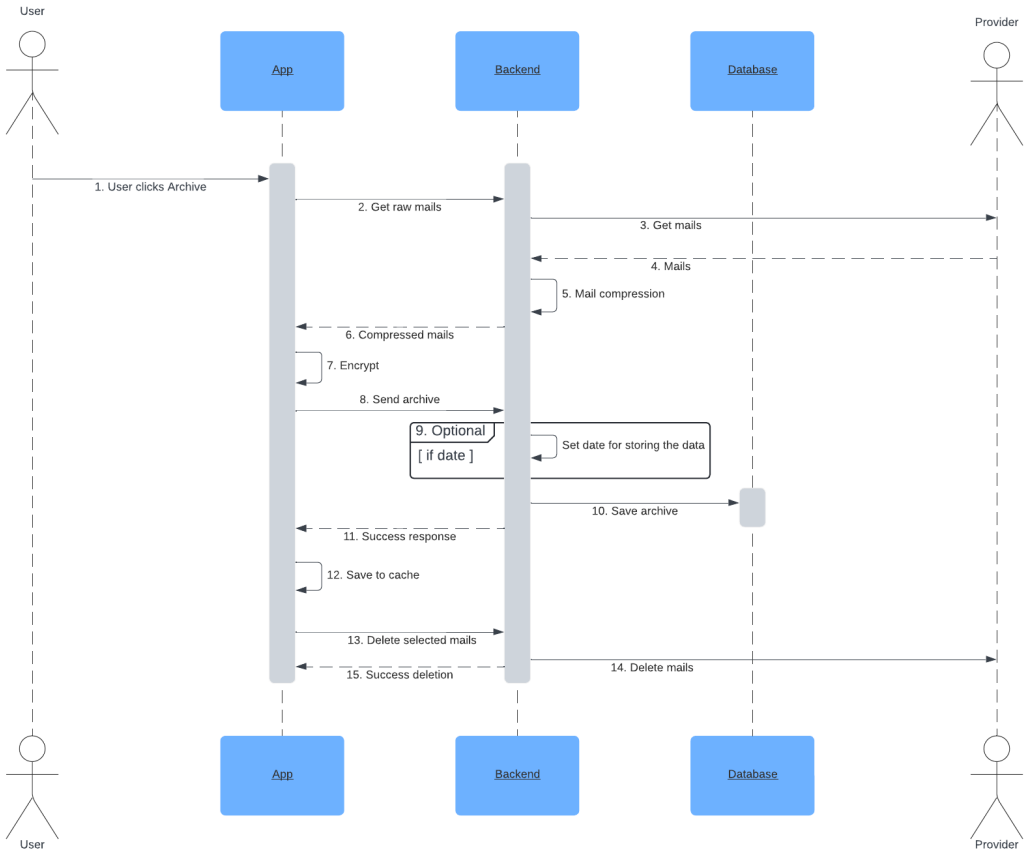
Obrázek 30: Proces lokální archivace

10.7.6 Proces serverové archivace

V této sekci bude navrhnut proces serverové archivace. Archivace se rovněž bude provádět na mobilním zařízení, ale tento archív bude následně uložen v databázi. Na mobilním zařízení bude archivace uchována pouze v cache paměti aplikace. To je z důvodu, aby v případě vymazání dat z databáze o archivované zprávy uživatel nepřišel. Tato forma archivace bude podporovat pouze ukládaní data v šifrované formě. Navíc bude možnost nastavit dobu uchování archívu v databázi.

Sekvenční diagram na obrázku č. 31 tento proces kontroly vizualizuje. Detailnější popis tohoto procesu je zhotovený níže. V rámci toho procesu je počítáno, že uživatel úspěšně prošel procesem kontroly autentizace ze sekce 10.7.2. Zároveň je bráno to, že přístupový token poskytovatele je validní, a tak není potřeba jeho kontroly anebo obnovy ze sekce 10.7.4.

1. Uživatel stiskne tlačítko sloužící pro započetí procesu archivace.
2. Aplikace pošle požadavek pro získání surových emailových dat.
3. Backend pošle požadavek na poskytovatele, pro získání emailů.
4. Poskytovatel zašle emailová data.
5. Backend provede kompresi těchto dat.
6. Backend zašle do aplikace komprimovaná data.
7. Aplikace provede proces šifrování.
8. Aplikace zašle zašifrovaná data na backend.
9. Fragment volitelnosti:
    a. Pokud uživatel zvolí délku uchování archívu:
    Backend nastaví dobu uchování.
10. Uložení archívu do databáze.
11. Backend pošle potvrzení úspěšném uložení archívu.
12. Aplikace uloží tento archív do cache paměti.
13. Aplikace pošle požadavek pro smazání emailových zpráv.
14. Backend pošle IMAP požadavek na poskytovatele pro smazání zpráv.
15. Backend pošle potvrzení o smazání.



Obrázek 31: Proces serverové archivace

10.7.7 Proces obnovy lokální archivace

V této sekci bude navrhnut proces obnovy lokální archivace. V rámci tohoto procesu je znázorněn krok nahrání archivace ze souborového systému.

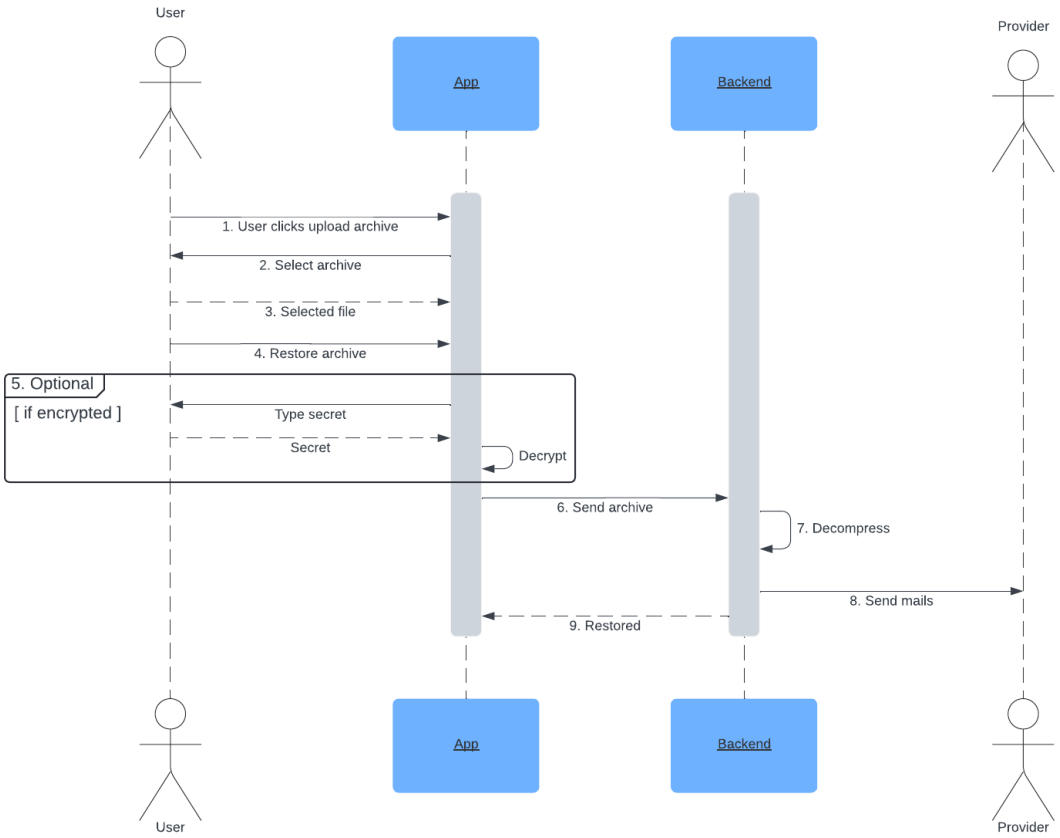
Sekvenční diagram na obrázku č. 32 tento proces kontroly vizualizuje. Detailnější popis tohoto procesu je popsán níže. V rámci toho procesu je počítáno, že uživatel úspěšně prošel procesem kontroly autentizace ze sekce 10.7.2. Zároveň je bráno to, že přístupový token poskytovatele je validní, a tak není potřeba jeho kontroly anebo obnovy ze sekce 10.7.4.

1. Uživatel stiskne tlačítko, pro nahrání archívu ze souborového systému zařízení.
2. Uživatelovi je zobrazeno okno, pro nahrání archívu.
3. Uživatel zvolí archív pro obnovu. V tento moment se nahraný archív zobrazí v seznamu všech lokálních archívů.
4. Uživatel klikne na tlačítko, pro obnovu archívu.
5. Fragment volitelnosti:
    a. Pokud je archív zašifrovaný:

Uživatel je dotázán, pro zadání tajného hesla.

Následně je proveden proces dešifrování.

6. Aplikace zašle archív na backend.
7. Backend provede dekompresi.
8. Backend zašle pomocí SMTP požadavek pro přidání emailových zpráv do mailboxu.
9. Backend zašle status o úspěšné obnově.



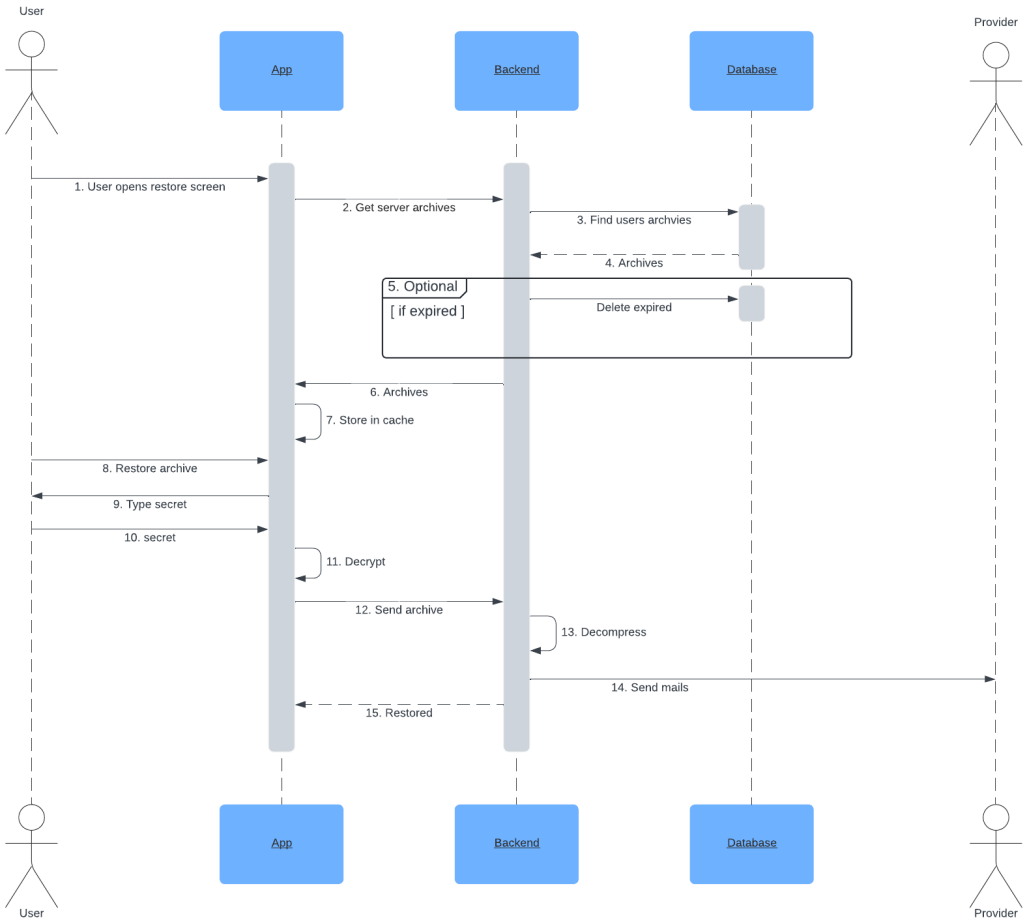
Obrázek 32: Proces obnovy lokální archivace

10.7.8 Proces obnovy serverové archivace

V této sekci bude navrhnut proces obnovy serverové archivace. Součástí toho procesu je zahrnut i krok, kdy jsou při otevření obrazovky pro obnovu archívu staženy serverové archívy.

Sekvenční diagram na obrázku č. 33 tento proces kontroly vizualizuje. Detailnější popis tohoto procesu je popsaný níže. V rámci toho procesu je počítáno, že uživatel úspěšně prošel procesem kontroly autentizace ze sekce 10.7.2. Zároveň je bráno, že přístupový token poskytovatele je validní, a tak není potřeba jeho kontroly anebo obnovy ze sekce 10.7.4.

1. Uživatel otevře obrazovku pro obnovu archívu.
2. Aplikace pošle požadavek na backend, pro získání archívů uložených v databází.
3. Backend pošle požadavek pro nalezení archívů korespondujícím s připojeným emailovým účtem.
4. Databáze vrátí archívy.
5. Fragment volitelnosti:
    a. Pokud mají archívy nastaveny datum uchování a již toto datum proběhlo: Backend provede požadavek na smazání exspirovaných archívů z databáze.
6. Backend navrátí seznam archívů.
7. Aplikace tento archív uloží do cache.
8. Uživatel klikne na tlačítko pro obnovu archívu.
9. Uživatel je dotázán pro zadání tajného hesla.
10. Uživatel zadá tajné heslo.
11. Aplikace dešifruje archív.
12. Aplikace zašle archív na backend.
13. Backend provede dekompresi.
14. Backend zašle pomocí SMTP požadavek pro přidání emailových zpráv do mailboxu.
15. Backend zašle status o úspěšné obnově.



Obrázek 33: Proces obnovy serverové archivace

11 VÝSLEDNÁ APLIKACE

V této kapitole budou představeny výsledky vyvíjené aplikace. Nejdříve bude popsána adresářová struktura aplikace. Následovat bude popsání implementace různých části aplikace se zakončenou částí o bezpečnostních prvcích aplikace.

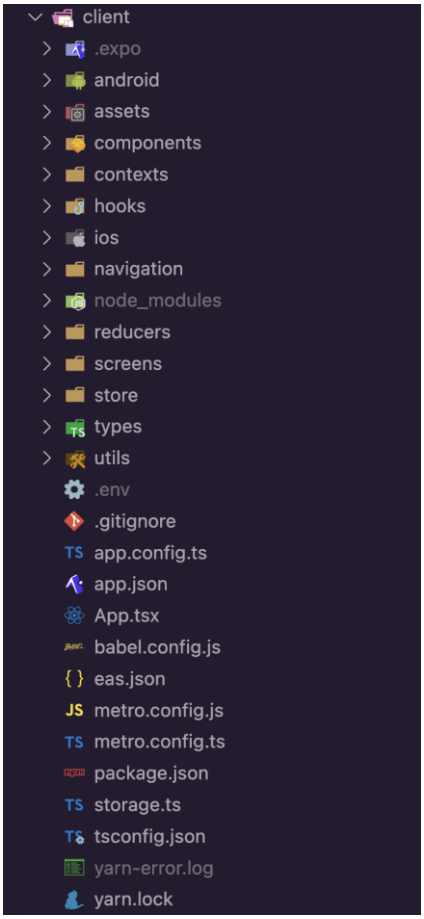
11.1 Adresářová struktura aplikace

Aplikace se skládá ze dvou celků, a to z klientské části a serverové části. V adresář *client/* představuje klientskou část aplikace a jedná se o kořenový adresář pro mobilní aplikaci. Složka *server/* na druhou stranu představuje serverovou část a je kořenovým adresářem pro backend. Posledním souborem je zde *.gitignore*, kde jsou vypsané soubory, které se nemají odesílat do vzdáleného uložiště.

Obrázek 34: Adresář aplikace

11.1.1 Struktura klientské části

Mobilní aplikace je napsaná pomocí frameworku Expo a React Native. Níže bude popsána adresářová struktura části mobilní aplikace.



Obrázek 35: Adresář klientské části

.expo/ – Dočasná složka využívaná frameworkem Expo.

android/ – Obsahuje specifická nastavení a zdroje pro Android verzi aplikace.

assets/ – Repositář sloužící pro uchování obrázků, ikon tlačítek, ikony aplikace a *„splash screen“* (úvodní obrazovka, která se zobrazuje při spuštění aplikace).

components/ – Adresář pro znovu použitelné komponenty. Dále se rozděluje atomické komponenty, jako například tlačítka a vstupní pole. A na molekulární komponenty, jako je komponenta pro výběr datumu nebo položka v seznamu.

contexts/ – Obsahuje *„React Contexts“* pro správu a sdílení stavů napříč komponentami.

hooks/ – Zde se nachází vlastní zhotovené *„hooks“* (funkce, které používají funkcionality frameworku React).

ios/ – Obsahuje specifická nastavení a zdroje pro iOS verzi aplikace.

navigation/ – Obsahuje konfiguraci navigace mezi různými obrazovkami.

node_modules/ – Adresář obsahující všechny nainstalované závislosti aplikace.

reducers/ – Složka pro funkce, které spravují stavové změny.

screens/ – Obsahuje jednotlivé obrazovky aplikace.

store/ – Adresář obsahující funkce pro správu a manipulaci s lokálním uložištěm aplikace.

types/ – Obsahuje typové definice a rozhraní pro TypeScript.

utils/ – Složka obsahující pomocné funkce.

.env – Soubor obsahující enviromentální proměnné, jako je URL adresa pro backend.

.gitignore – Soubor ve kterém se nachází výpis souborů v projektu, které mají být ignorovány systémem Git.

app.config.ts – Dynamický Expo konfigurátor projektu.

app.json – Statický Expo konfigurátor projektu.

App.tsx – Kořenová komponenta aplikace.

babel.config.js – Konfigurace pro Babel, který transpiruje JavaScript.

eas.json – Konfigurační soubor pro „*Expo Application Services“.*

metro.config.js – Konfigurace pro „*Metro bundler“*.

package.json – Seznam závislostí a skripty pro správu projektu.

storage.ts – Obsahuje vytvořenou instanci, sloužící pro ukládání a načítání dat.

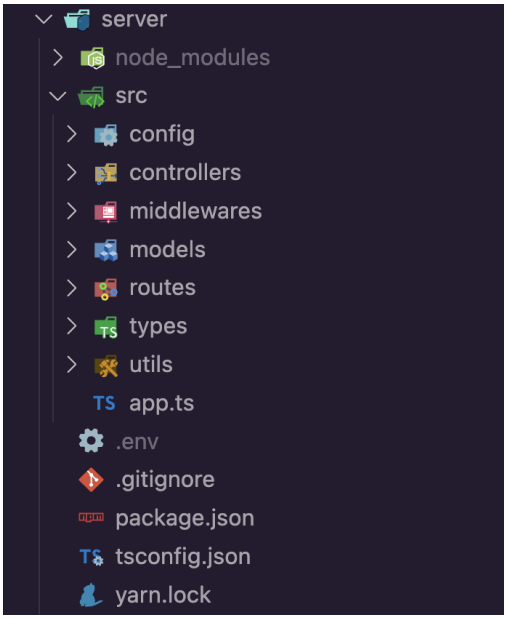
tsconfig.json – Konfigurační soubor pro TypeScript.

yarn-error.log – Soubor obsahující logy chyb z Yarn.

yarn.lock – Zajišťuje konzistenci zavilostí pomocí správci balíčku Yarn.

11.1.2 Struktura serverové části

Serverová část je napsána v runtime prostředí Node.js s frameworkem Express.js. Pro modelování dat a práci s databázi je použít Mongoose a samotná databáze je hostovaná v cloud službě MongoDB Atlas.



Obrázek 36: Adresář serverové části

node_modules/ – Adresář obsahující všechny nainstalované závislosti aplikace.

src/ – Kořenový adresář pro zdrojové soubory serveru.

- **config/** – Obsahuje konfigurační soubory pro protokoly IMAP, SMTP a OAuth 2.0.
- **controllers/** – Repositář, obsahující logiku pro zpracování http požadavků.
- **middlewares/** – Obsahuje funkce, které se provádí před zpracováním požadávku. Například kontrola validity přístupových tokenů
- **models/** – Obsahuje definice schémat pro Mongoose, které specifikují strukturu MongoDB databáze.
- **routes/** – Složka obsahující soubory, které definují trasy. Tyto trasy určují, jaké http požadavky aplikace přijímá a jaké *„controllers“* jsou volány pro zpracování těchto požadavků.
- **types/** – Složka obsahující typové definice a rozhraní pro TypeScript.
- **utils/** – Adresář obsahující pomocné funkce.
- **app.ts** – Kořenový soubor serveru.

.env – Soubor obsahující enviromentální proměnné.

.gitignore – Soubor ve kterém se nachází výpis souborů v projektu, které mají být ignorovány systémem Git.

package.json – Soubor obsahující seznam závislostí a skripty pro správu projektu.

tsconfig.json – Konfigurační soubor pro TypeScript.

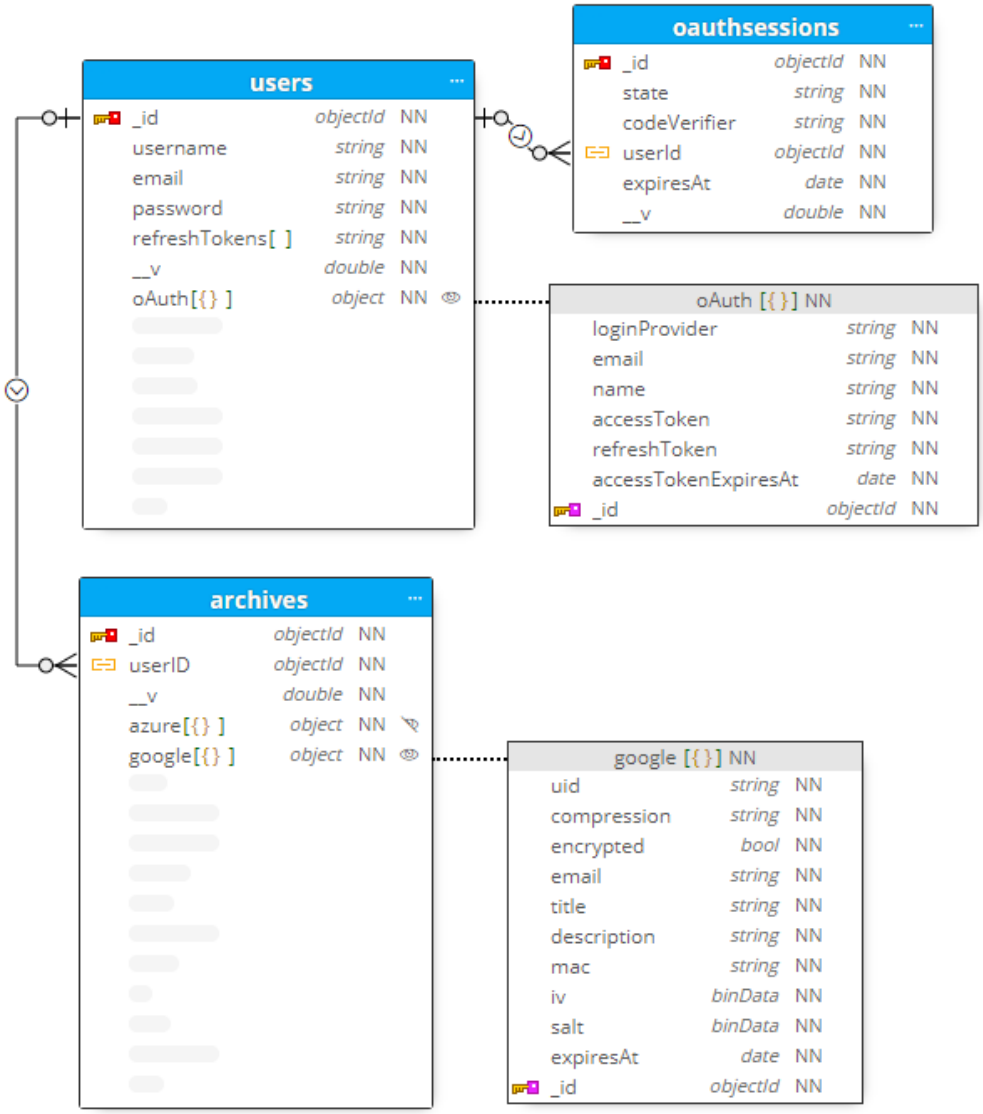
yarn.lock – Zajišťuje konzistenci zavilostí pomocí správci balíčku Yarn.

11.2 Popis výsledného datového modelu

Výsledný datový model aplikace na obrázku č. 37, je téměř totožný s tím navrhnutým v kapitole 10.3. Rozdíl je zde především ve struktuře uložených archívů, kde při návrhu tato struktura nebyla definována. Proto v této sekci bude popsaná pouze struktura objektu archívu, který se ukládá do databáze.

Každý uživatel může mít jeden dokument kolekce *archives*, jenž obsahuje záznam *azure a google*. Tyto záznamy reprezentují emailové poskytovatele, do kterých jsou zapisovány objekty archívu. Archív se skládá z následujících záznamů:

- **uid**: Unikátní identifikátor archívu.
- **compression**: Jedná se o samotný zašifrovaný a komprimovaný archív.
- **encrypted**: Označuje, zda je archivace zašifrovaná. V rámci serverové archivace vždy *„true“*.
- **email:** Hash hodnota připojené emailové adresy, ze které je archív proveden.
- **title**: Název archívu.
- **description**: Popisek archívu.
- **mac**: Hodnota, pro ověření správností zadaného heslo pro dešifrování.
- **iv**: Inicializační vektor pro dešifrování archívu s šifrovacím algoritmem AES.
- **salt**: Inivializační vektor pro získání dešifrovacího klíče ze zvoleného hesla pro archív.
- **expiresAt**: Datum uchování archívu v databázi.



Obrázek 37: Struktura datového modelu

11.3 Implementace přihlášení a registrace

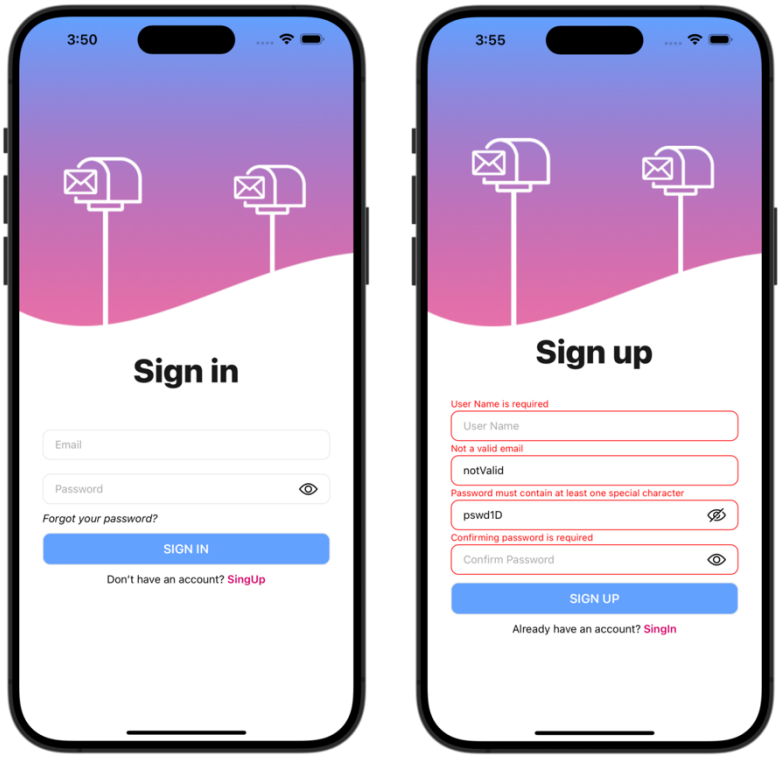
Návrh přihlášení byl zhotoven v sekci 10.7.1, kde tento proces je obdobný jako registrace uživatele.

Obrazovky přihlášení a registrace jsou implantovány v souborech *LoginScreen.tsx* a *RegisterScreen.tsx*. Pro oba formuláře je zhotovená validace jak na straně klienta, tak i na straně serveru. Na straně klienta se validace nachází přímo v souborech obrazovek a serverová validace je umístěná v repositáři *middleware/validation/* v souboru *user.ts*. Na druhé obrazovce obrázku č. 38 lze vidět v registračním formuláři ukázku kontroly validace. Je prováděna validace pro chybné pole, správný formát emailové adresy a správný formát hesla.

Formát hesla je definovaný následovně: Minimální délka 6 znaků, jeden velký znak, jeden speciální znak a jedna číslice.

Při úspěšném přihlášení či registraci, je vygenerován přístupový a obnovovací token. Obnovovací token je uložen do pole *refreshTokens* kolekce *users*. Uložení tokenu do pole tak umožňuje přihlášení na vícero mobilních zařízeních. Následně jsou oba tokeny zaslány do aplikace, kde jsou uloženy do bezpečného zašifrovaného uložiště. Manipulace se zašifrovaným uložištěm je implementována v souboru *secureStore.ts*.

Při každém spuštění aplikace dojde k ověření přítomnosti tokenů ve šifrovaném uložišti. Pokud jsou tokeny nalezeny, aplikace automaticky přesměruje uživatele na autentizované obrazovky. V případě, že tokeny nejsou dostupné, navigace uživatele směřuje na obrazovky určené pro neautentizované uživatele, tedy na obrazovky pro přihlášení, registraci a proces resetování hesla. Tato logika je implementována v souboru *RootNavigation.tsx*.



Obrázek 38: Obrazovky pro přihlášení a registraci

11.4 Implementace resetu hesla

Na obrázku č. 40 jsou tři obrazovky procesu resetování hesla při jeho zapomenutí. Implementace obrazovek a validace vstupních polí se nachází v souborech *ForgotPassword.tsx* a

PasswordReset.tsx. Validace formulářů je taktéž zhotovena na klientské a serverové části. Serverová validace se nachází v repositáři *middleware/validation/* v souboru *password.ts*.

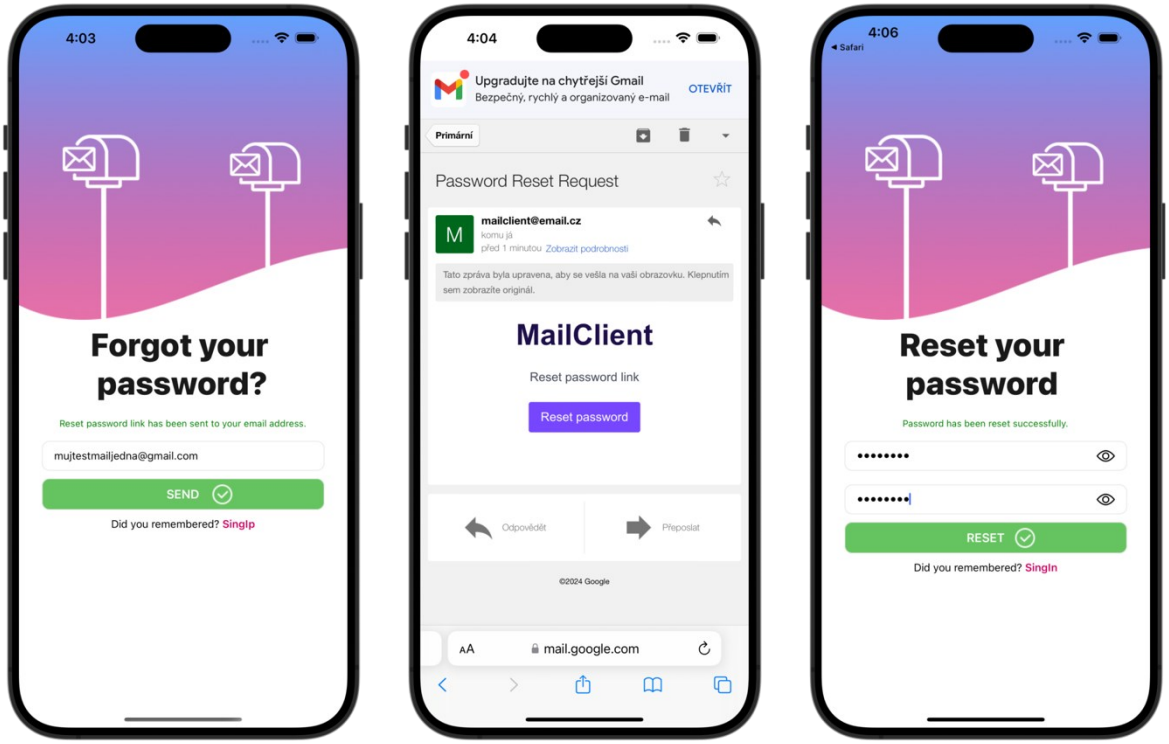
Implementace zpracování požadavků na resetování hesla se nachází v kontroléru *password.controller.ts*. Při prvotním požadavku, kde se vyplní emailová adresa, se na serveru zkontroluje existence uživatele. Pokud uživatel existuje, tak je vygenerován token pro reset hesla. Token je vložen do URL, který je integrován do tlačítka v emailové zprávě určené pro reset hesla, (druhá obrazovka na obrázku č. 40).

Z důvodu problémů s nefunkčním otevřením aplikace prostřednictvím zmíněného tlačítka bylo nezbytné do URL přidat koncový bod odkazující na server. Tento koncový bod zajišťuje přesměrování do aplikace spolu s tokenem (obrázek č. 39).

```
export const deeplinkReset = async (req: Request, res: Response) => {
  const token = req.params.token;
  res.redirect(`cz.mailclient.app:/reset-password?token=${token}`);
}
```

Obrázek 39: Koncový bod pro přesměrování do aplikace

Po přesměrování do aplikace se uživateli zobrazí obrazovka s formulářem pro nastavení nového hesla (třetí obrazovka na obrázku č. 40). S odesláním vyplněného formuláře se také odešle token získaný z emailové zprávy. Pří úspěšné validaci tokenu, se nastaví nové heslo.



Obrázek 40: Obrazovky procesu resetování hesla

11.5 Implementace autentizace

V Aplikace je implementována *„token-based“* autentizace, kde tento proces byl navrhnut v sekci 10.7.2. Před každým přístupem na chráněné koncové body, je kontrolován přístupový token (obrázek č. 41). Tato validace je implementována v souboru *tokenVerify.ts*, nacházející ve složce *middlewares/auth.* V případě, kdy je přístupový token nevalidní je jako odpověď poslán HTTP status kód 401. Jakmile aplikace tento kód zachytí, je započat proces obnovy přístupového tokenu, jenž je součástí návrhu v sekci 10.7.2. Aplikace původní požadavek uloží a vytvoří nový pro obnovu přístupového tokenu. Tato logika je implementována na straně klienta v souboru *axiosInstance.ts* nacházející v repositáři *utils*.

Proces obnovy přístupového tokenu na straně serveru je implementován v kontroléru *auth.controller.ts.* Provádí se v něm validace obnovovacího tokenu, součástí této validace je kromě kontroly expirace a modifikace, také kontrola pokusu o znovu použití obnovovacího tokenu. Pokud není nalezen uživatel s předloženým obnovovacím tokenem, tak je z tohoto tokenu extrahováno id uživatele a dotyčný uživatel je odhlášen ze všech zařízení. Protože při každé obnově přístupového tokenu, je předložený obnovovací token smazaný z pole obnovovacích tokenů.

Pokud je nalezený uživatel, tak se odebere obnovovací token z pole v databázi a provede validace tohoto tokenu. Pokud je validní, tak je vygenerován nový přístupový a obnovovací token, který jsou zaslány aplikaci. Nový obnovovací token je uložen do pole *refreshTokens* kolekce *users*.

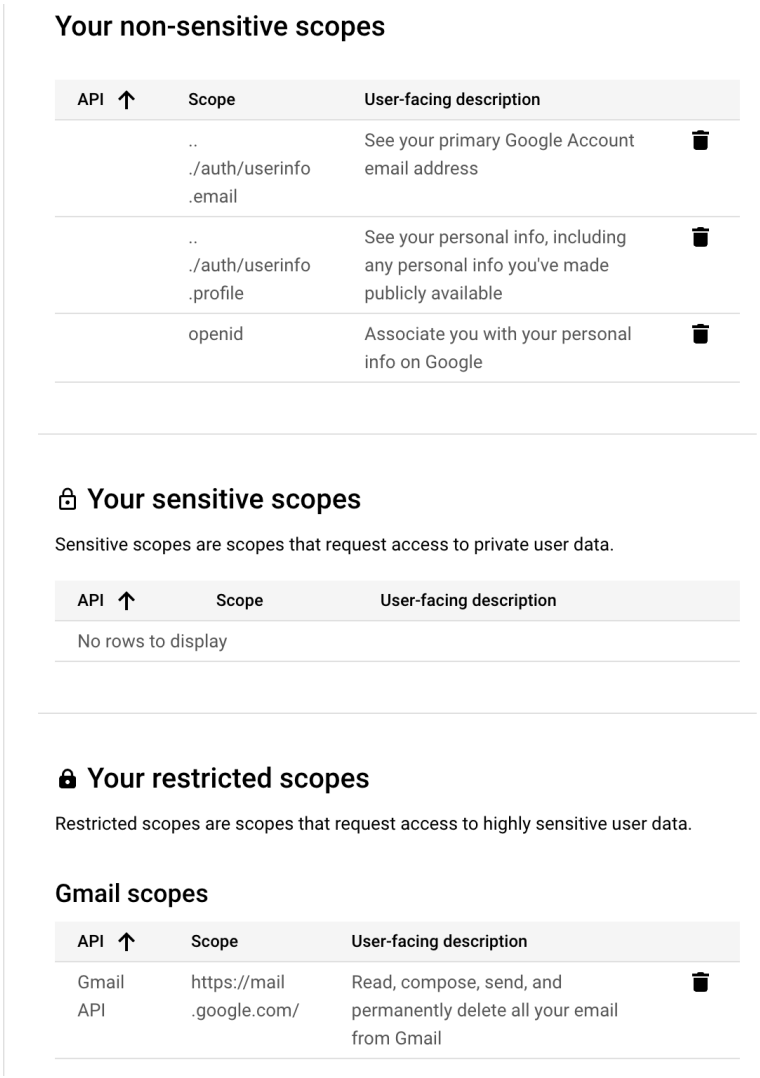
V případě, kdy validace předloženého obnovovacího tokenu selže, tak server vrátí HTTP status kód 403. Při obdržení tohoto kódu, dojde na straně klienta k odhlášení uživatele.

```
export const verifyJWT = (req: Request, res: Response, next: NextFunction) => {
  const authHeader = req.headers.authorization || req.headers.Authorization;

  if (!authHeader?.toString().startsWith('Bearer ')) return res.status(403).json({ error: "Request without Bearer header"});

  const token = authHeader.toString().split(' ')[1];

  jwt.verify(
    token,
    process.env.ACCESS_TOKEN_SECRET as string,
    (err: jwt.VerifyErrors | null, decoded: any) => {
      if (err) return res.status(401).json({ error: err }); //invalid token
      req.body.id = decoded.id;
      req.body.email = decoded.email;
      next();
    }
  );
}
```

Obrázek 41: Validace přístupového tokenu

11.6 Implementace OAuth 2.0

Pro implementaci protokolu OAuth 2.0 je nezbytné zaregistrovat aplikaci v prostředích Google a Azure (Outlook). Pro Google se aplikace registrují na portále *Google Cloud Platform* a pro Azure na *Microsoft Azure Portal*. Během registrace se nastavují detaily, jako jsou název aplikace a URI pro přesměrování. Součástí tohoto kroku je také definování „scopes“, které představují specifická oprávnění udělovaná aplikaci pro přístup k chráněným zdrojům a službám spojeným s uživatelem. Pro implementaci této aplikace byly zapotřebí tyto *„scopes“*:



Obrázek 42: Google scopes

- *./auth/userinfo.email* – Umožňuje získání emailové adresy uživatele.
- *./auth/userinfo.profile* – Umožňuje přístup k profilovým informacím uživatele, včetně jeho jména, které bylo nezbytné pro SMTP protokol. V rámci tohoto protokolu se kromě emailové adresy přidává do informací o odesílateli také jméno uživatele.
- *openid* – Umožňuje získat *id token* ve kterém se nachází emailová adresa.
- *https://mail.google.com/* - Umožňuje spravování a odesílaní emailových zpráv pomocí protokolů IMAP a SMTP.

Název rozhraní API / oprávnění	Typ	Popis
⌄ Microsoft Graph (6)		
email	Delegováno	View users' email address
IMAP.AccessAsUser.All	Delegováno	Read and write access to mailboxes via IMAP.
offline_access	Delegováno	Maintain access to data you have given it access to
openid	Delegováno	Sign users in
profile	Delegováno	View users' basic profile
SMTP.Send	Delegováno	Send emails from mailboxes using SMTP AUTH.

Obrázek 43 Azure scopes

- *email* – Umožňuje získání emailové adresy.
- *IMAP.AccessAsUser.All* – Umožňuje správu emailových zpráv pomocí protokolu IMAP.
- *Offline_access* – Umožňuje obnovu přístupového tokenu, pomocí obnovovacího tokenu.
- *Openid* – Umožňuje získat *id token* ve kterém se nachází emailová adresa a jméno uživatele.
- *profile* – Umožňuje získání profilu uživatele.
- *SMTP.Send* – Umožňuje odesílat emailové zprávy pomocí protokolu SMTP.

Po úspěšné registraci aplikace, jsou u obou služeb vygenerovány klíčové identifikátory a tajemství, s kterýma se aplikace při autorizačního procesu prokáže.

Na obrázku č. 44 je ukázka kódu obsahující konfiguraci pro autorizační proces OAuth 2.0. V této konfiguraci se nachází důležité identifikační údaje získané během registrace aplikace, společně s definovaným URI pro přesměrování a „*scopes*“. Součástí toho jsou i URL adresy koncových bodů, pro započetí autorizace a získání tokenů.

```
export const loginProviders: Record<LoginProviders, LoginProvider> = {
  google: {
    client_id: process.env.GOOGLE_CLIENT_ID,
    client_secret: process.env.GOOGLE_CLIENT_SECRET,
    token_endpoint: 'https://accounts.google.com/o/oauth2/token',
    authorization_endpoint: "https://accounts.google.com/o/oauth2/v2/auth",
    redirect_uri: process.env.BACKEND_URI + 'oauth/google/callback',
    scope: 'profile email openid https://mail.google.com/',
    access_type: 'offline',
    prompt: 'consent',
  },
  azure: {
    client_id: process.env.AZURE_CLIENT_ID,
    client_secret: process.env.AZURE_CLIENT_SECRET,
    token_endpoint: 'https://login.microsoftonline.com/consumers/oauth2/v2.0/token',
    authorization_endpoint: "https://login.microsoftonline.com/consumers/oauth2/v2.0/authorize",
    redirect_uri: process.env.BACKEND_URI + 'oauth/azure/callback',
    scope: "openid offline_access email profile https://outlook.office.com/IMAP.AccessAsUser.All"
  }
};
```

Obrázek 44: Konfigurace pro OAuth proces

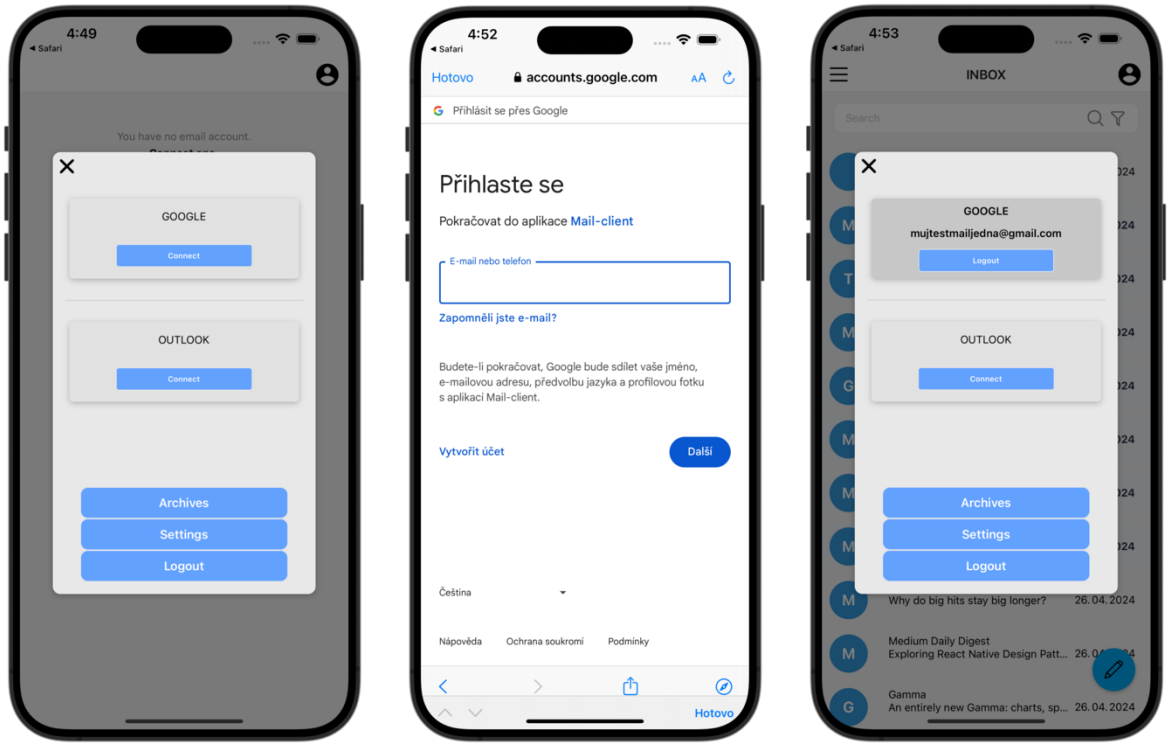
Autorizační proces pomocí protokolu OAuth 2.0. v kontextu této aplikace je využit k připojení emailových účtu. Tento proces byl navrhnut v sekci 10.7.3. a je implementovaný na straně serveru v souboru *oAuth.controller.ts*. V mobilní aplikaci je tento proces započat a ukončen v souboru *LoginView.tsx*, který se nachází ve složce *components*. Proces připojení z pohledu uživatele je ukázán na obrázku č. 45.

V rámci tohoto procesu, jsou potřebné hodnoty *code_verifier* a *state* ukládané v kolekci *oAuthSessions*, která slouží jako dočasná relace. Životnost jedné této relace je nastavená na 5 minut a po uplynutí této doby je záznam relace smazaný.

Po úspěšném dokončení autorizačního procesu, je přístupový a obnovovací token poskytovatele uložen do kolekce *users*. Přístupový token má svou živostnost, kterou je třeba před každým požadavkem kontrolovat. Poskytovatelé Google a Azure (Outlook) používají přístupové tokeny ve formě náhodného řetězců znaků, kde není uložen datum exspirace, oproti JWT, kde toto datum lze zjisti při dekódování tokenu. Také při neplatném tokenu neposílají jako odpověď specifický HTTP status kód. Proto bylo potřeba v databázi ukládat společně s tokeny i datum expirace. Tento datum expirace poskytovatelé posílají společně s tokeny.

Proces kontroly přístupového tokenu poskytovatele, a i jeho obnovy je navrhnut v sekci 10.7.4. Implementace kontroly je zhotovená v souboru *oAuthTokenVerify.ts*, kde se porovnává čas uložený v databází s aktuálním časem. Od aktuálního času jsou odečteny 2 minuty, aby se ošetřil případ, kdy by přístupový token exspiroval těsně po jeho validaci. V případě

neplatného tokenu, se započne proces jeho obnovy, který je zhotoven v souboru *oAuthRefreshToken.ts*. Oba tyto soubory se nachází v serverové části v repositáři *middlewares/oAuth.*



Obrázek 45: Obrazovky pro připojení emailového účtu

11.7 Implementace IMAP protokolu

Veškeré koncové body pro správu emailových zpráv jsou zhotovené v kontroléru *imap.controller.ts*. Pro práci s protokolem IMAP je použita knihovna *ImapFlow*, kde instance této knihovny je vytvořena v souboru *imapClient.ts* (obrázek č. 46)

```
import { ImapFlow, ImapFlowOptions } from 'imapflow';
import { LoginProviders } from '../config/oAuthConfig';
import { imapConfig } from '../config/imapConfig';

export const createImapClient = (provider: LoginProviders, email: string, accessToken: string): ImapFlow => {

    const configuration = imapConfig[provider];

    const options: ImapFlowOptions = {
        host: configuration.imapHost,
        port: configuration.imapPort,
        secure: true,
        auth: {
            user: email,
            accessToken: accessToken,
        },
        logger: false,
    };

    return new ImapFlow(options);
}
```

Obrázek 46: IMAP *client*

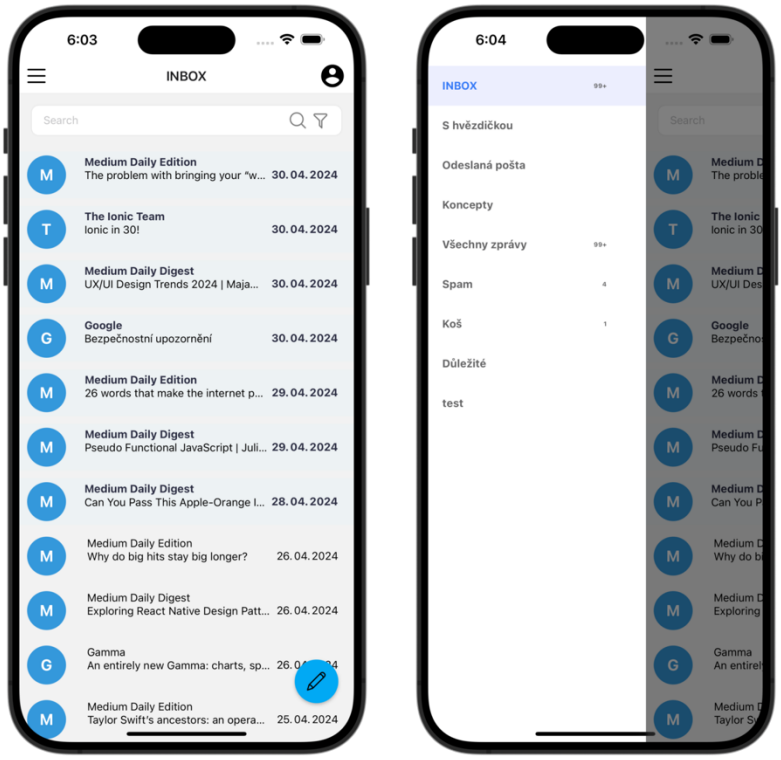
Pro vytvoření instance je potřeba port a adresa pro IMAP server poskytovatele. Tyto informace jsou definované v konfiguračním soboru *imapConfig.ts* (obrázek č. 47).

```
export const imapConfig: Record<LoginProviders, ImapStructure> = {
  google: {
    imapHost: 'imap.gmail.com',
    imapPort: 993,
  },
  azure: {
    imapHost: 'outlook.office365.com',
    imapPort: 993,
  }
};
```

Obrázek 47: Konfigurační soubor pro IMAP protokol

11.8 Implementace obrazovky mailové schránky

Obrazovka mailové schránky je zhotovená v souboru *MailScreen.tsx* (první obrazovka v obrázku č. 48). Přepínaní mezi schránkami je umožněno zvolením názvu schránky v navigačním menu zvané *„burger menu“* (druhá obrazovka v obrázku č. 48).



Obrázek 48: Obrazovky mailové schránky

Při připojení emailového účtu, jsou pomocí IMAP protokolů stažený schránky a následně dynamicky vykresleny do zmiňovaného navigačního menu. Současně jsou schránky uloženy do cache paměti, kde správa paměti je zhotovená v souboru *mailStore.ts*. Při následném přepínání mezi emailovými účty jsou také staženy emailové schránky, ale zároveň i načteny z paměti. Následně se porovnávají obě data a zjišťuje se, zda nějaké schránky nebyly odstraněny. V případě neshody, se emailová data odstraněných schránek vymažou z cache paměti. Tato veškerá logika je implementována v navigační komponentě *DrawerNavigator.tsx*.

Každá položka v menu je pak napojená na komponentu *MailScreen.tsx*, kde se provádí načítání emailových zpráv. Při otevření emailové schránky, v případě nového napojeného emailové účtu, se získají pomocí protokolu IMAP emailová náhledová data posledních (nejnovějších) třiceti zpráv. Tato náhledová data se následně uloží do cache paměti, kde jejích správa je taktéž implementovaná v souboru *mailStore.ts.* Při posunu obrazovkou směrem dolů, se při dosažení konce načte další várka emailových náhledů (první obrazovka v obrázku č. 49), které se taktéž uloží do paměti.

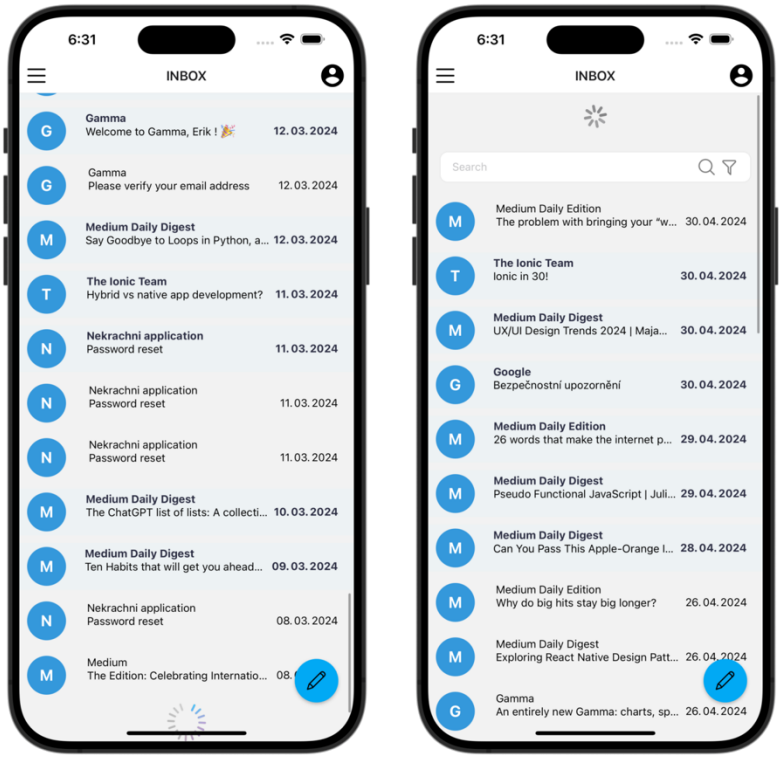
V případě, když již schránka byla někdy otevřená, tak jsou data uložená v cache paměti. Před jejich vykreslení do seznamu, je ale potřeba zkontrolovat hodnoty schránky *HighestModSeq* a *UidValidity.* Tyto hodnoty jsou uloženy společně se schránkami v paměti a jsou při

otevření schránky porovnávány s těmi získanými z požadavku o statusu, který se provádí při otevření schránky.

Pokud se změnila hodnota *HighestModSeq* tak zprávy ve schránce byly změněny. Jako změna se považuje odstranění některé zprávy anebo označení zprávy jak přečtenou a nepřečtenou. Jestliže se nerovná získaná hodnota *UidValidity* s tou uloženou v paměti, tak původní schránka byla odstraněna a zároveň byla založená nová se stejným názvem.

V obou případech je potřeba získat nejnovější emailová data schránky. Při této obnově se provede akce *„pull to refresh“* (druhá obrazovka v obrázku č. 49). Tuto akci lze provést i samostatně pro aktualizování schránky, kde stačí popotáhnout seznam mailů směrem dolů.

Veškerá logika pro práci s protokolem IMAP, je implementována na straně serveru v kontroléru *imap.controller.ts*.



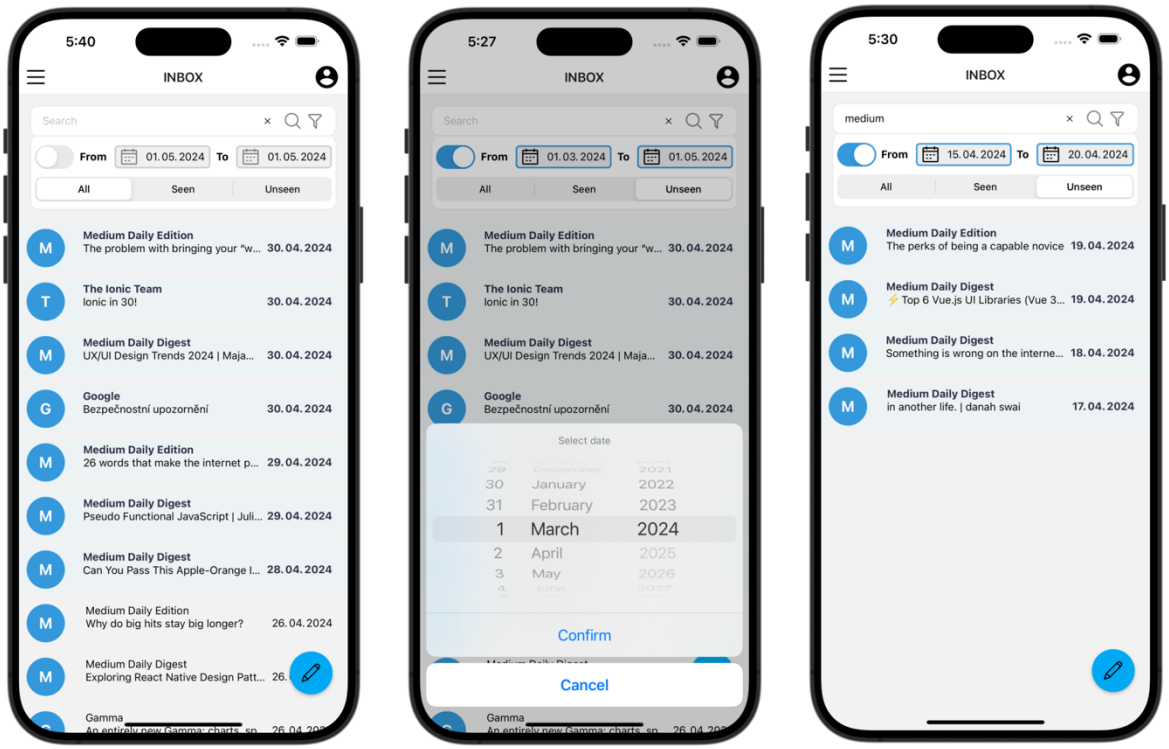
Obrázek 49: Obrazovky mailové schránky s funkcemi pro načítání

11.9 Implementace filtrace

Filtrace je zhotovená v komponentně *SearchBar.tsx* a na straně serveru je implementována v kontroléru *imap.controller.ts*. Základní filtrace probíhá prostřednictvím vstupního pole, které umožňuje vyhledávat emailové zprávy v aktuálně otevřené schránce podle jména

odesílatele nebo jeho emailové adresy a také podle předmětu zprávy. Navíc filtrace nabízí pokročilá vyhledávací kritéria, jako jsou datumová rozpětí a možnost filtrovat zprávy na základě toho, zda byly přečtené či nepřečtené. Při nastavení datového intervalu lze oba data (datum „od“ a datum „do“) nastavit na tentýž den, což umožní získat zprávy z konkrétního dne. Datumová pole jsou navržena tak, aby předešla nevalidním časovým rozpětím: pokud je datum „od“ nastaveno na pozdější než datum „do“, automaticky se upraví, aby odpovídalo datu „do“. Datum „do“ je omezeno maximálně na aktuální datum, aby se předešlo výběru budoucích dat.

Po odeslání kritérií pro vyhledávání je na straně serveru proveden IMAP vyhledávací požadavek. Vyhledané emailové zprávy jsou posléze navráceny do aplikace.



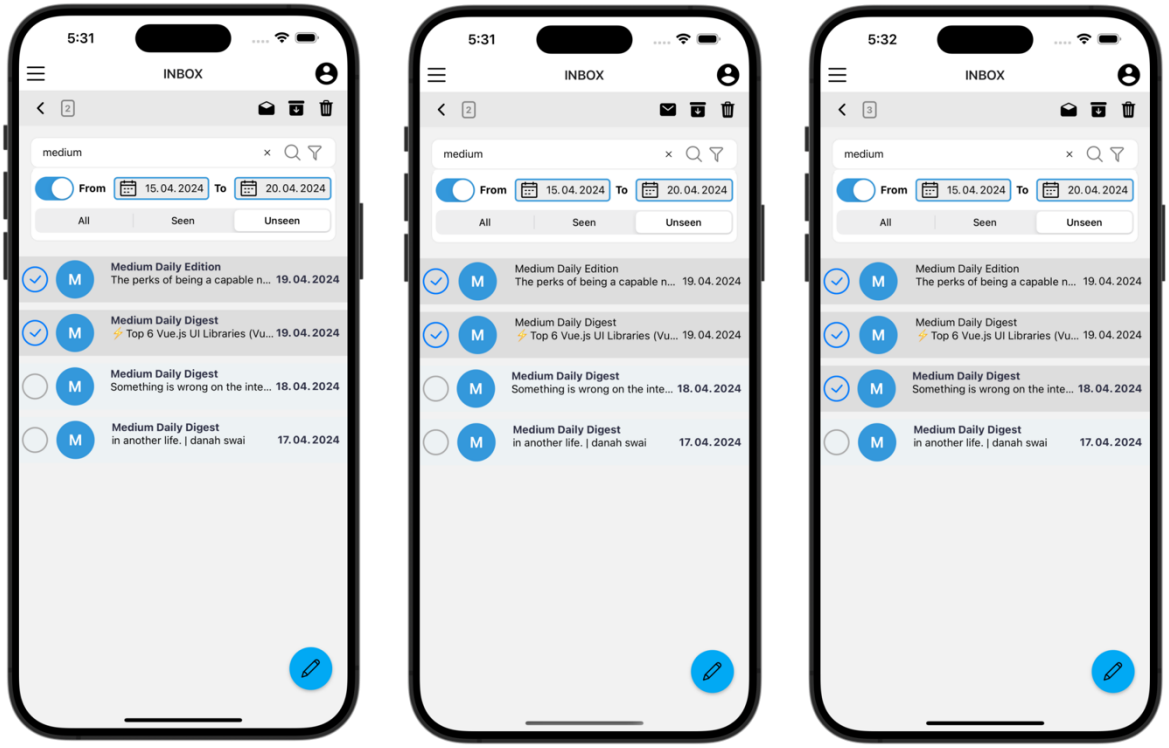
Obrázek 50: Obrazovky s filtrací emailových zpráv

11.10 Implementace správy emailových zpráv

Nástrojová lišta s funkcemi pro správu emailových zpráv se otevře, když je označený alespoň jedna emailová zpráva. Označení se provede podržením jedné zprávy a pro označování dalších zpráv stačí kliknout na další položky v seznamu. Logika označování je implementována v komponentně *MailList.tsx*. Nástrojová lišta lze použít jak na nefiltrované, tak i na vyfiltrované emailové zprávy.

Následně budu popsány funkcionality nástrojové lišty, jež jsou implementovány v komponentně *MailScreen.tsx*.

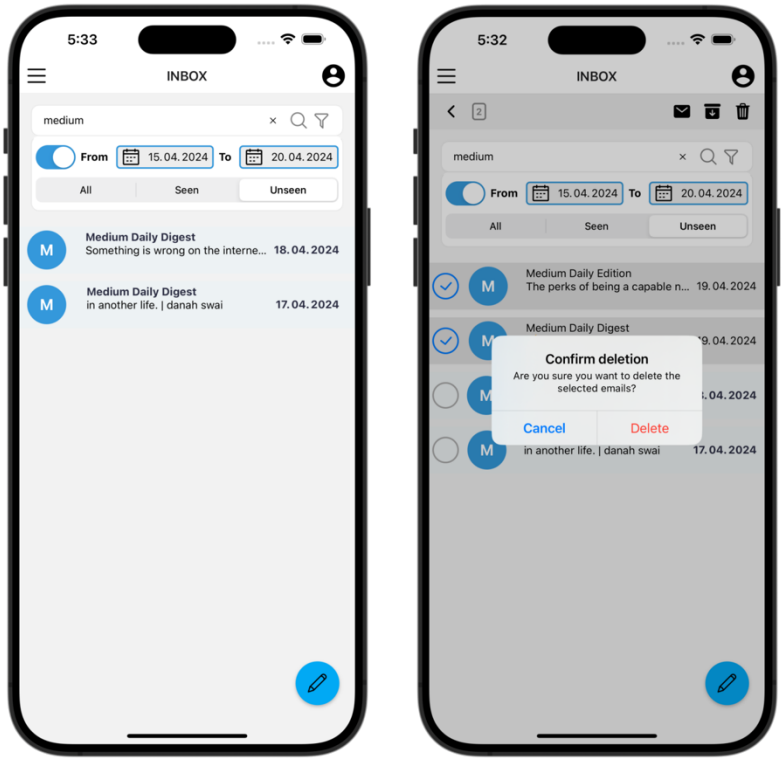
První z funkcí nástrojové lišty, je označení zprávy jako „přečtená“ nebo „nepřečtená“ (obrázek č. 51). Pokud je z označených emailových zpráv alespoň jedna zpráva nepřečtená, tak lze zprávy označit pouze jako „přečtené“ (třetí obrazovka obrázku č. 51). Označení jako „přečtené“ je reprezentované ikonou otevřené poštovní obálky a označení jako „nepřečtené“ je na druhou stranu reprezentované jako ikona zavřené poštovní obálky.



Obrázek 51: Obrazovky s označováním zpráv jako přečtené/nepřečtené

Druhou funkcionalitou je pokročilá archivace. Ta však bude popsána v následující sekci č. 11.11.

Poslední funkcionalitou je odstranění emailové zprávy (obrázek č. 52). Před trvalém odstranění emailové zprávy je uživatel dotázán pomocí „*pop-up*“ okna pro potvrzení o odstranění vybraných emailových zpráv.



Obrázek 52: Obrazovky s odstraněním emailových zpráv

11.11 Implementace pokročilé archivace

Archívy z procesu pokročilé archivace lze uchovat dvěma způsoby. Jedním z nich je uchování archívu na mobilním zařízení a druhý je uchování na serveru v databázi. Tento proces je implementován na straně klienta v souboru *ArchivationScreen.tsx* a na straně serveru jsou IMAP požadavky v kontrolérů *imap.controller.ts*.

Oba způsoby mají stejný základní princip:

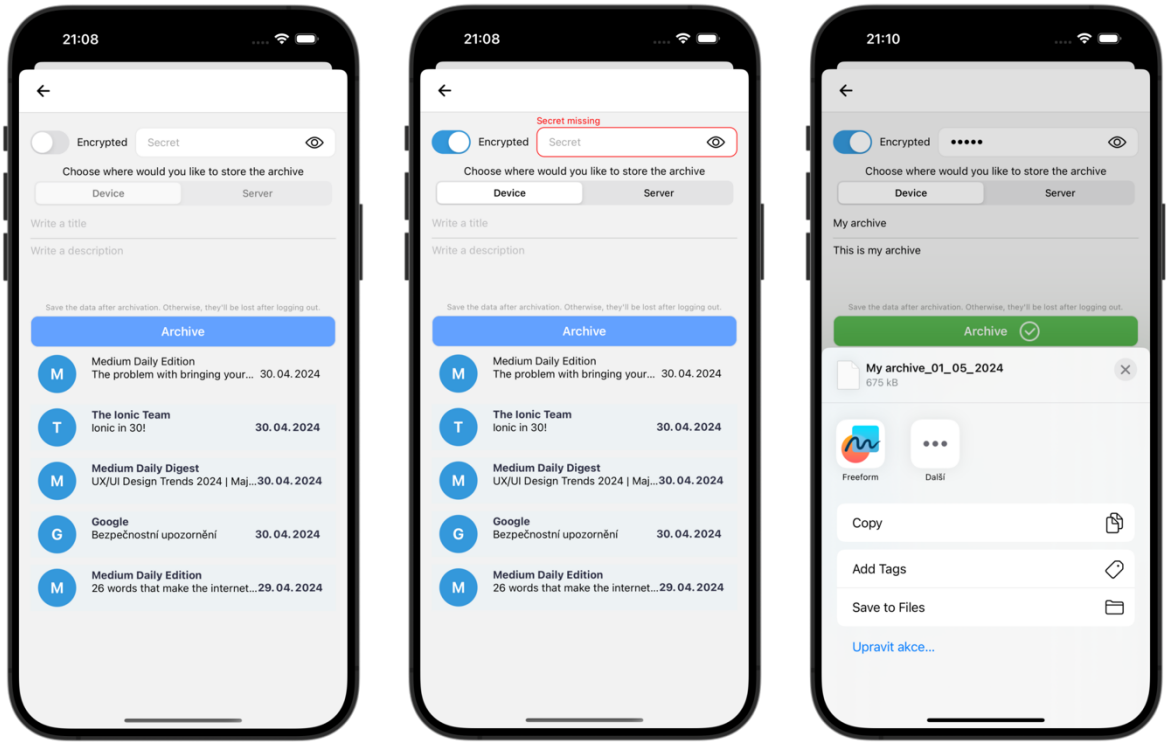
Po označení filtrovaných nebo nefiltrovaných zpráv a stisknutí tlačítka pro archivaci jsou tyto zprávy přeneseny do obrazovky pro pokročilo archivaci. Uživatel má k dispozici volitelná pole pro pojmenování archívů a přidání popisku. Po stisku tlačítka „Archive“ je poslán požadavek na server, pro získání surových emailových dat. Důvodem pro získání surových dat je zachování všech metadat emailových zpráv a následného ulehčení procesu jejich obnovy. Server tato data získá od poskytovatele pomocí IMAP protokolu a před odesláním zpět do aplikace je zkomprimuje. Funkce pro kompresi je zhotovená v souboru *compression.ts*, jenž se nachází v repositáři *utils/*.

11.11.1 Klientská pokročilá archivace

Proces klientské archivace byl navrhnut v sekci 10.7.5. Uchování archívu na mobilním zařízení lze provést jak v zašifrované, tak i v nezašifrované formě. V případě nezašifrované podoby je po přijetí komprimovaných dat ze serveru uživateli nabídnuto uložení archívu do souborového systému (třetí obrazovka v obrázku č. 53). Uživatel může tento krok přeskočit, protože je archív automaticky uložený v cache paměti. Ale v případě, kdy se uživatel odhlásí tak jsou všechna data uložená v cache smazána. Tohle upozornění je umístěno nad tlačítkem pro započetí archivace.

Druhý případ je uchování v zašifrované podobě, kde se po přijetí komprimovaných dat provede proces šifrování. Detailní postup procesu šifrování dat je popsán v sekci 11.12. Po úspěšném zašifrování je uživateli taktéž zobrazeno *„pop-up“* okno pro uložení s následným uchováním v paměti cache.

Po úspěšném uložení archívu se v obou případech odešle požadavek na server pro smazání archivovaných zpráv z emailové schránky. Server po přijetí pošle poskytovateli IMAP požadavek pro smazání emailových zpráv.



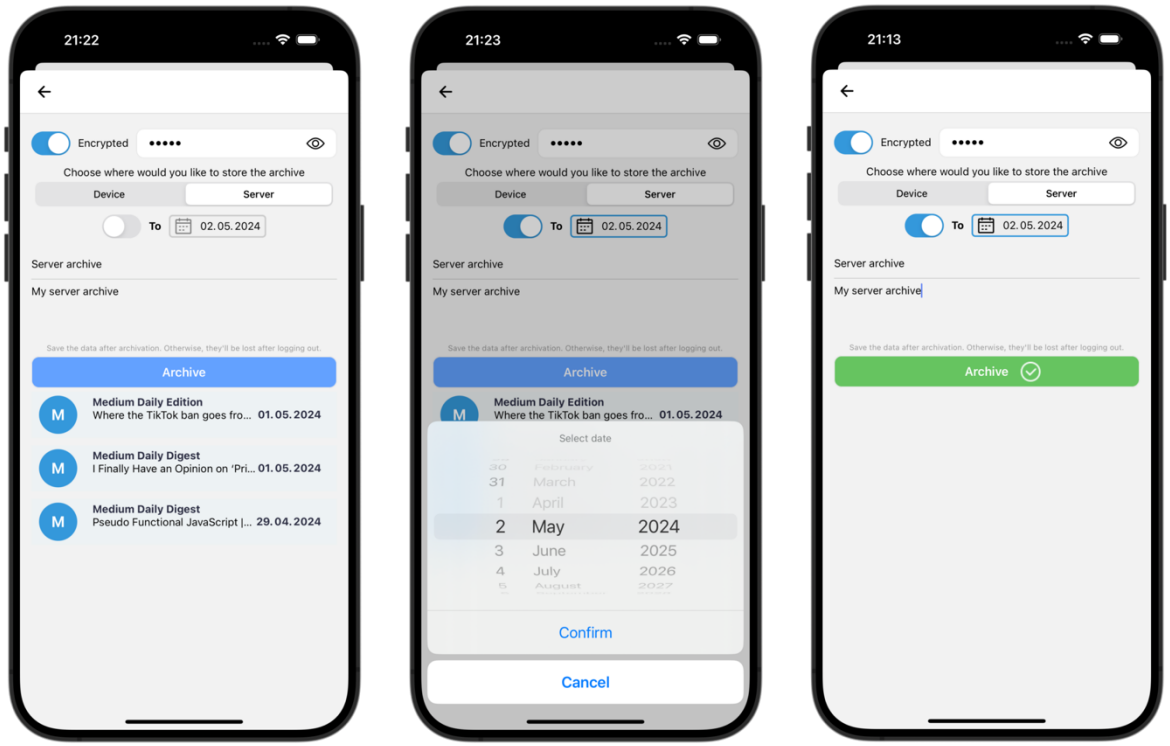
Obrázek 53: Obrazovky klientské pokročilé archivace

11.11.2 Serverová pokročilá archivace

Uchování archívu v databázi lze provést pouze v zašifrované formě, kde tento proces byl navrhnut v sekci 10.7.6. Po přijetí komprimovaných dat se provede proces šifrování, jenž je popsaný v sekci 11.12. Po tomto procesu se zašifrovaná data pošlou na server, kde se uloží v databázi. Proces uchování do databáze je proveden na straně serveru v kontroléru *archive.controller.ts*.

Serverová archivace má ještě navíc funkcionalitu nastavení doby uchování archívu (druhá obrazovka obrázku č. 54). Datumové pole pro nastavení této doby je ošetřeno proti výběru nevalidního data. Minimální možné nastavitelné datum je omezeno na následující den od aktuálního, což zabraňuje uživatelům v nastavení nerealistických nebo chybných termínů pro uchování archivu.

Po úspěšném uložení archivace v databázi jsou archivované emailové zprávy odstraněny ze schránky pomocí IMAP protokolu.



Obrázek 54: Obrazovky klientské pokročilé archivace

11.12 Implementace procesu šifrování dat

Uživatel si může zvolit své heslo na zašifrování dat, to je však potřeba převést na binární klíč o délce 256 bitů. To je docíleno pomocí KDF algoritmem PBKDF2 a náhodné binární hodnoty *„salt“* o délce 256 bitů. Funkce pro kryptografické účely jsou implementované na straně klienta v souboru *encryption.ts* nacházející v repositáři *utils*.

Obrázek 55: Schéma pro převedení hesla na binární klíč

Po vytvoření klíče je vygenerována binární hodnota IV o délce 128 bitů, která se použije při procesu šifrování. Pro šifrování je použitá symetrická šifra AES pracující s délkou klíče o velikost 256 bitů a s režimem blokových šifer CTR.

Obrázek 56: Schéma pro šifrování archívu

Pro ověření správnosti zadaného hesla při procesu dešifrování, je vypočítaná hodnota *„MAC code“* s funkcí HMAC využívající hash algoritmus SHA-256. Tato hodnota se vypočítá z již zašifrovaných dat s klíčem použitým při procesu šifrování.

Obrázek 57: Schéma pro vytváření MAC kódu.

11.13 Implementace obnovy archívů

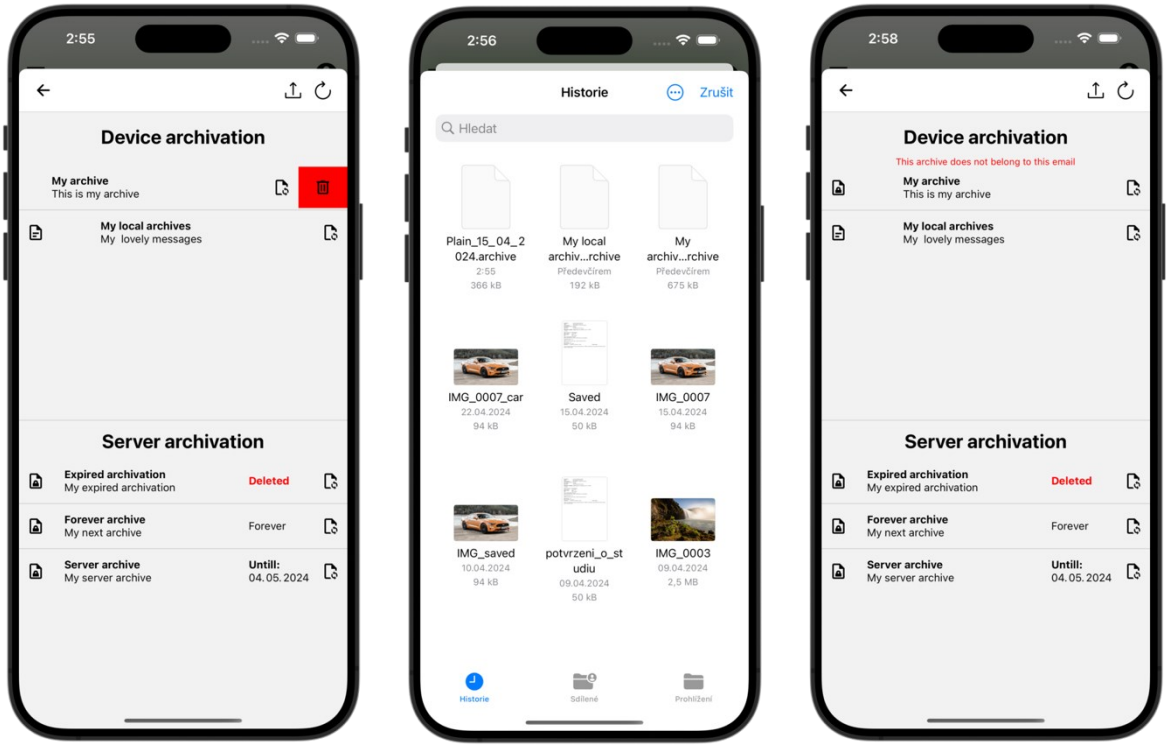
Obrazovka pro obnovu archívu je implementovaná v komponentně *RestoreScreen.tsx*. Na této obrazovce lze provést obnovu jak archívů uložených na zařízení, tak i těch uložených v databázi.

11.13.1 Obnova lokální archivace

Proces pro obnovu lokální archivace byl navrhnut v sekci 10.7.7. Lokální archívy jsou uložené v cache paměti již po jejich archivaci. Ty však jdou z aplikace odstranit pomocí tlačítka pro odstranění. Zobrazení tohoto tlačítka je zhotovené pomocí gesta prstem, a to posunutí jedné položky směrem do leva (první obrazovka na obrázku č. 58). Pokud uživatel archív uložil do souborového systému, tak jej lze nahrát zpět do aplikace (druhá obrazovka na obrázku č. 58).

Ošetření před nevalidním nahrání archívu je provedené následovně: Lze nahrát pouze soubory pouze s příponou „*.archive*“. Dále lze nahrát archív pouze na připojený emailový účet, ze kterého byl archív vytvořen. Toho je docíleno tak, že je při vytváření archívu vypočítaná hash hodnota z emailové adresy připojeného emailové účtu. Při nahrávání je vypočítán hash z aktuálního připojeného účtu a následně porovnáván s hash hodnotu připojenou k archívu.

Při pokusu o nahrání nevalidního souboru, či archívu z jiného emailového účtu, je zobrazená chybová hláška v části pro lokální archivaci (třetí obrazovka na obrázku č. 58).

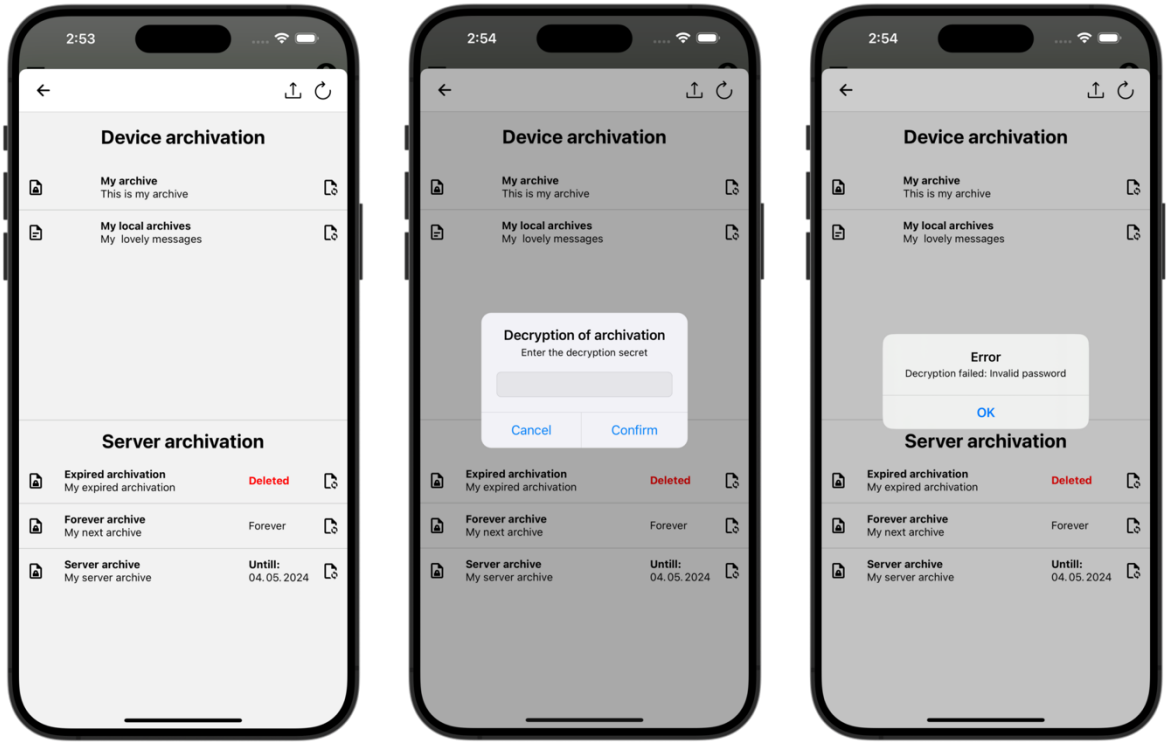


Obrázek 58: Obrazovky obnovy lokální archivace

Při stisknuti pro obnovu nezašifrovaného archívu jsou data poslána na server, kde se provede dekomprese. Následně jsou emailové zprávy přidané, pomocí IMAP protokolu, do korespondující schránky. Funkce pro dekompresi je implementována v souboru *compression.ts*, jenž

je v repositáři *utils*. Přidání archívu do emailové schránky je zhotovené v kontrolérů *imap.controller.ts*.

V případě, kdy je archív zašifrovaný, tak je uživateli zobrazen formulář pro zadání hesla (druhá obrazovka na obrázku č. 59). Pokud je zadané heslo špatné, tak se zobrazí chybová hláška (třetí obrazovka na obrázku č. 59). Po zadání správného hesla, je zahájen proces dešifrování, který je popsán v sekci 11.14.



Obrázek 59: Obrazovky pro zadání hesla k archívu

11.13.2 Obnova serverové archivace

Serverové archívy, které jsou uložené v databázi, jsou zobrazené v druhé polovině obrazovky. Proces tohoto typu archivace byl navrhnut v sekci 10.7.8. Při otevření obrazovky se provede požadavek pro získání těchto archívu ze serveru. Požadavky pro nahrání a získání serverových archívu jsou implementovány v kontroléru *archive.controller.ts*. Tento požadavek lze provést i manuálně s tlačítkem *„load“*, jenž je v záhlaví obrazovky.

Při získávání archívů je na serveru vypočítaná hash hodnota z aktuálně vybraného emailového účtu a porovnávána s hash hodnotou uloženou společně s každým archívem. Tato kontrola je prováděná, aby se například nenahrál archív z Google učtu A do Google účtu B. Tohle by mohlo nastat, kdyby si uživatel připojil jiný účet u stejného providera. Protože kolekce *archives* obsahuje pole archívu pro poskytovatele Google a Azure (Outlook).

Archívy se získávají na základě vybraného emailového účtu spadajícího pod konkrétního poskytovatele.

Následně jsou porovnávané datumy uchování, pokud je toto datum menší než aktuální, tak je archív z databáze smazaný. Archívy jsou mazaný až při jejich získávání, protože by při zautomatizovaném procesu mazání mohl uživatel emailové zprávy trvale ztratit.

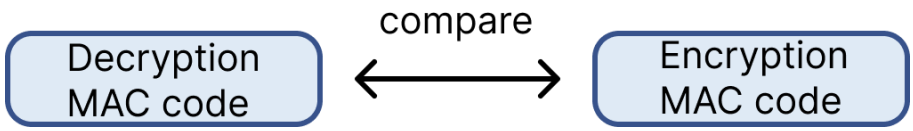
Jednotlivé položky serverové archivace, zobrazené s aplikaci, pak s sebou mají ukazatel statusu o doby uchování. Pokud nebyla nastavena doba pro uchování, tak je položka označená jako *„Forever“*. Na druhou stranu, pokud tato doba byla nastavena, tak je zobrazen datum do kdy bude archív uchován. Jestliže je datum překročen a provedlo se smazání archívu z databáze, tak je položka označená jako *„Deleted“*.

Pro obnovu archívu je uživatel vždy dotázán pro zadání hesla pro dešifrování, protože serverové uložení lze provést pouze v zašifrované podobě. Proces dešifrování je popsán v sekci 11.14. Po dešifrování jsou komprimované archívy zaslány na server, kde se provede jejich dekomprese. Poté jsou emailové zprávy přidané pomocí IMAP protokolu do emailové schránky.

11.14 Implementace procesu dešifrování dat

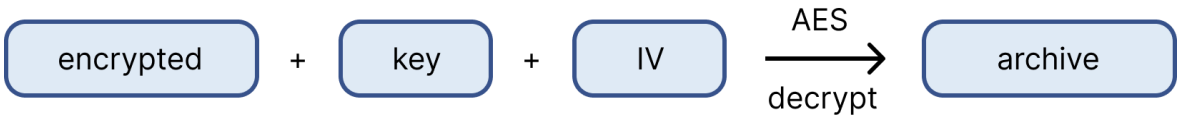
Zašifrované archívy jsou uloženy společně s hodnotami: *„salt“* pro odvození klíče z hesla, *„IV“* pro proces dešifrování a *„MAC code“* pro ověření správností klíče. Po zadání hesla je potřeba převést tuto hodnotu na binární klíč, tato operace je popsaná v sekci 11.12. a vyobrazená na obrázku č. 55. Tato operace je následována procesem vypočítáním hodnoty „*MAC code*“, kde schéma je ukázáno na obrázku č. 57.

Následně je porovnáván MAC code vypočítaný při procesu šifrování s vypočítaným při procesu dešifrování. Pokud se tyto hodnoty nerovnají, tak je uživatel upozorněný o nesprávném zadaném hesle.



Obrázek 60: Schéma pro porovnávání hodnot *„MAC code“*

Jestliže se *„MAC code“* hodnoty rovnají, tak je provedena operace pro dešifrování. Pro dešifrování se použije odvozený klíč a uložená hodnota *„IV“*. Výsledkem dešifrování je pak komprimovaný archív, který se odešle na server pro další zpracování.

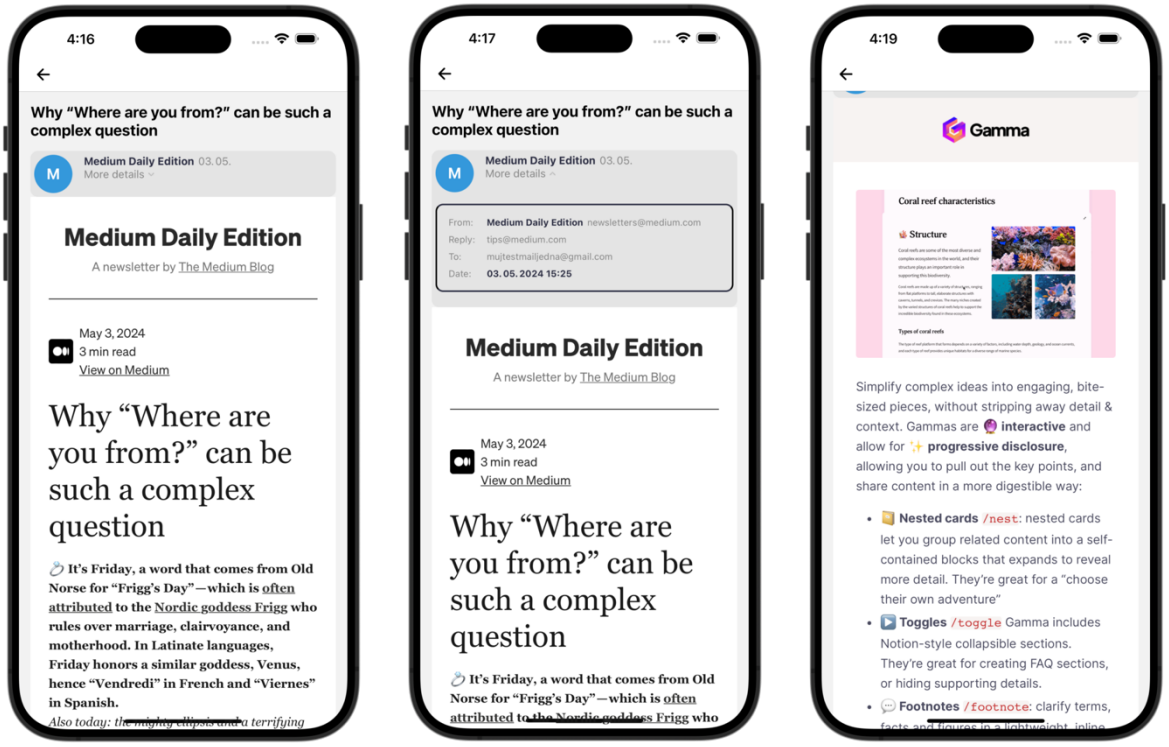


Obrázek 61: Schéma pro dešifrování archívu

11.15 Implementace čtení emailové zprávy

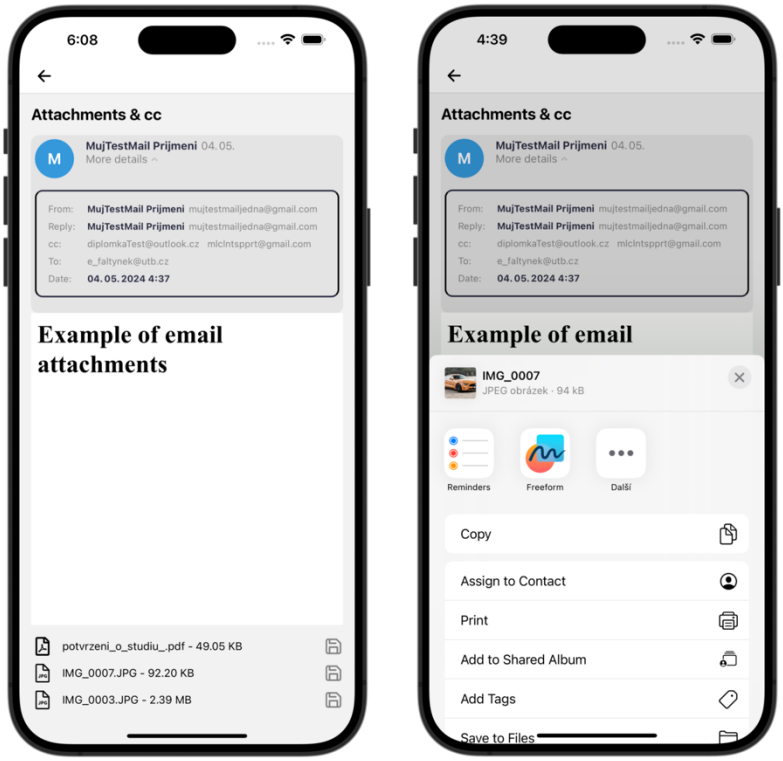
Čtení emailových zpráv je implementovaný v komponentě *EmailDetailScreen.tsx*. Po otevření emailové zprávy je proveden požadavek na server, který vykoná IMAP operaci pro získání celé zprávy včetně příloh. Jakmile klientská strana získá tato data, tak jej uloží do cache paměti. Zároveň je poslán IMAP požadavek pro označení zprávy jako „přečtená“, kde se toto označení aktualizuje i lokálně. Emailová zpráva je ve formě HTML kódu, který je vykreslený do *WebView*. Pro správné vykreslení emailové zprávy bylo zapotřebí zhotovit injekci JavaScript kódů do *WebView*. Vložený skript, mimo nastavení správné šířky dle rozlišení zařízení, také aktivuje odkazy pro přesměrování na webové stránky.

Informace o emailové zprávě, které lze vidět v horní částí obrazovky (obrázek č. 62), jsou již uloženy v aplikaci. Tato data totiž byla získána při výpisu emailových zpráv v obrazovce mailové schránky. Při rozbalení více detailů o emailové zprávě, jsou jednotlivé položky v *„horizontal scroll“*. Tento horizontální posun umožňuje přečíst celý řádek na zařízení s malým rozlišením. Je to obzvláště užitečné pro řádek obsahující emailové adresy o respondentech, kteří obdrželi kopii zprávy (obrázek č. 63).



Obrázek 62: Čtení emailové zprávy

Pokud emailová zpráva obsahuje přílohy, tak jsou zobrazeny v zápatí obrazovky. Zobrazení a jejich možnost uložení do souborového systému je implementováno v komponentě *AttachmentSave.tsx*.



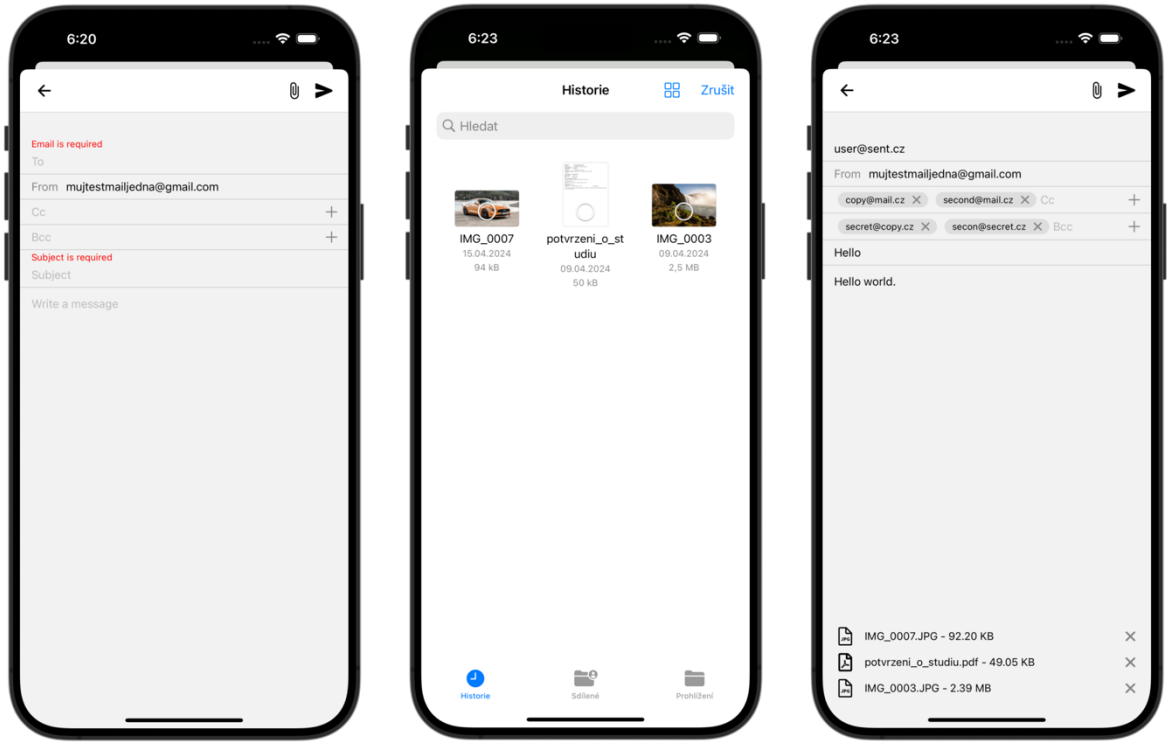
Obrázek 63: Přílohy emailové zprávy

11.16 Implementace odeslání emailové zprávy

Odesílání emailové zprávy je zhotovené na straně klienta v komponentě *WriteEmailScreen.tsx*. Povinná pole jsou pouze pro emailovou adresu adresáta a předmět zprávy. Ošetření těchto vstupních polí je na straně klienta (první obrazovka obrázku č. 64) a také na straně serveru v souboru *smtp.ts*, který se nachází v repositáři *middlewares/validation.*

Speciální vstupní pole jsou *„cc“* a *„bcc“*, který je zhotoveno jako přidání položky emailové adresy adresátu (třetí obrazovka obrázku č. 64). Tato molekulární komponenta je implementovaná v souboru *ItemInput.tsx*.

Je také umožněno nahrávat přílohy pomocí ikony v záhlaví. Po její stisknutí se otevře systémové okno pro nahrání jedné, čí více souborů ze souborového systému zařízení (druhá obrazovka obrázku č. 64). Následně se vybrané přílohy zobrazí v zápatí obrazovky s ikonami dle typu souboru (třetí obrazovka obrázku č. 64). Vykreslování příloh v zápatí je zhotoveno v komponentě *AttachmentViewer.tsx*. Po úspěšném odeslání emailové zprávy, se provede atomické přesměrování zpět na obrazovku schránky.



Obrázek 64: Obrazovky pro odeslání emailové zprávy

Po odeslání emailové zprávy v aplikaci, jsou data poslány na server ke zpracování a odeslání pomocí SMTP protokolu. Zpracování tohoto požadavku je vytvořené v kontroléru

smtp.controller.ts. Implementace SMTP protokolu je zhotovené pomocí knihovny *nodeilaer*. Instance této knihovny je vytvořena v souboru *smtpTransporter.ts* (obrázek č. 64).

```
import nodemailer from "nodemailer"
import { LoginProviders } from "../config/oAuthConfig";
import { smtpConfig } from "../config/smtpConfig";

export const createSmtpTransporter =
(provider: LoginProviders, email: string, accessToken: string): nodemailer.Transporter  => {

  const configuration = smtpConfig[provider];

  const transporter = nodemailer.createTransport({
    host: configuration.smtpHost,
    port: configuration.smtpPort,
    secure: false,
    auth: {
      type: 'OAuth2',
      user: email,
      accessToken: accessToken,
    },
    from: email,
  });

  return transporter;
}
```

Obrázek 65: SMTP *transporter*

Při vytváření instance pro *transporter* je potřeba adresa a port SMTP serveru poskytovatele. Tyto informace jsou definované v konfiguračním souboru *smtpConfig.ts* (obrázek č. 66)

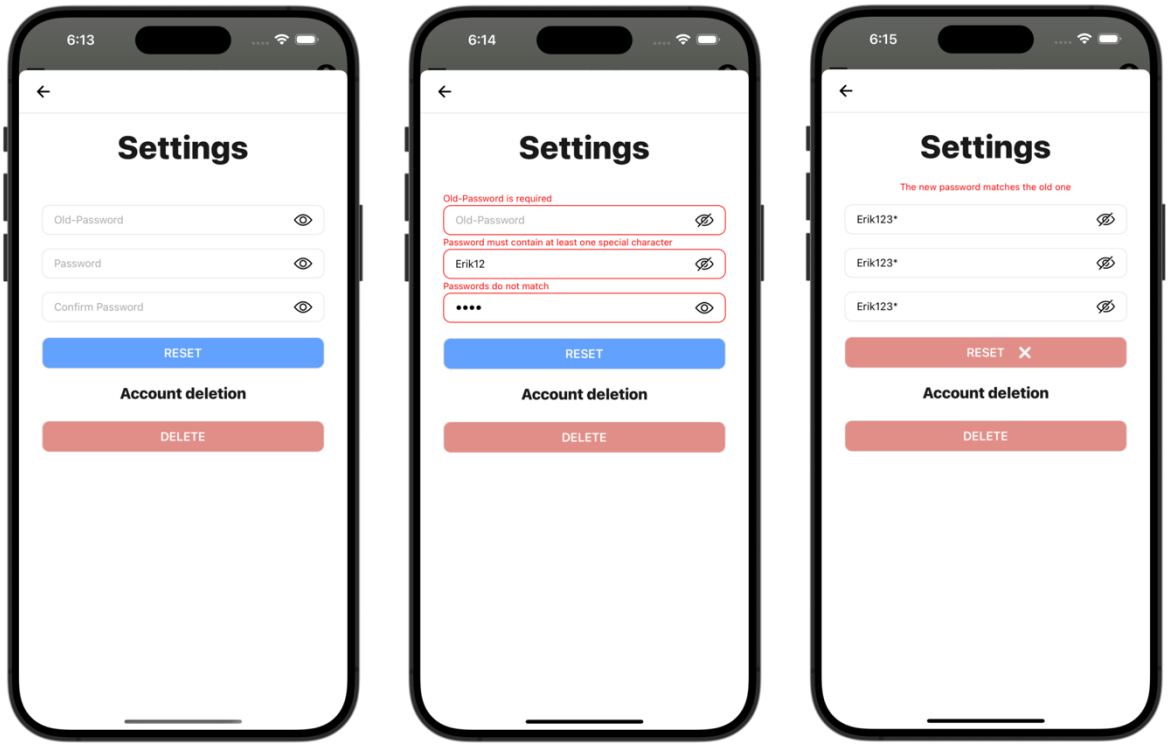
```
export const smtpConfig: Record<LoginProviders, SmtpStructure> = {
  google: {
    smtpHost: 'smtp.gmail.com',
    smtpPort: 587,
  },
  azure: {
    smtpHost: 'smtp-mail.outlook.com',
    smtpPort: 587,
  }
}
```

Obrázek 66: Konfigurační soubor pro SMTP protokol

11.17 Implementace uživatelského nastavení

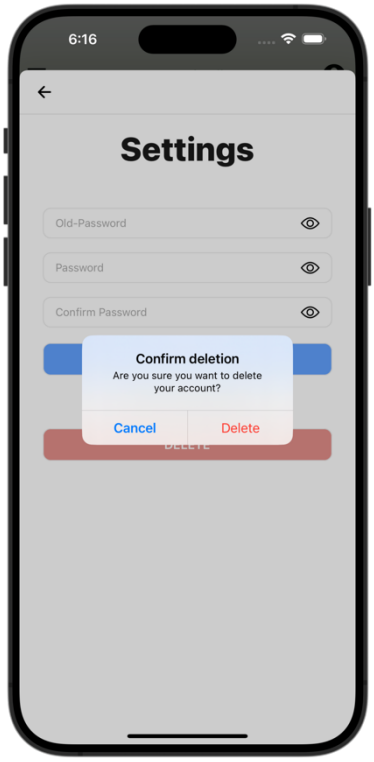
Uživatelské nastavení obsahuje formulář pro nastavení nového hesla a tlačítko pro odstranění účtu. Formulář je ošetřený na klientské straně a také na straně serveru v souboru *user.ts* ve složce *middlewares/validations*. Navíc je přidaná podmínka, že nového heslo se nesmí

rovnat tomu starému (třetí obrazovka obrázku č. 67). Kód obrazovky se nachází v komponentě *SettingsScreen.tsx* a implementace pro zpracování požadavku na straně serveru v kontroléru *user.controller.ts*.



Obrázek 67: Obrazovky uživatelského nastavení

Při stisku tlačítka pro odstranění účtu, je uživateli zobrazeno „pop-up“ okno pro potvrzení o odstranění účtu (obrázek č. 68). Při odstranění jsou odstraněny uživatelovi záznamy v databází a na zařízení je promazána cache paměť.



Obrázek 68: Formulář pro potvrzení odstranění uživatelského účtu

11.18 Bezpečnost

V této sekci budou popsaný veškerý bezpečnostní prvky použitý pro zvýšení míry zabezpečení. Bezpečnost patří ke klíčovým aspektům vývoje aplikací, protože je důležité chránit citlivá uživatelská data.

- **HTTPS**: Komunikace je mezi klientem a server zabezpečena pomocí protokolu HTTPS, který zajišťuje šifrovaný přenos dat.
- **OAuth 2.0**: Připojení emailových účtu je implementováno s využitím *Authorization Code Grant* společně s mechanismem PKCE. Tato kombinace zvyšuje bezpečnost tím, že minimalizuje riziko úniku autorizačních tokenů.
- **Argon2id**: Hesla v databázi jsou uložena v hash podobě. Tyto hodnoty jsou vytvořený za pomocí KDF algoritmu Argon2id.
- **MongoSanitize**: Jedná se o *middleware* pro Express aplikace, který slouží pro odstranění škodlivých kódu z uživatelských vstupů před jejich zpracováním. Je to efektivní ochrana před *„noSQL injection“*, kdy by útočník mohl využívat uživatelské vstupy k manipulaci s databázovými dotazy.
- **Autentizace**: Autentizace probíhá pomocí JWT tokenů, kde přístupový token má krátkou životnost a je potřeba ho obnovovat s obnovovacím tokenem.

- **SecureStore**: Zmíněné autentizační tokeny, jsou v mobilní aplikaci ukládaný v bezpečném šifrovaném uložišti.

12 NASAZENÍ APLIKACE

Tato kapitola se bude věnovat nasazení serverové části a sestavení mobilní aplikace. Budou ukázaný konfigurační soubory a spouštěcí skripty potřebné pro tyto účely.

12.1 Nasazení serverové části

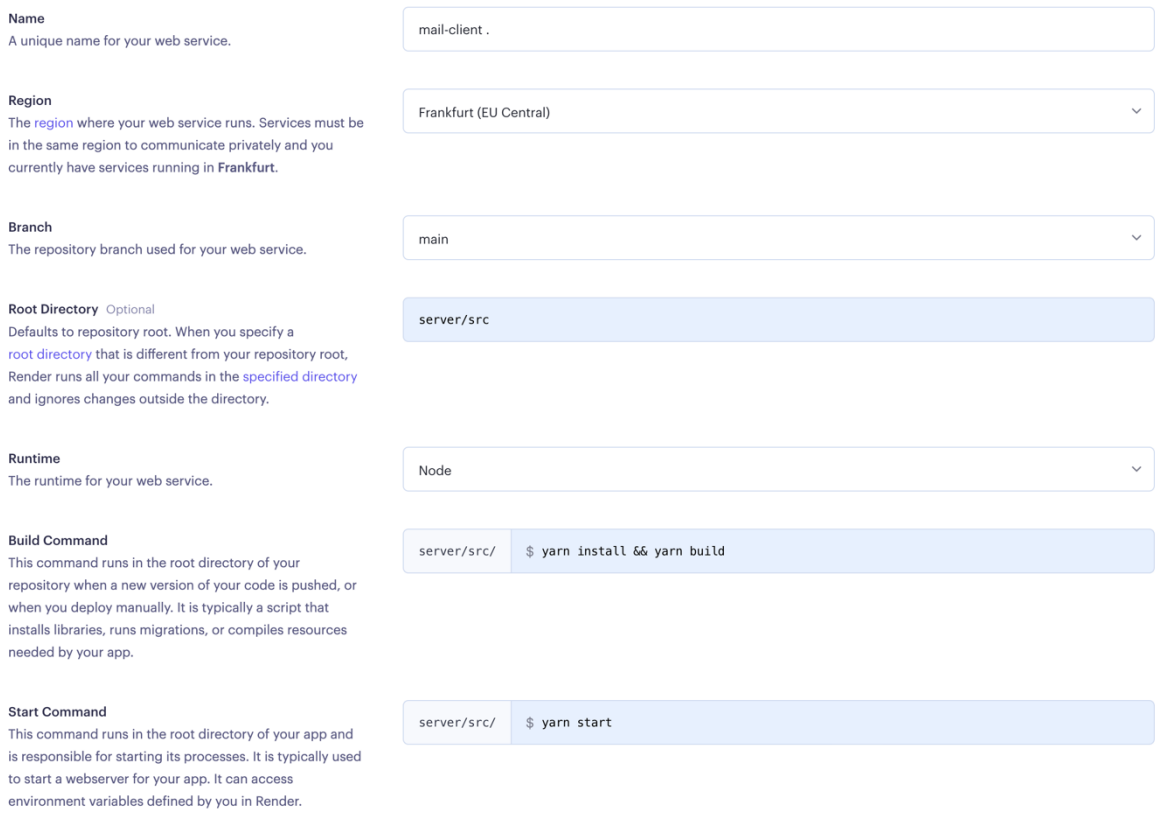
Serverová část napsána v Express.js byla nasazená na cloud platformu Render. Nasazení bylo provedeno s připojením GitHub repositáře do platformy Render.

Před samotným nasazením však bylo potřeba připravit spouštěcí skripty do konfiguračního souboru *package.json*. Jelikož je projekt napsaný v jazyce TypeScript, tak je potřeba tento kód převést do jazyka JavaScript. O tuto proceduru se stará se stará skript *„build“*, kde výsledný build se provede do složky *dist/*. Skript s názvem *„start“* je následně určený pro spuštění převedeného kódu.

```
"scripts": {
  "dev": "ts-node-dev --respawn --transpile-only src/app.ts",
  "start": "node dist/app.js",
  "build": "tsc"
},
```

Obrázek 69: Spouštěcí skripty serverové části

Poté již stačilo jen napojit GitHub repositář, nastavit cestu ke kořenové složce serveru a napsat spouštěče připravených skriptů, jenž lze vidět na obrázku č. 70.



Obrázek 70: Nasazování serveru na platformě Render

12.2 Sestavení mobilních aplikací

Pro sestavení Expo React Native aplikací existuje nástroj EAS (Expo Application Services). EAS je cloudová služba, která také umožňuje vydání aplikace v obchodě s aplikacemi a následně jí i aktualizovat.

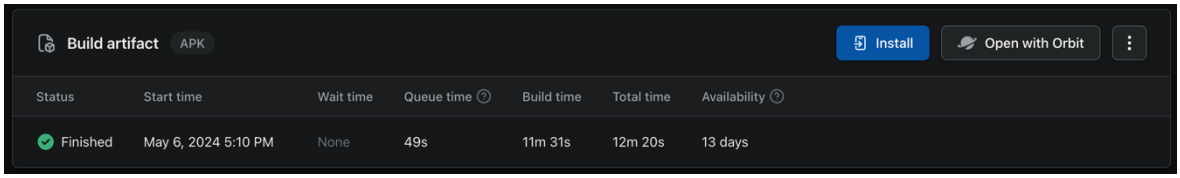
Pro definování různých sestavovacích profilů slouží konfigurační soubor *eas.json* na obrázku č. 71. Do toho souboru lze definovat specifické konfigurace, například pro testovaní, produkci a vývoj.

```
{
  "build": {
    "preview": {
      "distribution": "internal",
      "android": {
        "buildType": "apk"
      },
      "ios": {
        "simulator": false,
        "resourceClass": "m1-medium"
      },
      "env": {
        "BACKEND_URI": "https://mail-client-4ock.onrender.com"
      }
    }
  }
}
```

Obrázek 71: Konfigurační soubor *eas.json*

Po definování konfigurací pro sestavení aplikace, stačí zadat do terminálu příkaz *„eas build --profile preview --platform android“*. Pro použití této služby je nezbytné mít založený účet u EAS a být v kořenovém adresáři projektu.

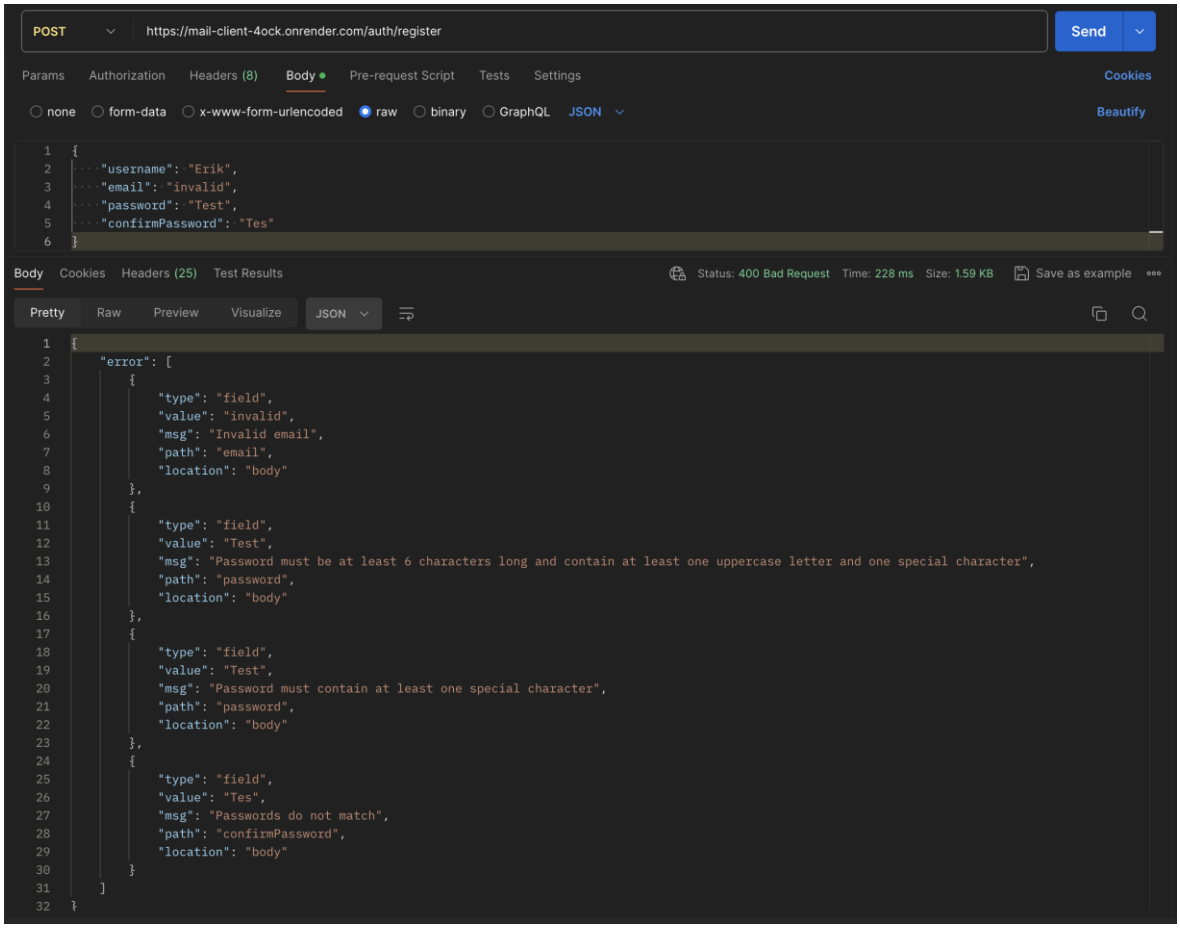
Sestavení aplikace proběhne na vzdáleném serveru služby AES, kde po úspěšném sestavení je vygenerován QR kód. Naskenováním tohoto QR kódu se sestavená aplikace stáhne a nainstaluje do mobilního zařízení.



Obrázek 72: Sestavená aplikace pro platformu Android

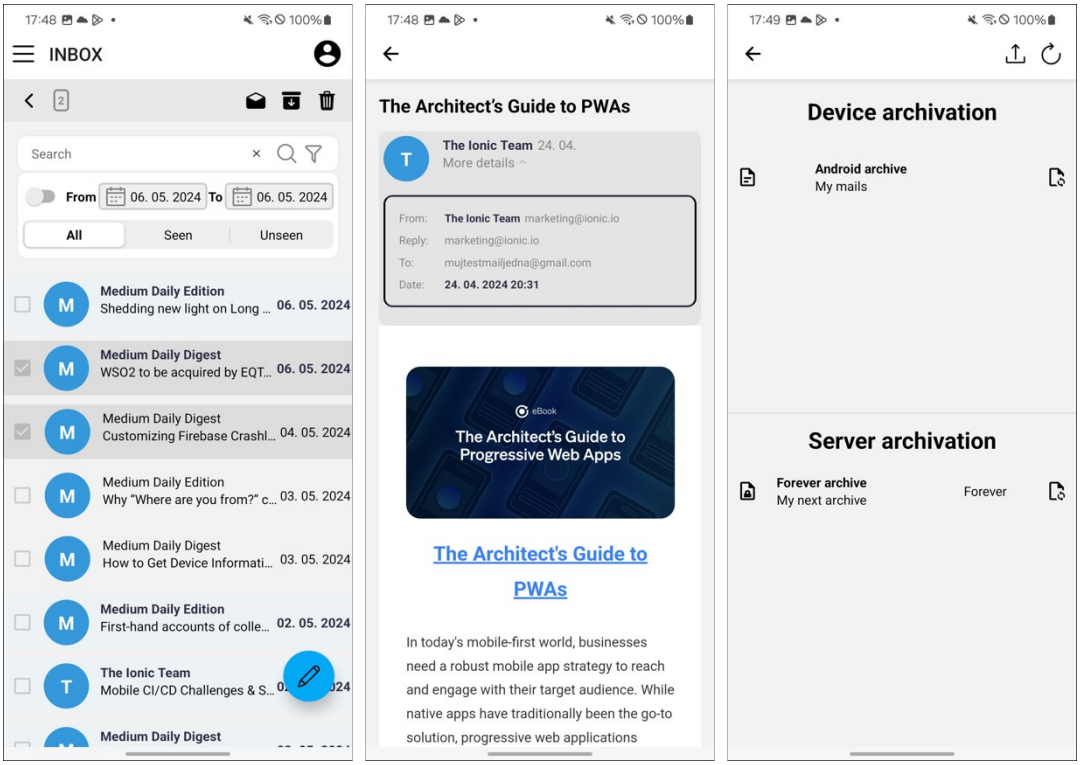
13 TESTOVÁNÍ

Pro aplikaci nebyly zhotovené manuální a ani automatické testy. Testování ale probíhalo v průběhu vývoje. Vytvořené koncové body byly testované v programu Postman, na obrázku č. 73 je například testován registrační koncový bod s chybnými daty.



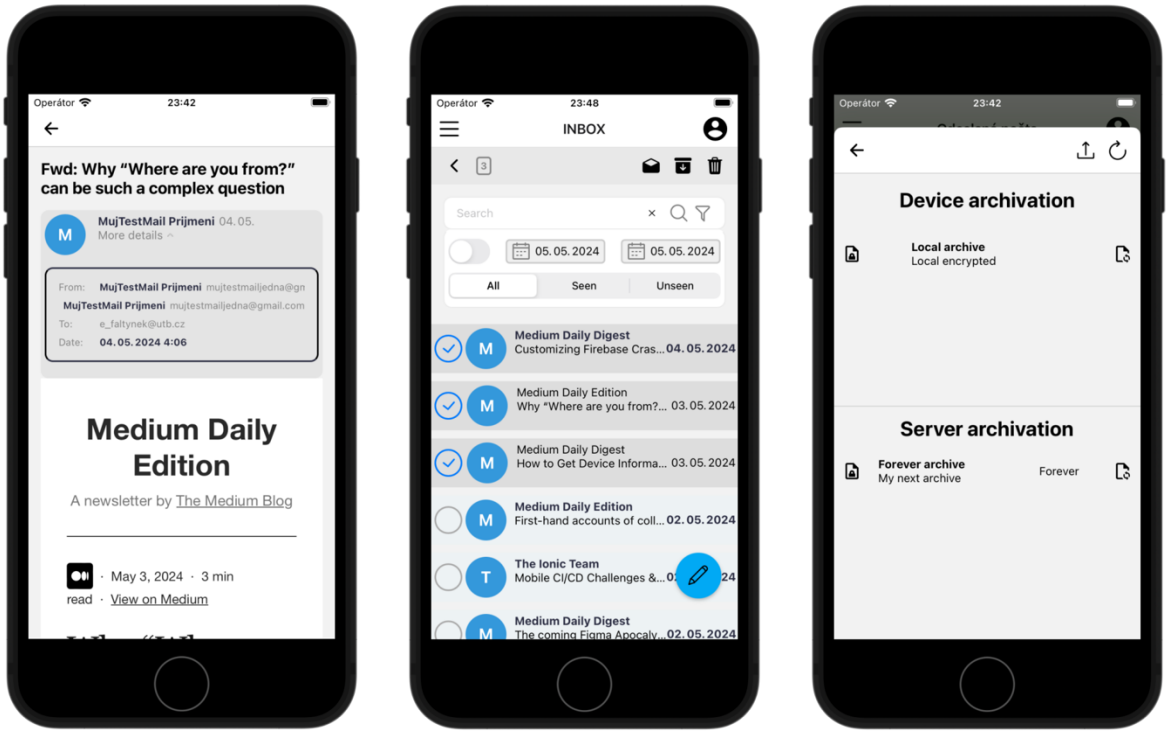
Obrázek 73: Testování koncového bodu pro registraci

Mobilní aplikace byla testovaná na obou platformách. Avšak pro platformu iOS nebylo zhotovené sestavení na fyzické zařízení, protože společnost Apple pro tyto účely vyžaduje placený vývojářský účet. Na obrázku č. 74 lze vidět ukázka testování na fyzickém zařízení platformy Android.



Obrázek 74: Ukázka aplikace na fyzickém zařízení platformy Android

Vývoj probíhal primárně na simulátoru zařízení iPhone 15 pro max. Responzivita byla testovaná na malém zařízení platformy iOS. Na obrázku č. 75 je ukázka výsledné aplikace na zařízení s malým rozlišením.



Obrázek 75: Ukázka aplikace na malém rozlišení

ZÁVĚR

Cílem diplomové práce bylo vytvoření mailového mobilního klienta s možnostmi pokročilé archivace. Výsledkem je implementovaná aplikace, která poskytuje běžné funkce emailového klienta. Přidanou hodnotou této práce jsou možnosti archivace emailových zpráv na základě filtračních kritérií. Archívy lze uložit na zařízení v zašifrované a nezašifrované podobě. V zašifrované podobě je lze uložit i v databázi s možnosti nastavení data ponechání.

V teoretické části byly objasněny základní principy vývoje mobilních aplikaci, důležité emailové protokoly, metody autentizace a autorizace. Dále byla představena symetrická šifra AES, která je vhodná pro účely bezpečného šifrování archívů. K tomu byly představeny bezpečné KDF funkce, pro bezpečné uchování hesel v databázi a odvození klíče z hesla pro potřeby šifrování. Navíc byla i zmíněna metoda pro ověření správnosti hesla pomocí MAC. Závěr teoretické části pak byl věnován popisu použitých technologií pro implementaci mobilní aplikace a serverové části.

Praktická část popisovala návrh a implementaci aplikace. Úvodem praktické části byly popsány funkcionální a nefunkcionální požadavky, ze kterých se následně prováděl návrh aplikace. Poté následoval návrh informační architektury, na který navazoval prvotní návrh uživatelského rozhraní v podobě drátěných modelů obrazovek. Další část návrhu zahrnovala určení datové struktury databáze, komunikační části, optimalizaci provozu a definování rozhraní. Poslední část návrhu aplikace obsahuje definici sekvenčních diagramů, důležitých částí aplikace. Další kapitola se věnuje popisu adresářové struktury a datového modelu výsledné aplikace. Dále je součástí nastínění implementace jednotlivých částí aplikace. Předposlední kapitole je pak věnovaná nasazením serverové části na cloudovou službu Render a sestavení mobilní aplikace na platformu Android. Poslední kapitola obsahuje popis testování aplikace během vývoje.

SEZNAM POUŽITÉ LITERATURY

[1] *What is Mobile Application Development?* Online. In: Aws. 2024. Dostupné z: https://aws.amazon.com/mobile/mobile-application-development/. [cit. 2024-02-28].

[2] RATURI, Gaurav. *Cross-Platform App Development: Bridging the Gap Between iOS and Android*. Online. In: Linkedin. 2023. Dostupné z: https://www.linkedin.com/pulse/cross-platform-app-development-bridging-gap-between-ios-gaurav-raturi/. [cit. 2024-02-28].

[3] GRIFFITH, Chris. *What is Hybrid Mobile App Development?* Online. In: Ionic. 2024. Dostupné z: https://ionic.io/resources/articles/what-is-hybrid-app-development. [cit. 2024-03-02].

[4] CHAUDHARY, Anjali. *Progressive Web Apps vs Native Apps: What Should You Pick?* Online. In: Turing. 2023, aktualizováno 20. 2. 2024. Dostupné z: https://www.turing.com/blog/progressive-web-apps-vs-native-apps/. [cit. 2024-03-02].

[5] STEINER, Thomas. *PWAs in app stores*. Online. In: Web.dev. 2023. Dostupné z: https://web.dev/articles/pwas-in-app-stores. [cit. 2024-03-02].

[6] BUCHA, Swati. *15 Best Email Apps for Android Users in 2024: Compared & Reviewed*. Online. In: Neo. 2024. Dostupné z: https://www.neo.space/blog/best-email-apps-for-android. [cit. 2024-03-06].

[7] TAKAHARA, Jenny. *Email provider vs. email client*. Online. In: Pipedrive. 2023. Dostupné z: https://support.pipedrive.com/en/article/email-provider-vs-email-client. [cit. 2024-03-06].

[8] EMNACE, Hazel. *10 Best Business Email Apps to Use for Work*. Online. In: Fitsmallbusiness. 2023. Dostupné z: https://fitsmallbusiness.com/best-email-apps/. [cit. 2024-03-09].

[9] *Internet Message Format*. Online. In: Loc. 2023. Dostupné z: https://www.loc.gov/preservation/digital/formats/fdd/fdd000393.shtml. [cit. 2024-03-18].

[10] *CRLF*. Online. In: Mozilla. 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/CRLF. [cit. 2024-03-18].

[11] *Email standards*. Online. In: Fastmail. 2024. Dostupné z: https://www.fastmail.help/hc/en-us/articles/1500000278382-Email-standards. [cit. 2024-03-18].

[12] *What is the Simple Mail Transfer Protocol (SMTP)?* Online. In: Cloudflare. 2024. Dostupné z: https://www.cloudflare.com/learning/email-security/what-is-smtp/. [cit. 2024-03-18].

[13] *Understanding the Message Header and Envelope*. Online. In: Barracuda. 2021. Dostupné z: https://campus.barracuda.com/product/emailgatewaydefense/doc/96022980/understanding-the-message-header-and-envelope/. [cit. 2024-03-18].

[14] *What is SMTP?* Online. In: Aws. 2024. Dostupné z: https://aws.amazon.com/what-is/smtp/. [cit. 2024-03-18].

[15] *POP Protocol*. Online. In: Javatpoint. 2021. Dostupné z: https://www.javatpoint.com/pop-protocol. [cit. 2024-03-19].

[16] *IMAP Protocol*. Online. In: Javatpoint. 2021. Dostupné z: https://www.javatpoint.com/imap-protocol. [cit. 2024-03-19].

[17] *Authentication vs. Authorization*. Online. Onelogin. 2023. Dostupné z: https://www.onelogin.com/learn/authentication-vs-authorization. [cit. 2023-11-17].

[18] *Mobile App Authentication Architectures*. Online. In: Owasp MASTG. 2023. Dostupné z: https://mobile-security.gitbook.io/mobile-security-testing-guide/general-mobile-app-testing-guide/0x04e-testing-authentication-and-session-management. [cit. 2024-03-21].

[19] *JSON Web Token Claims*. Online. In: Auth0. 2024. Dostupné z: https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-claims#public-claims. [cit. 2024-03-21].

[20] *OpenID Connect Protocol*. Online. Auth0. 2023. Dostupné z: https://auth0.com/docs/authenticate/protocols/openid-connect-protocol#openid-vs-oauth2. [cit. 2023-11-26].

[21] PONTARELLI, Brian; HASHESH, Ahmed a MOORE, Dan. *What is OAuth (The Modern Guide)*. Online. Fusionauth. 2023. Dostupné z: https://fusionauth.io/articles/oauth/modern-guide-to-oauth. [cit. 2023-11-19].

[22] HARDT ED., D. *RFC6749: The OAuth 2.0 Authorization Framework*. Online. Datatracker. 2012. Dostupné z: https://datatracker.ietf.org/doc/html/rfc6749. [cit. 2023-11-19].

[23] *OAuth 2.0 Grant Types*. Online. In: Vmware. 2023. Dostupné z: https://docs.vmware.com/en/Single-Sign-On-for-VMware-Tanzu-Application-Service/1.14/sso/GUID-grant-types.html. [cit. 2023-11-21].

[24] *The Complete List of OAuth 2 Grants*. Online. Fusionauth. 2023. Dostupné z: https://fusionauth.io/articles/oauth/complete-list-oauth-grants#implicit-grant. [cit. 2023-11-23].

[25] ALTMAN, Matthew a SMITH, Trevor. *OAuth Device Authorization*. Online. Fusionauth. 2023. Dostupné z: https://fusionauth.io/articles/oauth/oauth-device-authorization. [cit. 2023-11-25].

[26] MOORE, Dan. *Storing OAuth Tokens*. Online. Fusionauth. 2023. Dostupné z: https://fusionauth.io/articles/oauth/oauth-token-storage. [cit. 2023-11-26].

[27] NAKOV, Svetlin. *Practical Cryptography for Developers*. Online. Sofia: SoftUni Foundation, 2018. ISBN 978-619-00-0870-5. Dostupné z: https://cryptobook.nakov.com/. [cit. 2024-04-21].

[28] *What Is a Non-Relational Database?* Online. In: Mongodb. 2024. Dostupné z: https://www.mongodb.com/databases/non-relational. [cit. 2024-04-22].

[29] *What is NoSQL?* Online. In: Mongodb. 2024. Dostupné z: https://www.mongodb.com/nosql-explained. [cit. 2024-04-22].

[30] *Introduction to JavaScript*. Online. In: Geeksforgeeks. 2024. Dostupné z: https://www.geeksforgeeks.org/introduction-to-javascript/. [cit. 2024-04-22].

[31] *Introduction to Typescript*. Online. In: Medium. 2022. Dostupné z: https://medium.com/@diego.coder/introduction-to-typescript-6b328184dba1. [cit. 2024-04-22].

[32] *Chapter 1. What Is React Native?* Online. In: Oreilly. 2024. Dostupné z: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html. [cit. 2024-05-07].

[33] BUDZIŃSKI, Maciej. *What Is React Native? Complex Guide for 2024*. Online. In: Netguru. 2024. Dostupné z: https://www.netguru.com/glossary/react-native. [cit. 2024-05-07].

[34] *Expo vs React Native CLI*. Online. In: Medium. 2023. Dostupné z: https://medium.com/@softworthsolutionspvtltd/expo-vs-react-native-cli-7e47c7630039. [cit. 2024-05-07].

[35] *Getting Started*. Online. Axios. 2024. Dostupné z: https://axios-http.com/docs/intro. [cit. 2024-05-07].

[36] *Expo FileSystem*. Online. Expo. 2024. Dostupné z: https://docs.expo.dev/versions/latest/sdk/filesystem/. [cit. 2024-05-07].

[37] *Expo SecureStore*. Online. Expo. 2024. Dostupné z: https://docs.expo.dev/versions/latest/sdk/securestore/. [cit. 2024-05-07].

[38] *React Native Date Picker*. Online. Github. 2024. Dostupné z: https://github.com/henninghall/react-native-date-picker. [cit. 2024-05-07].

[39] *Document picker*. Online. React-native-documents. 2024. Dostupné z: https://react-native-documents.github.io/docs/public/document-picker. [cit. 2024-05-07].

[40] *MMKV*. Online. Github. 2024. Dostupné z: https://github.com/mrousavy/react-native-mmkv?tab=readme-ov-file. [cit. 2024-05-07].

[41] *React-native-quick-crypto*. Online. Github. 2024. Dostupné z: https://github.com/margelo/react-native-quick-crypto. [cit. 2024-05-07].

[42] *React Navigation - A Comprehensive Guide*. Online. In: Theknowledgeacademy. 2024. Dostupné z: https://www.theknowledgeacademy.com/blog/react-navigation/. [cit. 2024-05-08].

[43] SHELDON, Robert a DENMAN, James. *Node.js (Node)*. Online. In: Techtarget. 2022. Dostupné

z: https://www.techtarget.com/whatis/definition/Nodejs. [cit. 2024-05-08].

[44] *What is npm?* Online. In: W3schools. 2024. Dostupné z: https://www.w3schools.com/whatis/whatis_npm.asp. [cit. 2024-05-08].

[45] *Express.js Tutorial*. Online. In: Geeksforgeeks. 2024. Dostupné z: https://www.geeksforgeeks.org/express-js/. [cit. 2024-05-08].

[46] *What is MongoDB – Working and Features*. Online. In: Geeksforgeeks. 2021. Dostupné z: https://www.geeksforgeeks.org/what-is-mongodb-working-and-features/. [cit. 2024-05-08].

[47] HALL, Jesse. *Getting Started with MongoDB & Mongoose*. Online. In: Mongodb. 2022. Dostupné z: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/. [cit. 2024-05-08].

[48] *Compression*. Online. Github. 2024. Dostupné z: https://github.com/expressjs/compression. [cit. 2024-05-08].

[49] *Express Mongo Sanitize*. Online. Github. 2024. Dostupné z: https://github.com/fiznool/express-mongo-sanitize. [cit. 2024-05-08].

[50] *ImapFlow*. Online. Imapflow. 2023. Dostupné z: https://imapflow.com/index.html. [cit. 2024-05-08].

[51] *Jsonwebtoken*. Online. Github. 2024. Dostupné z: https://github.com/auth0/node-jsonwebtoken. [cit. 2024-05-08].

[52] *NODEMAILER*. Online. Nodemailer. 2024. Dostupné z: https://www.nodemailer.com/. [cit. 2024-05-08].

[53] *OAUTH2*. Online. Nodemailer. 2024. Dostupné z: https://nodemailer.com/smtp/oauth2/. [cit. 2024-05-08].

[54] *MAILPARSER*. Online. Nodemailer. 2024. Dostupné z: https://nodemailer.com/extras/mailparser/. [cit. 2024-05-08].

[55] PIKOVER, James. *The Comprehensive Guide to Information Architecture*. Online. In: Toptal. 2018. Dostupné z: https://www.toptal.com/designers/ia/guide-to-information-architecture. [cit. 2024-04-24].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

iOS	iPhone operating system.
SDK	Software development kit.
API	Application programming interface.
UX	User experience.
UI	User interface.
GPS	Global positioning system.
CSS	Cascading style sheets.
HTML	Hypertext markup language.
PWA	Progresive web application.
IMAP	Internet message access protocol.
POP	Post Office Protocol.
CRM	Customer relationship management.
IMF	Internet Message Format.
SMTP	Simple Mail Transfer Protocol.
RFC	Request for Comments.
CR	Carriage-return.
LF	Line-feed.
cc	Carbon copy.
bcc	Blind carbon copy.
TCP	Transmission Control Protocol.
MTA	Mail Transfer Agent.
DNS	Domain Name System.
SMTPS	Simple Mail Transfer Protocol Secure.
SSL	Secure Sockets Layer.

TLS	Transport Layer Security.
USB	Universal Seriál Bus.
PIN	Personal Identification Number.
SMS	Short Message Service.
HTTPS	Hypertext Transfer Protocol Secure.
JSON	JavaScript Object Notation.
JWT	JSON Web Token.
OIDC	OpenID Connect.
ACL	Access Control List.
OAuth	Open Authorization.
ASCII	American Standard Code for Information Interchange.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
ID	Identification.
CSRF	Cross-site request forgery.
PKCE	Proof Key for Code Exchange.
SHA	Secure Hash Algorithm.
SPA	Single-page application.
API	Application Programming Interface.
AES	Advanced Encryption Standard.
CBC	Cipher block chaining.
CFB	Cipher feedback.
OFB	Output feedback.
CTR	Counter.
EAX	Encrypt-then-authenticate-then-translate.
CCM	Counter with cipher block chaining message authentication code.

GCM	Galois/counter mode.
IV	Initialization vector.
MAC	Message Authentication Code.
HMAC	Hash-based message authentication code.
KDF	Key derivation function.
PBKDF2	Password-Based Key Derivation Function 2
JS	JavaScript.
TS	TypeScript.
RN	React Native.
JSX	JavaScript XML.
XML	Extensible Markup Language.
CLI	Command Line Interface.
EAS	Expo Application Services.
JSI	Javascript Interface.
JIT	Just-in-time.
BSON	Binary JSON.
CRUD	Create, Read, Update, Delete.
RSA	Rivest, Shamir, Adleman.
ECDSA	Elliptic Curve Digital Signature Algorithm.
RCE	Remote Code Execution.

SEZNAM OBRÁZKŮ

Obrázek 1: Proces odeslání a stažení emailu pomocí SMTP a POP3 [15] 23
Obrázek 2: Proces odeslání a získání emailu pomocí SMTP a IMAP [16] 24
Obrázek 3: Interakce mezi rolemi v OAuth 2.0 [22] 30
Obrázek 4: Authorization Code Grant Flow [23] 32
Obrázek 5: Implicit Grant Flow [23] 35
Obrázek 6: Resource Owner Password Grant Flow [23] 36
Obrázek 7: Client Credentials Flow [23] 37
Obrázek 8: Obrazovka na zařízení u Device grant [25] 38
Obrázek 9: Uchování tokenů v server-side session [26] 41
Obrázek 10: Informační architektura 61
Obrázek 11: Wireframe přihlašovací obrazovky 62
Obrázek 12: Wireframe registrační obrazovky 63
Obrázek 13: Wireframe obrazovky pro zapomenuté heslo 64
Obrázek 14: Wireframe obrazovky pro resetování hesla 65
Obrázek 15: Wireframe obrazovky bez připojeného emailového účtu 66
Obrázek 16: Wireframe uživatelského menu 67
Obrázek 17: Wireframe obrazovky mailboxu 68
Obrázek 18: Wireframe obrazovky pro přečtení emailové zprávy 69
Obrázek 19: Wireframe pro odeslání emailové zprávy 70
Obrázek 20: Wireframe pokročilé filtrace 71
Obrázek 21: Wireframe panelu nástrojů 72
Obrázek 22: Wireframe obrazovky pokročilé archivace 73
Obrázek 23: Wireframe obrazovky pro obnovu archívu 74
Obrázek 24: Wireframe obrazovky uživatelského nastavení 75
Obrázek 25: Návrh datového modelu 77
Obrázek 26: Proces přihlášení uživatele 87
Obrázek 27: Proces autentizace uživatele 89
Obrázek 28: Proces připojení emailového účtu 92
Obrázek 29: Proces obnovy přístupového tokenu poskytovatele 94
Obrázek 30: Proces lokální archivace 95
Obrázek 31: Proces serverové archivace 97
Obrázek 32: Proces obnovy lokální archivace 98

Obrázek 33: Proces obnovy serverové archivace 100
Obrázek 34: Adresář aplikace.....101
Obrázek 35: Adresář klientské části 102
Obrázek 36: Adresář serverové části 104
Obrázek 37: Struktura datového modelu 106
Obrázek 38: Obrazovky pro přihlášení a registraci 107
Obrázek 39: Koncový bod pro přesměrování do aplikace.....108
Obrázek 40: Obrazovky procesu resetování hesla 109
Obrázek 41: Validace přístupového tokenu.....110
Obrázek 42: Google scopes 111
Obrázek 43 Azure scopes 112
Obrázek 44: Konfigurace pro OAuth proces 113
Obrázek 45: Obrazovky pro připojení emailového účtu.....114
Obrázek 46: IMAP *client*.....115
Obrázek 47: Konfigurační soubor pro IMAP protokol.....115
Obrázek 48: Obrazovky mailové schránky.....116
Obrázek 49: Obrazovky mailové schránky s funkcemi pro načítání 117
Obrázek 50: Obrazovky s filtrací emailových zpráv 118
Obrázek 51: Obrazovky s označováním zpráv jako přečtené/nepřečtené 119
Obrázek 52: Obrazovky s odstraněním emailových zpráv 120
Obrázek 53: Obrazovky klientské pokročilé archivace 121
Obrázek 54: Obrazovky klientské pokročilé archivace 122
Obrázek 55: Schéma pro převedení hesla na binární klíč.....123
Obrázek 56: Schéma pro šifrování archívu.....123
Obrázek 57: Schéma pro vytváření MAC kódu.....123
Obrázek 58: Obrazovky obnovy lokální archivace.....124
Obrázek 59: Obrazovky pro zadání hesla k archívu 125
Obrázek 60: Schéma pro porovnávání hodnot *„MAC code“* 126
Obrázek 61: Schéma pro dešifrování archívu.....127
Obrázek 62: Čtení emailové zprávy.....128
Obrázek 63: Přílohy emailové zprávy 128
Obrázek 64: Obrazovky pro odeslání emailové zprávy.....129
Obrázek 65: SMTP *transporter*.....130

Obrázek 66: Konfigurační soubor pro SMTP protokol 130
Obrázek 67: Obrazovky uživatelského nastavení 131
Obrázek 68: Formulář pro potvrzení odstranění uživatelského účtu 132
Obrázek 69: Spouštěcí skripty serverové části 134
Obrázek 70: Nasazování serveru na platformě Render 135
Obrázek 71: Konfigurační soubor *eas.json* 136
Obrázek 72: Sestavená aplikace pro platformu Android 136
Obrázek 73: Testování koncového bodu pro registraci 137
Obrázek 74: Ukázka aplikace na fyzickém zařízení platformy Android 138
Obrázek 75: Ukázka aplikace na malém rozlišení 138

SEZNAM PŘÍLOH

Příloha P I: Přiložené CD

PŘÍLOHA P I: PŘILOŽENÉ CD

Obsah přiloženého CD:

- Text diplomové práce
- Zdrojové kódy vytvořené aplikace
- Návrh aplikace: Drátěné modely, sekvenční diagramy, kryptografické diagramy, datový model a informační architektura