Analýza architektúry a vylepšenie zdrojového kódu softvéru zimbra

Peter Plevko

Ústav informatiky, informačných systémov a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

December 14, 2022

Abstract

Nápad na túto prácu som získal zo stránky predmetu kde som si pod kolónkou software architecture recovery, pattern detection našiel tento program. Zimbra je open source program používaný na spracovanie mailov, synchronizovanie kalendára, kontaktov, dokumentov a podobne. Používa sa vo firmách všetkých veľkostí. Hlavnou náplňou tejto práce je podrobné preskúmanie softwaru zimbra. Dobrou správou je, že sa jedná o open-source projekt takže je ľahké dostať sa k zdrojovému kódu pomocou githubu. Zároveň sa jedná o veľmi rozsiahly projekt používaný mnohými firmami, tento projekt je používaný po celom svete. V mojej práci sa pozriem konkrétne na jeho architektúru ktorú pomocou diagramov aj vysvetlím a pozriem sa aké design patterny používa. Pozriem sa na každú funkcionalitu a nato aký design pattern bol použitý, ďalej sa pozriem nato prečo je jeho využitie dôležité či už to z hľadiska udržateľnosti kódu systému alebo na zlepšenie výkonu. Nájdené architektúry popíšem aj pomocou diagramov, uvediem výhody a nevýhody. Výsledkom dokumentu bude komplexný pohlaď na architektúru softvéru zimbra a možné spôsoby ako tuto architektúru vylepšiť. Konkrétnejšie sa pozriem na front end a backend tejto aplikácie ako aj na jej bezpečnostnú časť.

1 Úvod

Hlavnou úlohou tohto projektu je vysvetliť softvérovú architektúru programu Zimbra. Rozhodol som sa pre výber tohto programu, pretože to je jeden z najznámejších a najviac používaných softvérov na posielanie sprav a kolaboráciu. V tomto dokumente sa pozrieme na všeobecné informácie o softvéri Zimbra. Rozoberieme si back-end a front-end tejto aplikácie. Ďalej sa pozriem na bezpečnosť tejto aplikácie. Pozriem sa na to aké design patterny sa v tomto softvéri nachádzajú a vysvetlím ich. V neposlednom rade

skúsim vylepšiť zdrojový kód. Konkrétne sa budem venovať nasledujúcim github repozitárom:

1. https://github.com/Zimbra/zm-mailbox
2. https://github.com/Zimbra/zm-admin-console
3. https://github.com/Zimbra/zm-admin-ajax

2 Software Zimbra

V nasledujúcej sekcii si bližšie priblížime tento program. Informácie v tejto sekcii boli získane zo zdroja [1]. Zimbra je software na kolaboráciu obsahuje e-maily, kalendáre, kontakty, dokumenty a ďalšie údaje. Je to balík webových aplikácií, ktorý možno nasadiť na verejný cloud alebo na súkromný cloud. Hlavným cieľom softvéru je integrovať viaceré technológie na kolaboráciu do jedného. Server Zimbra je kompatibilný so systémami Windows, Linux a Apple a podporuje desktopové e-mailové aplikácie, ako je napríklad Windows Outlook. Ponúka aj bezdrôtovú synchronizáciu s mobilnými operačnými systémami vrátane iOS, Windows Mobile, BlackBerry a Android.

3 Komponentový pohlaď na softvér Zimbra

Architektúra Zimbra zahŕňa integrácie s open-source kódom aj od iných firiem. V nasledujúcej sekcii sú rozobrané jednotlivé komponenty ktoré sú čerpané zo zdroja [12].

1. Postfix: je to mail transfer agent (MTA) ktorý smeruje mailové správy na správny Zimbra server
2. Open LDAP software: protokol LDAP (Lightweight Directory Access Protocol) je implementácia ktorá obsahuje globálny zoznam adries Zimbra a zabezpečuje overovanie používateľov. Zimbra môže pracovať aj so službami GAL a autentifikačnými službami poskytovanými externými adresármi LDAP, ako je napríklad Active Directory
3. MariaDB: databázový softvér
4. Anti-virus/anti-spam: antivírusový program ClamAV, ktorý chráni pred škodlivými súbormi SpamAssassin je poštový filter, ktorý vyhľadáva spam
5. Lucene: full-text vyhľadávací engine
6. LMTP: local Mail Transfer Protocol

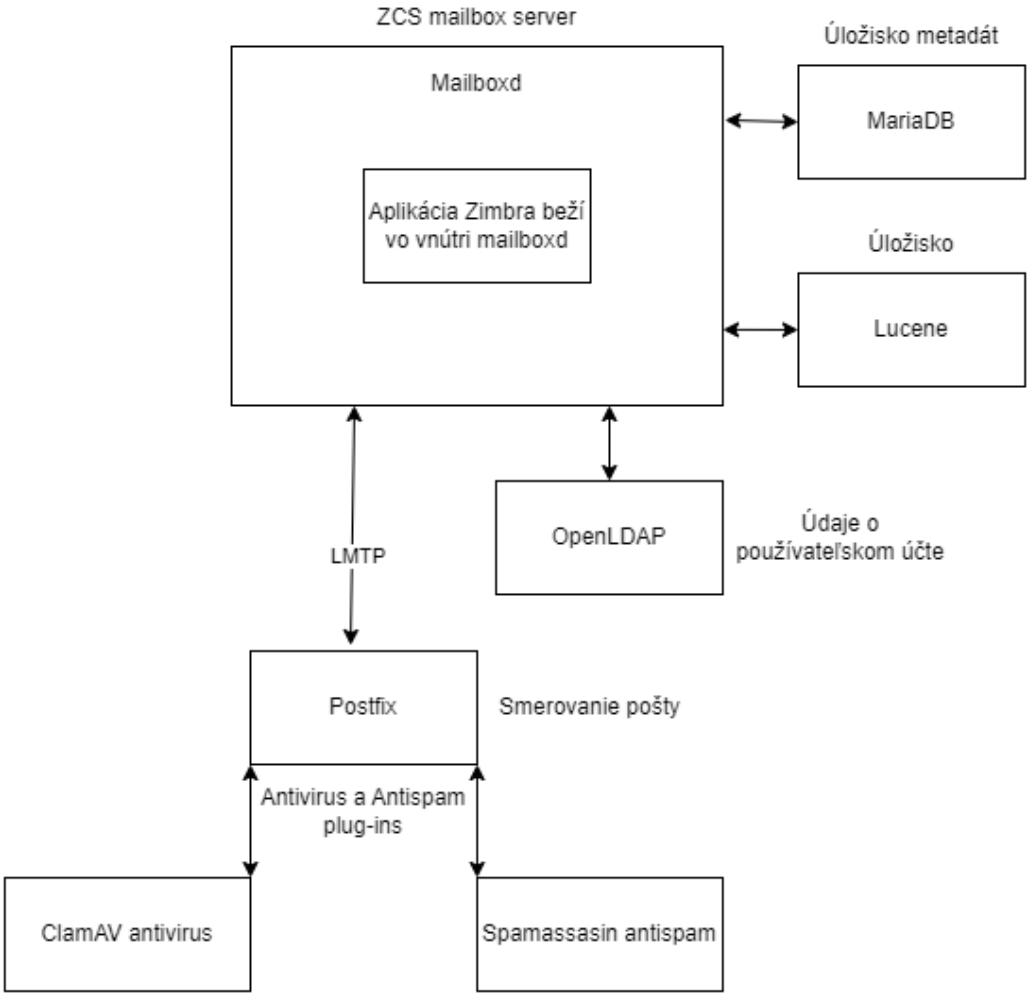

Figure 1: Komponenty [12]

4 Architektúra z pohľadu klienta

Od verzie 9.0.0 Máme k dizpocií dva klienty. Modern Web App - využíva moderné technológie je plne responzívna. To znamená, že je možné si túto stránku zobraziť aj na iných zariadeniach akými sú napríklad stolný počítač, mobil a tablet. Classic Web App – je naprogramovaná v Ajaxe a ponúka všetky funkcie potrebné pre webovú kolaboráciu. Podporuje iba webové prehliadače pre desktopy. Nie je responzívna pre mobil, tablet.

4.1 Moderná webová aplikácia

Informácie o Modernej webovej apke boli získane zo zdroja [6]. Najväčšou zmenou a výhodou oproti klasickej webovej aplikácií je to, že táto moderná verzia je plne responzívna. Keďže sa jedná o novú aplikáciu nie sú v nej ešte pridané všetky veci z tej starej alebo niektoré veci fungujú inak ako v starej. Kvôli tomuto by používateľa mohlo napadnúť zmeniť si typ klienta, to nie je problém pretože vo verzii 9.0.0 táto moderná apka koexistuje s klasickou. To znamená, že používateľ si môže pri prihlásení vybrať ktorú z nich chce použiť. Použité technológie:

1. PreactJS - UI framework založený na komponentoch

2. GraphQL - rozšíriteľné data API založené na datagrafoch

3. Apollo - implementácia jazyka GraphQL s pokročilými možnosťami správy vyrovnávacej pamäte

4.2 Klasická webová aplikácia

Informácie spomenuté v tejto kapitole som čerpal zo zdroja [3]. Používa Ajax (Asynchronous JavaScript And XML). Výhody použitia ajaxu:

1. Bohato interaktívne používateľské rozhranie

2. Bohatá komunikácia klient/server

3. Nulové náklady na správu webového klienta

V momente keď sa Zimbra programovala bolo XML veľmi populárne. V dnešnej dobe XML už pomaly upadá a do popredia sa dostáva novy formát a tým je JSON. Klient Zimbra Ajax je objektovo orientované používateľské rozhranie naprogramované v jazyku JavaScript. V predvolenom klientovi prehliadača Zimbra Ajax nie je žiadna perzistencia. Všetok trvalý stav sa uchováva na serveri. Na štýlovanie používateľského rozhrania sa používa CSS.

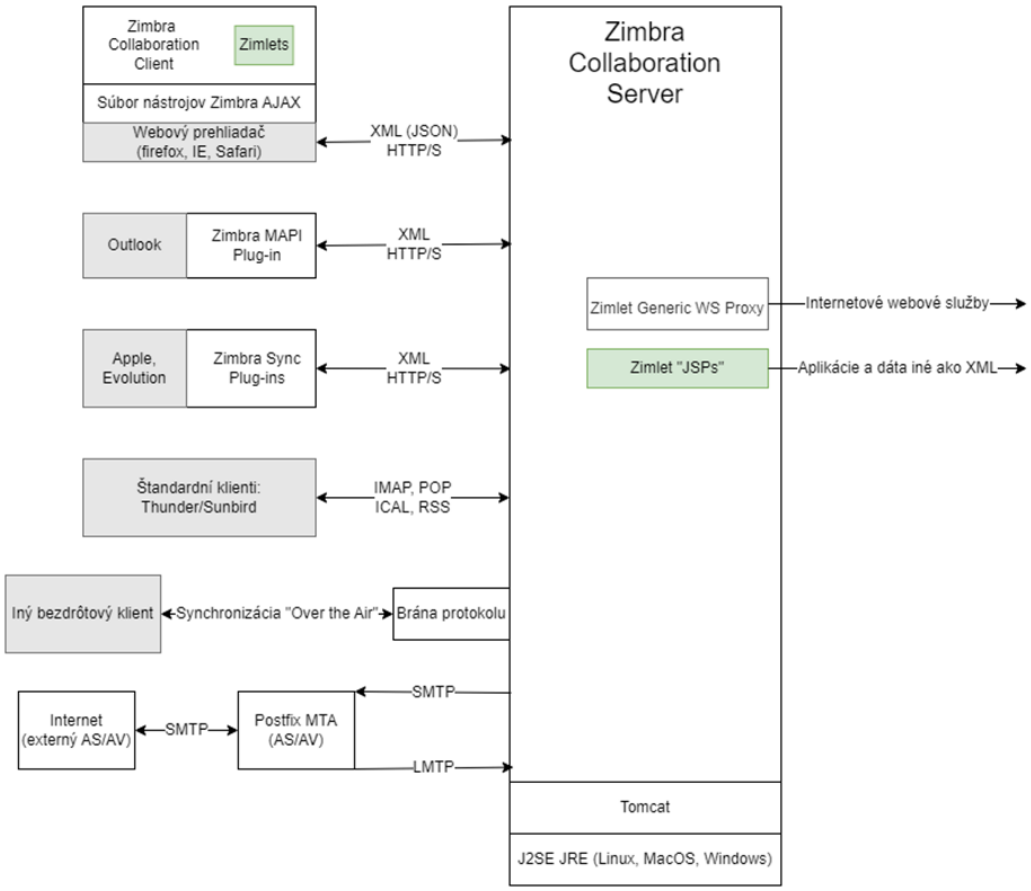

Figure 2: Architektúra klienta [3]

Komunikácia klient-server, programátori môžu slobodne používať akékoľvek služby Zimbra pre zasielanie správ a kolaboráciu. Napríklad posielanie e-mailov alebo organizovanie schôdzí, z akejkoľvek aplikácie podporujúcej webové služby, pretože Zimbra ma dobre zdokumentované endpointy. Na posielanie dát sa vo všeobecnosti používa XML. Avšak toto spracovanie XML nemusí byť pre rýchlosť klienta Ajax vždy najlepšie, preto zimbra ponúka aj možnosť "rýchlej cesty". Pri ktorej server zimbra použije namiesto XML JSON. Tento proces „rýchlej cesty“ je automaticky vybraný ak to je potrebné. Ako je znázornené v obrázku 1. Zimbra podporuje širokú škálu existujúcich aplikácii pre zasielanie správ a kalendárov. Zimbra poskytuje pluginy pre nasledujúce softvéry:

1. Outlook
2. Apple Desktop
3. Novell Evolution

Zimlety predstavujú architektúru widgetov pre integráciu na strane klienta aj na strane servera. Architektúra Zimbra Zimlet umožňuje vývojárom spracovávať data z rôznych zdrojov. Napríklad z internetu a informačných systémov. Takto spracovane dáta je možné v rámci ZCS používať v e-mailových správach, kontaktných poliach a poznámkach v kalendári. Zimlety sú používateľovi sprístupnené prostredníctvom klienta ZCSAjax. Zimlet poskytuje aj možnosť integrácie s vyhľadávacím nástrojom ZCSsearch (napríklad pre indexovanie objektov ako identity zákazníkov). Kľúčovým cieľom architektúry je umožniť implementáciu širokej škály týchto Zimletov. Od tých, ktoré si vyžadujú len malý alebo žiadny vlastný kód až po tie, ktoré využívajú JavaScript v rámci klienta ZCS alebo Javu v rámci servera. Väčšina prípadov Zimletov však nevyžadovala žiadne programovanie. Ich správanie je plne špecifikované prostredníctvom šablón XML.

5 Architektúra z pohľadu servera

Informácie spomenuté v tejto kapitole boli čerpané z [3].

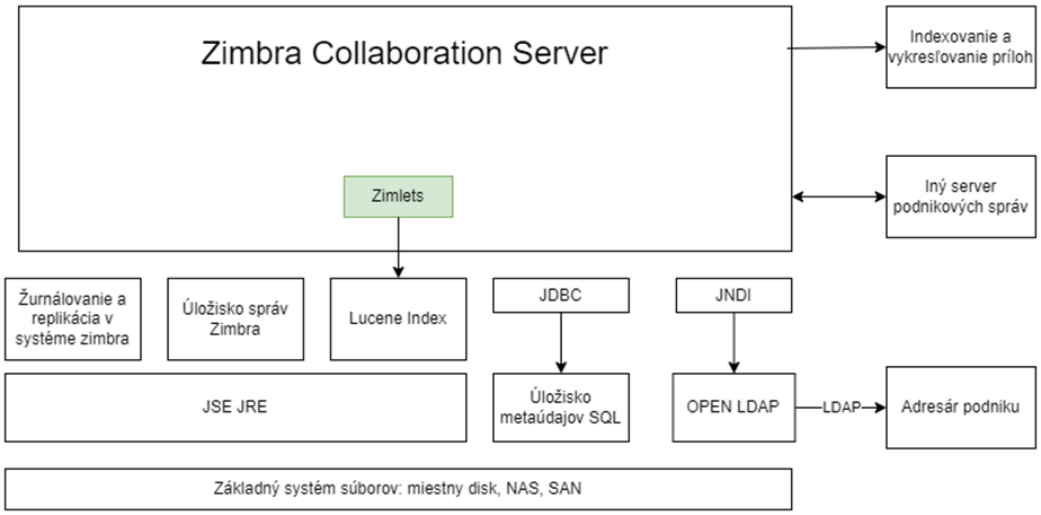

Figure 3: Architektúra servera [3]

Server je napísaný v jazyku Java. Medzi hlavné výhody javy patrí jej dobrá integrácia s webom a dobrá integrácia komponentov, v neposlednom rade výkon.

Archív správ. Základný súborový systém Unix/Linux slúži ako základ pre úložisko správ Zimbra. Pre každú správu v mapovaní sa používa jeden súbor, reprezentácia správy RFC822 MIME sa v skutočnosti zapisuje priamo do súboru. Má to niekoľko výhod:

1. Správy Zimbra sú po zapísaní nemenné a súborové systémy sú vynikajúce na ukladanie a načítanie nemenných neštruktúrovaných/pološtrukturovaných blokov údajov

2. Garbage collection zabezpečuje operačný systém

3. Vlastné možnosti operačného systému, ako je indexovanie/vyhľadávanie (napr, MacSpotlight), kompresia a zabezpečenie

4. Nehovoriac o viacúrovňovom ukladaní do vyrovnávacej pamäte - Zimbra ukladá údaje do vyrovnávacej pamäte sama, ale využíva aj ukladanie do vyrovnávacej pamäte na úrovni operačného systému

Úložisko metadát SQL. Zatiaľ čo samotné správy sú nemenné, metadáta spojené so správou sa menia keď sa používajú. Metadáta o správach zahŕňajú také položky, ako napríklad. V akom priečinku sa správa nachádza. Je správa prečítaná alebo neprečítaná atď. Balík Zimbra Collaboration Suite obsahuje relačnú databázu konkrétne sa jedná o MariaDB na správu metadát mailových schránok. Výhody použitia relačnej DB sú napríklad efektívne vyhľadávanie a spoľahlivá aktualizácia metadát. Relačné databázy poskytujú veľmi efektívne ukladanie do vyrovnávacej pamäte pre čítanie aj zápis.

5.1 Zabezpečenie

Zimbra zabezpečuje údaje vo svojej infraštruktúre pomocou rôznych bezpečnostných techník. Na ochranu komunikácie v sieti Zimbra sa používajú štandardné technológie, ako sú HTTPS, IMAPS, LDAPS a ďalšie. Systém v defaultnom nastavení šifruje všetky správy od začiatku do konca s využitím S/MIME. Certifikačná autorita by mala ponúkať certifikáty pre tieto šifrovacie procesy (CA).

Na boj proti škodlivému softvéru aj nevyžiadanej pošte sa používa third party software s open-source kódom, ako bolo uvedené v časti 2.3.

5.2 Možné implementačné zlepšenia

Takže ako som už vyššie spomínal zimbra sa skladá z dvoch hlavných častí. Prvou je client a druhou server. V dnešnej dobe je veľmi časté, že klient a server sú naprogramované v tom istom jazyku a tým je JS. Napríklad kombinácia klient (Vue.js) server (NestJS). Výhod je veľmi veľa. Ak používame rovnaký jazyk nemusí používateľ meniť IDE. Keďže je klient aj server napísaný v rovnakom jazyku stačí nám teoreticky jeden developer aj na front-end aj na back-end. V tomto prípade môžeme prijímať do zamestnania fullstack developerov takže ušetríme lebo nám treba polovicu zamestnancov oproti tomu keby neprogramujeme v tom istom jazyku. Problém s kompatibilitou taktiež nemôže nastať pretože robíme v tom istom jazyku. Keďže som spomenul len výhody tak aký je dôvod, že klient ani server neboli

napísané v nejakom JS frameworku. Odpoveď je jednoduchá keď sa Zimbra programovala Javascript sa skoro vôbec nepoužíval a jeho frameworky ani neexistovali. Zimbra už je veľmi starý softvér. Ako si možno spomínate v novom klientovi už je použitý JS framework takže aj samotná firma zimbra vidí výhodu v použití frameworku na front-ende.

Čo sa týka backendu ten zatiaľ ešte nebol prepísaný a momentálne je napísaný v jave. Takže medzi výhody javy patri, že je platformovo nezávislá to node v ktorom bežia všetky JS frameworky na servery nie je. Java je oveľa dlhšie na svete to znamená, že má aj viac knižníc a väčšiu komunitu kóderov, to znamená má aj viac fór na stack overflow. Taktiež je Java multi-tread kdežto JS je single thread. Tu som popísal všetky výhody javy aký je teda záver.

Oplatí sa prepísať javu do nejakého JS frameworku akým je napríklad (NestJS) ? Tento prepis by bol veľmi nákladný a zdĺhavý pretože zimbra server už je 20 rokov používaný softvér v ktorom je mnoho funkcionalít. podľa môjho názoru by to však bolo vo výsledku prospešné. Java sa scaluje dobre horizontálne avšak node sa scaluje ešte lepšie. JS ide stále viac a viac do popredia. Takže z hľadiska udržateľnosti by bol dobrý nápad prejsť na tento jazyk. Najväčším argumentom pre tento prechod je však jazyková konzistencia na front-ende a back-ende.

6 Identifikovane Návrhové vzory

6.1 Mvc (Model view controller)

Tento design pattern sa často uplatňuje pri tvorbe používateľských rozhraní a rozdeľuje príslušnú logiku programu na tri vzájomne prepojené komponenty. Týmto spôsobom je možné rozlíšiť vnútorné reprezentácie informácií a metódy, ktorými sa informácie poskytujú používateľom [2]. Slúži Môžeme ho nájsť v súbore zm-mailbox-develop. Informácie použité v tejto sekcii sú zo zdroja [10]. Jedným z najobľúbenejších aspektov systému ZCS je jeho používateľské rozhranie (UI) založené na technológii AJAX. Zimbra implementuje svoju službu webového klienta AJAX pomocou kombinácie návrhových vzorov REST (Representational State Transfer) a MVC (Model View Controller). Návrh REST oslobodzuje server od sledovania stavu webového klienta, zatiaľ čo návrh MVC poskytuje okamžité aktualizácie v reakcii na akcie používateľa. Intuitívny widget a ikony umožňujú používateľom objavovať nové funkcie, ako aj pomocou funkcie drag and drop spravovať udalosti kalendára a e-maily. Pomocou okna možno napríklad vybrať skupinu e-mailov a pretiahnuť ju do priečinka.

Keď servlet služby Zimbra prijme požiadavku, odovzdá túto požiadavku objektu SoapEngine, ktorý ju pomocou handlera spracuje. SoapEngine.java je MODEL a je kompletne oddelený od používateľského rozhrania a manažuje logiku a dáta. Webový klient je VIEW a handlery sú CONTROLLER.

Príkladom handleru je napríklad súbor: DocumentHandler.java Konštrukcia MVC v systéme Zimbra umožňuje vývojárom ľahko rozširovať funkčné vlastnosti rozšírením objektov kontroléra a obslužných objektov systému Zimbra. Ak je napríklad potrebné použiť inú alternatívnu metódu overovania ako predvolenú LDAP od spoločnosti Zimbra, triedu provisioning možno rozšíriť tak, aby bolo možné použiť podtriedu na prepísanie metódy overovania účtu. Podobne možno podobným spôsobom rozšíriť služby pre priečinky, poštové schránky, účty, kalendáre atď. využitím možností objektovej architektúry. Pokiaľ sú dátové objekty odvodené od dátového objektu Zimbra, možno vytvoriť alternatívnu architektúru na použitie iného úložiska pošty alebo správcu kalendára. Tento objektový framework umožňuje neobmedzene rozširovať systém a škálovať ho tak, aby podporoval veľký počet používateľov pomocou viacúrovňovej architektúry.

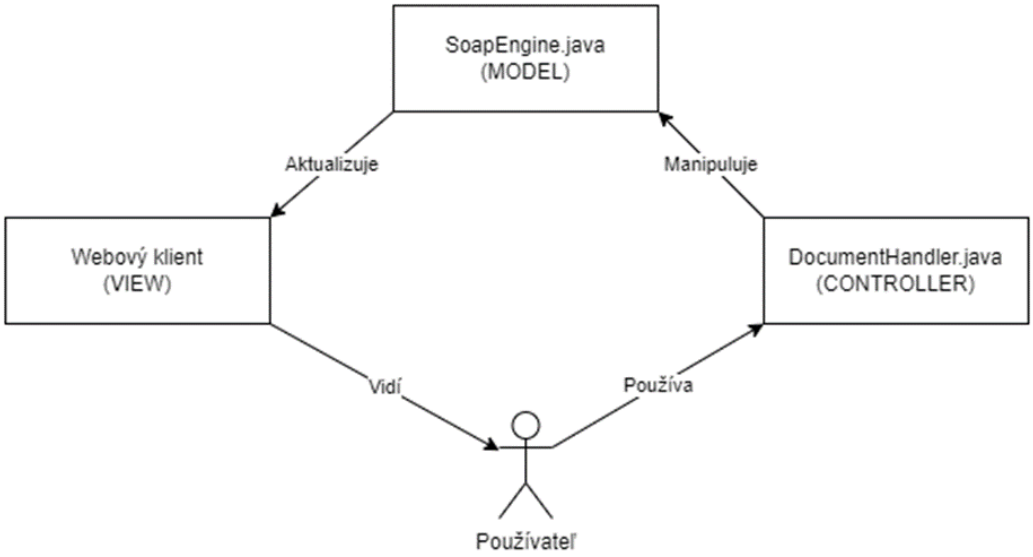

Figure 4: Model-view-controller

6.2 Singleton

Informácie o navrhovom vzore sinlgeton v software Zimbra boli získané z [9]. V tomto vzore je za vytvorenie objektu zodpovedná jedna trieda, pričom sa zabezpečí, že vznikne len jeden objekt. Táto trieda ponúka metódu priameho prístupu k jedinému objektu triedy, čím sa eliminuje potreba vytvorenia jeho inštancie [5]. Tento design pattern môžeme nájsť v dvoch priečinkoch a tými sú zm-admin-console-develop a ZM-ADMIN-AJAX-DEVELOP. Konkrétne sa jedná o súbory ZaMsgDialog.js a DwtMessageDialog.js.

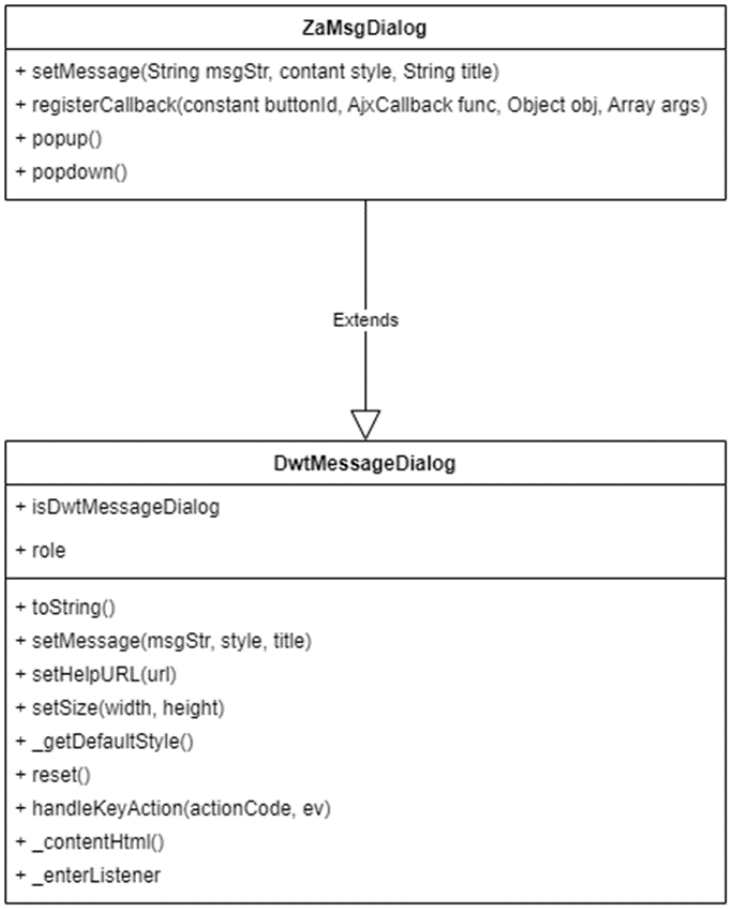

Figure 5: Singleton

DwtMessageDialog je základná trieda pre všetky dialógové okná správ v rámci Zimbra Ajax Framework. ZaMsgDialog, rozširuje DwtMessageDialog. ZaMsgDialog je potrebný na vytvorenie inštancie dialógového okna správ v používateľskom rozhraní admina. Framework administrátorského používateľského rozhrania obsahuje štyri singletonové inštancie ZaMsgDialog, ktoré pokrývajú väčšinu prípadov použitia dialógovych okien správ: msgDialog, errorDialog, confirmMessageDialog a confirmMessageDialog2. Vďaka týmto inštanciám nemusíme vytvárať vlastnú inštanciu ZaMsgDialog. Pri zobrazovaní dialógového okna správ používateľovi by sme mali vždy počítať s použitím týchto inštancií, aby sme zachovali efektívnosť používateľského rozhrania administrátora a zabránili zväčšovaniu jeho pamäťovej kapacity. Prostredníctvom triedy ZaApp možno k týmto inštanciám pristupovať nasledovne:

1. ZaApp.getInstance().dialogs[”errorDialog”]

2. ZaApp.getInstance().dialogs["confirmMessageDialog"]

3. ZaApp.getInstance().dialogs["confirmMessageDialog2"]

4. ZaApp.getInstance().dialogs["msgDialog"]

6.3 Factory method

Tento design pattern sa nachádza v priečinku Zm-admin-ajax-develop konkrétne v súbore XFormItem.js. Je to design pattern používaný v class-based programovaní na riešenie problému vytvárania objektov bez nutnosti definovať presnú triedu nového objektu. Namiesto použitia konštruktora new na vytvorenie objektu sa to dosiahne zavolaním factory metódy [7]. Informácie o factory designe boli získane z [11]. Na vytváranie nových typov widgetov väčšinou stačí štandardná sada ovládacích prvkov XForms. Implementácia však umožňuje pridávanie nových typov ovládacích prvkov v prípadoch, keď sú predpripravené ovládacie prvky nedostatočné. Nasledujúca časť popisuje ako vytvoriť takýto novy typ ovládacieho prvku. Zložený komponent sa vytvára nasledovne:

1. Zaregistrovať nový typ v XFormItemFactory;

2. Nastaviť zoznam podriadených komponentov ako vlastnosť items objektu prototypu nového typu

3. Použiť nový typ vo formulári.

Factory funkciu môžeme vidieť tu: XFormItemFactory.createItemType("_COLOR_", "color", ColorInput, _COMPOSITE_); Vidíme, že vytvárame nový typ v XFormItemFactory. Nevytvárame ho pomocou new vytvárame ho pomocou .createItemType.

6.4 Visitor

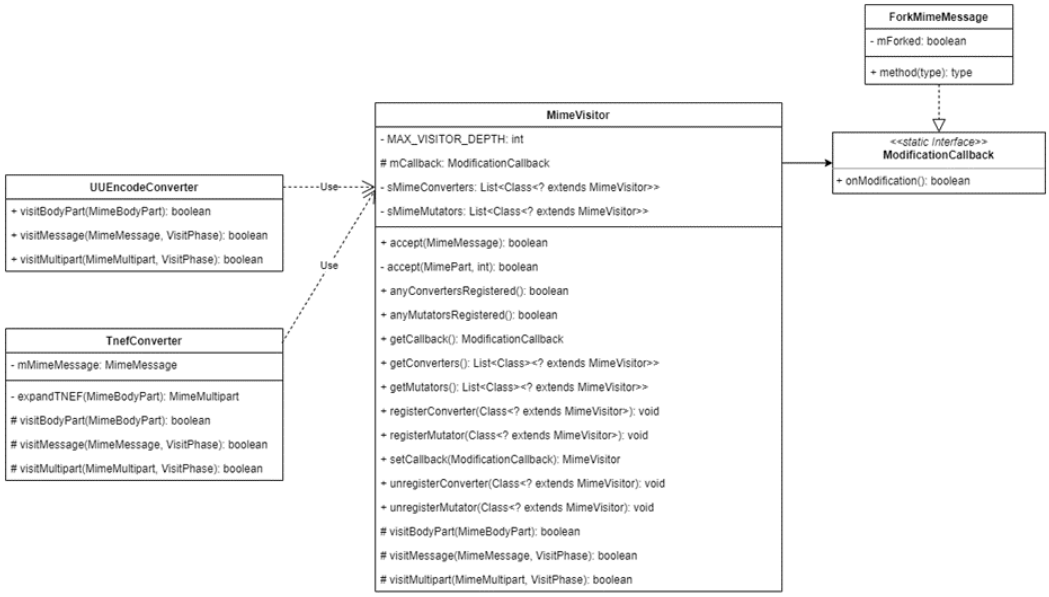

Figure 6: Visitor

Návrhový vzor visitor je technika používaná v objektovo orientovanom programovaní na oddelenie algoritmu od objektovej štruktúry, na ktorej je založený. Umožňuje pridávať nové funkcionality do existujúcich objektových štruktúr bez toho, aby sa tieto štruktúry menili [8]. Pri doručovaní súboru ako prílohy e-mailu sa okrem formátu MIME môže použiť viacero kódovaní súborov. Zimbra implementuje návrhový vzor visitor, ktorý je znázornený na obrázku 6, na spracovanie týchto kódovaní. Tieto triedy boli vytvorené na spravovanie kódovania TNEF a UUencoding pre prílohy.

Vidíme, že abstraktná trieda MimeVisitor slúži ako jadro štruktúry tohto návrhového vzoru visitor. V tejto triede sú tri abstraktné metódy. ktoré môžu implementovať jej podtriedy visitMessage, visitMultipart a visitBodyPart. Všetky tieto podtriedy konvertorov prijímajú ako argument podtriedu triedy MimePart a zvyčajne ich prepisujú. Ich abstraktné funkcie upravujú tieto argumenty, ktoré sú len objektmi obsahujúcimi hodnoty pre e-mailovú správu vo formáte MIME. Tieto funkcie vracajú logický výsledok, ktorý udáva, či správa bola alebo nebola úspešne dekódovaná.

MimeVisitor obsahuje aj statické metódy na registráciu niekoľkých objektov na spracovanie správy MIME, pretože e-mailová správa môže obsahovať viacero príloh v rôznych kódovaniach. ArrayList, ktorý používajú všetky tieto metódy, obsahuje len triedy, ktoré rozširujú triedu MimeVisitor. Na tento účel sa ovládajú celkovo dva ArrayListy, sMimeConverters a sMimeMutators. Keď sa správa načíta z úložiska alebo je pripravená na

indexovanie, automaticky sa vyvolá skupina konvertorov. Zmeny vykonané týmito objektmi sa vykonajú pri každom prístupe k správe namiesto toho, aby sa uložili na disk. Pred zápisom správy na disk alebo jej odoslaním cez SMTP sa správa trvalo upraví pomocou kolekcie objektov mutátorov. Okrem toho má trieda MimeVisitor vnútorné rozhranie ModificationCallback, ktoré umožňuje objektu používajúcemu funkciu accept dostávať oznámenia pred každou zmenou správy MIME vykonanou podtriedou MimeVisitor. Všetky objekty MimeVisitor majú synchronizované metódy accept, čo umožňuje zmrazenie zdrojov, s ktorými tieto objekty pracujú, a občas si vyžaduje použitie funkcií callback.

7 Súvisiace práca

Mojou úlohou bolo nájsť design patterny v zdrojovom kóde nato sú dva spôsoby. Prvým spôsobom je manuálna detekcia, to znamená prechádzanie kódu ručne. Druhý spôsob je automatický, to znamená detekcia pomocou umelej inteligencie [4]. Ja som si zvolil manuálny pristúp nakoľko Zimbra má výbornú dokumentáciu kde boli spomenuté použité design patterny. Design patterny ktoré neboli spomenuté v dokumentácii sa tiež hľadali veľmi ľahko pretože väčšinou bol ich názov priamo v názve súboru napríklad MimeVistior ako už z nazvú vyplýva obsahoval design pattern visitor.

Moja práca analyzovala architektúru softvéru Zimbra, niečim podobným sa zaoberal aj nasledujúci článok [3]. Táto práca sa však zaoberala architektúrou ako takou. Moja práca doplnila do rozboru architektúry aj to aké design patterny boli v tomto softvére použite a navrhol som aj prípadne implementačné zlepšenia.

8 Záver

V tejto práci som preskúmal viacero repozitárov ktoré ako celok vytvárajú software Zimbra. Popísal som načo nám tento software zimbra slúži. Potom som sa pozrel nato aké open-source komponenty používa a načo tieto komponenty slúžia. Bližšie som sa pozrel na architektúru serverovej ako aj klientskej časti. Popísal som aké technológie sa tam používajú a pomocou diagramov som bližšie ukázal ako architektúra funguje ako celok. Ďalej som v skratke spomenul čo tento softvér používa pre svoje zabezpečenie. Pre architektúru servera som aj navrhol implementačné vylepšenie a uviedol prečo si myslím, že toto vylepšenie by bolo správnym krokom vpred.

Na koniec som pomocou manual crawlingu našiel design paterny ktoré tento softvare používa. Design paterny som hľadal v celom balíku Zimbra. Keď som design pattern našiel popísal som načo tento design pattern vo všeobecnosti slúži. Následne som ukázal ako ho využil software Zimbra. Pre lepšie pochopenie som použil aj diagramy ktoré ukázali ktoré súbory sú

súčasťou tohto design paternu a akou tieto súbory a ich funkcie spolu súvisia a komunikujú.

References

[1] arimetrics. What is zimbra - definition, meaning and examples, May 2022.

[2] geeksforgeeks. Mvc design pattern, Feb 2018.

[3] V. Machado. Zimbra collaboration suite architectural overview, Jul 2015.

[4] N. Nazar, A. Aleti, and Y. Zheng. Feature-based software design pattern detection. *Journal of Systems and Software*, 185:111179, 2022.

[5] refactoring guru. Singleton.

[6] Z. wiki. Zimbra 9/modern web app.

[7] Wikipedia. Factory method pattern, Aug 2022.

[8] wikipedia. Visitor pattern, Jun 2022.

[9] Zimbra. Admin ui framework guide, Apr 2010.

[10] Zimbra. Deployment strategy, Jun 2011.

[11] Zimbra. Xforms reference, Oct 2020.

[12] Zimbra. Zimbra collaboration administrator guide, Mar 2020.