Poshet

Sabina-Alina Prodan

Universitatea "Alexandru Ioan Cuza", Iași, România

Abstract. Poshet este un client pentru serviciul POP3 în care se poate vizualiza, organiza și trimite e-mail. În continuare sunt detaliate atât tehnologiile utilizate, structura aplicației, cât și diverse detalii de implementare.

Keywords: C++ · Email · POP3 · SMTP · MIME · wxWidgets

1 Introducere

Poshet este un client cu interfață grafică, scris în C++ cu wxWidgets, pentru serviciile de mail electronic POP3 (pentru obținerea de mailuri) și SMTP (pentru trimiterea de mailuri și redirecționarea de mailuri către mailbox-urile corespunzătoare). Printre funcționalitățile acestuia se numără: vizualizarea și organizarea mailurilor primite, trimiterea de mailuri noi cu atașamente, reply sau forwarding la mailuri primite.

2 Tehnologii aplicate

Proiectul este scris în C++ utilizând OOP. Pentru compilarea sa sunt folosite atât fișiere `Makefile`, cât și CMake. Proiectul include un fișier `README.md` care detaliază procesul de compilare.

Comunicarea în rețea cu serverele POP3 și SMTP se realizează prin intermediul funcțiilor POSIX. Protocolul ales la nivel de transport este TCP, fiindcă ambele protocoale la nivel de aplicație sunt bazate pe acesta.

Comunicarea cu baza de date SQLite creată local se realizează cu ajutorul librăriei SQLite3. În comparație cu alternative precum XML, o bază de date concretă permite interogări mai complexe și o organizare mai ușoară a mailurilor dintr-un inbox.

Pentru interfața grafică este utilizată librăria wxWidgets (în pofida altor librării similare precum Qt) datorită simplității sale, a structurii bazate pe OOP și în mod special a integrării sale facile cu Visual Studio Code și CMake.

Pentru analizarea și procesarea mailurilor din conținut ASCII într-un format afișabil în client se utilizează o librărie externă (Mimetic) pentru fișiere scrise conform standardului MIME[3]. A fost aleasă o librărie externă pentru că protocolul MIME este destul de complex și se dorește o soluție implementată cât se poate de corect și testată riguros.

3 Structura aplicației

3.1 Arhitectura principală

Proiectul folosește clase și OOP, având o structură inspirată în linii mari de arhitectura MVP (Model-View-Presenter) pentru a separa responsabilitățile între clase.

3.2 Detalii despre clasele programului

Principala clasă care face legătura între interfețele aplicației și logica din spate (comunicarea cu servere, cu baza de date, etc.) este AppController. Aceasta administrează și comunică cu instanțe ale interfețelor LoginFrame, DashboardFrame și MailCreatorFrame și o instanță a clasei Session.

Session este clasa care se ocupă de comunicarea cu resurse externe pentru a obține și furniza informații. Aceasta solicită și trimite informații către instanțe ale altor clase specializate, care realizează operații precum obținerea mailurilor noi, salvarea lor în baza de date, trimiterea unui mail nou, sau salvarea locală a unui atașament.

În momentul primirii de notificări de la interfețele la care AppController este "abonat", AppController se ocupă de interacțiunea cu clasa Session pentru a obține informații și a le transmite către interfețe, care trebuie să folosească datele pentru a se actualiza și a le afișa către utilizator.

Clasa AppController poate fi privită ca un mediator între interfețe și logica programului, interacționând cât mai puțin cu librăria grafică și nefiind implicat în comunicarea efectivă în rețea.

3.3 Stocarea informațiilor

Datele care aparțin unui mail sunt păstrate cu ajutorul clasei Mail, care conține metode prin care se poate procesa conținutul unui buffer ASCII (scris conform protocolului MIME) și se pot obține informații precum headers, date plain-text, atașamente, etc. În momentul când se dorește afișarea conținutului unui mail pe ecran, interfața DashboardFrame este responsabilă cu afișarea datelor mailului în viewMailPanel, utilizând elemente specializate ale wxWidgets precum wxRichTextCtrl (care permite afișarea "inline" de imagini sau alte fișiere multimedia).

De asemenea, mailurile primite sunt stocate într-o bază de date locală, într-o tabelă `Mail` (legată de tabela `Users` printr-o relație de tip "one-to-many"). Pe lângă datele "raw" ale mailurilor sunt stocate și informații adiționale precum headers importante (To, Subject, etc.) sau un "tag" prin care utilizatorul a marcat respectivul mail în client pentru a își organiza inboxul.

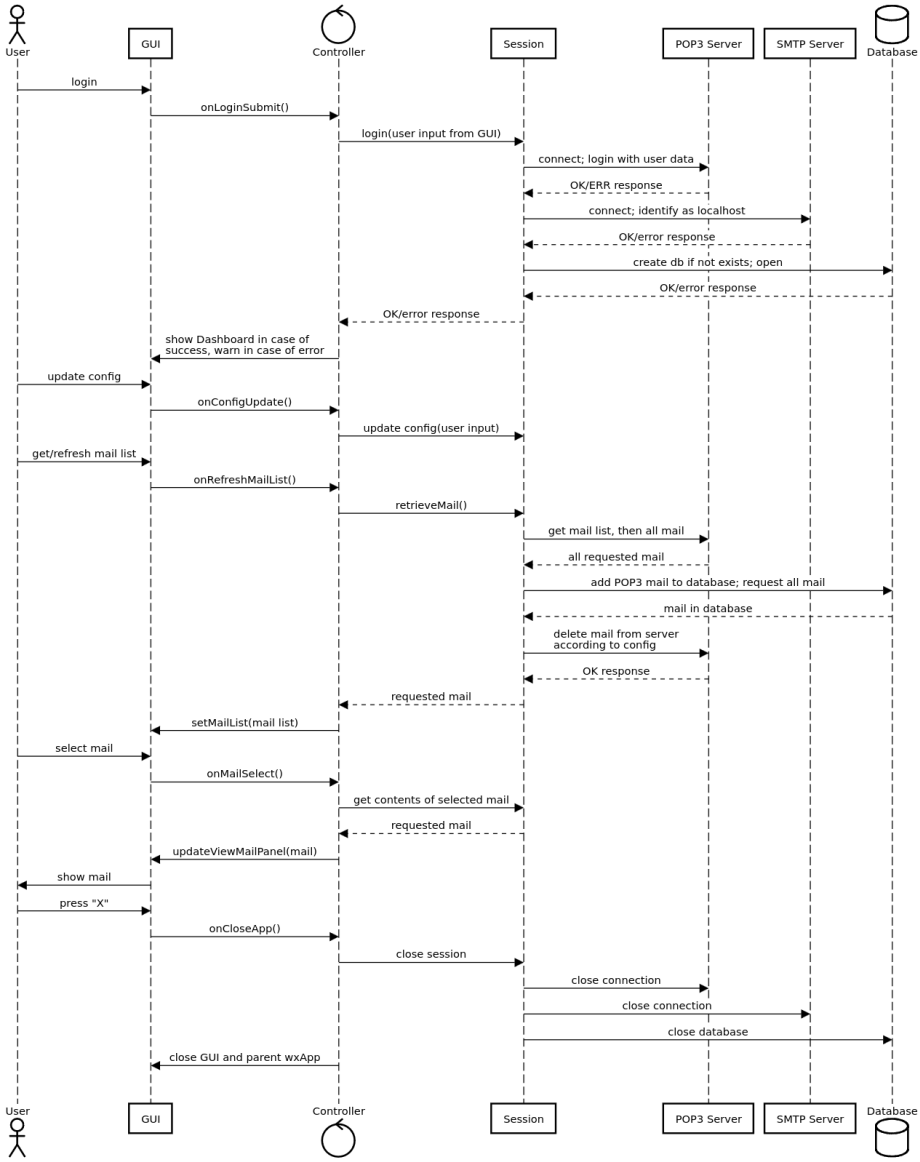

Fig. 1. Diagrama secvențială a aplicației, partea 1: aici sunt ilustrate mecanismele de logare, afișare a mailurilor primite, și închidere a aplicației.

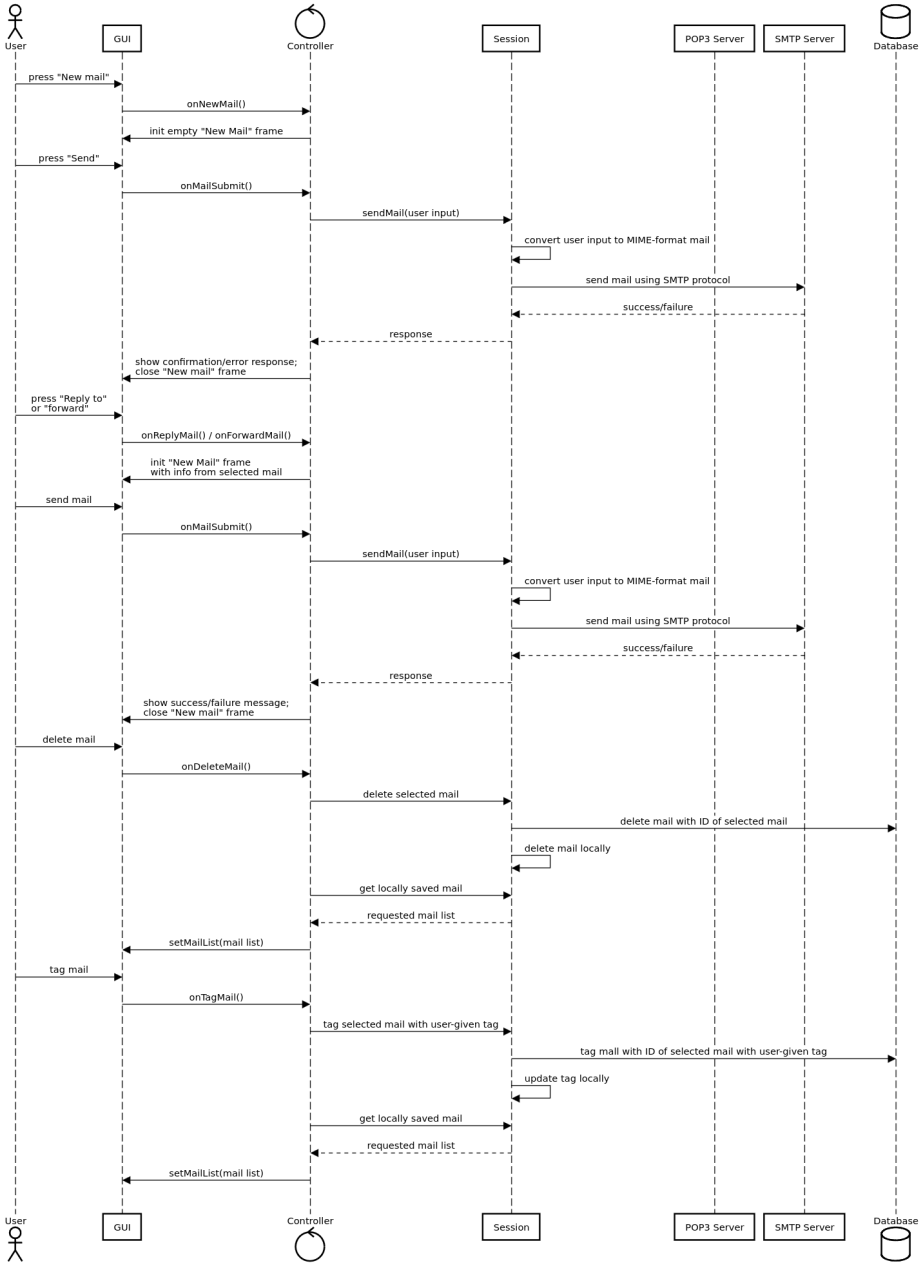

Fig. 2. Diagrama secvențială a aplicației, partea 2: aici sunt ilustrate mecanismele de trimitere de mail nou și organizare de mail primit.

4 Aspecte de implementare

4.1 Protocoale utilizate

Proiectul constă într-un client pentru servere existente, deci acesta se folosește de protocoale la nivel de aplicație deja existente.

Pentru obținerea mailurilor se folosește protocolul POP3[1], iar pentru trimiterea de mailuri se folosește protocolul SMTP[2]. Clasele responsabile de conexiunile POP3 și SMTP trimit comenzi corespunzătoare funcționalităților dorite de utilizator, recunoscute de servere; spre exemplu, `USER` / `PASS` pentru autentificare POP3, `RETR` pentru lista de mailuri, `LIST` pentru datele unui mail particular, `MAIL TO` și altele pentru trimiterea unui mail, etc.

Ambele protocoale sunt de tip comandă-răspuns; așadar, în urma trimiterii comenzii se așteaptă răspuns afirmativ (sau un mesaj de eroare) de la server, informația fiind apoi transmisă la funcțiile care au solicitat-o.

4.2 Scenarii de utilizare

Poshet este menit să comunice cu un server POP3 ne-încriptat, cu autentificare plain-text, și cu un server SMTP fără autentificare. În timpul dezvoltării acestui proiect s-au folosit programele Dovecot și Postfix pentru a configura servere POP3 și respectiv SMTP pe `localhost`.

Un client de acest tip poate fi folositor în situații precum testarea locală; unele procese pot fi configurate să trimită mail către serverul SMTP local, iar administratorul le poate citi ușor din aplicație.

4.3 Aspecte interesante legate de implementare

Utilizarea sistemului de "subscribers". Pentru a evita dependența circulară între clasa AppController și clasele de interfață, dar și pentru a minimiza interacțiunea cu AppController prin modalități specifice librăriei wxWidgets, se utilizează conceptul de "subscribers".

AppController implementează simultan trei clase pur virtuale: LoginFrameSubscriber, DashboardFrameSubscriber și MailCreatorFrameSubscriber. În momentul creerii LoginFrame, DashboardFrame și respectiv MailCreatorFrame, AppController apelează metodele publice `subscribe` ale acestora, iar fiecare instanță de interfață memorează o referință la AppController sub forma unui pointer de tipul "subscriber" corespunzător.

În wxWidgets, interacțiunea utilizatorului cu o interfață este semnalată acesteia prin intermediul unor `wxEvent`-uri. Clasele de interfață ale Poshet conțin "event handlers" pentru acestea, implementate conform cerințelor librăriei, care la rândul lor apelează funcții ale "subscriber"-ului implementate în AppController, precum `onRefreshMailList()`, `onCloseApp()` sau `onMailSend()`.

```
class FrameSubscriber {
public:
    virtual void onCloseApp() = 0;
};

class LoginFrameSubscriber : virtual public FrameSubscriber {
public:
    virtual void onLoginSubmit() = 0;
};

class DashboardFrameSubscriber : virtual public FrameSubscriber {
public:
    virtual void onSelectMail() = 0;
    virtual void onNewMail() = 0;
    virtual void onReplyMail() = 0;
    virtual void onForwardMail() = 0;
    virtual void onRefreshMailList() = 0;
};

class MailCreatorFrameSubscriber : virtual public FrameSubscriber {
public:
    virtual void onMailCreatorSend() = 0;
    virtual void onMailCreatorClose() = 0;
};
```

Fig. 3. Declararea claselor pur-virtuale de tip "subscriber" în `FrameSubscribers.hpp`.

```
class DashboardFrame : public wxFrame {
protected:
    DashboardFrameSubscriber* _subscriber;
```

Fig. 4. Definirea membrului `_subscriber` în `DashboardFrame.hpp`.

```cpp
void DashboardFrame::OnClose(wxEvent& e) {
    _subscriber->onCloseApp();
}

void DashboardFrame::OnRefreshMailList(wxCommandEvent& e) {
    _subscriber->onRefreshMailList();
}

void DashboardFrame::OnNewMail(wxCommandEvent& e) {
    _subscriber->onNewMail();
}
```

Fig. 5. Exemple de ”event handlers” care apelează funcțiile unui ”subscriber”, în `DashboardFrame.cpp`.

Administrarea conexiunilor POP3 și SMTP. Conexiunile cu serverele POP3 și SMTP sunt administrate de clasele POP3Connection și SMTPConnection, care asigură comunicarea corectă cu acestea conform protocoalelor corespunzătoare.

În general, după o anumită perioadă de inactivitate configurată de către administratorii serverelor, acestea pot încheia conexiunea. O particularitate interesantă a claselor POP3Connection și SMTPConnection este folosirea unui `std::thread` pentru a menține conexiunile active chiar și după mai mult timp: acesta trimite la un anumit interval repetat de timp, conform protocolului corespunzător, o operație de tip ”no-op”, prin care să semnaleze activitate fără a realiza o operație propriu-zisă.

În POP3Connection și SMTPConnection există membrul `_noopThread` și o metodă `keepAlive()` care este executată de către acesta. Pentru ca ”no-op”-urile trimise de acest thread să nu interfereze cu procesul comandă-răspuns se folosește un `_commandMutex`, folosit în funcția `execCommand` care este apelată atât de comenzi reale, cât și de ”no-op”. De asemenea, mutexul `_stateMutex` este utilizat, printre altele, pentru a actualiza și verifica în mod sigur statusul curent al conexiunii.

```cpp
State _state = State::DISCONNECTED;

std::thread _noopThread;
bool _threadStarted = false;

std::mutex _commandMutex;
std::mutex _stateMutex;
std::condition_variable cv;
```

Fig. 6. Declararea `_noopThread` și a altor membri ajutători în `POP3Connection.hpp`.

```
void POP3Connection::keepAlive() {
    while(true) {
        {
            std::unique_lock<std::mutex> lock(_stateMutex);
            auto result = cv.wait_for(lock, std::chrono::seconds(TIMEOUT_SECS), [this]() {return _state != State::TRANSACTION;});
            if (result == true) {
                break;
            }
        }
        execCommand("NOOP");
        log("Sent NOOP");
    }
}
```

Fig. 7. Definiția `keepAlive()` în `POP3Connection.cpp`.

5 Concluzii

Proiectul funcționează într-un scenariu limitat. Acesta comunică ne-încriptat cu un server POP3 pe `localhost` cu autentificare bazată pe username și parolă în plain-text, și cu un server SMTP ne-încriptat, fără autentificare, care trimite mailuri doar în rețeaua locală. Din păcate, majoritatea serverelor POP3 și SMTP externe existente astăzi cer un anumit nivel de încriptare a datelor trimise în rețea și de o configurare mai complexă; așadar, practicalitatea proiectului este limitată. Modificarea proiectului pentru a permite comunicarea cu aceste servere nu ar fi imposibilă, dar ar necesita un grad mai mare de complexitate și ar implica dependențe față de mai multe librării externe care să se ocupe de comunicarea securizată în rețea.

De asemenea, librăria wxWidgets are propriile limitări. Fiindcă se bazează pe elemente existente în interfețele sistemelor de operare cu care este compatibilă[4], aceasta nu este la fel de personalizabilă ca unele alternative precum Qt. O posibilă îmbunătățire (dacă se dorește personalizarea mai detaliată a proiectului) ar putea fi portarea la o altă librărie pentru interfața grafică.

Referințe bibliografice

1. RFC 1939, https://www.ietf.org/rfc/rfc1939.txt. Accesat cel mai recent pe 14 Dec 2023
2. RFC 5321, https://www.ietf.org/rfc/rfc5321.txt. Accesat cel mai recent pe 14 Dec 2023
3. RFC 2045, https://datatracker.ietf.org/doc/html/rfc2045. Accesat cel mai recent pe 14 Dec 2023
4. Pagina principală wxWidgets, https://www.wxwidgets.org/. Accesat cel mai recent pe 14 Dec 2023