Email Integration Application

Analysis and Design Document

Student: Bolba Raluca Maria

**Group: 30235**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 5

2.1 Conceptual Architecture 5

2.2 Package Design 5

2.3 Component and Deployment Diagrams 6

III. Elaboration – Iteration 1.2 7

1. Design Model 7

1.1 Dynamic Behavior 7

1.2 Class Design 8

2. Data Model 9

3. Unit Testing 10

IV. Elaboration – Iteration 2 10

V. Construction and Transition 10

1. System Testing 10

2. Future improvements 10

VI. Bibliography 10

# Project Specification

Proiectul isi propune implementarea unei aplicatii care integreaza mai multe tipuri de adrese de email (de la mai multi furnizori) pentru a permite unui utilizator sa le administreze intr-o maniera mai centralizata. Aplicatia se comporta ca serviciile de mesagerie electronica apartinatoare unor furnizori precum Google, Yahoo, Hotmail, avand aceleasi functii de citire, trimitere, stergere email, doar ca in loc de o singura adresa de email, acestea se pot efectua pe mai multe adrese de email, de furnizori diferiti sau nu.

Pentru a putea utiliza aplicatia, utilizatorul trebuie sa se autentifice la sistem, iar in cazul in care nu are un astfel de cont, poate crea gratuit unul. Dupa autentificare, acesta isi poate adauga sau elimina conturi de email pe care sa le administreze si poate de asemenea sa isi modifice datele personale, pe langa operatiile amintite anterior pe care le poate efectua asupra email-urilor.

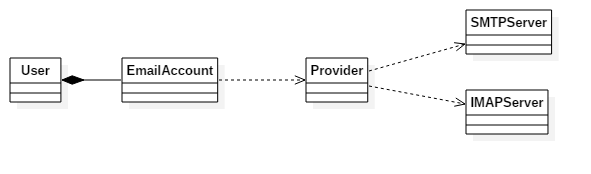
# Elaboration – Iteration 1.1

# Domain Model

Principalele concepte întâlnite în cadrul acestei aplicații și care vor fi parte a domeniului sunt : utilizatorul, sau clientul care va folosi aplicația, și care va trebui să aibe un nume de utilizator și o parolă pentru a fi recunoscut de sistem și pentru a putea folosi funcțiile acesteia, conturi de email, care vor fi deținute de utilizatori (unul sau mai multe) și care au un tip, furnizori, sau *provideri*, de mail, iar în cazul acestei aplicații îi vom consider pe cei mai comuni, și anume : Yahoo, Gmail și Hotmail. Pentru a efectua anumite operații precum trimitere de email, citire email, trebuie să existe o comunicare cu serverele de email prin așa numitele servere SMTP (Simple Mail Transfer Protocol) și IMAP (Internet Message Access Protocol). Acestea au configurații diferite pentru fiecare tip de provider în parte, setări cum ar fi numele, numărul portului, dacă se utilizează SSL (Secure Sockets Layer) etc.

Este necesat de asemenea un centru de control sau de administrare a aplicației, care să țină evidența utilizatorului aplicației și a adreselor de email a acestuia care le va folosi în cadrul aplicației. Tot în acest modul se vor procesa comenzile de adăugare de cont de utilizator, sau de adăugare de adresă de email, caz în care trebuie să i se atribuie adresei și un provider, cu serverele SMTP și IMAP corespunzătoare astfel încât să se poată efectua operațiile de citire, scriere, ștergere email.

**Conceptual class diagram :**



# Architectural Design

## Conceptual Architecture

Pattern-ul architectural pe care am ales să îl folosesc pentru această aplicație este pattern-ul MVC (Model View Controller) care, așa cum sugerează numerele, structurează aplicația în 3 categorii : Model, View și Controller. Am ales acest pattern pentru ca fiecare componenta a aplicației (de model, vizualizare sau controller) să fie cât mai delimitată de restul componentelor.

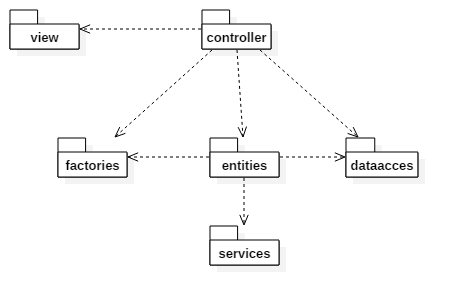
Partea de *Model* este responsabilă de păstrarea stării curente a aplicației sau a unui set de date. Pe lângă date, care sunt reprezentări ale conceptelor utilizate în aplicație, mai conține și logica care se aplica pe aceste date. Modelul nu are nici o informație cu privire la felul în care sunt reprezentate datele către utilizatori într-un mod grafic. Acesta oferă în schimb Controller-ului metode prin care starea curentă a aplicației poate fi accesată sau modificată. Pe partea de *Model* se intalnesc entitatile aplicatiei (utilizatorul, conturile de email, furnizorii) dar si clasele ce se ocupa de accesare a datelor din baza de date. In cadrul acestei aplicatii, din aceasta componenta fac parte pachetele de *factories, entities, services si dataaccess.*

Partea de *View* se ocupă de reprezentarea într-o manieră grafică a datelor primite de la model. Pot exista de exemplu mai multe moduri de reprezentare pentru același set de date dintre care Controller-ul poate alege unul. In cazul acestei aplicatii, partea de *View* este reprezentata de pachetul *view*, ce contine clasa *Gui*, de modelare a interfetei grafice.

Partea de *Controller* este partea ce se ocupă de procesarea cererilor venite de la utilizator, în acest caz ascultătorii de evenimente pentru componentele din interfața grafică, iar datele primite de la utilizator sunt eventual modificate și transmise către partea de model pentru a modifica starea curentă a sistemului. Această parte mai are rolul de a păstra modelul și view-ul sincronizate. In cadrul acestei componente se situeaza si pachetul *controller* cu clasele corespunzatoare.

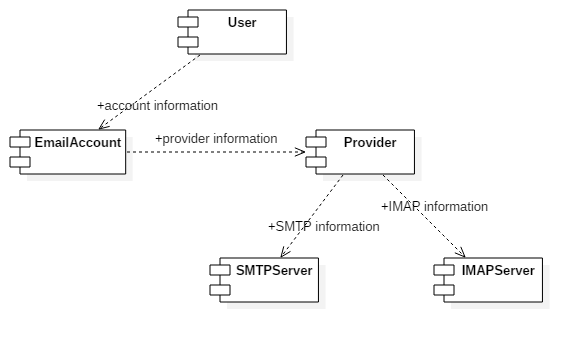
## Package Design

Pachetele corespund pattern-ului architectural MVC conform următoarelor: partea de View corespunde pachetului *view*,partea de Controller corespunde pachetului *controller* iar partea de Model este formată din pachetele *entities, dataaccess, services și factories.*

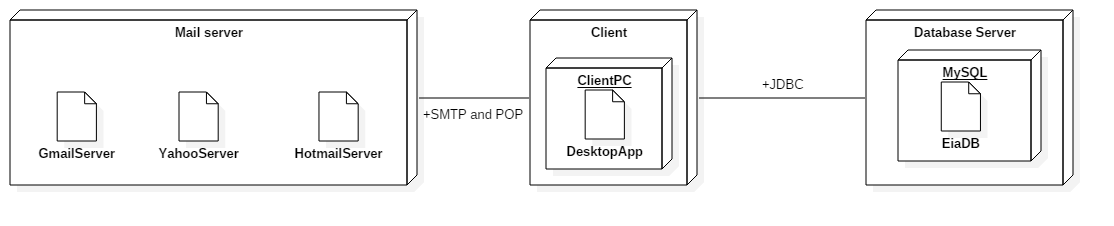
**

## Component and Deployment Diagrams

*Component diagram :*

**

*Deployment diagram :*

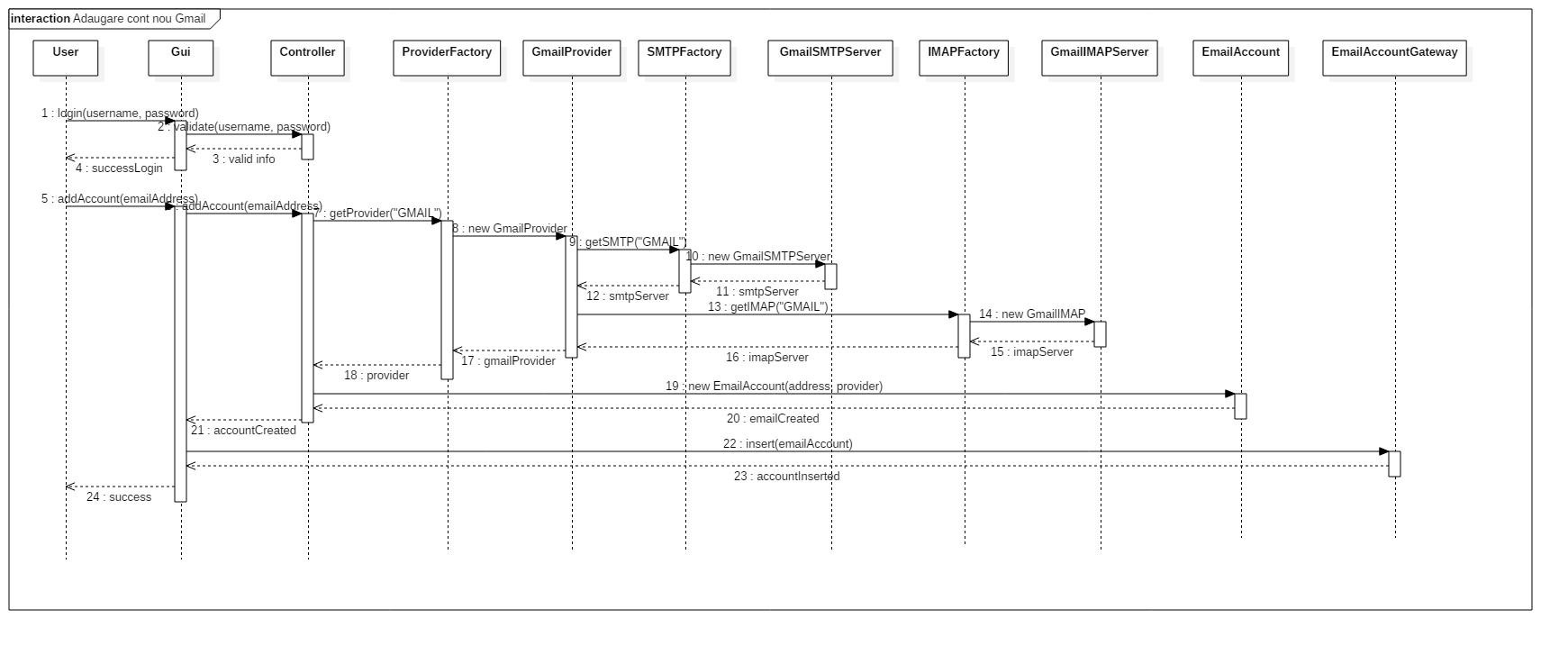


# Elaboration – Iteration 1.2

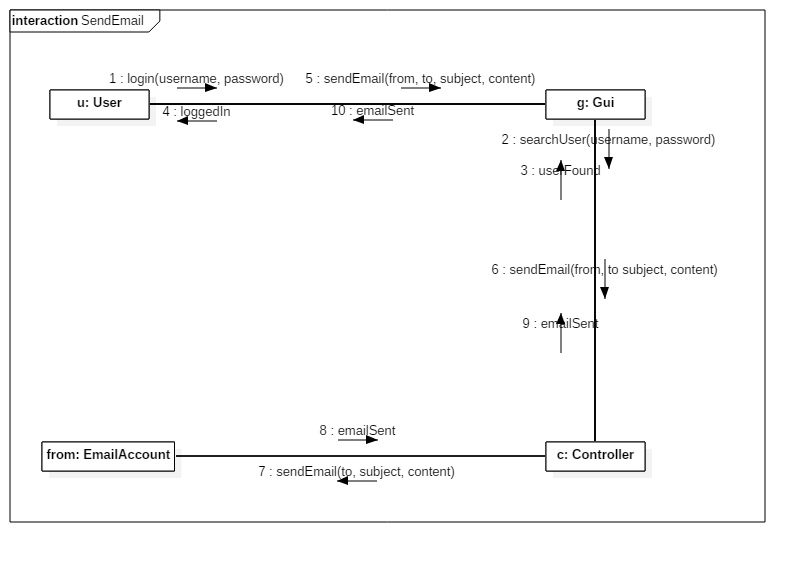
# Design Model

## Dynamic Behavior

Diagramă de secvenţă pentru adăugarea de către un utilizator a unei noi adrese de email de tip Gmail:

**

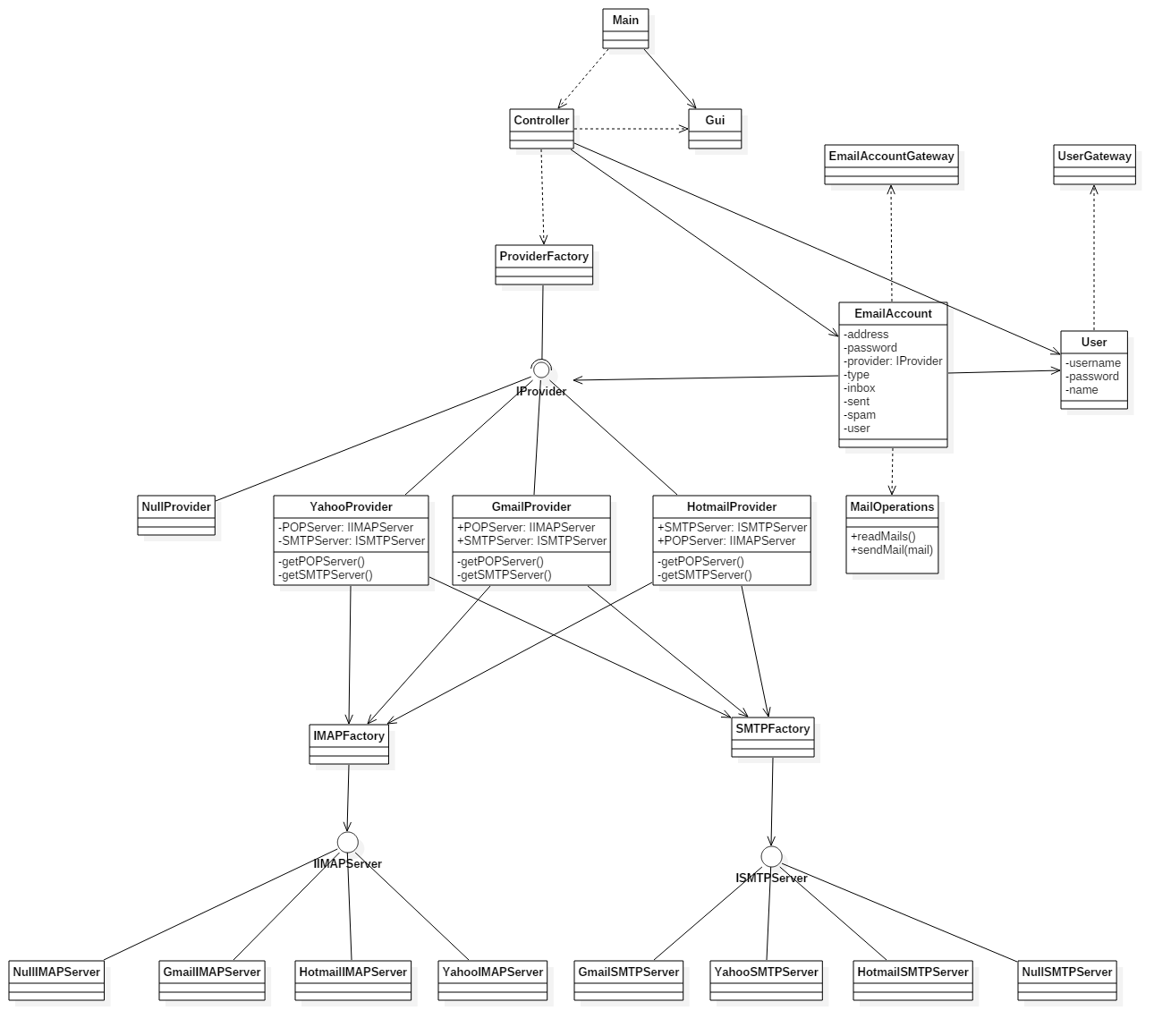
Diagramă de comunicare pentru operația de trimitere email :

**

## Class Design

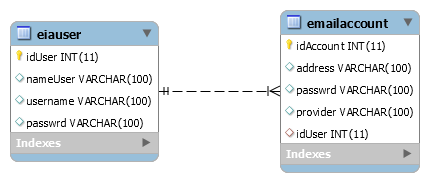
Pentru această aplicație am decis să utilizez design pattern-ul *Factory* pentru crearea unor obiecte de tip *Provider* întrucât acesta poate fi de trei tipuri : Gmail, Yahoo și Hotmail (însă se pot adăuga și alte tipuri de furnizori pentru aplicație)*, SMTPServer* și *IMAPServer*, întrucât acestea au attribute particulare pentru cei trei tipuri de furnizori precizați*.* Astfel fiecare obiect de tip *Provider* va avea un obiect de tip server SMTP și unul de tip IMAP în funcție de tipul furnizorului, iar crearea acestor obiecte va reveni în funcția claselor *SMTPFactory și IMAPFactory*, *Provider* netrebuind să fie conștient de acest proces, el doar folosindu-le în anumite situații. Obiectele de tip *ISMTPServer* si *IIMAPServer* sunt utile pentru citirea, respective trimiterea de email-uri, serverul SMTP ocupandu-se de trimiterea acestora, pe cand IMAP se ocupa de accesarea si citirea acestora.

Pentru accesarea, inserarea, modificare sau stergerea anumitor date din baza de date am decis utilizarea pattern-ului *Table Data Gateway*. *Table Data Gateway* este un design pattern care conține toate interogările SQL pentru accesarea unui singur tabel din baza de date. O singură instanță controlează toate rândurile unui tabel. Metodele uzuale întânite în cadrul unei astfel de clase sunt de creare, citire, modificare sau ștergere din baza de date, dar și metode de căutare după anumite câmpuri din tabel. Clasele care ilustreaza utilizarea acestui pattern sunt *EmailAccountGateway*  care opereaza asupra datelor referitoare la conturile de email detinute de clienti si *UserGateway* care operareaza asupra datelor referitoare la utilizatorii aplicatiei.

**

# Data Model

În cadrul aplicației voi folosi o bază de date pentru a stoca informații despre conturile utilizatorilor aplicației și despre conturile de email ale acestora. Așadar baza de date constă din două tabele, *EiaUser* care conține informații despre utilizatorii aplicației, informații precum numele, numele de utilizator și parola și tabela *EmailAccount* care stochează date despre adrese de email folosite de utilizator în cadrul aplicației. Informațiile stocate în *EmailAccount* sunt adresa de email, parola si tipul de cont de email (Yahoo, Gmail, Hotmail) și o cheie străina la tabela *EiaUser* care reține id-ul utilizatorului care deține adresa respectivă.

**

# Unit Testing

Pentru testarea aplicației am utilizat testarea manuala. Aplicatia s-a rulat și s-a verificat dacă operațiile sunt realizate în mod corect. De exemplu pentru operația de autentificare s-a verificat dacă datele introduse sunt cele prezente și în baza de date, iar în caz contrar s-a afișat un mesaj de eroare.

# Elaboration – Iteration 2

# Construction and Transition

# System Testing

Pentru testarea sistemului, am folosit testarea manuala si am urmarit corectitudinea si eficienta operatiile pentru un utilizator al aplicatiei. Am testat daca autentificarea se face in mod corespunzator, iar daca e vorba de un client neautorizat daca se afiseaza un mesaj de eroare. Am testat de asemenea operatiile de creare sau actualizare cont pentru client, dar si operatiile de citire email, raspundere email, trimitere sau stergere email, dar si daca filtrarea email-uriilor s-a facut in mod corespunzator (daca email-urile sunt impartite corespunzator in “folderele” *inbox, sent* sau *spam*). Am testat de asemenea daca datele sunt consistente dupa efectuare operatiilor asupra bazei de date.

# Future improvements

Aplicatia poate fi imbunatatita in multe privinte. De exemplu, aceasta se poate realiza sub forma de aplicatie web, si nu de tip desktop, pentru a fi cat mai asemanatoare cu serviciile de mesagerie electronica existente. De asemea, se mai pot aduce imbunatatiri la performanta acesteia, pentru a reduce timpul de raspuns la citirea email-urilor. Se pot include mai multe functionalitati (vizualizarea sau descarcarea atasamentelor) sau se poate imbunatati interfata grafica.

# Bibliography

<http://www.tutorialspoint.com/javamail_api/>

<http://www.howtogeek.com/56002/htg-explains-how-does-email-work/>

<http://www.tutorialspoint.com/design_pattern/factory_pattern.htm>

<http://martinfowler.com/eaaCatalog/tableDataGateway.html>