Sistem de file-sharing

- Document de Proiectare Arhitecturală -

Versiune intermediară - 17.04.2013

TheGunn3rs:

Țucă Laurențiu - 333 CA

Șerban-Rădoi Constantin - 333 CA

Culcuș Mihaela-Nicoleta - 333 CC

Petrescu Rareș - 333 CC

Îndrumător:

Oana Maria Ferche

Cuprins

[1) Tehnologii folosite 3](#_Toc354346227)

[2) Descrierea modulelor aplicației 3](#_Toc354346228)

[a) Descriere pe scurt a modulelor 3](#_Toc354346229)

[b) Detaliere clase aplicație 4](#_Toc354346230)

[c) Diagrame UML 5](#_Toc354346231)

[3) Explicații pe baza documentului Specificația sistemului 8](#_Toc354346232)

[4) Testarea Sistemului 9](#_Toc354346233)

[1. Testare Unitară 9](#_Toc354346234)

[2. Teste de integrare 9](#_Toc354346235)

[3. Teste de sistem 9](#_Toc354346236)

# Tehnologii folosite

###### Limbajul de programare

Limbajul de programare pe care îl vom folosi pentru implementarea sistemului de fişiere este Java. Am ales acest limbaj deoarece am considerat că ne va fi mai uşor în scrierea aplicaţiei, mai ales pentru partea de interfaţă grafică.

Aplicaţia scrisă în Java va fi portabilă pe orice sistem de operare (Windows, Unix, Mac OS). Java are un nivel de portabilitate, ce nu există de exemplu pe limbaje mai vechi cum ar fi C, fiind unul din motivele ce ne-a condus la această alegere.

Pagina oficială Java se află la adresa următoare:

<http://docs.oracle.com/javase/specs/jls/se7/html/index.html>

###### Mediu de dezvoltare

Eclipse este mediul de dezvoltare open-source ce îl vom folosi pentru dezvoltarea aplicaţiei. Folosim acest IDE deoarece toţi membrii echipei sunt familiarizaţi cu el şi din punctul nostru de vedere este eficient. Un alt motiv ar fi faptul că fiind scris în Java, mediul de programare Eclipse este destinat dezvoltării în special a aplicaţiilor Java.

Pagina oficială Eclipse se află la adresa următoare:

<http://www.eclipse.org/downloads/moreinfo/java.php>

###### Biblioteca pentru interfața grafică

Pentru interfaţa grafică vom folosi pachetul java.awt, deoarece conţine foarte multe componente utile pentru crearea de interfeţe grafice. Toate componentele necesare pentru construirea interfeţei grafice le găsim în acest pachet. Din nou, toţi membrii echipei sunt familiarizaţi cu acesta, fiind unul din motivele care ne-a condus la decizia de a-l folosi.

Pagina oficială pentru biblioteca AWT se află la adresa următoare:

<http://docs.oracle.com/javase/6/docs/api/java/awt/package-summary.html>

###### Framework pentru testare automată

Pentru testarea automată a sistemului, vom folosi framework-ul JUnit, deoarece oferă o flexibilitate foarte mare pentru crearea de teste unitare. Cu ajutorul acestui framework se pot crea diferite teste, care pot fi grupate în suite, ce pot fi rulate independent. Cu ajutorul acestui framework se pot crea de asemenea și teste de integrare a modulelor, prin care să se poată descoperi eventuale bug-uri în legăturile dintre acestea.

Pagina oficială pentru framework-ul Junit se află la adresa următoare:

<http://junit.org/>

Proiectul nostru nu necesită un sistem de gestiune a bazelor de date.

Cele 3 tehnologii descrise mai sus reprezintă baza proiectului si doar pe acestea le vom folosi.

# Descrierea modulelor aplicației

### Descriere pe scurt a modulelor

În cadrul proiectului avem patru clase principale (ce asigură partea de backend, comunicația în rețea): Server, Client, MiniServer și MiniClient și o clasă ce va îngloba chestiunile legate de Graphical User Interface(GUI) care va fi numită întocmai.

Cele două aplicații ce se vor obține sunt :

* aplicația **ServerApplication** ce va conține implementarea clasei server detașată în cadrul proiectului;
* aplicația **ClientApplication** ce va conține implementarea claselor Client, Miniserver și Miniclient.

Partea de GUI va fi de asemenea integrată în această aplicația client, pentru a oferi o interfață prietenoasă cu utilizatorul.

Observație: nu există parte de GUI în aplicația Server, motivul fiind că de servere se ocupă în general specialiști care lucrează mult mai ușor în CLI, și în plus serverul este în general automatizat, necesitând doar o foarte mică implicare a factorului uman.

Cele patru clase principale vor implementa trei interfețe:

* interfața ServerAplicationInterface, ce va fi implementată de către clasa Server.
* interfața ClientServerComunnication, ce va fi implementată de toate cele patru clase principale.
* interfața ClientAplicationInterface, ce va fi implementată de către clasa Client.

### Detaliere clase aplicație

#### ServerApplication

##### Clasa Server

* rol: după cum îi spune și numele, va fi serverul într-un model client-server standard. Așadar primește cereri de tipul ConnectToServer, DisconnectFromServer, GetListofClients, GetClientInfo și răspunde acestora.
* va implementa metodele din interfețele ServerAplicationInterface pentru a da răspunsuri la requesturile primite și ClientServerComunnication pentru a primi requesturi de la client și a le trimite înapoi clientului care a făcut aceste requesturi.

##### Clasa Main

Clasa main a acestei aplicații se va instanția un Server ce va primi anumiți parametri de tipul minshared, maxuptime etc., apoi se va apela o metodă de tip run ce va funcționa în regim while(1).

#### ClientApplication

##### Clasa Client

* rol: reprezintă clientul în modelul client-server standard. Primește răspunsuri la întrebarile descrise mai sus pe care le “pune” serverului la care se conectează.
* implementează așadar metodele din interfețele ClientApplicationInterface pentru a face cereri și “a descifra” răspunsurile la aceste cereri venite de la server și ClientServerComunnication pentru a trimite pe rețea requesturi și a primi tot prin rețea răspunsuri la aceste requesturi.

##### Clasa Miniclient

* rol:va face cereri de tipul GetClientFileList și GetFile deci cereri de a primi un fișier și de a primi lista de fișiere share-uite de un alt user.
* implementează metode din interfețele ClientApplicationInterface (cele descrise mai sus) pentru a codifica cererile pentru ca acestea să fie puse prin intermediul metodelor din interfața ClientServerComunnication pe rețea.

##### Clasa Miniserver

* rol: răspunde la cereri de tipul GetClientFileList și GetFile, și trimite în cazul în care cererile sunt corecte, răspunsuri adecvate (lista de fișiere sau fișierul cerut, în funcție de cererea primită).
* implementează metodele SendFile și SendFileList din interfața ClientApplicationInterface și send și receive, pentru a primi/trimite cereri pe rețea.

##### Clasa GUI

* rol: oferă o interfață prietenoasă utilizatorilor aplicației
* depinde de implementările celor 3 clase anterioare din aplicația Client.

##### Clasa Main

* rol: pornește o instanță de GUI care la rândul ei porneste un Client.
* depinde de tot ce a fost implementat mai înainte.

### Diagrame UML

#### Diagrama bloc

Diagrama bloc a aplicației poate fi urmărită în Fig. 1. După cum se poate observa și în diagramă, aplicatia este formată din aplicații separate, ServerApplication și ClientApplication. Aceste două aplicații sunt unificate prin intermediul interfeței de comunicare între cele două entități, ClientServerCommunication.

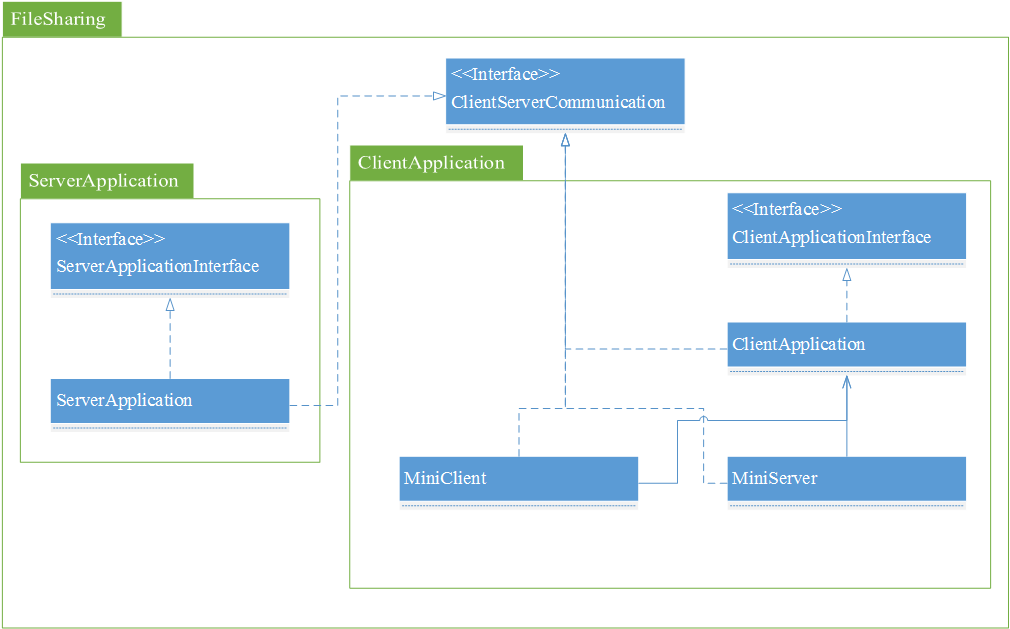


Fig. 1 - Diagrama bloc a aplicației

#### Interfața aplicației Server

Interfața aplicației Server este expusă în figura Fig. 2, și descrie principalele functionalități pe care le are entitatea Server, în cadrul aplicației. Această interfață va fi implementată de o clasă cu nume similar, anume ServerApplication. Tot această clasă va implementa și interfața de comunicare, ClientServerCommunication, care va dispune primitivele de comunicare între cele două aplicații.

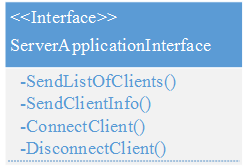


Fig. 2 - Interfața ServerApplicationInterface

#### Interfața aplicației Client

Interfața aplicației Client este expusă în figura Fig. 3, și descrie principalele funcționalități pe care le are entitatea Client, în cadrul aplicatiei. Această interfață va fi implementată de o clasă cu un nume similar, anume ClientApplication. Această clasă va implementa și interfața de comunicare, ClientServerCommunication, ce va dispune primitivele de comunicare între aplicații. După cum se poate observa și în imagine, interfața Client este mai cuprinzătoare, având o serie de funcționalități pe care clientul trebuie să le suporte.

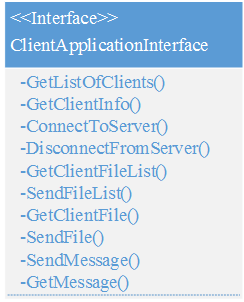


Fig. 3 - Interfața ClientApplicationInterface

#### Interfața de comunicare

Evident, pentru a suporta conectarea între clienți, clasa ClientApplication va depinde și de alte două clase, anume MiniClient și MiniServer, cu ajutorul cărora se va face efectiv comunicarea dintre clienți. Cele două clase vor implementa și ele interfața de comunicare, ClientServerCommunication, și astfel se va asigura posibilitatea de a trimite mesaje sau fișiere între clienți.

Interfața de comunicare ClientServer este prezentată în figura Fig. 4. Interfața este una simplă, fiind necesare doar două primitive: Una pentru trimitere de informații, și alta pentru primirea lor de cealaltă parte. În cadrul interfeței se pot defini și anumite constante pentru tipurile mesajelor ce pot fi folosite în implementare.

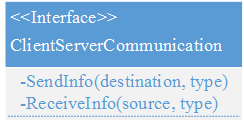


Fig. 4 - Interfața ClientServerCommunication

# Explicații pe baza documentului Specificația sistemului

După cum am prezentat în documentul “Specificația sistemului”, vor exista următoarele cazuri de utilizare:

**Conectarea la server** - Se va face de către interfața Client prin apelul funcției ConnectToServer(). În acest moment, interfața Server (care anterior aștepta cereri de conexiune) primește cererea curentă prin funcția ConnectClient().

**Adăugarea de fișiere partajate** - Se va face din meniul aplicației Client a unui utilizator, punându-se numele și calea lor într-o lista, urmând ca fișierele să fie vizibile la comanda GetClientFileList() din partea celuilalt utilizator.

**Vizualizarea utilizatorilor conectați** - Se va face de către interfața Client prin apelul funcției GetListOfClients(). În acest moment, interfața Server trimite lista utilizatorilor conectați la el prin SendListOfClients().

Dacă se dorește aflarea de informații asociate unui utilizator (nume, ip, oraș etc.), un client poate cere Serverului aceasta prin GetClientInfo(), acesta din urmă trimițându-le cu SendClientInfo().

**Conectarea la un anumit utilizator și listarea fișierelor** - Se realizează asemănător conectării la Server, de fapt conectarea se face chiar la MiniServerul din aplicația Client prin intermediul funcției ConnectToServer(). Fișierele se cer de către un client cu funcția GetClientFileList(), urmând ca celălalt să i le trimită cu SendFileList().

**Descărcarea fișierelor share-uite de un utilizator** - După ce utilizatorul 1 (Miniclient) și-a ales unul/mai multe fișiere de la utilizatorul 2 (Miniserver), descărcarea se face apelând funcția GetClientFile() pe Miniclient și funcția SendFile() pe MiniServer.

**Trimiterea de mesaje unui utilizator** - Se va realiza între interfața Client și interfața Server prin intermediul interfeței ClientServerCommunication. Cei doi utilizatori sunt, pe rând, inițiatori de mesaje(MiniClient) și receptori de mesaje(MiniServer). Deci inițiatorul va trimite mesajul prin intermediul funcției SendMessage(), acesta va ajunge la interfața ClientServerCommunication, care îl va primi prin ReceiveInfo(source,type). Apoi prin SendInfo(destination,type) mesajul este direcționat către MiniServerul din aplicația Client. Aceasta din urmă îl primește prin GetMessage().

**Deconectarea de la server** - Se va face de către interfața Client prin apelul funcției DisconnectFromServer(). În acest moment, interfața Server primește cererea curentă prin funcția DisconnectClient().

# Testarea Sistemului

## Testare Unitară

Testarea unitară presupune crearea unor teste la nivelul claselor. Pentru aceasta, se dorește existența unor suite de teste care să acopere cât mai bine cazurile de utilizare cele mai frecvente. Astfel, vom crea câte o suită de teste la nivel de API pentru clasele aplicației noastre.

Mai mult, urmărim dezvoltarea sistemului într-o manieră „Test Driven Development”, ce presupune mai întâi scrierea unui test, apoi scrierea metodei pentru care am scris testul. Astfel putem descoperi bug-urile încă de la începutul ciclului de dezvoltare.

## Teste de integrare

Aceste teste presupun verificarea corectitudinii legăturilor dintre modulele sistemului și interacțiunea dintre acestea. Pentru astfel de verificări, vom crea un set de teste care vor verifica simultan cel puțin două clase/module care vor comunica. Un exemplu de astfel de test ar putea fi faptul că dacă trimitem un mesaj „A” de la server la un client, clientul să recepționeze corect acel mesaj.

## Teste de sistem

Aceste teste presupun verificarea în ansamblu a sistemului. Aici vom folosi în principiu verificări manuale, referitoare la viteza de transfer a fișierelor, corectitudinea transmiterii, afișarea corectă a informațiilor în GUI și altele. Există și metode automate pentru această fază de testare, însă aplicația fiind de dimensiuni reduse, efortul dezvoltării unor teste automate de sistem ar fi mult prea mare față de efortul depus pentru cele câteva verificări manuale.