



UNIVERSITÀ
DEGLI STUDI
FIRENZE

UNIVERSITÀ DEGLI STUDI DI FIRENZE
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
ANNO ACCADEMICO 2024/2025

Relazione ServerTransfer

Autori:
Federico Monetti, Samuel Bruno

Corso:
Ingegneria del Software

N° Matricola:
7080562
7012352

Docente corso:
Enrico Vicario

Indice

1	Introduzione	4
1.1	Obiettivo	4
1.2	Architettura	5
1.2.1	Package Diagram	6
1.3	Strumenti e tecnologie utilizzate	6
2	Analisi e Progettazione	7
2.1	Casi d'uso	7
2.2	Use Case Templates	8
2.3	Class Diagram	10
2.4	Mockup	11
2.5	Page Navigation Diagram	13
2.6	Design Pattern utilizzati	14
2.6.1	Observer	14
2.6.2	Singleton	15
2.6.3	Command + Factory	15
3	Implementazione	16
3.1	Package Client	16
3.1.1	Classe Client	16
3.2	Package Server	16
3.2.1	Classe Server	16
3.2.2	Classe User	17
3.2.3	Classe UserAuthenticator	17
3.2.4	Classe ClientHandler	18
3.2.5	Package observers	19
Interfaccia Observer	19	
LoggerObserver	20	
Interfaccia Observable	20	
UserActionObservable	20	
AdminActionObservable	20	
DownloadObservable	20	
3.2.6	Package Commands	21
Classe CommandFactory	21	
Interfaccia Command	22	
Classe CdCommand	22	
Classe DownloadCommand	23	
Classe ListCommand	23	
Classe UploadCommand	24	
Classe DeleteCommand	25	

4 Testing del codice	27
4.1 Tecnologie utilizzate e struttura dei test	27
4.2 Codice di Test	27
4.2.1 white-box	27
UserAuthenticatorWhiteBoxTest	27
ServerWhiteBoxTest	28
ObserverWhiteBoxTest	29
4.2.2 black-box	29
AuthBlackBoxTest	30
CommandFactoryBlackBoxTest	30
FileOperationsBlackBoxTest	31
4.3 Risultati dei test	33

Capitolo 1

Introduzione

1.1 Obiettivo

Questo progetto consiste in un sistema software client-server per la condivisione e il trasferimento di file, sviluppato in linguaggio Java. L'applicazione è concepita per permettere a utenti remoti di accedere a un archivio di file centralizzato: un programma client si connette a un server dedicato, effettua un'autenticazione mediante credenziali e consente il download dei file autorizzati. Il sistema segue un modello simile a un semplice servizio FTP (File Transfer Protocol) personalizzato: il client può infatti collegarsi al server, eseguire il login con username e password, quindi richiedere e scaricare file dal server in base ai permessi assegnati. Gli utenti coinvolti nel sistema sono principalmente gli utenti finali che utilizzano l'applicazione client per accedere ai file condivisi. Ciascun utente deve essere preventivamente registrato (ossia le sue credenziali sono note al server) per poter effettuare il login ed usufruire del servizio. Non è previsto un utilizzo da parte di utenti non autenticati: tutte le funzionalità di download sono disponibili solo dopo il successo del login. Le funzionalità principali offerte dal sistema possono essere riassunte come segue:

- **Autenticazione degli utenti** : All'avvio il client richiede le credenziali all'utente e le invia al server, il quale verifica username e password confrontandoli con quelli memorizzati nell'archivio degli utenti autorizzati. Solo in caso di credenziali valide l'utente viene ammesso e può interagire con il sistema (in caso contrario il server nega l'accesso).
- **Visualizzazione dei file disponibili** : Dopo il login, il client può richiedere al server la lista dei file (o delle directory) a cui l'utente ha accesso. Il server risponde inviando l'elenco dei contenuti disponibili, permettendo così all'utente di sapere quali risorse può scaricare.
- **Download di file** : L'utente può selezionare, tramite un comando, uno dei file disponibili per scaricarlo. Il client invia quindi una richiesta di trasferimento al server, il quale trasmette il contenuto del file verso il client. Il trasferimento avviene in modo affidabile tramite protocollo TCP/IP, assicurando che il file giunga integro a destinazione.
- **Supporto multi-utente concorrente** : Il server è in grado di gestire più client contemporaneamente. Utenti diversi possono connettersi da postazioni diverse e usufruire del servizio in parallelo, senza interferire gli uni con gli altri. Ciò significa, ad esempio, che più utenti possono effettuare il download di file differenti allo stesso tempo, con il server che serve ciascuno attraverso connessioni dedicate.
- **Chiusura della connessione** : In qualsiasi momento l'utente può decidere di terminare la sessione. È prevista la funzionalità di logout/uscita sicura: il client invia

un segnale di chiusura e il server rilascia le risorse allocate per quel client, terminando il thread di servizio dedicato. In questo modo la connessione viene chiusa correttamente da entrambe le parti senza lasciare risorse impegnate.

Nel programma sono presenti due tipologie di utente:

- **User**: si può registrare e/o accedere per poi eseguire le operazioni di navigazione tra le cartelle, o il download di file.
- **Admin**: oltre a possedere tutte le operazioni dell'User, può anche aggiungere o rimuovere i file presenti all'interno del server

1.2 Architettura

L'applicazione adotta un'architettura multi-threaded client-server a due livelli:

- **Server**: un singleton che rimane in ascolto sulla porta TCP predefinita (12345), gestisce autenticazione, autorizzazione, gestione dei file e logging degli eventi;
- **Client**: un programma console che si connette al server, effettua il login e invia comandi testuali per navigare directory, elencare o scaricare file.

Il client interagisce con il server centralizzato per accedere a file condivisi. Il server resta in esecuzione su una macchina host, in ascolto su una porta predefinita (12345), mentre uno o più client remoti possono connettersi per effettuare l'autenticazione e utilizzare i servizi offerti. La comunicazione avviene su socket TCP mediante un protocollo testuale. Ad ogni nuova connessione il server crea un thread **ClientHandler** dedicato, abilitando il servizio concorrente di più client.

Ogni **ClientHandler**:

- gestisce autenticazione e registrazione attraverso **UserAuthenticator**, che utilizza un file di testo (**credentials.txt**) per salvare credenziali e ruoli;
- instrada i comandi (*list*, *cd <dir>*, *download <file>*, *upload <file>*, *delete <file>*, *exit*) tramite il pattern *Command*, implementato da una Factory dedicata;
- notifica gli observer (es. **LoggerObserver**) sugli eventi di download, secondo il pattern *Observer*;
- accede e modifica i file condivisi nella cartella **server_files**.

Il trasferimento dei file è affidabile: il server legge i byte del file, li codifica in Base64 e li invia come stringa al client; il client, tramite un listener asincrono, decodifica il payload e salva il file nella cartella locale.

Questa architettura, priva di database relazionali o GUI complesse, è stata progettata per garantire semplicità, chiarezza e facilità di estensione.

1.2.1 Package Diagram

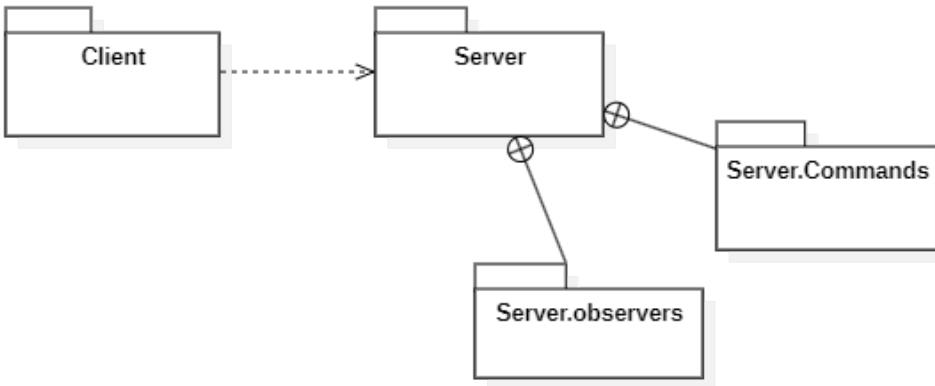


Figura 1.1: Package Diagram

Di seguito una breve descrizione delle responsabilità di ciascun package del progetto:

Client : Contiene la classe principale Client.java che gestisce l’interfaccia testuale, la connessione al server, il thread listener asincrono per le risposte e il salvataggio dei file scaricati in locale.

Server : Racchiude il cuore dell’applicazione server-side, incluse le classi Server.java (singleton), ClientHandler.java (thread per client), UserAuthenticator e User per la gestione dell’autenticazione.

Server.Commands : Implementa il pattern Command: contiene la CommandFactory e tutte le classi comando (ListCommand, DownloadCommand, UploadCommand, DeleteCommand, ecc.), ognuna responsabile dell’elaborazione di un’operazione specifica.

Server.Officers : Definisce il pattern Observer/Observable: include DownloadObservable, UserActionObservable, AdminActionObservable e le relative implementazioni di observer come LoggerObserver, per il logging centralizzato degli eventi di sistema.

1.3 Strumenti e tecnologie utilizzate

Il progetto è stato sviluppato in **Java** utilizzando l’IDE **IntelliJ IDEA**. I diagrammi UML sono stati realizzati con **StarUML** e i mockup delle interfacce (dove presenti) con **Figma**. La stesura della relazione e dei documenti è avvenuta in **LATEX** su **Overleaf**.

Capitolo 2

Analisi e Progettazione

Durante la fase di progettazione è stato realizzato un **diagramma UML** delle classi per visualizzare la struttura del sistema e le interazioni tra i vari componenti software. Inoltre, sono stati elaborati un **Use Case Diagram** e alcuni **Use Case Templates** per rappresentare il funzionamento di alcune delle principali operazioni utilizzabili nel programma.

2.1 Casi d'uso

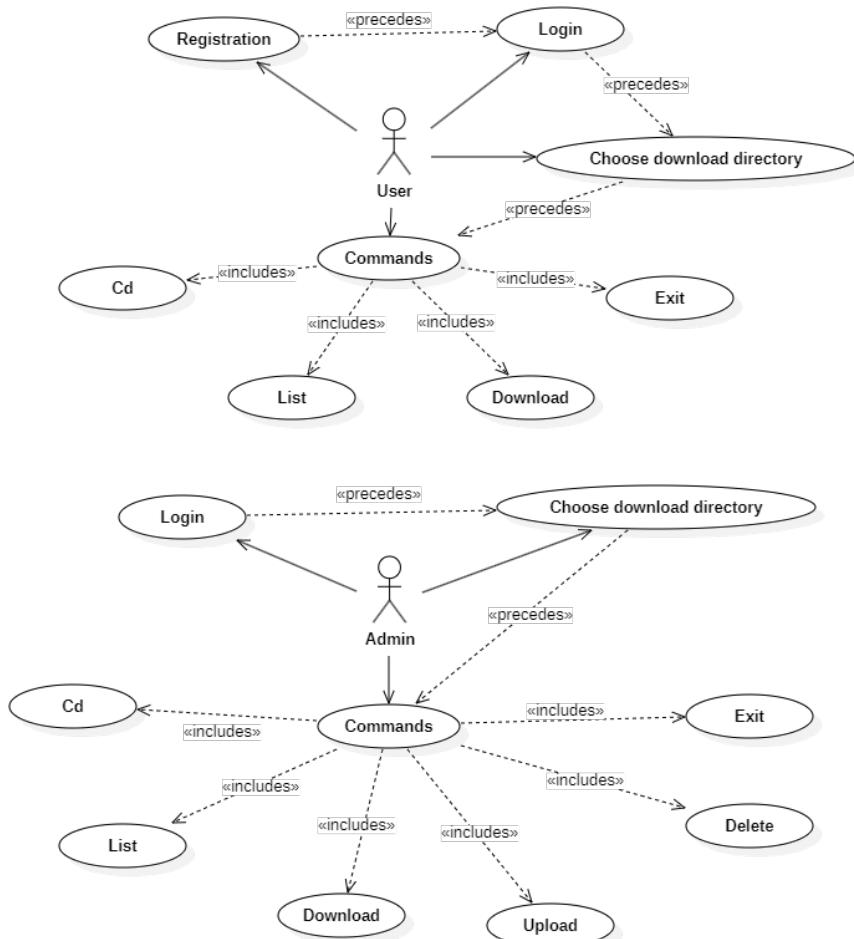


Figura 2.1: Use Case Diagram

2.2 Use Case Templates

In seguito sono mostrati degli Use Case Template che mostrano il flusso di esecuzione di alcuni casi d'uso rappresentati nella Figura 2.1. Sono stati scelti casi considerati più importanti o rappresentativi.

Use Case 1	Registrazione
Level	User goal
Description	Un utente effettua la registrazione
Actor	User
Pre-condition	L'utente deve essere nella pagina principale
Normal Flow	<ol style="list-style-type: none"> 1. L'utente inserisce username 2. Inserisce la password 3. Il Server verifica se l'username è già presente 4. Il Server aggiunge le credenziali alla lista delle credenziali 5. Il Server notifica l'avvenuta registrazione
Alternative Flow	<ol style="list-style-type: none"> 1. L'username è già presente, viene notificato un messaggio di errore 2. si ritenta la registrazione
Post-Conditions	L'utente è ora registrato e ha accesso a tutte le sue funzioni

Tabella 2.1: Use Case Template 1

Use Case 2	Download di un file
Level	User goal
Description	Un utente fa il download di un File
Actor	User
Pre-condition	L'utente è autenticato (ha completato login o registrazione con successo)
Normal Flow	<ol style="list-style-type: none"> 1. L'utente sceglie la cartella dove scaricare i file 2. L'utente sceglie la cartella all'interno del server dove è presente il file che vuole scaricare 3. esegue il comando di download, specificando il file da voler scaricare 4. il Server cerca il file e lo invia all'Utente 5. Il client salva il file nella directory scelta e viene mostrato un messaggio di conferma
Alternative Flow	<ol style="list-style-type: none"> 1. Il file specificato non esiste tra i file disponibili per il download 2. L'utente viene notificato con un messaggio di errore 3. si ritorna al menu principale
Post-Conditions	L'utente possiede adesso il file scaricato nella cartella scelta

Tabella 2.2: Use Case Template 2

Use Case 3	Upload di un file
Level	User goal
Description	Un Admin carica un File sul Server
Actor	Admin
Pre-condition	L'Admin deve aver eseguito il login
Normal Flow	<ol style="list-style-type: none"> 1. L'Admin effettua il comando di upload, specificando il file da voler caricare 2. il Server cerca se il file è già presente 3. Il Server salva il file nella directory dei file scaricabili 4. Viene notificato il caricamento completato
Alternative Flow	<ol style="list-style-type: none"> 1. Il file specificato esiste già tra i file disponibili per il download 2. L'Admin viene notificato con un messaggio di errore 3. si ritorna al menu principale
Post-Conditions	Il Server possiede adesso il nuovo file caricato dall'Admin

Tabella 2.3: Use Case Template 3

2.3 Class Diagram

Il diagramma delle classi riportato successivamente rappresenta le classi implementate e come queste interagiscono tra loro.

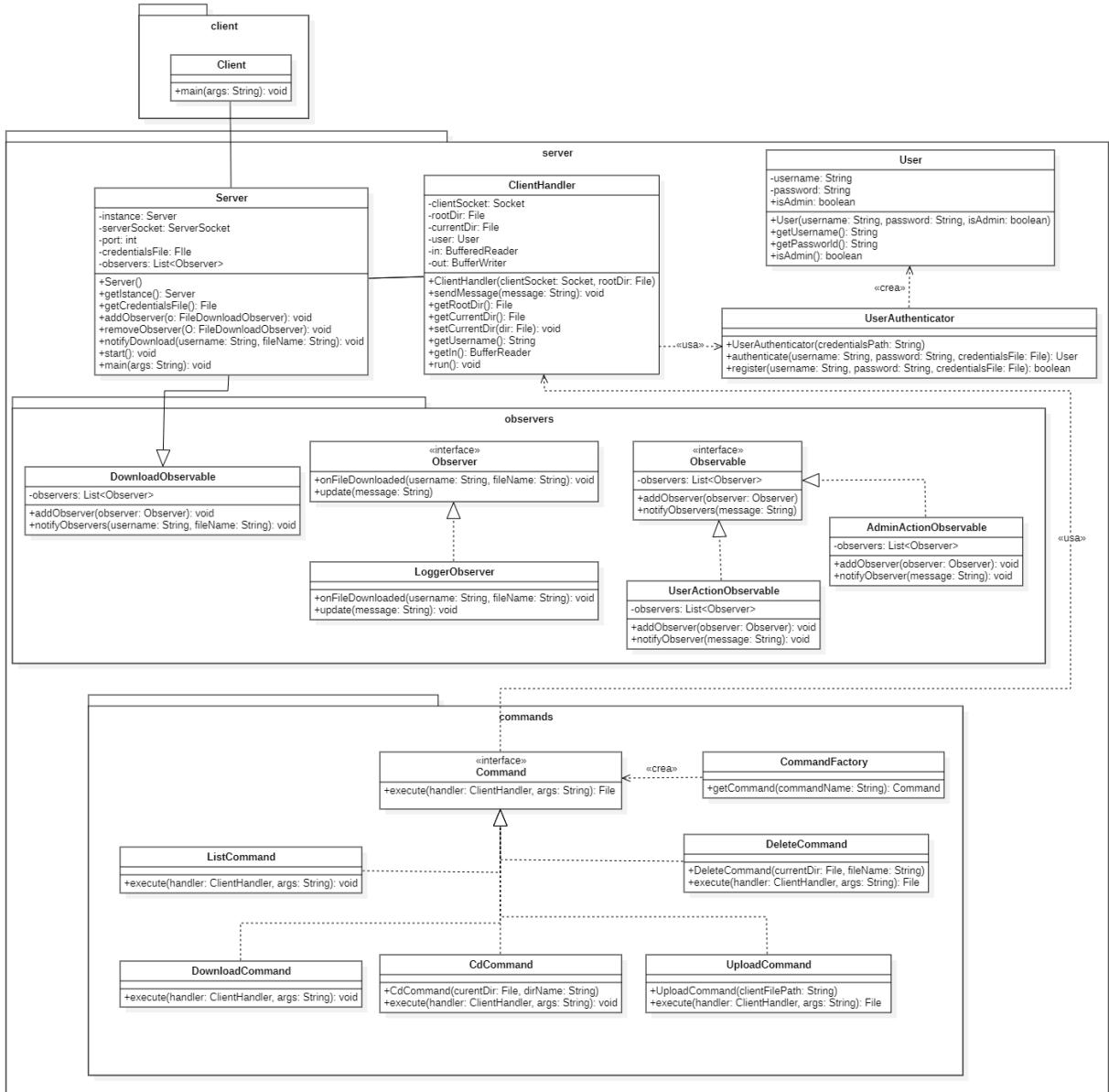


Figura 2.2: Diagramma UML delle classi

L'implementazione delle varie classi è spiegata e mostrata nel Capitolo 3

2.4 Mockup

Per facilitare la comprensione del flusso di utilizzo dell'applicazione, sono stati realizzati alcuni mockup delle schermate principali, rappresentanti un'ipotetica GUI per questo programma. In questa sezione vengono presentati i layout più significativi:

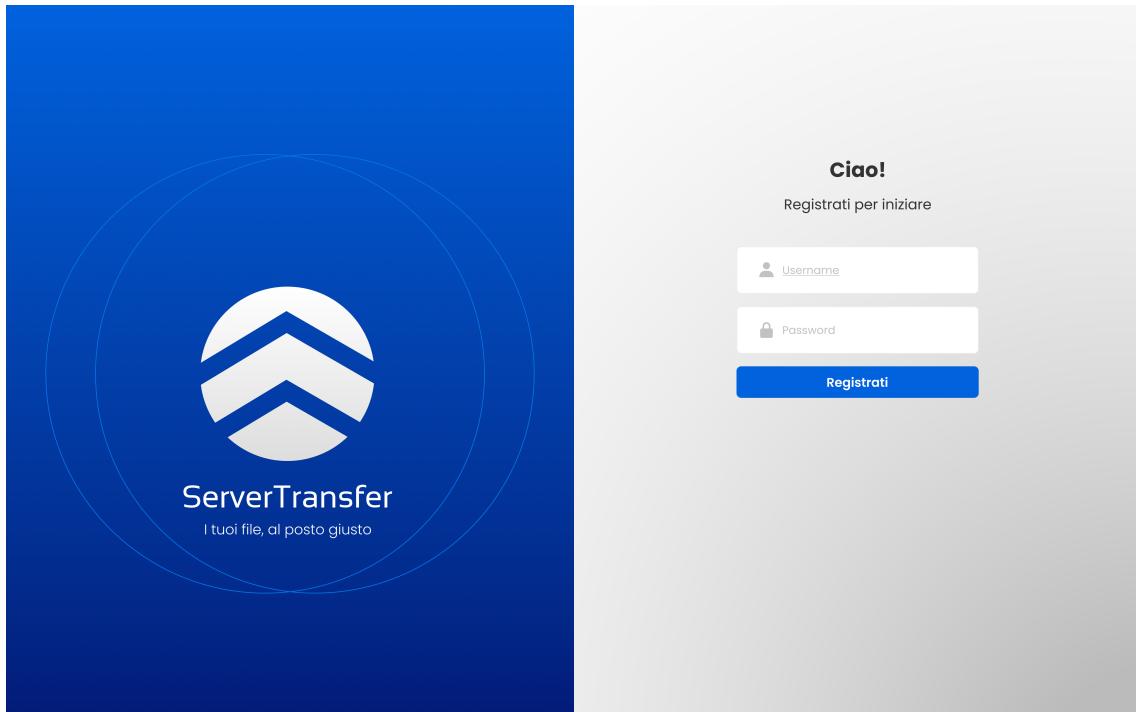


Figura 2.3: Mockup della schermata di registrazione

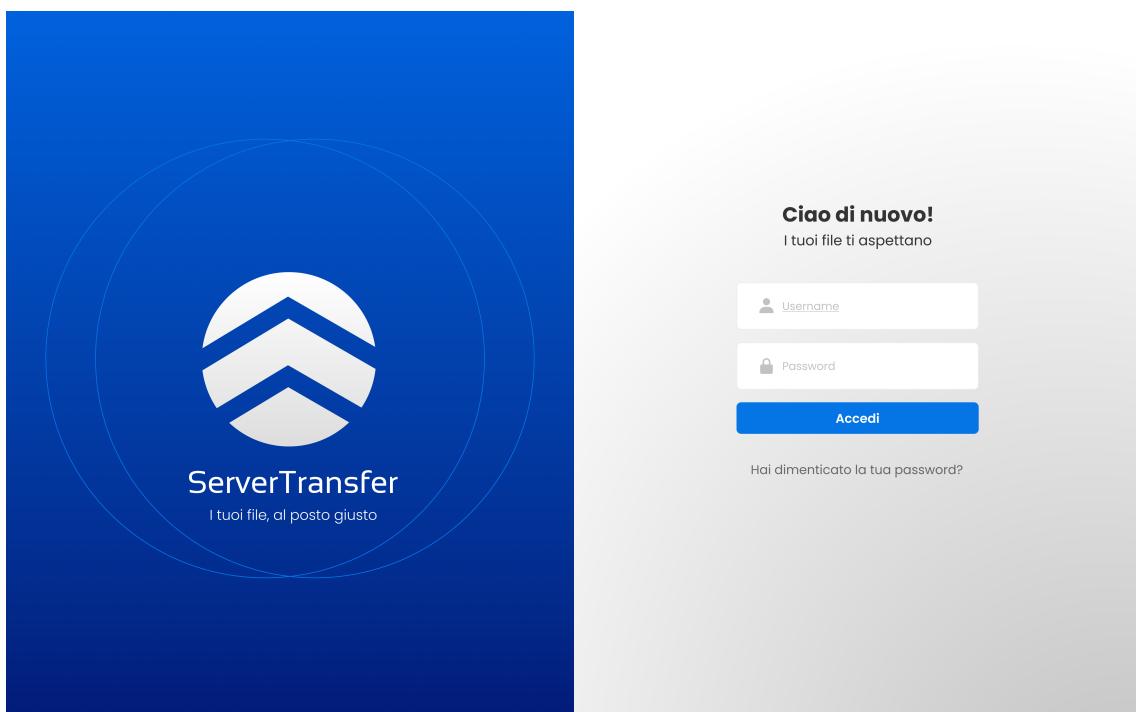


Figura 2.4: Mockup della schermata di accesso

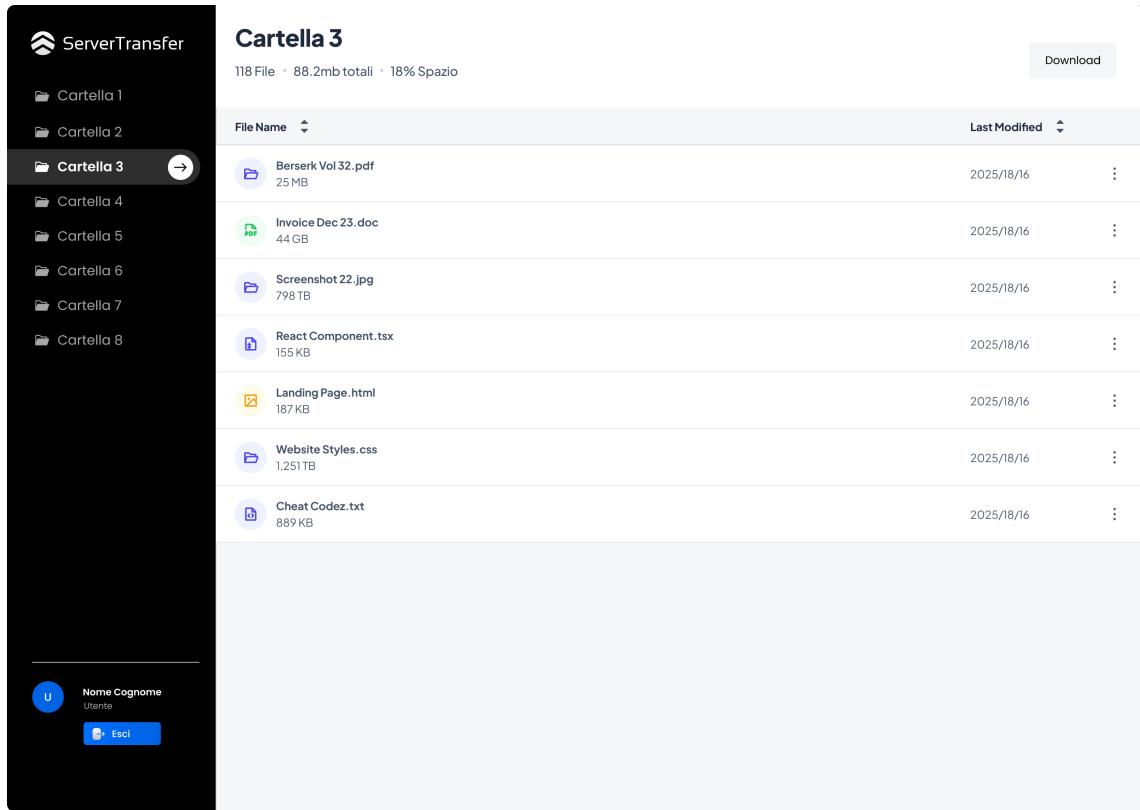


Figura 2.5: Mockup della Dashboard dell’User

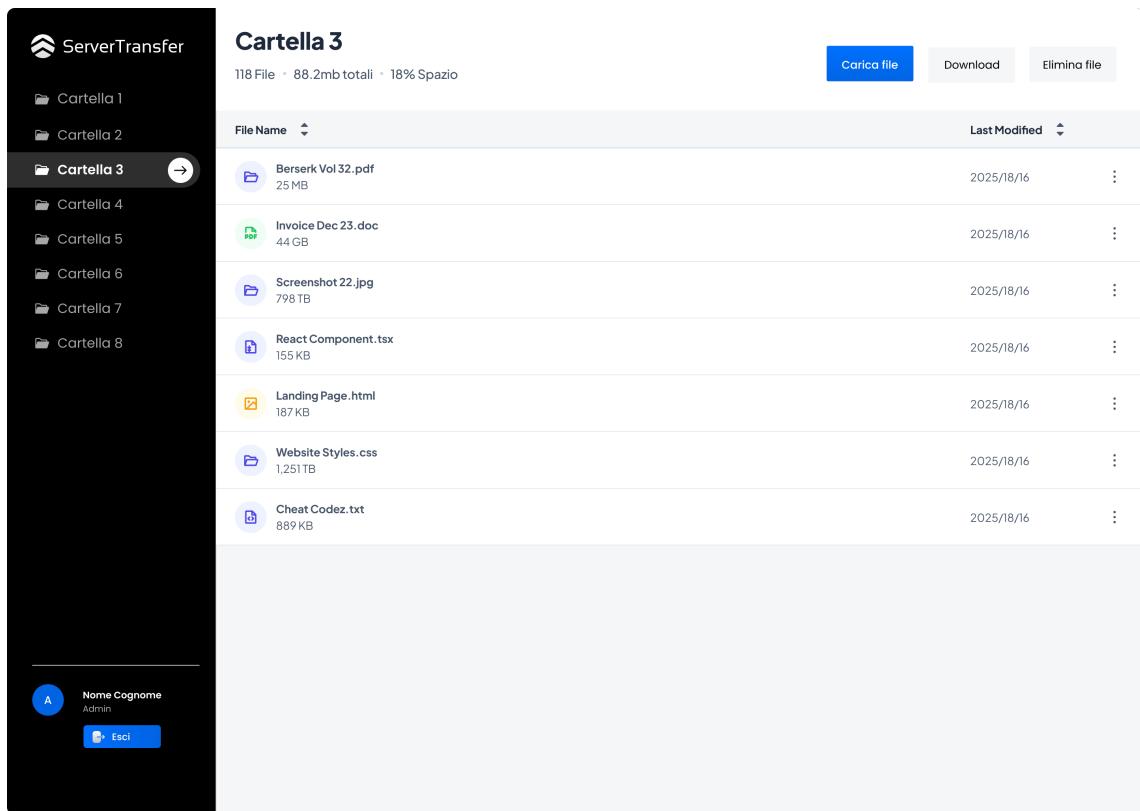


Figura 2.6: Mockup della Dashboard dell’Admin

2.5 Page Navigation Diagram

Immaginando di rendere disponibile un’interfaccia grafica completa all’utente abbiamo rappresentato la logica di navigazione tra alcune pagine del sistema. Si ha un Diagramma per l’User e uno per l’Admin.

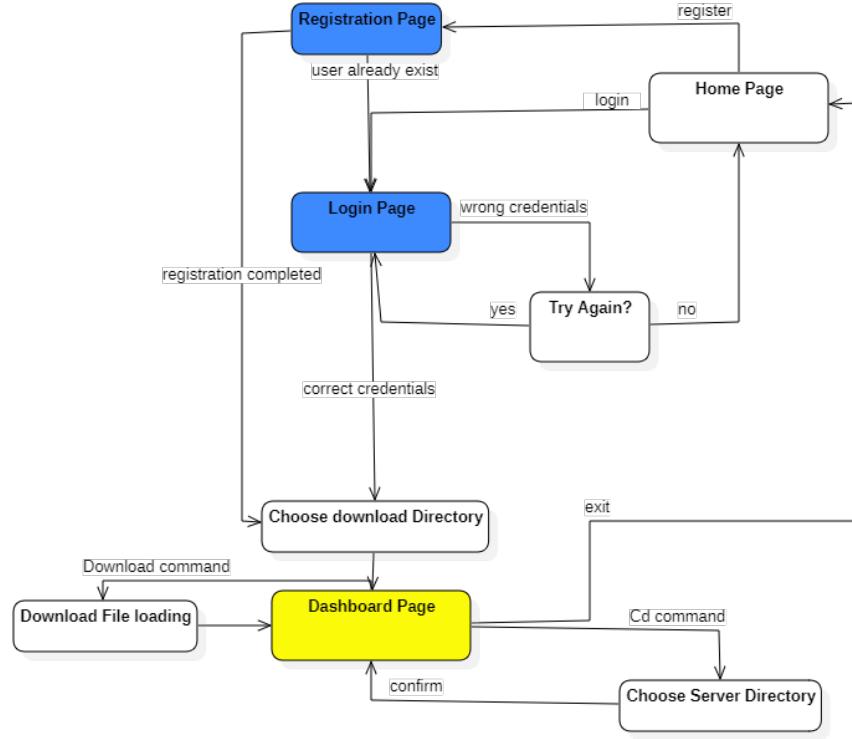


Figura 2.7: User Page Navigation Diagram

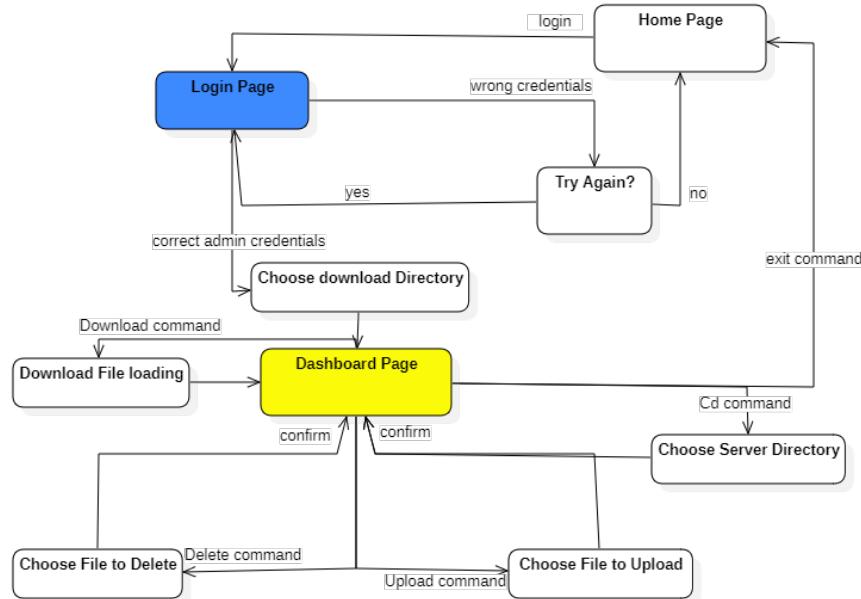


Figura 2.8: Admin Page Navigation Diagram

Le pagine colorate nelle immagini chiamate Registration Page, Login Page, Dashboard Page sono mostrate nei mockups in Sezione 2.4.

2.6 Design Pattern utilizzati

Nel corso dell’implementazione sono stati applicati alcuni design pattern noti, al fine di rendere il codice più modulare, riusabile e manutenibile. In particolare, si sono impiegati i seguenti pattern architetturali: Singleton, Factory e Observer. Di seguito se ne descrive l’utilizzo e le motivazioni.

2.6.1 Observer

Nel progetto il pattern *Observer* viene utilizzato per separare la logica delle operazioni dalle azioni di *logging* e monitoraggio. Si distinguono due “famiglie” di Observable:

DownloadObservable È una classe concreta (non implementa l’interfaccia `Observable`) perché la firma del metodo di notifica richiede due parametri (username e nome del file).

- Mantiene internamente una lista di `LoggerObserver` (oggetti che implementano il metodo `onFileDownloaded(String, String)`).
- Espone `addObserver(Observer)` per aggiungere un listener.
- Al termine di un download invoca `notifyObservers(username, fileName)`, che cicla su tutti gli observer e chiama `onFileDownloaded(...)`.

AdminActionObservable e UserActionObservable implementano l’interfaccia Observable.

Ciascuna:

- Mantiene una propria lista di `Observer` (che espongono `update(String message)`).
- Implementa `addObserver(Observer)` per l’iscrizione.
- Quando deve segnalare un evento generico (ad esempio un’azione amministrativa o un’azione utente), chiama `notifyObservers(message)`, che invoca `update(message)` su tutti gli observer registrati.

In entrambi i casi:

- I singoli `Observer` (per esempio `LoggerObserver`) si registrano chiamando `addObserver(...)` sulla specifica istanza di `Observable`.
- Quando si verifica l’evento (login, download, upload o delete), viene chiamato il corrispondente `notifyObservers(...)` che propaga l’informazione a tutti gli observer.

Questo approccio consente di:

- Aggiungere nuove tipologie di observer senza toccare la logica del server o dei comandi.
- Disaccoppiare il soggetto osservato dagli effetti collaterali, migliorando la manutenibilità e la testabilità del codice.

2.6.2 Singleton

Il pattern Singleton è stato adottato per la classe Server, che rappresenta l'istanza centrale del server in ascolto. È fondamentale che esista una sola istanza globale del server durante l'esecuzione, poiché questa gestisce:

- La lista di observer registrati.
- Il socket di ascolto.
- L'accesso al file delle credenziali.

L'implementazione prevede:

- Costruttore privato.
- Attributo statico privato `instance`.
- Metodo `getInstance()` che crea e restituisce l'unica istanza.

Questo garantisce un punto di accesso globale al server e impedisce la creazione di istanze multiple, che causerebbero conflitti nella gestione delle connessioni e risorse condivise.

2.6.3 Command + Factory

Il pattern Command è stato utilizzato per incapsulare ogni comando del client in un oggetto dedicato che ne gestisce l'esecuzione. Tutti i comandi implementano l'interfaccia `Command`, che espone il metodo `execute(ClientHandler handler, String[] args)`.

I comandi implementati sono:

- `ListCommand`: elenca i file nella directory corrente.
- `CdCommand`: consente la navigazione tra directory.
- `DownloadCommand`: gestisce il download di un file.
- `UploadCommand`, `DeleteCommand` (per gli admin): gestisce la rimozione o l'aggiunta di file dal server

Per creare dinamicamente l'oggetto comando corretto in base al testo inviato dal client, viene usato il pattern Factory:

- La classe `CommandFactory` espone il metodo statico `getCommand(String name)`, che restituisce l'istanza del comando associato alla stringa ricevuta ("list", "cd", "download" ...).

Questo approccio elimina la necessità di lunghi blocchi if/else o switch, semplifica l'aggiunta di nuovi comandi e mantiene separata la logica di parsing da quella esecutiva.

Capitolo 3

Implementazione

Nel diagramma delle classi sono stati identificati due package principali, client e server.. Di seguito sono descritte le principali classi e le loro responsabilità:

3.1 Package Client

3.1.1 Classe Client

Classe principale dell'applicazione client, contenente il main. All'avvio acquisisce dall'utente l'indirizzo del server (se non prestabilito) e tenta la connessione. Una volta connesso, gestisce l'interazione con l'utente locale: ad esempio, chiede di inserire username e password, li invia al server e attende la risposta. Se il login ha successo, presenta le operazioni disponibili (elencare file, scaricarne uno o uscire) e quindi invia i comandi appropriati al server, ricevendo e trattando le risposte. In caso di download, il Client riceve dallo stream di rete i dati del file e li salva su disco locale (eventualmente mostrando all'utente l'avanzamento). In pratica, questa classe coordina la UI e la comunicazione di rete.

3.2 Package Server

3.2.1 Classe Server

Classe principale del lato server. Contiene il metodo main che avvia il server sulla porta specificata e inizializza le risorse necessarie (es. caricamento delle credenziali autorizzate in memoria). Implementa il pattern Singleton per garantire che esista una sola istanza globale del server durante l'esecuzione: questo è fondamentale per gestire correttamente le connessioni, gli observer registrati e l'accesso centralizzato a risorse condivise. Il server apre un ServerSocket e rimane in ascolto di connessioni entranti. Per ogni nuova connessione accettata, istanzia un oggetto ClientHandler dedicato, che gestisce la comunicazione con il relativo client in un thread separato. Questo approccio consente una gestione concorrente di più client.

```
1 public void start() {
2     try {
3         ServerSocket serverSocket = new ServerSocket(port);
4         System.out.println("Server in ascolto sulla porta " + port)
5         ;
6         while (true) {
7             Socket clientSocket = serverSocket.accept();
8             new Thread(new ClientHandler(clientSocket, rootDir)).
9                 start();
10        }
11    }
```

```

9     } catch (IOException e) {
10         System.out.println("Errore nel server: " + e.getMessage())
11         );
12     } finally {
13         if (serverSocket != null && !serverSocket.isClosed()) {
14             try {
15                 serverSocket.close();
16             } catch (IOException e) { }
17         }
18     }
}

```

Listing 3.1: Metodo start() del server

3.2.2 Classe User

La classe User rappresenta un’entità utente all’interno del sistema, contenente le informazioni essenziali per l’autenticazione e la gestione dei privilegi. È utilizzata dal server per identificare l’utente connesso e verificare i permessi di accesso ai comandi.

Utilizzo nel sistema:

- Gli oggetti User vengono creati in fase di login, dopo aver letto le credenziali da un file o da una struttura dati interna.
- L’attributo isAdmin consente di abilitare o disabilitare comandi sensibili come upload o delete, accessibili solo agli amministratori.

3.2.3 Classe UserAuthenticator

Componente è responsabile della gestione delle credenziali utente lato server. Fornisce funzionalità sia per l’autenticazione di utenti esistenti che per la registrazione di nuovi, operando su un file di testo che funge da archivio persistente.

Espone i metodi:

- `authenticate(username, password)` che verifica se le credenziali fornite corrispondono a un utente registrato.
- `register(username, password, isAdmin)` che registra un nuovo utente verificando che l’username non esista già.

Responsabilità:

- Carica all’avvio del server la lista degli utenti autorizzati leggendo da un file di configurazione (ad esempio un file di testo con username, password e ruolo).
- Memorizza i dati utente in un file di testo per consentire controlli rapidi e sicuri. Ogni riga del file contiene `<username> <password> <ruolo>`
- Incapsula completamente la logica di accesso alle credenziali, permettendo al resto del sistema (es. ClientHandler) di autenticare utenti senza conoscere i dettagli dell’archiviazione.

```

1 public User authenticate(String username, String password) {
2     try (BufferedReader reader = new BufferedReader(new FileReader(
3         credentialsPath))) {
4         String line;
5         while ((line = reader.readLine()) != null) {
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
307
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343

```

```

5     String[] parts = line.trim().split("\\s+");
6     if (parts.length >= 3 &&
7         parts[0].trim().equals(username.trim()) &&
8         parts[1].trim().equals(password.trim())) {
9         boolean isAdmin = parts[2].trim().equalsIgnoreCase(
10            "admin");
11        return new User(username.trim(), password.trim(),
12                         isAdmin);
13    }
14    } catch (IOException e) {
15        System.out.println("Errore durante l'autenticazione: " + e.
16                           getMessage());
17    }
18    return null;
19 }
```

Listing 3.2: Metodo authenticate() del UserAuthenticator

3.2.4 Classe ClientHandler

Questa classe si occupa di gestire la comunicazione con un singolo client. Implementa l'interfaccia Runnable (o estende Thread), in modo che ogni istanza possa essere eseguita in parallelo rispetto alle altre. Al suo interno, il metodo run() contiene il loop di gestione della sessione: ricezione dei comandi dal client (tramite stream di input sul socket) e chiamata dei servizi appropriati sul server (ad esempio metodi di autenticazione o di invio file). ClientHandler funge dunque da controller per la singola connessione, interpretando i messaggi secondo il protocollo applicativo e utilizzando le componenti di business logic (autenticazione, accesso file) per soddisfare le richieste.

```

1 UserAuthenticator authenticator = new UserAuthenticator("credentials.txt");
2 boolean authenticated = false;
3
4 while (!authenticated) {
5     String risposta;
6     while (true) {
7         sendMessage("Sei registrato? (si/no):");
8         risposta = in.readLine();
9         if (risposta == null || risposta.trim().isEmpty()) continue
10            ;
11        if (risposta.equalsIgnoreCase("si") || risposta.
12            equalsIgnoreCase("no")) break;
13    }
14
15    if (risposta.equalsIgnoreCase("si")) {
16        out.write("Inserisci username:\n"); out.flush();
17        String usernameInput = in.readLine();
18        out.write("Inserisci password:\n"); out.flush();
19        String passwordInput = in.readLine();
20
21        user = authenticator.authenticate(usernameInput,
22                                         passwordInput);
23        if (user != null) {
24            authenticated = true;
25            out.write("Login effettuato con successo!\n");
26            out.write("Benvenuto " + user.getUsername() + (user.
27                  isAdmin() ? " [ADMIN]" : "") + "\n");
28        }
29    }
30 }
```

```

24         out.flush();
25     } else {
26         out.write("Credenziali errate.\n"); out.flush();
27     }
28 } else {
29     out.write("Registrazione - Inserisci username:\n"); out.
30         flush();
31     String newUser = in.readLine();
32     out.write("Inserisci password:\n"); out.flush();
33     String newPass = in.readLine();
34
35     if (authenticator.register(newUser, newPass, false)) {
36         user = authenticator.authenticate(newUser, newPass);
37         authenticated = true;
38         out.write("Registrazione completata con successo!\n");
39         out.write("Benvenuto " + user.getUsername() + "\n");
40         out.flush();
41     } else {
42         out.write("Errore: username già esistente.\n");
43         out.flush();
44     }
45 }

```

Listing 3.3: Gestione dell'autenticazione nel ClientHandler

```

1 String commandLine;
2 while ((commandLine = in.readLine()) != null) {
3     String trimmed = commandLine.trim();
4
5     if (trimmed.equalsIgnoreCase("exit")) {
6         sendMessage("Chiusura connessione. Arrivederci!");
7         break;
8     }
9
10    Command command = CommandFactory.getCommand(trimmed, currentDir,
11        user.isAdmin());
12    if (command != null) {
13        String[] parts = commandLine.trim().split("\\s+", 2);
14        currentDir = command.execute(this, parts);
15    } else {
16        sendMessage("Comando non valido oppure non hai l'-
17            autorizzazione.");
18    }
19 }

```

Listing 3.4: Loop principale di gestione dei comandi del ClientHandler

3.2.5 Package observers

Interfaccia Observer

L’interfaccia Observer definisce il contratto per tutti gli oggetti osservatori, che devono reagire a determinati eventi nel sistema. Include due metodi:

- update(String message): chiamato per notifiche generiche (es. azioni degli utenti o eventi di sistema).

- `onFileDownloaded(String username, String fileName)`: chiamato specificamente quando un utente scarica un file, per registrare o monitorare questa azione.

Questo consente di centralizzare e modularizzare il comportamento di logging o tracciamento.

```

1 public interface Observer {
2     void update(String message);
3     void onFileDownloaded(String username, String fileName);
4 }
```

Listing 3.5: Implementazione dell’interfaccia degli Observer

LoggerObserver

LoggerObserver è un’implementazione concreta dell’interfaccia Observer, responsabile del logging a console delle azioni rilevanti

Interfaccia Observable

L’interfaccia Observable rappresenta la parte osservabile nel pattern Observer, ovvero i soggetti che generano eventi (come le azioni di utenti o amministratori).

Definisce:

- Una lista di Observer che si registrano per ricevere notifiche.
- Il metodo `addObserver()` per aggiungere osservatori.
- Il metodo `notifyObservers(String message)` per inviare aggiornamenti agli osservatori registrati.

```

1 public interface Observable {
2     void addObserver(Observer observer);
3     void notifyObservers(String message);
4 }
```

Listing 3.6: Implementazione dell’interfaccia degli Observer

UserActionObservable

UserActionObservable è un’implementazione concreta dell’interfaccia Observable, responsabile di notificare la lista di observer registrati quando un User esegue una qualsiasi operazione.

AdminActionObservable

AdminActionObservable è un’implementazione concreta dell’interfaccia Observable, responsabile di notificare la lista di observer registrati quando un Admin esegue una qualsiasi operazione.

DownloadObservable

DownloadObservable è una classe che rappresenta un oggetto osservabile specializzato nel contesto del download, ma non implementa direttamente l’interfaccia Observable tradizionale. Questo perché il suo metodo `notify()` differisce leggermente dalla versione standard definita nell’interfaccia Observable:

```

1 void notifyObservers(String username, String fileName){
2     for (Observer observer : observers) {
3         observer.onFileDownloaded(username, fileName);
4     }
5 }
```

Listing 3.7: Implementazione dell’interfaccia degli Observer

Inoltre, DownloadObservable viene estesa dalla classe Server, che ne eredita la funzionalità di notifica e la integra nel contesto più ampio della gestione server, mantenendo così un controllo più fine sui meccanismi di comunicazione degli eventi relativi al download.

3.2.6 Package Commands

Il package Server.Commands incapsula la logica relativa all’esecuzione dei comandi testuali inviati dal client al server. Ogni comando (es. list, cd, upload, download, delete) è modellato come una classe separata che implementa un’interfaccia comune, promuovendo modularità, estensibilità e aderenza al design pattern Command.

Questa struttura consente di:

- separare la gestione dei comandi dalla logica di rete;
- aggiungere facilmente nuovi comandi (basta creare una nuova classe che implementa Command);
- gestire autorizzazioni e controlli specifici per ogni comando.

Classe CommandFactory

CommandFactory è una factory class responsabile dell’istanziazione dinamica dei comandi sulla base della stringa ricevuta dal client. Il metodo:

```

1 public static Command getCommand(String commandLine, File
2     currentDir, boolean isAdmin){
3     String [] parts = commandLine.split(" ");
4     String command = parts[0];
5
6     switch (command.toLowerCase()) {
7         case "list":
8             return new ListCommand(currentDir);
9         case "cd":
10            if (parts.length > 1) {
11                return new CdCommand(currentDir, parts[1]);
12            }
13            break;
14        case "download":
15            if (parts.length > 1) {
16                return new DownloadCommand(currentDir, parts
17                    [1]);
18            }
19            break;
20        case "upload":
21            if (isAdmin && parts.length > 1) {
22                return new UploadCommand(parts[1]);
23            }
24            break;
25        case "delete":
26            if (isAdmin && parts.length > 1) {
```

```

25         return new DeleteCommand(currentDir, parts[1]);
26     }
27     break;
28 case "exit":
29     break;
30 }
31 return null;
32 }
```

Listing 3.8: metodo getCommand della CommandFactory

- Analizza il comando testuale (commandLine) ricevuto dal client.
- Verifica la validità degli argomenti e se l'utente ha i permessi necessari (es. alcuni comandi come upload e delete sono riservati agli admin).
- Restituisce un'istanza della classe comando corrispondente (ListCommand, CdCommand, DownloadCommand, ecc.), oppure null se il comando non è valido.

La CommandFactory incapsula la logica di selezione del comando, evitando if-else o switch sparsi nel codice del server, facilitando la manutenibilità.

Interfaccia Command

L'interfaccia Command definisce un contratto per tutte le classi comando. Contiene un unico metodo:

```

1 public interface Command {
2     File execute(ClientHandler handler);
3 }
```

Listing 3.9: metodo getCommand della CommandFactory

- handler: l'oggetto ClientHandler associato alla sessione client attuale.
- Il valore restituito è una cartella (tipo File) che può rappresentare una modifica allo stato corrente del client (es. cambio directory).

Questo approccio consente al server di eseguire comandi in modo polimorfico, senza conoscere i dettagli dell'implementazione specifica.

Classe CdCommand

Implementazione concreta dell'Interfaccia Command. Implementa il comando per navigare all'interno delle directory dell'archivio dei file:

```

1 public File execute(ClientHandler handler) {
2     File newDir = new File(currentDir, dirName);
3     if (newDir.exists() && newDir.isDirectory()) {
4         handler.sendMessage("Directory cambiata in: " + newDir.
5             getName());
6         try {
7             UserActionObservable observable = new
8                 UserActionObservable();
9             observable.addObserver(new LoggerObserver());
10            observable.notifyObservers("L'utente ha cambiato la
11                directory in: " + newDir.getAbsolutePath());
```

```

8     } catch (Exception e) {
9         System.err.println("Errore logging cd: " + e.getMessage()
10            ());
11    }
12    return newDir;
13  } else {
14    handler.sendMessage("Directory non trovata.");
15  }
16}

```

Listing 3.10: metodo getCommand della CommandFactory

Classe DownloadCommand

Implementazione concreta dell'Interfaccia Command. Implementa il comando per scaricare in locale i file presenti all'interno dell'archivio del server:

```

1 public File execute(ClientHandler handler) {
2     File file = new File(currentDir, fileName);
3     try {
4         if (!file.exists() || !file.isFile()) {
5             handler.sendMessage("File non trovato.");
6             return currentDir;
7         }
8         byte[] fileBytes = Files.readAllBytes(file.toPath());
9         String encoded = Base64.getEncoder().encodeToString(
10            fileBytes);
11        handler.sendMessage("FILE_CONTENT:" + file.getName() + ":" +
12            encoded);
13        try {
14            DownloadObservable observable = new DownloadObservable
15                ();
16            observable.addObserver(new LoggerObserver());
17            observable.notifyObservers(handler.getUsername(), file.
18                getName());
19        } catch (Exception e) {
20            System.err.println("Errore nel logging degli observers:
21                " + e.getMessage());
22        }
23    } catch (IOException e) {
24        handler.sendMessage("Errore nel download del file: " + e.
25            getMessage());
26        e.printStackTrace(); // stampa lato server
27    }
28    return currentDir;
29}

```

Listing 3.11: metodo getCommand della CommandFactory

Classe ListCommand

Implementazione concreta dell'Interfaccia Command. Implementa il comando per visualizzare la lista di file scaricabili all'interno della directory del server corrente:

```

1 public File execute(ClientHandler handler) {
2     File[] files = currentDir.listFiles();
3     if (files == null || files.length == 0) {

```

```

4     handler.sendMessage("La directory e' vuota.");
5 } else {
6     StringBuilder sb = new StringBuilder();
7     for (File file : files) {
8         sb.append(file.getName()).append(file.isDirectory() ? "
9             /" : "").append("\n");
10    }
11    String list = sb.toString().trim(); // rimuove l'ultimo \n
12    handler.sendMessage(list);
13}
14try {
15    UserActionObservable observable = new UserActionObservable();
16    observable.addObserver(new LoggerObserver());
17    observable.notifyObservers("Utente ha visualizzato il
18        contenuto di: " + currentDir.getAbsolutePath());
19} catch (Exception e) {
20    System.err.println("Errore logging list: " + e.getMessage());
21}
22return currentDir;
23}

```

Listing 3.12: metodo getCommand della CommandFactory

Classe UploadCommand

Implementazione concreta dell’Interfaccia Command. Implementa il comando, esclusivo per l’Admin, per aggiungere nuovi file all’interno dell’archivio del server:

```

1 public File execute(ClientHandler handler) {
2     // Estrai solo il nome del file
3     String fileName = new File(clientFilePath).getName();
4
5     // Cartella di destinazione fissa
6     File dest = new File(handler.getRootDir(), fileName);
7
8     // Se gia' esiste sul server
9     if (dest.exists()) {
10         handler.sendMessage("Il file '" + fileName + "' esiste gia',
11             sul server.");
12         return handler.getCurrentDir();
13     }
14
15     // Prompt al client per inviare i dati
16     handler.sendMessage("Pronto per ricevere il file.");
17
18     try {
19         // Leggi la riga intera inviata dal client
20         String uploadLine = handler.getIn().readLine();
21         if (uploadLine == null || !uploadLine.startsWith(
22             "FILE_UPLOAD:")) {
23             handler.sendMessage("Formato di upload non valido.");
24             return handler.getCurrentDir();
25         }
26     }
27 }

```

```

25     // Splitta in tre parti: [0]="FILE_UPLOAD", [1]=nomeFile,
26     // [2]=base64Data
27     String [] parts = uploadLine.split(":", 3);
28     if (parts.length < 3) {
29         handler.sendMessage("Dati di upload corrotti.");
30         return handler.getCurrentDir();
31     }
32     String base64Data = parts[2];
33
34     // Decodifica e salva
35     byte[] fileBytes = Base64.getDecoder().decode(base64Data);
36     Files.write(dest.toPath(), fileBytes);
37
38     handler.sendMessage("Upload completato con successo: " +
39                         fileName);
40
41     // Logging osservatori
42     AdminActionObservable obs = new AdminActionObservable();
43     obs.addObserver(new LoggerObserver());
44     obs.notifyObservers("Admin ha caricato: " + dest.
45                         getAbsolutePath());
46 } catch (Exception e) {
47     handler.sendMessage("Errore durante l'upload: " + e.
48                         getMessage());
49     System.err.println("Errore in UploadCommand: " + e.
50                         getMessage());
51 }
52 return handler.getCurrentDir();
53 }
```

Listing 3.13: metodo getCommand della CommandFactory

Classe DeleteCommand

Implementazione concreta dell’Interfaccia Command. Implementa il comando, esclusivo per l’Admin, per rimuovere eventuali file dall’archivio del server:

```

1 public File execute(ClientHandler handler) {
2     File file = new File(currentDir, fileName);
3     if (!file.exists()) {
4         handler.sendMessage("File non trovato.");
5     } else if (file.delete()) {
6         handler.sendMessage("File eliminato.");
7         try {
8             AdminActionObservable observable = new
9                 AdminActionObservable();
10            observable.addObserver(new LoggerObserver());
11            observable.notifyObservers("Admin ha eliminato il file:
12                            " + file.getAbsolutePath());
13        } catch (Exception e) {
14            System.err.println("Errore logging delete: " + e.
15                            getMessage());
16        }
17    } else {
18        handler.sendMessage("Tentativo di eliminazione fallito.");
19    }
20    return currentDir;
21 }
```

Listing 3.14: metodo getCommand della CommandFactory

Capitolo 4

Testing del codice

4.1 Tecnologie utilizzate e struttura dei test

Per garantire l'affidabilità e la correttezza del software sviluppato, è stata sviluppato del codice di testing, adottando una strategia mista che combina sia test **white-box** che **black-box**. L'obiettivo è stato quello di verificare sia la corretta implementazione interna del codice che il corretto comportamento esterno rispetto alle specifiche funzionali.

I test sono stati realizzati utilizzando il framework **JUnit 5**, integrato all'interno del progetto tramite il sistema di build **Maven**. L'esecuzione automatica dei test è stata gestita attraverso il plugin **Maven Surefire**, che consente di integrare la fase di testing nel ciclo di build e verificare l'effettiva copertura delle funzionalità implementate. Per una maggiore leggibilità e precisione nell'espressione delle asserzioni, è stata inoltre utilizzata la libreria **AssertJ**, che offre un set di metodi fluenti e intuitivi per la verifica delle condizioni nei test.

Nel dettaglio:

- I test **white-box** sono stati utilizzati per verificare il corretto funzionamento di singoli metodi, condizioni logiche e percorsi di esecuzione interni. Questo ha permesso di individuare e correggere potenziali bug a livello di implementazione.
- I test **black-box** si sono concentrati invece sul comportamento esterno delle componenti principali, validando l'interazione tra input e output senza considerare i dettagli interni dell'implementazione.

Questo approccio ha permesso di ottenere una copertura dei test bilanciata e completa, assicurando il corretto funzionamento delle funzionalità principali e una maggiore robustezza del sistema.

4.2 Codice di Test

4.2.1 white-box

Questi test accedono alla struttura interna del codice, verificando comportamenti specifici, percorsi condizionali e struttura dati. Il codice di test di tipo white-box è stato suddiviso in tre classi di testing:

UserAuthenticatorWhiteBoxTest

Obiettivo: Verificare il comportamento interno della classe UserAuthenticator.

Cosa verifica:

- Che il costruttore crei il file di credenziali se non esiste.

- Che il metodo register scriva correttamente la riga nel file.
- Che authenticate restituisca un oggetto User coerente con i dati memorizzati.
- Che authenticate ritorni null con password errata.

```

1 class UserAuthenticatorWhiteBoxTest {
2     private static final String CRED_FILE = "test_credentials_wb.
3         txt";
4     private UserAuthenticator auth;
5     @BeforeEach
6     void init() throws IOException {
7         Files.deleteIfExists(Path.of(CRED_FILE));
8         auth = new UserAuthenticator(CRED_FILE);
9     }
10    @AfterEach
11    void cleanup() throws IOException {
12        Files.deleteIfExists(Path.of(CRED_FILE));
13    }
14    @Test
15    void constructorCreatesFileIfMissing() {
16        assertThat(Files.exists(Path.of(CRED_FILE)))
17            .as("Il costruttore deve creare il file delle
18                credenziali")
19            .isTrue();
20    }
21    @Test
22    void registerAppendsCorrectLine() throws IOException {
23        auth.register("federico", "pass", true);
24        String content = Files.readString(Path.of(CRED_FILE));
25        assertThat(content).contains("federico pass admin");
26    }
27    @Test
28    void authenticateReturnsCorrectUserObject() {
29        auth.register("samuel", "pw", false);
30        User u = auth.authenticate("samuel", "pw");
31        assertThat(u)
32            .satisfies(user -> {
33                assertThat(user.getUsername()).isEqualTo("samuel");
34                assertThat(user.isAdmin()).isFalse();
35            });
36    }
37    @Test
38    void authenticateReturnsNullOnBadPwd() {
39        auth.register("egidio", "pwd", false);
40        assertThat(auth.authenticate("egidio", "wrong")).isNull();
41    }
42 }
```

Listing 4.1: classe di test UserAuthenticatorWhiteBoxTest

ServerWhiteBoxTest

Obiettivo: Verificare il corretto funzionamento del singleton Server.

Cosa verifica:

- Che getInstance() restituisca sempre la stessa istanza.

- Che il file delle credenziali sia correttamente inizializzato.

```

1 class ServerWhiteBoxTest {
2     @Test
3     void singletonBehaviorAndCredentialsFilePath() {
4         Server s1 = Server.getInstance();
5         Server s2 = Server.getInstance();
6         assertThat(s1).isSameAs(s2);
7         String credPath = s1.getCredentialsFile().getPath();
8         assertThat(credPath)
9             .as("Il server deve avere un file di credenziali
10                configurato")
11            .isNotEmpty();
12 }

```

Listing 4.2: classe di test ServerWhiteBoxTest

ObserverWhiteBoxTest

Obiettivo: Controllare che gli observer vengano notificati correttamente in caso di download.

Cosa verifica:

- Che l'aggiunta di un observer funzioni.
- Che la chiamata a notifyDownload() invochi effettivamente onFileDownloaded() sugli observer registrati.

```

1 class ObserverWhiteBoxTest {
2     private static class SpyObserver extends LoggerObserver {
3         boolean notified = false;
4         @Override
5         public void onFileDownloaded(String user, String fname) {
6             notified = true;
7         }
8     }
9     @Test
10    void notifyDownloadInvokesObserverMethod() {
11        Server srv = Server.getInstance();
12        SpyObserver spy = new SpyObserver();
13        srv.addObserver(spy);
14
15        srv.notifyDownload("utenteTest", "report.pdf");
16        assertThat(spy.notified).as("L'Observer deve essere
17            notificato in caso di download").isTrue();
18    }

```

Listing 4.3: classe di test ObserverWhiteBoxTest

4.2.2 black-box

Questi test verificano il comportamento osservabile del sistema senza conoscere la struttura interna del codice, simulando il punto di vista di un utente o altro sistema esterno. Il codice di test di tipo black-box è stato suddiviso in tre classi di testing:

AuthBlackBoxTest

Obiettivo: Testare la registrazione e l'autenticazione degli utenti tramite l'interfaccia pubblica di UserAuthenticator.

Cosa verifica:

- La possibilità di registrare un utente e accedere con credenziali corrette.
- Il blocco della registrazione di un utente con username già esistente.
- Il fallimento dell'autenticazione con password sbagliata o username inesistente.

```
1 class AuthBlackBoxTest {
2     private static final String TEST_CRED_FILE = "test_credentials_bb.txt";
3     private UserAuthenticator auth;
4     @BeforeEach
5     void setUp() throws IOException {
6         // Assicuriamoci di partire sempre da un file vuoto
7         Files.deleteIfExists(Path.of(TEST_CRED_FILE));
8         auth = new UserAuthenticator(TEST_CRED_FILE);
9     }
10    @AfterEach
11    void tearDown() throws IOException {
12        Files.deleteIfExists(Path.of(TEST_CRED_FILE));
13    }
14    @Test
15    void registerNewUserAndLoginSuccess() {
16        assertTrue(auth.register("aaa", "pwd123", false));
17        User u = auth.authenticate("aaa", "pwd123");
18        assertNotNull(u);
19        assertEquals("aaa", u.getUsername());
20        assertFalse(u.isAdmin());
21    }
22    @Test
23    void cannotRegisterSameUserTwice() {
24        assertTrue(auth.register("alice", "secret", true));
25        assertFalse(auth.register("alice", "secret", true));
26    }
27    @Test
28    void loginWithWrongPasswordFails() {
29        auth.register("testLogin", "pwd", false);
30        assertNull(auth.authenticate("testLogin", "wrongpwd"));
31    }
32    @Test
33    void loginNonexistentUserFails() {
34        assertNull(auth.authenticate("pippo", "nopass"));
35    }
36 }
```

Listing 4.4: classe di test AuthBlackBoxTest

CommandFactoryBlackBoxTest

Obiettivo: Verificare la corretta creazione dei comandi in base al ruolo utente (admin o normale).

Cosa verifica:

- Gli utenti normali possono eseguire solo list, cd e download.
- Gli admin possono anche upload e delete.
- Comandi non validi restituiscono null.

```

1 class CommandFactoryBlackBoxTest {
2     private final File root = new File("server_files");
3     @Test
4     void regularUserCanUseOnlyListCdDownload() {
5         assertTrue(CommandFactory.getCommand("list", root, false)
6             instanceof ListCommand);
7         assertTrue(CommandFactory.getCommand("cd subdir", root,
8             false) instanceof CdCommand);
9         assertTrue(CommandFactory.getCommand("download file.txt",
10            root, false) instanceof DownloadCommand);
11    }
12    @Test
13    void regularUserCannotUploadOrDelete() {
14        assertNull(CommandFactory.getCommand("upload file.txt",
15            root, false));
16        assertNull(CommandFactory.getCommand("delete file.txt",
17            root, false));
18    }
19    @Test
20    void adminCanUseUploadAndDelete() {
21        assertTrue(CommandFactory.getCommand("upload new.txt", root
22            , true) instanceof UploadCommand);
23        assertTrue(CommandFactory.getCommand("delete old.txt", root
24            , true) instanceof DeleteCommand);
25    }
26    @Test
27    void unknownCommandsReturnNull() {
28        assertNull(CommandFactory.getCommand("testcomando", root,
29            true));
30        assertNull(CommandFactory.getCommand("", root, false));
31    }
32 }
```

Listing 4.5: classe di test CommandFactoryBlackBoxTest

FileOperationsBlackBoxTest

Obiettivo: Testare direttamente le funzionalità principali legate ai file dal punto di vista utente.

Cosa verifica:

- Il download effettivo dei file da server_files/ a downloaded_files/.
- L'upload corretto da file_to_upload/ a server_files/ solo da admin.
- La rimozione di file da server_files/ solo da parte di un admin.

```

1 class FileOperationsBlackBoxTest {
2     private final File root = new File("server_files");
3     private final User admin = new User("a", "p", true);
4     private final User user = new User("u", "p", false);
5     void downloadCommandSendsBase64Content() throws Exception {
```

```

6    // 1) Assicuriamoci che il file esista
7    Path src = Path.of("server_files/file.txt");
8    assertTrue(Files.exists(src), "server_files/file.txt deve
9      esistere");
10
11   // 2) Recuperiamo il comando
12   Command cmd = CommandFactory.getCommand("download file.txt"
13     , root, false);
14   assertNotNull(cmd);
15   assertTrue(cmd instanceof DownloadCommand,
16     "getCommand su 'download' deve restituire
17     DownloadCommand");
18
19   // 3) Creiamo il nostro handler di test che registra i
20   // messaggi
21   ClientHandlerForTesting handler = new
22     ClientHandlerForTesting(root);
23
24   // 4) Eseguiamo il comando
25   File after = cmd.execute(handler);
26
27   // 5) La directory corrente non cambia
28   assertEquals(root.getCanonicalFile(), after.
29     getCanonicalFile());
30
31   // 6) Verifichiamo fra i messaggi registrati quello con
32   // FILE_CONTENT
33   List<String> msgs = handler.getMessages();
34   assertFalse(msgs.isEmpty(), "handler deve aver ricevuto
35     almeno un messaggio");
36
37   boolean found = msgs.stream()
38     .anyMatch(m -> m.startsWith("FILE_CONTENT:file.txt:
39       "));
40   assertTrue(found, "Deve esserci un messaggio che inizia con
41     'FILE_CONTENT:file.txt:'");
42 }
43 void uploadAddsFileOnlyForAdmin() throws Exception {
44   // Assicuriamoci che esista il file di test da "caricare"
45   Path up = Path.of("file_to_upload/upload_test.txt");
46   assertTrue(Files.exists(up), "file_to_upload/upload_test.
47     txt deve esistere");
48
49   // Prepara i dati Base64 per quel file
50   byte[] bytes = Files.readAllBytes(up);
51   String base64 = Base64.getEncoder().encodeToString(bytes);
52
53   // Costruiamo il comando upload
54   Command cmdAdmin = CommandFactory.getCommand("upload
55     upload_test.txt", root, true);
56   assertNotNull(cmdAdmin, "Admin deve poter ottenere l'
57     UploadCommand");
58
59   // Creiamo il nostro handler di test, che risponde con la
60   // riga di upload
61   class UploadTestHandler extends ClientHandler {
62     UploadTestHandler(File rootDir) {
63       super((Socket) null, rootDir);

```

```

50         // preparo il reader con la riga "FILE_UPLOAD:
51         upload_test.txt:<base64>" +
52         String line = "FILE_UPLOAD:upload_test.txt:" +
53         base64 + "\n";
54         this.in = new BufferedReader(new StringReader(line))
55         );
56     }
57     @Override
58     public void sendMessage(String message) {
59         // no-op: ignoriamo i prompt
60     }
61     // Usa il handler per eseguire
62     UploadTestHandler handler = new UploadTestHandler(root);
63     File after = cmdAdmin.execute(handler);
64     // Ora il file deve esistere in server_files
65     assertTrue(Files.exists(Path.of("server_files/upload_test.
66     txt")),
67     "Il file deve finire in server_files");
68     assertEquals(root.getCanonicalFile(), after.
69     getCanonicalFile());
70     // Puliamo per non lasciare tracce
71     Files.deleteIfExists(Path.of("server_files/upload_test.txt"
72     ));
73
74     void deleteRemovesFileOnlyForAdmin() throws Exception {
75     Path toDelete = Path.of("server_files/tmp.txt");
76     Files.writeString(toDelete, "temp");
77     assertTrue(Files.exists(toDelete));
78
79     // Utente normale non ottiene il comando
80     assertNull(CommandFactory.getCommand("delete tmp.txt", root
81     , false));
82
83     // Comando delete per admin
84     Command cmd = CommandFactory.getCommand("delete tmp.txt",
85     root, true);
86     assertNotNull(cmd);
87
88     ClientHandlerForTesting adminHandler = new
89     ClientHandlerForTesting(root);
90     File afterDelete = cmd.execute(adminHandler);
91
92     assertFalse(Files.exists(toDelete), "Il file deve essere
93     rimosso da server_files");
94     assertEquals(root.getCanonicalFile(), afterDelete.
95     getCanonicalFile());
96 }

```

Listing 4.6: classe di test FileOperationsBlackBoxTest

4.3 Risultati dei test

Il codice dei test descritto precedentemente viene eseguito correttamente senza nessun errore come si può vedere:

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running blackbox.AuthBlackBoxTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.076 s - in blackbox.AuthBlackBoxTest
[INFO] Running blackbox.CommandFactoryBlackBoxTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.013 s - in blackbox.CommandFactoryBlackBoxTest
[INFO] Running blackbox.FileOperationsBlackBoxTest
[LOG] Admin ha caricato: C:\Users\monet\GitHub\ServerTransfer\server_files\upload_test.txt
[LOG] Admin ha eliminato il file: C:\Users\monet\GitHub\ServerTransfer\server_files\tmp.txt
[DOWNLOAD] L'utente User non valido ha scaricato il file: file.txt
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.04 s - in blackbox.FileOperationsBlackBoxTest
[INFO] Running whitebox.ObserverWhiteBoxTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.064 s - in whitebox.ObserverWhiteBoxTest
[INFO] Running whitebox.ServerWhiteBoxTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.008 s - in whitebox.ServerWhiteBoxTest
[INFO] Running whitebox.UserAuthenticatorWhiteBoxTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.02 s - in whitebox.UserAuthenticatorWhiteBoxTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.408 s
[INFO] Finished at: 2025-06-09T21:12:27+02:00
[INFO] -----
```

Process finished with exit code 0

Figura 4.1: Risultati dei Test