Relazione finale

A cura di
Curci Nicola
Dimeo Giovanni
Lagatta Valentina
Lischio Ottavio
Gruppo Nygaard

I.Motivazione del progetto e creazione del gruppo

- SNA4SLACK: progetto e contest
- Descrizione del progetto e tools di supporto
- Gruppo Nygaard **II.**Modello concettuale

III. Requisiti specifici

IV.Architettura

- Stile Architetturale adottato
- Diagramma dei package
- Diagramma dei componenti
- Commento delle decisioni prese

V.System Design

- Users story: diagramma delle classi e di sequenza
- Pattern applicati
- Segnalazioni di PMD
- Commento delle decisioni prese

VI.Riepilogo dei test

- Coveralls

VII.Manuale utente

VIII. Processo di sviluppo e organizzazione del lavoro

IX. Analisi retrospettive

- Cosa ha funzionato bene
- Cosa non ha funzionato
- Cosa rifaremmo

Introduzione SNA4SLACK: progetto e contest

Il progetto SNA4SLACK nasce come contest indetto dal professore Filippo Lanubile, docente di Informatica (INF/01). Il caso di studio in questione è stato scelto come prova di esonero del corso di studio dell'anno accademico in corso 2018, con termine ultimo il 4.06.2018.

Il progetto vede gli studenti suddivisi in gruppo, autonomamente creati secondo le preferenze degli studenti stessi: i gruppi ammessi al progetto sono composti da un minimo di 3 persone, fino ad un massimo di 5. Le direttive per il corretto svolgersi delle mansioni sono state comunicate in aula tramite lezioni frontali e in remoto usufruendo di Slack.

Descrizione del progetto e tool di supporto

Slack è un tool, sfruttato da diversi gruppi di sviluppo, incentrato sulla possibilità di favorire il dialogo fra i membri dello stesso, condivisione dei file e comunicazione fra diversi tool di sviluppo. Le conversazioni dei team sono solitamente organizzate in channel pubblici o privati, solitamente organizzati secondo un tema di dialogo ben preciso.

I messaggi di Slack di un channel pubblico sono visibili ad ogni utente, i quali riceveranno una notifica in caso di iscrizioni ai canali, presenza di nuovi messaggi e menzioni.

In questo progetto, gli sviluppatori hanno avuto il compito di creare un sistema che permetta di analizzare e visualizzare le interazioni sociali che si creano attraversi l'utilizzo di questo tool.

Il progetto prende il nome di Sna4Slack e punta a ricavare, attraverso l'utilizzo di file Json, informazioni basilari come i membri all'interno dei canali o anche, attraverso l'uso di un grafo i cui vertici sono gli utenti e gli archi entranti e uscenti i mention effettuati, il numero effettivo di volte di quando un utente ne menziona un altro.

Il linguaggio di programmazione utilizzato è stato Java, sull'IDE Eclipse, con codice supportato dalla modellazione di diagrammi dei package, dei componenti e delle classi in UML attraverso draw.io; controllo di versione e l'utilizzo di repository sono

stati affidati a Git e Github Classroom, in modo tale da favorire il lavoro a distanza fra gli sviluppatori e l'aggiornamento costante del progetto.
Altri tool utilizzati sono stati: Junit e Jacoco per la creazione di test che supportassero l'efficienza e la funzionalità del progetto; Checkstyle, PMD e Findbugs come aiuto per la supervisione del codice, affinché si potesse mantenere un buon codice sulla base di un corretto clean code; Travis CI, invece, per la continua integrazione e distribuzione del prodotto; e Docker, che è una tecnologia di containerizzazione che consente la creazione e l'utilizzo dei container, consentendo il deployment a partire da un'immagine. Ciò semplifica la condivisione di un'applicazione o di un insieme di servizi, con tutte le loro dipendenze, nei vari ambienti.

Gruppo Nygaard

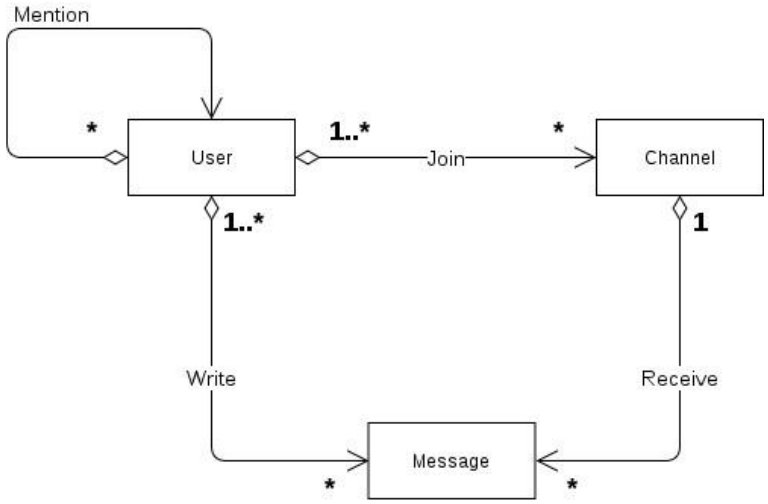
La relazione in questione è basata e redatta secondo il metodo di sviluppo applicato dai membri del gruppo Nygaard: Curci Nicola, Dimeo Giovanni, Lagatta Valentia e Lischio Ottavio.
Il gruppo prende il nome da Nygaard Kristen, Informatico norvegese (Oslo 1926 - ivi 2002). Laureatosi in matematica all'università di Oslo nel 1956, alla metà degli anni Sessanta collaborò con Ole-Johan Dahl alla realizzazione di SIMULA che, inizialmente ideato come simulatore di eventi discreti, venne in seguito aggiornato, divenendo un vero e proprio linguaggio di programmazione. Nygaard e Dahl sono stati i primi a introdurre concetti del paradigma orientato ad oggetti e per tale rilevante contributo nel 2001 sono stati insigniti ex aequo del premio Turing assegnato dall'ACM (Association for computing machinery).

Modello concettuale

Il modello concettuale ha come scopo quello di comprendere i concetti fondamentali del dominio dell'applicativo in questione; precisamente, in questo caso, viene rappresentato il dominio di Sna4Slack con le entità principali che prendono parte alle funzioni principali e le loro relazioni.

Le entità principali che si è andati a modellare sono: User, Channel, Message e Mention.

La scelta di queste entità è giustificata dal fatto che l'applicativo ha bisogno di questo dominio per poter operare e soddisfare i requisiti funzionali.



Requisiti specifici

Per lo sviluppo di questo prodotto sono state ricevute differenti istruzioni affinché si potesse venire in contro alle esigenze del committente.

Agli sviluppatori è stato affidato il compito di modellare differenti caratteristiche, che permettessero ad un utente generico di:

1. In qualità di utente voglio visualizzare la lista dei Member
   - verificare che sia possibile fare la richiesta da standard input
   - Verificare che l'output sia visualizzato su standard output
   - Verificare che sia possibile specificare il workspace
   - Verificare che ci sia un file esportato associato al workspace

- Verificare che i Member siano visualizzati uno per riga
- Verificare che i Member del workspace siano tutti presenti
- Verificare che non siano visualizzati Member estranei al workspace

2. In qualità di utente voglio visualizzare la lista dei Channel:
- Verificare che sia possibile fare la richiesta da standard input
- Verificare che l'output sia visualizzato su standard output
- Verificare che sia possibile specificare il workspace
- Verificare che ci sia un file esportato associato al workspace
- Verificare che i Channel siano visualizzati uno per riga
- Verificare che i Channel del workspace siano tutti presenti
- Verificare che non siano visualizzati Channel estranei al workspace

3. In qualità di utente voglio visualizzare la lista dei membri raggruppati per Channel:
- Verificare che sia possibile fare la richiesta da standard input
- Verificare che l'output sia visualizzato su standard output
- Verificare che sia possibile specificare il workspace
- Verificare che ci sia un file esportato associato al workspace
- Verificare che i Member e Channel siano visualizzati uno per riga, con I Member visualizzati subito ○ dopo il
  Channel a cui appartengono
- Verificare che sia possibile distinguere quale nome è un "Member" e quale è un Channel
- Verificare che i Channel siano tutti presenti
- Verificare che non siano visualizzati Channel estranei al workspace
- Verificare che i Member di un Channel siano tutti presenti
- Verificare che non siano visualizzati Member estranei al Channel

4. In qualità di utente voglio visualizzare la lista dei Member di un Channel:
- Verificare che sia possibile fare la richiesta da standard input
- Verificare che l'output sia visualizzato su standard output
- Verificare che sia possibile specificare il workspace
- Verificare che ci sia un file esportato associato al workspace
- Verificare che sia possibile specificare il Channel
- Verificare che i Member siano visualizzati uno per riga dopo il Channel specificato

- Verificare che i Member del Channel specificato siano tutti presenti
- Verificare che non siano visualizzati Member estranei al Channel specificato

5. In qualità di utente voglio poter avere informazioni di Help:
- Verificare che l'help possa essere richiesto digitando il nome del programma senza parametri
- aggiuntivi
- Verificare che l'help sia suggerito se un comando digitato non è valido
- Verificare l'help sia mostrato su standard output
- Verificare che siano presenti i comandi per tutte le funzionalità

6. In qualità di utente voglio visualizzare la lista dei @mention:
- Verificare che per ogni @mention sia visualizzata una riga con la coppia (From, To) dove From è lo
- User che scrive il messaggio con il @mention e To è lo User menzionato.
- Verificare che le coppie (From, To) non siano ripetute
- Verificare che le coppie (From, To) corrispondenti a un @mention siano tutte presenti
- Verificare che siano visualizzate solo coppie (From, To) corrispondenti a un @mention
- Verificare che sia possibile specificare il Channel e, nel caso sia specificato, la lista sia ristretta ai soli
- @mention del Channel

7. In qualità di utente voglio visualizzare la lista dei @mention che partono da un User:
- Verificare che sia possibile specificare lo User da cui partono i @mention
- Verificare che per ogni @mention sia visualizzata una riga con la coppia (From, To) dove From è lo ○ User specificato nel comando e To è lo User menzionato.
- Verificare che le coppie (From, To) non siano ripetute
- Verificare che le coppie (From, To) corrispondenti a un @mention siano tutte presenti
- Verificare che siano visualizzate solo coppie (From, To) corrispondenti a un @mention
- Verificare che sia possibile specificare il Channel e, nel caso sia specificato, la lista sia ristretta ai soli
- @mention del Channel

8.In qualità di utente voglio visualizzare la lista di @mention che arrivano ad un User:

- Verificare che sia possibile specificare lo User a cui arrivano i @mention
- Verificare che per ogni @mention sia visualizzata una riga con la coppia (From, To) dove è lo User
- che scrive il messaggio con il @mention e To è lo User menzionato e specificato nel comando.
- Verificare che le coppie (From, To) non siano ripetute
- Verificare che le coppie (From, To) corrispondenti a un @mention siano tutte presenti
- Verificare che siano visualizzate solo coppie (From, To) corrispondenti a un @mention
- Verificare che sia possibile specificare il Channel e, nel caso sia specificato, la lista sia ristretta ai soli
- @mention del Channel

9.In qualità di utente voglio visualizzare la lista pesata dei @mention:

- Verificare che per ogni @mention sia visualizzata una riga con la tripla**(From, To, Weight)** dove
- From è lo User che scrive il messaggio con il @mention, To è lo User menzionato, e Weight. è il
- peso associato che riporta il numero di mention da From a To:.
- Verificare che le triple (From, To, Weight) non siano ripetute
- Verificare che le triple (From, To, Weight) corrispondenti a un @mention siano tutte presenti
- Verificare che siano visualizzate solo triple (From, To, Weight)* corrispondenti a un @mention
- Verificare che sia possibile specificare il Channel e, nel caso sia specificato, la lista sia ristretta ai soli
- @mention del Channel_

10. In qualità di utente voglio visualizzare la lista pesata dei @mention che partono da un User:

- Verificare che sia possibile specificare lo User da cui partono i @mention
- Verificare che per ogni @mention sia visualizzata una riga con la tripla (From, To, Weight) dove

- From è lo User specificato nel comando e To è lo User menzionato, e Weight il numero di mention.
- Verificare che le triple (From, To, Weight) non siano ripetute
- Verificare che le triple (From, To, Weight) corrispondenti a un @mention siano tutte presenti
- Verificare che siano visualizzate solo triple (From, To, Weight) corrispondenti a un @mention
- Verificare che sia possibile specificare il Channel e, nel caso sia specificato, la lista sia ristretta ai soli
- @mention del Channel

11. In qualità di utente voglio visualizzare la lista pesata dei @mention che arrivano ad un User:

- Verificare che sia possibile specificare lo User a cui arrivano i @mention
- Verificare che per ogni @mention sia visualizzata una riga con a tripla (From, To, Weight) dove
- From è lo User specificato nel comando e To è lo User menzionato, e Weight il numero di mention.
- Verificare che le triple (From, To, Weight) non siano ripetute
- Verificare che le triple (From, To, Weight) corrispondenti a un @mention siano tutte presenti
- Verificare che siano visualizzate solo triple (From, To, Weight) corrispondenti a un @mention
- Verificare che sia possibile specificare il Channel e, nel caso sia specificato, la lista sia ristretta ai soli
- @mention del Channel

Architettura Stile Architetturale adottato

L'applicativo implementa la variante del pattern architetturale MVC (**MVP**): il pattern divide in tre macro-componenti l'intera struttura dell'applicativo, questo garantisce un notevole vantaggio; ovvero ogni componente è indipendente dagli altri offrendo all'intero sistema indipendenza, scalabilità e manutenzione.
I tre componenti sono:

- **Model**: Il modello gestisce direttamente i dati, la logica e le regole dell'applicazione, nel nostro caso gestisce al suo interno il parsing json,

gli users, i messages, i channels e infine il grafo che rappresenta la rete sociale. Nel caso dell'MVP il Model comunica solo con il Presenter, senza preoccuparsi di come rappresentare i dati al View.

- **View**: si occupa solamente di presentare i dati inviati dal Model all'utente. Nel view viene specificata la forma di presentazione più opportuna per l'utente (UI/CLI/Web). Nel nostro caso il View rappresenta all'utente liste di archi, di utenti e canali nella System Console.
- **Presenter**: sarebbe il Controller del classico MVC, chiamato Presenter perché fa da tramite tra Model e View, il Model in questa variante non comunica con il View, e il view non conosce proprio la struttura del Model. Il Presenter è il nucleo principale del pattern architetturale, il suo compito è quello di gestire la logica dell'applicativo, interpretare le richieste dell'utente, chiamare eventualmente il Model per un supporto con i dati, e comunicare a richiesta esaudita al View per la rappresentazione dell'output.

Diagramma dei Package

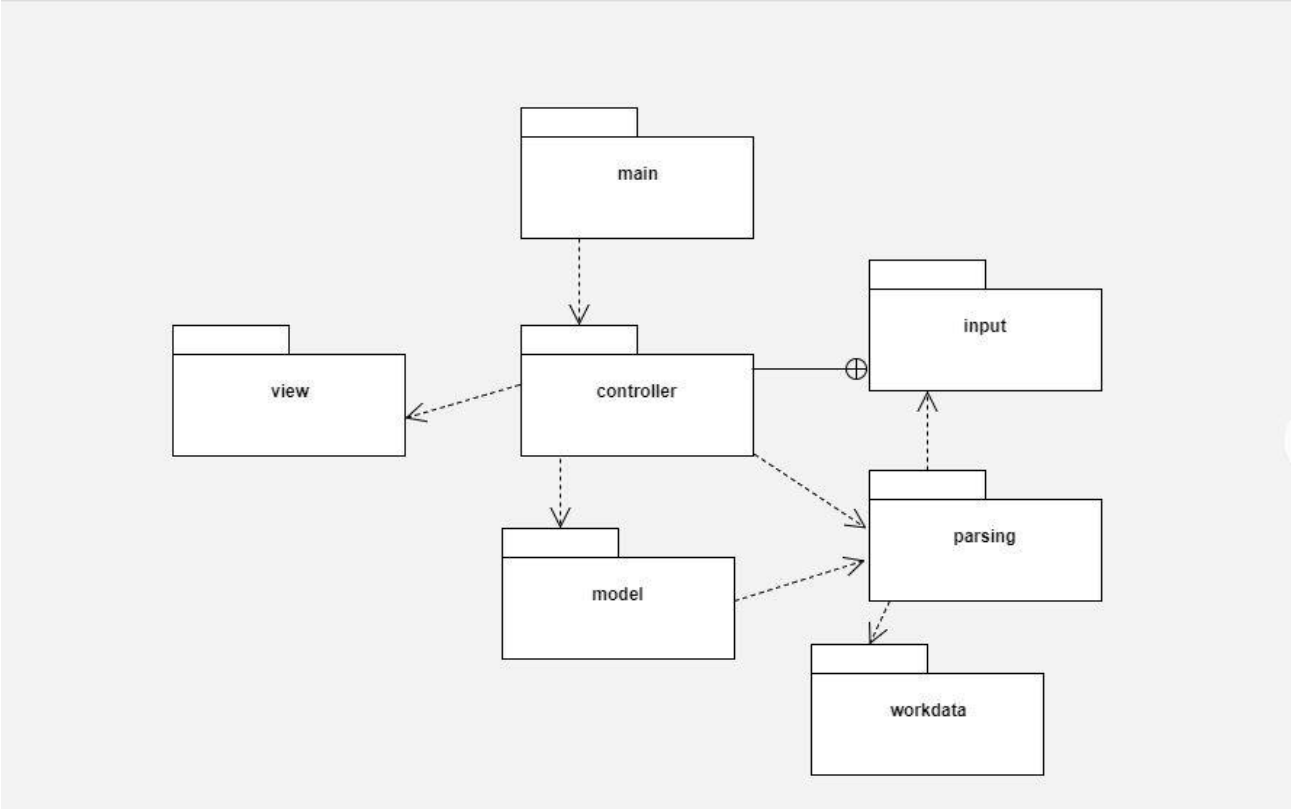
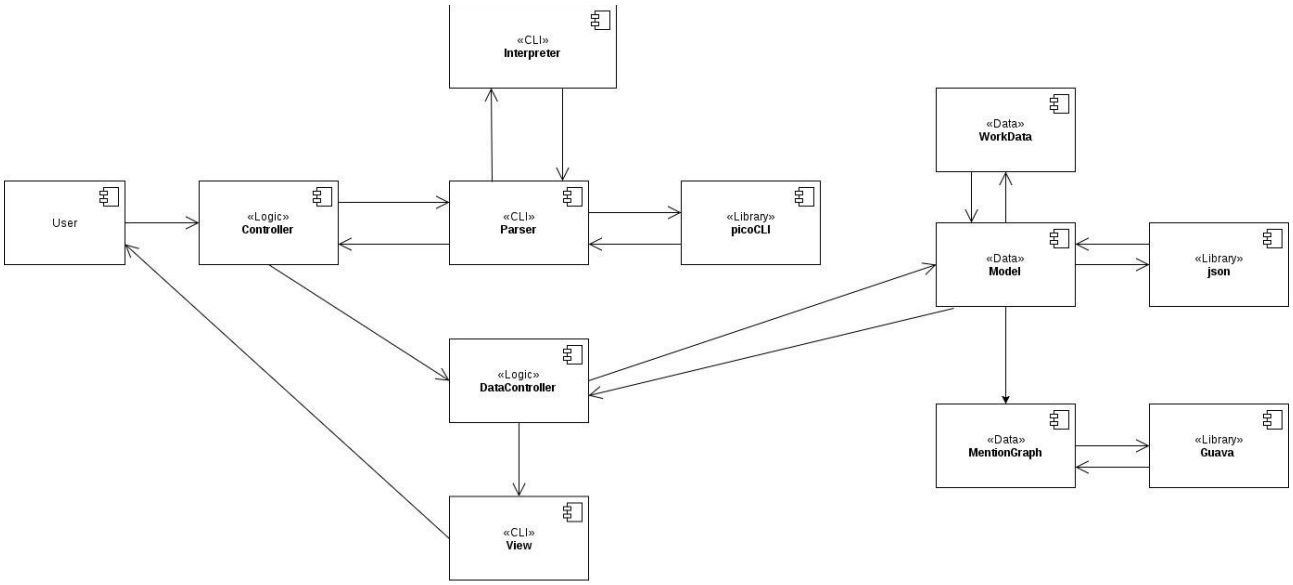


Diagramma dei componenti

Mostra i componenti di un sistema e le loro connessioni



Commento delle decisioni prese

Le scelte architetturali sono state prese in base al nostro background e alle lezioni di Ingegneria del Software che hanno di certo approfondito le nostre conoscenze. La maggior parte dei membri del team avevano già avuto esperienze con l'OO e in particolare con il pattern applicato, l'MVC o (MVP). Il singolo background di ogni membro è stato il punto di forza del team, che ha saputo valutare i costi di tale migrazione e soprattutto tutti i vantaggi che poteva portare un design architetturale cosi invasivo perché applicato nello sprint 2. E' stata una migrazione a questo design architetturale, perché nello sprint 1 non abbiamo applicato nessuna architettura. dallo sprint 2 con l'introduzione del grafo per la gestione della rete sociale il team ha pensato di fare un refactoring abbastanza invasivo per poter applicare il pattern MVP. Questo piccolo costo però ci ha regalato numerosi vantaggi, uno tra i tanti è la Manutenzione e la scalabilità dell'intero sistema, infatti possiamo aggiungere nuove funzioni senza preoccuparci troppo delle dipendenze tra i vari componenti, come è avvenuto per l'intero sprint 3, dove modificando solo il View abbiamo esaudito le user stories.

Precisiamo che è stato costruito da zero, senza nessun framework che potesse darci una linea guida per una migliore implementazione.
Per quanto riguarda le scelte delle librerie esterne, attraverso delle ricerche abbiamo pensato di adottare per puro caso tre librerie:
-Guava, per la modellazione grafi;
-Gson, per il parsing da file JSON;
-PicoCLI, per un miglior supporto del parsing da linea di comando.

System Design User story: diagramma delle classi e di sequenza

Per motivi progettuali i seguenti diagrammi sono stati semplificati, in modo tale da rispecchiare le modalità d'uso del programma.
Per lo sprint 1, infatti, le componenti che cooperano sono le medesime per ogni users story; l'unica cosa che varia è l'input utente, che a seconda del comando inserito farà variare la stampa a video effettuata da View.

Sprint 1: *Diagramma di sequenza*

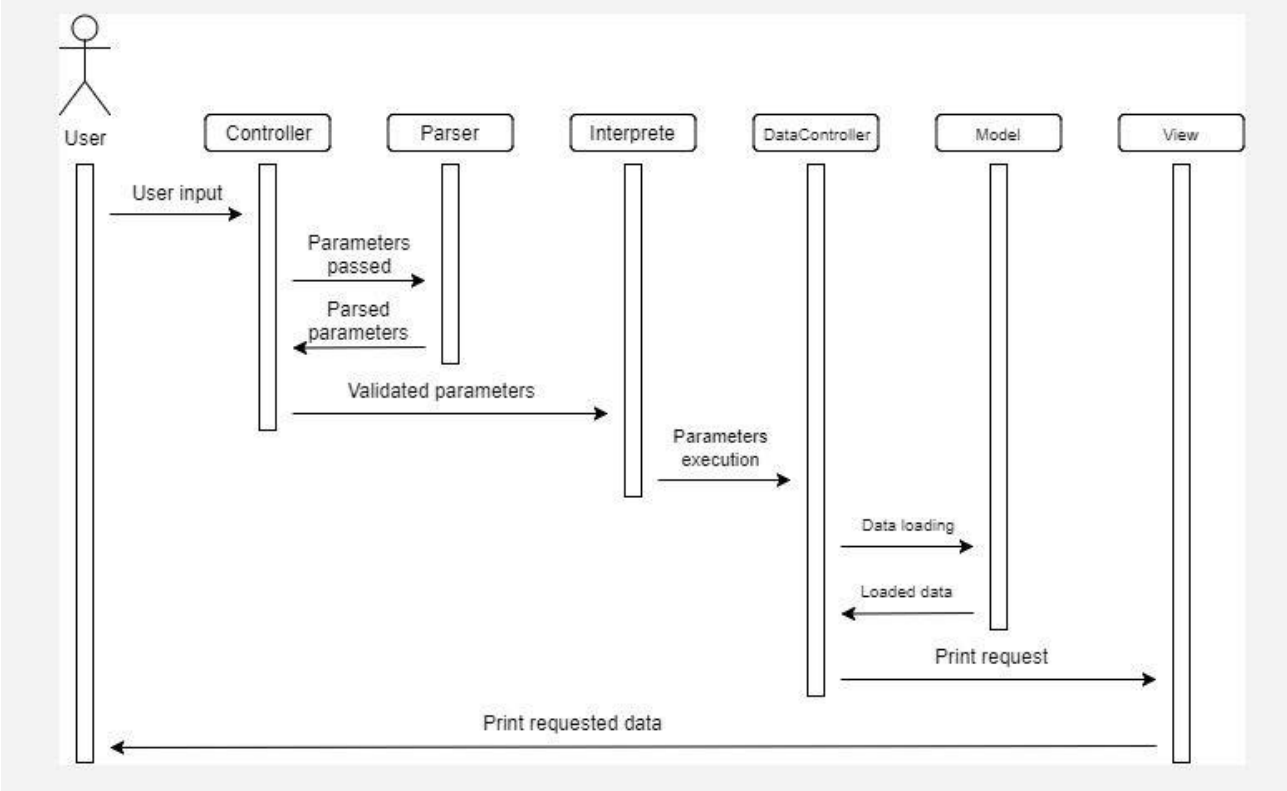
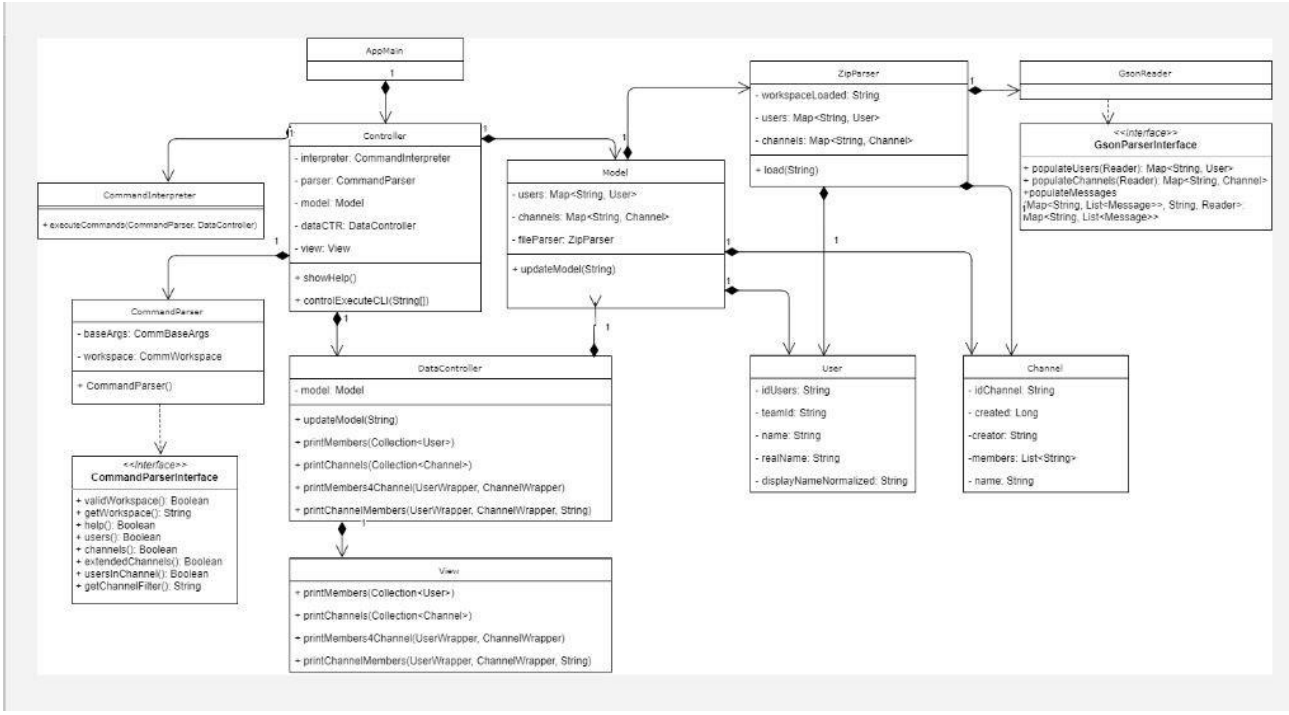


Diagramma delle Classi



Sprint 2: *Diagramma di sequenza*

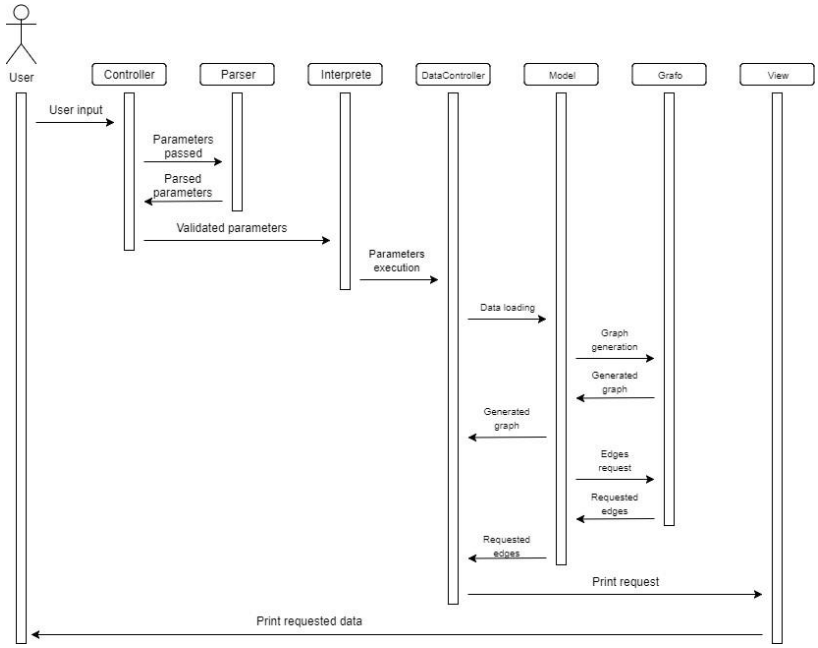
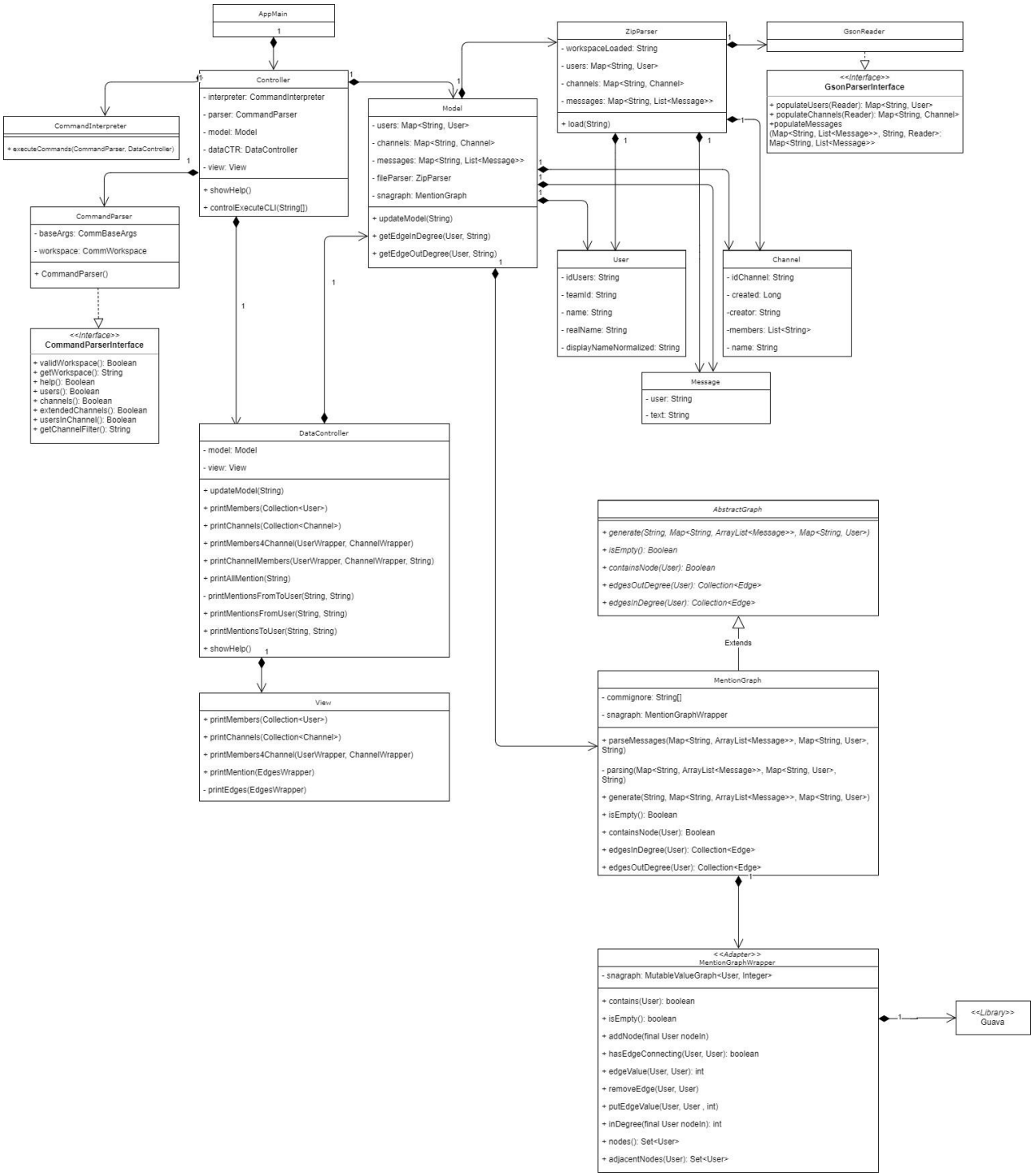


Diagramma delle classi



Per lo spint 3 si è riusciti a risolvere 3 user story, questo perché è stata applicata una variante del pattern
architetturale MVC (MVP), che ha facilitato l'elaborazione di questo sprint: il View attraverso la funzione printEdges, sfruttando un flag booleano, stampa una collezione di archi con o senza peso.

Sprint 3: *Diagramma di sequenza*

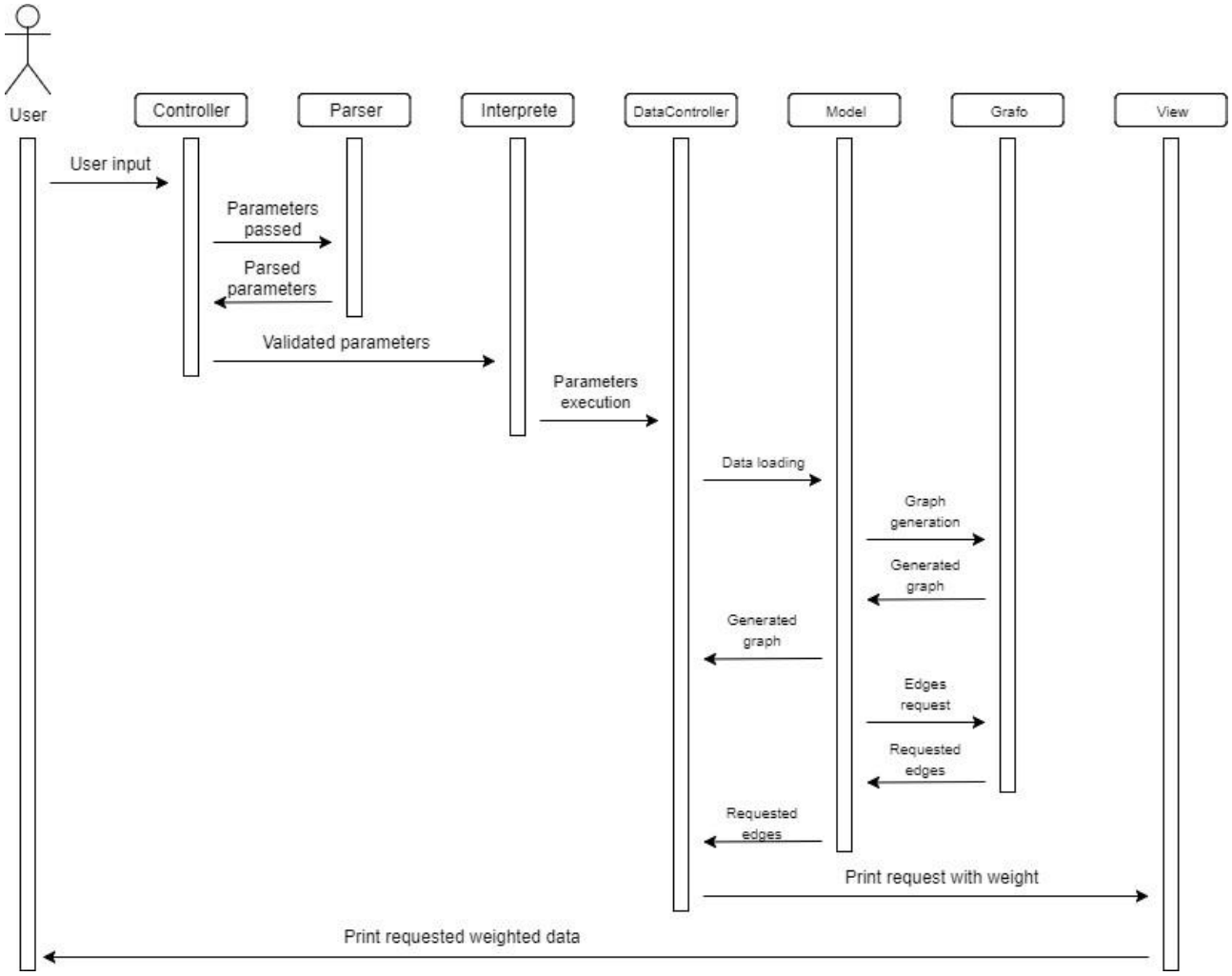
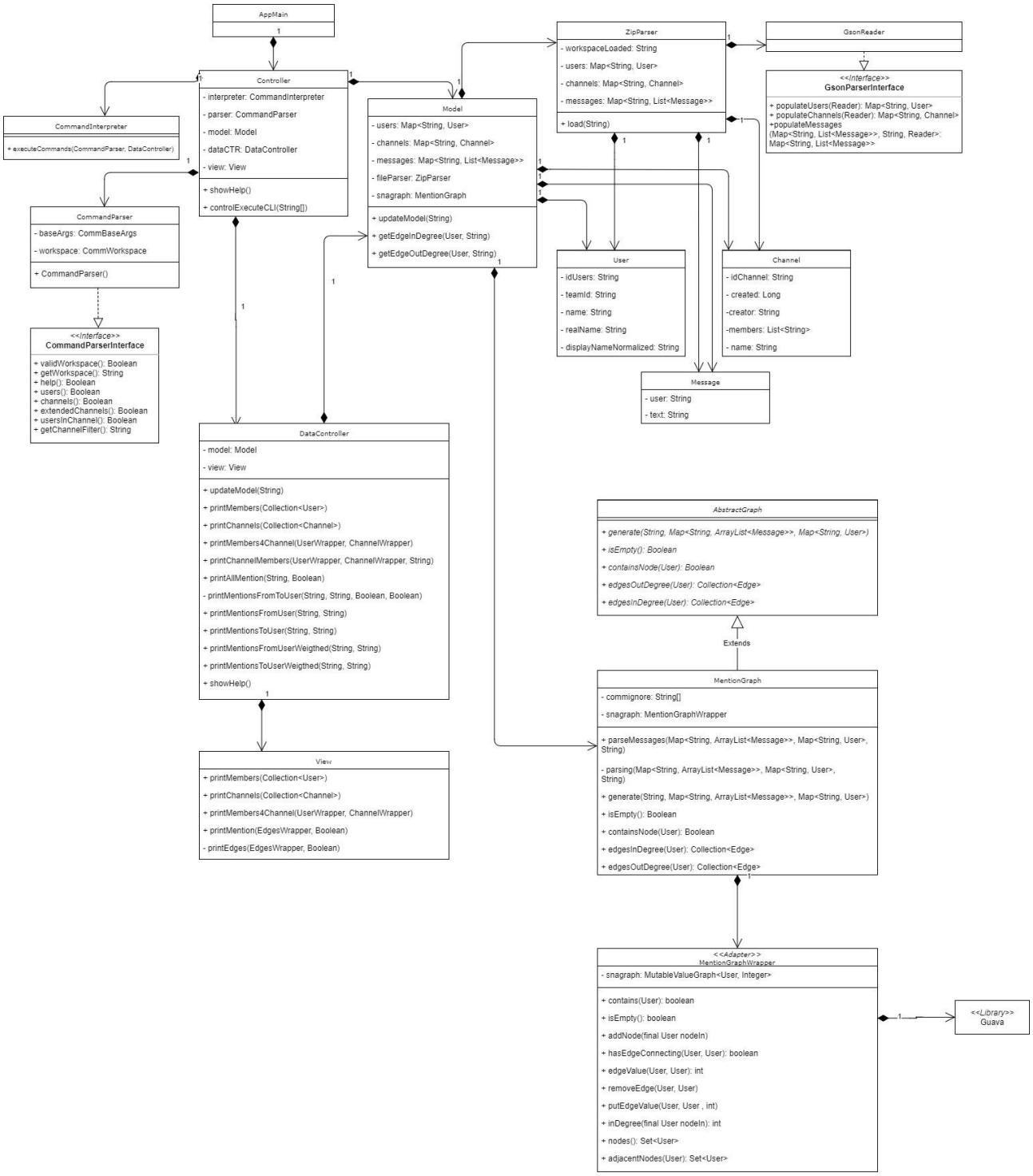


Diagramma delle classi



Pattern applicati Nel lavoro di progettazione si è deciso di applicare il design pattern strutturale Adapter, implementando dei wrapper(o adapter) su alcuni componenti principali del progetto, come: UsersWrapper, ChannelsWrapper., EdgesWrapper e infine MentionGraphWrapper.

I Wrapper sono moduli software che ne rivestono un altro, ovvero che funzionano da tramite fra i propri clienti (che usano l'interfaccia del wrapper) e il modulo rivestito (che svolge effettivamente i servizi richiesti, su delega dell'oggetto wrapper). Per quanto riguarda l' implementazione dei nostri wrapper, si è scelto per motivi di tempo di creare al posto delle interfacce le Classi, e i clienti per composizione possono utilizzare il wrapper indicato per le loro responsabilità.
Abbiamo scelto di implementare i wrapper per rispettare la legge di Demetra e soprattutto per quanto riguarda l'ultimo wrapper "MentionGraphWrapper" per rispettare uno dei 5 principi **S.O.L.I.D.**: **Dependecy Inversion**;
I moduli di alto livello, in questo caso il MentionGraph, non si interfaccia direttamente con la libreria ma con il wrapper (per composizione) e solo il wrapper (MentionGraphWrapper) delega a Guava(un libreria di Google per la gestione dei grafi) le operazioni di basso livello.
Per garantire l'applicazione di questo pattern si sono creati i seguenti componenti:

- Una classe astratta *AbstractGraph* che astrae tutti i comportamenti base per operare con un generico grafo.
- Un wrapper **MentionGraphWrapper** che si interfaccia direttamente con la libreria di Guava.
- una classe **MentionGraph** che implementa tutti i comportamenti astratti della classe AbstractGraph e opera sul grafo grazie al **MentionGraphWrapper** ottenuto per composizione.

Il **Dependecy Inversion** è stato applicato anche per il parsing dei messaggi json, perché questa parte dell'applicativo utilizzava una libreria esterna gson-2.6 una libreria di Google, per il parsing dei file json.

Per non avere classi di alto livello in questo caso ZipParser che potessero dipendere direttamente dalla libreria di terze parti, si sono creati i seguenti componenti:

- Un'interfaccia *JsonParserInterface* che offre il comportamento base di un generico parsing per sna4Slack, ovvero caricare utenti, canali e messaggi dal workspace in formato zip contenente files json.
- Una classe **GsonReader** che implementa tutti i comportamenti dell'interfaccia JsonParserInterface e gestisce direttamente la libreria di terze parti.
- (Gson-2.6.)

- Infine la classe di alto livello, **ZipParser** che attraverso l'uso di **GsonReader** carica i dati json nelle strutture dati apposite

Segnalazioni di PMD

Nel main abbiamo ignorato i seguenti warning:

- "Useless parentheses.": in conflitto con eliminando le parentesi Checkstyle richiedeva di aggiungerle;
- "Avoid short class names like View/User/Egde": i nomi rispecchiano la realtà modellata;
- "Avoid using short method names": i nomi rispecchiano la realtà modellata (fanno riferimento a funzioni to() e from() dell'interfaccia e dell'implementazione;
- "This class has too many methods, consider refactoring it.": da come abbiamo definito le varie classi, i metodi sono necessariamente numerosi;
- "Potential violation of Law of Demeter (object not created locally)" & "Potential violation of Law of Demeter (method chain calls)": abbiamo eliminato numerosi warning e
- i restanti li abbiamo lasciati per motivi di efficienza in quanto richedevano ulteriori istanziazioni di oggetti in fase di caricamento del model o del parsing sia dei comandi che nel file JSON, impossibilità di istanziare determinati oggetti (nel ZipParser);
- "The method/class '<name method/class>' has a Modified/Standard Cyclomatic Complexity of X." : questi warning richedevano delle modifiche strutturali delle classi o metodi
- che per loro natura hanno un grado di complessità ciclomatica elevato (ad esempio la funzione del parsing dei comandi da eseguire causato dai numerosi "if").
- "Private field '<name_field>' could be made final; it is only initialized in the declaration or constructor.": alcuni attributi in cui viene segnalato questo warning necessitano
- che non siano final perchè l'aggiunta porta a malfunzionamenti dell'applicativo.

- "Avoid instantiating new objects inside loops": questa tipologia di warning vengono segnalati nelle zone di codice di caricamento dei dati del grafo delle mentioni o del Model
- per questo inevitabili;
- "Avoid autogenerated methods to access private fields and methods of inner / outer classes": la risoluzione di questo warning generava un altro il quale richiedeva
- una modifica importate nella struttura della definizione delle classi in cui compare.
- Warning riquardanti il 'catching' NPE o RE : non vi è una stretta necessità di specializzare l'eccezione perchè sappiamo che sarà un nullPointer e per questo abbiamo preferito
- rendere l'applicativo più stabile gestendo un eventuale NPE.
- "Avoid if (x != y) ..; else ..;": per motivi di leggibilità del codice abbiamo deciso non risolverlo;
- "Possible God class": PMD segnala questo warning in base al numero di righe ma CommadParser (la classe in cui è segnalato) implementa l'interfaccia CommandParserInterface
- che da come è definita ha numerosi metodi, inoltre CommadParser ha delle inner class quindi rispettando l'OO.

Commento delle decisioni prese

Grazie alle conoscenze acquisite durante il corso, nei due ultimi sprint il team ha cercato di implementare al meglio i pattern per garantire robustezza nell'applicativo. Dopo la lezione sui principi **S.O.L.I.D**. abbiamo pensato subito di applicare l'ultimo principio: *Dependency Inversion*, questo perché nel progetto si è fatto uso di librerie di terze parti. Il *Dependecy Inversion* si occupa di eliminare le dipendenze tra un modulo di alto livello con uno di basso livello (in questo caso le librerie utilizzate). Dal momento che il principio è stato applicato, le classi di alto livello non operano più con le librerie ma con degli *Adapter* che si occupano di interrogare la libreria e restituire i risultati alle classi di alto livello. In questo modo abbiamo garantito anche il primo principio **S.O.L.I.D.**, il *Single Responsability* dove ogni classe ha una sola e singola responsabilità, e il quarto principio, quello dell'interface Interface *Segregation*.Le interfacce che operano sui dati di basso

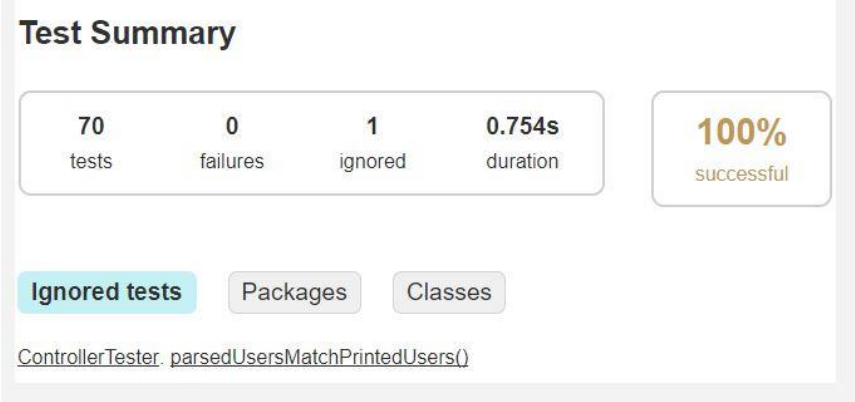
livello non sono eccessivamente generiche e permettono, inoltre, una facile modifica ed estensione della codebase.

Riepilogo dei test

Coveralls

SNA4Slack

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
it.uniba.main		18%		50%	1	2	21	26	0	1	0	1
it.uniba.view		85%		86%	7	24	10	80	4	13	0	2
it.uniba.workdata		81%		20%	7	38	10	69	3	33	0	5
it.uniba.controller		91%		62%	15	39	9	84	0	19	0	3
it.uniba.controller.input		100%		91%	8	81	0	138	0	34	0	4
it.uniba.model		100%		87%	9	63	0	114	0	28	0	4
it.uniba.parsing		100%		100%	0	29	0	71	0	18	0	3
it.uniba.wrapping		100%		n/a	0	29	0	59	0	29	0	4
Total	174 of 2.813	93%	44 of 260	83%	47	305	50	641	7	175	0	26



Manuale utente

L'utente avrà a disposizione differenti comandi, ognuno dei quali metterà a disposizione dell'utente una funzione differente, anche attraverso l'unione di più comandi:

a. -w analizza uno specificato workspace;
b. -c mostra tutti i canali appartenenti al workspace specificato;
c. -u mostra tutti gli utenti del workspace specificato;
d. -uc <channelFilter> mostra tutti gli utenti di uno specifico canale;
e. -cu mostra tutti i canali coi relativi utenti;

f. -m mostra tutti i @mention del workspace;
g. -m from <user> filtra i @mention che appartengono ad un utente specificato;
h. -m to <user> filtra i @mention che citano un utente specificato;
i. -m in channel filtra i @mention all'interno di un channel specificato;
j. -m from <user> O to <user> in <channel> filtra i @mention da o per un utente di uno specifico canale.

Processo di sviluppo e organizzazione del lavoro

La partizione del lavoro durante i tre sprint è stata sempre preceduta da un confronto post lezione, nel quale i vari membri del gruppo hanno avuto modo di discutere riguardo i propri punti di forza e di debolezza riguardo ciascuna features da implementare nei rispettivi sprint.

Successivamente, si passava a progettare su carta una bozza delle classi che sarebbero state sviluppate nei successivi 9 giorni di sprint; inoltre veniva redatta una tabella su github con le mansioni principali che avrebbero dovuto essere svolte ed evolute con l'avanzare del progetto.

Alla fine di questa breve sessione preparatoria chi si riteneva motivato e capace di assumersi la responsabilità di un compito, sempre considerando le proprie capacità e il tempo a disposizione, si incaricava di portare a termine un preciso lavoro discusso con gli altri membri.

La comunicazione è avvenuta ogni giorno, di persona e anche sfruttando Slack, in modo tale da evolvere le varie mansioni in base alle esigenze e a varie ed eventuali che potessero nascere durante lo sprint corrente.

Analisi retrospettive Cosa ha funzionato bene

Il gruppo ha avuto un'ottima comunicazione; non ci sono stati problemi durante l'assegnazione dei vari compiti, in quanto ogni membro del gruppo è stato capace di farsi carico di issue in cui era più competente, senza interferire col lavoro altrui, favorendo quindi un'ottima organizzazione

Cosa non ha funzionato

Una cosa che non ha funzionato è stata la scelta del design architetturale, poiché, sia per necessità che per tempi ridotti, è stata fatta durante lo sprint 2.

Cosa rifaremmo

Una cosa che rifaremmo è la scelta con la quale sono stati assegnati i task; le assegnazioni sono state fatte prendendo come criterio principale il background che ognuno di noi aveva; tutto è stato forgiato in base alle nostre competenze.