

Relazione progetto Brew Day!

Cazzetta Davide, Cusini Matteo, Locatelli Federico

Gennaio 2020

Indice

1	Introduzione	2
2	Analisi	2
2.1	Diagramma di Gantt iniziale	3
2.2	Diagramma dei casi d'uso	4
2.3	Casi d'uso	5
2.3.1	Create Recipe	5
2.3.2	Modify Recipe	5
2.3.3	Delete Recipe	6
2.3.4	Create recipe instance	6
2.3.5	"What should I brew today?" feature	7
2.3.6	Insert equipment	7
2.3.7	Modify instruments	8
2.3.8	Update available ingredients	8
2.3.9	Notify about missing ingredients	9
2.3.10	Create note	9
3	Progettazione	10
3.1	Diagramma delle classi a livello di dominio	10
3.2	Diagramma delle classi a livello di progettazione	11
3.3	Diagramma dell'architettura software	14
3.4	Diagramma di sequenza	16
3.5	Diagramma di stato	17
3.6	Diagrammi di attività	17
3.6.1	Ciclo di vita della ricetta	18
3.6.2	Feature "What should I brew today?"	19
3.7	Pattern utilizzati e design principle	20
3.7.1	Design pattern	20
3.7.2	Pattern architetturali	20
3.7.3	Design principle	20
4	Implementazione	21
4.1	Stesura codice	21
4.2	Git	22
4.3	Sonar	22
4.4	Understand	23
5	Considerazioni finali	25
5.1	Diagramma di Gantt finale	25
5.2	Commenti	25
6	Note	26

1 Introduzione

Questa relazione definisce il percorso di analisi, progettazione e sviluppo del progetto "Brew Day!", svolto dagli studenti Cazzetta Davide, Cusini Matteo e Locatelli Federico.

2 Analisi

Il primo passo è stato l'analisi, fase determinante per chiarire quali requisiti avrebbe dovuto soddisfare il software che avremmo dovuto sviluppare. Il processo di analisi è iniziato definendo il diagramma di Gantt tramite la stesura di un insieme di attività che avremmo dovuto eseguire in queste settimane; abbiamo quindi compilato il diagramma con una serie di attività, ed assegnato le nostre risorse ad ognuna di esse. Le attività si sarebbero susseguite durante tutto l'arco di tempo a nostra disposizione per arrivare alla fine al compimento del progetto.

Completato il diagramma di Gantt abbiamo iniziato la fase di analisi dei requisiti leggendo più volte il testo del progetto e sottolineando le parti che ritenevamo di maggior rilevanza nei termini dei requisiti funzionali. Una volta individuate le parti che reputavamo importanti ci siamo confrontati ed abbiamo stilato un insieme di casi d'uso, di attori e di scenari che a nostro parere definivano il contesto di utilizzo del software.

2.1 Diagramma di Gantt iniziale

La metodologia di stesura del diagramma di Gantt iniziale (Figura 1 e Figura 2) è stata molto complessa in quanto nessuno di noi aveva mai partecipato ad un progetto di sviluppo software di questa portata. Le tempistiche e le varie attività infatti sono state giustamente smentite nel prosieguo del progetto.

Necessario notare che seppure il diagramma pone una visione delle attività a cascata, durante lo svolgimento del progetto è stato seguito un approccio agile.

ID	Modalità attività	Nome attività	Durata	Inizio	Fine	Predecessori	Nomi risorse
1	✚	Use Case Diagram	2 g?	sab 21/12/19	lun 23/12/19		Locatelli;Cusini;Cazzetta
2	✚	Domain Model Diagram	2 g	ven 27/12/19	dom 29/12/19		Locatelli;Cazzetta;Cusini
3	✚	Class Diagram	4 g?	lun 30/12/19	gio 02/01/20	2	Cazzetta;Cusini;Locatelli
4	✚	Sequence Diagram	1 g?	ven 03/01/20	ven 03/01/20	3	Cazzetta
5	✚	State Diagram	1 g?	ven 03/01/20	ven 03/01/20	3	Cusini
6	✚	Activity Diagram	1 g?	ven 03/01/20	ven 03/01/20	3	Locatelli
7	✚	Software Architecture Diagram	1 g?	sab 04/01/20	sab 04/01/20	3	Cazzetta;Cusini;Locatelli
8	✚	Coding	18 g?	dom 05/01/20	mar 28/01/20	7	Cazzetta;Cusini;Locatelli
9	✚	Testing	2 g?	mer 29/01/20	gio 30/01/20	8	Cazzetta;Cusini;Locatelli
10	✚	Release	1 g?	ven 31/01/20	ven 31/01/20	9	Cazzetta;Cusini;Locatelli

Figura 1: Tabella relativa al diagramma di Gantt iniziale

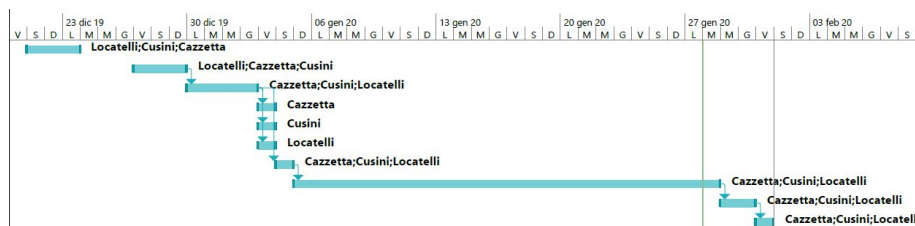


Figura 2: Diagramma di Gantt iniziale

2.2 Diagramma dei casi d'uso

L'analisi dei requisiti ha prodotto un diagramma dei casi d'uso che è stato integrato e ridefinito man mano che il progetto evolveva, sulla base di funzionalità che secondo noi il software doveva fornire. La versione finale (Figura 3) comprende tutti i requisiti funzionali che il software concede all'utente. La descrizione di ogni caso d'uso è riportata di seguito.

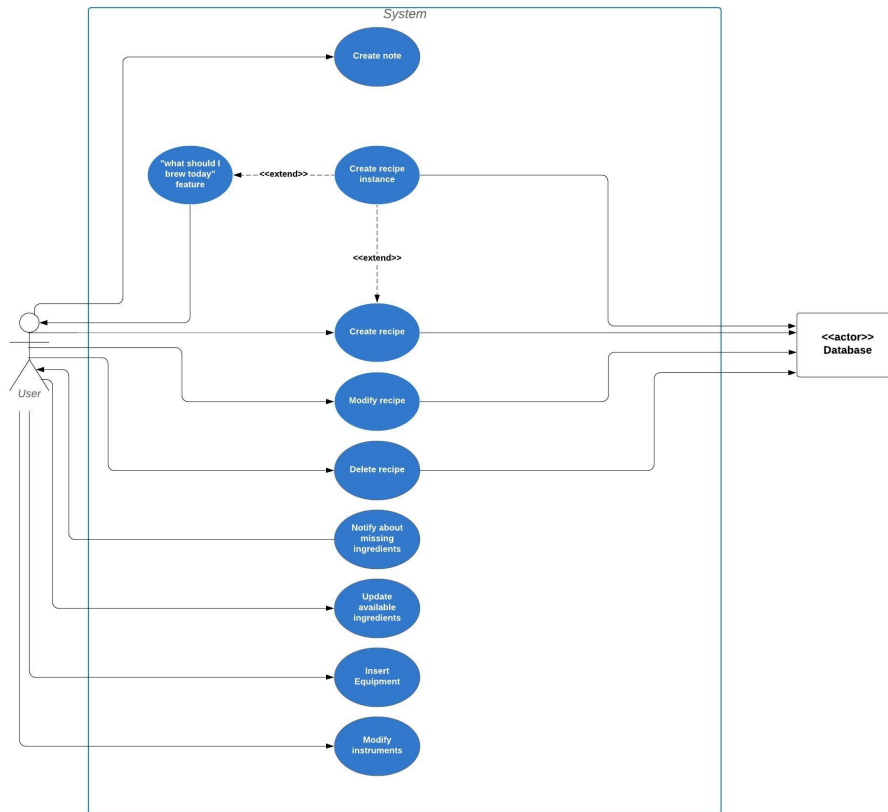


Figura 3: Diagramma dei casi d'uso

2.3 Casi d'uso

I casi d'uso presentati nel diagramma sono descritti uno a uno di seguito

2.3.1 Create Recipe

Nome del caso d'uso	Create recipe
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole creare una nuova ricetta e salvarla nel sistema
Pre-condizioni	Deve necessariamente essere presente un equipment (insieme di strumenti dell'utente)
Garanzia di successo	La ricetta viene salvata, ed è fruibile finchè non viene eliminata
Scenario principale di successo	L'utente preme il bottone "New Recipe", inserisce le quantità di ingredienti che preferisce e preme il bottone "Save"
Estensioni	In qualsiasi momento durante la creazione l'utente può decidere di tornare indietro e di non creare la ricetta
Requisiti speciali	Interfaccia grafica minimale, con pulsante "Save" e "Back"
Frequenza di ripetizione	Saltuaria ma necessaria

2.3.2 Modify Recipe

Nome del caso d'uso	Modify recipe
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole modificare una ricetta e salvarla nel sistema
Pre-condizioni	Deve necessariamente essere presente una ricetta salvata
Garanzia di successo	La ricetta viene modificata, ed è fruibile finchè non viene eliminata
Scenario principale di successo	L'utente preme il bottone "Modify", modifica le quantità di ingredienti della ricetta preme il bottone "Save"
Estensioni	In qualsiasi momento durante la modifica l'utente può decidere di tornare indietro e di non modificare la ricetta
Requisiti speciali	Interfaccia grafica minimale, con pulsante "Save" e "Back"
Frequenza di ripetizione	Potrebbe non accadere mai

2.3.3 Delete Recipe

Nome del caso d'uso	Delete recipe
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole eliminare una ricetta dal sistema
Pre-condizioni	Deve necessariamente essere presente una ricetta salvata
Garanzia di successo	La ricetta viene eliminata, e non sarà più fruibile
Scenario principale di successo	L'utente preme il bottone "Delete"
Estensioni	In qualsiasi momento durante l'eliminazione, l'utente può decidere di tornare indietro e di non eliminare la ricetta
Requisiti speciali	Interfaccia grafica minimale, con pulsante "Delete" a fianco della ricetta
Frequenza di ripetizione	Potrebbe non accadere mai

2.3.4 Create recipe instance

Nome del caso d'uso	Create recipe instance
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole produrre una ricetta e salvare la produzione
Pre-condizioni	Deve necessariamente essere presente una ricetta salvata
Garanzia di successo	La produzione viene inizializzata e gli ingredienti corrispondenti alla ricetta vengono sottratti al magazzino
Scenario principale di successo	L'utente preme il bottone "Brew it!" in corrispondenza della ricetta che vuole produrre
Estensioni	Dopo la creazione, l'utente può eliminare o cancellare una produzione dal sistema
Requisiti speciali	Interfaccia grafica minimale, con bottone "Brew it!" a fianco della ricetta
Frequenza di ripetizione	Ogni volta che l'utente inizia una produzione

2.3.5 "What should I brew today?" feature

Nome del caso d'uso	"What should I brew today?" feature
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole produrre una ricetta massimizzando l'utilizzo degli ingredienti a disposizione nel magazzino
Pre-condizioni	Deve necessariamente essere presente almeno una ricetta salvata, e deve esserci disponibilità di ingredienti
Garanzia di successo	La produzione viene inizializzata e gli ingredienti corrispondenti alla ricetta che massimizza l'utilizzo degli ingredienti vengono sottratti al magazzino
Scenario principale di successo	L'utente preme il bottone "Brew it!" nella schermata principale, dove viene indicata la ricetta che massimizza l'utilizzo degli ingredienti
Estensioni	
Requisiti speciali	Interfaccia grafica minimale, con bottone "Brew it!" sotto il nome della ricetta
Frequenza di ripetizione	Ogni volta che l'utente vuole massimizzare l'utilizzo degli ingredienti per una produzione

2.3.6 Insert equipment

Nome del caso d'uso	Insert equipment
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole inserire i suoi strumenti di lavoro e la loro capacità
Pre-condizioni	Nessuna
Garanzia di successo	L'utente inserisce gli strumenti definendo il proprio equipment, dove, per ogni strumento viene specificata la capacità in litri
Scenario principale di successo	L'utente crea un nuovo equipment inserendo il numero di strumenti che possiede e specificando per ogni strumento la capacità dello stesso
Estensioni	
Requisiti speciali	Interfaccia grafica minimale, con bottone "Create Equipment"
Frequenza di ripetizione	Un'unica volta, al primo utilizzo

2.3.7 Modify instruments

Nome del caso d'uso	Modify instruments
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole modificare i suoi strumenti di lavoro e la loro capacità
Pre-condizioni	Deve essere presente un equipment
Garanzia di successo	L'utente modifica gli strumenti ridefinendo la capacità oppure inserendo un nuovo strumento
Scenario principale di successo	L'utente preme il bottone "Modify Equipment" e modifica la capacità degli strumenti già presenti oppure preme il bottone "Insert new instrument" e inserisce un nuovo strumento
Estensioni	L'utente può decidere anche di rimuovere degli strumenti
Requisiti speciali	Interfaccia grafica minimale, con bottone "Modify Equipment" e bottone "Insert new instrument"
Frequenza di ripetizione	All'occorrenza

2.3.8 Update available ingredients

Nome del caso d'uso	Update available ingredients
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole aggiornare la quantità di ingredienti in magazzino
Pre-condizioni	
Garanzia di successo	L'utente modifica le quantità degli ingredienti in magazzino
Scenario principale di successo	L'utente visualizza il magazzino, preme il bottone "Modify Storage" e modifica le quantità di ogni ingrediente
Estensioni	
Requisiti speciali	Interfaccia grafica minimale, con bottone "Modify Storage"
Frequenza di ripetizione	All'occorrenza

2.3.9 Notify about missing ingredients

Nome del caso d'uso	Notify about missing ingredients
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: viene notificato della mancanza di specifici ingredienti quando fa partire una produzione che richiede quantità di ingredienti maggiori rispetto alla giacenza
Pre-condizioni	Deve essere presente almeno una ricetta
Garanzia di successo	L'utente viene notificato delle quantità degli ingredienti mancanti in magazzino
Scenario principale di successo	L'utente preme il bottone "Brew it!" relativo ad una ricetta e viene notificato che gli ingredienti in magazzino non sono sufficienti per la produzione
Estensioni	
Requisiti speciali	Interfaccia grafica minimale, con finestra di alert che definisce gli ingredienti mancanti
Frequenza di ripetizione	All'occorrenza

2.3.10 Create note

Nome del caso d'uso	Create note
Portata	Brew Day!
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente: vuole inserire delle note relativamente ad una produzione
Pre-condizioni	Deve essere presente almeno una produzione
Garanzia di successo	L'utente crea la nota relativa alla produzione
Scenario principale di successo	L'utente preme il bottone "Add Note" relativo ad una produzione e crea una nota di tipo "tasting" o normale
Estensioni	L'utente può modificare o eliminare la nota
Requisiti speciali	Interfaccia grafica minimale, con bottone "Add Note" vicino alla produzione
Frequenza di ripetizione	All'occorrenza

3 Progettazione

Una volta terminata la fase di analisi, siamo passati alla progettazione. Abbiamo stabilito quali diagrammi fosse necessario redigere sulla base delle funzionalità più complesse e delicate del programma, quindi, oltre ai diagrammi delle classi e di architettura, abbiamo scelto di definire un diagramma di sequenza e un diagramma di stato per la fase di produzione di una ricetta, un diagramma di attività che illustra il ciclo di vita di una ricetta, ed un altro diagramma di attività per la feature "What should I brew today?".

Questi diagrammi sono stati soggetti a continue riformulazioni e rivisitazioni durante tutto l'arco di tempo, concedendoci una visione sempre migliore e raffinata del progetto che portavamo avanti.

3.1 Diagramma delle classi a livello di dominio

Il diagramma delle classi a livello di dominio (Figura 4) è stato modellato sulla base della realtà, cercando di rappresentare concettualmente ciò che poi avremmo trasformato in un diagramma delle classi a livello di progettazione e poi in software applicativo. Il punto di forza di questo diagramma risiede nella semplicità di trasmettere tutto il necessario, per fornire un'idea chiara di ciò che avevamo davanti agli occhi durante la progettazione. Infatti questo diagramma è risultato fondamentale per l'evoluzione del progetto, aiutandoci nello sviluppo degli altri diagrammi e del codice applicativo.

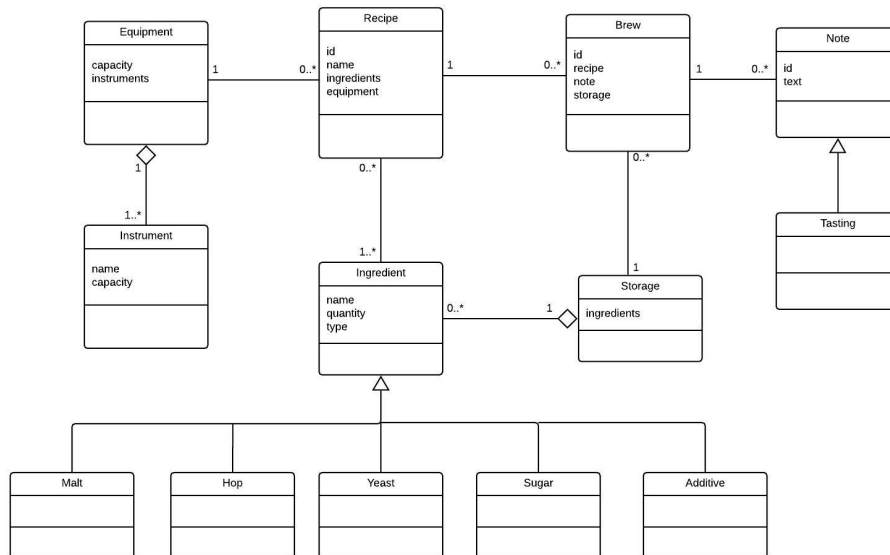


Figura 4: Diagramma delle classi a livello di dominio

3.2 Diagramma delle classi a livello di progettazione

Lo sviluppo del diagramma delle classi a livello di progettazione è stato continuo e ininterrotto. Basandoci sul diagramma delle classi a livello di dominio abbiamo sviluppato un primo diagramma delle classi di progettazione (Figura 5), che non ci ha messo molto a venire stravolto e modificato in base a ciò che reputavamo fosse una scelta implementativa migliore.

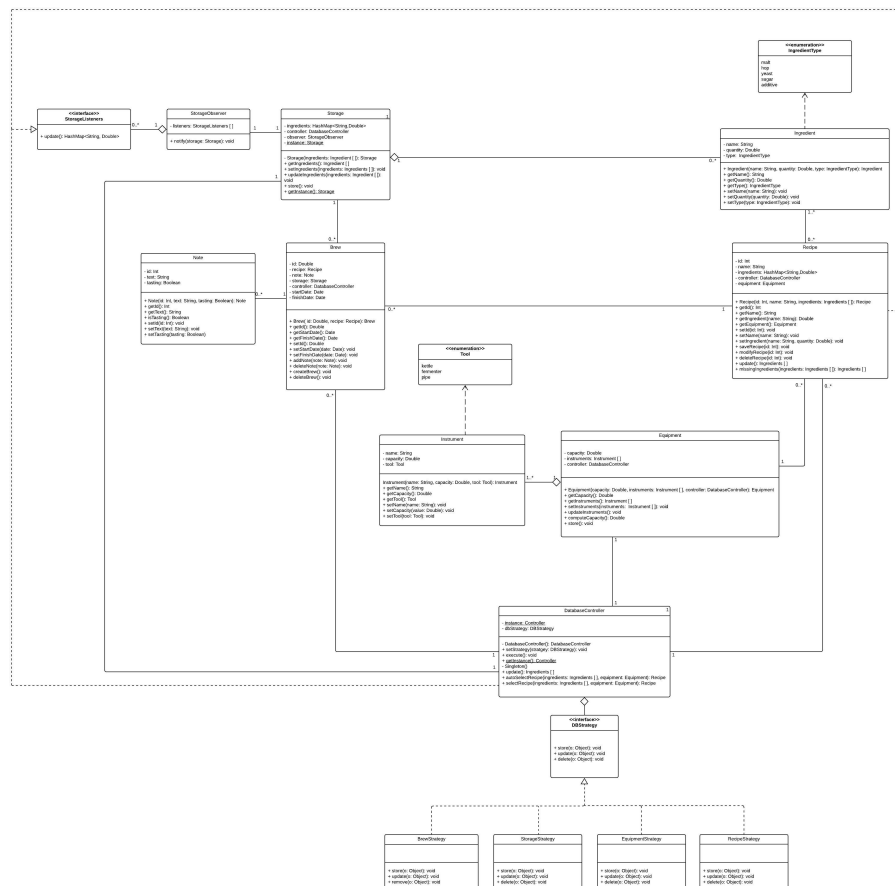


Figura 5: Primo diagramma delle classi a livello di progettazione

Questo primo diagramma delle classi a livello di progettazione prevedeva l'utilizzo di tre design pattern: il pattern Singleton per le classi DatabaseController e Storage che dovevano rimanere singole istanze, il pattern Observer per la classe Storage e il pattern Strategy per l'accesso al database.

Dopo poco tempo il diagramma delle classi a livello di progettazione si è evoluto in un'altra forma, mantenendo l'idea di base precedente e gli stessi pattern, ma raffinando diverse parti (Figura 6).

Una prima implementazione in questa forma è stata necessaria per farci sorgere il dubbio se fosse questa la scelta implementativa migliore oppure se ce ne fosse un'altra che potesse permettere una migliore separazione degli interessi e uno stile più snello e performante. I pattern Observer e Strategy scelti in precedenza



La scelta è poi ricaduta sull'adozione di un pattern ibrido che avesse alla base il Model-View-Controller, e che fondesse diversi principi tra i quali l'organizzazione in procedure della logica di business, dove ogni procedura gestisce una singola richiesta dallo strato di presentazione (Transaction Script) e aspetti derivanti da design principles, come l'Acyclic Dependencies Principle e il Common-Reuse Principle.

12

stessi package. Il design pattern che abbiamo mantenuto fin dal principio è stato Singleton; utilizzato per tutti i Controller e per le classi Equipment e Storage. Il diagramma delle classi a livello di progettazione è risultato quindi essere quello in Figura 7.

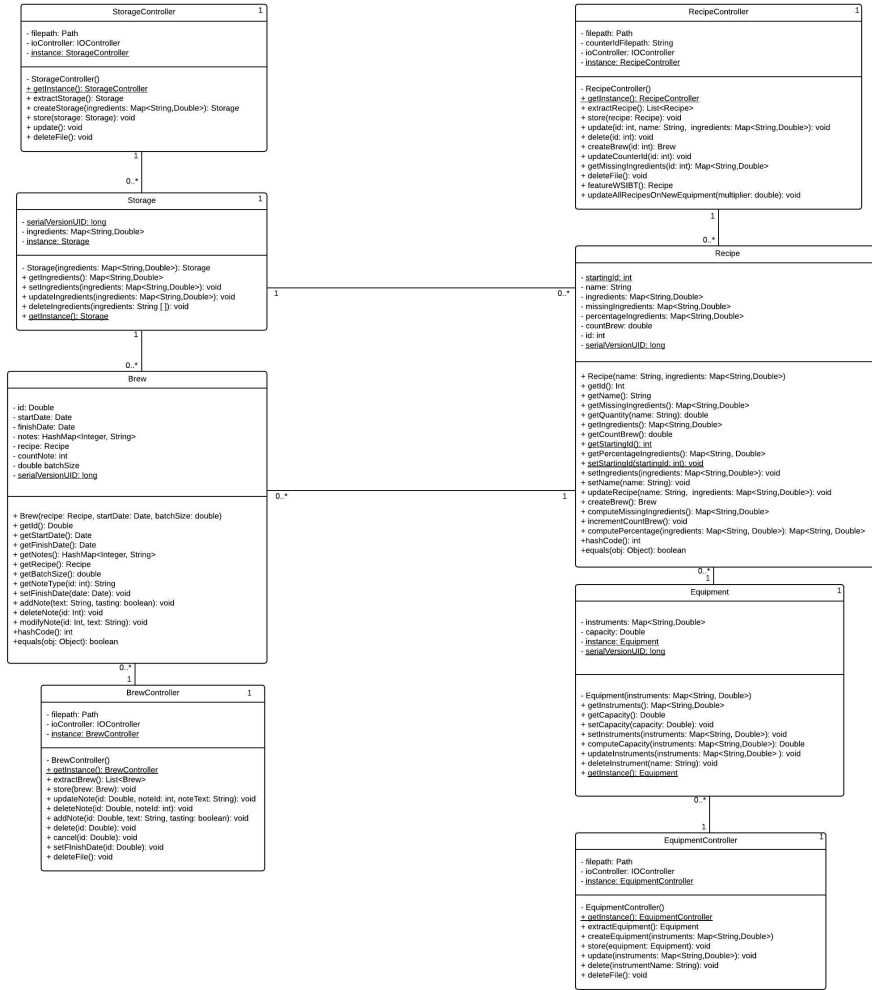


Figura 7: Diagramma delle classi a livello di progettazione definitivo

3.3 Diagramma dell'architettura software

Fin dal principio, l'idea che circolava tra di noi su quale architettura software seguire è stata quella della divisione in multilayer, scelta avvalorata anche dall'utilizzo del pattern architetturale Model-View-Controller. Come prevede il pattern abbiamo quindi suddiviso l'architettura in quattro layer (Figura 8):

1. Presentation layer: livello più alto e vicino all'utente, in cui si trova tutta la parte relativa alla GUI. Questo layer comunica a stretto contatto con l'Application layer, passando ad esso la responsabilità di gestire gli input da parte dell'utente. Definisce la parte di View del pattern MVC.
2. Application layer: livello intermedio, al suo interno si trovano tutti i Controller che gestiscono il traffico di informazioni derivanti dal Presentation layer, smistando e delegando la responsabilità computazionale al livello di Business logic. Definisce la parte di Controller del pattern MVC.
3. Business logic layer: livello intermedio, al suo interno si trova tutta la parte di elaborazione che rende attivo il software, sono presenti tutte le classi che definiscono gli oggetti utilizzati nell'applicazione. Definisce la parte di Model del pattern MVC.
4. Service layer: livello più basso, contiene tutti i file di persistenza del software. Viene chiamato in causa tramite chiamate di input/output.

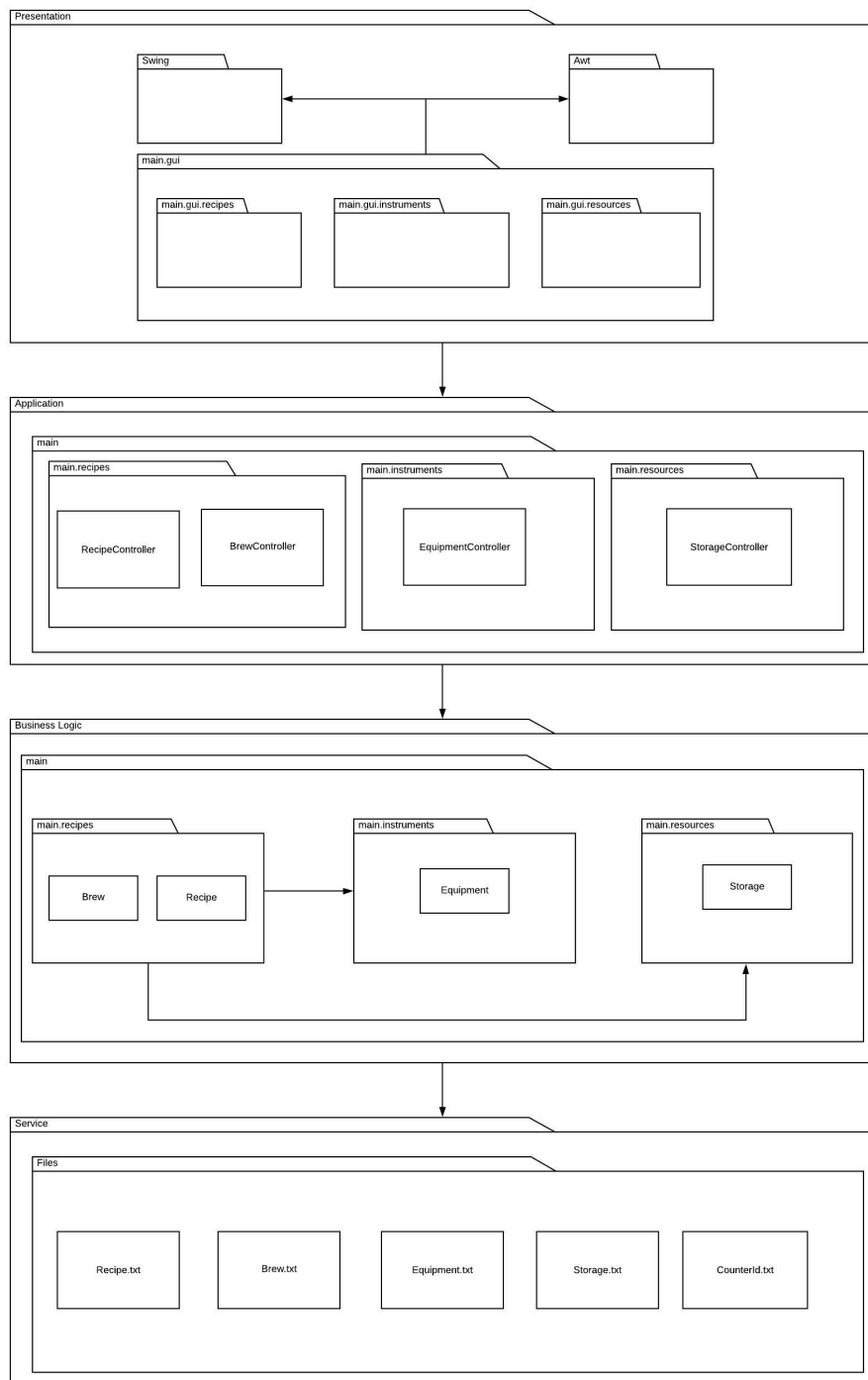


Figura 8: Diagramma dell'architettura software

3.4 Diagramma di sequenza

Come detto precedentemente, abbiamo creato un diagramma di sequenza (Figura 9) per avere una visione migliore sul procedimento da seguire per implementare il processo di creazione di un'istanza di una ricetta (Brew).

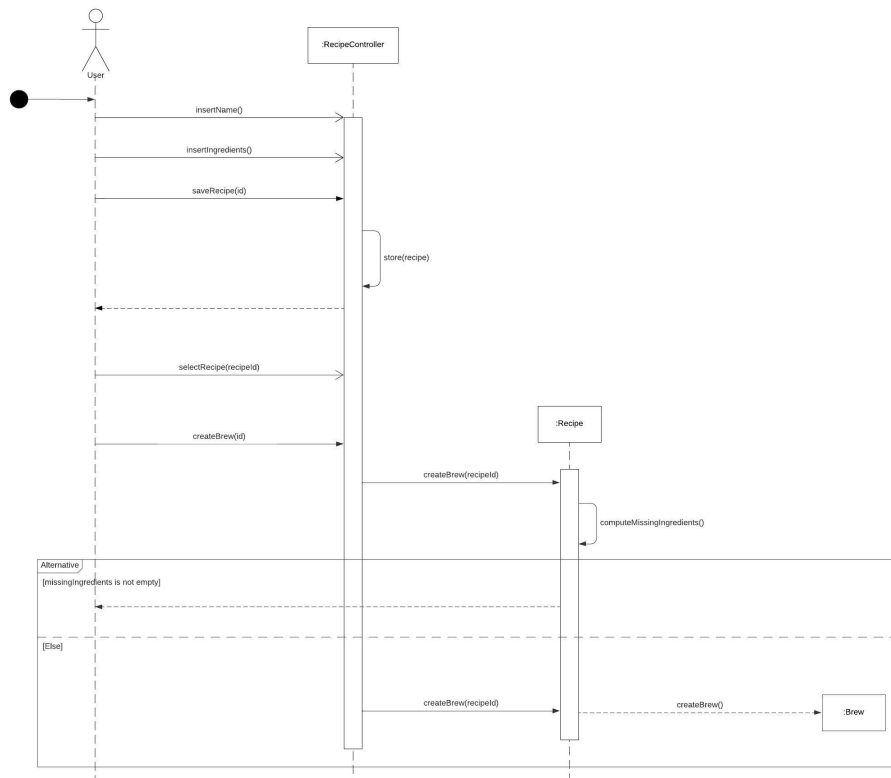


Figura 9: Diagramma di sequenza

3.5 Diagramma di stato

Analogamente al diagramma di sequenza, il diagramma di stato (Figura 10) che abbiamo prodotto cerca di analizzare in maniera dettagliata il processo di produzione di una ricetta (Brew).

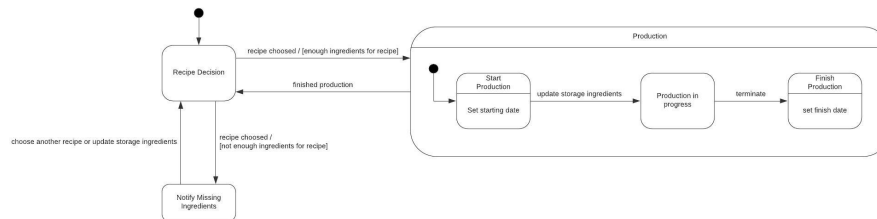


Figura 10: Diagramma di stato

3.6 Diagrammi di attività

Per quanto riguarda i diagrammi di attività la nostra scelta è stata quella di definire un diagramma per il ciclo di vita di una ricetta e il processo di produzione di una ricetta (come per i due diagrammi sopra), e un altro diagramma per la feature "What should I brew today?".

3.6.1 Ciclo di vita della ricetta

In questo diagramma di attività (Figura 11) viene posto sotto un altro punto di vista il processo di produzione di una ricetta. In questo modo assieme al diagramma di sequenza e al diagramma di stato è possibile avere una visione di insieme che non lascia più alcun dubbio sulla modalità da seguire per l'implementazione del processo.

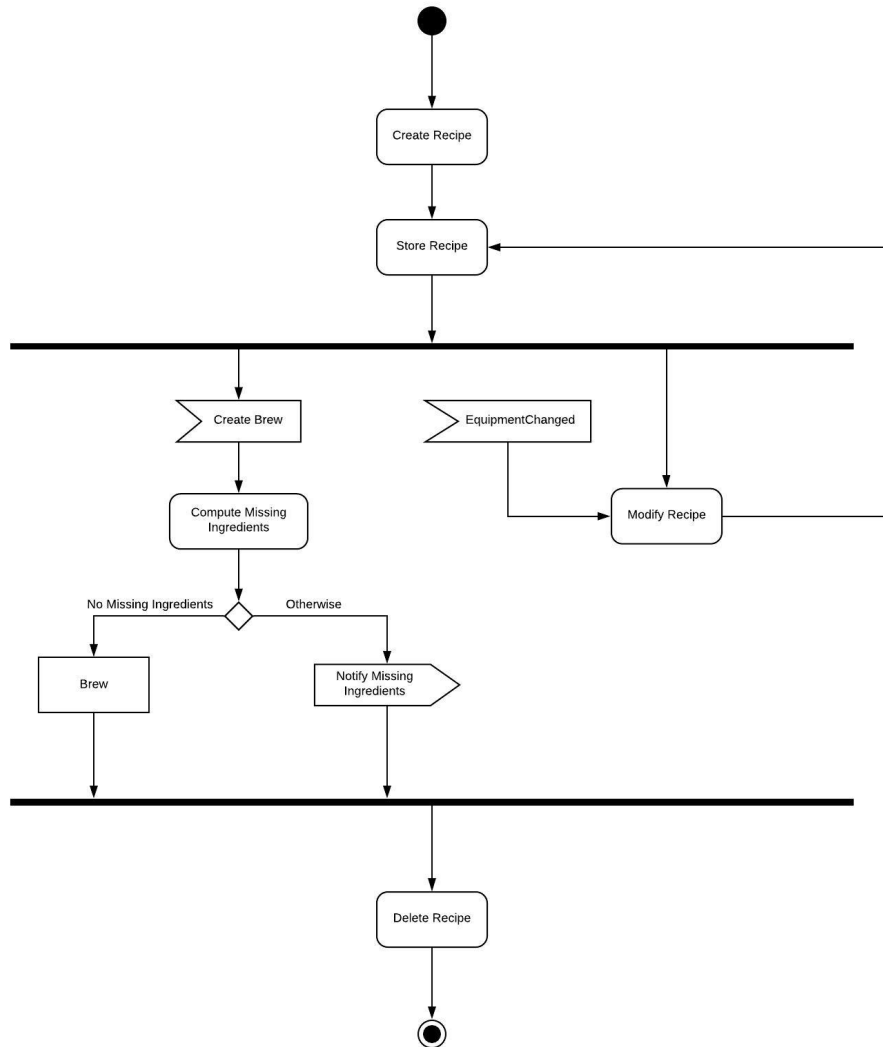


Figura 11: Diagramma di attività - Ciclo di vita della ricetta

3.6.2 Feature "What should I brew today?"

In questo diagramma di attività (Figura 12) viene posta l'attenzione su un'altra funzionalità del programma che abbiamo ritenuto essere abbastanza complessa. Tramite questo diagramma la stesura del codice per l'implementazione della feature è stato molto più fluido.

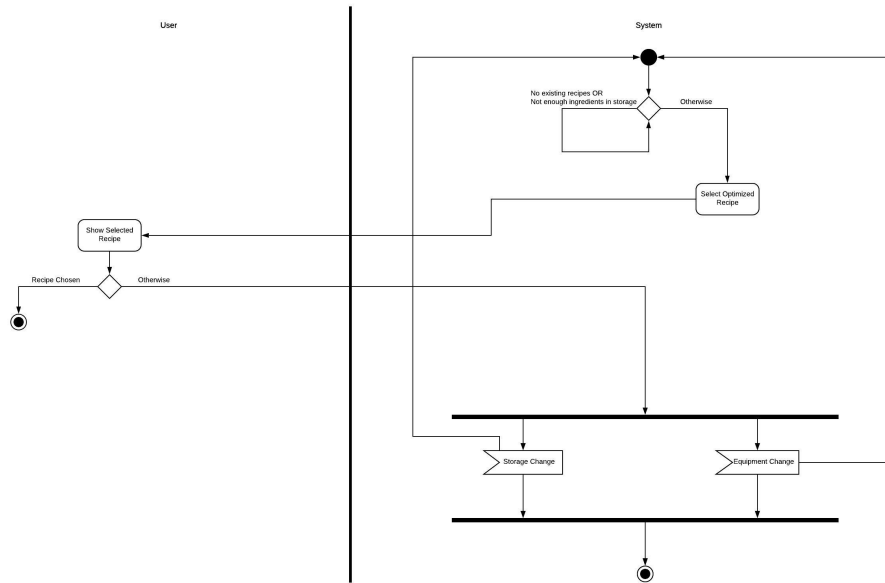


Figura 12: Diagramma di attività - Feature "What should I brew today?"

3.7 Pattern utilizzati e design principle

La progettazione e lo sviluppo del software ci hanno permesso di sperimentare diversi approcci sulla base di diversi pattern. Infine la scelta è ricaduta sull'utilizzo dei pattern e dei design principles elencati di seguito

3.7.1 Design pattern

Il design pattern Singleton è stato l'unico design pattern che abbiamo utilizzato. Fin dal primo diagramma delle classi a livello di progettazione abbiamo ritenuto necessario l'utilizzo di questo pattern, in quanto esigevamo che determinate classi fossero istanziate una sola volta.

Abbiamo preso in considerazione anche diversi altri pattern, tra i quali Observer e Strategy, ma dopo la prima implementazione li abbiamo scartati. Questo perché il pensiero che ci ha guidato nella scelta è stato quello di utilizzare i pattern solo se fosse stato strettamente necessario e soprattutto utile. Questo modus operandi è dovuto sia alla ricerca dello sviluppo di un progetto più leggero in termini generali di complessità, sia per il semplice fatto che i design pattern sono definiti per aiutare lo sviluppo. Risulta quindi inutile utilizzarli se viene meno quest'ultima caratteristica.

3.7.2 Pattern architetturali

Per quanto riguarda i pattern architetturali la scelta è ricaduta sul pattern Model-View-Controller (MVC), poiché fornisce una separazione degli interessi di notevole valore, permettendo inoltre versatilità e semplicità. Un ulteriore punto a favore per il MVC è la possibilità di incastonatura del pattern in una struttura multilayer.

Inoltre abbiamo utilizzato come pattern guida nell'implementazione del software anche il Transaction Script, che va di pari passo al MVC, e permette di gestire in singole procedure ogni singola richiesta dallo strato di presentazione.

3.7.3 Design principle

I design principle che abbiamo cercato di seguire sono due: CRP (Common Reuse Principle) e ADP (Acyclic Dependency Principle). Tramite il CRP e la separation of concerns hanno guidato la suddivisione delle classi in package distinti, mentre con l'ADP abbiamo cercato di fare in modo di non avere dipendenze cicliche tra package.

4 Implementazione

La fase di implementazione e sviluppo è stata la più lunga. Spesso è capitato di doversi fermare, fare un passo indietro e ripartire, oppure cercare di sviluppare in contemporanea diverse soluzioni per determinati scogli che ci si presentavano davanti, per scegliere alla fine la soluzione migliore.

A nostro parere, questa è stata la fase più delicata e difficile di tutto il progetto, che ha consolidato il gruppo di lavoro e ha permesso di sperimentare il vero lavoro in team. In questa fase ci sono stati molti scambi di idee e diversi tentativi falliti, ma alla base di questo c'è sempre stata la massima disponibilità da parte di tutti i membri del team di scegliere con criterio l'alternativa migliore per il successo del progetto.

4.1 Stesura codice

Come detto, la parte di stesura del codice è stata la più lunga del progetto, e diverse volte è stata anche sospesa e ripresa per lasciare spazio ad ulteriori analisi e progettazioni. Una prima stesura del codice è iniziata dopo la prima fase di analisi e progettazione, avendo alla base il primo diagramma delle classi a livello di progettazione (Figura 5). L'idea era quella di implementare il software utilizzando un singolo punto di accesso al database, da implementare in maniera relazionale in SQL.

Dopo una successiva fase di analisi, introducendo il pattern MVC, abbiamo però optato per una diversa implementazione, gestendo il traffico di input/output tramite singoli file in locale, evitando così l'utilizzo di SQL che presupponeva la connessione dell'applicazione in remoto al server centrale, con database SQL. L'implementazione tramite file in locale invece ha permesso una portabilità migliore del software e una semplificazione sul lato di accesso al database.

La modalità scelta per l'input/output è derivata da una fase di sperimentazione in cui abbiamo testato le librerie Gson e Jackson per la rappresentazione degli oggetti Java in formato JSON, e la più semplice ed immediata forma di serializzazione degli oggetti tramite l'interfaccia Serializable. Alla fine abbiamo optato per quest'ultima alternativa siccome era la scelta che rendeva più performante l'applicazione.

Conclusa la parte di Model e Controller, siamo passati al testing. Fase critica in cui la risoluzione di alcuni bug ha portato alla conseguente risoluzione di altri bug e ad un lieve refactoring. La gestione delle eccezioni è stato il passo successivo, per poi arrivare all'implementazione della GUI.

La codifica della parte View è stata eseguita in JFrame. Abbiamo elaborato una grafica minimale che potesse mettere in luce tutti gli aspetti dell'applicazione, ma che fosse semplice ed usabile. Tramite l'utilizzo del software con l'interfaccia grafica, è stato possibile scovare altri bug e vulnerabilità che non erano ancora emersi, permettendo una ulteriore fase di debugging, di refactoring e di collaudo, seguita da ulteriori test.

Infine, come ultima fase, c'è stata la parte più emozionante in cui è stato generato l'eseguibile.

4.2 Git

L'utilizzo di Git per la riuscita del progetto è stata fondamentale, abbiamo usato costantemente questo software tramite l'estensione di Eclipse e ci siamo trovati molto bene. L'unico piccolo pelo nell'uovo è stata la risoluzione dei conflitti, da cui non sempre è stato semplice uscirne.

4.3 Sonar

Per tenere costantemente sotto controllo l'andamento del progetto, abbiamo utilizzato l'alternativa SonarCloud. Tool molto utile e versatile, ha permesso sempre una visione chiara dei bug e dei code smells presenti nel codice. Mantenerlo costantemente sott'occhio ha facilitato l'azione di refactoring che è risultata veloce e immediata.

La situazione finale risulta essere quella proposta nella Figura 13.

Necessario specificare che determinati tipi di code smell sono stati etichettati come falsi positivi. Questa scelta è stata dettata dalla nostra decisione di seguire un determinato tipo di implementazione. Infatti, le categorie di code smell trascurate sono:

- Utilizzo di un logger: la nostra scelta implementativa è stata quella di usare il classico `"System.out.println()"`. Questa scelta è dovuta al fatto che l'applicazione risulta di piccole dimensioni e senza livelli di errore da definire, quindi l'uso di `"System.out.println()"` ci è parsa la soluzione migliore.
- Definire serializzabili determinati attributi: la problematica relativa agli attributi che SonarCloud specificava di definire serializzabili è il fatto che la serializzazione degli stessi comportava la serializzazione di svariate classi in librerie Java. Questo problema ci ha portato a definire questa categoria come falso positivo.
- Albero di ereditarietà troppo profondo: questo code smell è apparso non appena abbiamo iniziato a lavorare sulla parte grafica. Infatti, per ogni finestra che creavamo, compariva il relativo code smell. La scelta di etichettarlo come falso positivo risiede nel fatto che ogni finestra estende l'interfaccia `JFrame`, ed essa risulta ad una profondità tale da scaturire code smell per ogni classe che si collega al nodo di `JFrame` nell'albero dell'ereditarietà.

Infine, rimangono alcuni code smell sulla complessità ciclomatica e dei security hotspot sulle espressioni regolari e sulla deserializzazione. I primi sono visibili anche tramite le metriche di Understand nella Figura 14, e rimangono irrisolti perché sono comunque semplici da interpretare e una loro riformulazione potrebbe costare giorni interi di lavoro. Per quanto riguarda le espressioni regolari ci siamo assicurati che non dessero dei problemi di sicurezza, mentre per la deserializzazione la sicurezza rimane contemplata in quanto gli oggetti deserializzati sono interni al software e gestiti in maniera corretta.

Per quanto riguarda la percentuale di coverage e la percentuale di codice duplicato, ci sono da fare un paio di considerazioni: per la prima, abbiamo escluso

dal calcolo della percentuale le classi dei package "main.java.gui", poiché non reputavamo necessaria l'azione di testing di queste ultime; per quanto riguarda il codice duplicato invece, abbiamo notato un'impennata nella percentuale proprio nel momento di inizio della programmazione dell'interfaccia grafica. Così, una volta finita quest'ultima, è stata eseguita un'azione di refactoring volta all'eliminazione del codice duplicato delle classi relative alla GUI tramite la creazione di una nuova classe contenente i metodi che risultavano essere duplicati nelle altre classi.

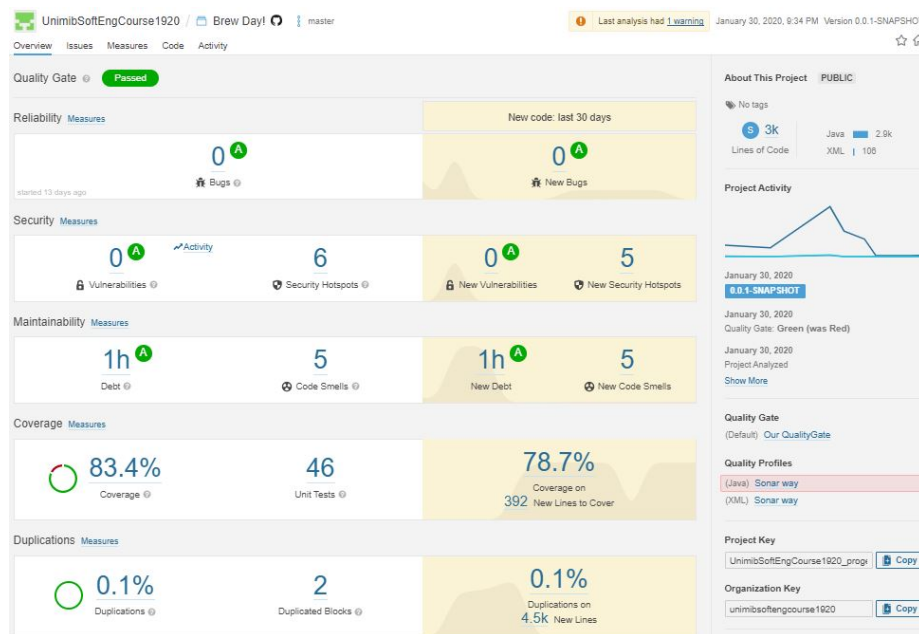


Figura 13: Schermata SonarCloud al termine del progetto

4.4 Understand

Questo tool è stato utilizzato nella parte finale del progetto. Tramite Understand è stato possibile visualizzare graficamente statistiche, metriche e dipendenze che compongono il codice. Il report generato tramite Understand fornisce interessanti grafici e diagrammi che volgono il progetto sotto un punto di vista statistico e generale.

Inoltre tramite Understand abbiamo generato diversi grafi delle dipendenze per cercare di capire se fossero presenti eventuali antipattern strutturali legati a problemi di dipendenza, ma essendo il progetto abbastanza piccolo non è risaltata nessuna situazione per cui fosse possibile stabilire con fermezza che in quel determinato punto fosse presente un certo antipattern.

Ad ogni modo, di seguito abbiamo riportato un paio di grafici prodotti da understand (Figura 14 e Figura 15), che secondo noi sono tra i più rilevanti.

Il primo rappresenta la metrica standard di Understand, dove viene reso visibile tramite la grandezza dei rettangoli il conteggio delle linee per ogni classe (rappresentate dai rettangoli), e la relativa massima complessità cicломatica

definita tramite la colorazione, chiara se la complessità ciclomatica è bassa, scura se è alta. Mentre il secondo grafico visualizza la distribuzione del volume di codice, si può quindi vedere il numero di linee di codice bianche, di linee contenenti commenti, e di linee contenenti codice.

Un ulteriore report generale fornito da Understand è presente nella cartella "Report Understand" locata nella cartella principale del progetto.

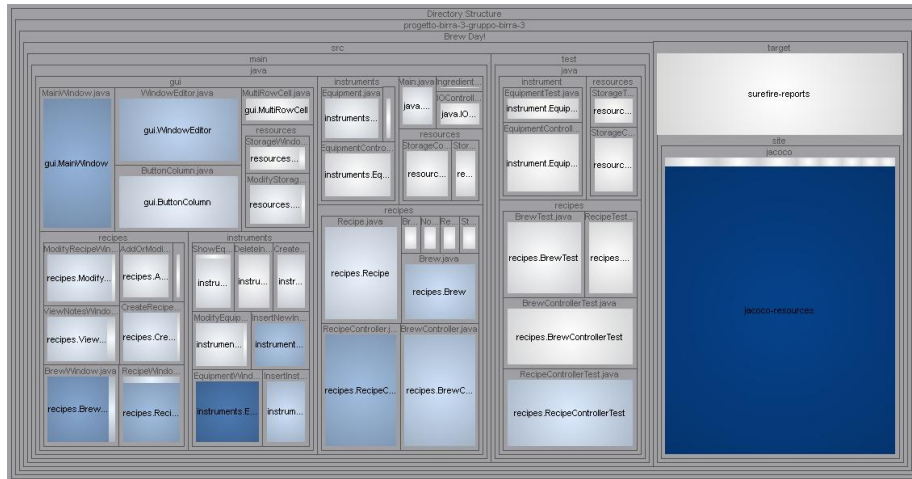


Figura 14: MetricsTreemap-CountLine-MaxCyclomatic

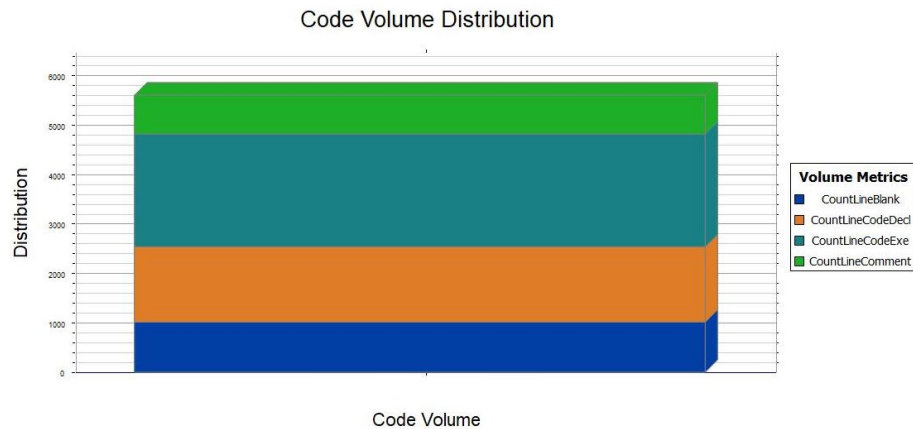


Figura 15: Distribuzione del volume di codice

5 Considerazioni finali

5.1 Diagramma di Gantt finale

Il diagramma di Gantt finale (Figura 16 e Figura 17) vede confermata l'ipotesi iniziale che prevedeva uno stravolgimento nelle tempistiche delle varie attività. Infatti si può notare che rispetto al diagramma di Gantt iniziale (Figura 1 e Figura 2), in questo diagramma innanzitutto sono presenti ulteriori attività e milestone, e inoltre anche la durata prevista è stata smentita.

Nota: la fase di release comprende più attività, quali la stesura della documentazione e della relazione, oltre alla creazione dell'eseguibile dell'applicazione.
















ID	Modalità attività	Nome attività	Durata	Inizio	Fine	Predecessori	Nomi risorse
1		Use Case Diagram	1 g	dom 22/12/19	dom 22/12/19		Locatelli;Cazzetta;Cusini
2		Domain Model Diagram	1 g	lun 23/12/19	lun 23/12/19		Locatelli;Cazzetta;Cusini
3		Class Diagram	3 g	mar 24/12/19	gio 26/12/19	2	Cazzetta;Cusini;Locatelli
4		Software Architecture Diagram	1 g	ven 27/12/19	ven 27/12/19	3	Cazzetta;Cusini;Locatelli
5		Sequence Diagram	1 g	sab 28/12/19	sab 28/12/19	3	Cazzetta
6		State Diagram	1 g	sab 28/12/19	sab 28/12/19	3	Cusini
7		Activity Diagram	1 g	sab 28/12/19	sab 28/12/19	3	Locatelli
8		Fundamental analysis completed	0 g	sab 28/12/19	sab 28/12/19	3	
9		Coding	7 g	dom 29/12/19	sab 04/01/20	4	Cazzetta;Cusini;Locatelli
10		Class Diagram Review	1 g	sab 04/01/20	sab 04/01/20	3	Cazzetta;Cusini;Locatelli
11		Coding	20 g	dom 05/01/20	gio 30/01/20	4;10	Cazzetta;Cusini;Locatelli
12		Business logic completed	0 g	lun 20/01/20	lun 20/01/20	11	
13		GUI completed	0 g	sab 25/01/20	sab 25/01/20	12	
14		Testing	13 g	mar 14/01/20	gio 30/01/20	11;13	Cazzetta;Cusini;Locatelli
15		Release	4 g	mar 28/01/20	ven 31/01/20	14;15	Cazzetta;Cusini;Locatelli

Figura 16: Tabella relativa al diagramma di Gantt finale

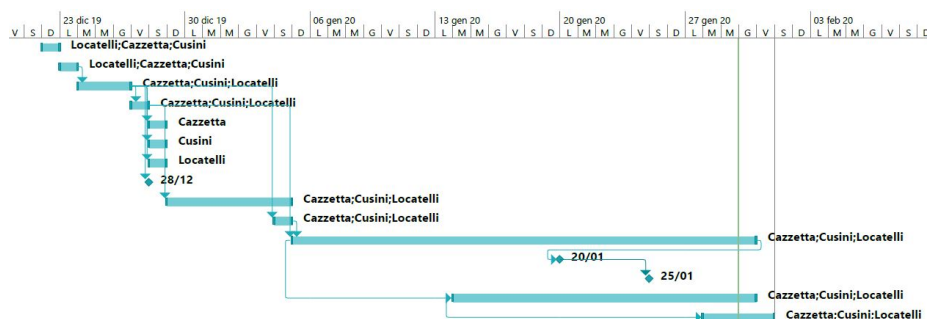


Figura 17: Diagramma di Gantt finale

5.2 Commenti

In generale il progetto è stato una bella sfida, ci ha messo alla prova davanti a molti aspetti verso cui non ci eravamo mai posti. L'inesperienza ha avuto un ruolo rilevante sia in termini positivi, sia in termini negativi, in quanto non avendo mai svolto un progetto di tale portata, per molti aspetti abbiamo dovuto affidarci all'autodidattica e all'istinto, ma questo non ha fatto altro che rafforzarci e concederci una visione a tutto tondo del progetto.

Tramite questo progetto è stato possibile mettere in pratica diverse conoscenze

acquisite in questi anni, che di fatto non avevamo mai potuto testare concretamente. Abbiamo potuto vedere da vicino e toccare con mano tutte le varie fasi di analisi, progettazione e sviluppo del software, e soprattutto abbiamo capito com'è lavorare in team utilizzando diversi tool che sono utilizzati da moltissime aziende che operano in quest'ambito. In conclusione ci riteniamo molto soddisfatti e con un bagaglio di esperienza molto più pesante rispetto all'inizio.

6 Note

Tutte le immagini relative ai grafici e ai diagrammi presentate nella relazione possono essere visualizzate in modo dettagliato accedendo alla cartella "Analysis and design diagrams", dalla cartella principale.