

Relazione Finale del Progetto d'Esame del corso di Ingegneria del Software Brew-Day!

Componenti del gruppo:
Gilardi Alessandro - 866035
Qazim Toska - 847361
Refolli Francesco - 865955

Anno Accademico 2022-2023

Contents

| | | |
|----------|---|-----------|
| 1 | Introduzione | 3 |
| 1.1 | Descrizione del Progetto | 3 |
| 1.2 | Processo Adottato | 4 |
| 1.3 | Utilizzo di GitHub | 7 |
| 2 | Analisi | 8 |
| 2.1 | Glossario | 8 |
| 2.2 | Requisiti Funzionali | 9 |
| 2.3 | Requisiti non funzionali | 10 |
| 2.4 | Casi d'Uso | 11 |
| 2.4.1 | Identificazione dei Casi d'Uso | 12 |
| 2.4.2 | Caso d'Uso in formato breve - aggiungiRicetta | 12 |
| 2.4.3 | Caso d'Uso in formato breve - modificaRicetta | 13 |
| 2.5 | Modello di Dominio | 14 |
| 2.6 | Diagrammi di Sequenza di Sistema | 15 |
| 2.6.1 | SSD - aggiungiRicetta | 15 |
| 2.6.2 | SSD - modificaRicetta | 16 |
| 3 | Progettazione | 17 |
| 3.1 | Architettura del Software | 17 |
| 3.1.1 | Design Principles | 18 |
| 3.2 | Diagramma delle Classi di Progettazione | 18 |
| 3.2.1 | Diagramma delle Classi Api | 18 |
| 3.2.2 | Diagramma delle Classi Logic | 19 |
| 3.2.3 | Diagramma delle Classi Validation | 19 |
| 3.2.4 | Diagramma delle Classi Persistence | 20 |
| 3.2.5 | Diagramma delle Classi Generation | 20 |
| 3.3 | Diagrammi di Sequenza | 21 |
| 3.3.1 | SD - modificaQuantitaIngrediente | 21 |
| 3.3.2 | SD - modificaNomeIngrediente | 22 |
| 3.4 | Diagramma degli Stati | 22 |
| 3.5 | Diagramma delle Attività | 23 |
| 3.6 | Pattern Utilizzati | 24 |
| 3.6.1 | Pattern Architetturali | 24 |
| 3.6.2 | Design Pattern | 24 |

| | | |
|----------|--|-----------|
| 4 | Implementazione e Testing | 25 |
| 4.1 | Scelte Implementative - Backend | 25 |
| 4.2 | Scelte Implementative - Frontend | 25 |
| 4.3 | Testing - Frontend | 25 |
| 4.4 | Testing - Backend | 25 |
| 5 | Conclusioni | 26 |
| 5.1 | Analisi con SonarQube e SonarCloud | 26 |
| 5.2 | Analisi con Understand | 28 |
| 5.3 | Considerazioni Finali | 30 |
| 5.4 | Link Utili | 30 |

Chapter 1

Introduzione

1.1 Descrizione del Progetto

Brew Day! è un'applicazione che consente ai birrai casalinghi produttori di birra "all-grain" di mantenere un database organizzato delle loro ricette di birra. Consente quindi agli utenti di creare, archiviare e modificare le ricette e successivamente eliminarle.

Ogni home brewer ha un'attrezzatura specifica, le cui caratteristiche portano a un particolare "batch size", il numero massimo di litri che possono essere erogati in un singolo ciclo. Le ricette prevedono l'uso di ingredienti comuni come l'acqua, malti, luppolo, lieviti, zuccheri, additivi, oltre ad ingredienti aggiuntivi che rendono la birra particolare. L'applicazione memorizza le quantità degli ingredienti in una misura "assoluta", in unità per Litro, in modo da consentire una conversione diretta della ricetta quando l'apparecchiatura, e di conseguenza la dimensione del batch, viene aggiornata.

Oltre alle ricette effettive, l'applicazione offre la possibilità di mantenere le istanze di ricette eseguite, ovvero particolari miscele basate su una ricetta.

Le birre create possono essere accompagnate da note per fare riferimento a problemi che possono influire sulla birra risultante e che i birrai vorrebbero tenere registrati. Un particolare tipo di note sono le note di degustazione, che consentono ai birrai di tenere traccia delle opinioni su una birra di una particolare birra.

Oltre a queste funzionalità più tradizionali di Brew Day!, l'applicazione mantiene un elenco di ingredienti disponibili. Ciò consente ai birrai di essere informati sugli ingredienti mancanti per la birra successiva.

Un'istanza di ricetta, ovvero una particolare birra, consente agli utenti di aggiornare l'elenco degli ingredienti disponibili, sottraendo gli ingredienti utilizzati da quelli disponibili.

Brew Day! supporta la funzione "cosa dovrei preparare oggi?" che consiglia la ricetta che massimizza l'uso degli ingredienti disponibili, tenendo conto della capacità dell'attrezzatura. Per informazioni sull'installazione e uso dell'applicazione è possibile visitare la sezione Wiki su GitHub.

1.2 Processo Adottato

E' stato adottato il Framework Scrum al fine tenere sotto controllo lo stato di avanzamento del progetto, ispezionando frequentemente il lavoro fatto per verificare che si stia procedendo verso gli obiettivi posti e, nel caso si stia deviando dagli obiettivi, fare delle correzioni.

Abbiamo deciso di effettuare 4 sprint, ognuno della durata di 5 giorni.

Sono state svolte le seguenti attività:

- Meeting iniziale, il primo giorno, in cui abbiamo creato la Product Backlog e il Gantt (figure 1.1, 1.2, 1.3, 1.4)
- Sprint Planning, all'inizio di ogni Sprint, in cui abbiamo pianificato la Sprint Backlog e identificato lo Sprint Goal
- Daily Scrum, meeting online giornaliero di 15 minuti in cui ci siamo confrontati sul lavoro svolto il giorno precedente e pianificato i compiti del giorno per il raggiungimento dello Sprint Goal
- Sprint Review e Sprint Retrospective, al termine di ogni Sprint, in cui abbiamo ispezionato il lavoro svolto, valutato se l'obiettivo prefissato è stato raggiunto e con quali risultati e infine valutato cosa continuare a fare, cosa smettere di fare e cosa migliorare nello sprint successivo per ottenere performance ancora più efficienti.

La qualità del codice è stata monitorata utilizzando i tool Understand e SonarQube, oltre a SonarCloud.

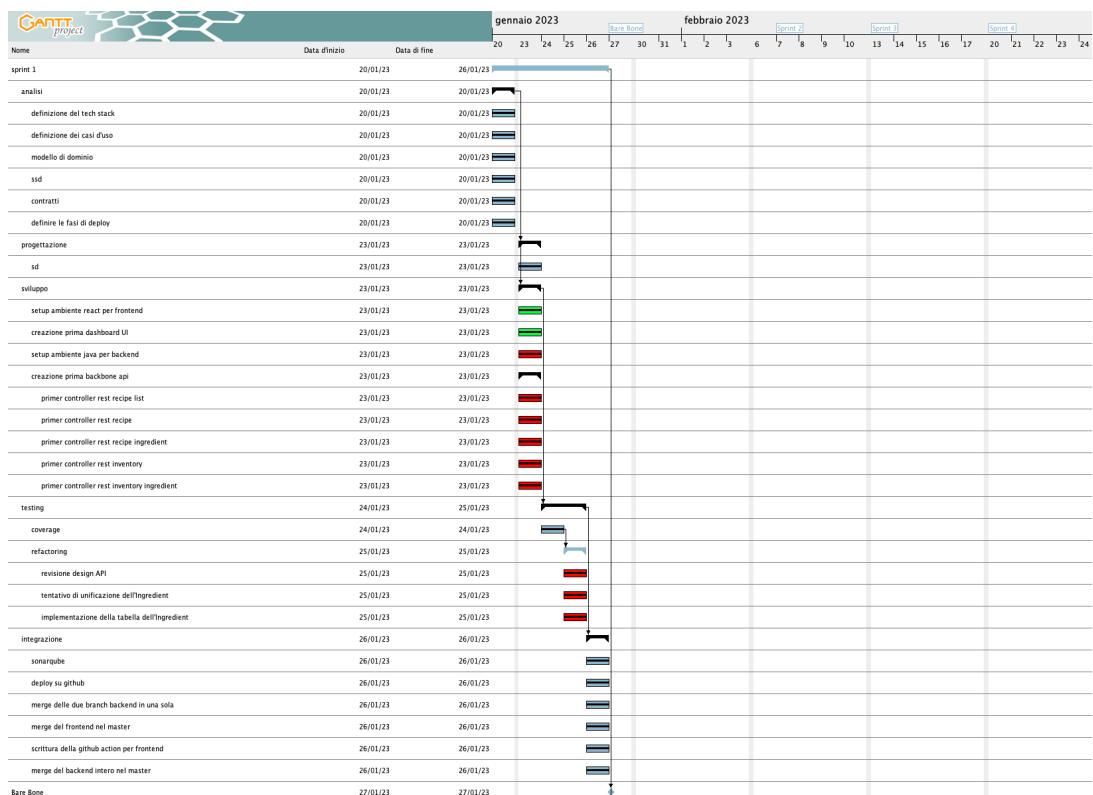


Figure 1.1: Gantt - Sprint 1

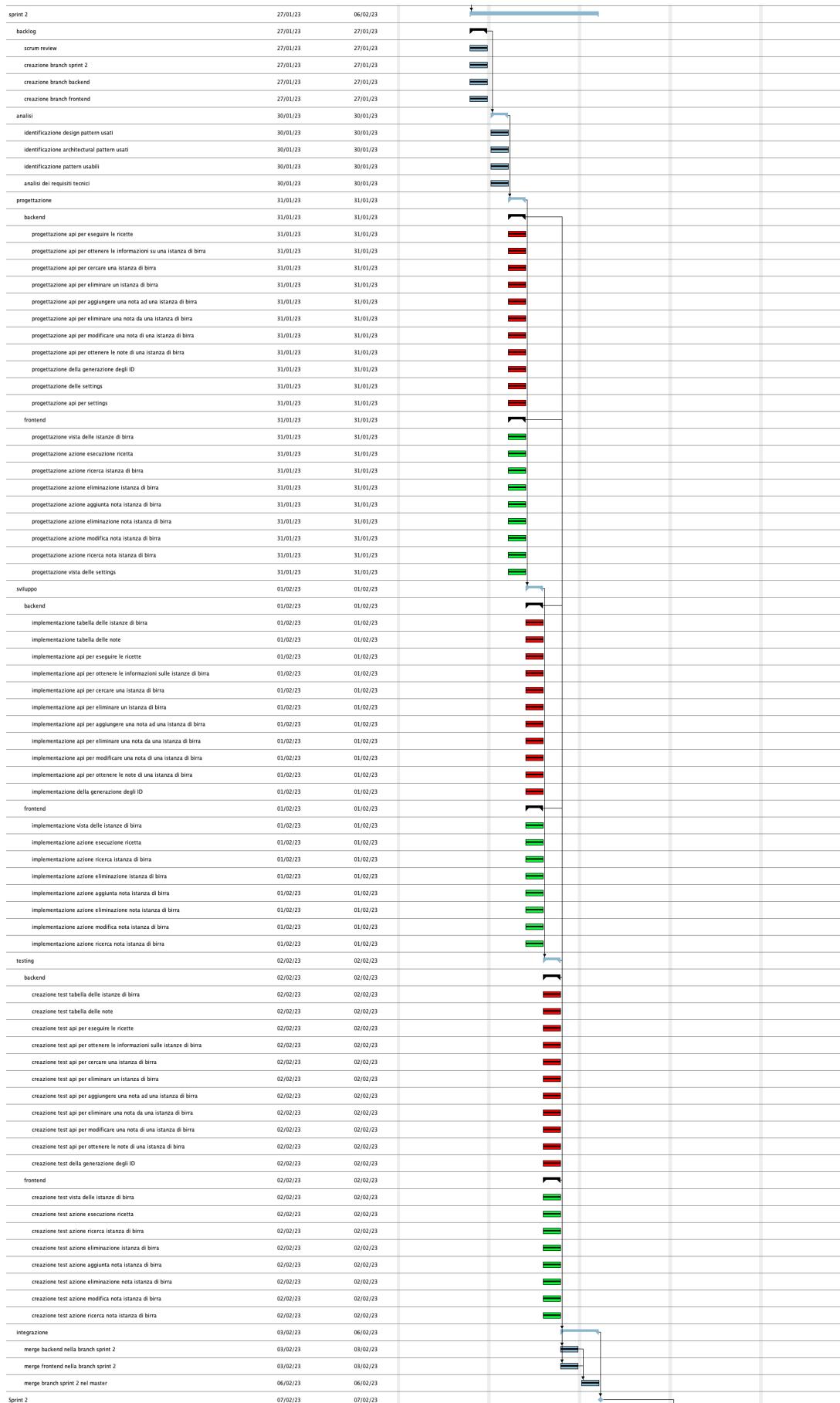


Figure 1.2: Gantt - Sprint 2

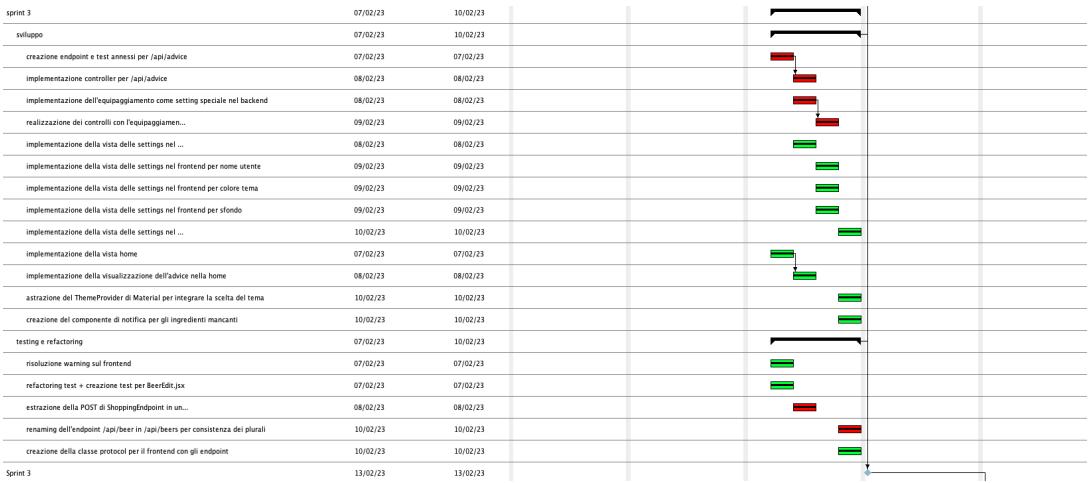


Figure 1.3: Gantt - Sprint 3

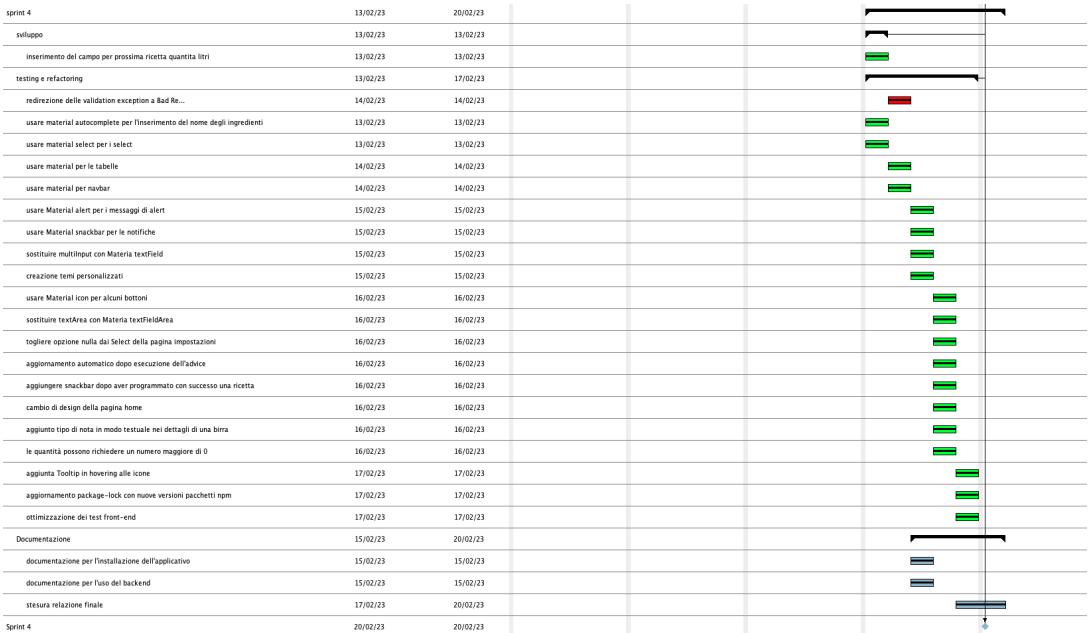


Figure 1.4: Gantt - Sprint 4

1.3 Utilizzo di GitHub

Per tenere traccia dell'attività e dello stato di avanzamento del progetto abbiamo utilizzato gitHub:

- Per ciascuno Sprint abbiamo definito una milestone che la caratterizza
- Per ogni attività di sviluppo abbiamo definito una issue che la descriva, all'interno della corrispondente milestone
- Abbiamo fatto ampio uso dei branch per lavorare su soluzioni parziali o questioni più ampie
- Sono state utilizzate le GitHub Action per automatizzare la fase di test, build e analisi con SonarCloud
- E' stato utilizzato il task Tracker offerto da gitHub nella sezione Project per condividere i progressi effettuati tra i componenti del gruppo
- E' stata adottata la seguente strategia di gestione del branch (figura 1.5): una branch master, con il codice stabile; una branch per ogni sprint (associato ad una milestone), da cui si forkano le varie branch che contengono le feature della milestone; ogni k merge / commit sulla branch di sprint si rilascia una preview sul master

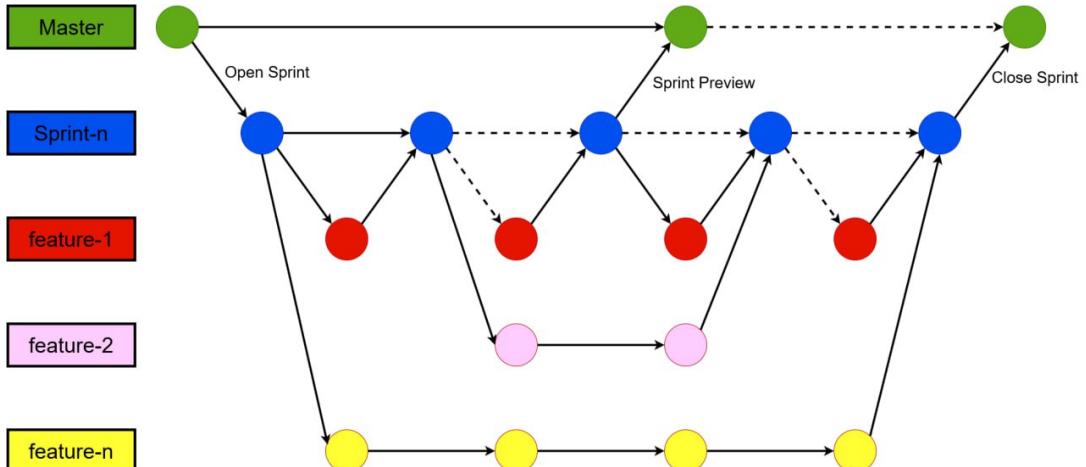


Figure 1.5: Branch Strategy

Chapter 2

Analisi

2.1 Glossario

All'inizio del progetto è stato realizzato il glossario riportato nella tabella 2.1.

| Nome | Descrizione |
|------------------|---|
| birraio | un utente |
| equipaggiamento | bollitore, fermentatore, pipa da birra |
| capacita' | quantita' in litri che un equipaggiamento puo' supportare in una turnata |
| ingrediente | malto, luppolo, lievito, zucchero, acqua, additivi |
| ricetta | collezione di ingredienti con associata una quantita' |
| inventario | collezione di ingredienti che l'home brewer ha a disposizione |
| consiglio | ricetta che massimizza l'uso degli ingredienti nell'inventario |
| istanza di birra | anche chiamata 'istanza di ricetta' nel testo, birra prodotta con una certa ricetta |

Table 2.1: Glossario

2.2 Requisiti Funzionali

All'inizio del progetto sono stati individuati i requisiti funzionali, a ciascuno dei quali è stata attribuita un'importanza, come prevede la tecnica MoSCoW (tabella 2.2).

| Requisito | MoSCoW |
|---|--------|
| il sistema deve permettere all'utente di mantenere, aggiornare, eliminare ricette | M |
| il sistema deve permettere all'utente di mantenere l'inventario | M |
| il sistema deve permettere all'utente di indicare che una ricetta e' stata eseguita e quindi aggiornare l'inventario di conseguenza | M |
| il sistema deve permettere all'utente di indicare che ha fatto la spesa e quindi aggiornare l'inventario di conseguenza | M |
| il sistema deve permettere all'utente di produrre la lista della spesa per gli ingredienti mancanti di una ricetta | M |
| il sistema deve permettere all'utente di generare un consiglio per la prossima birra | M |
| il sistema deve permettere all'utente di mantenere le istanze di una ricetta | M |
| il sistema deve permettere all'utente di aggiungere, aggiornare, eliminare note alle istanze di una birra | M |
| il sistema deve notificare l'utente quando mancano degli ingredienti per la prossima birra | M |

Table 2.2: Requisiti Funzionali

2.3 Requisiti non funzionali

Oltre ai requisiti funzionali, abbiamo individuato anche quelli non funzionali (tabella 2.3).

| Requisito | MoSCoW |
|---|--------|
| il sistema deve mantenere le quantità degli ingredienti nelle ricette (e nell'inventario) in termini di unità assolute (anche diverse), in modo che sia più semplice calcolare i multipli | M |
| il sistema deve supportare le note normale e le note di sapore per le istanze di una ricetta | M |
| il suggerimento della birra deve massimizzare l'uso di ingredienti e equipaggiamento | M |
| il sistema può supportare la possibilità di aggiungere immagini alle istanze di birra | C |
| si deve permettere di eliminare una ricetta che ha associate delle birre prodotte | M |

Table 2.3: Requisiti Non Funzionali

2.4 Casi d'Uso

Nei primi giorni del progetto abbiamo identificato i seguenti casi d'uso sulla base dei requisiti del progetto.

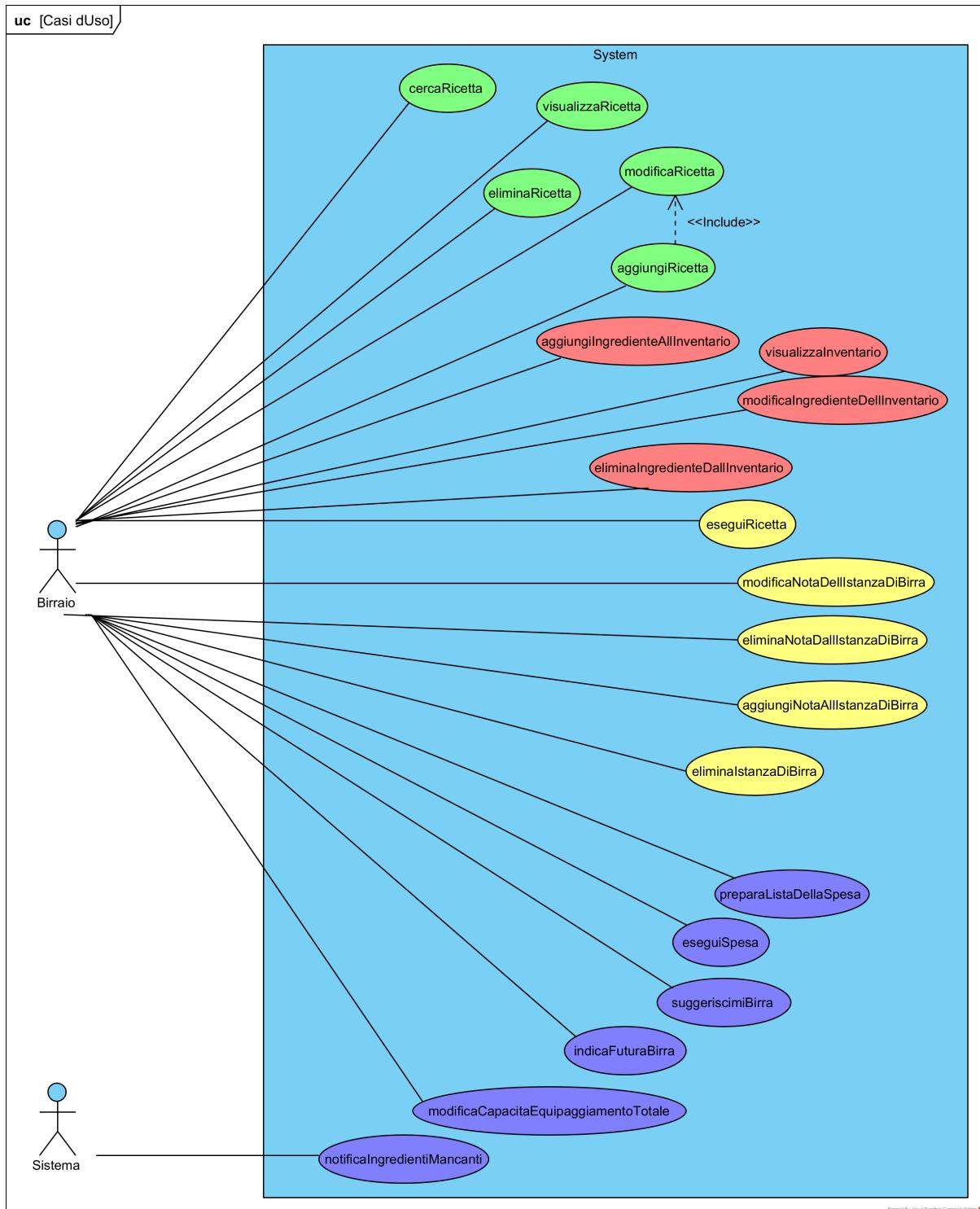


Figure 2.1: Casi d'Uso

2.4.1 Identificazione dei Casi d'Uso

Casi d'Uso suddivisi nelle 4 iterazioni svolte:

1. Prima Iterazione

- aggiungiRicetta
- modificaRicetta
- eliminaRicetta
- cercaRicetta
- visualizzaRicetta
- visualizzaInventario
- aggiungiIngredienteAllInventario
- modificaIngredienteNellInventario
- eliminaIngredienteDallInventario

2. Seconda Iterazione

- eseguiRicetta
- eliminaIstanzaDiBirra
- aggiungiNotaAllIstanzaDiBirra
- eliminaNotaDallIstanzaDiBirra
- modificaNotaDellIstanzaDiBirra

3. Terza Iterazione

- eseguiSpesa
- indicaFuturaBirra
- preparaListaDellaSpesa
- suggeriscimiBirra
- notificaIngredientiMancanti
- modificaCapacitaEquipaggiamentoTotale

4. Quarta Iterazione - nessuno

2.4.2 Caso d'Uso in formato breve - aggiungiRicetta

E' stato realizzato il caso d'Uso in formato breve di aggiungiRicetta nella prima iterazione

1. Il birraio inizia l'immissione di una nuova ricetta.
2. Il birraio inserisce il nome della nuova ricetta.
3. Il birraio può inserire una descrizione della ricetta.
4. import caso d'uso modificaRicetta
5. il sistema salva la ricetta

2.4.3 Caso d'Uso in formato breve - modificaRicetta

E' stato realizzato il caso d'Uso in formato breve di modificaRicetta nella prima iterazione

1. Il birraio inizia la modifica di una ricetta.
2. while:
 - (a) if opt1:
 - i. inserisce il nome di un ingrediente
 - ii. inserisce la quantita' dell'ingrediente
 - (b) if opt2:
 - i. individua l'ingrediente da rinominare
 - ii. inserisce il nuovo nome dell'ingrediente
 - (c) if opt3:
 - i. individua un ingrediente
 - ii. inserisce la nuova quantita' dell'ingrediente
 - (d) if opt4:
 - i. seleziona l'ingrediente da eliminare
 - (e) if opt5:
 - i. inserisce il nuovo nome della ricetta
 - (f) if opt6:
 - i. inserisce la nuova descrizione della ricetta
- (g) il sistema salva la ricetta

2.5 Modello di Dominio

Durante le iterazioni è stato realizzato il modello di dominio (figura 2.2), il quale fornisce una rappresentazione visuale delle classi concettuali e degli oggetti del mondo reale del dominio.

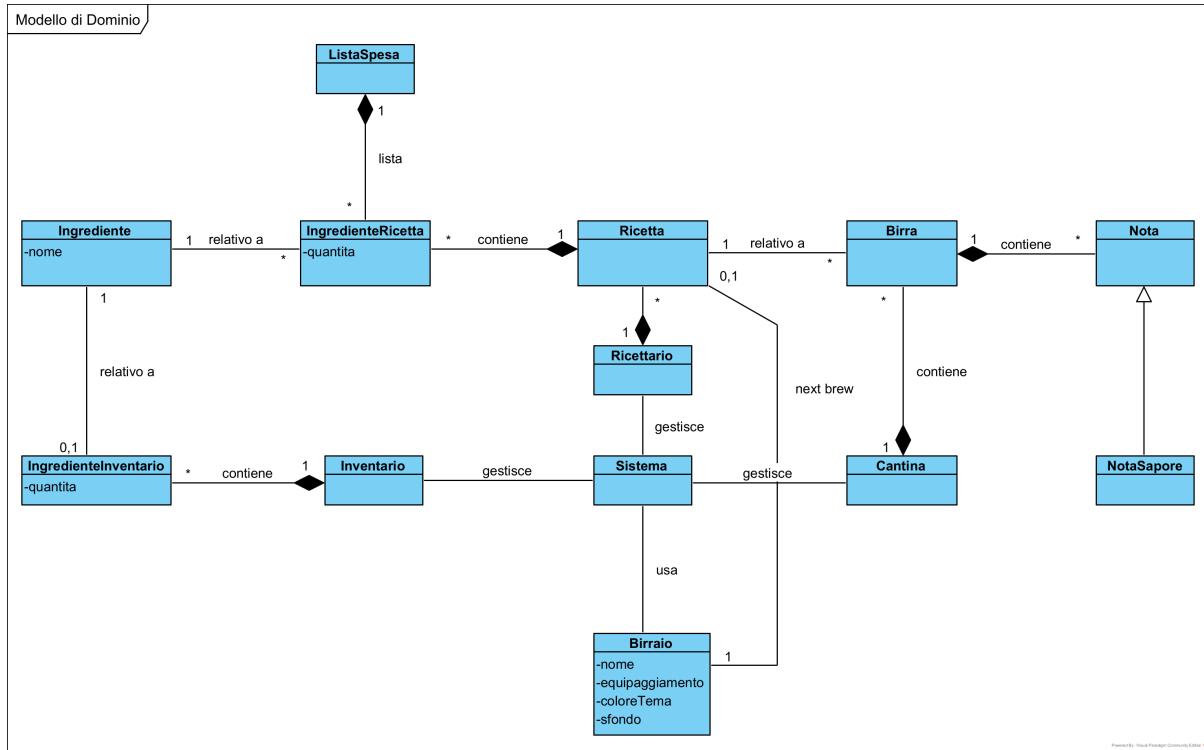


Figure 2.2: Modello di Dominio

2.6 Diagrammi di Sequenza di Sistema

Nella prima iterazione sono stati realizzati due diagrammi di sequenza di sistema per rappresentare l'interazione tra l'utente esterno al sistema con l'applicazione. I casi scelti sono l'aggiunta di una ricetta nel sistema e la sua modifica.

2.6.1 SSD - aggiungiRicetta

E' stato realizzato il diagramma di sequenza di sistema del caso d'uso aggiungiRicetta

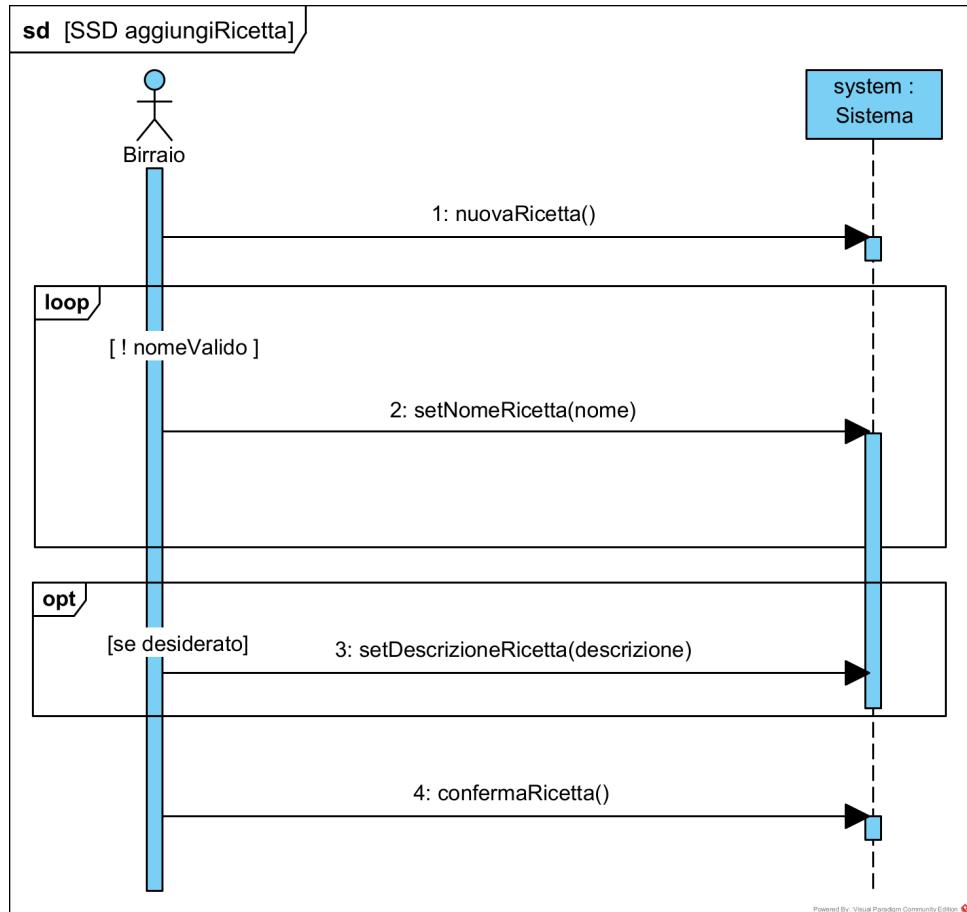


Figure 2.3: SSD - aggiungiRicetta

2.6.2 SSD - modificaRicetta

E' stato realizzato il diagramma di sequenza di sistema del caso d'suo modificaRicetta

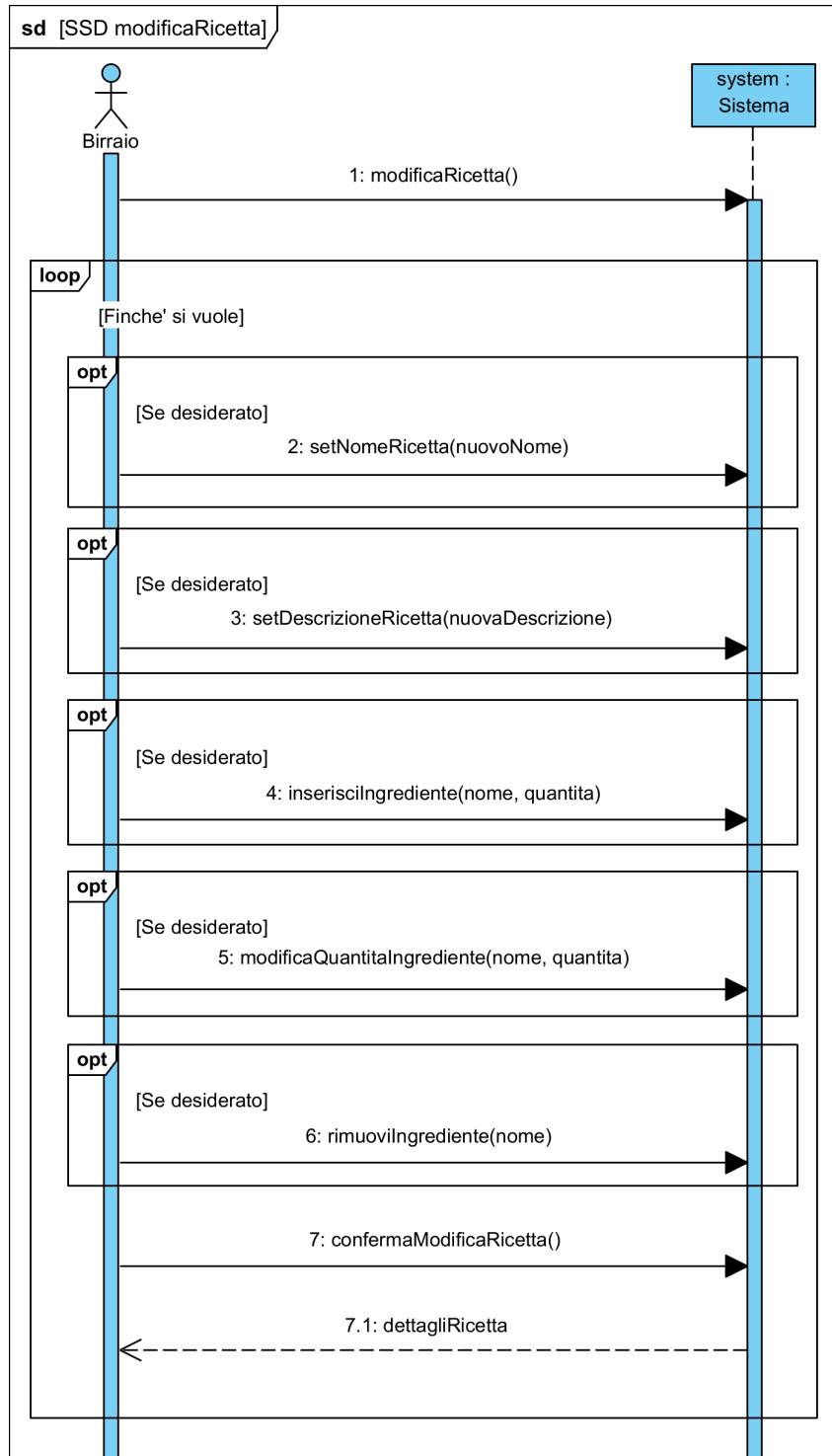


Figure 2.4: SSD - modificaRicetta

Chapter 3

Progettazione

3.1 Architettura del Software

Come architettura software è stato scelto di usare un'architettura multi-strato di tipo client-server. Abbiamo suddiviso logicamente le varie funzionalità del software su più strati differenti in comunicazione tra loro. Gli strati individuati sono UI, Application, Domain, Technical Services.

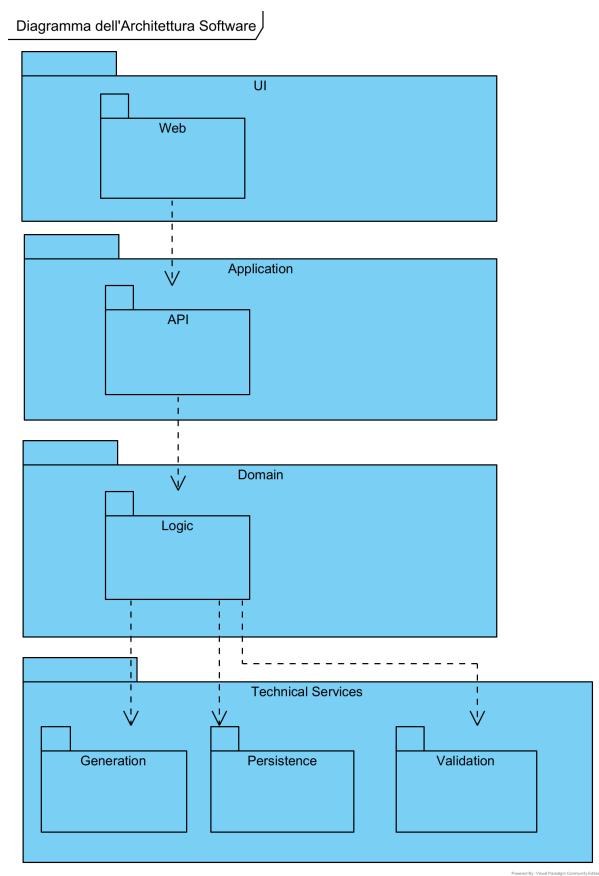


Figure 3.1: Diagramma dell'Architettura Software

3.1.1 Design Principles

Interface Segregation Principle

Applicando questo principio di sviluppo ci siamo sincerati di aver suddiviso il più possibile le responsabilità e le attività svolte dai singoli controller.

Single Responsibility Principle

Le classi Handler per la validazione dei dati e le classi che compongono il package Generation aggiungono una e una sola responsabilità al contesto in cui sono utilizzate. Per esempio si è scelto appositamente di spezzettare il più possibile la catena per la generazione degli ID separando la classe che usa una funzione di Hashing da quella che genera un Seed randomico da usare insieme al Seed di input. Altro esempio è il fatto che ogni Handler si occupa di un punto specifico e atomico della validazione dell'input.

Open-closed principle

Le classi del package Generation permettono di aggiungere pezzi di computazione senza modificare la classi da cui derivano.

3.2 Diagramma delle Classi di Progettazione

Durante le iterazioni sono stati creati/aggiornati i diagrammi delle classi, che descrivono gli aspetti statici del sistema, classi, la loro struttura e le relazioni con altre classi. Di seguito vengono riportati tutti i diagrammi di classi realizzati, suddivisi per Package. Nello specifico, abbiamo il diagramma delle classi per il package Api, Logic, Persistence, Validation e Generation.

3.2.1 Diagramma delle Classi Api

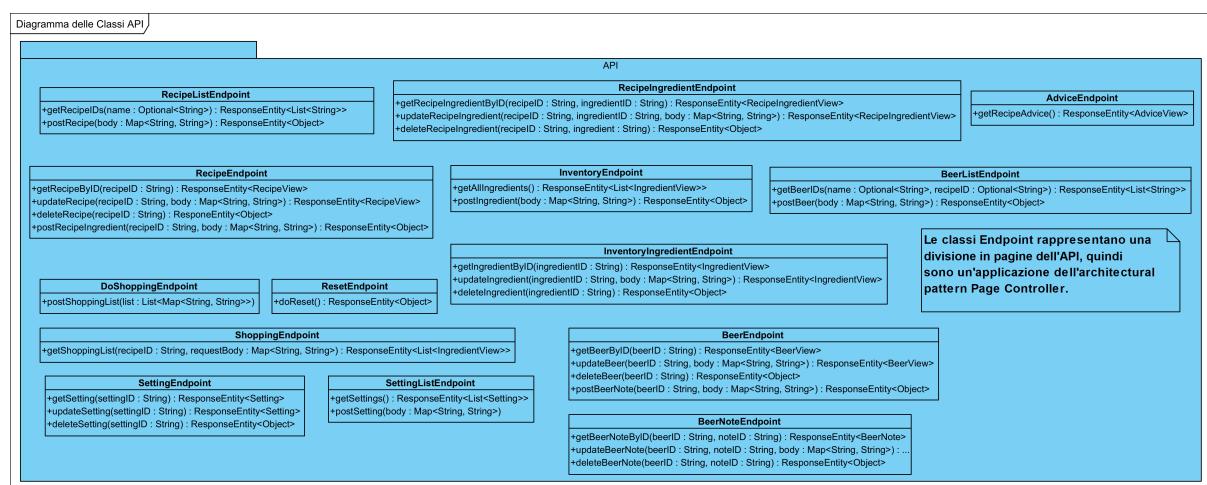


Figure 3.2: Diagramma delle Classi Api

3.2.2 Diagramma delle Classi Logic

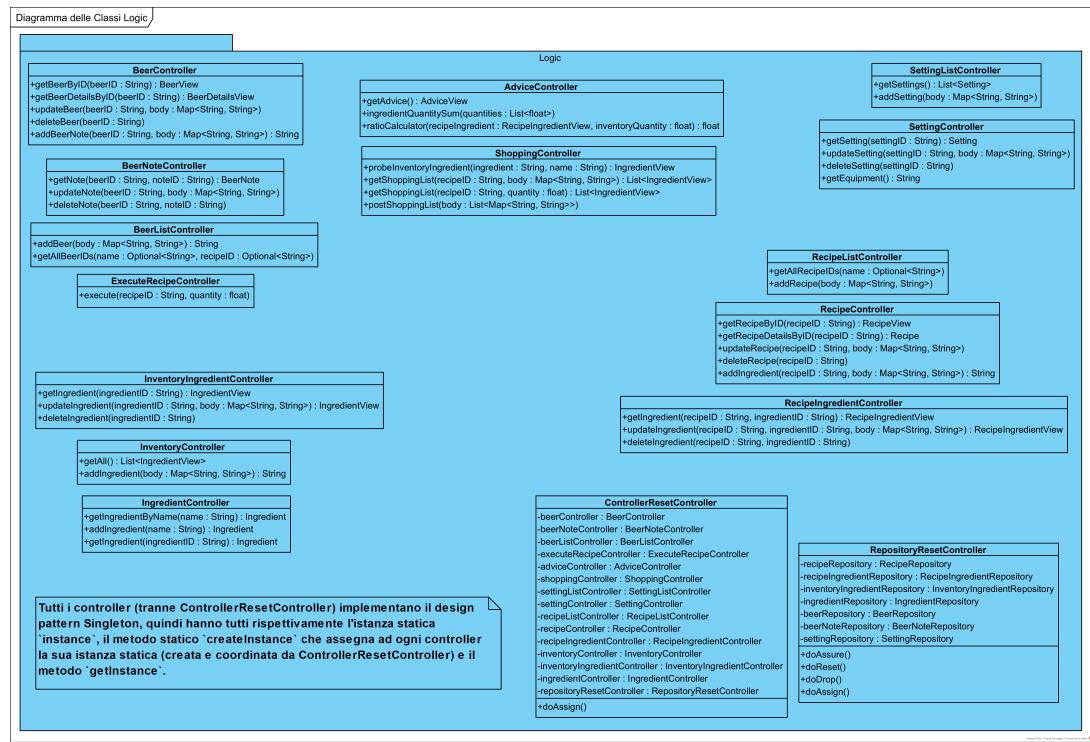


Figure 3.3: Diagramma delle Classi Logic

3.2.3 Diagramma delle Classi Validation

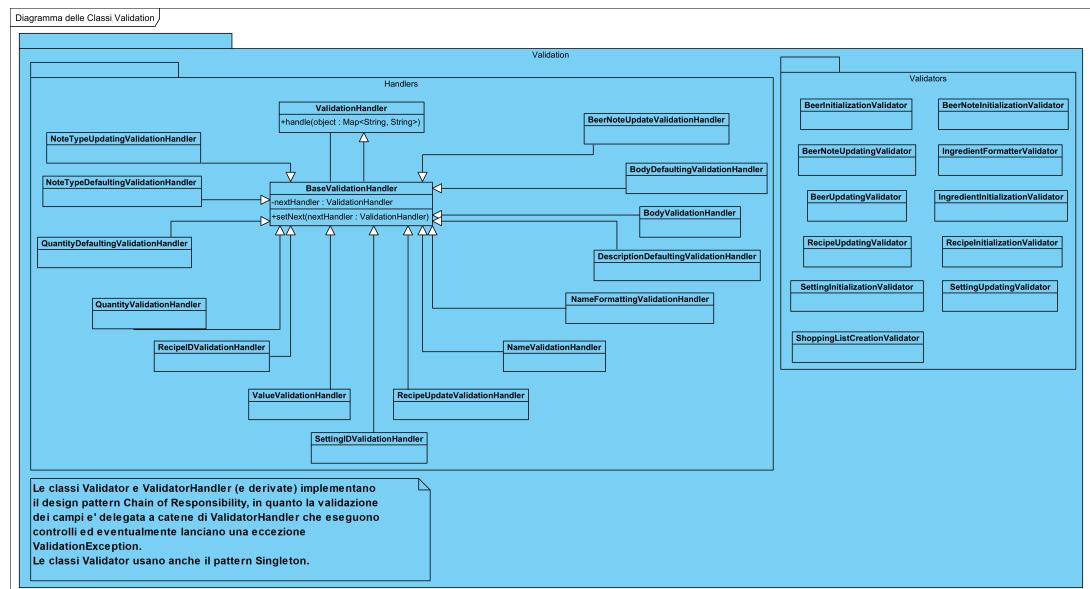


Figure 3.4: Diagramma delle Classi Validation

3.2.4 Diagramma delle Classi Persistence

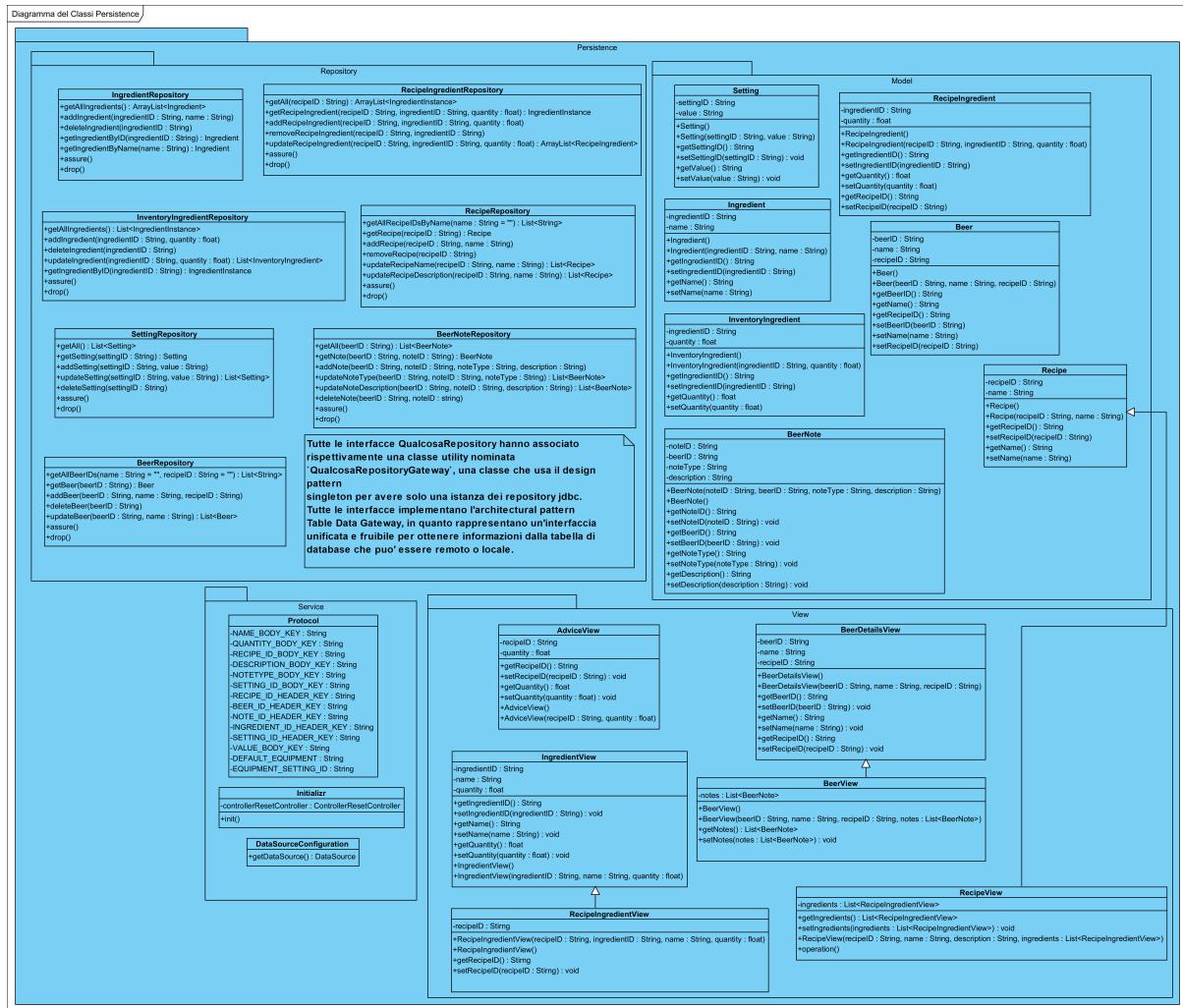


Figure 3.5: Diagramma delle Classi Persistence

3.2.5 Diagramma delle Classi Generation

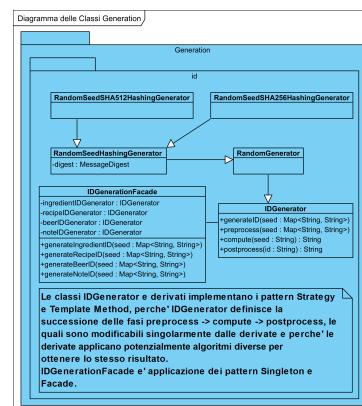


Figure 3.6: Diagramma delle Classi Generation

3.3 Diagrammi di Sequenza

Nella prima iterazione abbiamo realizzato due diagrammi di sequenza per illustrare le interazioni tra gli oggetti del sistema.

3.3.1 SD - modificaQuantitaIngrediente

E' stato realizzato il diagramma di sequenza di modificaQuantitaIngrediente appartenente al caso d'suo modificaRicetta (figura 3.7). Si ha l'interazione tra l'oggetto Ricetta e IngredienteRicetta.

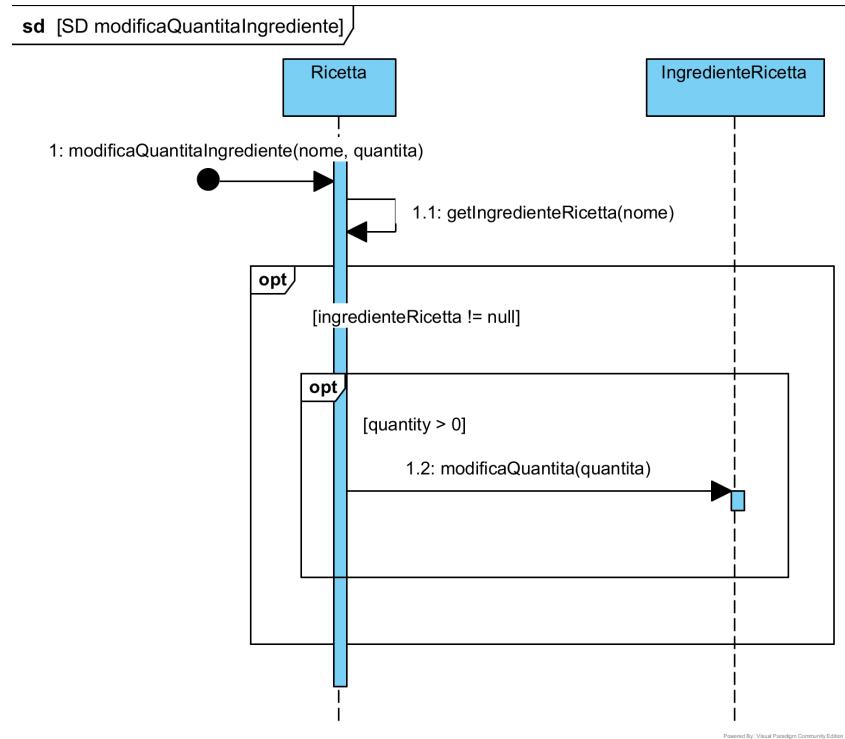


Figure 3.7: SD - modificaQuantitaIngrediente

3.3.2 SD - modificaNomeIngrediente

E' stato realizzato il diagramma di sequenza di modificaNomeIngrediente appartenente al caso d'suo modificaRicetta (figura 3.8). Si ha l'interazione tra l'oggetto Ricetta, IngredienteRicetta e Ingrediente.

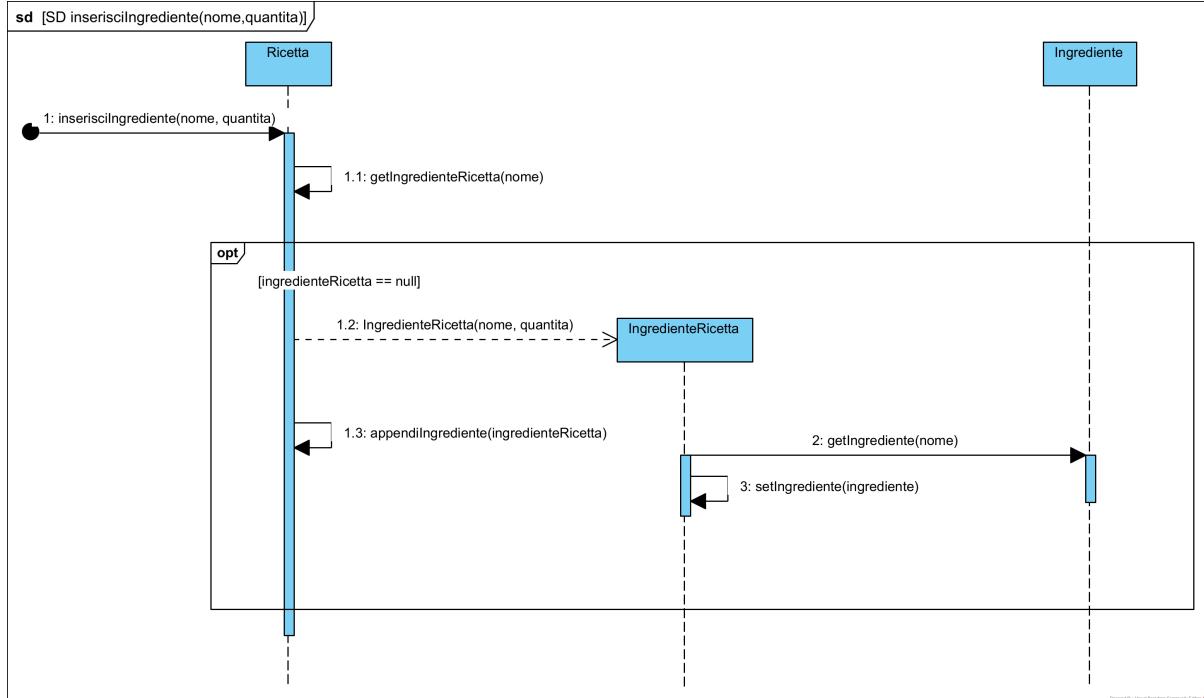


Figure 3.8: SD - inserisciIngrediente

3.4 Diagramma degli Stati

E' stato realizzato il diagramma degli stati della entità Birra per descrivere le modifiche dinamiche degli stati dell'entità stessa (figura 3.9).

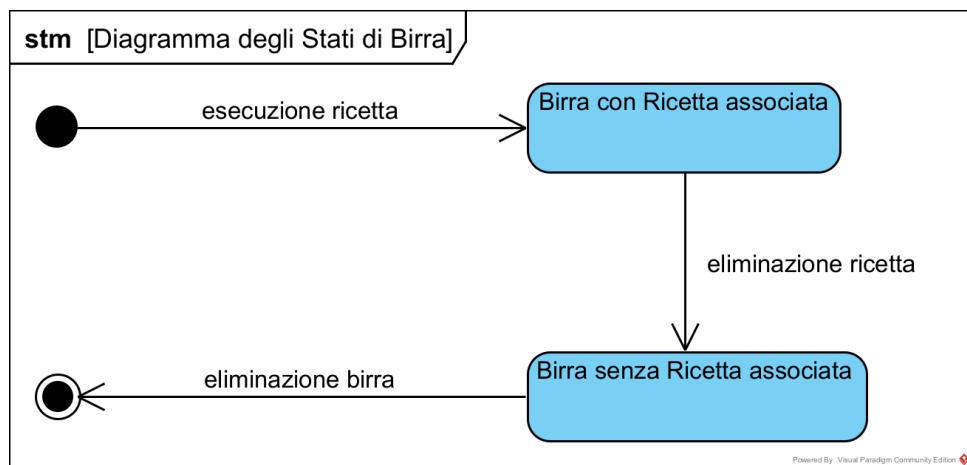


Figure 3.9: Diagramma degli Stati Birra

3.5 Diagramma delle Attività

E' stato realizzato il diagramma delle attività per descrivere il processo di esecuzione di una ricetta (figura 3.10).

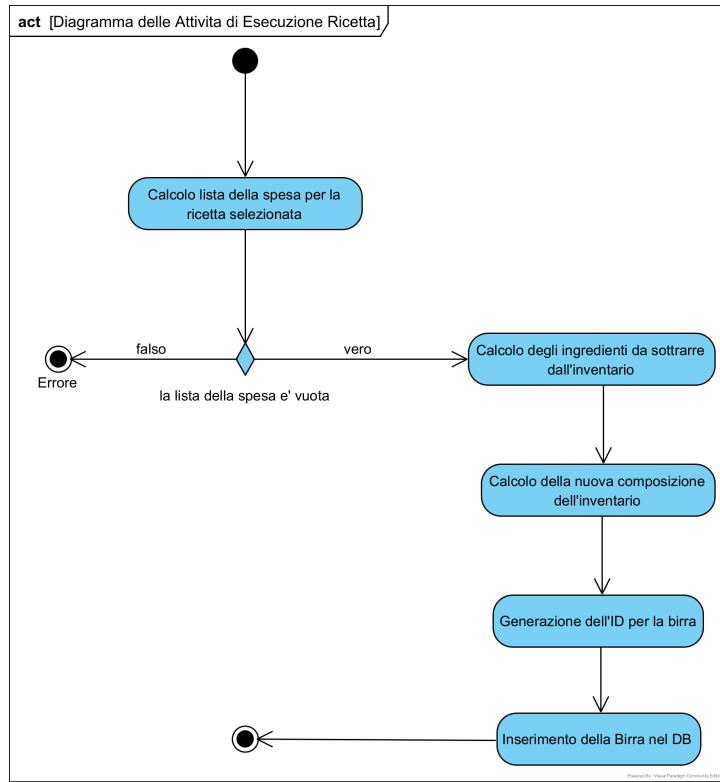


Figure 3.10: Diagramma delle Attività di Esecuzione Ricetta

3.6 Pattern Utilizzati

Nel corso delle fasi di progettazione, implementazione e refactoring abbiamo individuato dei pattern architetturali e dei design pattern.

3.6.1 Pattern Architetturali

Page Controller

Le classi Endpoint rappresentano una divisione in pagine dell'API, quindi sono un'applicazione del pattern architettonico Page Controller.

Table Data Gateway

Tutte le interfacce Repository implementano pattern architettonico Table Data Gateway, in quanto rappresentano un'interfaccia unificata e fruibile per ottenere informazioni dalla tabella di database che può essere remoto o locale.

3.6.2 Design Pattern

Chain of Responsibility

Le classi Validator e ValidatorHandler (e derivate) implementano il design pattern Chain of Responsibility, in quanto la validazione dei campi è delegata a catene di ValidatorHandler che eseguono controlli ed eventualmente lanciano una eccezione ValidationException.

Singleton

Tutti i controller (tranne ControllerResetController) implementano il design pattern Singleton, quindi hanno tutti rispettivamente l'istanza statica instance, il metodo statico createInstance che assegna ad ogni controller la sua istanza statica (creata e coordinata da ControllerResetController) e il metodo getInstance. Tutte le interfacce Repository hanno associato rispettivamente una classe utility nominata RepositoryGateway, una classe che usa il pattern singleton per avere solo una istanza dei repository JDBC. Implementano questo pattern anche le classi Validator, ValidatorHandler e le classi IDGenerator e derivate.

Strategy, Template Method

Le classi IDGenerator e derivate implementano i pattern Strategy e Template Method dal momento che IDGenerator definisce la successione delle fasi preprocess -> compute -> postprocess, le quali sono modificabili singolarmente dalle derivate, e perché le derivate applicano potenzialmente algoritmi diversi per ottenere lo stesso risultato.

Facade

La classe IDGenerationFacade è applicazione dei design pattern Facade.

Chapter 4

Implementazione e Testing

4.1 Scelte Implementative - Backend

Per realizzare il backend abbiamo deciso di utilizzare Spring, framework open source per lo sviluppo di applicazioni su piattaforma Java.

Come interfaccia per il database abbiamo usato JDBC.

4.2 Scelte Implementative - Frontend

Per realizzare l'interfaccia utente abbiamo deciso di utilizzare la libreria open-source React, la quale utilizza come linguaggio JavaScript/JSX.

4.3 Testing - Frontend

Per effettuare i test nel frontend abbiamo utilizzato il framework di test Jest

4.4 Testing - Backend

Per effettuare i test nel backend abbiamo utilizzato il framework di unit testing JUnit.

Chapter 5

Conclusioni

5.1 Analisi con SonarQube e SonarCloud

Durante le fasi di implementazione abbiamo utilizzato il tool SonarQube e la GitHub Action che automatizza l'analisi con SonarCloud per mantenere controllato il livello di qualità del progetto.

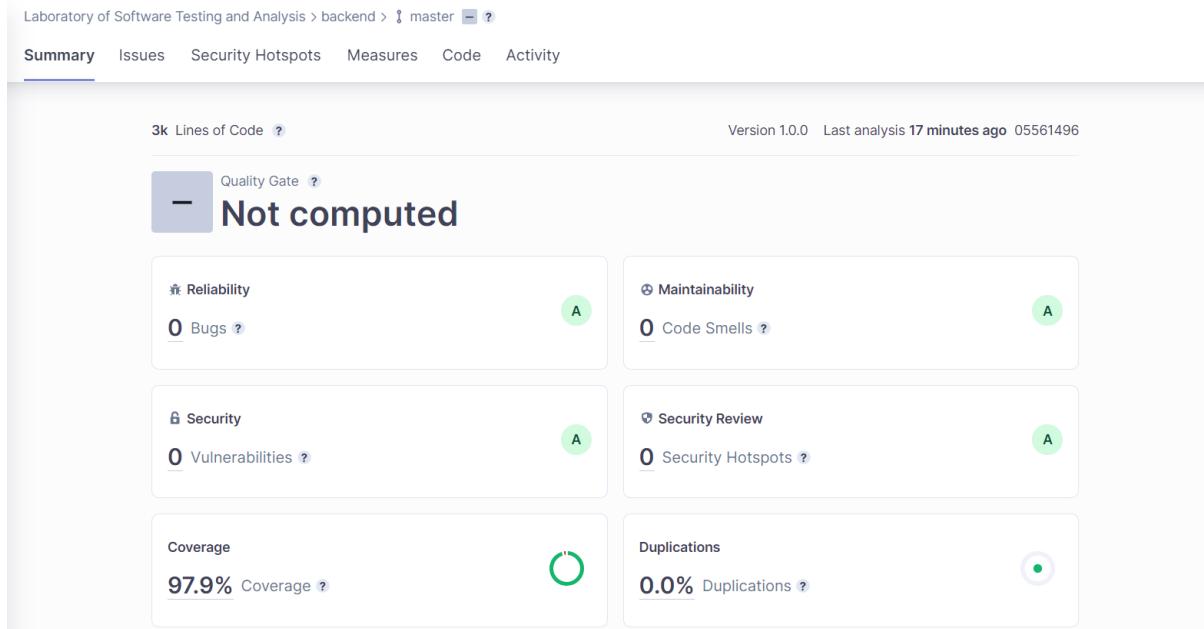


Figure 5.1: sonarCloud-backend

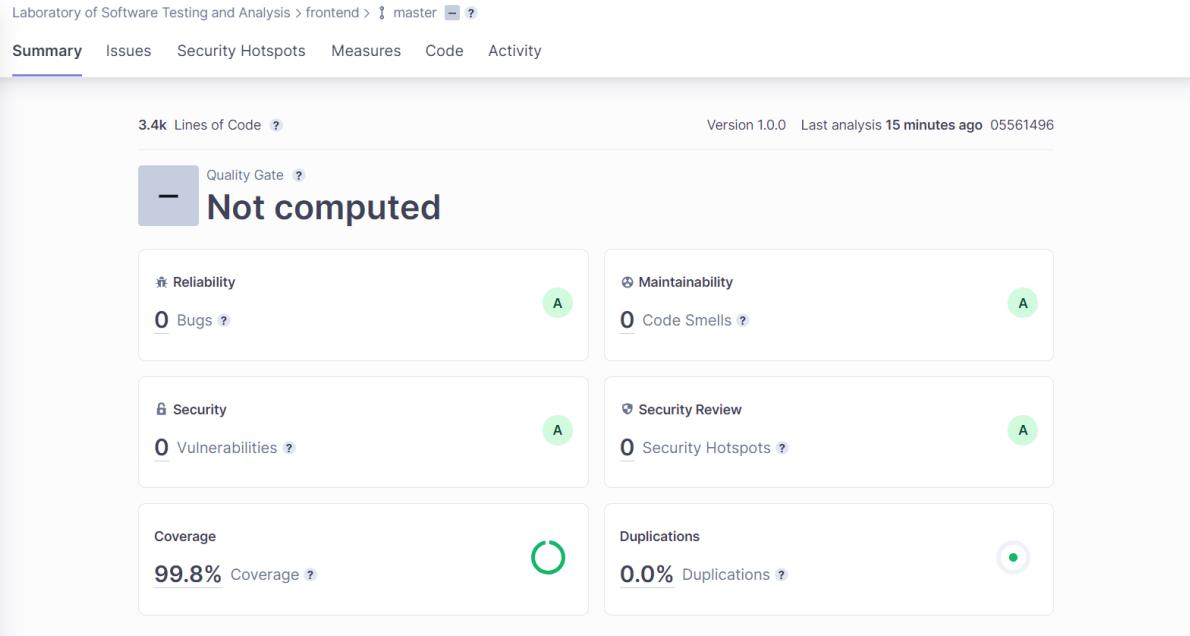


Figure 5.2: sonarCloud-frontend

Dalle analisi finali svolte, i cui risultati sono riportati nelle figure 5.1 e 5.2, si può affermare che il progetto non presenta bugs, code Smells, vulnerabilità e problemi di sicurezza.

Utilizzando i test JUnits per il backend e Jest per il frontend, siamo inoltre riusciti ad ottenere una coverage elevata del codice, pari al 97.8% nel backed e del 94.2% nel frontend.

5.2 Analisi con Understand

Abbiamo utilizzato il tool Understand per capire se nel progetto fossero presenti anti-pattern strutturali legati a problemi di dipendenza o altri problemi.

Secondo un'analisi generale (figura 5.3) si può intuire come il progetto sia privo di problemi e presenti una struttura ben bilanciata.

Sono stati realizzati i grafici metrici per le classi e i metodi sia del front-end (figura 5.4 e 5.5) sia per il back-end (figura 5.6 e 5.7).

Nei grafici la grandezza rappresenta il numero di istruzioni, mentre il colore la complessità ciclomatica.



Figure 5.3: Understand

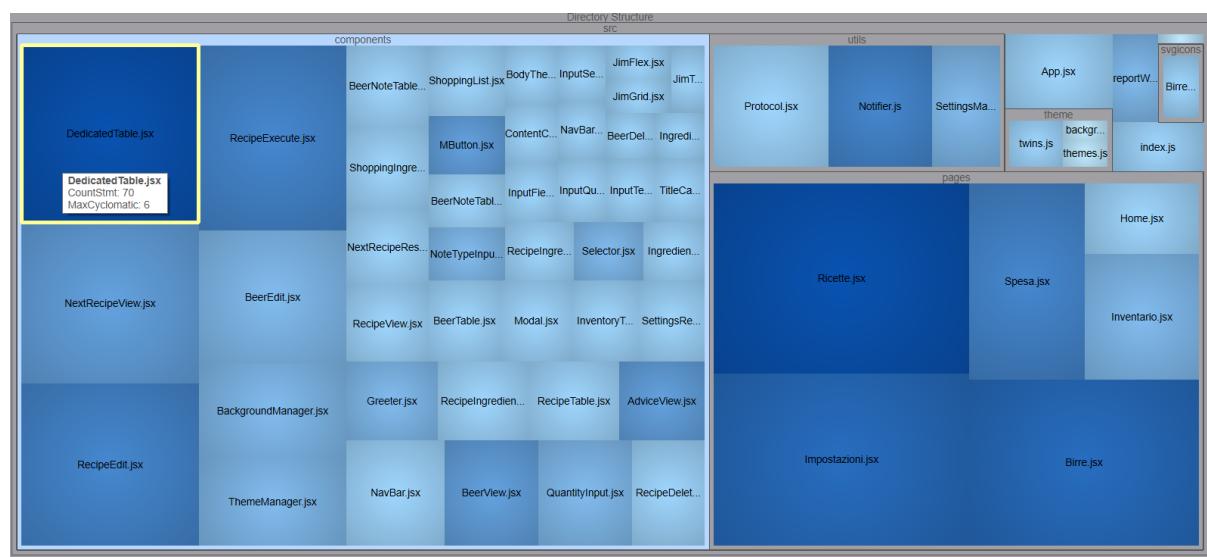


Figure 5.4: Mappatura per Classi - Frontend

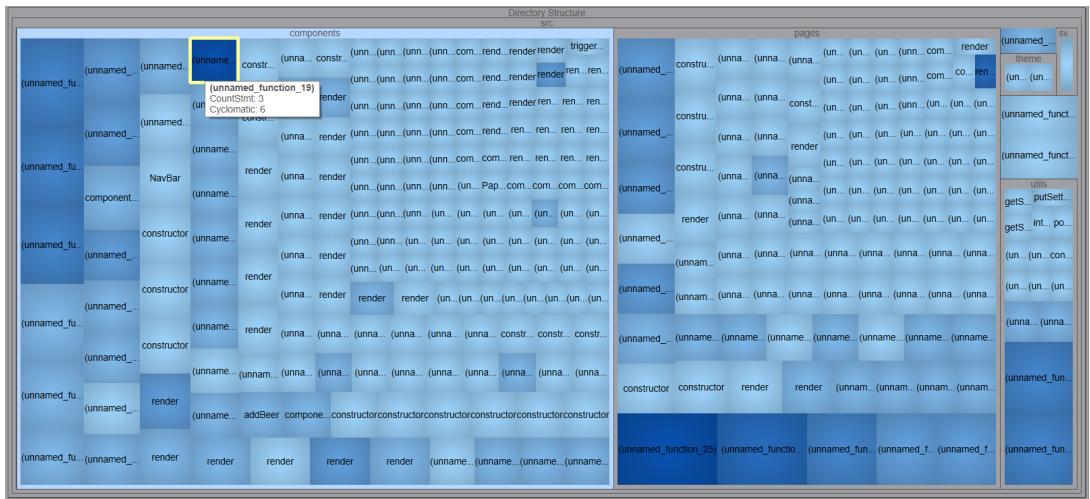


Figure 5.5: Mappatura per Metodi - Frontend



Figure 5.6: Mappatura per Classi - Backend

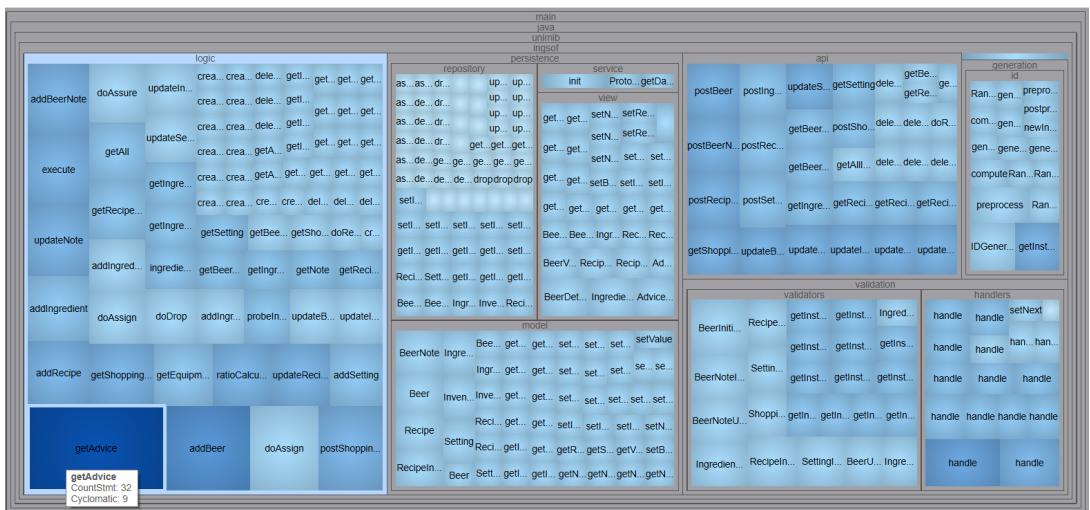


Figure 5.7: Mappatura per Metodi - Backend

5.3 Considerazioni Finali

Consideriamo il progetto completo nelle funzionalità richieste.

Con questo progetto abbiamo potuto realizzare un'applicazione lavorando in team e andando a coprire tutte le fasi, dall'analisi, la progettazione, l'implementazione, il refactoring, fino al rilascio.

Siamo soddisfatti di come abbiamo lavorato in gruppo dal momento che siamo sempre riusciti a comunicare e ad avere le idee chiare su ciò che andava fatto. Tutto ciò è stato semplificato dall'utilizzo del framework Scrum e l'uso proficuo di GitHub.

Abbiamo avuto la possibilità di mettere in pratica le conoscenze di gestione del progetto acquisite da ogni componente del gruppo durante il corso di Analisi e Progettazione del Software e di Ingegneria del Software, oltre ad altri corsi per quanto riguarda l'implementazione.

5.4 Link Utili

Repository GitHub

Wiki su GitHub

Installazione Guidata

Uso dell'Applicazione

Documentazione API

Documentazione Scrum