Relazione finale del progetto di

**Informatica III**

**Modulo di Progettazione e**

**algoritmi**

Prof. Patrizia Scandurra

cod. 38068

Gaetano Comandatore 1090952

Nicholas Iotti 1058728

Laurea Magistrale in Ingegneria Informatica

Università degli Studi di Bergamo

A.A. 2023/2024

I Anno

**Contents**

[1 Introduzione al progetto 4](#_Toc175983561)

[1.1 Modello Organizzativo adottato 4](#_Toc175983562)

[2 ITERAZIONE 0 AMDD (Requirements & Architecture Envisioning) 6](#_Toc175983563)

[2.1 Requirements Envisioning 6](#_Toc175983564)

[2.1.1 Nonfunctional requirements 6](#_Toc175983565)

[2.1.2 Functional requirements (Customer’s requirements) 6](#_Toc175983566)

[2.2 Architecture Envisioning 6](#_Toc175983567)

[2.2.1 Use Case Diagram 7](#_Toc175983568)

[2.2.2 Informal Deployment Diagram 8](#_Toc175983569)

[3 ITERAZIONE 1 AMDD (Early Architecture Design) 9](#_Toc175983570)

[3.1 Deployment View 10](#_Toc175983571)

[3.2 Functionality View 11](#_Toc175983572)

[3.3 Activity Diagram 12](#_Toc175983573)

[3.4 Use Case Stories 13](#_Toc175983574)

[3.4.1 Responsabile della foresteria e/o Receptionist 13](#_Toc175983575)

[3.4.2 Addetto Pulizia 13](#_Toc175983576)

[3.4.3 Noleggiatore 13](#_Toc175983577)

[3.4.4 Partecipante 14](#_Toc175983578)

[3.4.5 Cliente 14](#_Toc175983579)

[3.5 Use Case Stories (fully dressed description) 14](#_Toc175983580)

[3.6 Requirements Analysis 18](#_Toc175983581)

[4 ITERAZIONE 2 AMDD 20](#_Toc175983582)

[4.1 Tool Chain e Tecnologie 20](#_Toc175983583)

[4.1.1 codeMR 20](#_Toc175983584)

[4.1.2 Gson 20](#_Toc175983585)

[4.1.3 MongoDB e Atlas 20](#_Toc175983586)

[4.2 Implementazione 21](#_Toc175983587)

[4.2.1 Selezione funzioni da implementare 21](#_Toc175983588)

[4.2.2 Component Diagram 21](#_Toc175983589)

[4.2.3 Analisi complessità metodo: *WRoom.mettiDati* 22](#_Toc175983590)

[4.2.4 Analisi dinamica 22](#_Toc175983591)

[4.2.5 Analisi statica 24](#_Toc175983592)

[4.2.6 Maschera delle camere 25](#_Toc175983593)

[4.2.7 Conclusioni iterazione 2 25](#_Toc175983594)

[5 ITERAZIONI INTERMEDIE 26](#_Toc175983595)

[6 ITERAZIONE FINALE 28](#_Toc175983596)

[6.1 Architectural Pattern 28](#_Toc175983597)

[6.2 Formal Deployment Diagram 29](#_Toc175983598)

[6.2.1 Versione 1 (two tiers) 29](#_Toc175983599)

[6.2.2 Versione 2 (three tiers) 31](#_Toc175983600)

[6.3 Project Tree 33](#_Toc175983601)

[6.4 Class Diagram (Design Pattern) 35](#_Toc175983602)

[6.4.1 Visitor 35](#_Toc175983603)

[6.4.2 Singleton 36](#_Toc175983604)

[6.4.3 Observer 37](#_Toc175983605)

[6.5 Class Diagrams 38](#_Toc175983606)

[6.5.1 Class: Visitor 38](#_Toc175983607)

[6.5.2 Class: Resource 39](#_Toc175983608)

[6.5.3 Class: Event 40](#_Toc175983609)

[6.5.4 Class: Service 41](#_Toc175983610)

[6.5.5 Class: Controller 42](#_Toc175983611)

[6.5.6 Class: WResource 43](#_Toc175983612)

[6.5.7 Class: WEvent 44](#_Toc175983613)

[6.5.8 Class: Application 45](#_Toc175983614)

[6.5.9 Class: AppConfig 46](#_Toc175983615)

[6.6 Interfacce grafiche 47](#_Toc175983616)

[6.6.1 Maschera Menu 47](#_Toc175983617)

[6.6.2 Maschera WCalendar 47](#_Toc175983618)

[6.6.3 Maschera WRoom 47](#_Toc175983619)

[6.6.4 Maschera WEvent 48](#_Toc175983620)

[6.7 Analisi statica: SonarLint 49](#_Toc175983621)

[6.8 Analisi statica: codeMR 50](#_Toc175983622)

[6.9 Analisi dinamica: JUnit5 55](#_Toc175983623)

[7 DBMS: MongoDB – AtlasDB 56](#_Toc175983624)

[8 Documentazione: JavaDoc 57](#_Toc175983625)

# Introduzione al progetto

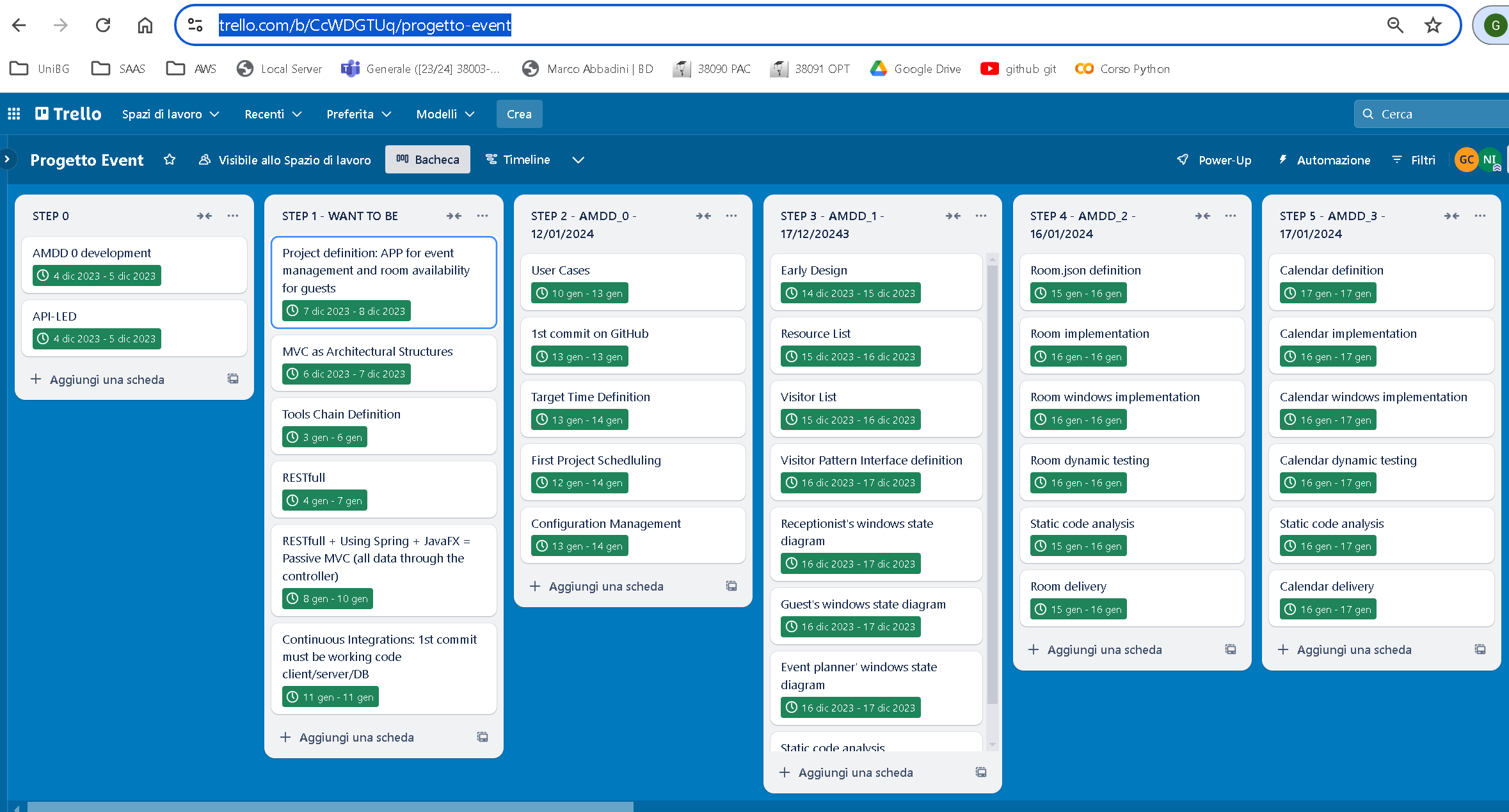
Il progetto consiste nell’implementazione di un’applicazione per l’amministrazione e la gestione di una foresteria che permetta la prenotazione delle sale conferenze e delle camere per gli ospiti.

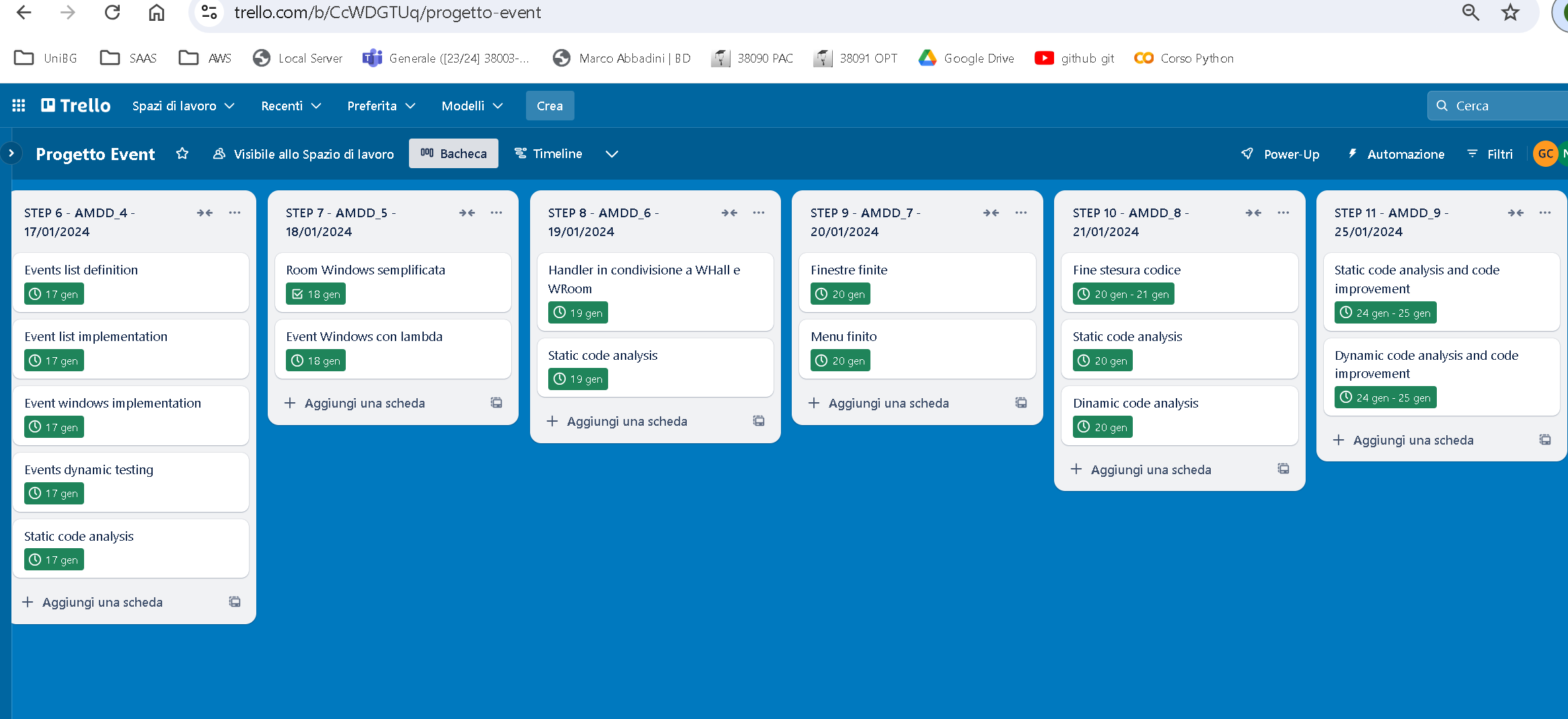
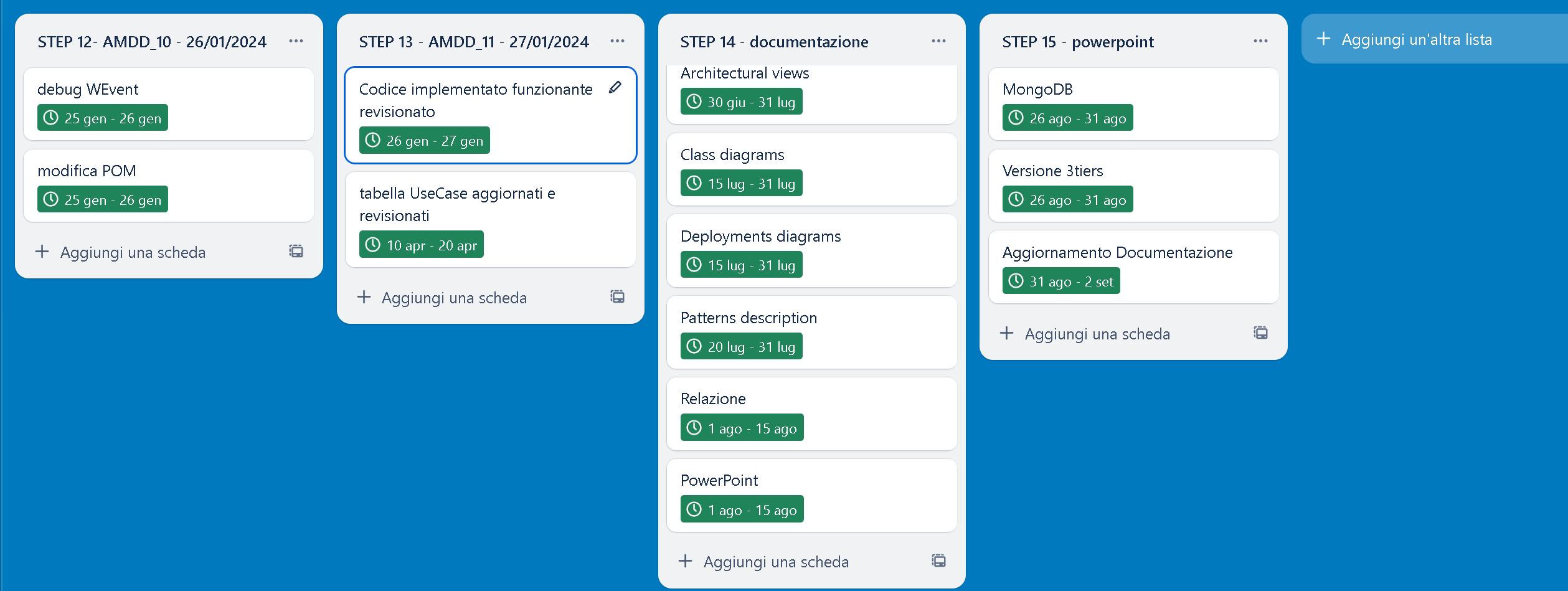
## Modello Organizzativo adottato

Il team ha adottato l’AMDD (Agile Model Driven Development) ed ha messo in pratica le seguenti best practices:

* Planning game: a ciascun Use Case Storie è stata assegnata una priorità (vedi § 3.5 Requirements Analysis)
* Pair Programming: tutto il progetto è stato sviluppato in coppia
* System Metaphor: entrambi i membri del gruppo hanno contribuito a tutte le fasi del progetto
* SCRUM: è stato estremamente utile in quanto il progetto è stato sviluppato in singoli sprint (più o meno settimanali) ma a volte intervallati da lunghe pause.
* Continuous Integration (ogni funzionalità è stata sviluppata “a sé stante”, a cui è seguito lo unit testing (vedi §6.2 Analisi dinamica: JUnit5) e l’immediata integrazione con il resto del codice. Vista la natura accademica del progetto, non c’è stato un vero e proprio delivery.

Alla pagina web <https://trello.com/b/CcWDGTUq/progetto-event> è disponibile la suddivisione in sprint del progetto



Il limitato tempo dedicato alla progettazione (in linea con l’approccio AMDD) unitamente all’inesperienza ci ha portato a sviluppare un codice che solo l’analisi statica ci ha evidenziato deficitario dal punto di vista della coesione (vedi § 6.1 Analisi statica: codeMR); siamo stati costretti ad un pesante re-factoring del codice del client per ridurre le dimensioni e migliorare la coesione di alcune classi. Il re-factoring del codice del server (controller e service) ha ridotto il numero di classi esterne ed ha migliorato la scalabilità del codice.

In extremis abbiamo modificato il codice al fine di sostituire i file json con un DB in cloud (Atlas), questo si può considerare un extra sprint ovvero il rilascio della versione 2 del progetto.

# ITERAZIONE 0 AMDD (Requirements & Architecture Envisioning)

## Requirements Envisioning

La natura accademica del progetto ha fatto sì che alcuni dei requisiti non funzionali siano stati imposti dalla Professoressa Scandurra anziché dal Cliente, ciò ha dato un tocco di realismo al progetto: i vincoli sono sempre imposti e mai scelti.

### Nonfunctional requirements

* Adozione dell’architectural pattern MVC (three layers) ma non necessariamente three tiers: per semplicità abbiamo optato per il MVC passivo/pull (tutte le comunicazioni avvengono attraverso il controller) su due tiers (Client e Server)
* La comunicazione fra tiers deve utilizzare il framework SpringBoot
* L’interfaccia utente deve usare il framework JavaFX
* Approccio API-led
* Utilizzo del maggior numero possibile di design patterns (in particolare abbiamo usato: il Singleton, il Visitor, il Publisher-Subscriber)
* La prima versione del progetto prevedeva che non si utilizzasse alcun DBMS considerando sufficiente l’utilizzo dei file JSON. Solo dopo aver completato la documentazione abbiamo pensato di utilizzare anche MongoDB come DBMS (sia in locale che in cloud).

### Functional requirements (Customer’s requirements)

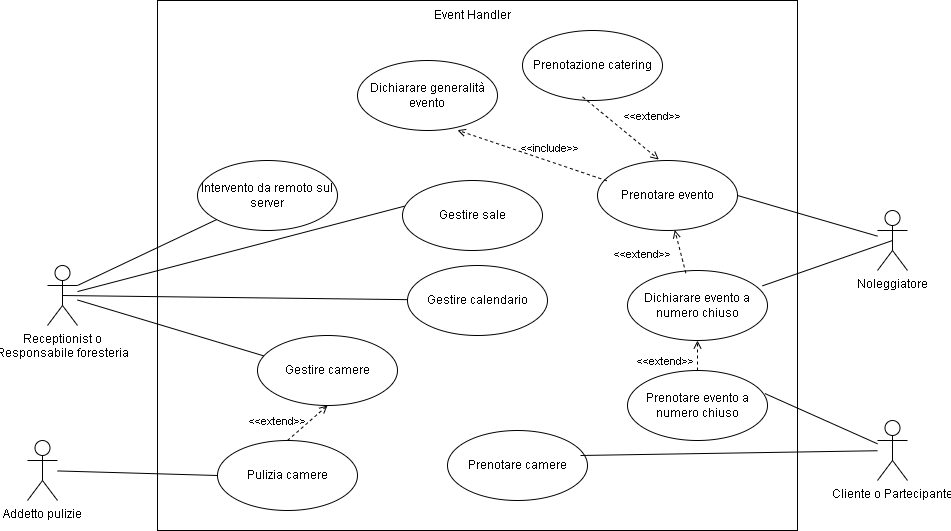
Realizzare un progetto che consenta di gestire una foresteria in cui:

* Il NOLEGGIATORE possa prenotare una sala convegni (di dimensione adeguati al numero di partecipanti previsti), scegliere fra diversi tipi di rinfresco, specificare l’elenco degli interventi (speech), il prezzo di partecipazione, ecc. I convegni durano un giorno solo.
* Il PARTECIPANTE possa prenotare la propria partecipazione e, se desidera, prenotare una camera per il pernottamento. Le camere non sono tutte uguali
* Il RESPONSABILE della FORESTERIA deve poter vedere l’impegno delle sale su una vista di tipo calendario
* L’assegnazione della camera deve essere tale che il CLIENTE possa rimanere nella foresteria per più notti senza dover cambiare camera.
* L’ADDETTO ALLA PULIZIA deve poter cambiare lo stato della camera per indicare che è stata pulita, o che non è disponibile perché la sta pulendo.

## Architecture Envisioning

Partendo dai requisiti del Cliente abbiamo definito l’ Use Case Diagram e un primo Deployment Diagram (realizzato in free style)

### Use Case Diagram

****

### Informal Deployment Diagram

Immagine che contiene schermata, diagramma, testo

Descrizione generata automaticamente

# ITERAZIONE 1 AMDD (Early Architecture Design)

In questa fase si è definita l’architettura software da implementare: si è optata la three layers architecture (ovvero architettura a tre strati), ovvero un'architettura che organizza le applicazioni in tre strati di codice: il layer presentazione, o interfaccia utente, il layer applicazione, dove i dati vengono elaborati, e il layer dati, dove i dati associati all'applicazione vengono memorizzati e gestiti.

Per semplicità abbiamo optato per la soluzione two tiers (Client – Server) con il server che ospita sia il codice del controller che quello per la memorizzazione dei dati (in file JSON). La seconda versione del progetto è invece three tiers poiché la memorizzazione dei dati è stata sposta in cloud (Atlas DB),

Il Client e il Server comunicano mediante il framework Spring Boot che abbiamo utilizzato la tecnologia REST, le API sono rigorosamente RESTful in quanto:

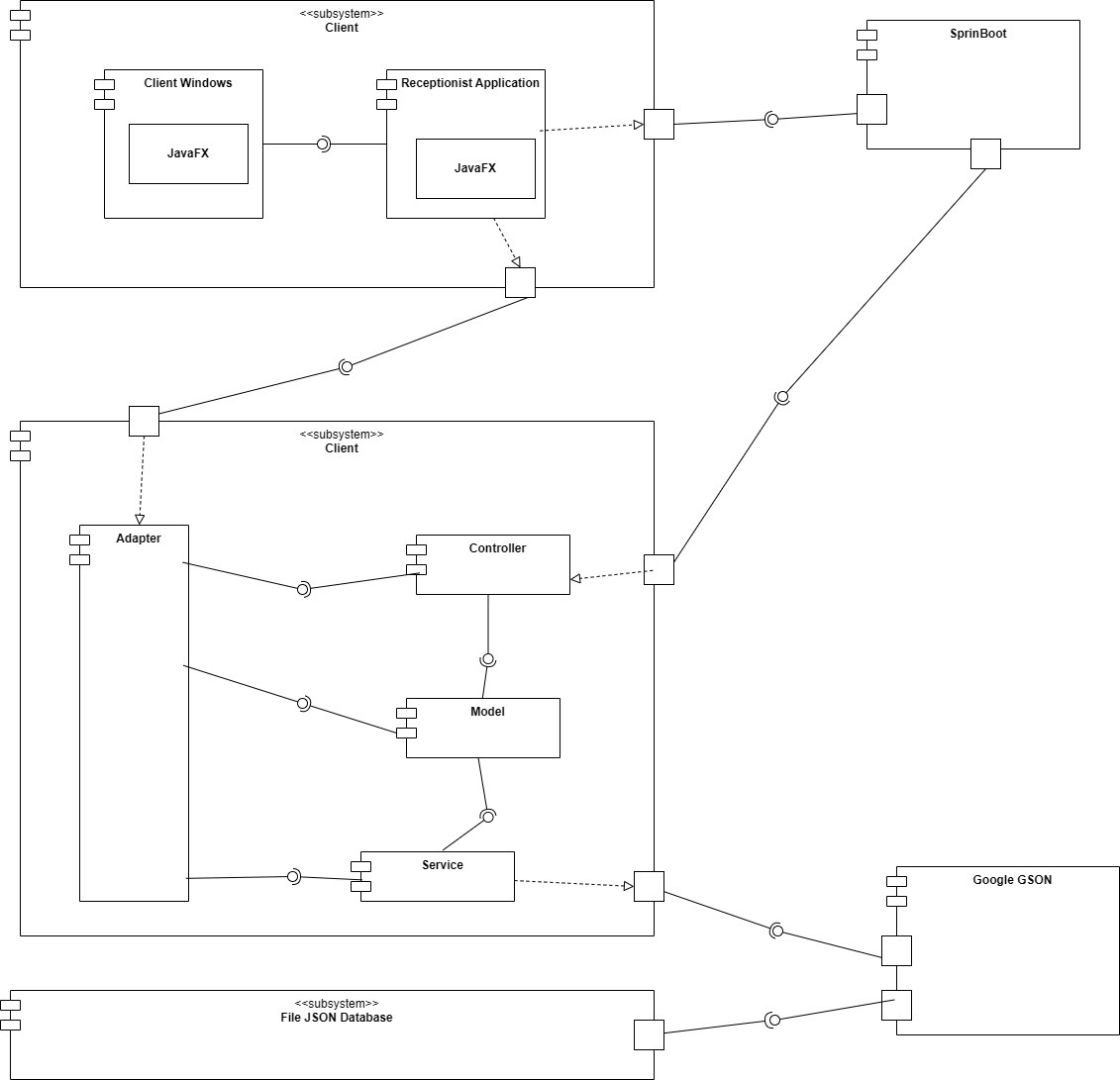
* tutte le richieste sono gestite tramite HTTP.
* le comunicazioni client server sono stateless (non prevede la memorizzazione delle informazioni del client tra le richieste Get) in altre parole ogni richiesta è distinta e non connessa.
* sono trasmessi sempre risorse (Camera, Evento, ecc.) “complete” che sono memorizzati nel client al fine di limitare le interazioni client server.
* l'interfaccia dei componenti è uniforme in modo che le informazioni vengano trasferite in una forma standard

## Deployment View

Immagine che contiene diagramma, testo, Disegno tecnico, Piano

Descrizione generata automaticamente

## Functionality View



**1° versione: File JSON Database**

**2° versione: Atlas Database**

**1° versione: Google GSON**

**2° versione: MongoDB**

## Activity Diagram

Attraverso l’activity diagram possiamo vedere il flusso di dati delle funzioni rese disponibili dal sistema.

Immagine che contiene testo, diagramma, linea, Diagramma

Descrizione generata automaticamente

## Use Case Stories

L’analisi dei requisiti funzionali è stata svolta partendo dall’ Use Case Diagram: ciò ci ha portato a capire quali possibili usi di un software fossero utili in una vera foresteria, immedesimandoci nei vari attori che sono stati individuati:

* il Responsabile della Foresteria e/o la Receptionist (la distinzione è solo a livello di login, l’unica funzionalità precipua del responsabile della foresteria è la definizione del calendario di apertura della foresteria)
* l’Addetto alla Pulizia delle camere e delle sale (è possibile prevedere nuove risorse, quali gli impianti audio-video, per il quale l’addetto della pulizia equivarrebbe al responsabile della manutenzione)
* l’organizzatore dell’evento (Noleggiatore)
* la persona che vuole assistere ad un evento a numero chiuso (Partecipante)
* la persona che vuole prenotare una camera (Cliente)

Quanto sopra ci ha consentito di definire le seguenti Use Case Stories

### Responsabile della foresteria e/o Receptionist

**GESTIRE CALENDARIO**

* Come Responsabile della foresteria voglio poter modificare a piacimento il calendario di apertura della foresteria

**GESTIRE SALE**

* Come Responsabile della foresteria e/o come Receptionist voglio poter accedere alla visualizzazione delle sale conferenza e poter modificare la loro disponibilità

**GESTIRE CAMERE**

* Come Responsabile della foresteria e/o come Receptionist voglio poter accedere alla visualizzazione delle camere e poter modificare la loro disponibilità

### Addetto Pulizia

**GESTIRE CAMERE**

* Voglio avere accesso alla lista di camere e cambiare gli stati “in pulizia” e “disponibile”.

### Noleggiatore

**PRENOTAZIONE EVENTO**

* Voglio poter accedere alla visualizzazione della disponibilità delle sale conferenze in una determinata data
* Nella data da me scelta voglio poter prenotare, se libera, la sala conferenza per un mio evento

**DICHIARARE GENERALITÁ EVENTO**

* Voglio poter dichiarare, dopo averlo prenotato, che tipo di evento voglio ospitare dichiarando il nome dell’evento, il main topic, il relatore e una breve gli speech che lo caratterizzeranno.

**DICHIARARE EVENTO NUMERO CHIUSO**

* Voglio poter avere la possibilità di effettuare il mio evento come evento a numero chiuso (ovvero richiede il pagamento della relativa quota di iscrizione)

**PRENOTAZIONE CATERING**

* Voglio avere la possibilità di prenotare un servizio catering che serva un pasto da me scelto (colazione, brunch, pranzo o cena)

### Partecipante

**PRENOTAZIONE PARTECIPAZIONE**

* Voglio avere accesso alla lista degli eventi, visualizzarne i dati ed iscrivermi fornendo i dati del pagamento della quota di iscrizione.

### Cliente

**PRENOTAZIONE CAMERA**

* Voglio avere accesso alla lista di camere disponibili alla foresteria in una determinata data e poterne prenotare una se libera

## Use Case Stories (fully dressed description)

Per ogni Use Case, è stata redatta una descrizione completa (fully dressed description) di ogni possibile scenario coinvolto. Ogni scenario descrive una storia di interazione con il sistema, sia essa con esito positivo o negativo. In quest’ultimo caso, viene specificata anche una possibile risoluzione. Gli scenari sono stati raffinati di volta in volta con il procedere delle diverse iterazioni. Le fully dressed description di ogni use case sono presentate nelle figure delle pagine seguenti.

|  |  |  |
| --- | --- | --- |
| **Codice** | RF01 | |
| **Nome** | Gestire sale | |
| **Attori** | Responsabile foresteria | |
| **Goals** | Aggiungere sale prenotabili in delle precise date | |
| **Precondizioni** | Selezione delle date in cui ciascuna aula è disponibile alla prenotazione | |
| **Breve descrizione** | Il responsabile aggiunge le sale che potranno essere prenotate per degli eventi | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
| Responsabile della foresteria seleziona le aule disponibili all’esecuzione di un evento e assegna le date in cui esse sono prenotabili | Sale con rispettive date in cui sono prenotate memorizzate |
| **Casi d’uso correlati e processi alternativi** |  | |
| **Post condizioni** | Aggiornamento delle date in cui le sale sono prenotabili | |

Nelle iterazioni successive ci siamo resi conto che il numero delle camere e delle sale cambiano solo in occasione di importanti ristrutturazioni della foresteria e che la gestione dell’anagrafica di tali risorse può essere confinata alla fase di configurazione del sistema e quindi non abbiamo sviluppato tale codice: tutte le sale risultano prenotabili in base al calendario di apertura della foresteria.

|  |  |  |
| --- | --- | --- |
| **Codice** | RF02 | |
| **Nome** | Gestire calendario | |
| **Attori** | Responsabile foresteria | |
| **Goals** | Aggiornare i giorni di chiusura della foresteria | |
| **Precondizioni** | Nessuna | |
| **Breve descrizione** | Il responsabile della foresteria aggiorna il calendario di apertura/chiusura della foresteria a seconda delle sue necessità | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
| Responsabile della foresteria seleziona le date del calendario che vuole aggiornare decidendo se la foresteria nella data in questione è chiusa o aperta. | Il sistema aggiorna le date di chiusura nel calendario |
| **Casi d’uso correlati e processi alternativi** |  | |
| **Post condizioni** | Calendario con date di chiusura scelte aggiornato.  Eventuali eventi o camere prenotati per date che erano precedentemente dichiarate di apertura vengono annullate. Il sistema deve produrre un elenco delle prenotazioni annullate affinché si possa avvisare la clientela. | |

|  |  |  |
| --- | --- | --- |
| **Codice** | RF03 | |
| **Nome** | Gestire camere | |
| **Attori** | Responsabile foresteria / Receptionist | |
| **Goals** | Aggiornare le prenotazioni delle camere della foresteria | |
| **Precondizioni** | Aver aggiornato calendario | |
| **Breve descrizione** | Il responsabile aggiorna i giorni in cui ogni camera è stata prenotata da un cliente, in fase di pulizia, o disponibile alla prenotazione | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
| Responsabile della foresteria seleziona le camere ne cambia lo stato (per singola data) | Il sistema cambia lo stato camera per singola data, ne cambia la visualizzazione e notifica l’addetto alle pulizie |
| **Casi d’uso correlati e processi alternativi** |  | |
| **Post condizioni** | Camere nel nuovo stato | |

|  |  |  |
| --- | --- | --- |
| **Codice** | RF04 | |
| **Nome** | Intervento da remoto sul server | |
| **Attori** | Responsabile foresteria / Receptionist | |
| **Goals** | Intervenire da remoto sul server | |
| **Precondizioni** | Aver aggiornato calendario | |
| **Breve descrizione** | Il responsabile interviene da remoto sul server, per esempio per disporne lo spegnimento | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
| Responsabile della foresteria, da remoto compie un’azione sul server | Il sistema effettua l’azione voluta |
| **Casi d’uso correlati e processi alternativi** |  | |
| **Post condizioni** |  | |

|  |  |  |
| --- | --- | --- |
| **Codice** | AP01 | |
| **Nome** | Pulizia camere | |
| **Attori** | Addetto pulizie | |
| **Goals** | Poter effettuare la pulizia delle camere desiderate | |
| **Precondizioni** | Aver aggiornato calendario | |
| **Breve descrizione** | Dopo che il responsabile della foresteria o receptionist aggiorna i giorni in cui ogni camera deve essere in fase di pulizia, l’addetto alle pulizie effettua tale servizio nelle date indicate | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
| Dopo aver ricevuto la notifica data da “Gestire camere”, l’addetto alle pulizie pulisce le camere nelle date indicate dal sistema |  |
| **Casi d’uso correlati e processi alternativi** | Estende Gestione Camere | |
| **Post condizioni** |  | |

|  |  |  |
| --- | --- | --- |
| **Codice** | N01 | |
| **Nome** | Prenotazione Evento | |
| **Attori** | Noleggiatore | |
| **Goals** | Prenotare una sala della foresteria per ospitare un evento in una determinata data | |
| **Precondizioni** | Sale messe a disposizione dal responsabile della foresteria | |
| **Breve descrizione** | Il noleggiatore apre dall’applicazione la finestra per effettuare la prenotazione dello spazio apposito della foresteria adibito a ospitare un evento in date disponibili | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
| Il noleggiatore clicca sul menu dell’app la voce prenota “evento”. Seleziona le sale che lo aggradano per costo e capienza e seleziona le date in cui prenotarle. | Sale prenotate in memoria nelle date indicate. |
| **Casi d’uso correlati e processi alternativi** | Non ci sono sale libere in nessuna data disponibile. Il noleggiatore non può prenotare nessuna sala per l’evento. | |
| **Post condizioni** | L’evento è salvato in memoria con i suoi dati. | |

Come sopra indicato, la precondizione di N01 è stata successivamente cambiata in “Aver aggiornato calendario”.

|  |  |  |
| --- | --- | --- |
| **Codice** | N02 | |
| **Nome** | Definire generalità evento | |
| **Attori** | Noleggiatore | |
| **Goals** | Dichiarare i dati generali dell’evento | |
| **Precondizioni** | Aprire la finestra degli eventi | |
| **Breve descrizione** | Dichiarare titolo, nome del relatore, descrizione, ora inizio, ora fine, nome dell’organizzatore e costo di partecipazione dell’evento da effettuare nelle date in cui sono state prenotate le sale alla foresteria. | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
| Il noleggiatore inserisce negli appositi campi i dati dell’evento. | Il sistema registra i dati dell’evento. |
| **Casi d’uso correlati e processi alternativi** | Incluso in prenotazione evento. | |
| **Post condizioni** | L’evento modificato è salvato in memoria con i suoi dati. | |

|  |  |  |
| --- | --- | --- |
| **Codice** | N03 | |
| **Nome** | Dichiarare evento a numero chiuso | |
| **Attori** | Noleggiatore | |
| **Goals** | Dichiarare l’evento a numero chiuso. | |
| **Precondizioni** | Aprire la finestra degli eventi | |
| **Breve descrizione** | Il noleggiatore nella finestra della prenotazione dichiara l’evento come evento a numero chiuso (definisce una quota di partecipazione diversa da zero) | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
|  | Il noleggiatore inserisce nell’apposito campo l’opzione di evento a numero chiuso. | Il sistema registra l’evento come evento a numero chiuso. |
| **Casi d’uso correlati e processi alternativi** | Estensione di “Dichiarare generalità evento”. | |
| **Post condizioni** | L’evento modificato è salvato in memoria con i suoi dati. | |

|  |  |  |
| --- | --- | --- |
| **Codice** | N04 | |
| **Nome** | Prenotazione catering | |
| **Attori** | Noleggiatore | |
| **Goals** | Noleggiare un servizio catering per l’evento. | |
| **Precondizioni** | Aprire la finestra degli eventi | |
| **Breve descrizione** | Il noleggiatore nella finestra delle prenotazioni dichiara di volere un servizio catering durante l’evento con quale pasto servire gli ospiti (coffee break, brunch, pranzo, buffet, …). | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
|  | Il noleggiatore inserisce nell’apposito campo l’opzione di avere un servizio catering durante l’evento dichiarando per quale pasto. | Il sistema registra l’avere un servizio catering durante l’evento. |
| **Casi d’uso correlati e processi alternativi** | Estensione di “Dichiarare generalità evento”. | |
| **Post condizioni** | L’evento presenta in memoria il servizio catering con i pasti scelti. | |

|  |  |  |
| --- | --- | --- |
| **Codice** | P01 | |
| **Nome** | Prenotazione camera | |
| **Attori** | Cliente | |
| **Goals** | Prenotare una stanza all’albergo della foresteria | |
| **Precondizioni** | Aver aggiornato calendario  Aprire la finestra delle camere | |
| **Breve descrizione** | Il partecipante effettua la prenotazione di una stanza per pernottare alla foresteria | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
|  | Il partecipante fa accesso all’applicazione e prenota una stanza in data | Una camera dell’albergo viene prenotata, se ce n’è almeno una libera |
| **Casi d’uso correlati e processi alternativi** | Processo alternativo: non ci sono camere libere, il partecipante non può prenotare la camera | |
| **Post condizioni** | Camera prenotata dal partecipante | |

|  |  |  |
| --- | --- | --- |
| **Codice** | P02 | |
| **Nome** | Prenotazione evento numero chiuso | |
| **Attori** | Partecipante | |
| **Goals** | Prenotare un evento a numero chiuso | |
| **Precondizioni** | A calendario vi sia un evento a numero chiuso creato da un noleggiatore | |
| **Breve descrizione** | Il partecipante effettua la prenotazione di evento a numero chiuso che si ospiterà nella foresteria in una determinata data | |
| **Processo** | **Azioni degli attori** | **Risposta del sistema** |
|  | Il partecipante fa accesso all’applicazione e prenota un evento a numero chiuso | Il cliente risulta prenotato all’evento a numero chiuso da lui selezionato |
| **Casi d’uso correlati e processi alternativi** | Processo alternativo: l’evento è full booked, il partecipante è escluso | |
| **Post condizioni** | Il partecipante è prenotato all’evento | |

## Requirements Analysis

Ad ogni specifica funzionale è stata assegnata una priorità definita in base all’interdipendenza con le altre specifiche:

* **Priorità alta:** specifiche che realizzano gli aspetti vitali del sistema e, quindi, da implementare prima di quelle con bassa priorità;
* **Priorità bassa**: specifiche marginali da cui non dipendono altre specifiche.

La seguente tabella riepiloga le specifiche funzionali e indica se è stata implementata.

| **Codice Specifica** | **Descrizione** | **Priorità** | **Implementato** | **Codice User Case** |
| --- | --- | --- | --- | --- |
| **R01** | **Gestire Sale:** l’applicazione deve consentire di aggiungere sale prenotabili | **Alta** | **No** | **RF01** |
| **R02** | **Calendario di apertura della foresteria**: l’applicazione deve permettere la visualizzazione del calendario in cui la foresteria è aperta. | **Alta** | **Si** | **RF02** |
| **R03** | **Lista sale conferenza**: l’applicazione deve permettere la visualizzazione della lista delle sale per le conferenze della foresteria. Essa deve mostrare inoltre quali sale sono disponibili in una data selezionata e quanti posti ha una singola sala. | **Alta** | **Si** | **RF01** |
| **R04** | **Lista camere:** l’applicazione deve permettere la visualizzazione della lista delle camere della foresteria. Essa deve mostrare inoltre quali camere sono disponibili in una data selezionata. | **Alta** | **Si** | **RF03**  **P01**  **AP01** |
| **R05** | **Intervento da remoto sul server:** l’applicazione deve permettere funzioni volute dal responsabile della foresteria da remoto (lo spegnimento) | **Bassa** | **Si** | **RF04** |
| **R06** | **Prenotare evento**: l’applicazione deve poter permettere la creazione di un evento in una sala conferenze compilando dei requisiti minimi come nome dell’evento, il nome del relatore, il topic dell’evento e il numero di partecipanti previsti. | **Alta** | **Si** | **N01, N02** |
| **R07** | **Dichiarare evento a numero chiuso**: una volta creato un evento, si deve dare l’opzione di effettuarlo a numero chiuso. | **Bassa** | **Si** | **N03** |
| **R08** | **Servizio catering:** una volta creato un evento, si deve dare l’opzione di avere un servizio catering specificando quale pasto si deve servire (colazione, pranzo o cena). | **Bassa** | **Si** | **N04** |
| **R09** | **Prenotazione numero chiuso:** un cliente deve avere l’opzione di prenotarsi a un evento a numero chiuso; | **Bassa** | **No** | **P02** |
| **R10** | **Login/Logout**: in fase di login l’applicazione accerta che lo user sia il responsabile della foresteria, un noleggiatore delle sale conferenza o un cliente, così da permettere le funzionalità della singola categoria di utilizzatore. | **Bassa** | **No** | **LOG** |

# ITERAZIONE 2 AMDD

## Tool Chain e Tecnologie

Questa fase iniziale ha comportato anche la scelta dei tool e della tecnologia da utilizzare per la memorizzazione dei dati su database e la tecnologia da utilizzare per l’implementazione del software applicativo.

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Funzione** |
| Windows | Sistema Operativo | SO |
| Java 17 | Linguaggio di programmazione | Language |
| JDK 20 | Development Kit | Ambiente Sviluppo |
| Eclipse 4.29.0 | IDE | Ambiente Sviluppo |
| Maven 4.0.0 | Build Tool | Gestione dipendenze |
| sonarLint 10.5 | Plugin Eclipse | Quality + Security |
| codeMR 2020..4.1 | Plugin Eclipse | Analisi Statica |
| JUnit5 5.10.1 | Libreria di test | Analisi Dinamica |
| JavaDoc | Plugin Eclipse | Strumento di Documentazione |
| SpringBoot 3.2.0 | Framework | Framework per Applicazioni WEB |
| JavaFX | Framework | Componenti UI |
| JSON | Formato scambio dati | Data Base |
| Gson | Libreria | SerDe formato json |
| Git Hub | SAAS | WEB Repository Management |
| MongoDB | Software | DMBS NoSQL in locale |
| AtlasDB | SAAS | DBMS NoSQL in cloud |
| Trello | SAAS | Gestione progetti |
| Teams | SAAS | Call + chat |
| WhatsApp | SAAS | Call + chat |
| Draw.io | SAAS | Grafici e diagrammi UML |
| MS Office | Software (Word e PowerPoint) | Documentazione |

### codeMR

CodeMR consente l’analisi statice del codice e ne visualizza le metriche e gli attributi di qualità (accoppiamento, complessità, coesione e dimensione) in diverse viste come Package Structure, TreeMap, Sunburst, Dependency e Graph Views.

Abbiamo utilizzato una trial licence.

### Gson

Si è utilizzato Gson, libreria sviluppata da Google, per convertire oggetti Java nella loro rappresentazione JSON e viceversa.

* Le caratteristiche principali che ci hanno portato all’utilizzo di Gson sono le seguenti:
* Semplice e veloce da utilizzare.
* Indipendente dai file sorgenti.

Ha consentito la creazione di adapter personalizzati per definire la modalità di serializzazione degli oggetti di tipo LocalDate e LocalTime.

### MongoDB e AtlasDB

Visto che la 1a versione usava file json, abbiamo optato per un DMBS NoSQL; il passaggio da MongoDB (locale) ad AtlasDB (cloud) è stato immediato.

## Implementazione

Lo scopo del progetto è creare un’API che gestisca in tempo reale tutte le informazioni che vengono immesse nel sistema della foresteria, a partire dalla gestione delle camere fino a quella delle sale conferenza e dei loro eventi. L’applicazione permette agli utenti (il responsabile della foresteria, i noleggiatori delle sale conferenza, i partecipanti ai convegni ed ai clienti che vogliono soggiornare alla foresteria) di avere una visione in tempo reale di dati immessi nel DB della foresteria stessa.

### Selezione funzioni da implementare

In questa prima fase di implementazione si è deciso di implementare la funzionalità che gestisce la gestione delle camere.

Immagine che contiene testo, diagramma, linea, Diagramma

Descrizione generata automaticamente

**Receptionist**

### Component Diagram

L’implementazione del caso d’uso Gestire Camere ha dunque portato alla creazione di un listener RoomController con i metodi elencati in figura.

Immagine che contiene testo, schermata, Carattere, schermo

Descrizione generata automaticamente

Tale entità interagisce con RoomService che gestisce il file Camere.json nel quale conserva i dati inerenti a nome, costo, numero letti e disponibilità di tutte le camere della foresteria.

La comunicazione tra la UI e il Controller avviene grazie alla dependencies definita nel framework Spring Boot mentre la comunicazione fra Controller e Service è una semplice chiamata di un metodo.

Si rimanda al § 5.3 Class Diagrams per gli schemi definitivi.

### Analisi complessità metodo: *WRoom.mettiDati*

La versione inziale della classe di visualizzazione delle camere prevedeva il seguente codice con cui si eseguiva una Get a cui il Listener/Controller rispondeva con l’intera lista di camere e la loro disponibilità.

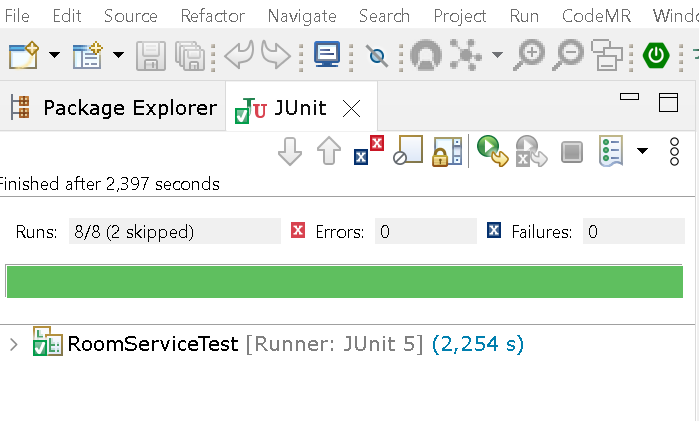
Immagine che contiene testo, schermata, Carattere

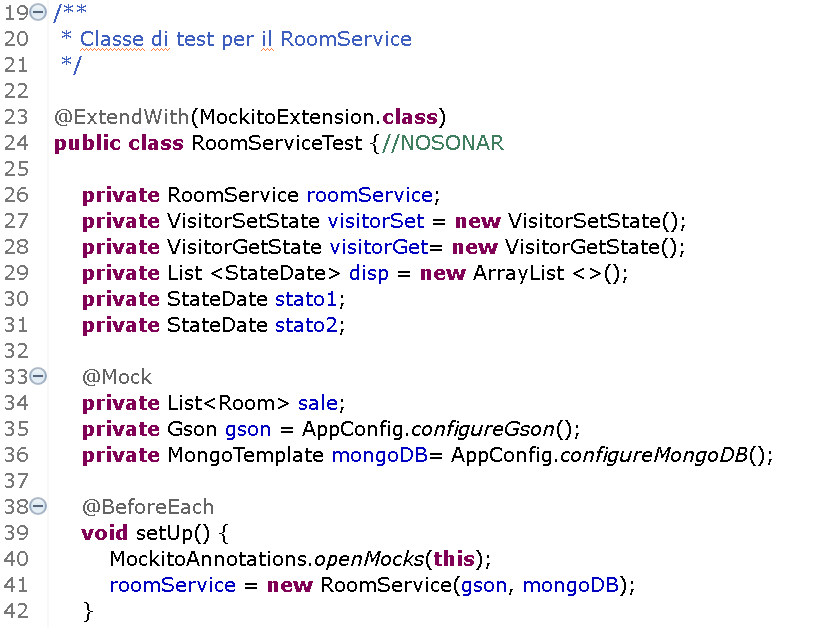
Descrizione generata automaticamente

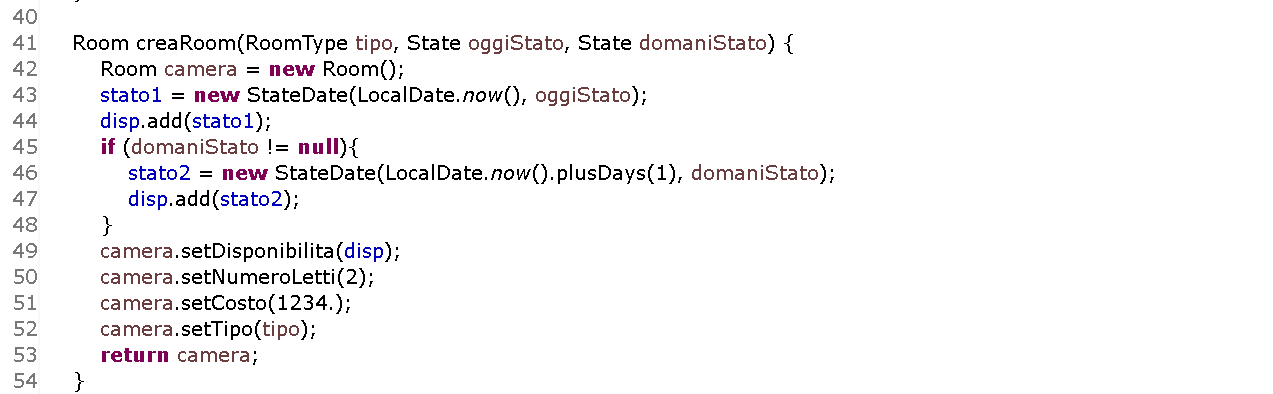
Al fine di creare una visualizzazione tabellare (una data per ciascuna colonna ed una riga per ciascuna camera) il metodo raggruppai dati per singola data. Osservando il codice ci si accorge della presenza di due cicli for annidati e, considerando che il numero delle camere (k) è fisso e molto inferiore al numero delle date (n), la complessità totale del metodo è O(k n) = **O(n)**.

Nelle iterazioni successive, il metodo è stato “generalizzato” ed inserito nella classe WResource.

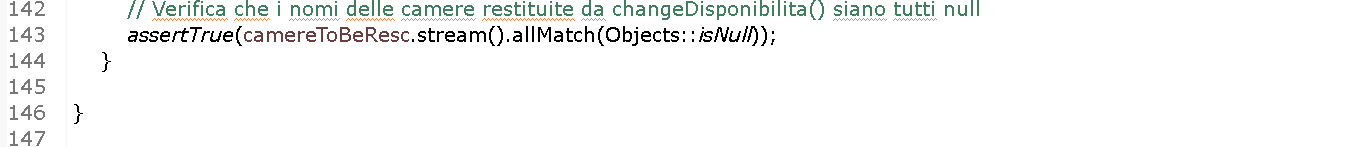
### Analisi dinamica

I seguenti screen shots mostrano il codice della classe per il testing di RoomService l’esito positivo dell’esecuzione di tutti gli 8 test previsti dal codice.



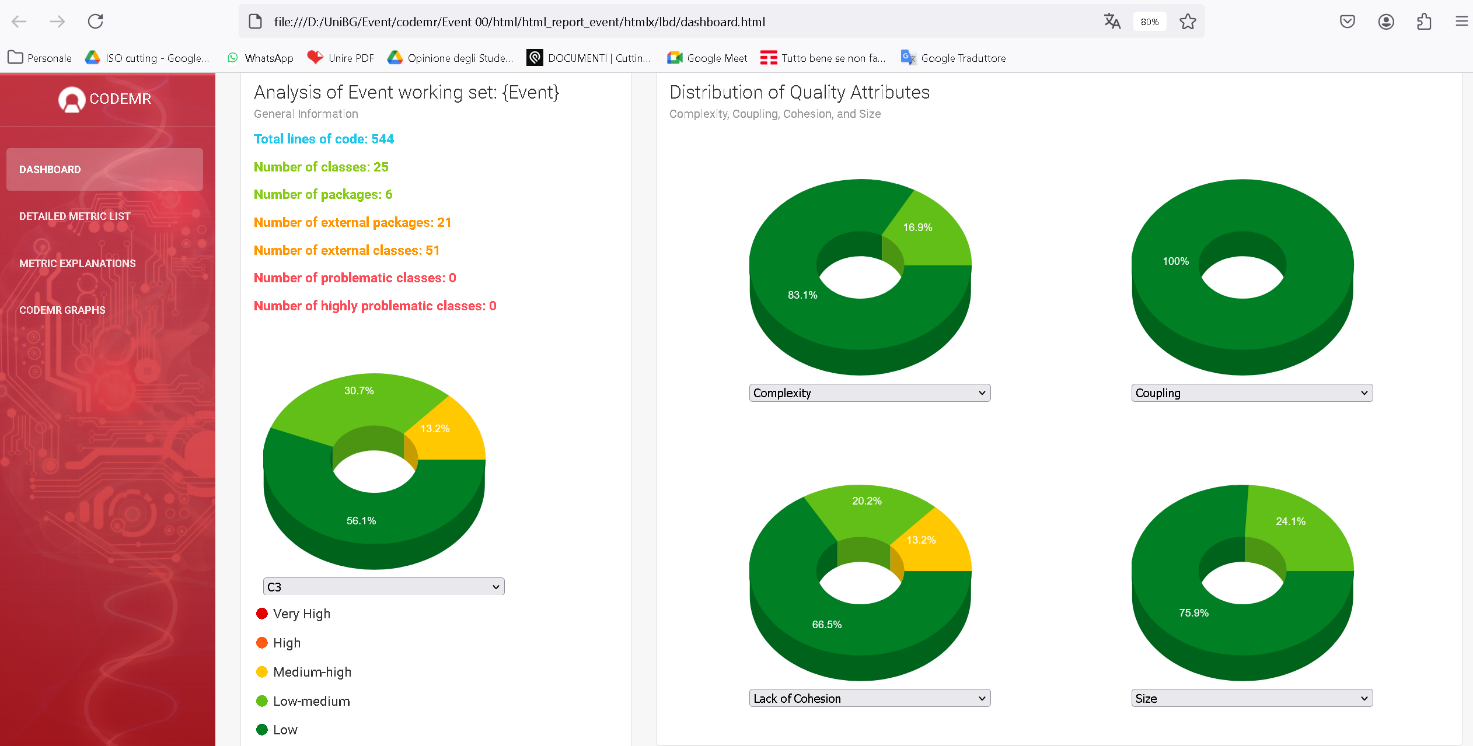
  
  



### Analisi statica

Al termine di questa iterazione (544 righe di codice vs le oltre 1700 della iterazione finale), solo la coesione non era “perfetta”).



### Maschera delle camere

Immagine che contiene testo, numero, schermata, linea

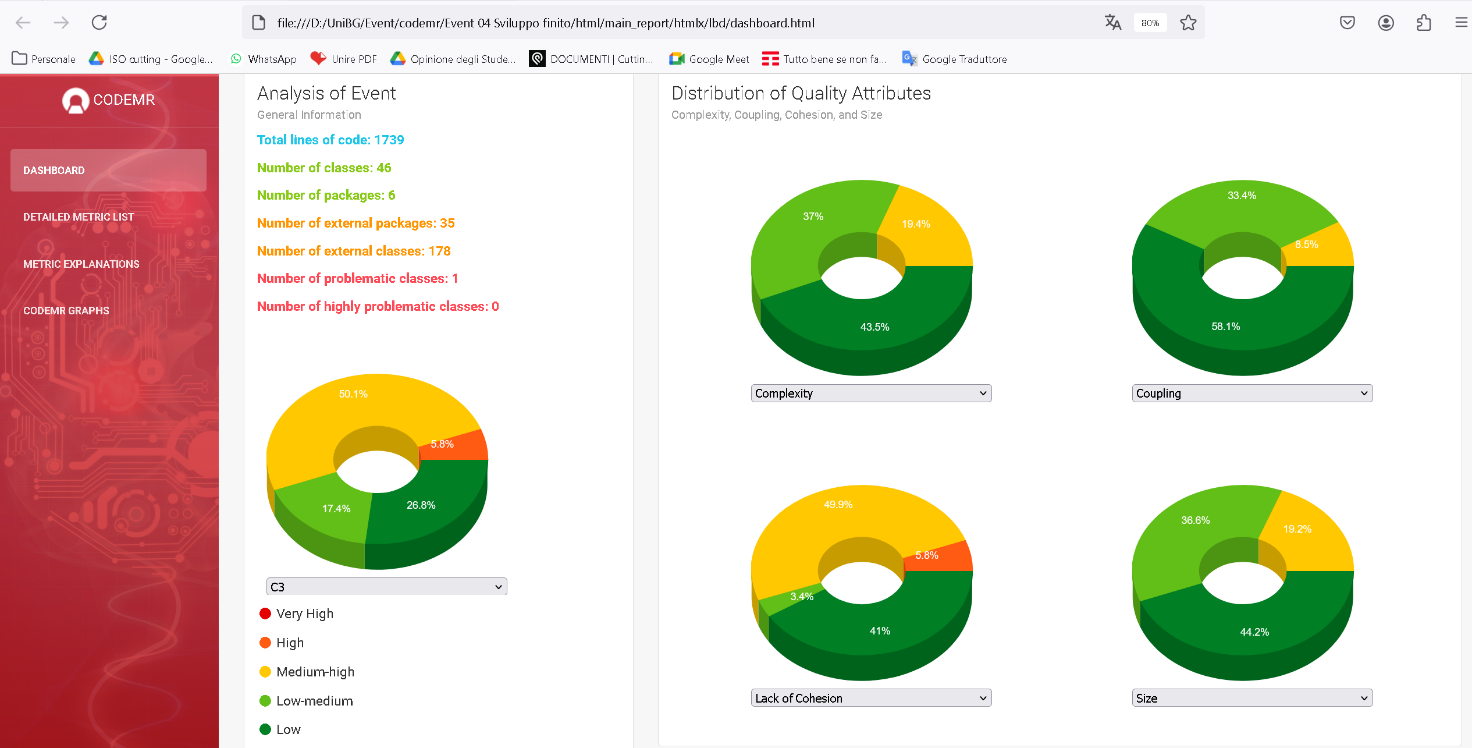
Descrizione generata automaticamente

### Conclusioni iterazione 2

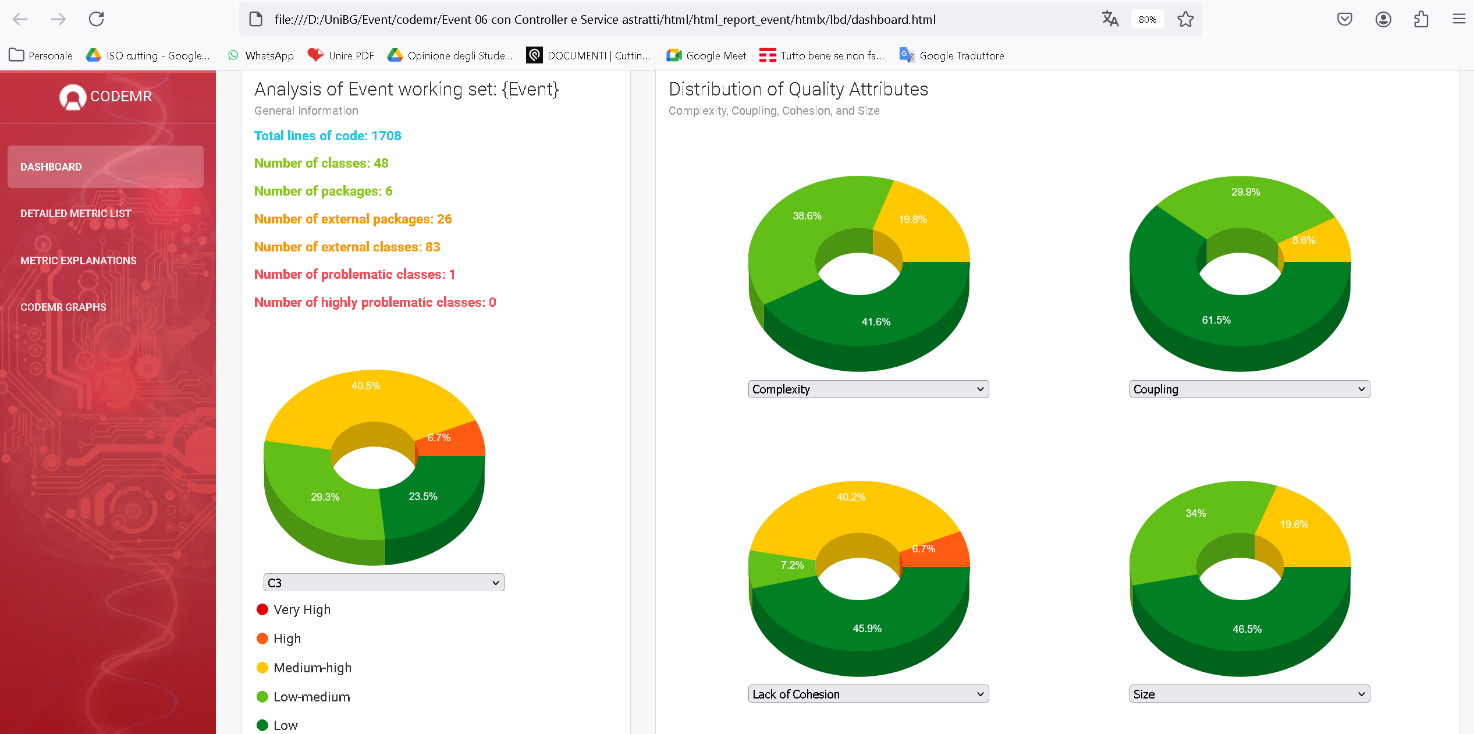
| **Cod. Specifica** | **Descrizione** | **Implementato** | **Cod. User Case** |
| --- | --- | --- | --- |
| **R01** | **Gestire Sale** | **No** | **RF01** |
| **R02** | **Calendario di apertura della foresteria** | **No** | **RF02** |
| **R03** | **Lista sale conferenza** | **No** | **RF01** |
| **R04** | **Lista camere** | **Si** | **RF03**  **P01**  **AP01** |
| **R05** | **Intervento da remoto sul server** | **No** | **RF04** |
| **R06** | **Prenotare evento** | **No** | **N01, N02** |
| **R07** | **Dichiarare evento a numero chiuso** | **No** | **N03** |
| **R08** | **Servizio catering** | **No** | **N04** |
| **R09** | **Prenotazione numero chiuso** | **No** | **P02** |
| **R10** | **Login/Logout** | **No** | **LOG** |

# ITERAZIONI INTERMEDIE

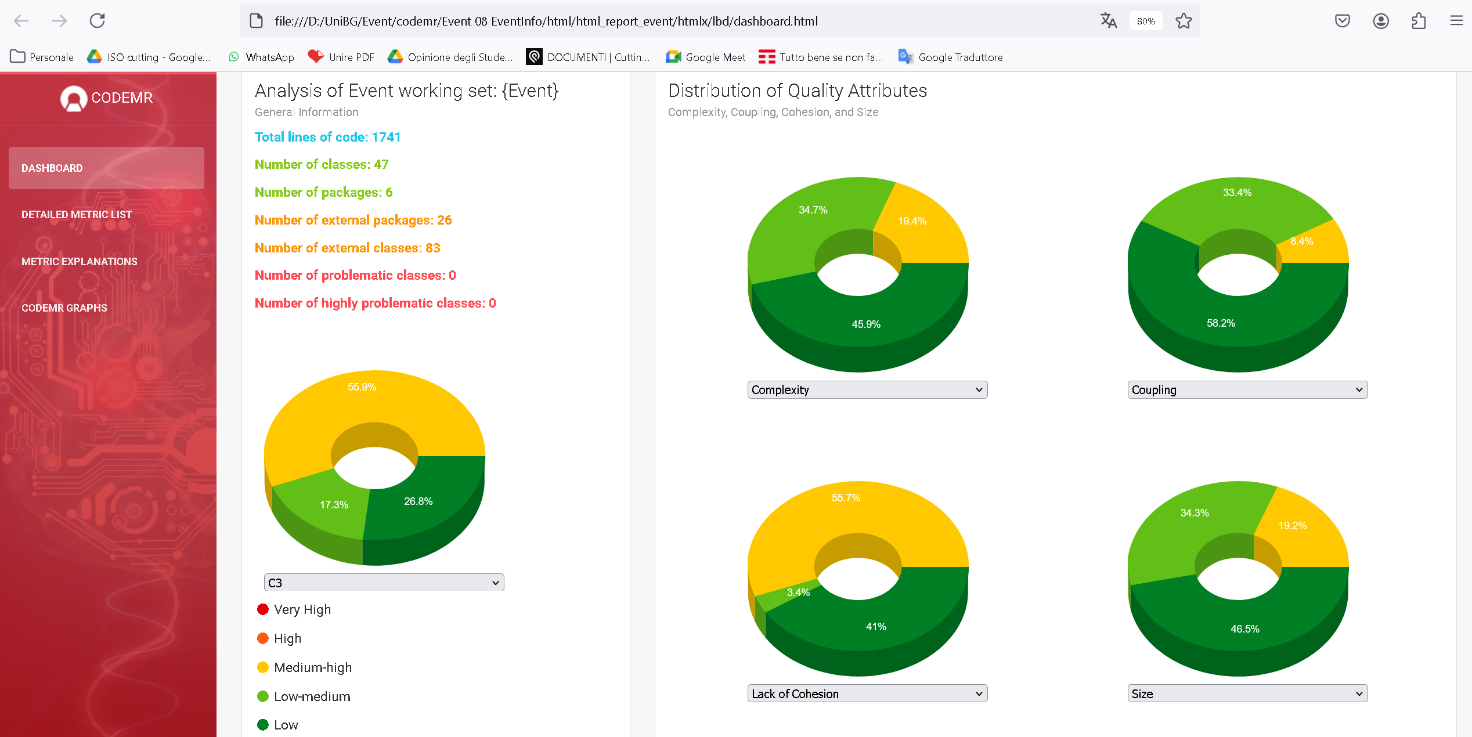
Al termine della iterazione 4 abbiamo completato lo sviluppo degli user case che ci eravamo prefissi, ma l’analisi statice eseguito con codeMR mostrava problemi di coesione di tipo “grave”



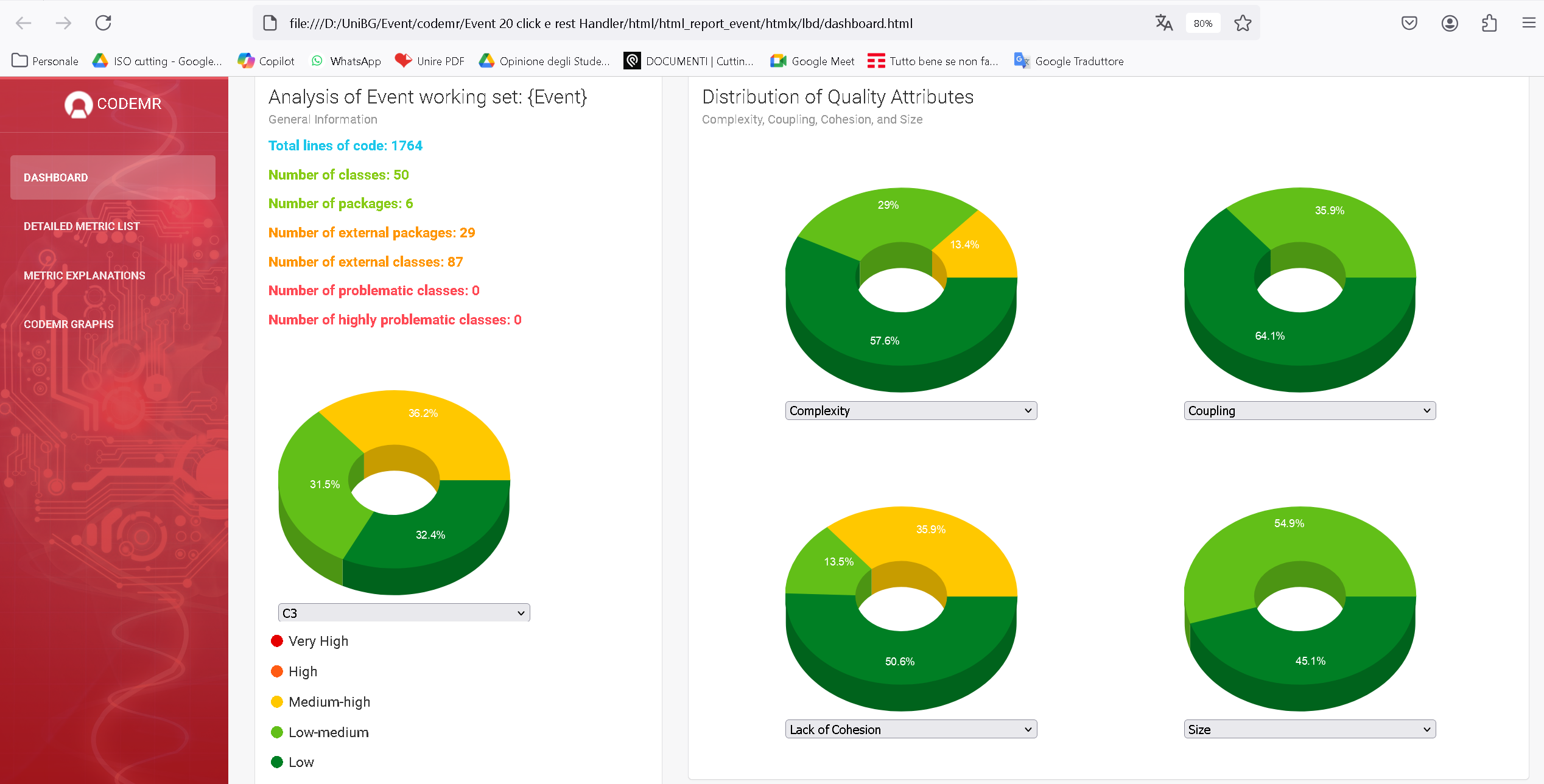
Abbiamo, quindi, cominciato il re-factoring del codice dapprima rivedendo le classi del server introducendo una classe astratta e il pattern observer. Come si può vedere, tale intervento ha praticamente dimezzato il numero di classi esterne e il numero di package esterni, ma non è servito a migliorare la coesione.



Il re-factoring del codice del server, per quanto utile per migliorare la struttura del codice, ha ottenuto il risultato finale solo con l’introduzione della classe WEventInfo nella quale sono confluiti tutti gli attributi di Event si è riuscito ad eliminare i problemi di coesione



L’introduzione della classe WEventLayout nella quale è confluito il codice per la gestione dei numerosi controlli presenti nella maschera WEvent ed altre interventi “minori” hanno consentito di ottenere un coupling ottimale:



# ITERAZIONE FINALE

## Architectural Pattern

Il pattern architetturale delinea gli elementi essenziali e coesi dell'architettura del sistema. Il seguente schema evidenzia:

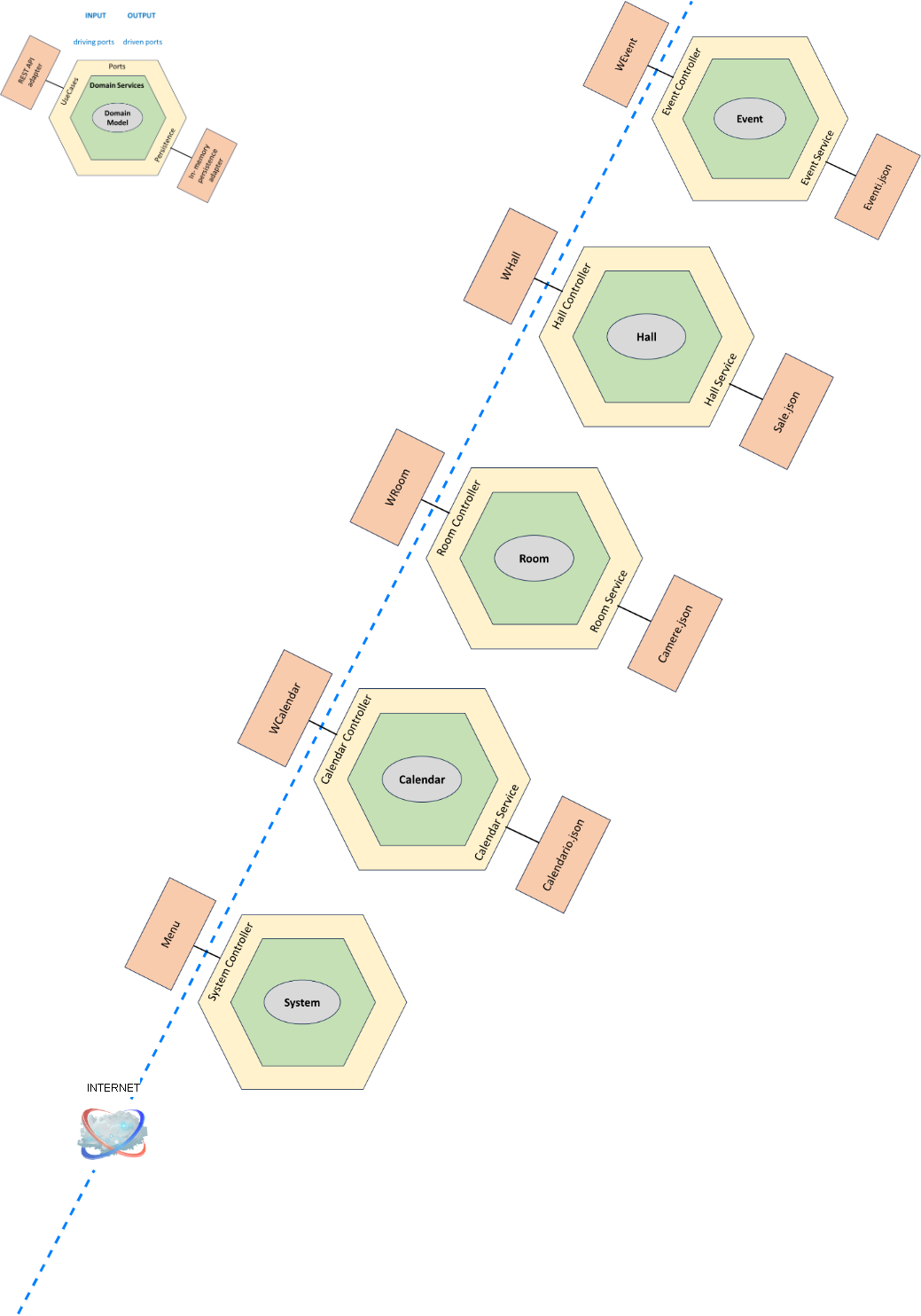
* La struttura 2 tiers (il modello e i dati sono nello stesso tier, il SERVER)
* La comunicazione via internet ad opera di un REST Generator e di un Controller (basati sul framework SpringBoot)
* L’interfaccia utente (realizzata usando JavaFX) è tenuta separata dal codice che gestisce gli eventi (e che esegue le request ed elabora le risposte del server)
* Il server è suddiviso in controller (listener) ed il service che interviene sia sul Model che sul Data Base (i file json)
* L’architectural pattern è palesemente un MVC; poiché tutte le comunicazioni passano attraverso il controller è un MVC passivo (Pull)



## Formal Deployment Diagram

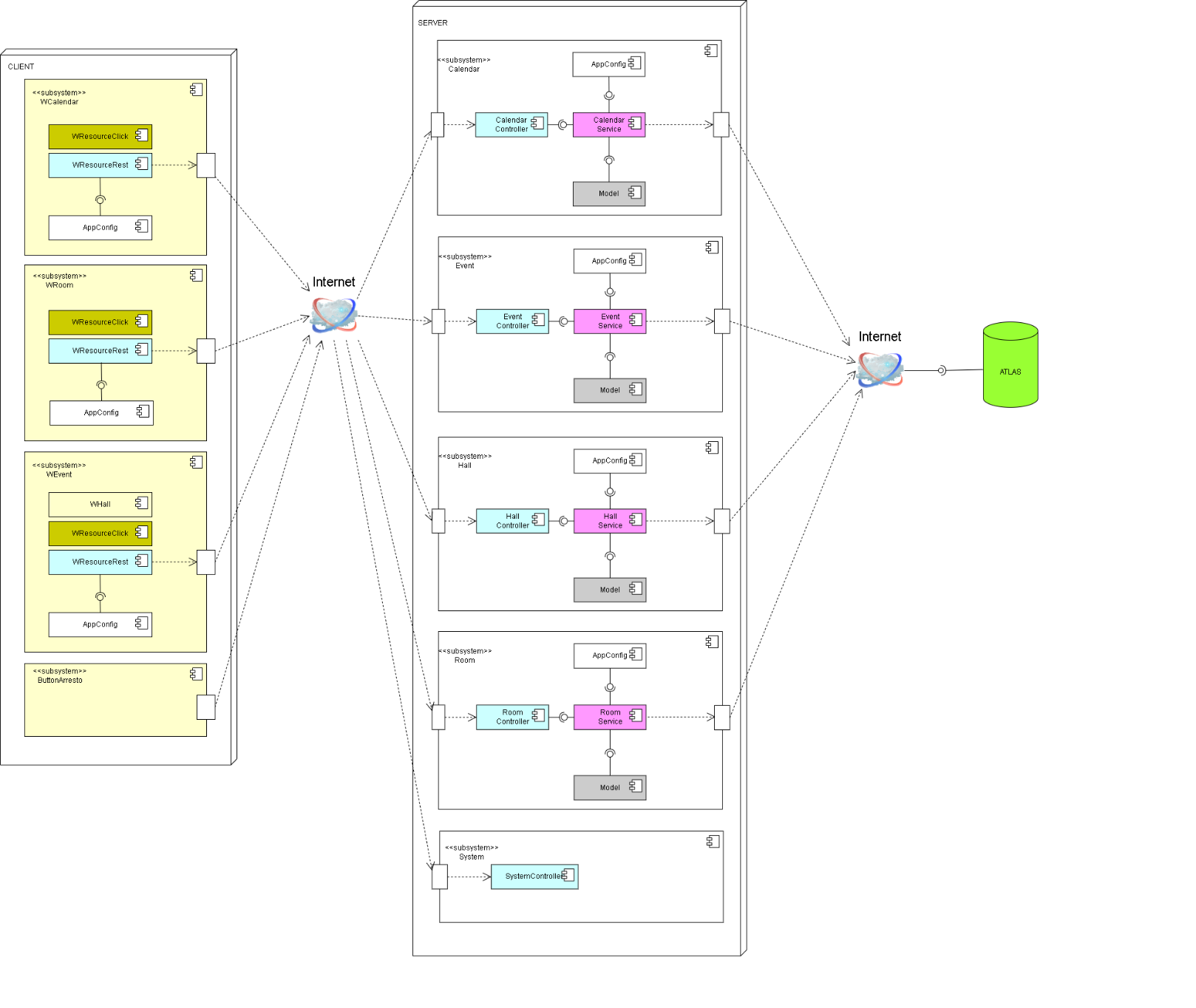
I design pattern utilizzati hanno consentito di rendere il codice estremamente modulare: ciascun modulo è dedicato ad uno specifico tipo di risorsa (Calendar, Room, Hall, Event …) ed ogni modulo rispecchia l’architectural pattern precedentemente illustrato (i singoli elementi hanno lo stesso colore nei due schemi).

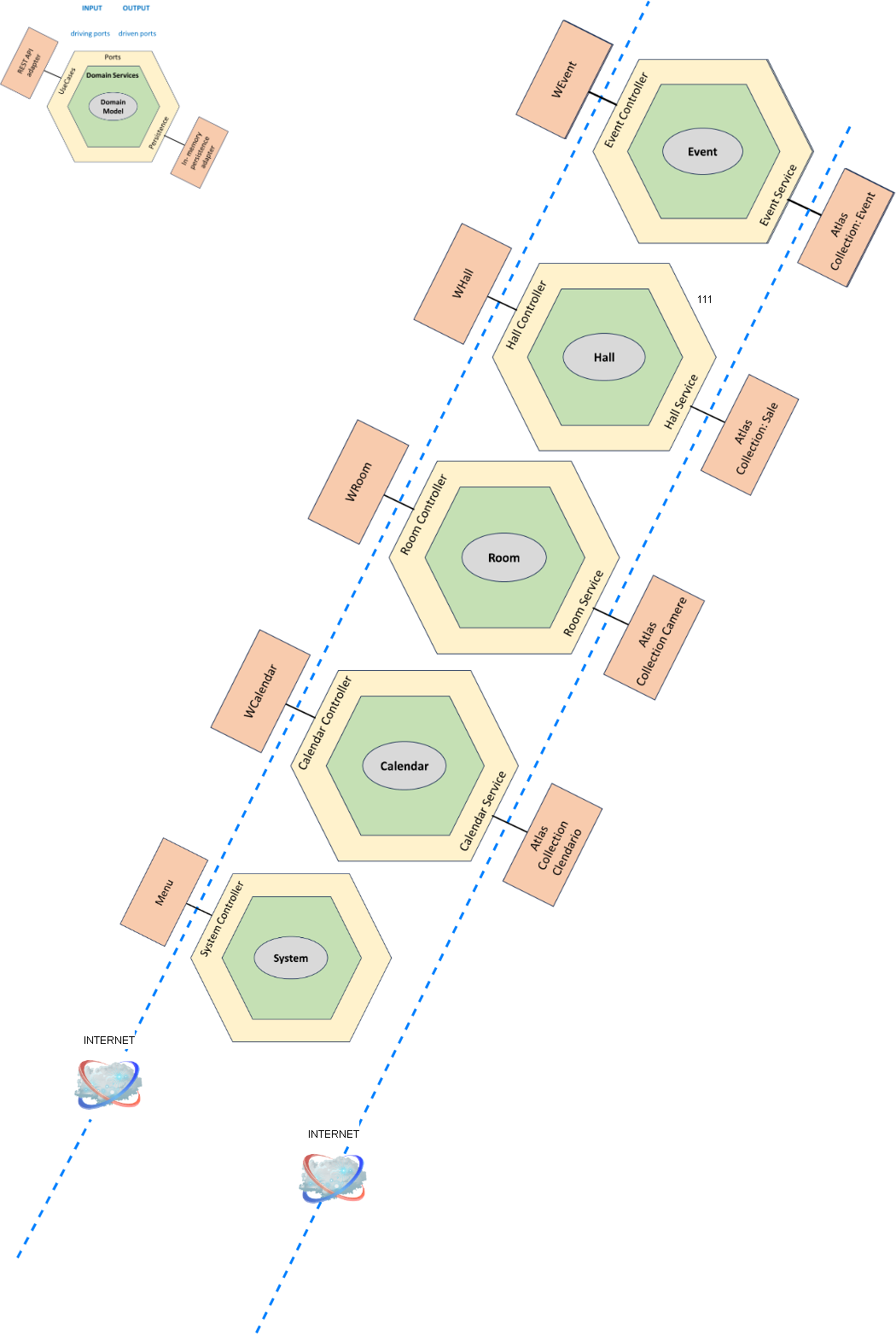
### Immagine che contiene testo, diagramma, schermata, linea Descrizione generata automaticamenteVersione 1 (two tiers)



### Versione 2 (three tiers)

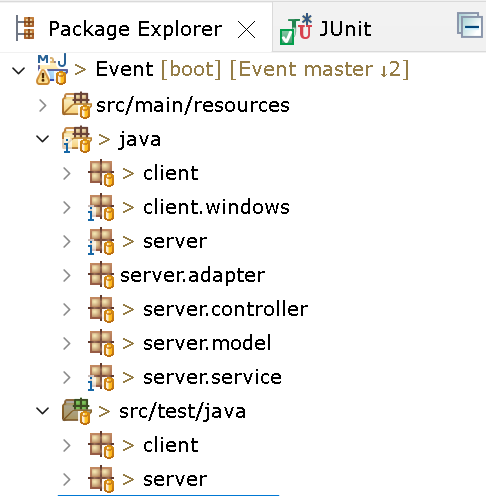
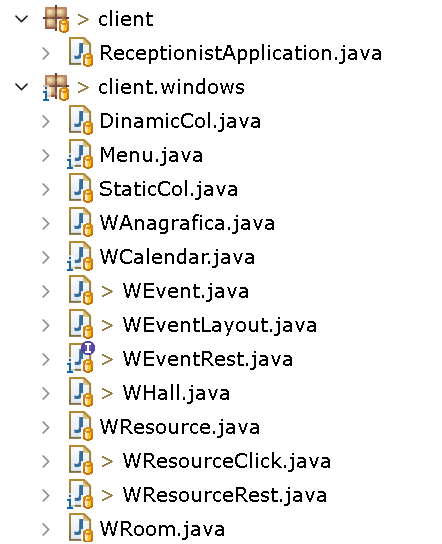
La modularità del progetto ha consentito il passaggio dai file json a MongoDB modificando solo il layer service. La versione 2, attraverso un flag in AppConfig è in grado di funzionare sia nella configurazione 2tiers che 3 tiers

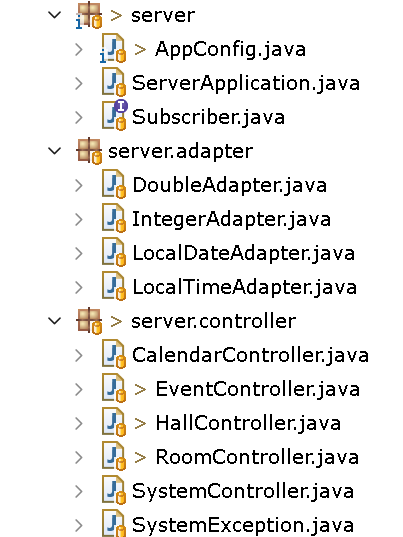
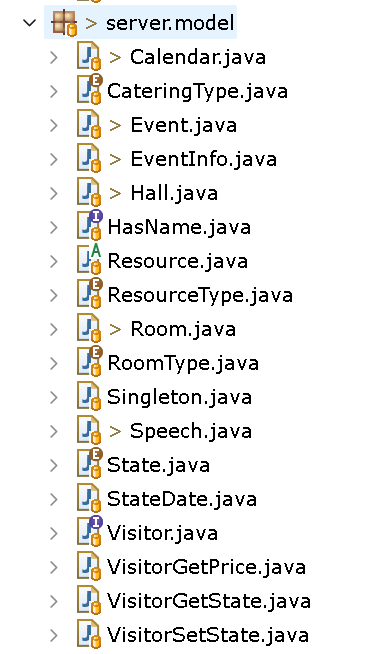
****

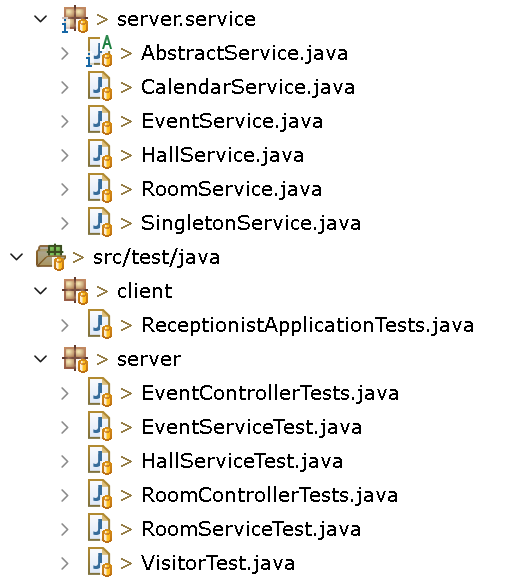


## Project Tree

I seguenti screen shots mostrano come gli schemi precedenti siano aderenti alla struttura del codice

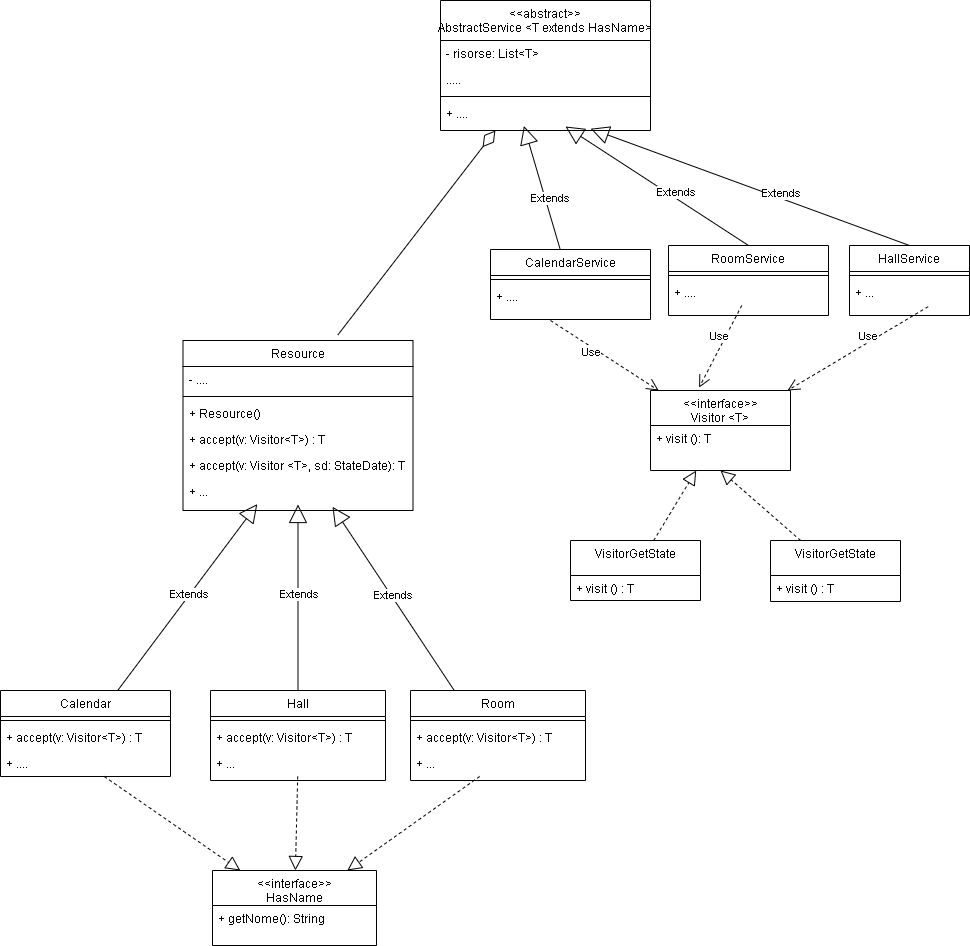
 

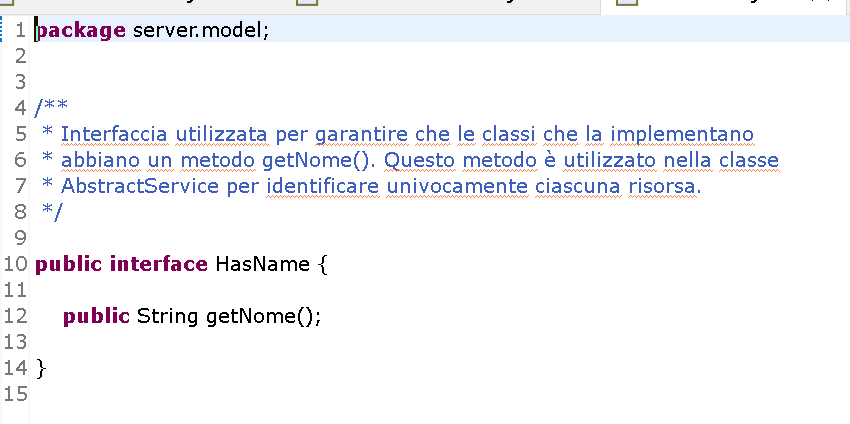


## Class Diagram (Design Pattern)

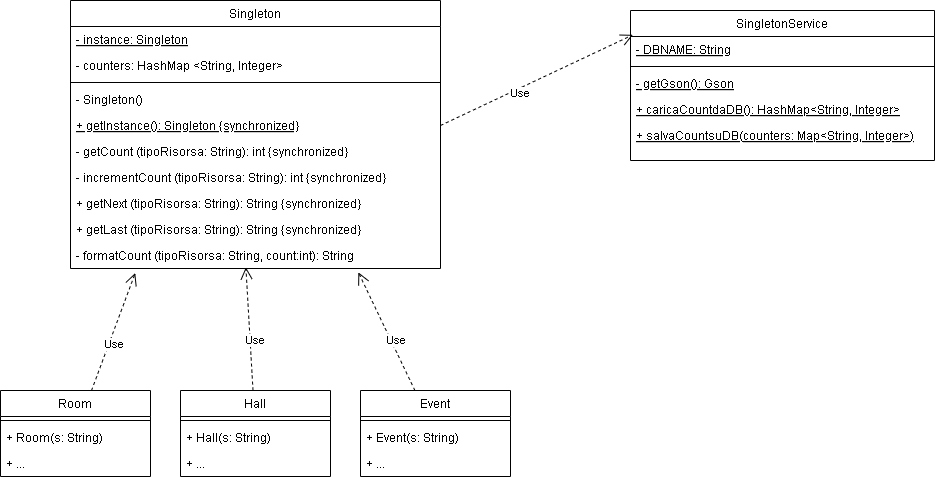
Per le class diagram complete si veda il §6.5 Class Diagrams

### Visitor

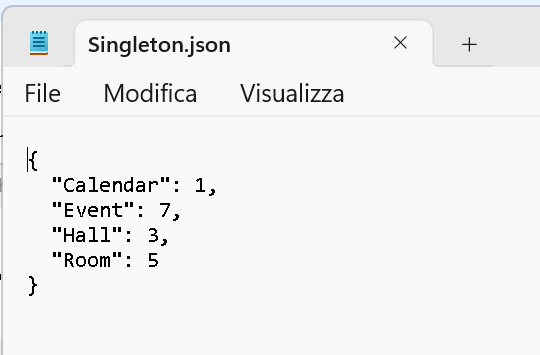




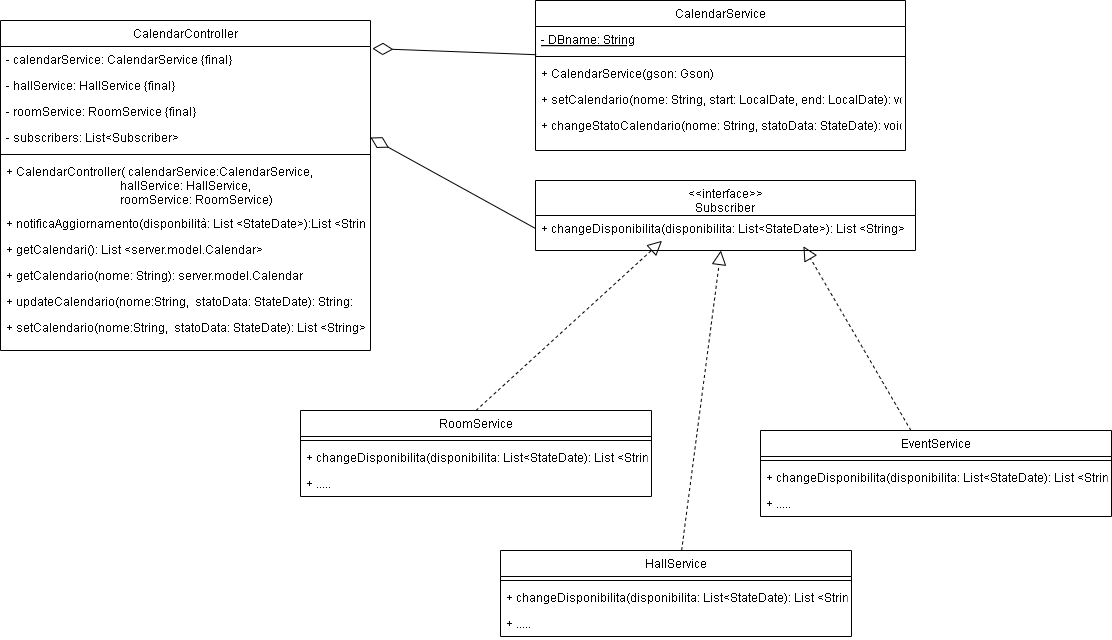
### Singleton

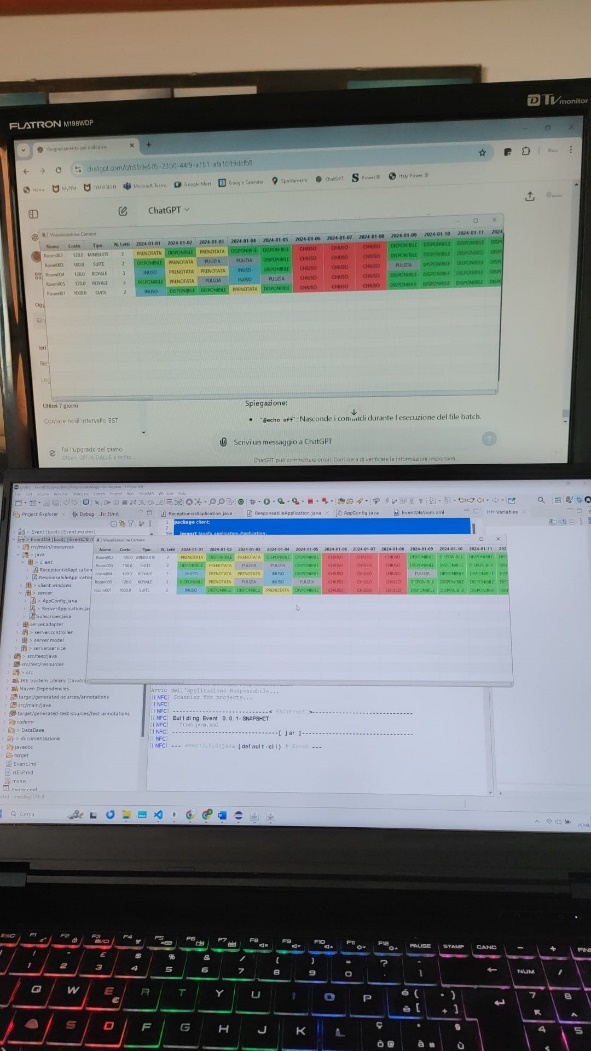


La classe SingletonService s’è resa necessaria per poter gestire il file Songleton.json volto a memorizzare i contatori dei vari tipi di Resource.



### Observer



L’Observer è utilizzato per notificare a tutti i service che è stato modificato il calendario, a seguito di tale notifica si procede al cambio degli stati delle risorse e alla generazione dei report utili per comunicare ai clienti/partecipanti/noleggiatori eventuali disdette delle prenotazioni per gravi motivi organizzativi.

Quando abbiamo testato il sistema con due client (con lo stesso codice) è emerso con chiarezza la necessità di un secondo pattern che a fronte della richiesta di modifica dei dai da parte di uno dei client notifichi a tutti gli altri l’avvenuta modifica. In realtà i pattern dovrebbero essere quante sono le risorse/API (Calendar, Event, Hall e Room) e vedrebbero tutte le relative finestre come Observer mentre i relativi service sarebbero i Subject), la dinamica potrebbe essere la seguente:

1. L’inizializzazione di un client dovrebbe utilizzare l’iniezione fornita da Spring per ottenere le istanze dei controller e chiedere di essere inserito nella lista degli Observer.
2. I vari controller aggiornano le rispettive liste con i riferimenti alle varie WResource relative
3. Ogni qualvolta che un client chiede la modifica di una risorsa il service che la esegue provvede anche alla sua notifica a tutti gli interessati.

Tale modifica è stata solo analizzata ma non realizzata.

## Class Diagrams

### Class: Visitor

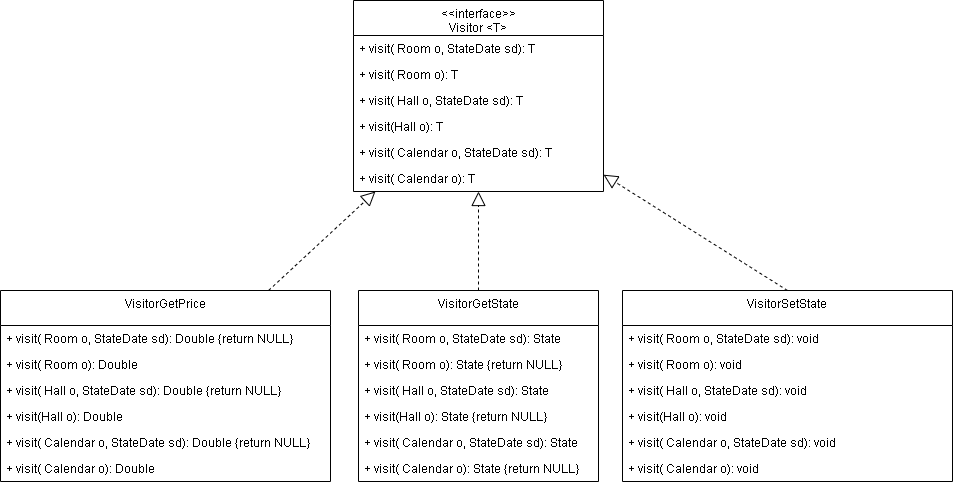
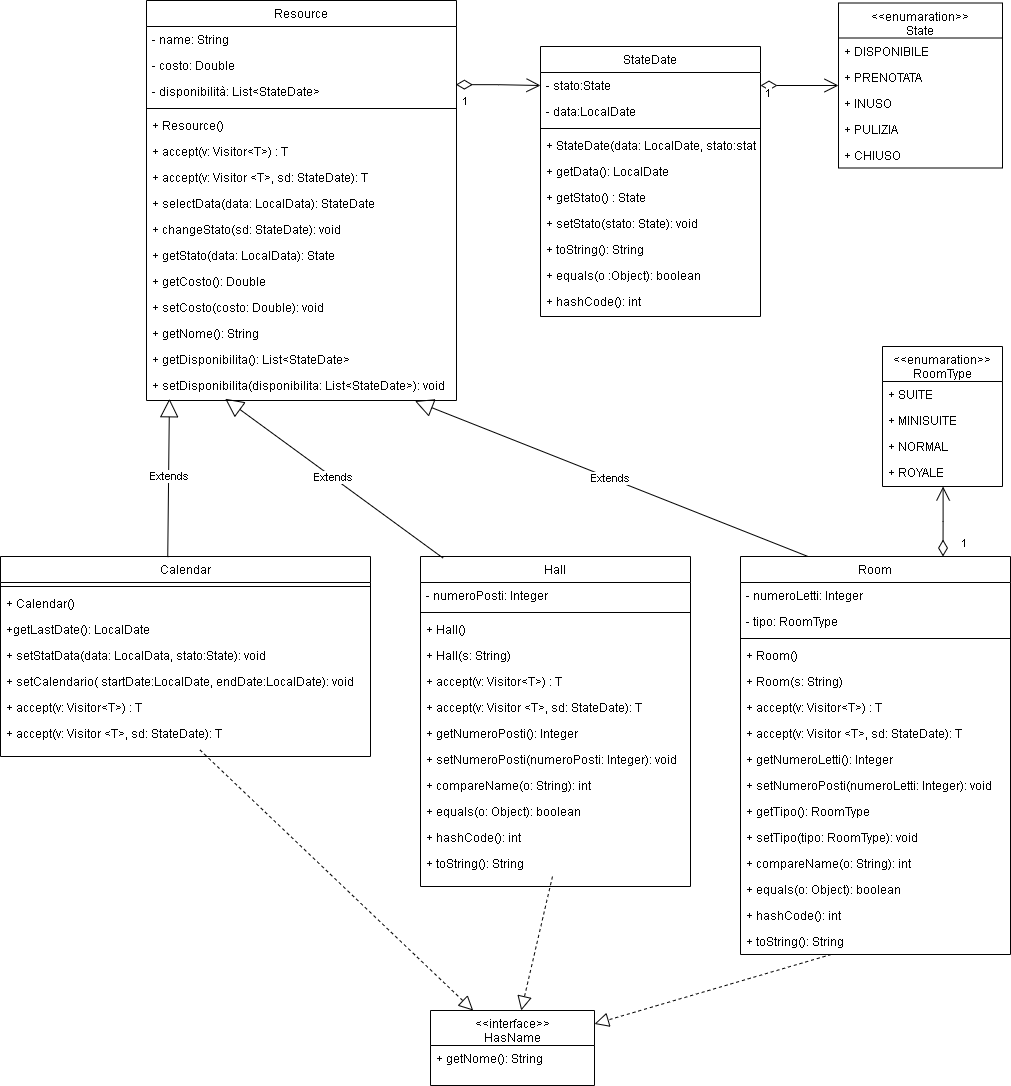


Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

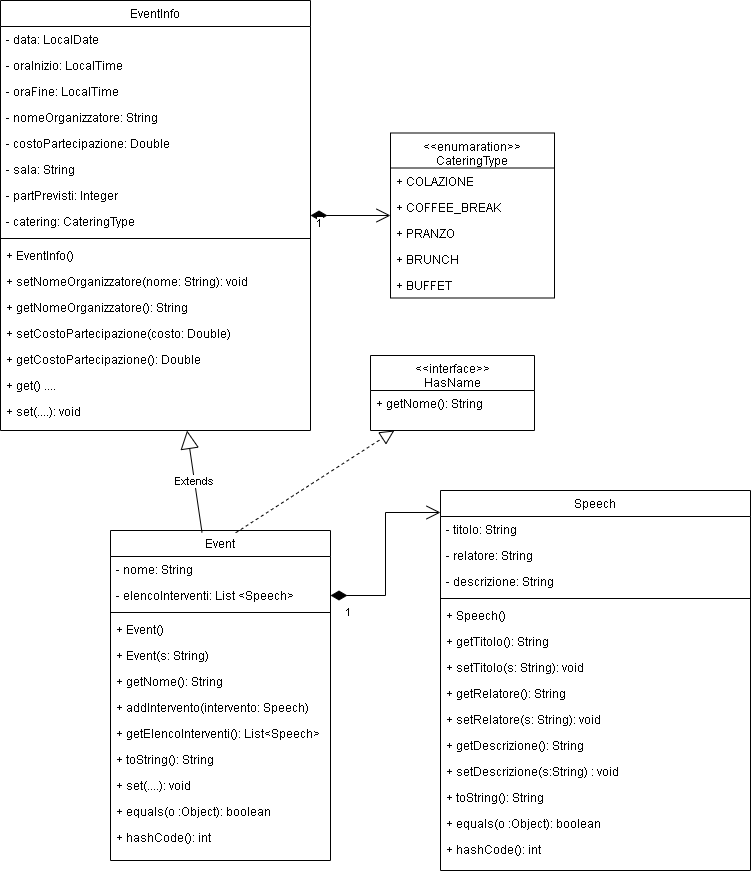
Il pattern Visitor è presente sin dalla prima stesura del codice.

### Class: Resource

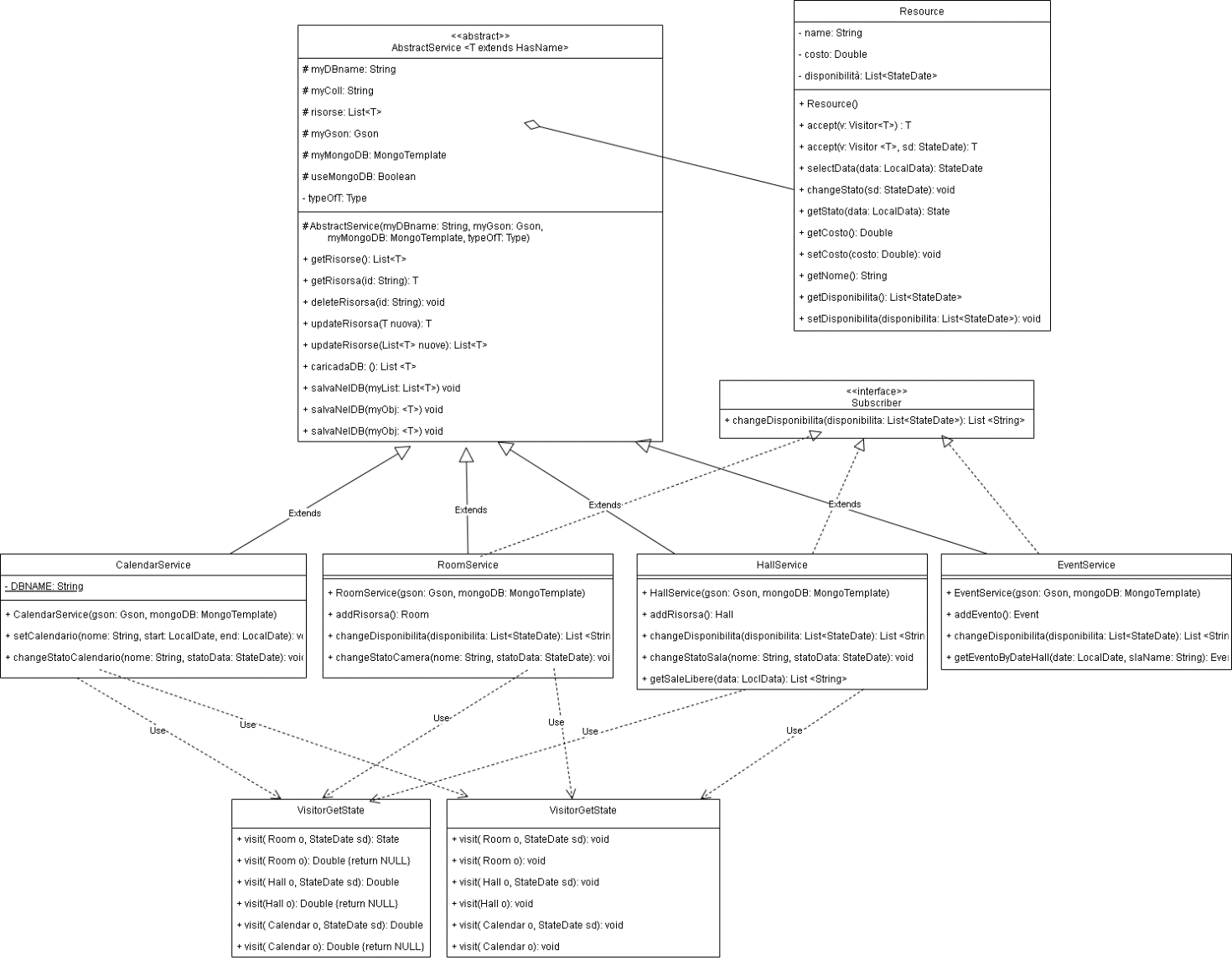


La Classe Resource è presente sin dalla prima stesura del codice.

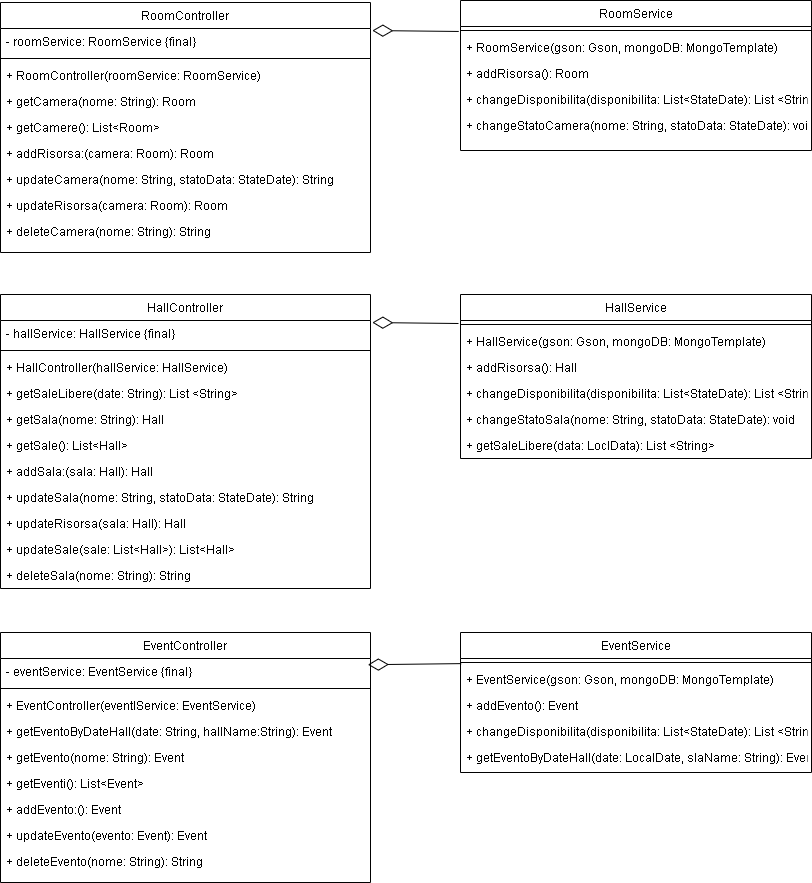
### Class: Event

****

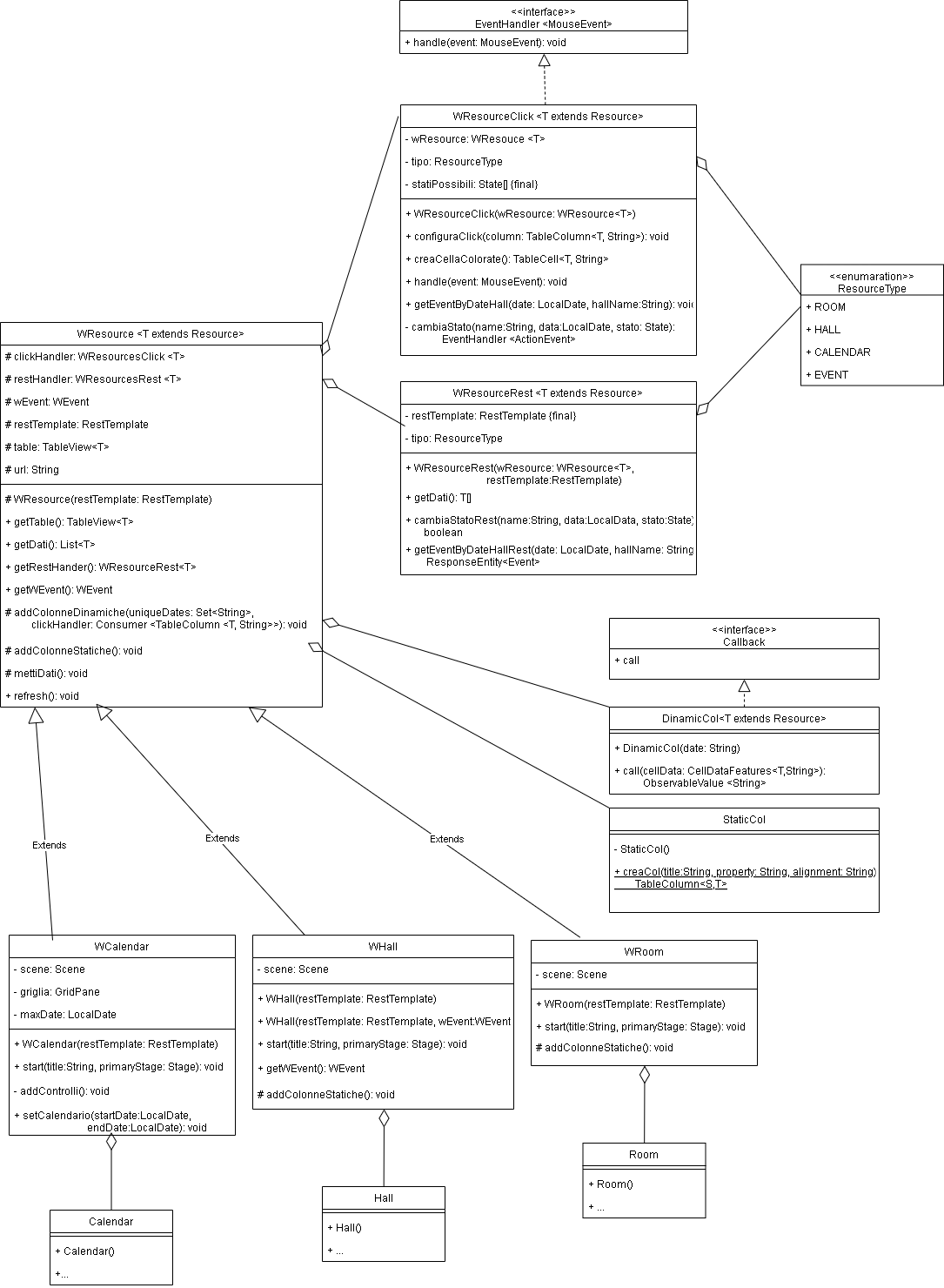
### Class: Service

****

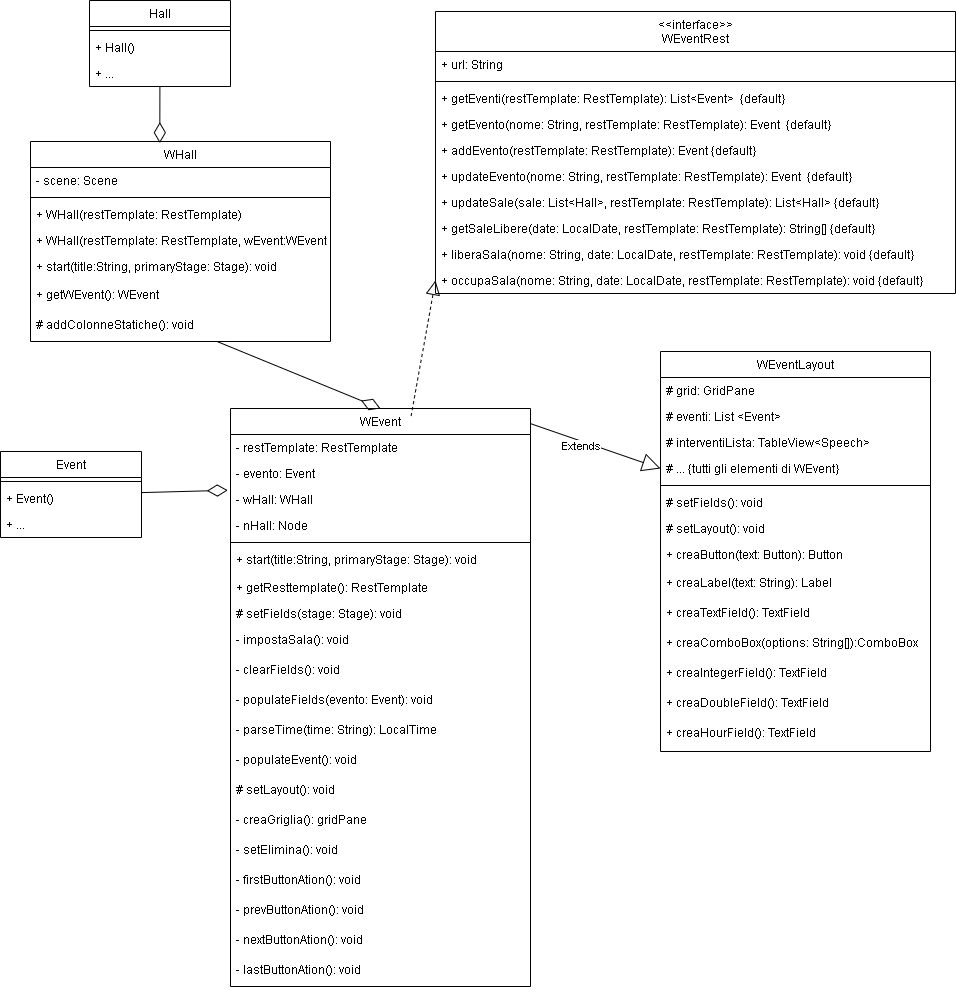
### Class: Controller

****

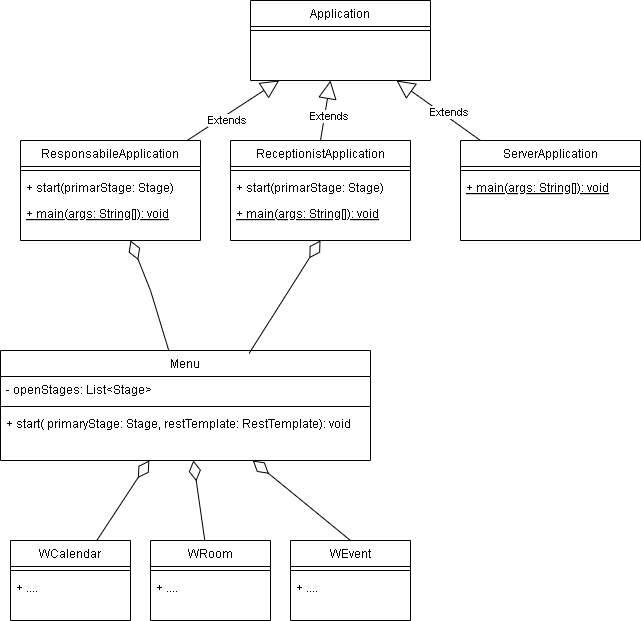
### Class: WResource

****

### Class: WEvent

****

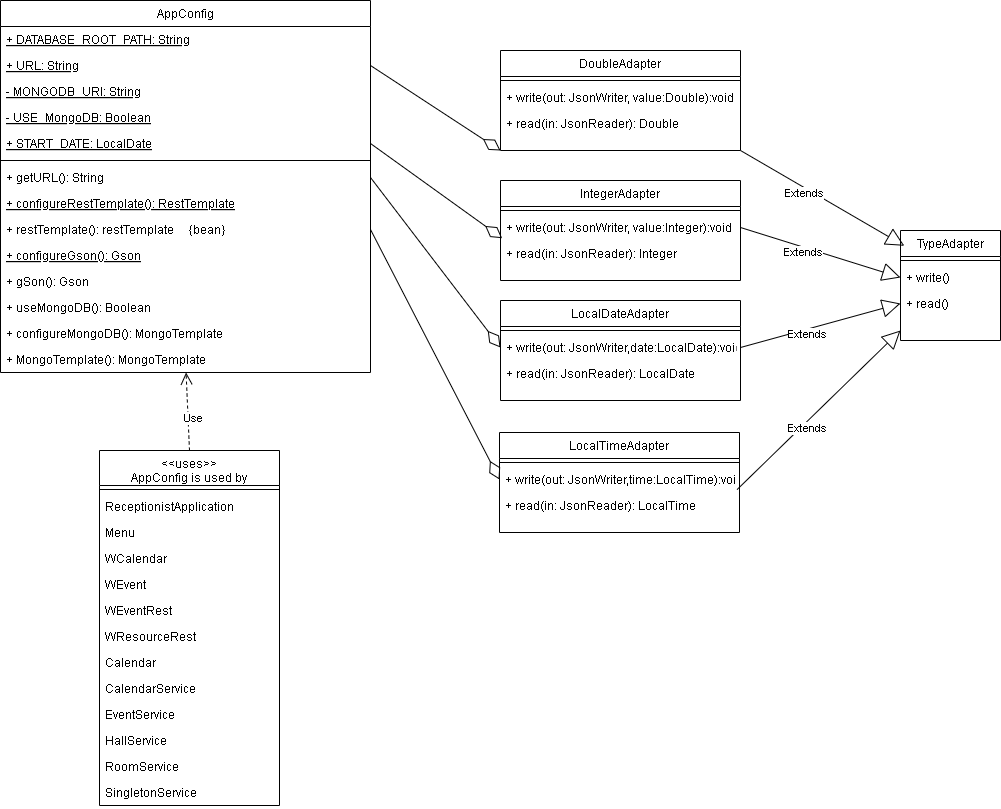
### Class: Application

****

Lo schema mostra le due classi che fungono da main per i due Client (ResponsabileApplication e ReceptionistApplication) e per il Server (ServerApplication) e la classe menu da cui si accede alle singole finestre. Poiché JavaFX impedisce l’esecuzione di più applicazioni all’interno della stessa JVM, uno dei due client viene lanciato attraverso il seguente file .bat



### Class: AppConfig

****

La classe AppConfig è chiamata da tutte le classi che necessitano di serializzare e/o de-serializzare gli oggetti facendo ricorso, se server, agli adattatori implementati.

La AppConfig è utilizzata anche per memorizzare:

* l’URL su cui sono in ascolto i controller
* la MongoDB\_URI: l’indirizzo dell’Atlas DataBase e relativa password
* USE\_MongoDB: il flag con cui si gestisce lo switch tra utilizzo dei file json e MongoDB
* la path (diversa per ciascun membro del gruppo di sviluppo) e la data che funge da T0 (Start Date)

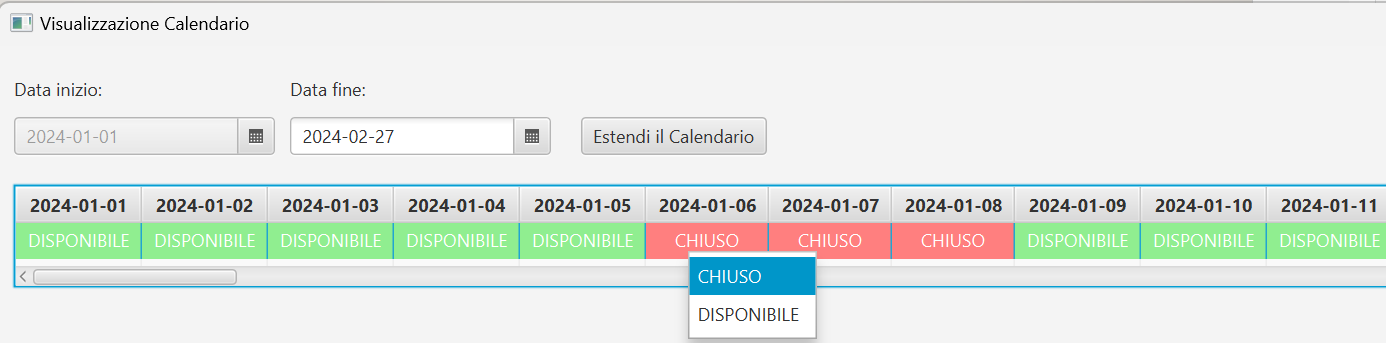
## Interfacce grafiche

Create grazie alla API JavaFX, sono interfacce grafiche dinamiche.

### Maschera Menu

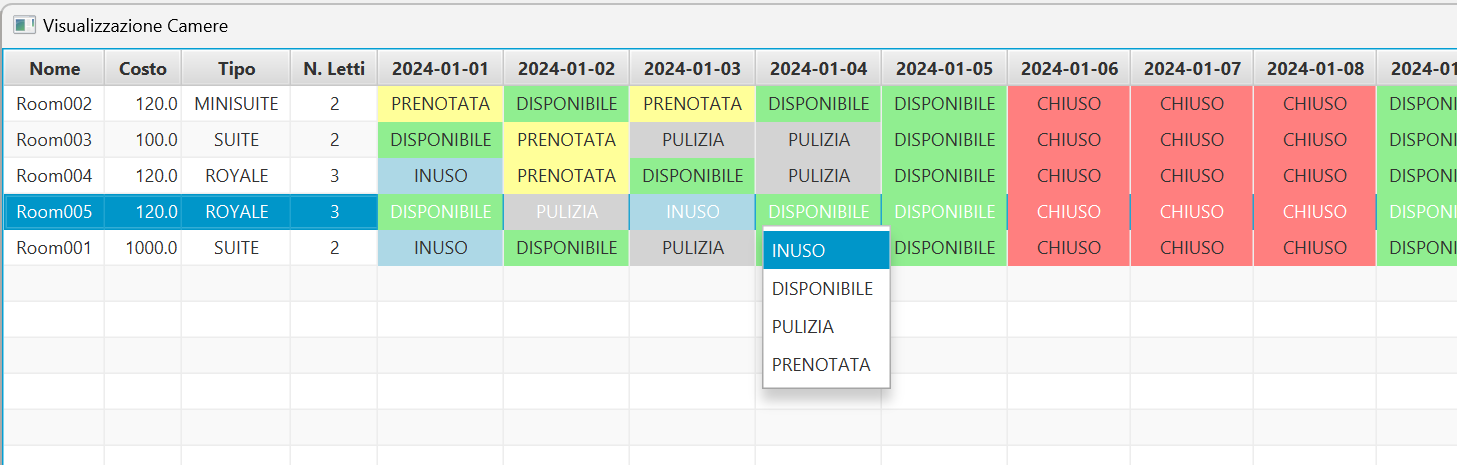


### Maschera WCalendar



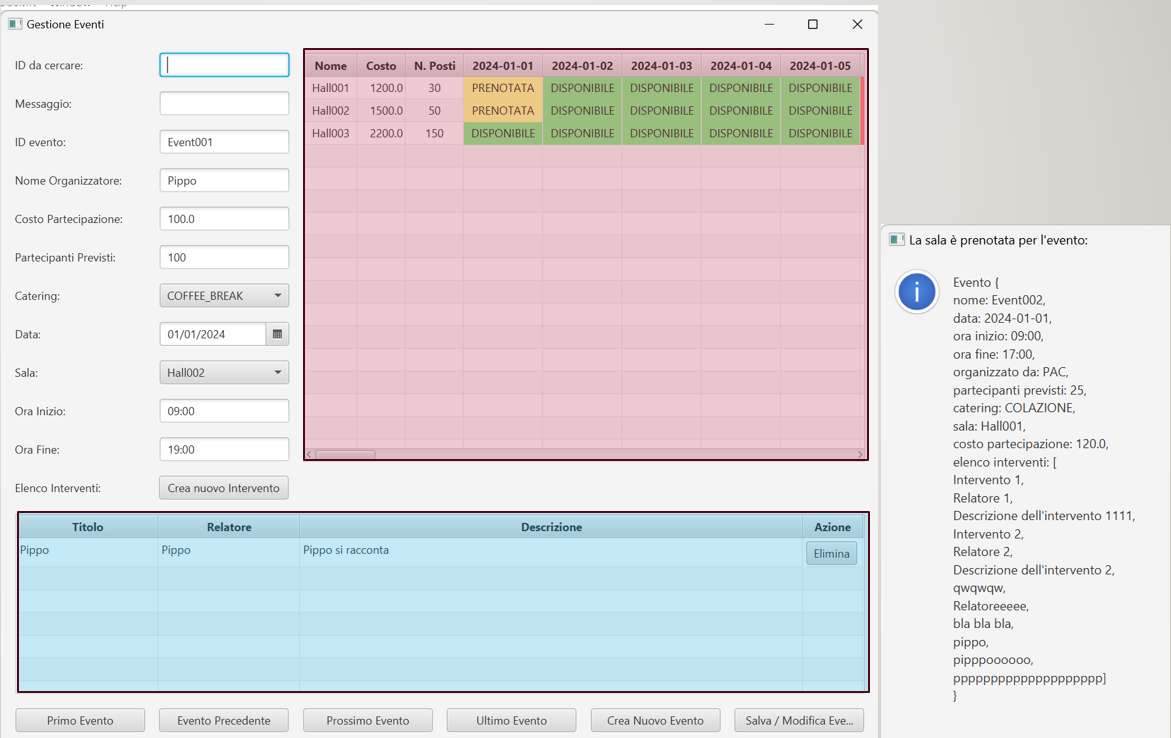
Cliccando sul singolo giorno è possibile cambiare lo stato

### Maschera WRoom



Il cambio dello stato di una camera per un determinato giorno è ottenuto cliccando sulla singola cella e selezionando il nuovo stato

### Maschera WEvent



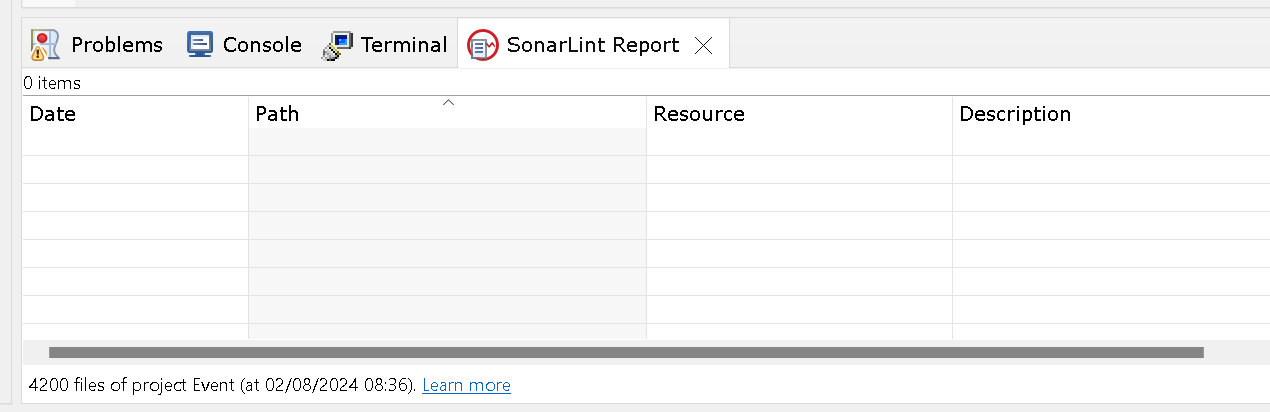
La maschera degli eventi ha due sotto-maschere:

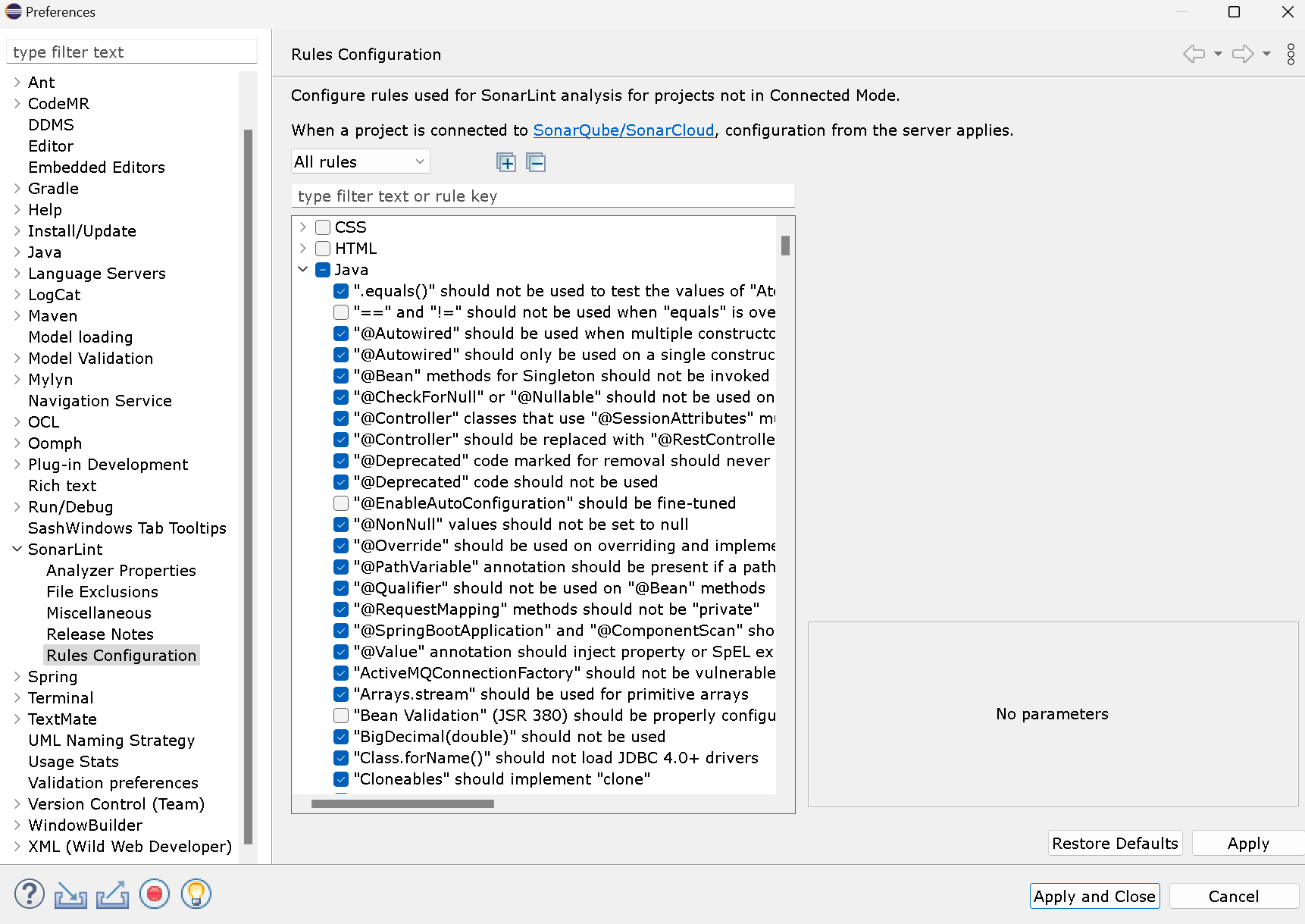
* La maschera delle sale (evidenziata in rosa) palesemente simile a quella delle camere, differisce per la gestione di stati diversi e per la gestione dei click. Cliccando su una data prenotata si ottengono le informazioni relative all’evento per cui è stata prenotata la sala. Può servire al Responsabile della foresteria per valutare una migliore allocazione delle sale in base al numero delle prenotazioni.
* La tabella degli speech (evidenziata in azzurro) consente l’eliminazione della singola riga (speech). L’eliminazione degli speech è l’unica delete prevista (la cancellazione delle camere o delle sale è stata considerata come una riconfigurazione del sistema, la cancellazione degli eventi dovrebbe essere gestita con un flag e non con una eliminazione dell’oggetto)
* La fila di bottoni in calce alla maschera consente di navigare nell’anagrafica degli eventi.

La numerosità dei controlli (utilizzati sia per popolare gli attributi dell’oggetto che per navigare da un evento all’altro) ha imposto la una classe ad hoc per la loro gestione.

## Analisi statica: SonarLint

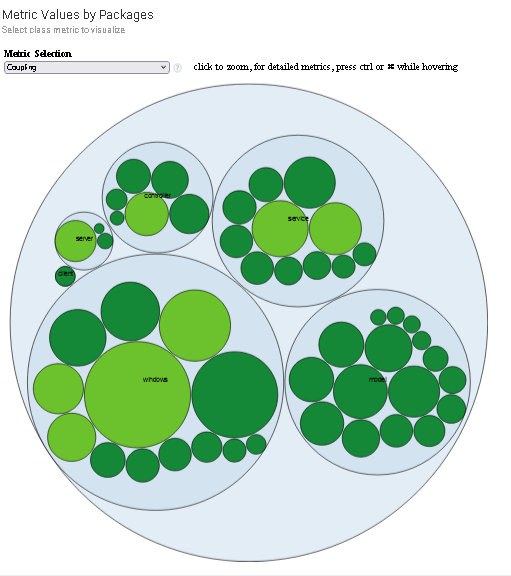
Per tutta la durata del progetto è stato usato SonarLint (in locale e quindi senza SonarQube o SonarCloud) disabilitando solo pochissime regole (Java) e usando il meno possibile il comando //NOSONAR (reso necessario nel codice delle classi di test che per essere visibili da JavaDoc devono essere dichiarate pubbliche).



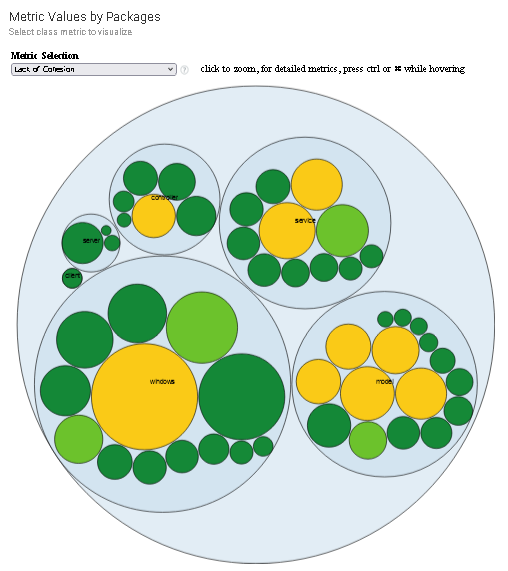


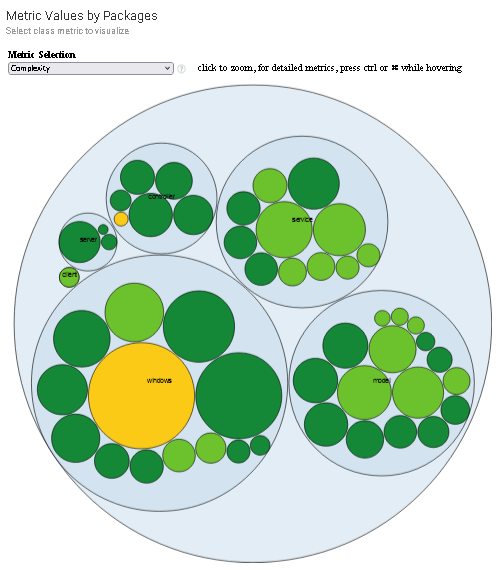
## Analisi statica: codeMR

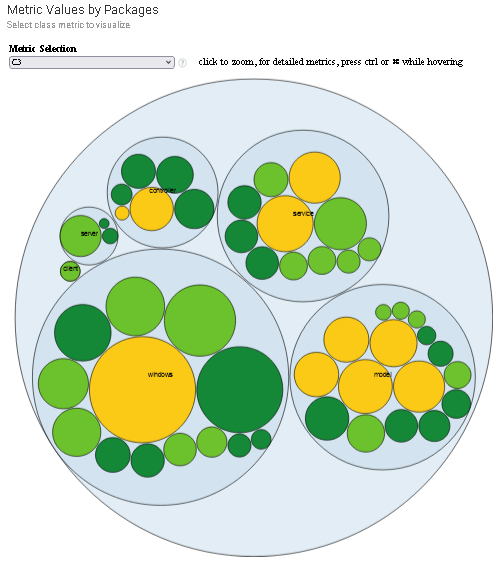
Si rimanda al § 5 ITERAZIONI INTERMEDIE per l’uso che si è fatto dello strumento, si riportano i grafici per la versione finale del codice in cui con C3 si intende la Related Quality Attributes (ovvero il valore massimo fra Coupling, Lack of Cohesion, Complexity metrics)

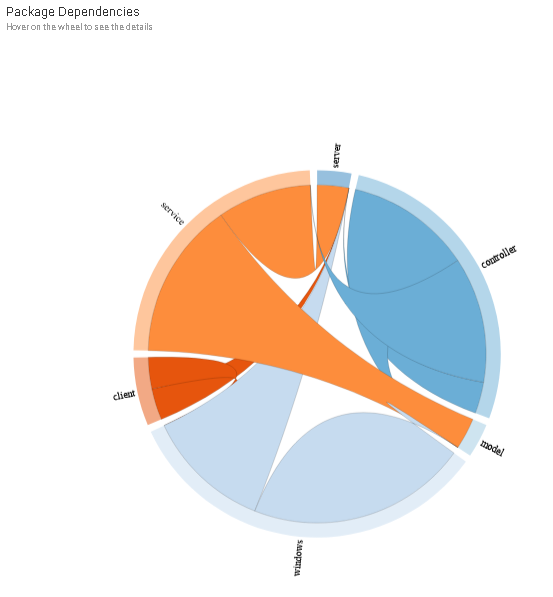


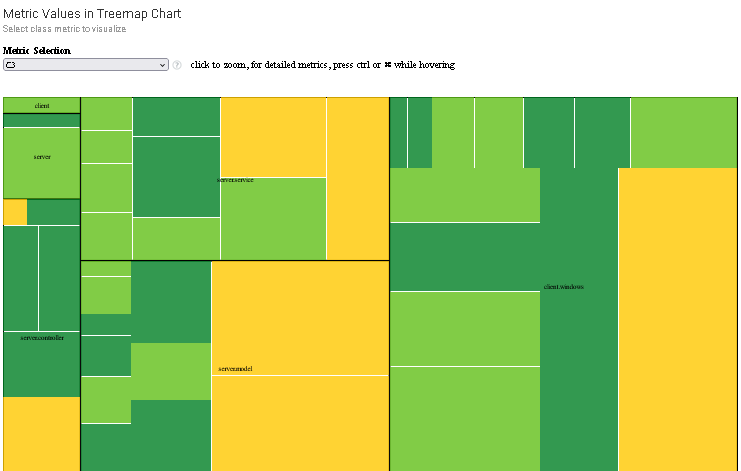












## Analisi dinamica: JUnit5

Sono state sviluppate complessivamente 7 classi di test per un totale di 20 test; ovviamente ciò non ha la pretesa di coprire tutto il codice, ma i pattern design utilizzati garantiscono comunque una discreta copertura. La classe meglio testata è la RoomService che insieme alla RoomController sono quelle che sono state realizzate nella iterazione 2 AMDD.

Tutto il codice relativo alla UI è stato testato verificandone direttamente il funzionamento.

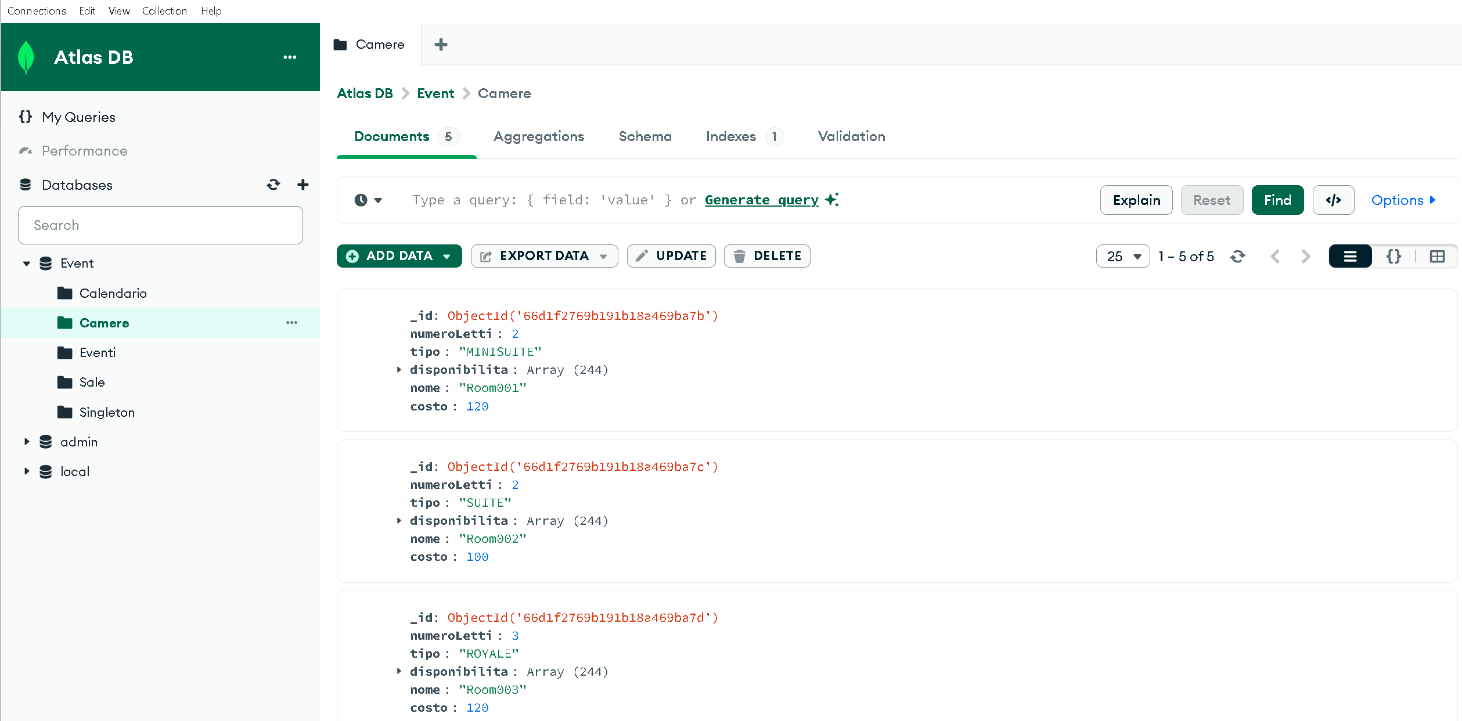
Tutti i test vengono superati dal codice (keep it green).

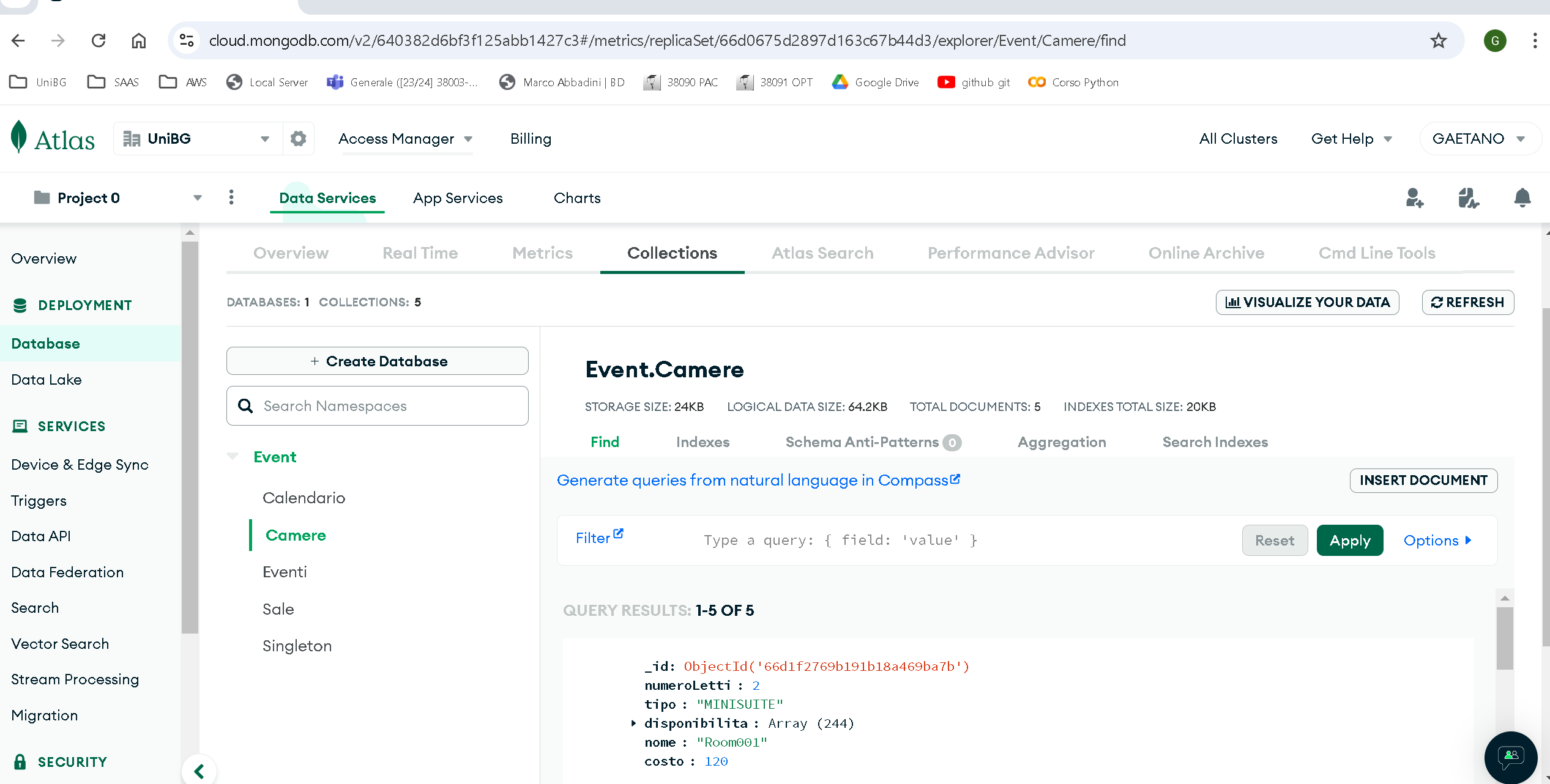
# DBMS: MongoDB – AtlasDB

Volendo semplificare al massimo la migrazione dai file json ad un DBMS abbiamo optato per MongoDB in quanto NoSQL. La versione desktop di MongoDB consente di gestire sia i DB in locale che quelli in cloud.

I DB sono suddivisi in Collection (l’equivalente delle relazioni/tabelle di un DBMS relazionale) a loro volta suddivisi in documenti (le tuple/record per un DBMS relazionale). MongoDB assegna un id a ciascun documento.

Importare file json è semplicissimo.



Su Atlas è necessario, innanzitutto creare un account e un Cluster definendone la dimensione e l’ubicazione (ed altre caratteristiche che concorrono a definirne il costo). Ogni cluster può contenere più DB. L’entità logica di più alto livello è il Progetto (che può usare più cluster).

# Documentazione: JavaDoc

Tutte le classi e tutti i metodi (ad eccezione di quelli banali) sono stati commentati usando i comandi di JavaDoc, ciò ha consentito di ottenere la documentazione del codice.

