

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
1.1	Modello Organizzativo adottato.....	4
2	ITERAZIONE 0 AMDD (Requirements & Architecture Envisioning) .....	6
2.1	Requirements Envisioning .....	6
2.1.1	Nonfunctional requirements .....	6
2.1.2	Functional requirements (Customer's requirements).....	6
2.2	Architecture Envisioning.....	6
2.2.1	Use Case Diagram.....	7
2.2.2	Informal Deployment Diagram.....	8
3	ITERAZIONE 1 AMDD (Early Architecture Design) .....	9
3.1	Deployment View .....	10
3.2	Functionality View .....	11
3.3	Activity Diagram .....	12
3.4	Use Case Stories .....	13
3.4.1	Responsabile della foresteria e/o Receptionist.....	13
3.4.2	Addetto Pulizia.....	13
3.4.3	Noleggiatore .....	13
3.4.4	Partecipante .....	14
3.4.5	Cliente.....	14
3.5	Use Case Stories (fully dressed description).....	14
3.6	Requirements Analysis .....	18
4	ITERAZIONE 2 AMDD .....	20
4.1	Tool Chain e Tecnologie.....	20
4.1.1	codeMR.....	20
4.1.2	Gson .....	20
4.1.3	MongoDB e Atlas .....	20
4.2	Implementazione.....	21
4.2.1	Selezione funzioni da implementare .....	21
4.2.2	Component Diagram .....	21
4.2.3	Analisi complessità metodo: <i>WRoom.mettiDati</i> .....	22
4.2.4	Analisi dinamica.....	22
4.2.5	Analisi statica.....	24
4.2.6	Maschera delle camere .....	25
4.2.7	Conclusioni iterazione 2 .....	25
5	ITERAZIONI INTERMEDIE .....	26
6	ITERAZIONE FINALE .....	28

6.1	Architectural Pattern .....	28
6.2	Formal Deployment Diagram .....	29
6.2.1	Versione 1 (two tiers) .....	29
6.2.2	Versione 2 (three tiers) .....	31
6.3	Project Tree .....	33
6.4	Class Diagram (Design Pattern) .....	35
6.4.1	Visitor .....	35
6.4.2	Singleton .....	36
6.4.3	Observer .....	37
6.5	Class Diagrams .....	38
6.5.1	Class: Visitor .....	38
6.5.2	Class: Resource .....	39
6.5.3	Class: Event .....	40
6.5.4	Class: Service .....	41
6.5.5	Class: Controller .....	42
6.5.6	Class: WResource .....	43
6.5.7	Class: WEvent .....	44
6.5.8	Class: Application .....	45
6.5.9	Class: AppConfig .....	46
6.6	Interfacce grafiche .....	47
6.6.1	Maschera Menu .....	47
6.6.2	Maschera WCalendar .....	47
6.6.3	Maschera WRoom .....	47
6.6.4	Maschera WEvent .....	48
6.7	Analisi statica: SonarLint .....	49
6.8	Analisi statica: codeMR .....	50
6.9	Analisi dinamica: JUnit5 .....	55
7	DBMS: MongoDB – AtlasDB .....	56
8	Documentazione: JavaDoc .....	57

# 1 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.

## 1.1 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 § 3.6 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 §6.9 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

The screenshot shows a Trello board titled "Progetto Event". The board is organized into five columns, each representing a step in the AMDD process:

- STEP 0:** Contains cards for "AMDD 0 development" (due 4 dic 2023 - 5 dic 2023) and "API-LED" (due 4 dic 2023 - 5 dic 2023). It also has a "+ Aggiungi una scheda" button.
- STEP 1 - WANT TO BE:** Contains cards for "Project definition: APP for event management and room availability for guests" (due 7 dic 2023 - 8 dic 2023), "MVC as Architectural Structures" (due 6 dic 2023 - 7 dic 2023), "Tools Chain Definition" (due 3 gen - 6 gen), "RESTfull" (due 4 gen - 7 gen), and "RESTfull + Using Spring + JavaFX = Passive MVC (all data through the controller)" (due 8 gen - 10 gen). It also has a "+ Aggiungi una scheda" button.
- STEP 2 - AMDD\_0:** Contains cards for "User Cases" (due 10 gen - 13 gen), "1st commit on GitHub" (due 15 gen - 13 gen), "Target Time Definition" (due 13 gen - 14 gen), "First Project Scheduling" (due 12 gen - 14 gen), and "Configuration Management" (due 13 gen - 14 gen). It also has a "+ Aggiungi una scheda" button.
- STEP 3 - AMDD\_1:** Contains cards for "Early Design" (due 14 dic 2023 - 15 dic 2023), "Resource List" (due 15 dic 2023 - 16 dic 2023), "Visitor List" (due 15 dic 2023 - 16 dic 2023), "Visitor Pattern Interface definition" (due 16 dic 2023 - 17 dic 2023), "Receptionist's windows state diagram" (due 16 dic 2023 - 17 dic 2023), "Guest's windows state diagram" (due 16 dic 2023 - 17 dic 2023), "Event planner' windows state diagram" (due 16 dic 2023 - 17 dic 2023), and "Static code analysis" (due 16 gen - 16 gen). It also has a "+ Aggiungi una scheda" button.
- STEP 4 - AMDD\_2:** Contains cards for "Room.json definition" (due 15 gen - 16 gen), "Room implementation" (due 16 gen - 16 gen), "Room windows implementation" (due 16 gen - 16 gen), "Room dynamic testing" (due 16 gen - 16 gen), "Static code analysis" (due 15 gen - 16 gen), "Room delivery" (due 15 gen - 16 gen), and "Calendar definition" (due 17 gen - 17 gen). It also has a "+ Aggiungi una scheda" button.

trell.com/b/CcWDGTUq/progetto-event

UnIBG SaaS AWS Local Server Generale [23/24] 38003... Marco Abbadini | BD 38090 PAC 38091 OPT Google Drive github git Corso Python

Trello Spazi di lavoro Recenti Preferita Modelli Crea

Progetto Event Visibile allo Spazio di lavoro Bacheca Timeline Power-Up Automazione Filtri GC

Step	Description	Completion Date
STEP 6 - AMDD_4 - 17/01/2024	Event list definition, Event list implementation, Event windows implementation, Events dynamic testing, Static code analysis	17 gen
STEP 7 - AMDD_5 - 18/01/2024	Room Windows semplificata, Event Windows con lambda	18 gen
STEP 8 - AMDD_6 - 19/01/2024	Handler in condivisione a WHello e WRoom, Static code analysis	19 gen
STEP 9 - AMDD_7 - 20/01/2024	Finestre finite, Menu finito	20 gen
STEP 10 - AMDD_8 - 21/01/2024	Fine stesura codice, Static code analysis	20 gen
STEP 11 - AMDD_9 - 25/01/2024	Static code analysis and code improvement, Dynamic code analysis and code improvement	24 gen - 25 gen
STEP 12 - AMDD_10 - 26/01/2024	debug WEvent, modifica POM	25 gen - 26 gen
STEP 13 - AMDD_11 - 27/01/2024	Codice implementato funzionante revisionato, tabella UseCase aggiornati e revisionati	26 gen - 27 gen
STEP 14 - documentazione	Architectural views, Class diagrams, Deployments diagrams, Patterns description, Relazione	30 gio - 31 lug
STEP 15 - powerpoint	MongoDB, Versione 3tiers, Aggiornamento Documentazione	26 ago - 31 ago

+ Aggiungi un'altra lista

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 § 0)

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.

## 2 ITERAZIONE 0 AMDD (Requirements & Architecture Envisioning)

### 2.1 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.

#### 2.1.1 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).

#### 2.1.2 Functional requirements (Customer's requirements)

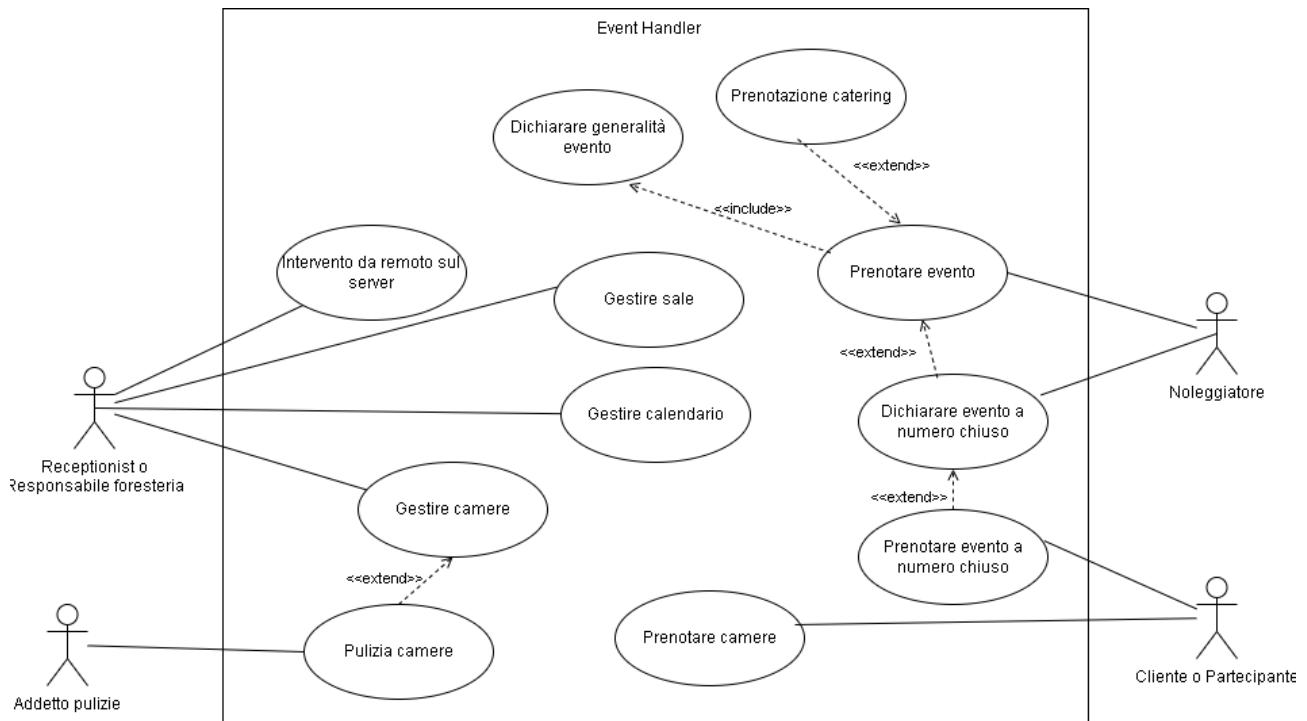
Realizzare un progetto che consenta di gestire una foresteria in cui:

- 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.

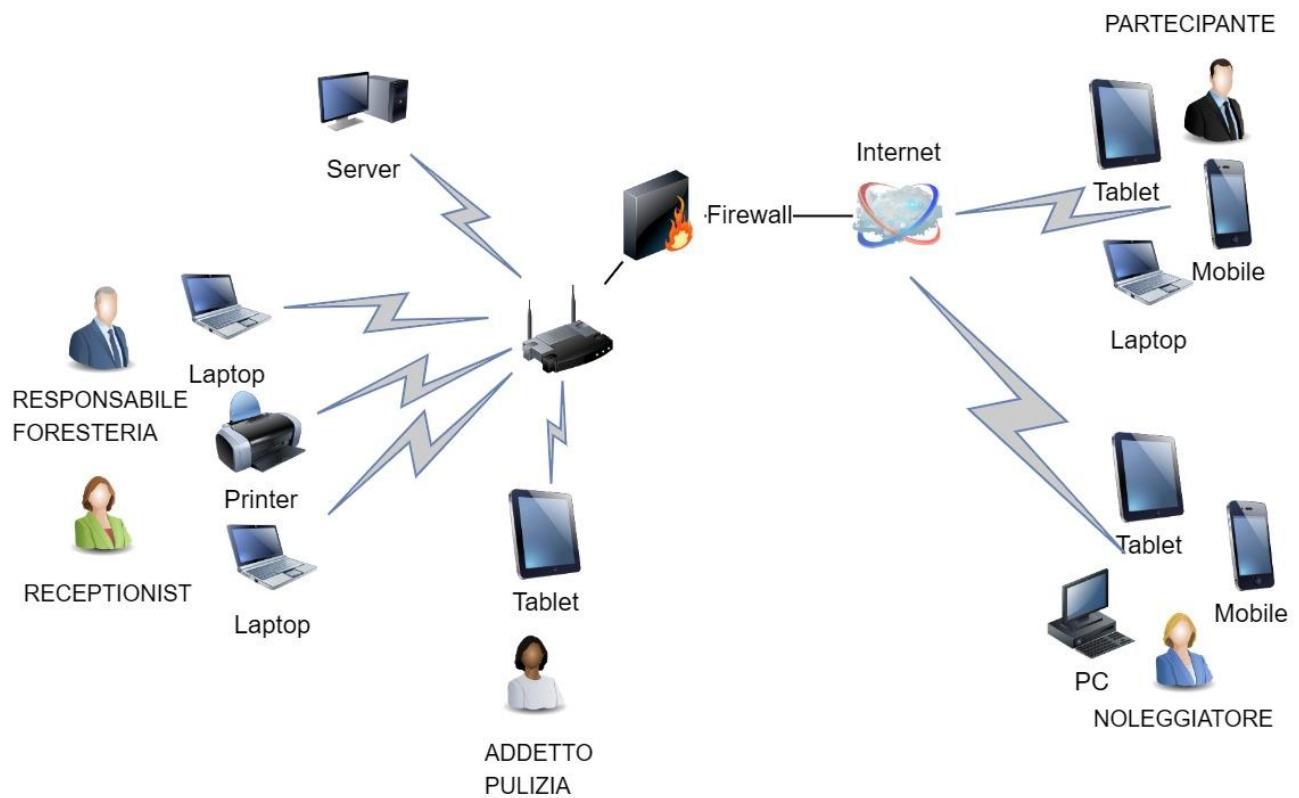
### 2.2 Architecture Envisioning

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

## 2.2.1 Use Case Diagram



## 2.2.2 Informal Deployment Diagram



### **3 ITERAZIONE 1 AMDD (Early Architecture Design)**

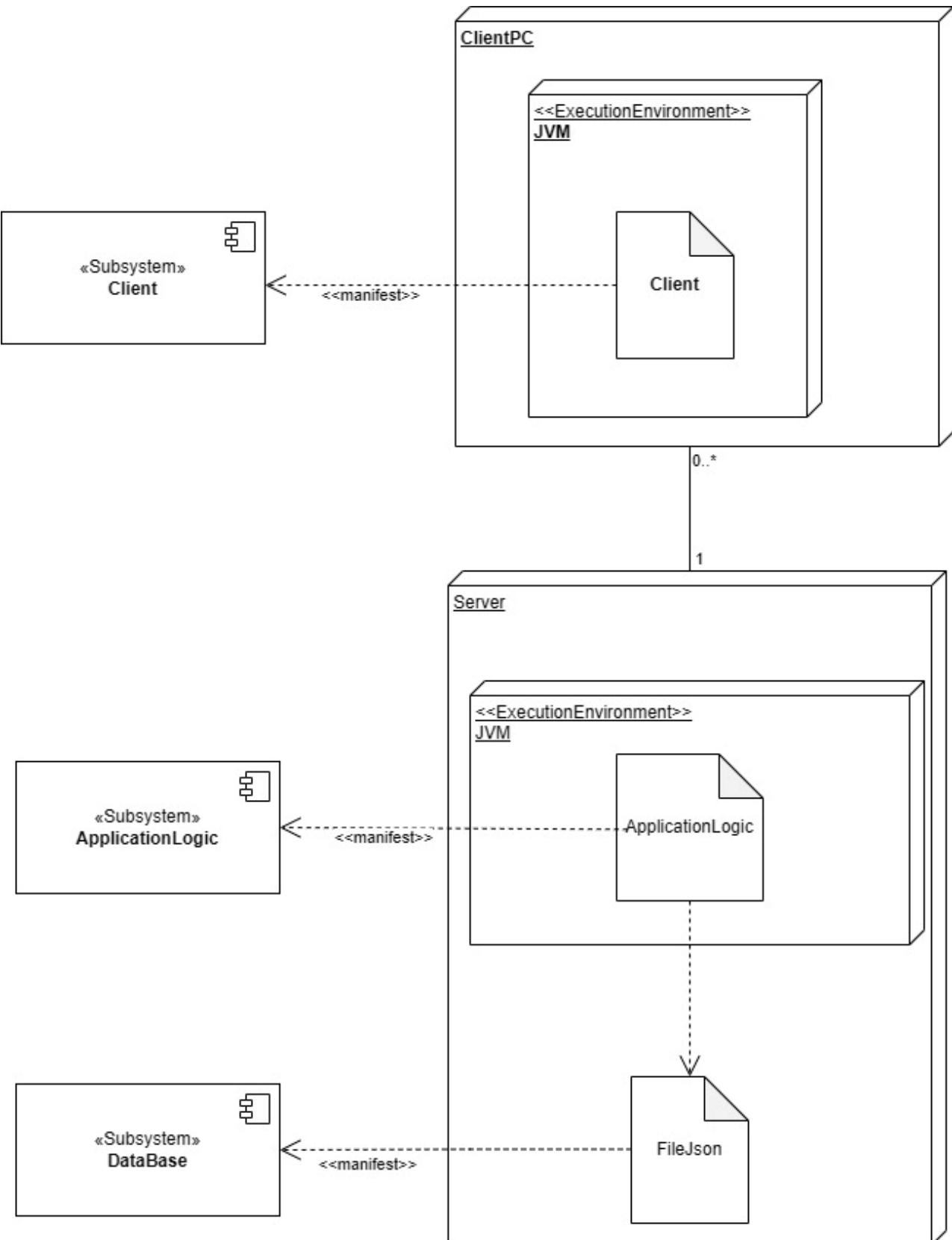
In questa fase si è definita l'architettura software da implementare: si è optata la three layers architecture (ovvero architettura a tre strati), ovvero un'architettura che organizza le applicazioni in tre strati di codice: il layer presentazione, o interfaccia utente, il layer applicazione, dove i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

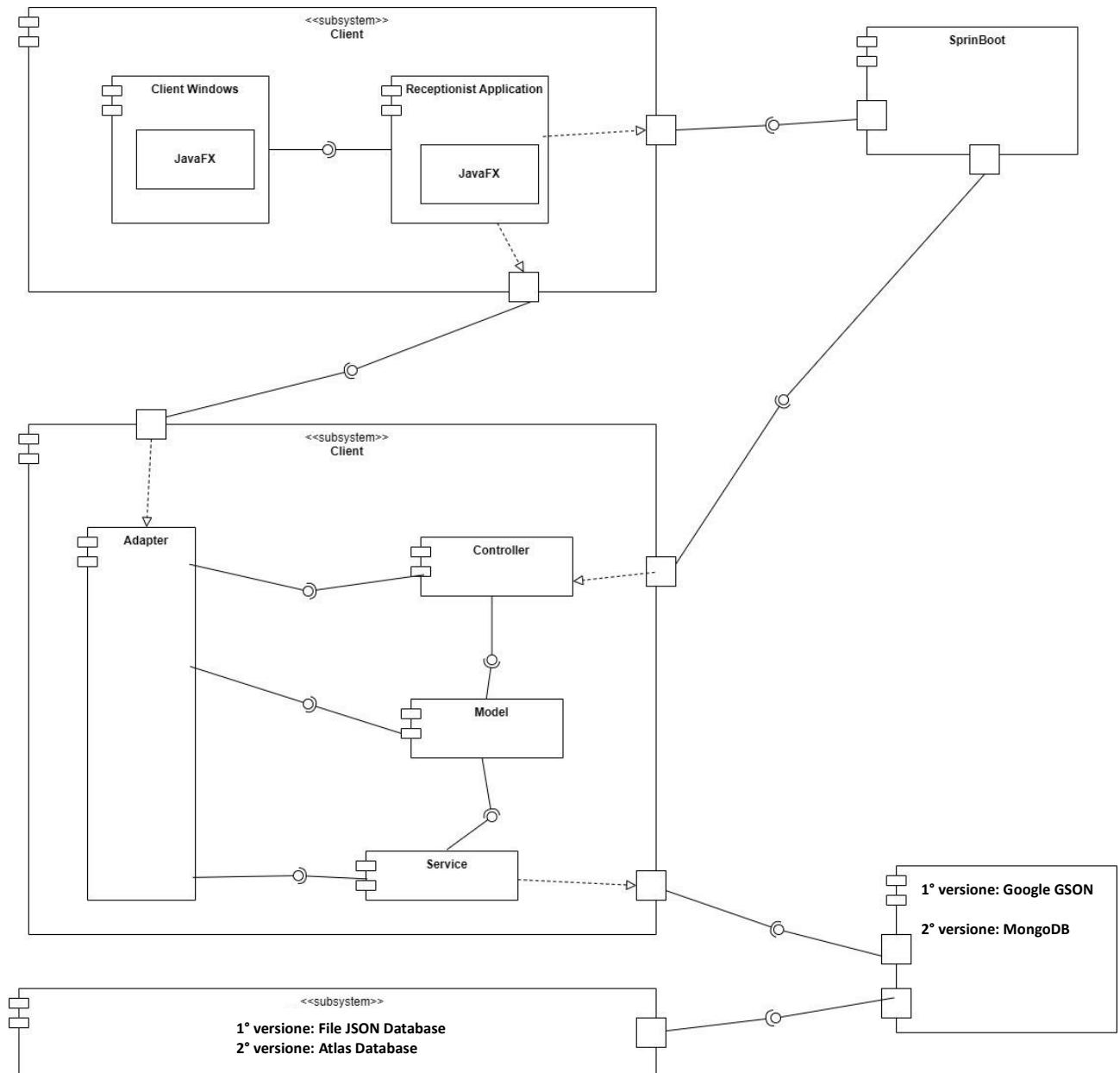
Il Client e il Server comunicano mediante il framework Spring Boot che abbiamo utilizzato la tecnologia REST, le API sono rigorosamente RESTful in quanto:

- 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

### 3.1 Deployment View

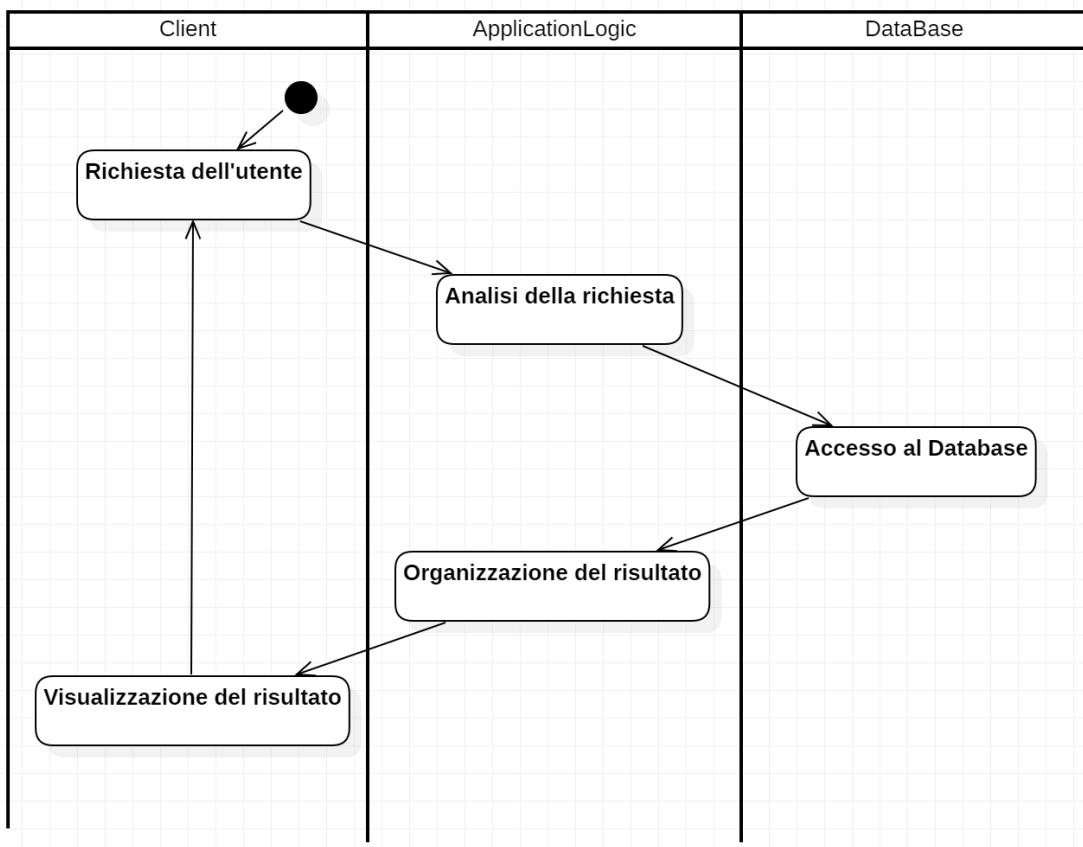


## 3.2 Functionality View



### 3.3 Activity Diagram

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



## 3.4 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

### 3.4.1 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

### 3.4.2 Addetto Pulizia

#### GESTIRE CAMERE

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

### 3.4.3 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 GENERALITÀ 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)

### **3.4.4 Partecipante**

## **PRENOTAZIONE PARTECIPAZIONE**

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

### **3.4.5 Cliente**

## **PRENOTAZIONE CAMERA**

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

## **3.5 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.

<b>Codice</b>	RF01	
<b>Nome</b>	Gestire sale	
<b>Attori</b>	Responsabile foresteria	
<b>Goals</b>	Aggiungere sale prenotabili in delle precise date	
<b>Precondizioni</b>	Selezione delle date in cui ciascuna aula è disponibile alla prenotazione	
<b>Breve descrizione</b>	Il responsabile aggiunge le sale che potranno essere prenotate per degli eventi	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	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
<b>Casi d'uso correlati e processi alternativi</b>		
<b>Post condizioni</b>	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.

<b>Codice</b>	RF02	
<b>Nome</b>	Gestire calendario	
<b>Attori</b>	Responsabile foresteria	
<b>Goals</b>	Aggiornare i giorni di chiusura della foresteria	
<b>Precondizioni</b>	Nessuna	
<b>Breve descrizione</b>	Il responsabile della foresteria aggiorna il calendario di apertura/chiusura della foresteria a seconda delle sue necessità	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	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
<b>Casi d'uso correlati e processi alternativi</b>		
<b>Post condizioni</b>	<p>Calendario con date di chiusura scelte aggiornato.</p> <p>Eventuali eventi o camere prenotati per date che erano precedentemente dichiarate di apertura vengono annullate.</p> <p>Il sistema deve produrre un elenco delle prenotazioni annullate affinché si possa avvisare la clientela.</p>	

<b>Codice</b>	RF03	
<b>Nome</b>	Gestire camere	
<b>Attori</b>	Responsabile foresteria / Receptionist	
<b>Goals</b>	Aggiornare le prenotazioni delle camere della foresteria	
<b>Precondizioni</b>	Aver aggiornato calendario	
<b>Breve descrizione</b>	Il responsabile aggiorna i giorni in cui ogni camera è stata prenotata da un cliente, in fase di pulizia, o disponibile alla prenotazione	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	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
<b>Casi d'uso correlati e processi alternativi</b>		
<b>Post condizioni</b>	Camere nel nuovo stato	

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

<b>Codice</b>	AP01	
<b>Nome</b>	Pulizia camere	
<b>Attori</b>	Addetto pulizie	
<b>Goals</b>	Poter effettuare la pulizia delle camere desiderate	
<b>Precondizioni</b>	Aver aggiornato calendario	
<b>Breve descrizione</b>	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	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	Dopo aver ricevuto la notifica data da "Gestire camere", l'addetto alle pulizie pulisce le camere nelle date indicate dal sistema	
<b>Casi d'uso correlati e processi alternativi</b>	Estende Gestione Camere	
<b>Post condizioni</b>		

<b>Codice</b>	N01	
<b>Nome</b>	Prenotazione Evento	
<b>Attori</b>	Noleggiatore	
<b>Goals</b>	Prenotare una sala della foresteria per ospitare un evento in una determinata data	
<b>Precondizioni</b>	Sale messe a disposizione dal responsabile della foresteria	
<b>Breve descrizione</b>	Il noleggiatore apre dall'applicazione la finestra per effettuare la prenotazione dello spazio apposito della foresteria adibito a ospitare un evento in date disponibili	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	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.
<b>Casi d'uso correlati e processi alternativi</b>	Non ci sono sale libere in nessuna data disponibile. Il noleggiatore non può prenotare nessuna sala per l'evento.	
<b>Post condizioni</b>	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".

<b>Codice</b>	N02	
<b>Nome</b>	Definire generalità evento	
<b>Attori</b>	Noleggiatore	
<b>Goals</b>	Dichiarare i dati generali dell'evento	
<b>Precondizioni</b>	Aprire la finestra degli eventi	
<b>Breve descrizione</b>	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.	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	Il noleggiatore inserisce negli appositi campi i dati dell'evento.	Il sistema registra i dati dell'evento.
<b>Casi d'uso correlati e processi alternativi</b>	Incluso in prenotazione evento.	
<b>Post condizioni</b>	L'evento modificato è salvato in memoria con i suoi dati.	

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

<b>Codice</b>	N04	
<b>Nome</b>	Prenotazione catering	
<b>Attori</b>	Noleggiatore	
<b>Goals</b>	Noleggiare un servizio catering per l'evento.	
<b>Precondizioni</b>	Aprire la finestra degli eventi	
<b>Breve descrizione</b>	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, ...).	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	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.
<b>Casi d'uso correlati e processi alternativi</b>	Estensione di "Dichiarare generalità evento".	
<b>Post condizioni</b>	L'evento presenta in memoria il servizio catering con i pasti scelti.	

<b>Codice</b>	P01	
<b>Nome</b>	Prenotazione camera	
<b>Attori</b>	Cliente	
<b>Goals</b>	Prenotare una stanza all'albergo della foresteria	
<b>Precondizioni</b>	Aver aggiornato calendario Aprire la finestra delle camere	
<b>Breve descrizione</b>	Il partecipante effettua la prenotazione di una stanza per pernottare alla foresteria	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	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
<b>Casi d'uso correlati e processi alternativi</b>	Processo alternativo: non ci sono camere libere, il partecipante non può prenotare la camera	
<b>Post condizioni</b>	Camera prenotata dal partecipante	

<b>Codice</b>	P02	
<b>Nome</b>	Prenotazione evento numero chiuso	
<b>Attori</b>	Partecipante	
<b>Goals</b>	Prenotare un evento a numero chiuso	
<b>Precondizioni</b>	A calendario vi sia un evento a numero chiuso creato da un noleggiatore	
<b>Breve descrizione</b>	Il partecipante effettua la prenotazione di evento a numero chiuso che si ospiterà nella foresteria in una determinata data	
<b>Processo</b>	<b>Azioni degli attori</b>	<b>Risposta del sistema</b>
	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
<b>Casi d'uso correlati e processi alternativi</b>	Processo alternativo: l'evento è full booked, il partecipante è escluso	
<b>Post condizioni</b>	Il partecipante è prenotato all'evento	

### 3.6 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.

<b>Codice Specifica</b>	<b>Descrizione</b>	<b>Priorità</b>	<b>Implementato</b>	<b>Codice User Case</b>
<b>R01</b>	<b>Gestire Sale:</b> l'applicazione deve consentire di aggiungere sale prenotabili	Alta	No	<b>RF01</b>

Codice Specifica	Descrizione	Priorità	Implementato	Codice User Case
R02	<b>Calendario di apertura della foresteria:</b> l'applicazione deve permettere la visualizzazione del calendario in cui la foresteria è aperta.	Alta	Si	RF02
R03	<b>Lista sale conferenza:</b> 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	<b>Lista camere:</b> 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	<b>Intervento da remoto sul server:</b> l'applicazione deve permettere funzioni volute dal responsabile della foresteria da remoto (lo spegnimento)	Bassa	Si	RF04
R06	<b>Prenotare evento:</b> 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	<b>Dichiarare evento a numero chiuso:</b> una volta creato un evento, si deve dare l'opzione di effettuarlo a numero chiuso.	Bassa	Si	N03
R08	<b>Servizio catering:</b> 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	<b>Prenotazione numero chiuso:</b> un cliente deve avere l'opzione di prenotarsi a un evento a numero chiuso;	Bassa	No	P02
R10	<b>Login/Logout:</b> in fase di login l'applicazione accetta 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

## 4 ITERAZIONE 2 AMDD

### 4.1 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

#### 4.1.1 codeMR

CodeMR consente l'analisi statica 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.

#### 4.1.2 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.

#### 4.1.3 MongoDB e AtlasDB

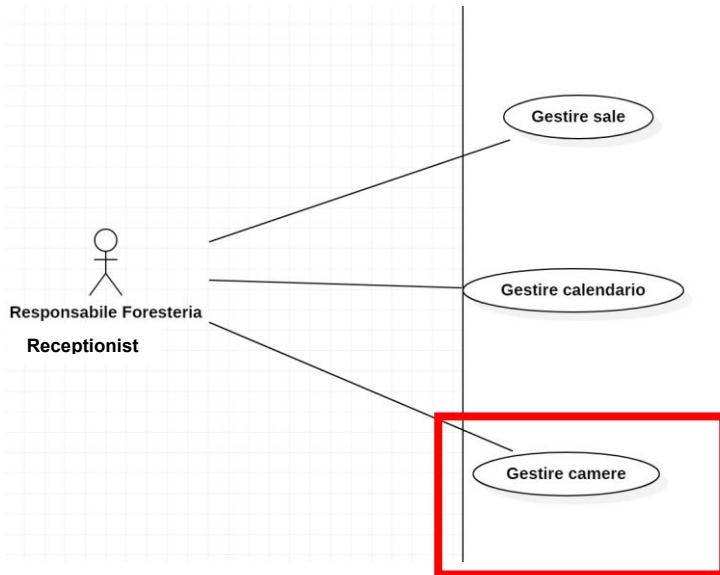
Visto che la 1a versione usava file json, abbiamo optato per un DMBS NoSQL; il passaggio da MongoDB (locale) ad AtlasDB (cloud) è stato immediato.

## 4.2 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.

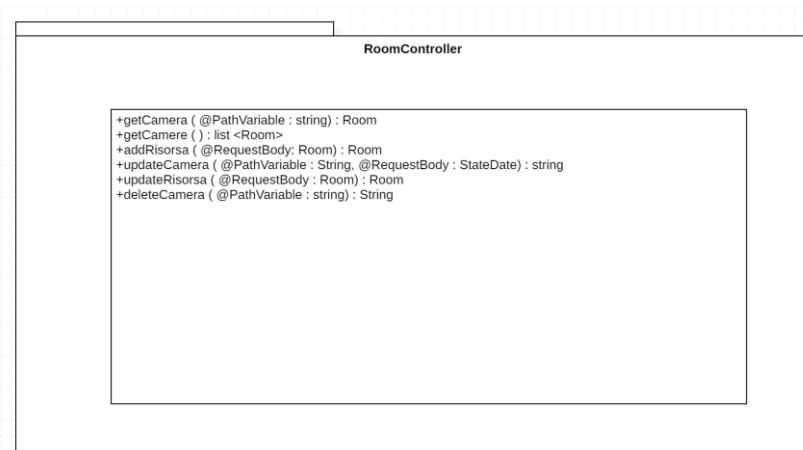
### 4.2.1 Selezione funzioni da implementare

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



### 4.2.2 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.



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 alle dependencies definite nel framework Spring Boot mentre la comunicazione fra Controller e Service è una semplice chiamata di un metodo.

Si rimanda al § 0

Class Diagrams per gli schemi definitivi.

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

La versione iniziale 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.

```
private void mettiDati() {
    // Invia una richiesta GET al server per ottenere i dati di tutte le camere
    ResourceRoom[] rooms = restTemplate.getForObject("http://localhost:8080/api/room", ResourceRoom[].class);

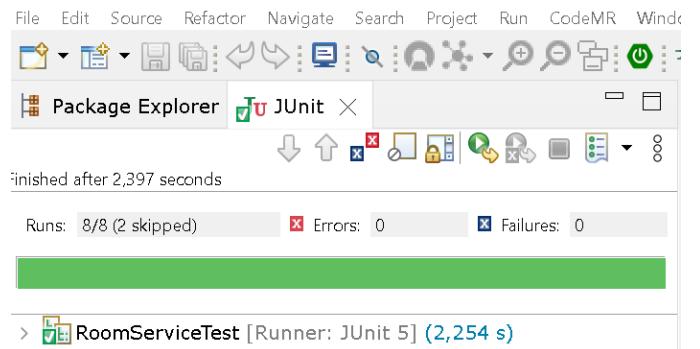
    // Crea un insieme di date uniche
    Set<String> uniqueDates = new HashSet<>();
    for (ResourceRoom room : rooms) {
        for (StateDate stateDate : room.getDisponibilita()) {
            uniqueDates.add(stateDate.getData().toString());
        }
    }
}
```

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.

### 4.2.4 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.



```
19 */
20 * Classe di test per il RoomService
21 */
22
23 @ExtendWith(MockitoExtension.class)
24 public class RoomServiceTest { //NOSONAR
25
26     private RoomService roomService;
27     private VisitorSetState visitorSet = new VisitorSetState();
28     private VisitorGetState visitorGet= new VisitorGetState();
29     private List <StateDate> disp = new ArrayList <>();
30     private StateDate stat01;
31     private StateDate stat02;
32
33     @Mock
34     private List<Room> sale;
35     private Gson gson = AppConfig.configureGson();
36     private MongoTemplate mongoDB= AppConfig.configureMongoDB();
37
38     @BeforeEach
39     void setUp() {
40         MockitoAnnotations.openMocks(this);
41         roomService = new RoomService(gson, mongoDB);
42     }
```

```

41 Room creaRoom(RoomType tipo, State oggiStato, State domaniStato) {
42     Room camera = new Room();
43     stato1 = new StateDate(LocalDate.now(), oggiStato);
44     disp.add(stato1);
45     if (domaniStato != null){
46         stato2 = new StateDate(LocalDate.now().plusDays(1), domaniStato);
47         disp.add(stato2);
48     }
49     camera.setDisponibilita(disp);
50     camera.setNumeroLetti(2);
51     camera.setCosto(1234.);
52     camera.setTipo(tipo);
53     return camera;
54 }

43
44 Room creaRoom(RoomType tipo, State oggiStato, State domaniStato) {
45     Room camera = new Room();
46     stato1 = new StateDate(LocalDate.now(), oggiStato);
47     disp.add(stato1);
48     if (domaniStato != null){
49         stato2 = new StateDate(LocalDate.now().plusDays(1), domaniStato);
50         disp.add(stato2);
51     }
52     camera.setDisponibilita(disp);
53     camera.setNumeroLetti(2);
54     camera.setCosto(1234.);
55     camera.setTipo(tipo);
56     return camera;
57 }
58
59 @Test
60 void GsonTest() {
61     Room camera = creaRoom(RoomType.SUITE, State.INUSO, State.PRENOTATA);
62     String eventJson = gson.toJson(camera);
63     Room deserializedHall = gson.fromJson(eventJson, Room.class);
64     assertEquals(camera, deserializedHall);
65 }
66
67 @Test
68 void getRisorseTest() {
69     List<Room> camere = roomService.getRisorse();
70     assertFalse(camere.isEmpty());
71     System.out.println("Numero di camere: " + camere.size());
72     for (Room camera : camere) {
73         assertNotNull(camera.getNome());
74         assertFalse(camera.getNome().isEmpty());
75     }
76 }
77
78 @Test
79 void getRisorsaTest() {
80     Room camera = roomService.getRisorsa("Room001");
81     assertNotNull(camera);
82     assertEquals("Room001", camera.getNome());
83 }
84
85 @Test
86 void UpdateRisorsaTest() {
87     Room camera = roomService.getRisorsa("Room001");
88     assertNotNull(camera);
89     assertEquals("Room001", camera.getNome());
90     Double oldCosto = camera.getCosto();
91     camera.setCosto(1000.);
92     roomService.updateRisorsa(camera);
93     camera = roomService.getRisorsa("Room001");
94     assertNotNull(camera);
95     assertEquals("Room001", camera.getNome());
96     assertEquals(1000., camera.getCosto());
97     camera.setCosto(oldCosto);
98 }
99
100 @Test
101 @Disabled("Test disabilitato a causa della dipendenza dal singleton.")
102 void addRisorsaTest() { //NOSONAR
103     // eseguito solo per Eventi
104 }

```

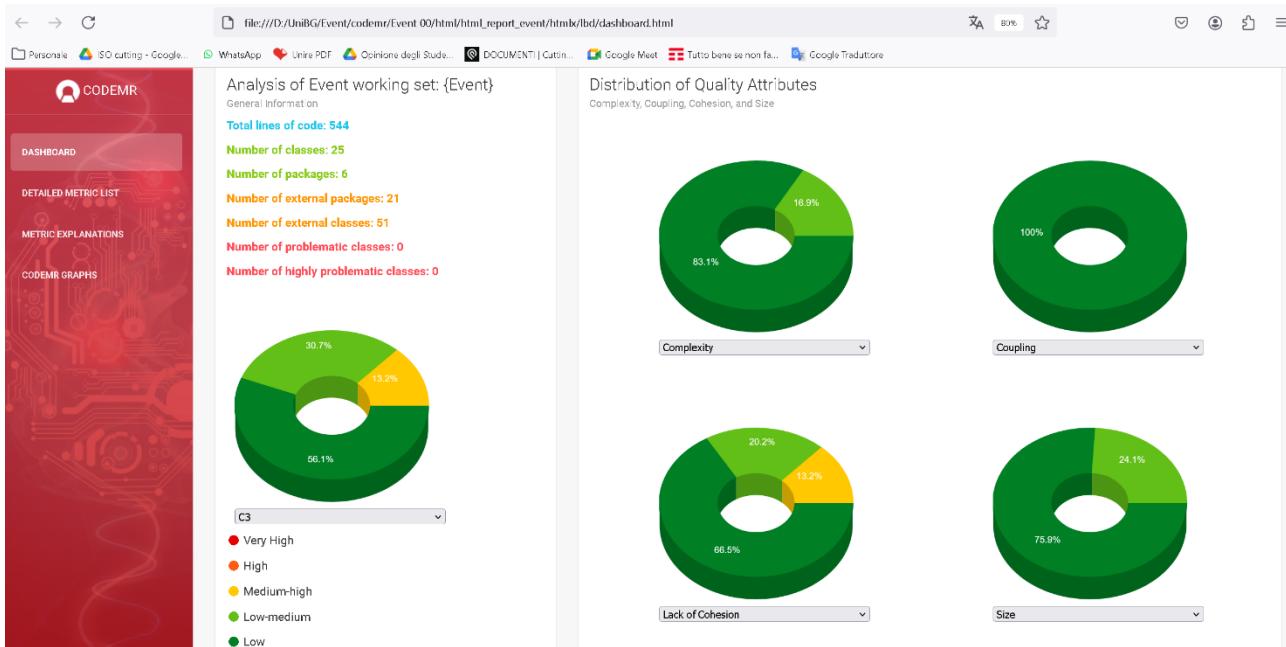
```

105
106 @Test
107 @Disabled("Test disabilitato a causa della dipendenza dal singleton.")
108 void DeleteRisorsaTest() { //NOSONAR
109     // eseguito solo per Eventi
110 }
111
112 @Test
113 void changeStatoCameraTest() {
114     Room camera = creaRoom(RoomType.SUITE, State.INUSO, State.PRENOTATA);
115     StateDate old = new StateDate(LocalDate.now(), State.INUSO);
116     StateDate nuovo = new StateDate(LocalDate.now(), State.DISPONIBILE);
117     assertEquals(State.INUSO, visitorGet.visit(camera, old));
118     visitorSet.visit(camera, nuovo);
119     assertEquals(State.DISPONIBILE, visitorGet.visit(camera, nuovo));
120 }
121
122 @Test
123 void changeDisponibilitaTest() {
124     // Crea un mock di RoomService
125     RoomService roomServiceMock = mock(RoomService.class);
126     // Configura il mock per restituire una lista di camere quando viene chiamato getRisorse()
127     Room camera1 = creaRoom(RoomType.SUITE, State.INUSO, State.PRENOTATA); // in disp ha domani
128     Room camera2 = creaRoom(RoomType.NORMAL, State.DISPONIBILE, State.DISPONIBILE); // in disp ha domani
129     Room camera3 = creaRoom(RoomType.NORMAL, State.DISPONIBILE, null); // in disp non ha domani
130     List<Room> camereMock = Arrays.asList(camera1, camera2, camera3);
131     // Configura il mock per restituire una lista di nomi di camere quando viene chiamato changeDisponibilita()
132     List<String> nomiCamere = Arrays.asList(null, null);
133     when(roomServiceMock.changeDisponibilita(any())).thenReturn(nomiCamere);
134     // Esegue il test utilizzando il mock di RoomService
135     List<StateDate> newDisp = new ArrayList<>();
136     newDisp.add(new StateDate(LocalDate.now().plusDays(1), State.CHIUSO));
137     List<String> camereToBeResc = roomServiceMock.changeDisponibilita(newDisp);
138     // Verifica che il numero totale di camere date al mock sia 3
139     assertEquals(3, camereMock.size());
140     // Verifica che il numero di camere restituite da changeDisponibilita() sia 2
141     assertEquals(2, camereToBeResc.size());
142     // Verifica che i nomi delle camere restituite da changeDisponibilita() siano tutti null
143     assertTrue(camereToBeResc.stream().allMatch(Objects::isNull));
144 }
145
146 }
147

```

## 4.2.5 Analisi statica

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



## 4.2.6 Maschera delle camere

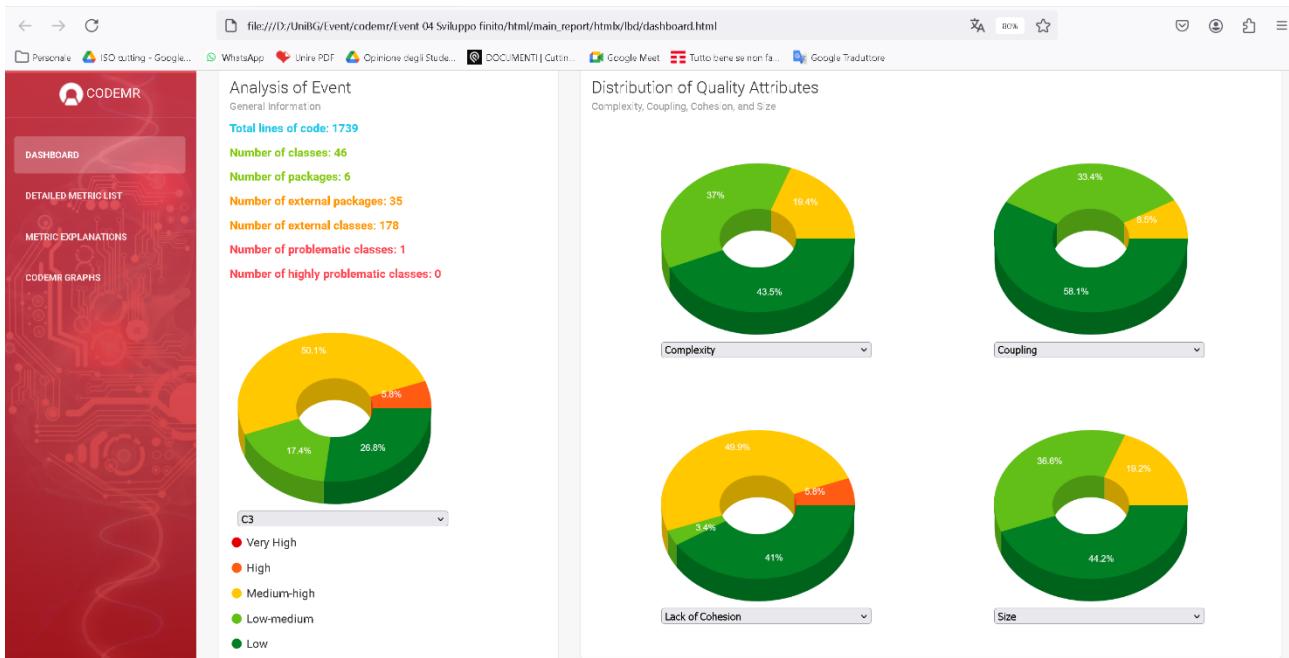
Visualizzazione Camere										
Camera	Tipo	Costo	N. Letti	2024-01-01	2024-01-02	2024-01-03	2024-01-04	2024-01-05		
Room001	SUITE	120.0	2	INUSO	DISPONIBILE	PULIZIA				
Room002	MINISUITE	120.0	2	DISPONIBILE	PRENOTATA	PRENOTATA				
Room003	SUITE	100.0	2	PRENOTATA	DISPONIBILE	INUSO				
Room004	ROYALE	120.0	3	DISPONIBILE	CHIUSO	PULIZIA				
Room005	ROYALE	120.0	3			PRENOTATA	DISPONIBILE	PRENOTATA		

## 4.2.7 Conclusioni iterazione 2

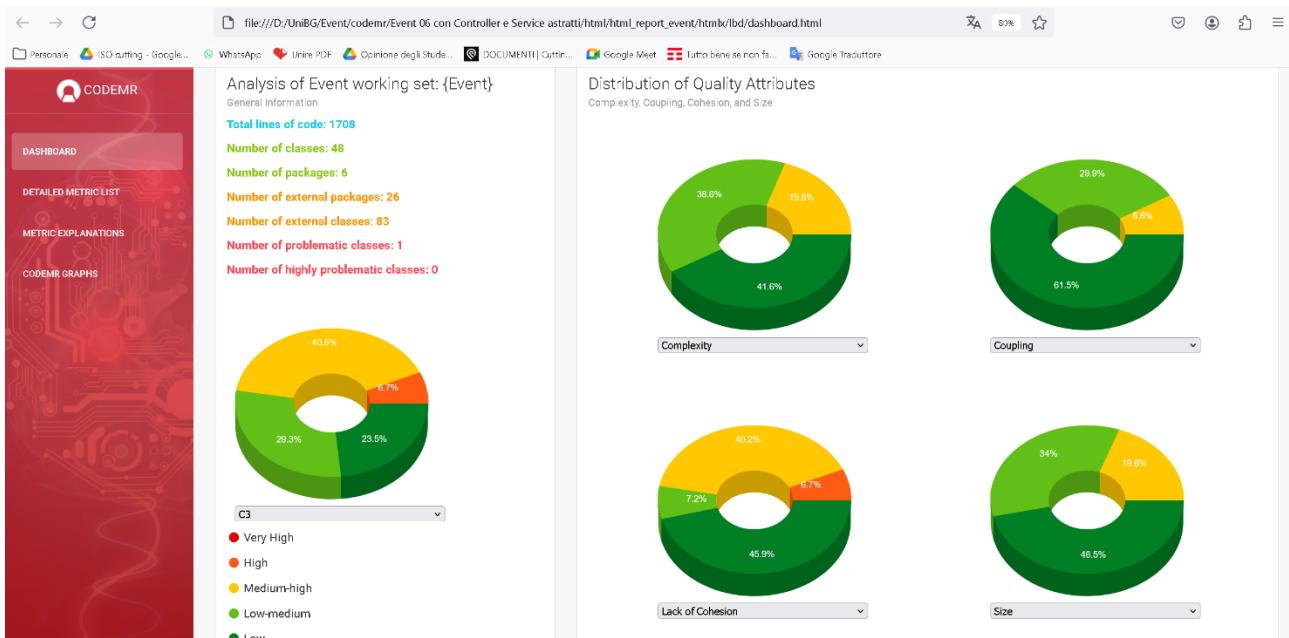
Cod. Specifica	Descrizione	Implementato	Cod. User Case
R01	Gestire Sale	No	RF01
R02	Calendario di apertura della foresteria	No	RF02
R03	Lista sale conferenza	No	RF01
R04	Lista camere	Si	RF03 P01 AP01
R05	Intervento da remoto sul server	No	RF04
R06	Prenotare evento	No	N01, N02
R07	Dichiarare evento a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

## 5 ITERAZIONI INTERMEDI

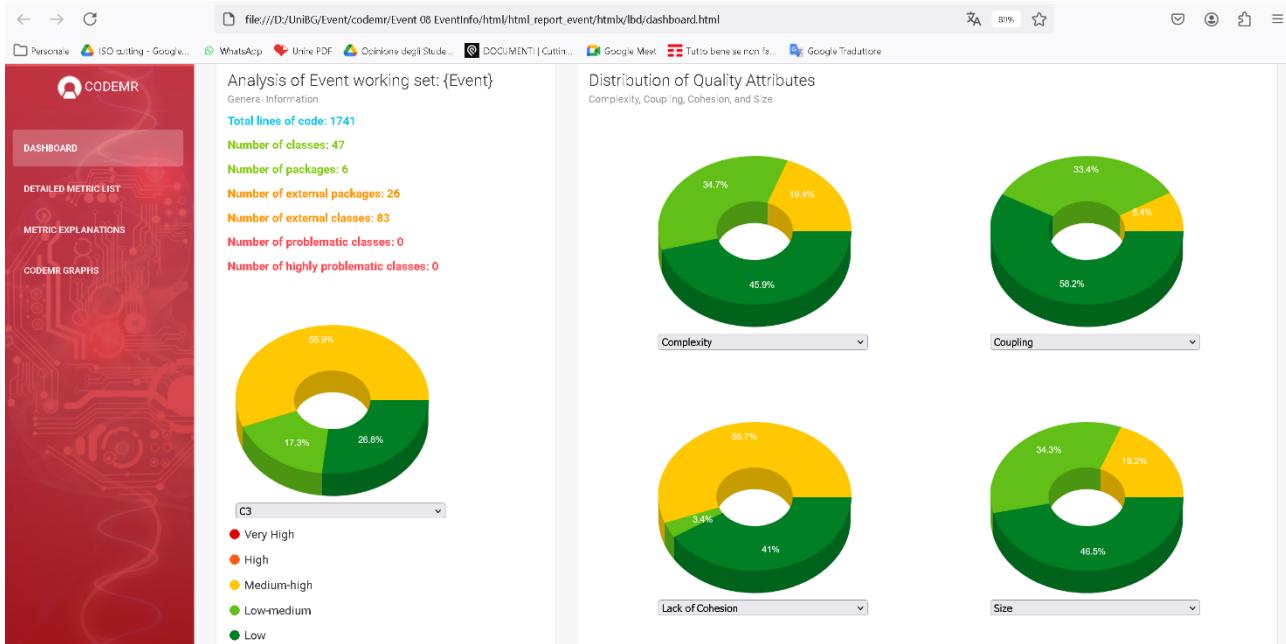
Al termine della iterazione 4 abbiamo completato lo sviluppo degli user case che ci eravamo prefissi, ma l'analisi statica eseguito con codeMR mostrava problemi di coesione di tipo "grave"



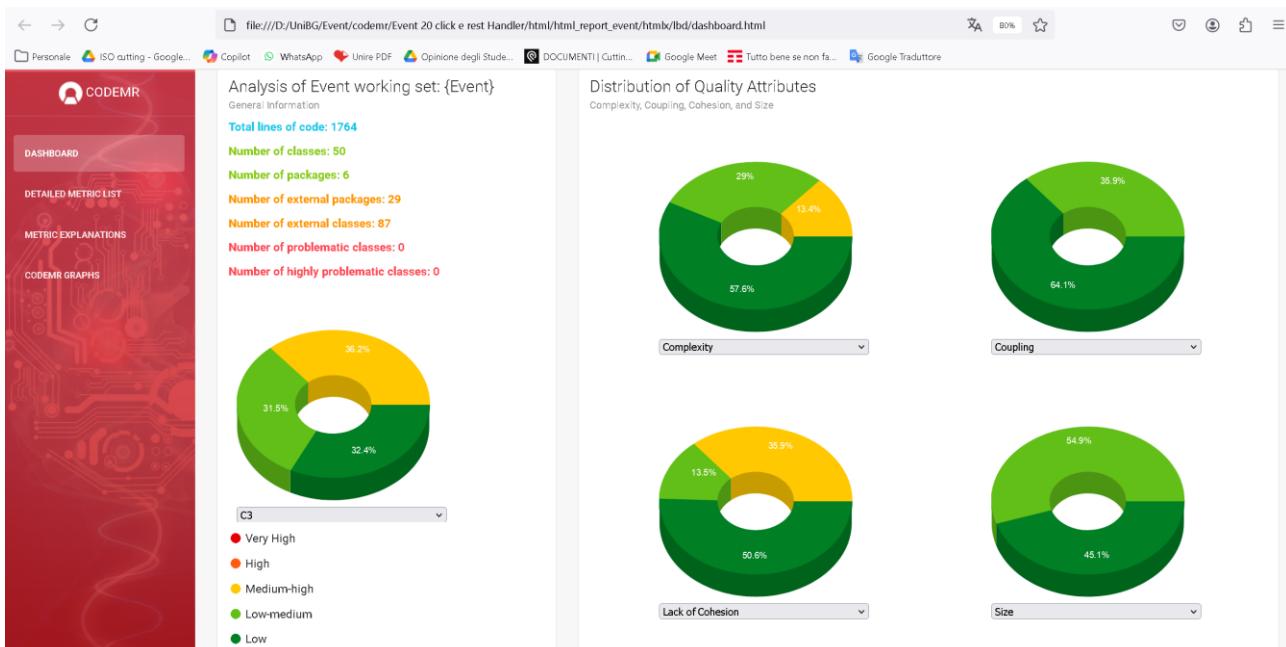
Abbiamo, quindi, cominciato il re-factoring del codice dapprima rivedendo le classi del server introducendo una classe astratta e il pattern observer. Come si può vedere, tale intervento ha praticamente dimezzato il numero di classi esterne e il numero di package esterni, ma non è servito a migliorare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

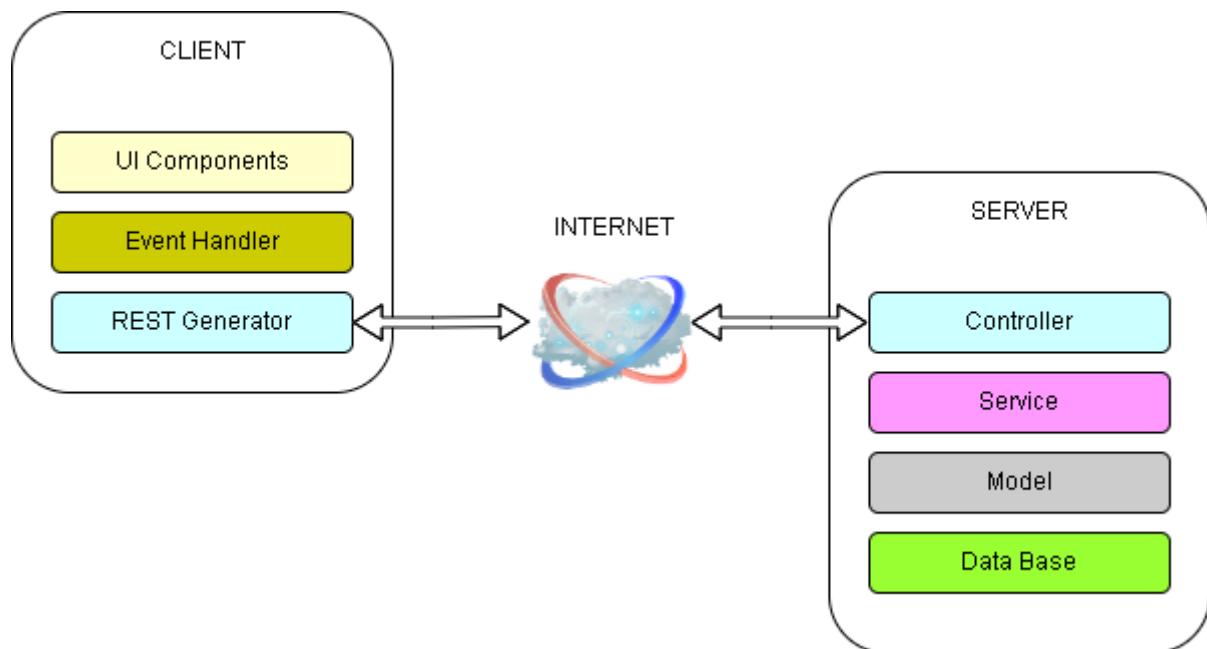


# 6 ITERAZIONE FINALE

## 6.1 Architectural Pattern

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

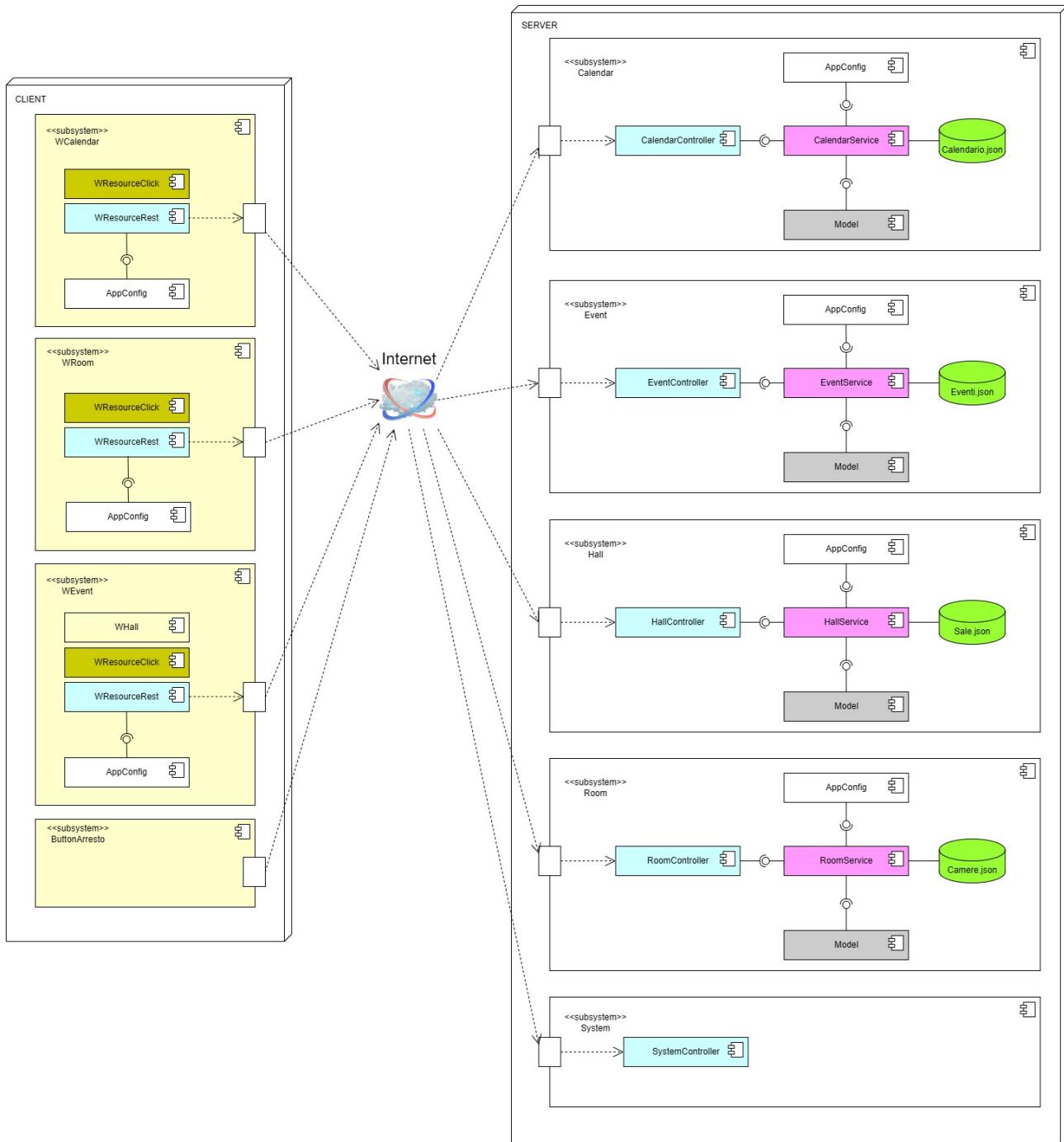
- La struttura 2 tiers (il modello e i dati sono nello stesso tier, il SERVER)
- La comunicazione via internet ad opera di un REST Generator e di un Controller (basati sul framework SpringBoot)
- L'interfaccia utente (realizzata usando JavaFX) è tenuta separata dal codice che gestisce gli eventi (e che 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)

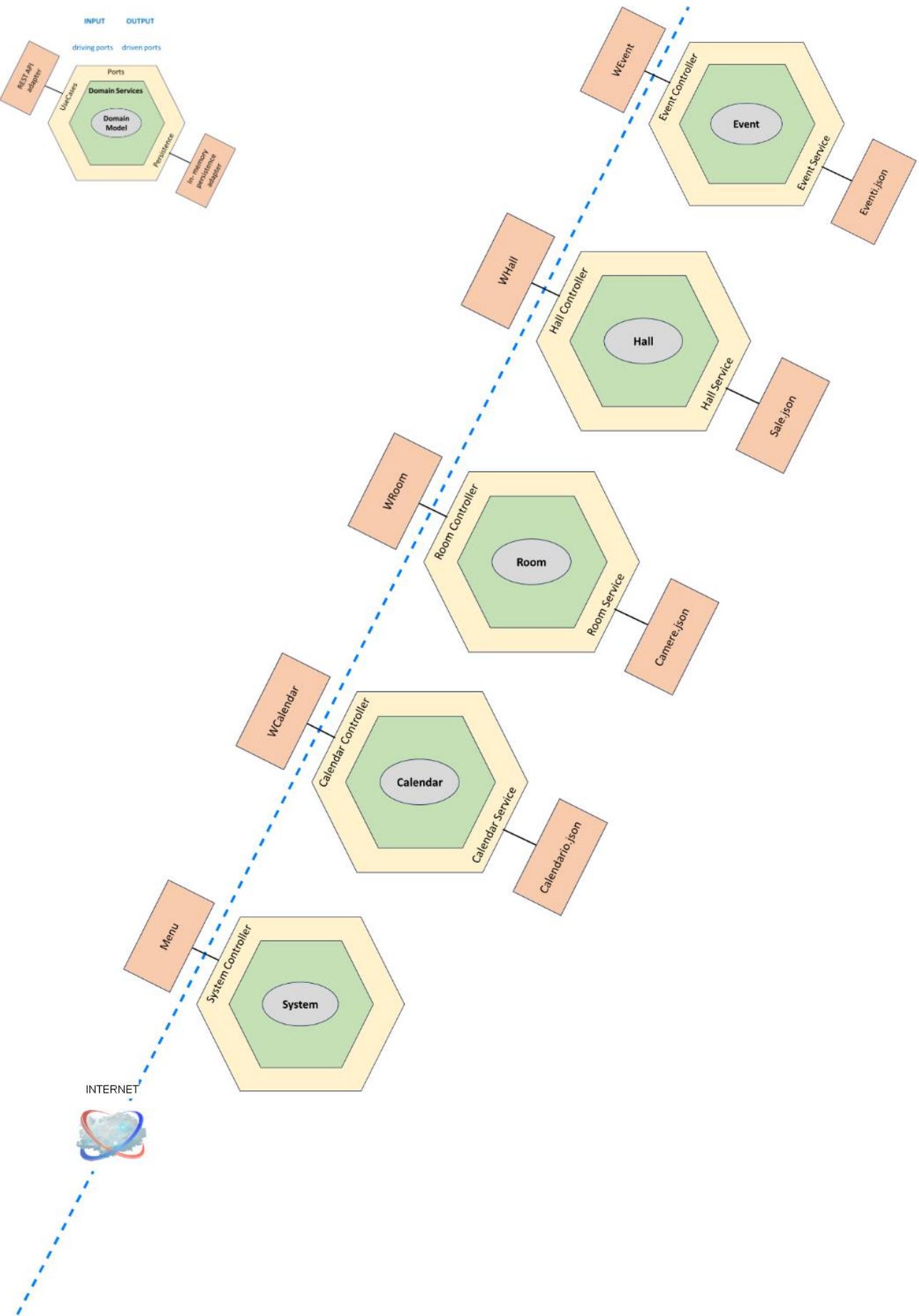


## 6.2 Formal Deployment Diagram

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).

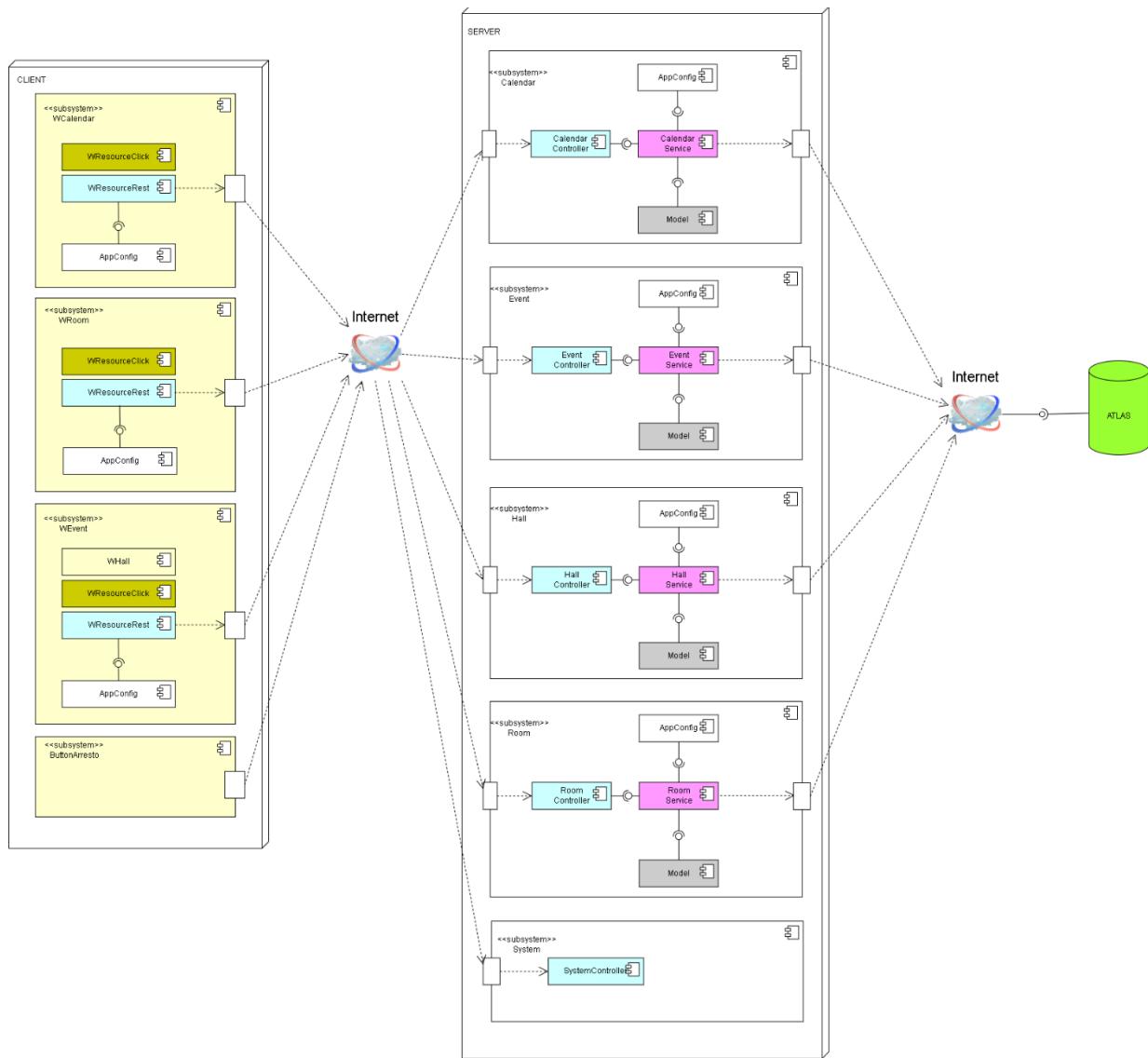
### 6.2.1 Versione 1 (two tiers)

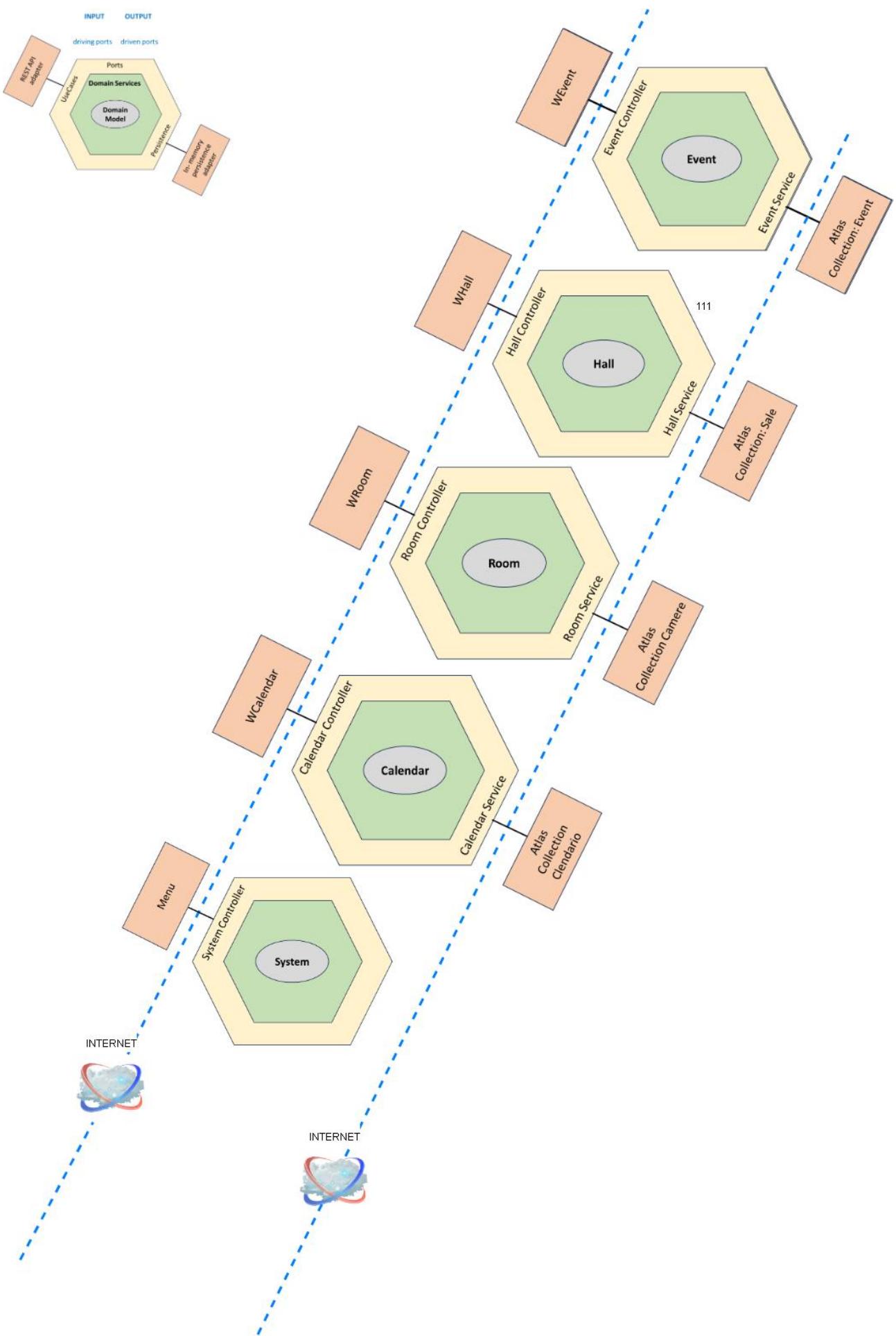




### **6.2.2 Versione 2 (three tiers)**

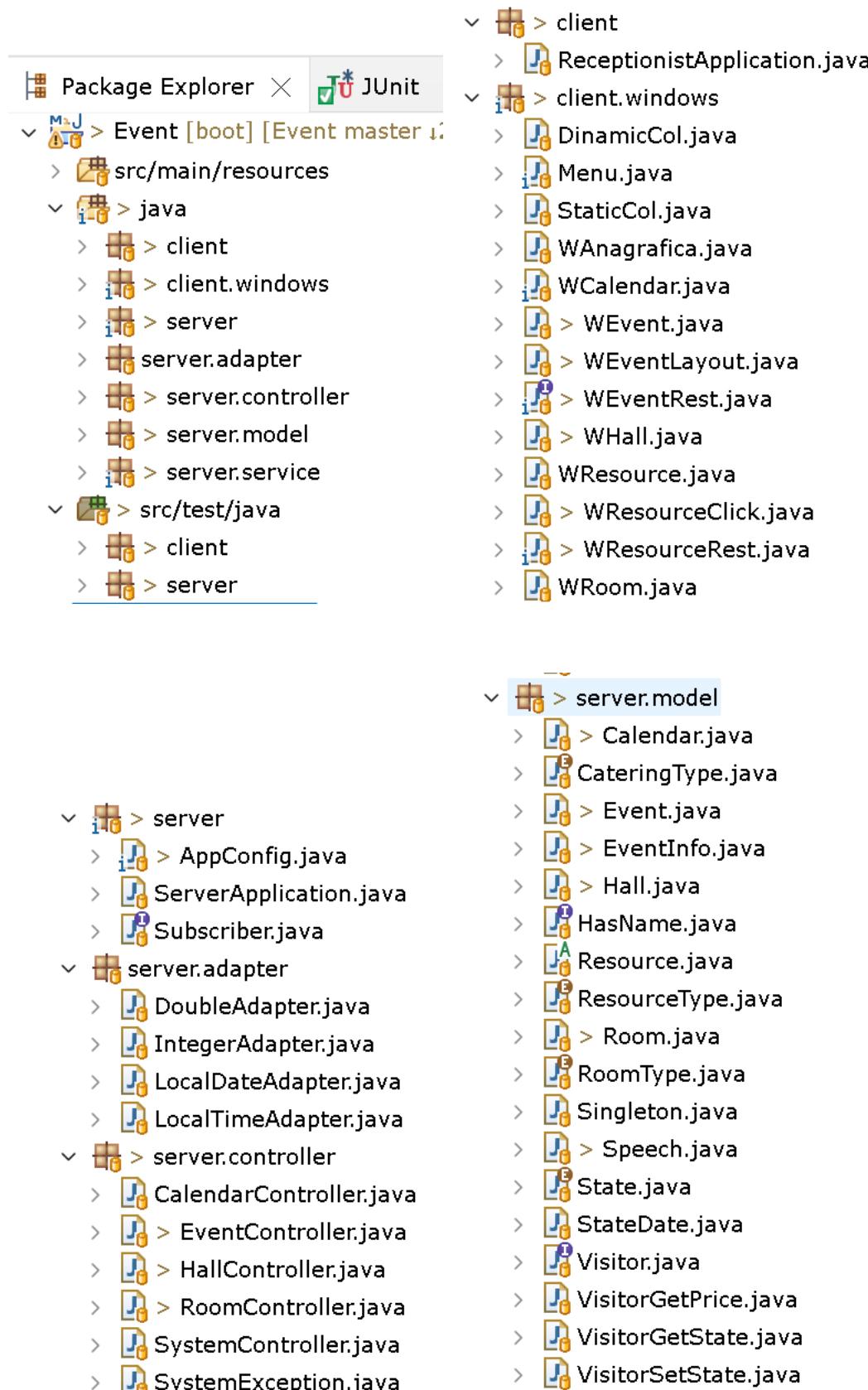
La modularità del progetto ha consentito il passaggio dai file json a MongoDB modificando solo il layer service. La versione 2, attraverso un flag in AppConfig è in grado di funzionare sia nella configurazione 2tiers che 3 tiers





## 6.3 Project Tree

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

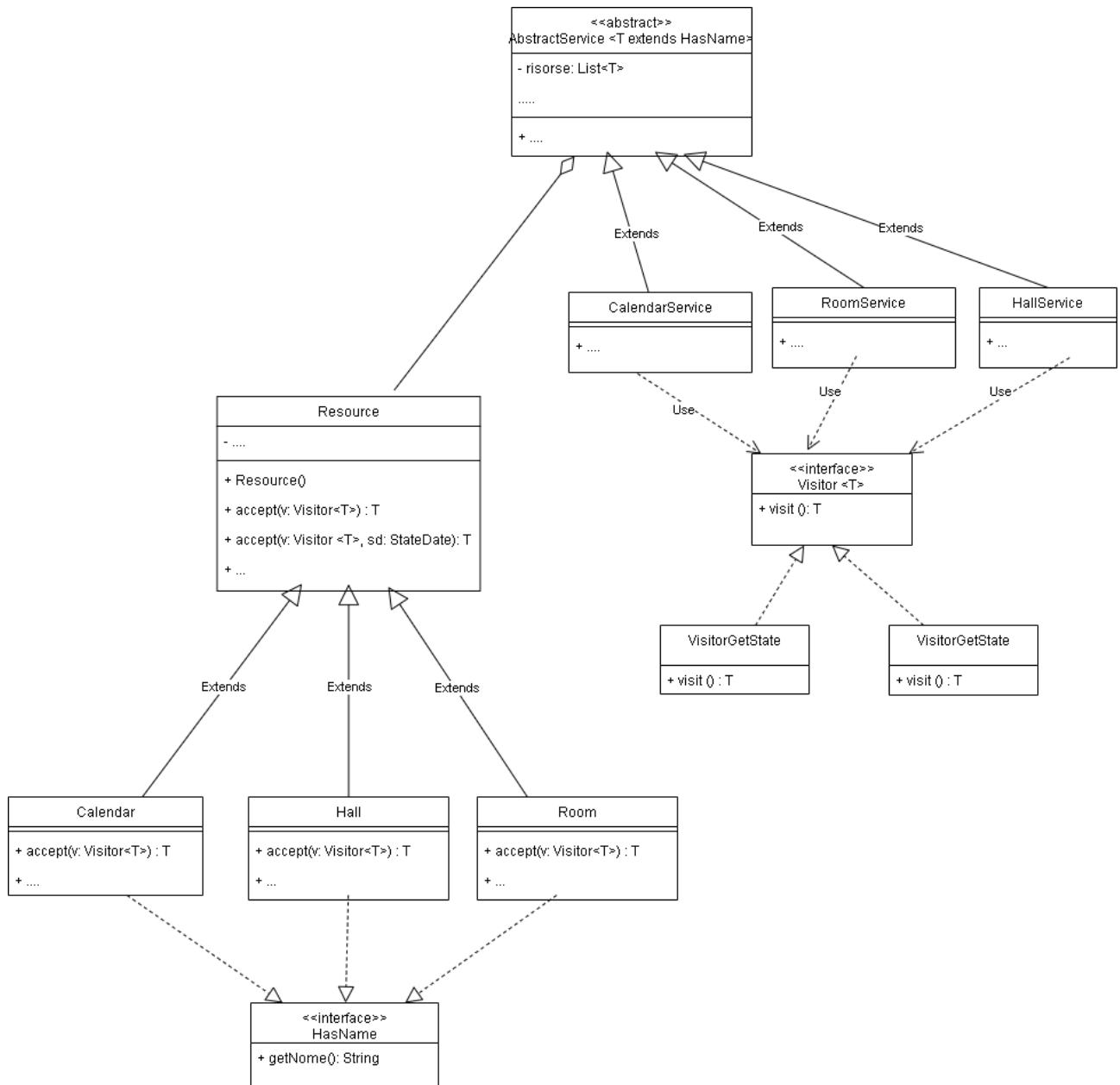


```
✓   > server.service
  >  > AbstractService.java
  >  > CalendarService.java
  >  > EventService.java
  >  > HallService.java
  >  > RoomService.java
  >  > SingletonService.java
✓   > src/test/java
  ✓   > client
    >  > ReceptionistApplicationTests.java
  ✓   > server
    >  > EventControllerTests.java
    >  > EventServiceTest.java
    >  > HallServiceTest.java
    >  > RoomControllerTests.java
    >  > RoomServiceTest.java
    >  > VisitorTest.java
```

## 6.4 Class Diagram (Design Pattern)

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

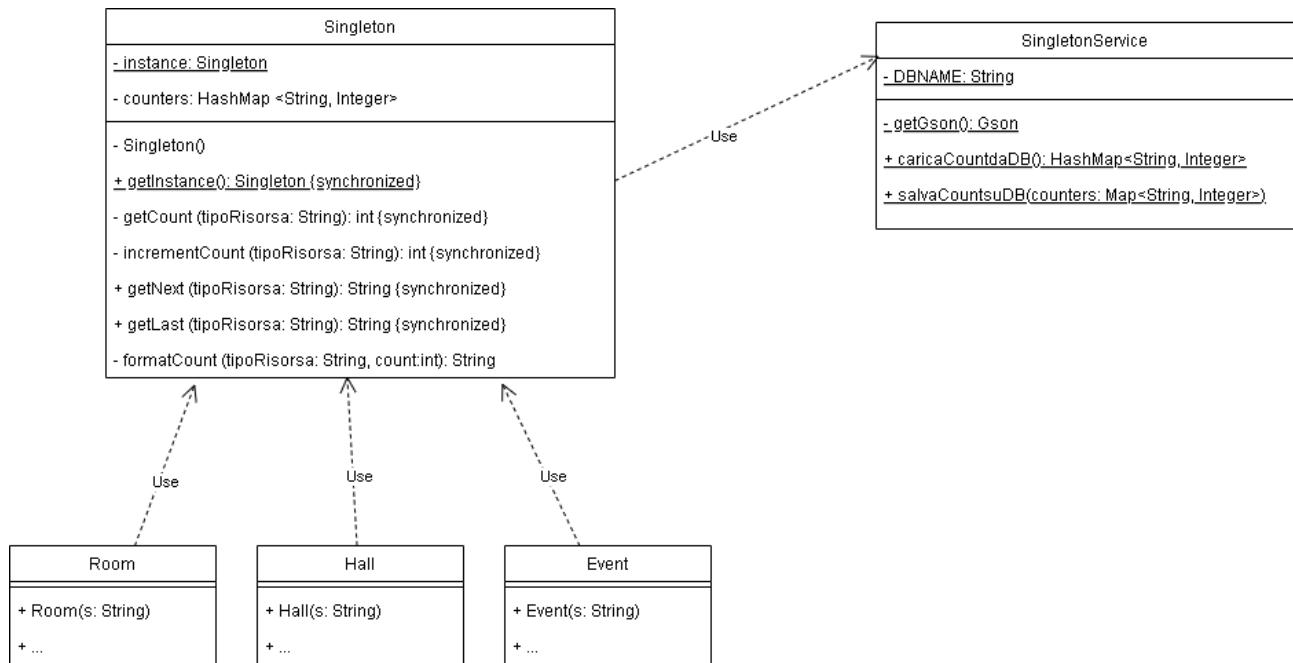
### 6.4.1 Visitor



```

1 package server.model;
2
3
4 /**
5 * Interfaccia utilizzata per garantire che le classi che la implementano
6 * abbiano un metodo getName(). Questo metodo è utilizzato nella classe
7 * AbstractService per identificare univocamente ciascuna risorsa.
8 */
9
10 public interface HasName {
11
12     public String getName();
13
14 }
15
  
```

## 6.4.2 Singleton



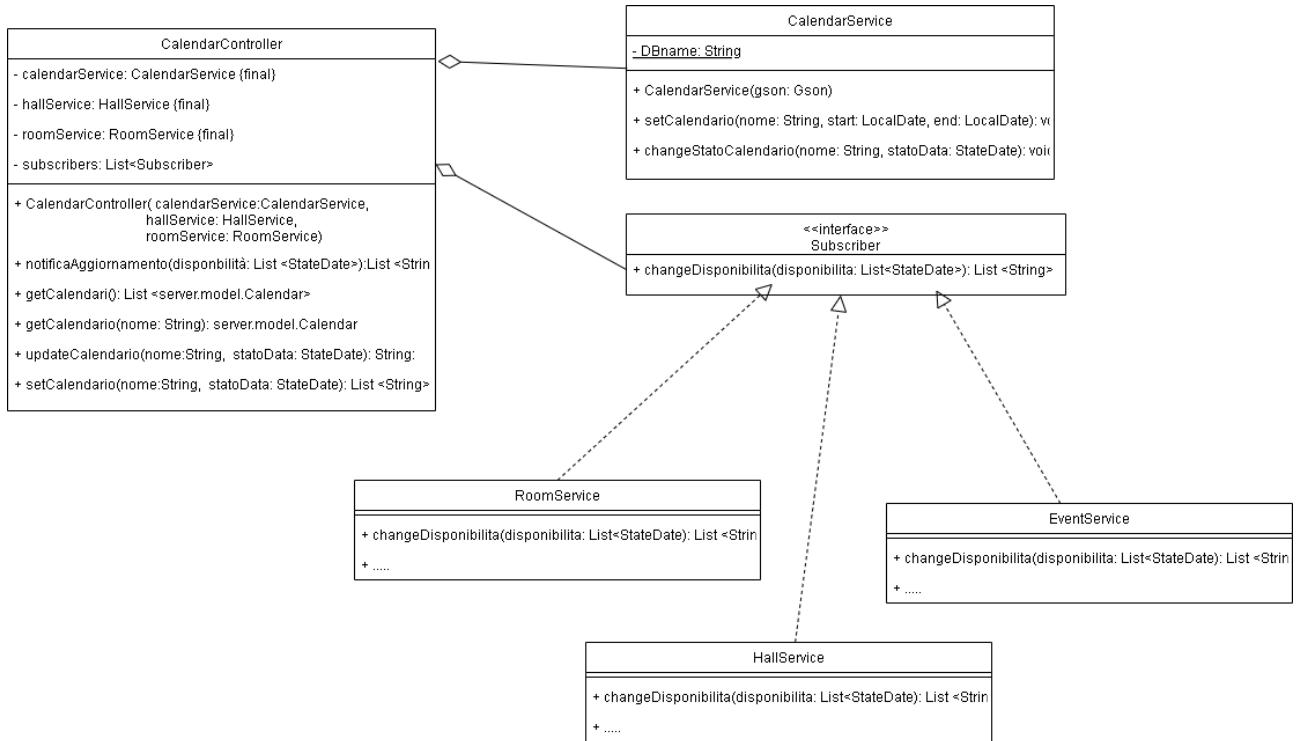
La classe SingletonService s'è resa necessaria per poter gestire il file `Songleton.json` volto a memorizzare i contatori dei vari tipi di Resource.

```

{
  "Calendar": 1,
  "Event": 7,
  "Hall": 3,
  "Room": 5
}
  
```

A screenshot of a JSON editor window titled "Singleton.json". The window has a menu bar with "File", "Modifica", and "Visualizza". The main area displays a JSON object with four properties: "Calendar" (value 1), "Event" (value 7), "Hall" (value 3), and "Room" (value 5). The JSON is formatted with indentation and line breaks.

### 6.4.3 Observer

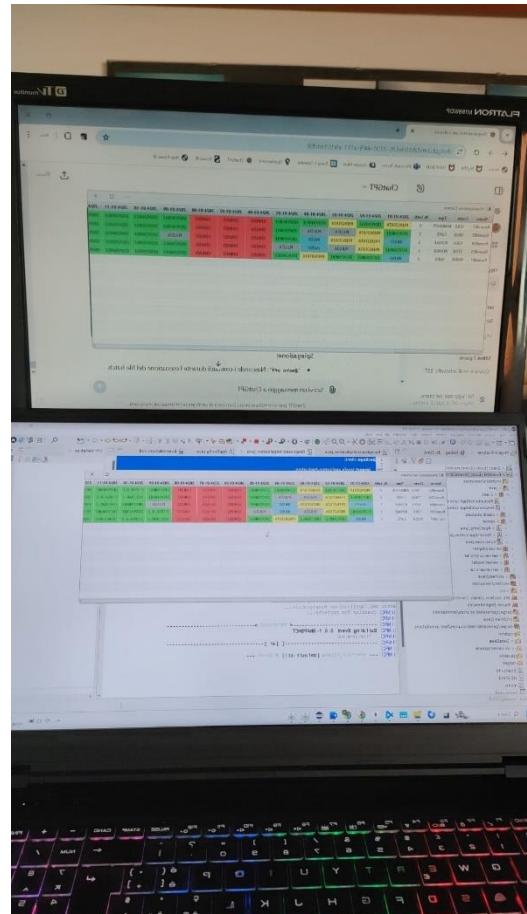


L'Observer è utilizzato per notificare a tutti i service che è stato modificato il calendario, a seguito di tale notifica si procede al cambio degli stati delle risorse e alla generazione dei report utili per comunicare ai clienti/partecipanti/noleggiatori eventuali disdette delle prenotazioni per gravi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

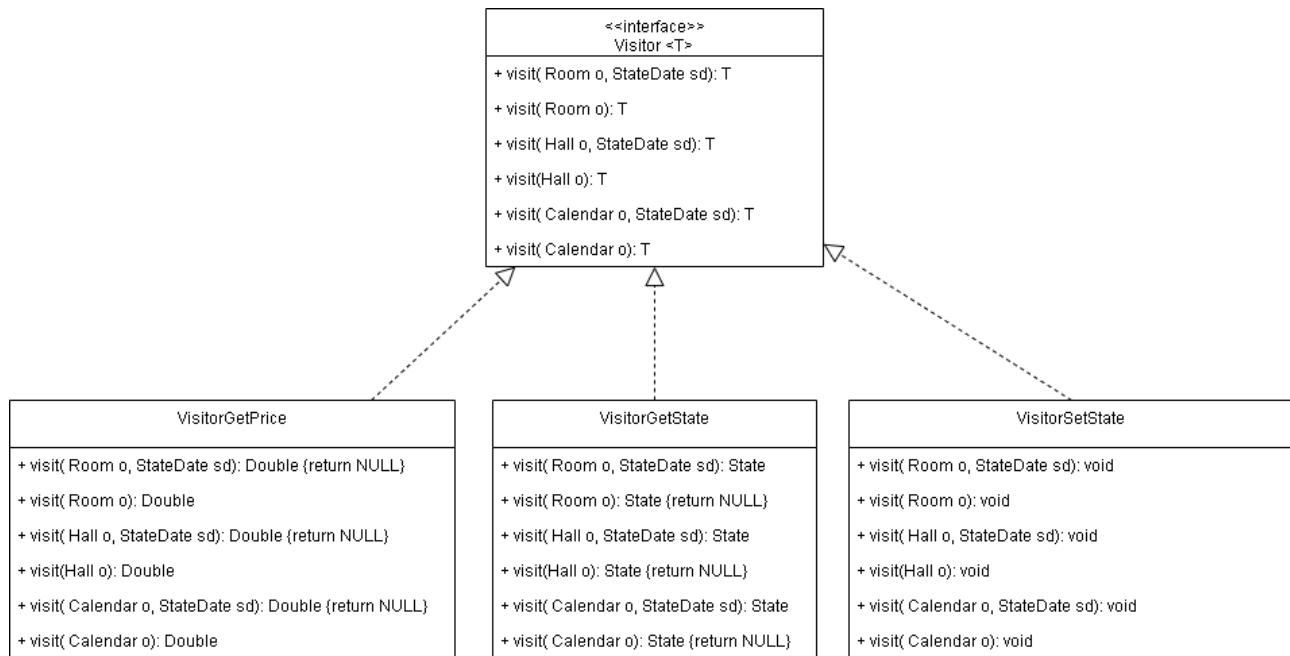
1. L'inizializzazione di un client dovrebbe utilizzare l'iniezione fornita da Spring per ottenere le istanze dei controller e chiedere di essere inserito nella lista degli Observer.
2. I vari controller aggiornano le rispettive liste con i riferimenti alle varie WResource relative
3. Ogni qualvolta che 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.



## 6.5 Class Diagrams

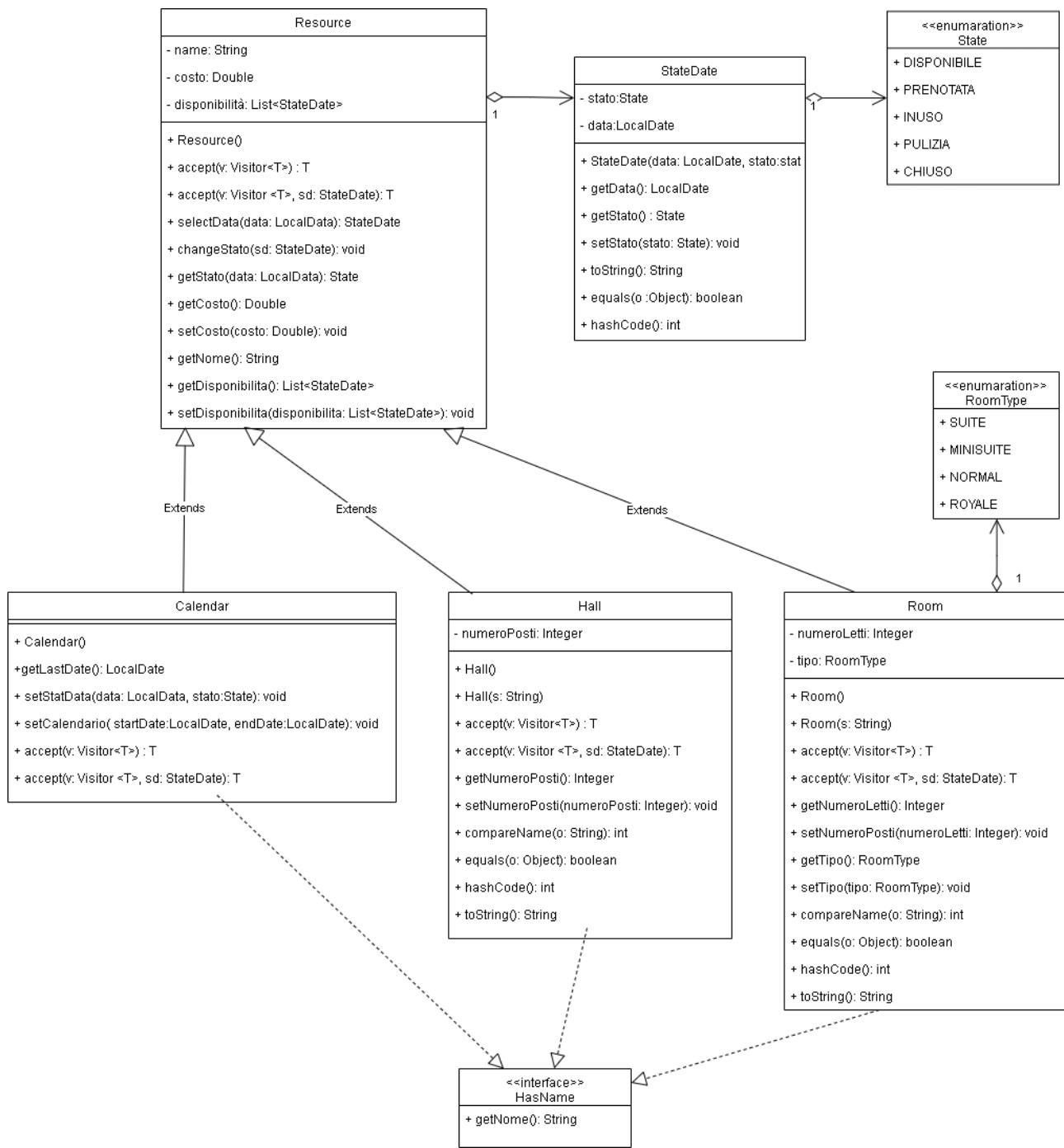
### 6.5.1 Class: Visitor



```
1 package server.model;
2
3 /**
4  * Interfaccia Visitor
5  * ha visit overloaded per consentire l'implementazione
6  * anche del visitor che hanno lo StatoData come parametro
7 */
8
9 public interface Visitor <T>{
10
11     public T visit (Room o, StateDate sd);
12     public T visit (Room o);
13
14     public T visit (Hall o, StateDate sd);
15     public T visit (Hall o);
16
17     public T visit (Calendar o, StateDate sd);
18     public T visit (Calendar o);
19
20
21 }
```

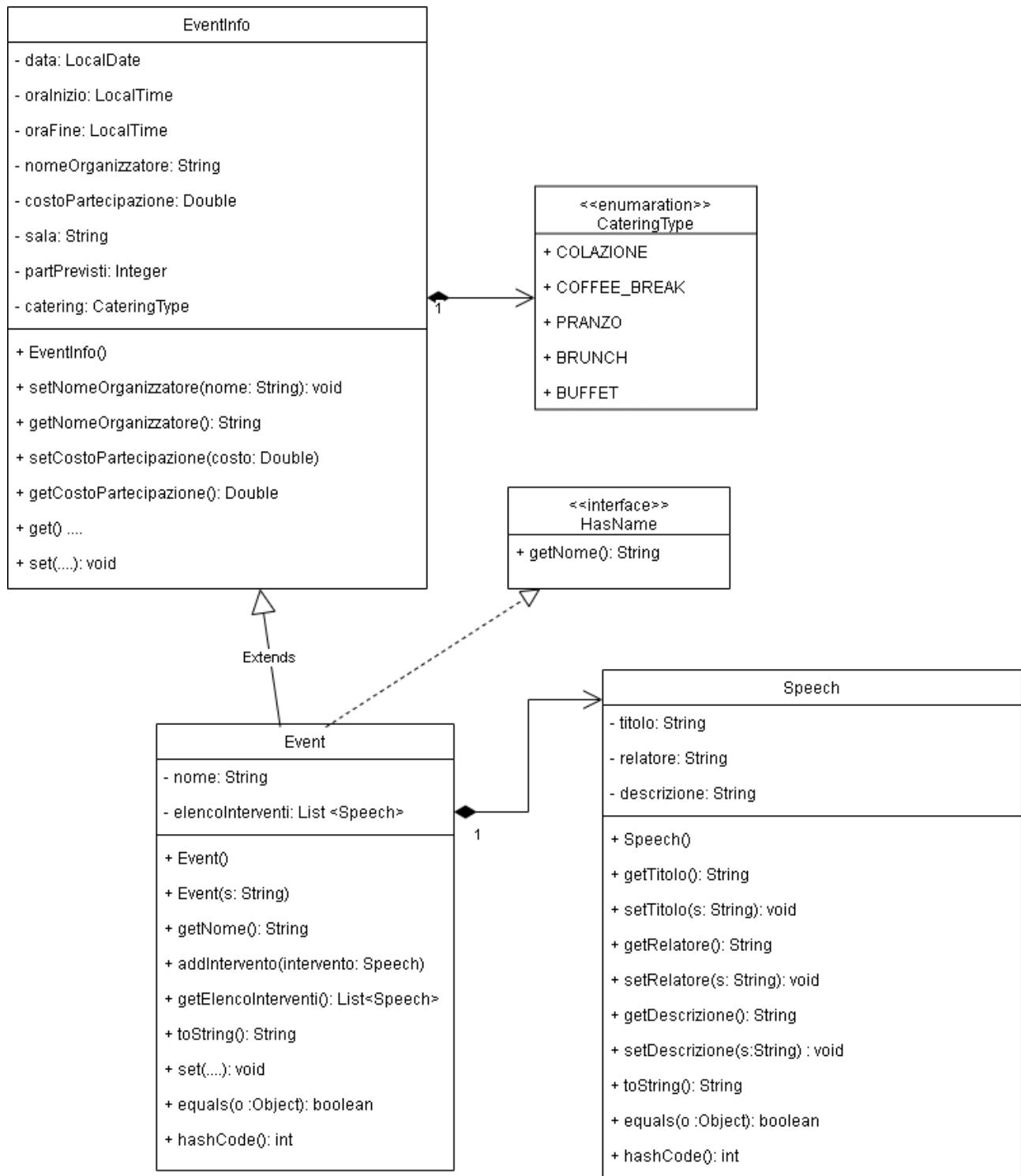
Il pattern Visitor è presente sin dalla prima stesura del codice.

## 6.5.2 Class: Resource

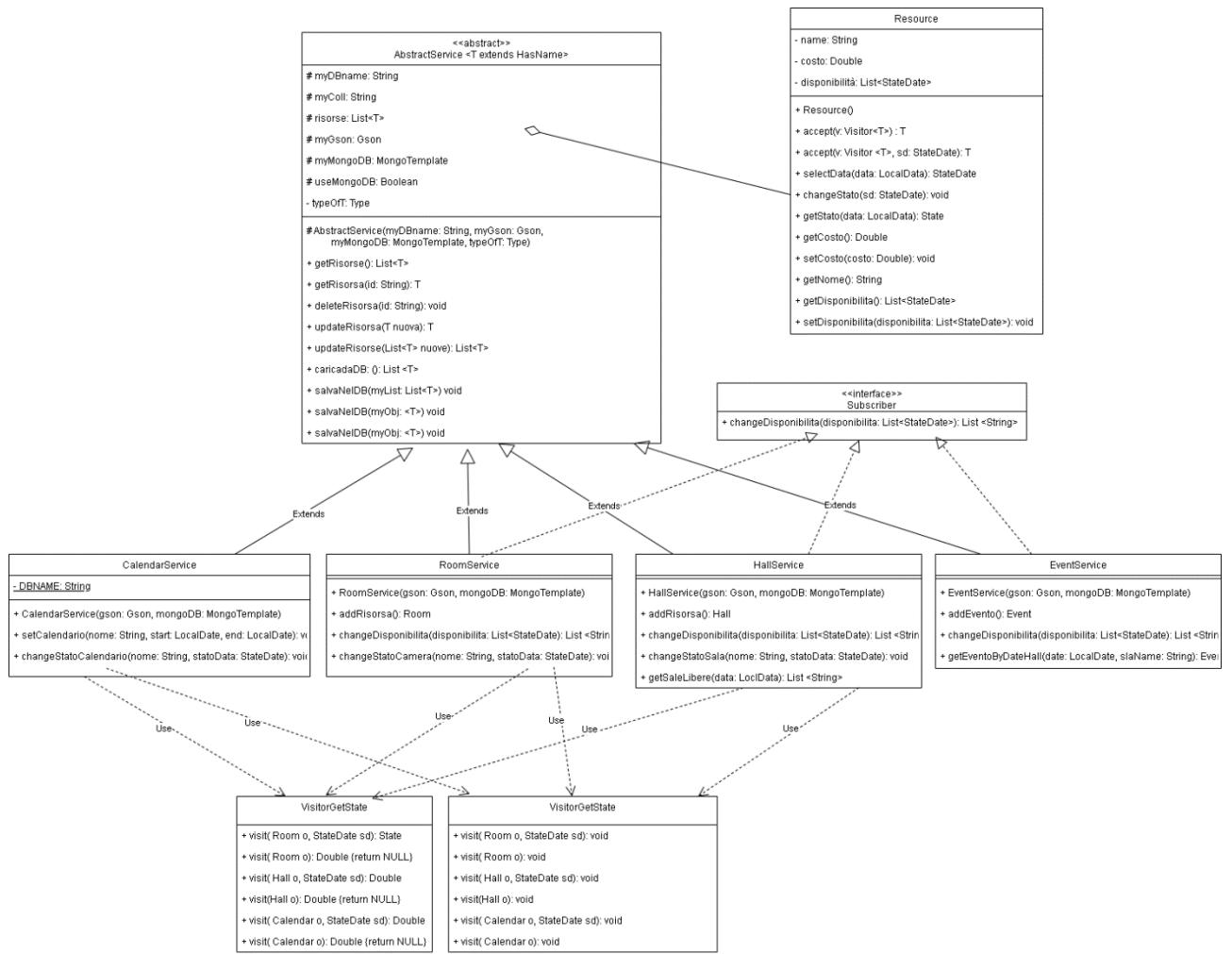


La Classe Resource è presente sin dalla prima stesura del codice.

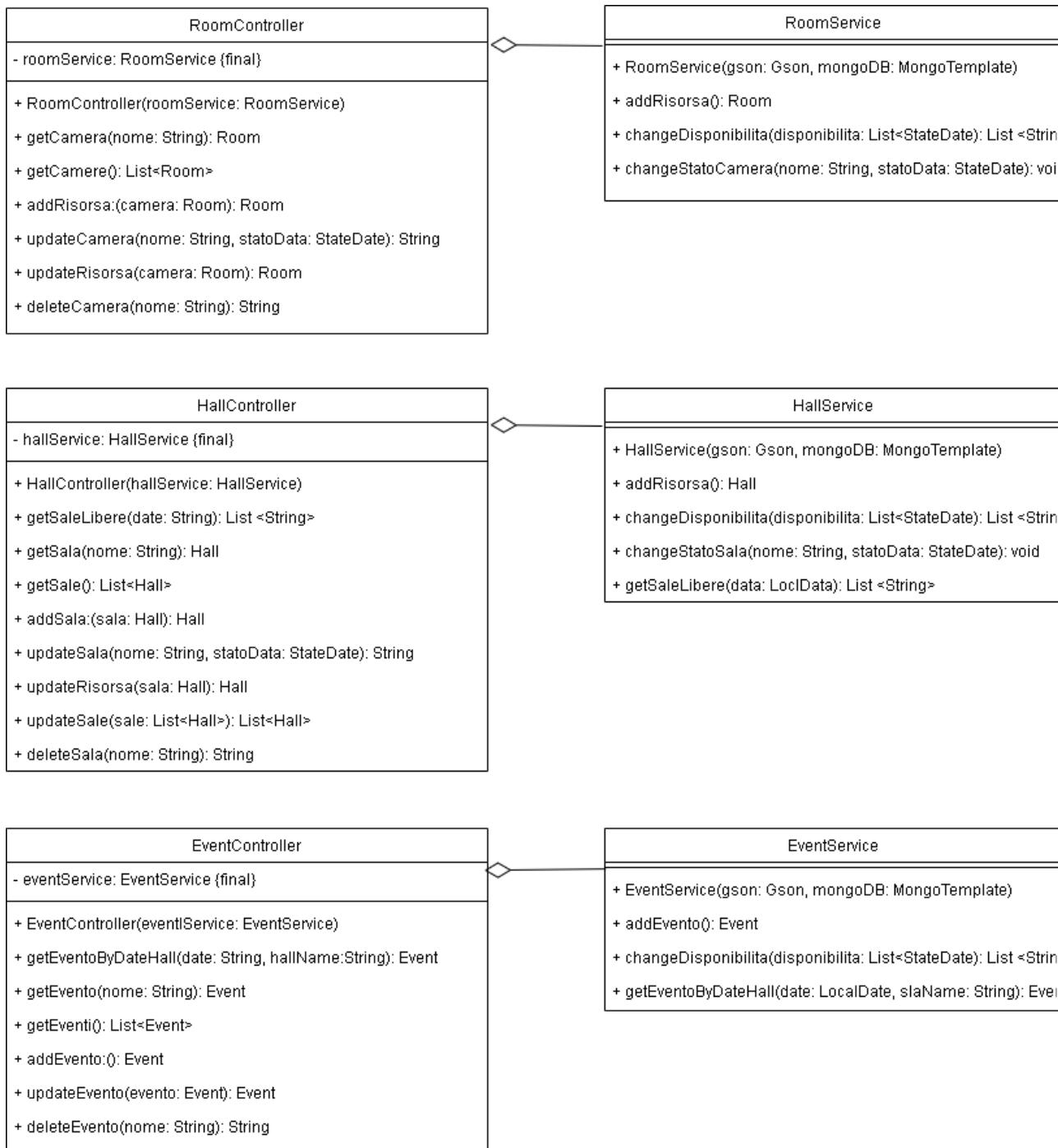
### 6.5.3 Class: Event



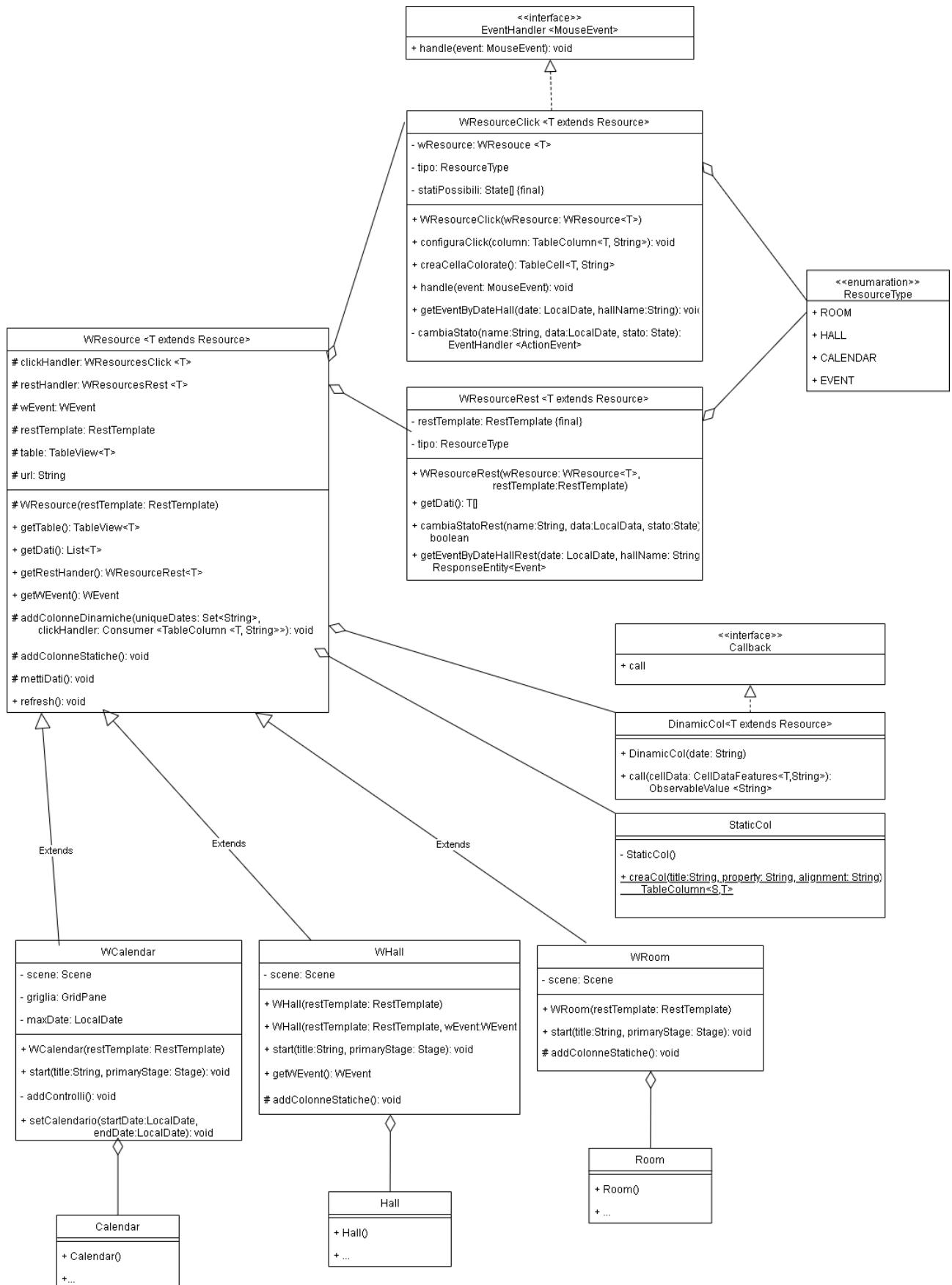
## 6.5.4 Class: Service



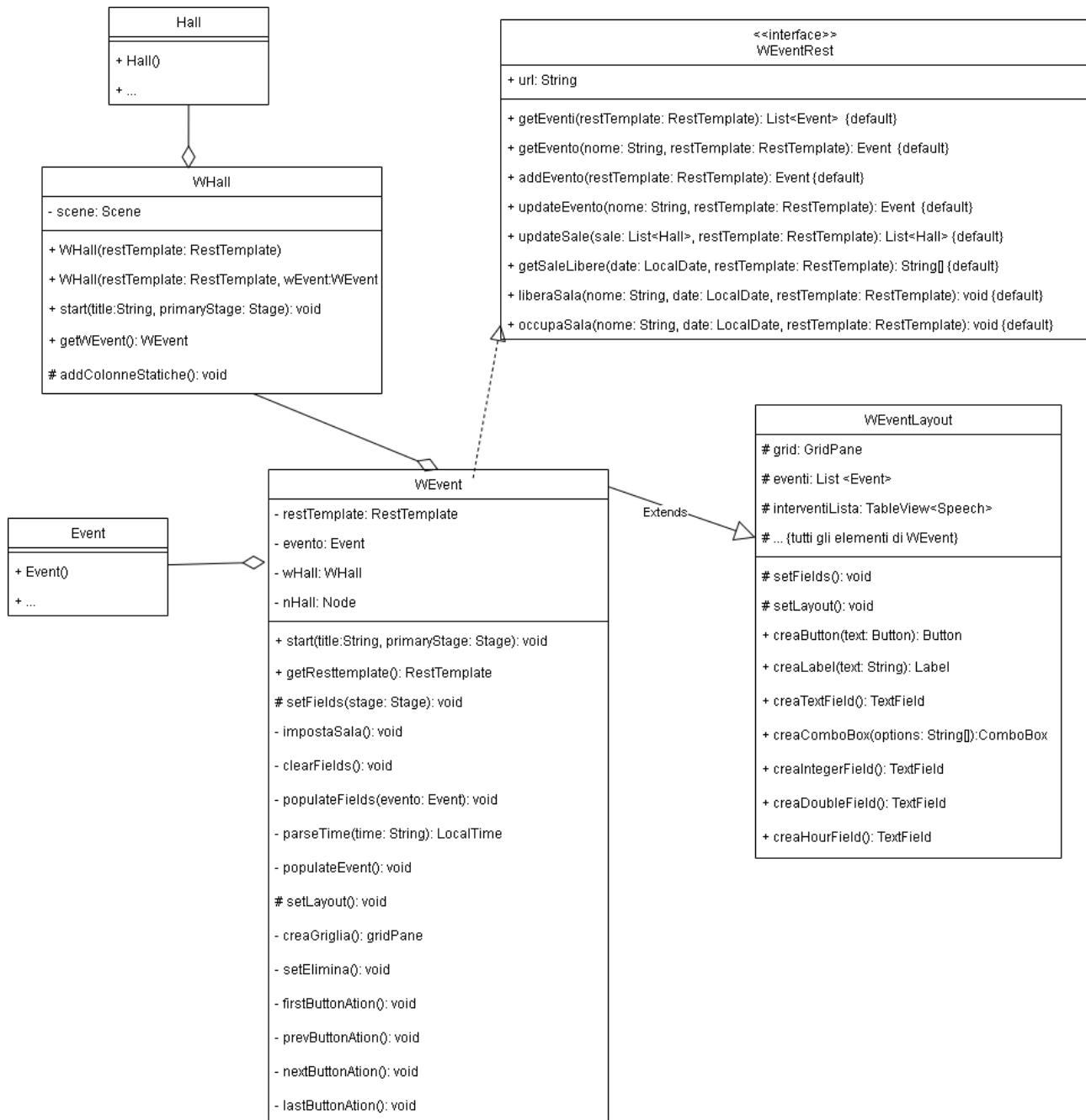
## 6.5.5 Class: Controller



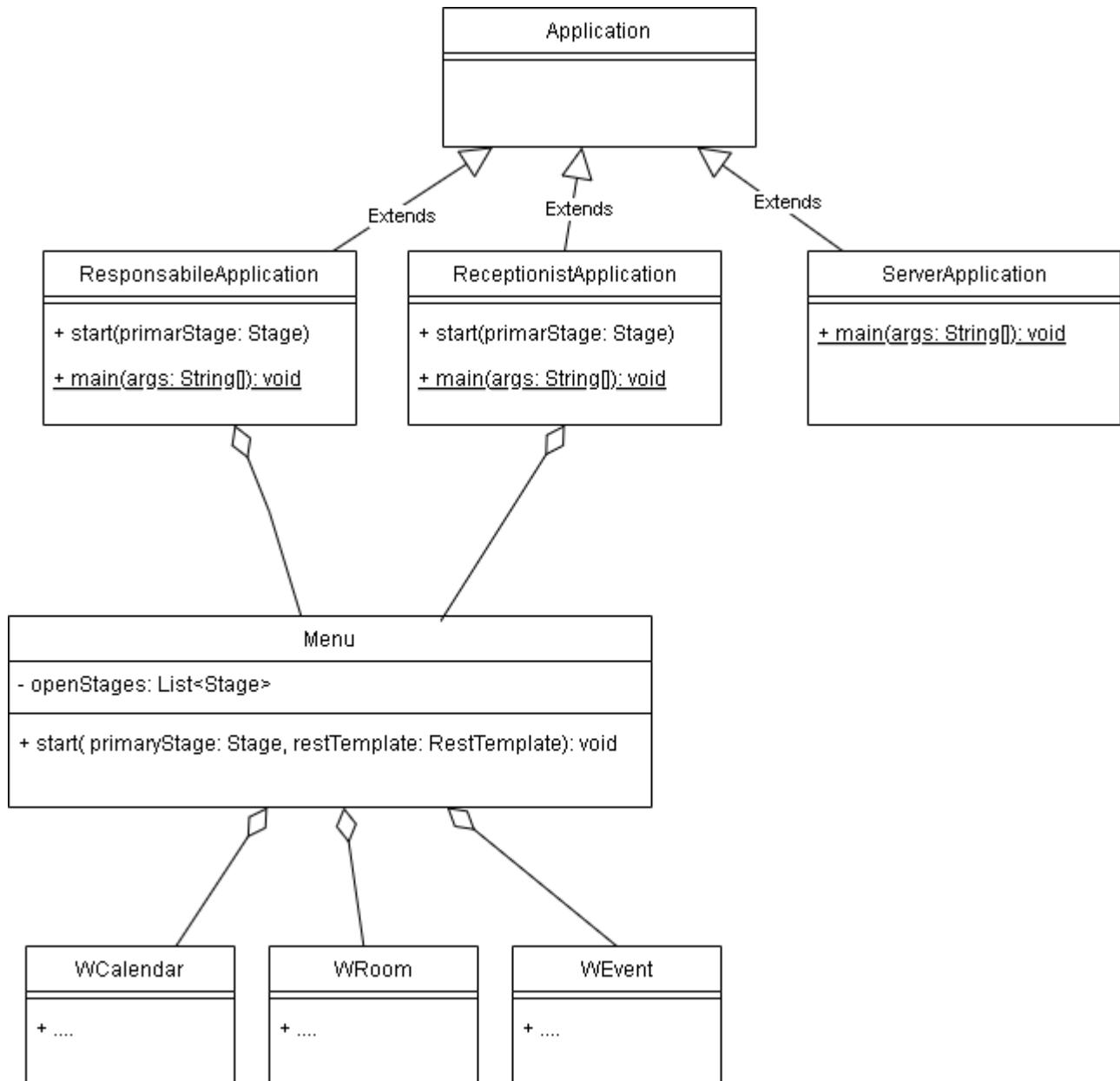
## 6.5.6 Class: WResource



## 6.5.7 Class: WEvent



## 6.5.8 Class: Application

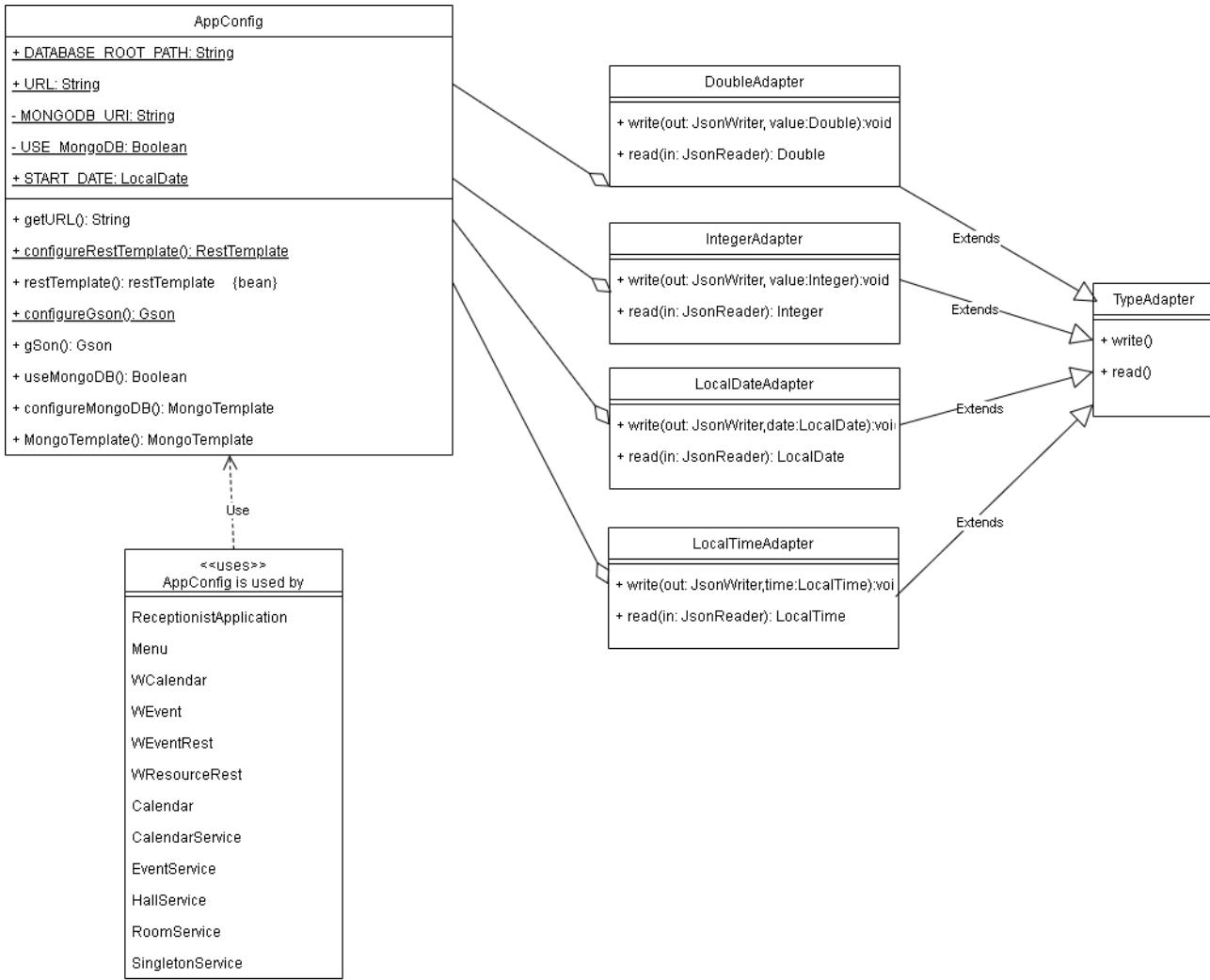


Lo schema mostra le due classi che fungono da main per i due Client (ResponsabileApplication e ReceptionistApplication) e per il Server (ServerApplication) e la classe menu da cui si accede alle singole finestre. Poiché JavaFX impedisce l'esecuzione di più applicazioni all'interno della stessa JVM, uno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

```

1 @echo off
2 echo Avvio dell'applicazione Responsabile...
3
4 rem Imposta il percorso del progetto
5 set PROJECT_DIR=%~dp0
6
7 rem Esegui l'applicazione ResponsabileApplication utilizzando Maven
8 mvn exec:java -Dexec.mainClass="client.ResponsabileApplication"
9
10 pause
11
  
```

## 6.5.9 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)

## 6.6 Interfacce grafiche

Create grazie alla API JavaFX, sono interfacce grafiche dinamiche.

### 6.6.1 Maschera Menu



### 6.6.2 Maschera WCalendar

A screenshot of a JavaFX application window titled "Visualizzazione Calendario". It shows a grid of dates from January 1 to January 11, 2024. The grid cells are colored green for availability and red for unavailability. A context menu is open over the cell for January 6, 2024, with two options: "CHIUSO" (red) and "DISPONIBILE" (green).

2024-01-01	2024-01-02	2024-01-03	2024-01-04	2024-01-05	2024-01-06	2024-01-07	2024-01-08	2024-01-09	2024-01-10	2024-01-11
DISPONIBILE	DISPONIBILE	DISPONIBILE	DISPONIBILE	DISPONIBILE	CHIUSO	CHIUSO	CHIUSO	DISPONIBILE	DISPONIBILE	DISPONIBILE

Cliccando sul singolo giorno è possibile cambiare lo stato

### 6.6.3 Maschera WRoom

A screenshot of a JavaFX application window titled "Visualizzazione Camere". It shows a grid of rooms and their statuses for each day of the week. The grid has columns for Nome, Costo, Tipo, N. Letti, and dates from 2024-01-01 to 2024-01-07. A context menu is open over the cell for Room 001 on Monday, showing options: "INUSO" (blue), "DISPONIBILE" (green), "PULIZIA" (gray), and "PRENOTATA" (yellow).

Nome	Costo	Tipo	N. Letti	2024-01-01	2024-01-02	2024-01-03	2024-01-04	2024-01-05	2024-01-06	2024-01-07	2024-01-08	2024-01-09	2024-01-10	2024-01-11
Room002	120.0	MINISUITE	2	PRENOTATA	DISPONIBILE	PRENOTATA	DISPONIBILE	DISPONIBILE	CHIUSO	CHIUSO	CHIUSO	DISPONIBILE	DISPONIBILE	DISPONIBILE
Room003	100.0	SUITE	2	DISPONIBILE	PRENOTATA	PULIZIA	PULIZIA	DISPONIBILE	CHIUSO	CHIUSO	CHIUSO	CHIUSO	DISPONIBILE	DISPONIBILE
Room004	120.0	ROYALE	3	INUSO	PRENOTATA	DISPONIBILE	PULIZIA	DISPONIBILE	CHIUSO	CHIUSO	CHIUSO	CHIUSO	DISPONIBILE	DISPONIBILE
Room005	120.0	ROYALE	3	DISPONIBILE	PULIZIA	INUSO	DISPONIBILE	DISPONIBILE	CHIUSO	CHIUSO	CHIUSO	CHIUSO	DISPONIBILE	DISPONIBILE
Room001	1000.0	SUITE	2	INUSO	DISPONIBILE	PULIZIA	INUSO	INUSO	DISPONIBILE	CHIUSO	CHIUSO	CHIUSO	CHIUSO	DISPONIBILE

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

#### **6.6.4 Maschera WEvent**

Gestione Eventi

ID da cercare:	<input type="text"/>
Messaggio:	<input type="text"/>
ID evento:	Event001
Nome Organizzatore:	Pippo
Costo Partecipazione:	100.0
Partecipanti Previsti:	100
Catering:	COFFEE_BREAK
Data:	01/01/2024
Sala:	Hall002
Ora Inizio:	09:00
Ora Fine:	19:00
Elenco Interventi:	<a href="#">Crea nuovo Intervento</a>

Nome	Costo	N. Posti	2024-01-01	2024-01-02	2024-01-03	2024-01-04	2024-01-05
Hall001	1200.0	30	PRENOTATA	DISPONIBILE	DISPONIBILE	DISPONIBILE	DISPONIBILE
Hall002	1500.0	50	PRENOTATA	DISPONIBILE	DISPONIBILE	DISPONIBILE	DISPONIBILE
Hall003	2200.0	150	DISPONIBILE	DISPONIBILE	DISPONIBILE	DISPONIBILE	DISPONIBILE

**i** La sala è prenotata per l'evento:

Evento {  
nome: Event002,  
data: 2024-01-01,  
ora inizio: 09:00,  
ora fine: 17:00,  
organizzato da: PAC,  
partecipanti previsti: 25,  
catering: COLAZIONE,  
sala: Hall001,  
costo partecipazione: 120.0,  
elenco interventi: [  
Intervento 1,  
Relatore 1,  
Descrizione dell'intervento 1111,  
Intervento 2,  
Relatore 2,  
Descrizione dell'intervento 2,  
qwqwqw,  
Relatoreeeee,  
bla bla bla,  
pippo,  
pippoooooo,  
oooooooooooooppppppppppppppppppppppp}

Titolo	Relatore	Descrizione	Azione
Pippo	Pippo	Pippo si racconta	<a href="#">Elimina</a>

[Primo Evento](#) [Evento Precedente](#) [Prossimo Evento](#) [Ultimo Evento](#) [Crea Nuovo Evento](#) [Salva / Modifica Eve...](#)

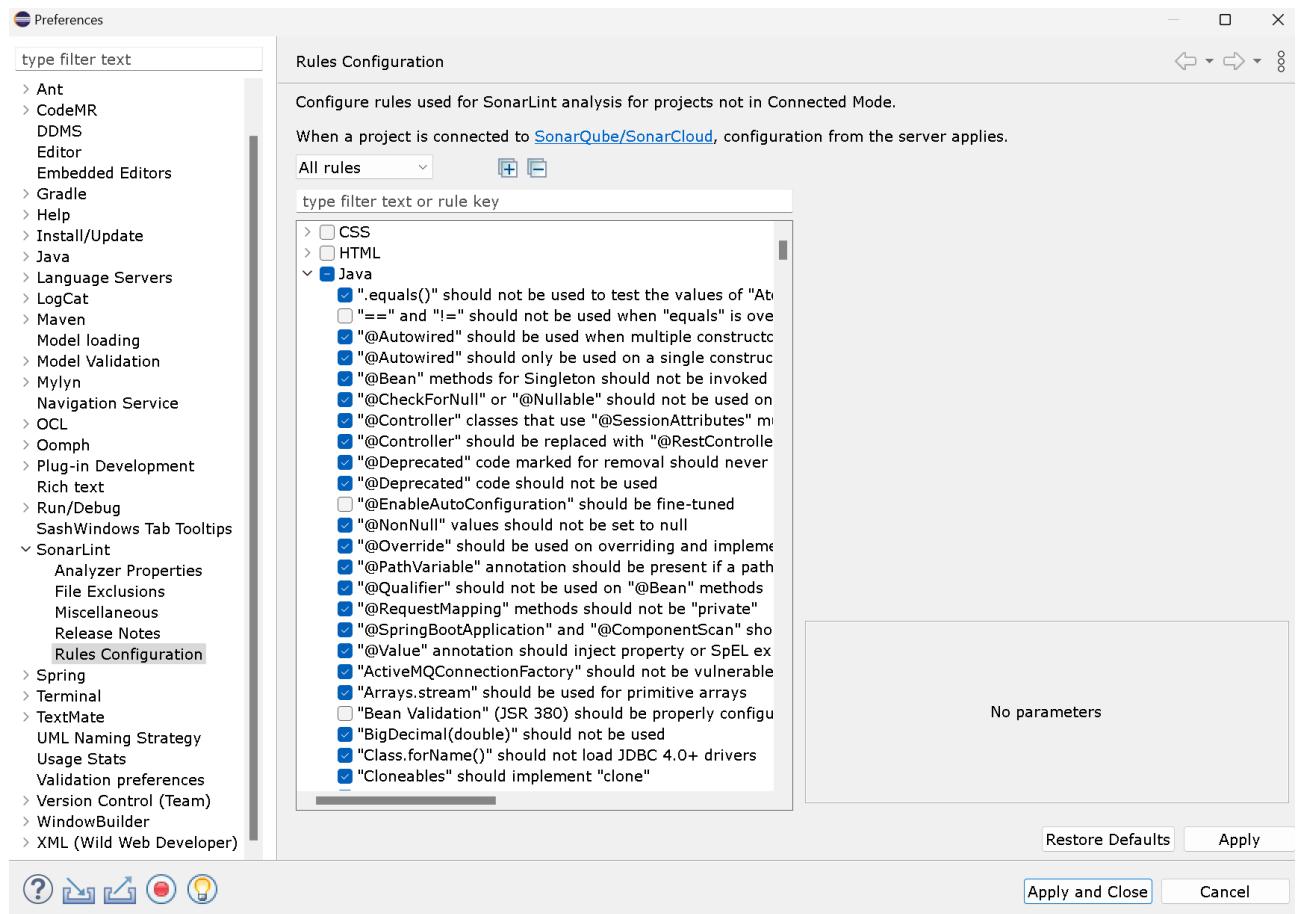
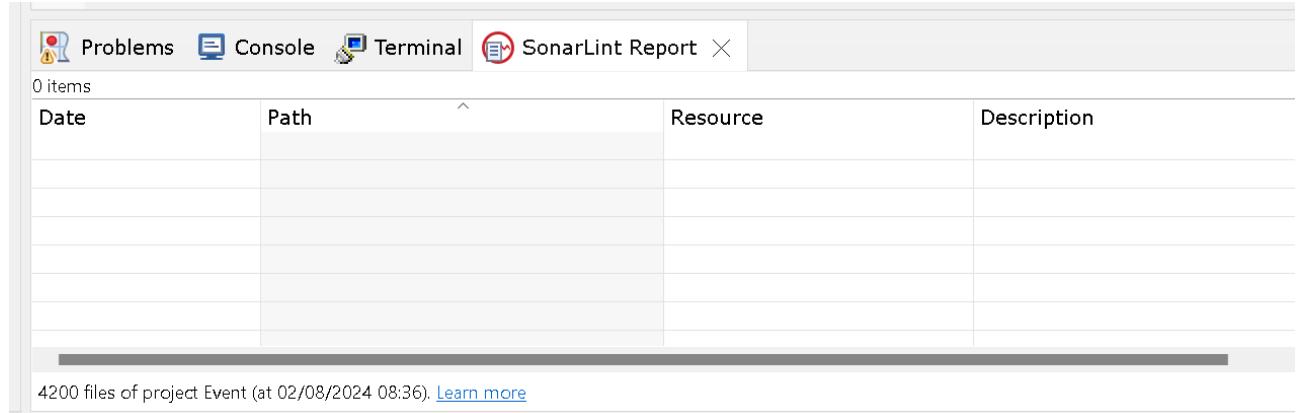
La maschera degli eventi ha due sotto-maschere:

- 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 buttoni 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.

## 6.7 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).



## 6.8 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)

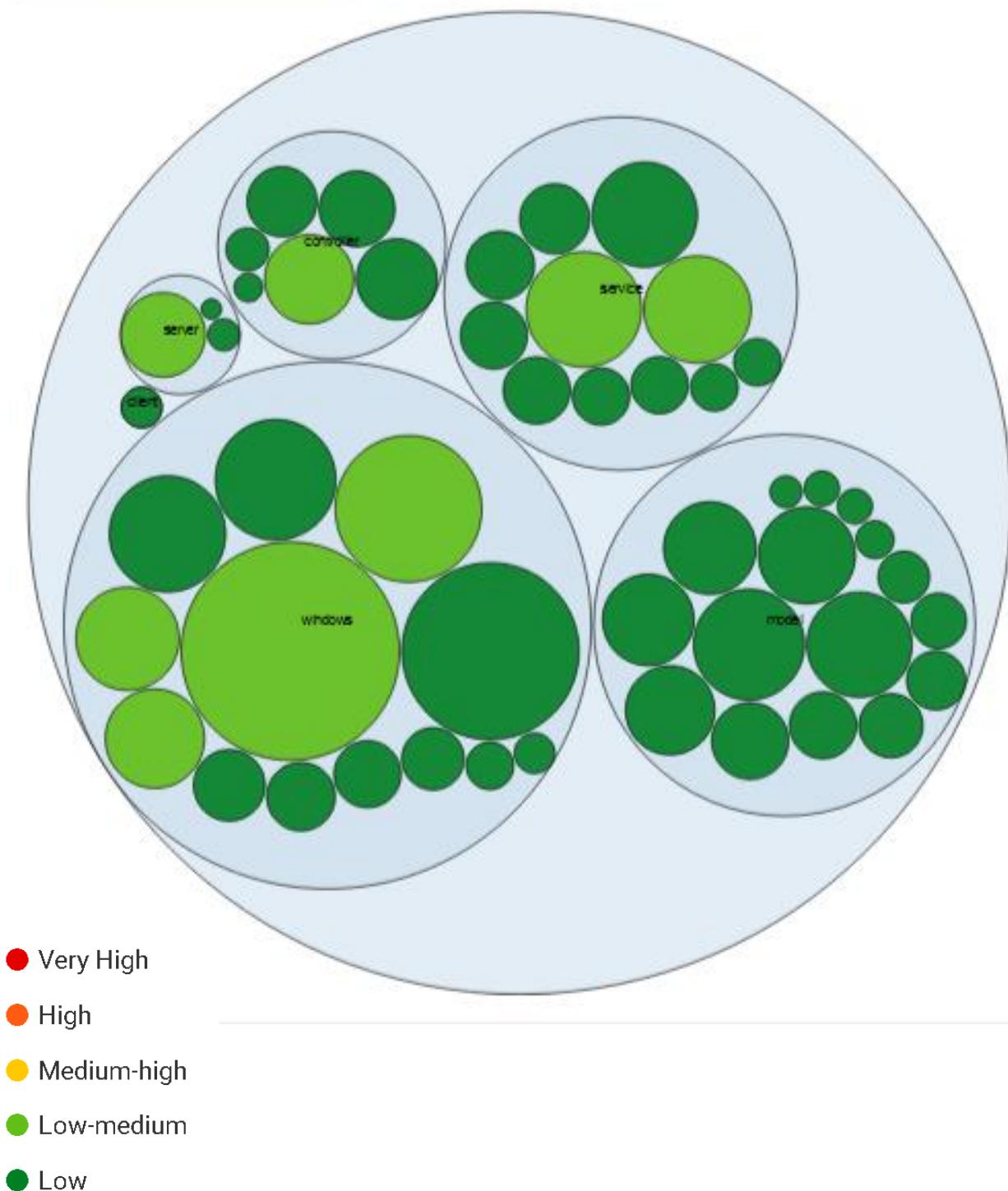
Metric Values by Packages

Select class metric to visualize

**Metric Selection**

Coupling

click to zoom, for detailed metrics, press ctrl or  $\text{ctrl} + \text{mouse}$  while hovering



## Metric Values by Packages

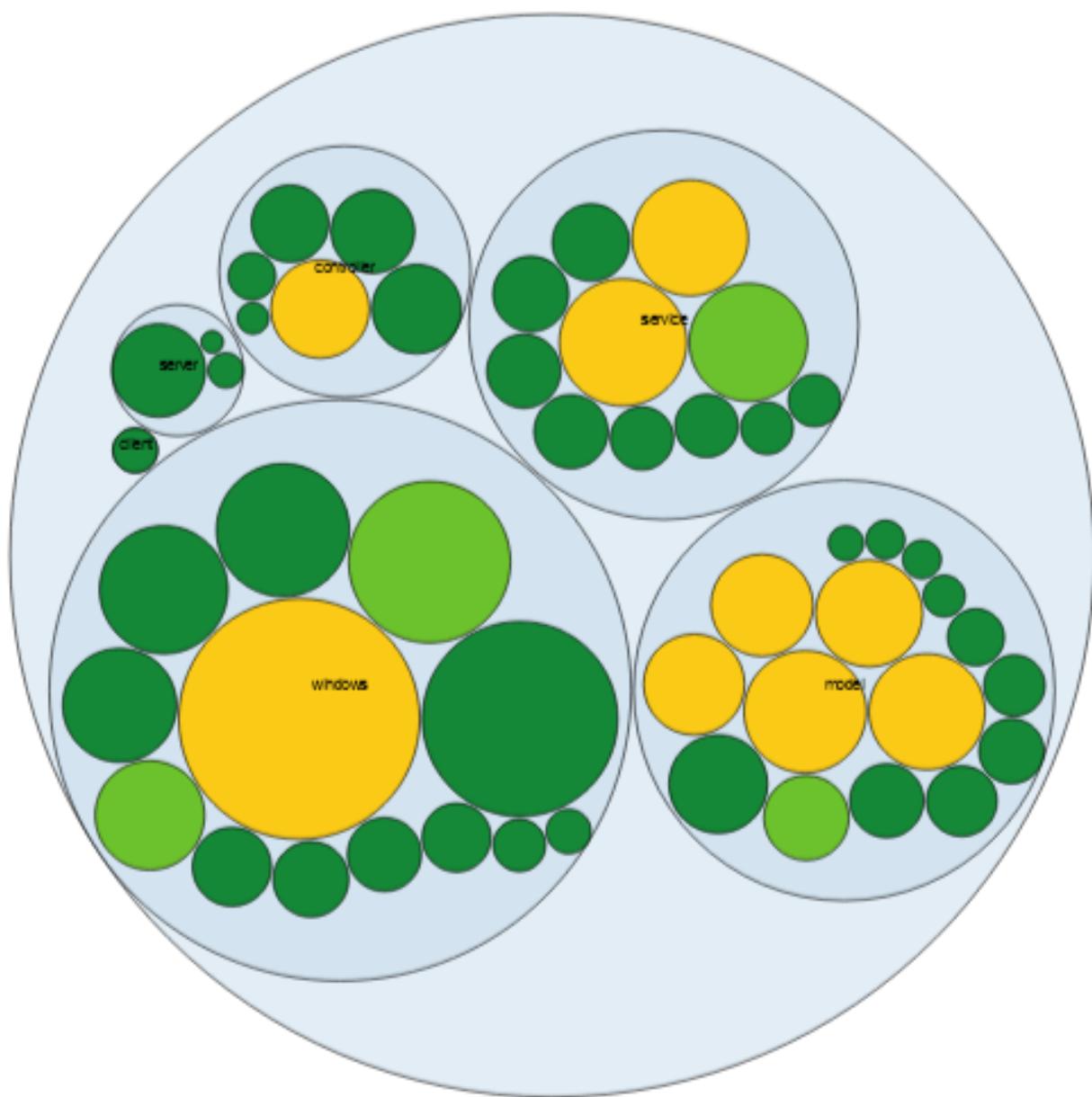
Select class metric to visualize

### Metric Selection

Lack of Cohesion



click to zoom, for detailed metrics, press ctrl or ⌘ while hovering



## Metric Values by Packages

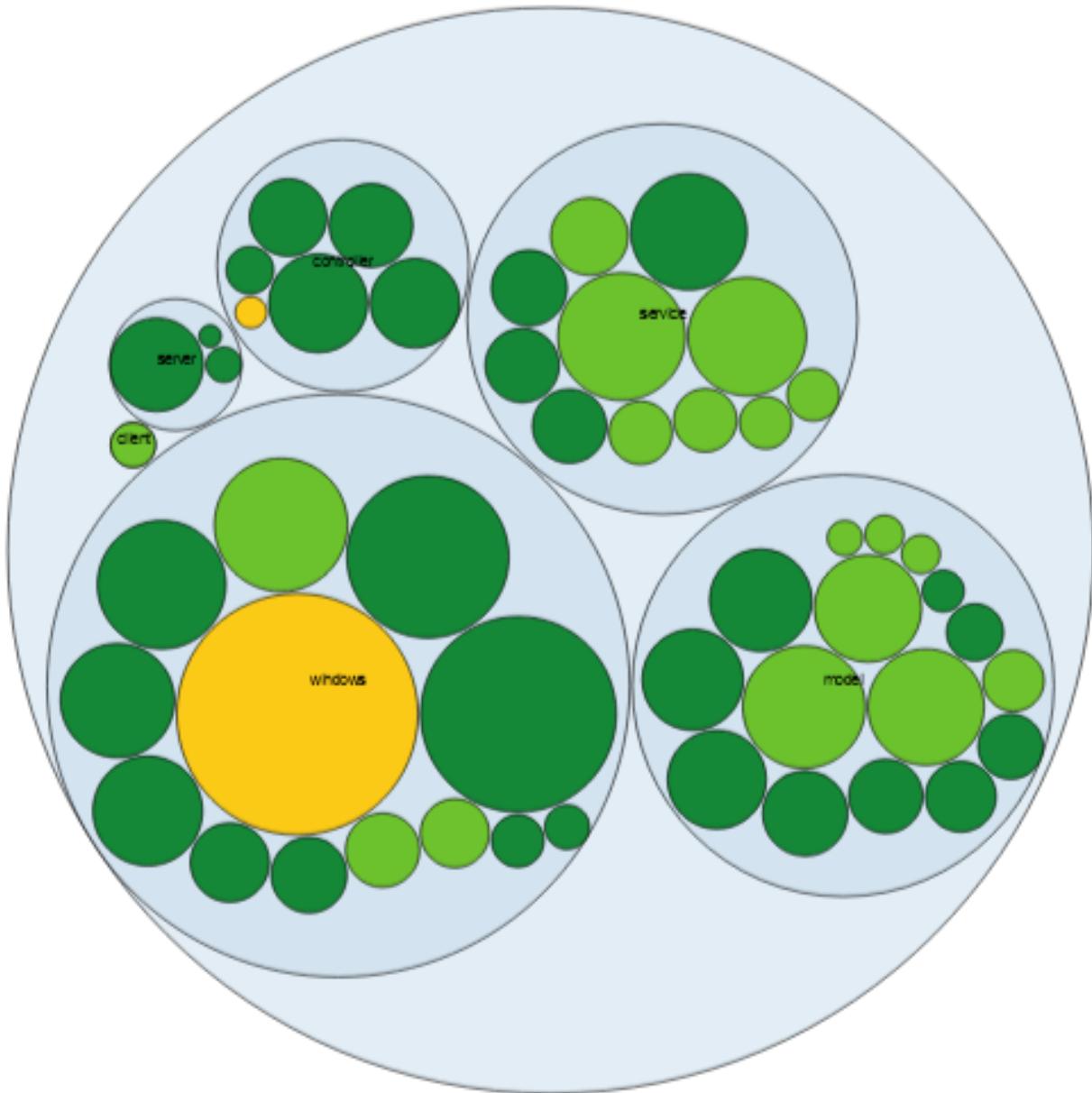
Select class metric to visualize

### Metric Selection

Complexity

?

click to zoom, for detailed metrics, press ctrl or  $\text{ctrl} + \text{mouse}$  while hovering



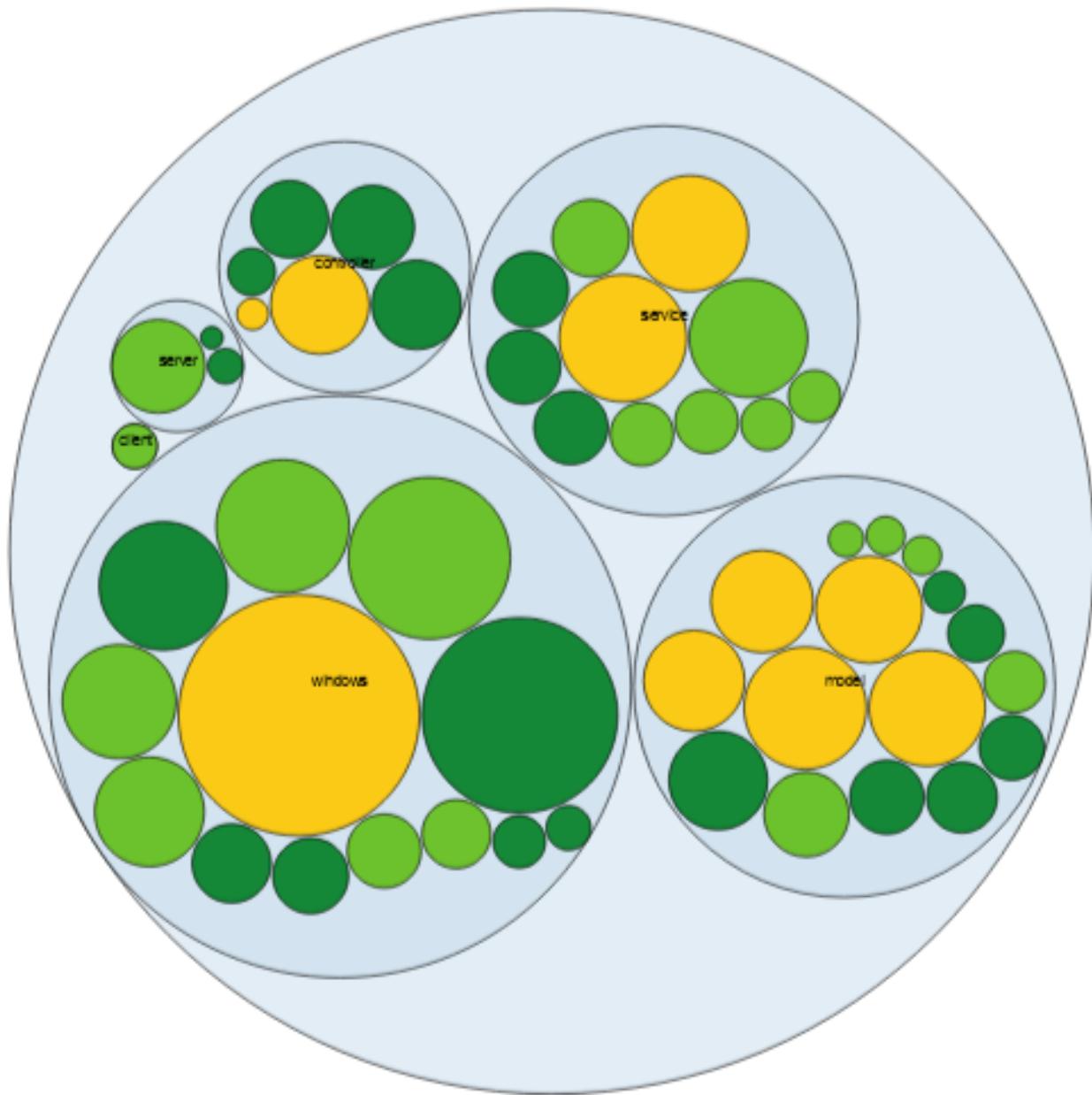
## Metric Values by Packages

Select class metric to visualize

### Metric Selection

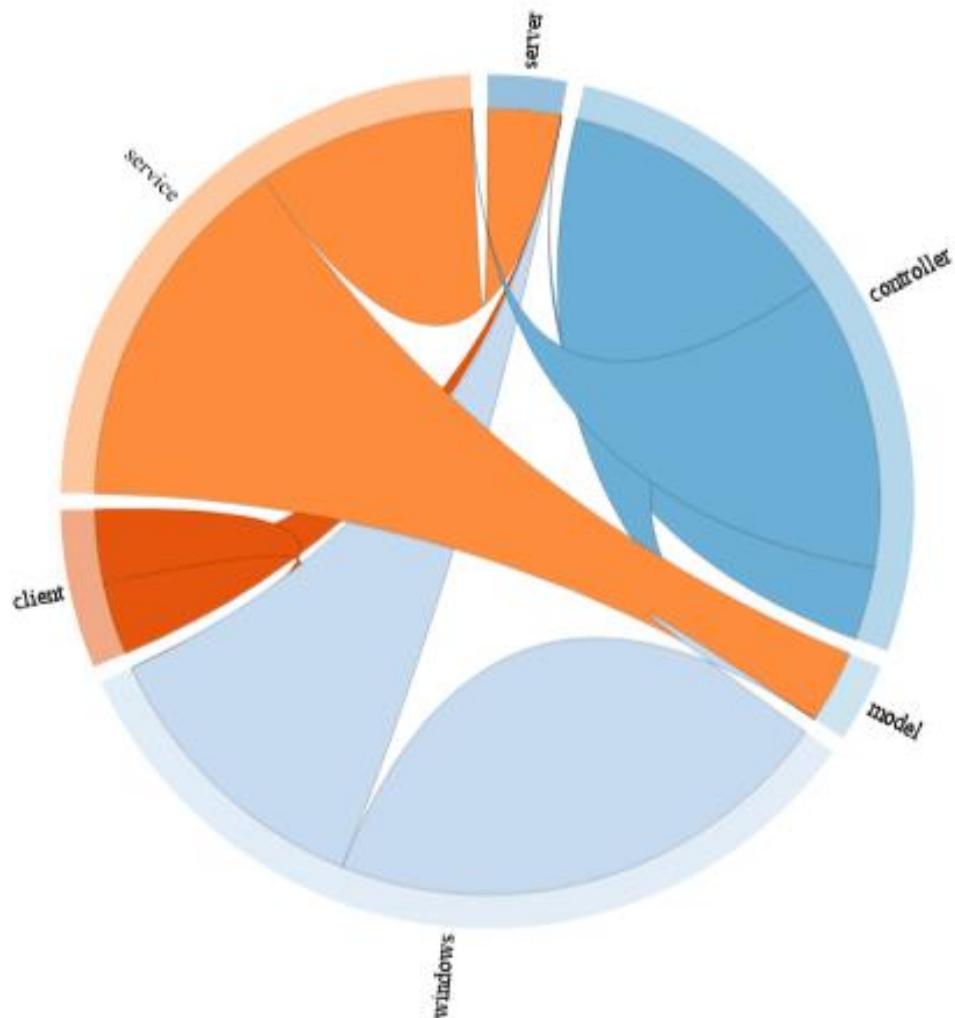


click to zoom, for detailed metrics, press ctrl or \* while hovering



## Package Dependencies

Hover on the wheel to see the details

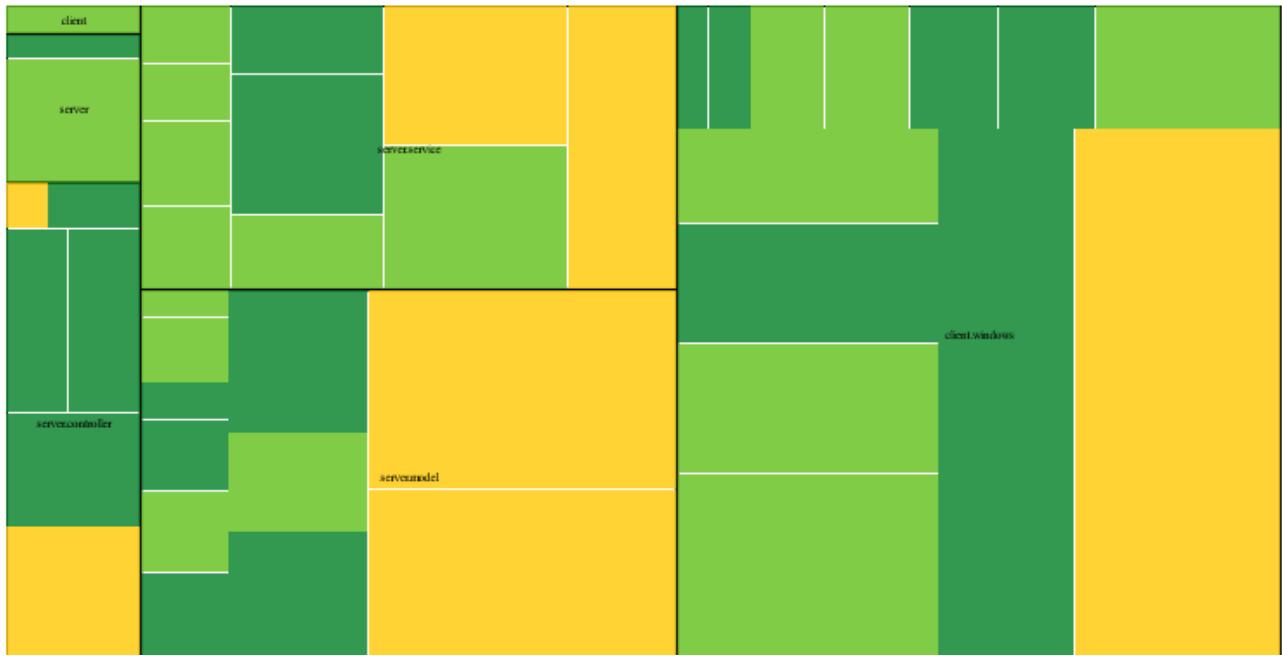


### Metric Values in Treemap Chart

Select class metric to visualize

#### Metric Selection

click to zoom, for detailed metrics, press ctrl or  $\times$  while hovering



## 6.9 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).

# 7 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.

The screenshot shows the MongoDB Compass application interface. At the top, there's a navigation bar with 'Connections', 'Edit', 'View', 'Collector', and 'Help'. Below it is a dark green header with the word 'Compass' and a gear icon. A button for 'New connection +' is visible. Underneath, a section titled 'Saved connections' lists two entries: 'Atlas DB' (last updated 30 ago 2024, 13:59) and 'Locale' (last updated 29 ago 2024, 09:15). The main area shows the 'Event' database and the 'Camere' collection. It has tabs for 'Documents' (5), 'Aggregations', 'Schema', 'Indexes' (1), and 'Validation'. A search bar and a toolbar with 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE' buttons are present. Below, three documents are listed with their JSON content:

```
_id: ObjectId('66d1f2769b191b18a469ba7b')
numeroletti: 2
tipo: "MINISUITE"
disponibilita: Array (244)
nome: "Room901"
costo: 120

_id: ObjectId('66d1f2769b191b18a469ba7c')
numeroletti: 2
tipo: "SUITE"
disponibilita: Array (244)
nome: "Room902"
costo: 180

_id: ObjectId('66d1f2769b191b18a469ba7d')
numeroletti: 3
tipo: "ROYALE"
disponibilita: Array (244)
nome: "Room903"
costo: 120
```

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).

The screenshot shows the MongoDB Atlas web interface. At the top, there's a browser-like header with back, forward, and search buttons, and a URL bar showing 'cloud.mongodb.com/v2/640382d6bf3f125abb1427c3#/metrics/replicaSet/66d0675d2897d163c67b44d3/explorer/Event/Camere/find'. Below it is a navigation bar with 'Atlas', 'UniBG', 'Access Manager', 'Billing', 'All Clusters', 'Get Help', and 'GAETANO'. The left sidebar has sections for 'Project 0', 'Data Services', 'App Services', 'Charts', 'Overview', 'DEPLOYMENT', 'Database', 'Data Lake', 'SERVICES', 'Device & Edge Sync', 'Triggers', 'Data API', 'Data Federation', 'Search', 'Vector Search', 'Stream Processing', 'Migration', and 'SECURITY'. The main area shows the 'Event' database and the 'Camere' collection. It has tabs for 'Overview', 'Real Time', 'Metrics', 'Collections' (selected), 'Atlas Search', 'Performance Advisor', 'Online Archive', and 'Cmd Line Tools'. A 'VISUALIZE YOUR DATA' button is present. The 'Collections' tab shows 'DATABASES: 1' and 'COLLECTIONS: 5'. It includes a 'Create Database' button, a 'Search Namespaces' input, and a 'Find' button. Below, a section for 'Event.Camere' provides storage details: 'STORAGE SIZE: 24KB', 'LOGICAL DATA SIZE: 64KB', 'TOTAL DOCUMENTS: 5', and 'INDEXES TOTAL SIZE: 20KB'. It also includes 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes' buttons. A 'Generate queries from natural language in Compass' link is available. A 'FILTER' button and a 'Type a query: { field: 'value' }' input field are shown. The 'QUERY RESULTS: 1-5 OF 5' section displays the same three documents as the Compass screenshot. A 'RESET', 'APPLY', and 'OPTIONS' button are at the bottom of the query input.

# 8 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.

The screenshot shows the JavaDoc interface with the 'CLASS' tab selected. A table lists various classes with their descriptions:

Class	Description
DinamicCol<T extends Resource>	Classe che implementa l'interfaccia Callback per la gestione di colonne dinamiche in una tabella.
Menu	La classe rappresenta la finestra di menu principale del client.
StaticCol	Il metodo statico per la creazione di colonne statiche in una tabella.
WAnagrafica	Bozza di finestra per eventuale futura implementazione di ricerca e/o gestione analoga di una camera (da generalizzare alla classe astratta WResource).
WCalendar	Classe per la gestione della finestra di visualizzazione del calendario.
WEvent	Classe per la gestione della finestra degli eventi
WEventLayout	Superclasse di WEvent, contiene i metodi per la definizione dei campi e del layout della finestra.
WEventRest	Interfaccia per la gestione delle richieste REST effettuate dalla finestra WEvent.
WHall	Classe per la gestione della finestra di visualizzazione delle sale.
WResource<T extends Resource>	Classe generica per la gestione della finestra con l'elenco delle risorse e la loro disponibilità.
WResourceClick<T extends Resource>	Classe per la gestione degli eventi di click sulle celle della tabella gestita da WResource.
WResourceRest<T extends Resource>	Classe per la gestione delle risorse tramite richieste REST ai vari controller.
WRoom	Classe per la gestione della finestra di visualizzazione delle stanze.

The screenshot shows the JavaDoc interface with the 'CLASS' tab selected. It displays the documentation for the `Visitor<T>` interface.

**Package** server.model  
**Interface Visitor<T>**

All Known Implementing Classes:  
`VisitorGetPrice`, `VisitorGetState`, `VisitorSetState`

---

**public interface Visitor<T>**

Interfaccia Visitor ha visit overloaded per consentire l'implementazione anche del visitor che hanno lo StatoData come parametro

**Method Summary**

All Methods   Instance Methods   Abstract Methods

Modifier and Type	Method	Description
T	<code>visit(Calendar o)</code>	
T	<code>visit(Calendar o, StateDate sd)</code>	
T	<code>visit(Hall o)</code>	
T	<code>visit(Hall o, StateDate sd)</code>	
T	<code>visit(Room o)</code>	
T	<code>visit(Room o, StateDate sd)</code>	