

Relazione finale del progetto "GetOut!"

Enrico Corradini, Edoardo Balducci, Alessandro Mentuccia, Andrea Chiorrini,
Jacopo Carloni

14/06/2018

Indice

1	Introduzione al progetto	3
2	Elicitazione dei requisiti	3
2.1	Definizione degli Stakeholders	3
3	Analisi dei requisiti	3
3.1	Server	3
3.2	Client	4
3.3	Requisiti Utente	4
3.3.1	Utenti	4
3.3.2	Ambiente	4
3.3.3	Obiettivi	5
3.4	Requisiti di Sistema	5
3.4.1	Requisiti funzionali	5
3.4.2	Requisiti non funzionali	5
3.4.3	Requisiti di Dominio	5
3.5	Glossario dei termini	6
4	Metodi adottati	8
4.1	eXtreme Programming	8
4.2	Pianificazione del progetto	8
4.2.1	Release Planning Game	8
4.2.2	Iteration Planning Game	9
4.2.3	Iteration to Release	10
4.2.4	Interfacce Utente	10
5	Tecnologie scelte	11
5.1	Sistema Android per l'applicativo e Android Studio	11
5.2	Sistema Linux per il server	11
5.3	Intellj e Datagrip	12
5.4	Java e JDK 10	12
5.5	Database SQLite e MySQL	12
5.6	SensorTag T.I. con B.L.E.	12
5.7	Bash	12
5.8	LaTeX	12
6	User Story 2: Gestione Mappa (Lato Client)	13
6.1	Storia utente	13
6.2	Divisione in task	13
6.3	CRC	14
6.4	Mockup	14
6.5	Diagrammi di dialogo e navigazione	15
6.6	Diagramma delle classi di analisi	15
6.7	Diagramma dei package	16
6.8	Diagramma delle classi di progettazione	17
6.9	Diagramma delle architetture	18
6.10	Diagramma delle sequenze	18
6.11	Casi di Test	19
7	User Story 3: Gestione Mappa (Lato Server)	21
7.1	Storia utente	21
7.2	Divisione in task	21
7.3	CRC	22
7.4	Diagramma delle classi di analisi	22
7.5	Diagramma dei package	22
7.6	Diagramma delle classi di progettazione	22
7.7	Diagramma delle architetture	24

7.8	Diagramma delle sequenze	24
7.9	Casi di Test	24
8	User Story 4: Modalità di emergenza	26
8.1	Storia utente	26
8.2	Divisione in task	27
8.3	CRC	27
8.4	Diagramma delle classi di analisi	28
8.5	Diagramma dei package	29
8.6	Diagramma delle classi di progettazione	30
8.6.1	Progettazione lato Client	30
8.6.2	Progettazione lato Server	31
8.7	Diagramma delle architetture	32
8.8	Diagramma delle sequenze	33
8.9	Casi di Test	34
9	User Story 1: Modalita Ordinaria	35
9.1	Storia utente	35
9.2	CRC	36
9.3	Diagramma delle classi di analisi	36
9.4	Diagramma dei package	37
9.5	Diagramma delle classi di progettazione	37
9.6	Diagramma delle architetture	38
9.7	Diagramma delle Sequenze	39
9.8	Casi di Test	39
10	User Story 5: Interfaccia amministrazione Server	40
10.1	Storia utente	40
10.2	Divisione in task	40
10.3	CRC	41
10.4	Diagramma delle classi di analisi	42
10.5	Diagramma dei package	42
10.6	Diagramma delle classi di progettazione	42
10.7	Diagramma delle architetture	43
10.8	Diagramma delle sequenze	43
10.9	Casi di Test	43
11	User Story 6: Sicurezza Server	46
11.1	Storia utente	46
11.2	Divisione in task	46
11.3	IPTABLES	47
11.4	Ulteriori cause di DoS	47
11.5	Casi di Test	47

1 Introduzione al progetto

Il progetto consiste nello sviluppo di un sistema automatico di indirizzamento delle persone verso la più vicina via di fuga durante un'emergenza. L'ecosistema prevede un lato server e un lato client. Il primo si occupa della gestione dei dati, ovvero inserimento, lettura e storage, e del rilevamento di emergenze. Lato client, invece, ogni utente è dotato di un'applicazione mobile che prevede due modalità di funzionamento:

- utilizzo in condizioni ordinarie, dove l'utente può richiedere il miglior percorso per raggiungere un'aula dell'edificio;
- utilizzo in condizioni di emergenza, dove l'utente è avvertito in caso si verifichi un incendio o una qualsiasi altra emergenza nell'edificio e viene guidato nel percorso più veloce per l'evacuazione.

Il sistema prevede una distribuzione di beacon bluetooth all'interno dell'edificio: tramite questi il client sarà in grado di conoscere la propria posizione e il percorso da seguire tramite passaggio da un beacon al successivo.

2 Elicitazione dei requisiti

Una volta individuati gli stakeholders, la raccolta dei requisiti è avvenuta tramite una presentazione iniziale del progetto, seguita da un'intervista. Inoltre, durante le fasi di analisi e progettazione degli stessi, i requisiti precedentemente raccolti sono stati raffinati grazie a successive interviste e incontri.

2.1 Definizione degli Stakeholders

- Luca Spalazzi
- Gabriele Bernardini
- Lucio Ciabattoni
- Lucia Pepa
- Silvia Santarelli

3 Analisi dei requisiti

In questa sezione verranno spiegati i requisiti del progetto, per il server e per il client.

3.1 Server

Il server deve essere di tipo REST, ovvero deve garantire la comunicazione distribuita, a più client, tramite il solo protocollo HTTP, evitando quindi eventuali altri livelli di comunicazione, come ad esempio il SOAP. Partendo da questa caratteristica il server deve poter ricevere richieste dai vari client, poter leggere all'interno del database dei dati e restituire le informazioni cercate.

Il database con cui comunica il server contiene tutti i dati relativi agli edifici di cui il sistema si occupa. Il database dovrà quindi modellare la composizione di un edificio nel seguente modo:

1. lista di tutti gli EDIFICI del complesso;
2. lista dei PIANI di ogni edificio;
3. per ogni piano identificare i TRONCHI che corrispondono ai corridoi o comunque alle zone attraversabili;
4. per ogni piano identificare le AULE
5. lista dei BEACON distribuiti nell'edificio

6. lista dei CLIENT collegati

Tali dati saranno inseriti e gestiti da un amministratore di sistema tramite un'interfaccia grafica prevista per tale scopo.

Controllando periodicamente i dati, il server dovrà essere in grado di riconoscere un'eventuale emergenza e di conseguenza inviare una notifica a tutti gli utenti presenti nell'edificio. Non appena l'emergenza sarà rientrata invierà una nuova notifica.

Nel caso ci fosse un'emergenza, il server deve essere in grado di calcolare il percorso più breve verso l'uscita per ogni client collegato. Per fare ciò sono definiti i parametri che definiscono la scelta del percorso, ogni parametro è relativo ad un tronco generico i:

- Vulnerabilità, probabilità che si possa verificare un'emergenza nel tronco i, $V(i) \in [0,1]$;
- Rischio vita, possibilità di rischio per la vita nel tronco i, $R(i) \in \{0,1\}$;
- Presenza fumo, densità di fumo per il tronco i in percentuale, $K(i) \in [0,1]$;
- Affollamento di un tronco, densità di persone rispetto all'area del tronco, $LOS(i) \in \mathbb{R}$;
- Lunghezza del tronco i (metri), $L(i) \in \mathbb{R}$.

Ad ognuno di questi parametri è associato un peso da considerare durante il calcolo del costo di un tronco. Tali pesi sono definiti dall'amministratore di sistema durante la configurazione del sistema. In caso di funzionamento in modalità ordinaria tutti i pesi avranno valore 0 tranne quello della lunghezza.

3.2 Client

Il client corrisponde ad una singola applicazione. Deve essere in grado di comunicare con il server in modo da richiedere tutti i dati necessari per il funzionamento, ovvero: la mappa del piano in cui si trova l'utente e la sua posizione sulla mappa. Nel caso in cui il server non sia raggiungibile l'applicazione deve essere in grado di funzionare offline e garantire le funzionalità previste.

Il client ha due modalità di funzionamento.

1. Modalità ordinaria: in condizioni normali il sistema presenterà una pianta della struttura e la posizione attuale. Sarà possibile scegliere un'aula verso cui si vuole conoscere il percorso più breve, il quale sarà successivamente visualizzato sulla mappa.
2. Modalità di emergenza: nel caso in cui si ricevesse una notifica di avvenuta emergenza, il client dovrà richiedere il percorso più breve verso l'uscita al server sulla base della posizione attuale. Tale percorso dovrà essere visualizzato sulla mappa.

L'applicazione deve anche prevedere il passaggio manuale tra le due modalità.

Durante il movimento dell'utente il percorso deve essere ricalcolato nel caso di variazioni delle condizioni dell'edificio. Se l'utente è uscito dall'edificio, allora l'applicazione ha raggiunto il suo scopo.

Nel caso in cui l'applicazione fosse offline, è necessario prevedere una copia locale dei dati necessari per il funzionamento.

3.3 Requisiti Utente

3.3.1 Utenti

Gli utenti che utilizzeranno l'applicazione saranno coloro che frequentano le strutture gestite dal sistema. In caso di emergenza è importante che l'utente non debba pensare a dover interagire con l'applicazione, visto che si tratta di una situazione di grande stress. Per questo è necessario che la gestione dell'emergenza sia automatizzata e l'utente si preoccupi solo di seguire le indicazioni ed evadere l'edificio. In questo modo si evitano anche problemi di utilizzo con utenti con poca conoscenza tecnologica o dell'ambiente circostante.

3.3.2 Ambiente

L'utente deve essere in grado di utilizzare l'applicazione durante il movimento sia in condizioni ordinarie che di emergenza. L'applicazione deve poter essere utilizzata all'interno delle strutture gestite dal sistema, al di fuori non ha senso utilizzarla.

3.3.3 Obiettivi

1. **Visualizzazione posizione:** all'apertura dell'app l'utente deve essere in grado di visualizzare la pianta del piano dove si trova e la propria posizione rispetto ad essa.
Ambiente: Piano dell'edificio.
2. **Modalità ordinaria:** in modalità ordinaria l'utente deve essere in grado di selezionare un'aula di destinazione e poter visualizzare il percorso più veloce per raggiungerla.
Ambiente: Edificio.
3. **Modalità emergenza:** in modalità emergenza l'utente deve subito aver ben chiaro il tragitto da percorrere verso l'uscita, senza ulteriori interazioni.
Ambiente: Edificio.
4. **Notifica:** in caso di emergenza l'utente deve ricevere una notifica. Quando clicca su di essa l'applicazione deve essere aperta in modalità emergenza.
Ambiente: Edificio.
5. **Cambio manuale di modalità:** l'utente deve essere in grado di poter cambiare manualmente la modalità di funzionamento dell'applicazione.
Ambiente: Edificio.

3.4 Requisiti di Sistema

3.4.1 Requisiti funzionali

Il sistema deve prevedere la comunicazione HTTP tra ciascun client e il server. Il server deve poter leggere tutte le informazioni relative agli edifici del complesso gestito e deve poter comunicare tali informazioni al client. Il server deve rilevare eventuali emergenze e inviare una notifica ai vari client connessi. Quando richiesto il server deve calcolare un percorso verso l'uscita o verso una destinazione scelta, tenendo conto dei parametri attuali. Il client deve essere in grado di funzionare offline nel caso in cui non sia possibile la comunicazione con il server. In questo caso l'utente non deve accorgersi della differenza e le funzionalità devono essere garantite. Il client deve mantenere una copia locale dei dati. Il ricalcolo del percorso durante il movimento deve essere automatico, le interazioni dell'utente durante la fuga devono essere nulle. Se l'utente è uscito dall'edificio, l'applicazione deve terminare.

3.4.2 Requisiti non funzionali

1. La comunicazione tra client e server si deve basare solamente sul protocollo HTTP (Server REST).
2. Il sistema deve poter funzionare con Beacon "Texas Instruments SimpleLink Multi-Standard SensorTag" (modello CC2650STK).
3. L'app mobile deve essere sviluppata per Android.
4. L'app deve funzionare anche nel caso in cui non riesca a connettersi al server.
5. I dati inviati e ricevuti devono essere in formato JSON.
6. Il server usa un Database relazionale per la memorizzazione dei dati.

3.4.3 Requisiti di Dominio

La connessione tra client e server deve avvenire nella rete LAN del complesso gestito dal sistema.

3.5 Glossario dei termini

Termino	Descrizione	Sinonimi	Collegamenti
Server	Parte dell'ecosistema progettato che si occupa della lettura e salvataggio dei dati e della comunicazione di questi ultimi ai vari client.		Client, Database, Amministratore
Client	Parte dell'ecosistema che consiste in un'applicazione Android utilizzata dall'utente.	Applicazione (o App), Server Utente	
Amministratore	Responsabile dell'installazione, configurazione e manutenzione del server.		Server, Database
Richiesta	Comunicazione tra client e server tramite protocollo HTTP.		Client, Server
Database	Contenitore di tutti i dati necessari al sistema.	Storage	Server
Edificio	Struttura in cui si trova l'utente, parte del complesso gestito dal sistema.	Struttura	Client, Piano, Tronco, Aula, Beacon
Piano	Sezione di un edificio in cui può trovarsi un utente.		Client, Edificio, Tronco, Aula, Beacon
Tronco	Zona attraversabile del piano, solitamente un corridoio. Il tronco è dotato di larghezza e lunghezza.	Corridoio	Edificio, Piano, Tronco, Aula, Beacon
Aula	Stanza appartenente ad un piano di un edificio alla quale è associato un beacon.		Client, Edificio, Piano, Tronco, Beacon
Beacon	Dispositivi bluetooth low energy presenti all'interno dei vari tronchi del piano. Servono per l'identificazione della posizione dell'utente e del suo spostamento all'interno dell'edificio.	SensorTag	Edificio, Piano, Tronco, Aula, Client, Posizione
Posizione	Coordinate dell'utente sulla mappa.		Mappa, Piano, Client
Mappa	Immagine che definisce la pianta di un piano.	Pianta	Piano
Emergenza	Qualsiasi condizione che necessita di un'evacuazione dell'edificio.		Edificio, Notifica

Notifica	Segnalazione tramite messaggio sonoro sul client dell'occorrenza o della fine di un'emergenza.	Client, Emergenza
Percorso	Sequenza di tronchi da attraversare per raggiungere la destinazione scelta o l'uscita.	Tronco, Piano

4 Metodi adottati

La metodologia di lavoro adottata è quella dell'eXtreme Programming (XP). Questa rientra tra le metodologie di progettazione agile.

4.1 eXtreme Programming

La metodologia agile è un approccio all'ingegneria del software che pretende di garantire la massima soddisfazione del committente, producendo sin da subito e in maniera continuativa software funzionante. Tale metodologia risulta essere per sua natura particolarmente flessibile alle variazioni dei requisiti provenienti dagli stakeholders.

XP può essere classificato come incrementale: l'idea di fondo è quella di suddividere il progetto in diverse release ognuna delle quali implementa una o più funzionalità. Gli stakeholder descrivono in linguaggio naturale strutturato tali funzionalità su apposite schede, chiamate "user stories". Una release, dunque, può comprendere una o più storie utente. A ciascuna storia viene associata una priorità dagli stakeholder.

L'aver adottato questa metodologia di progettazione è risultata vincente grazie alla sua flessibilità e alla sua leggerezza in relazione alla documentazione da produrre (si noti, per esempio, che non risulta necessario stilare il documento iniziale di specifica dei requisiti) permettendoci di rispondere in maniera tempestiva alle richieste avanzate dagli stakeholder.

4.2 Pianificazione del progetto

Il progetto consta di un'unica release, la quale comprende un certo numero di funzionalità, e quindi storie utente. Si sono affrontate le varie fasi di analisi, progettazione e sviluppo come da metodologia, e si riportano di seguito l'esito di ciascuna di esse.

Nella fase di pianificazione abbiamo raggiunto i seguenti risultati, spiegati nel seguente diagramma di Gantt, figura 1.

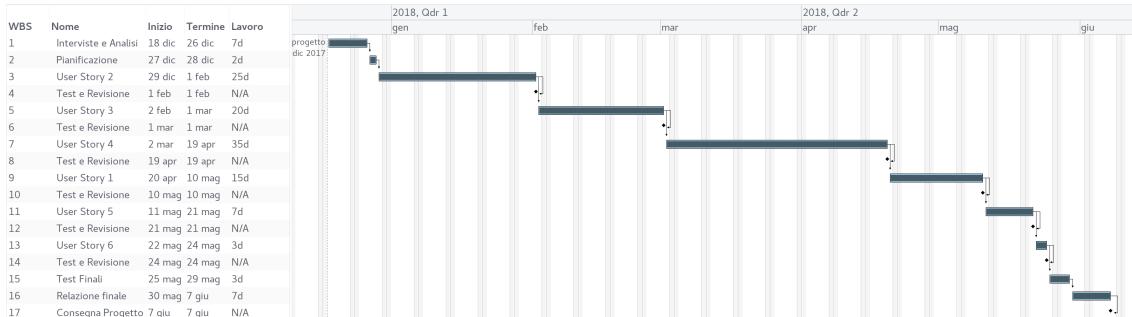


Figura 1: Diagramma di Gantt

Per quanto riguarda l'analisi delle responsabilità, queste verranno specificate nei task per ogni iteration planning game. Abbiamo inoltre scelto di riportare le storie utente nell'ordine di priorità assegnato in cui le abbiamo svolte, come riportato nel gantt.

4.2.1 Release Planning Game

Pur essendo notoriamente costituito da più iterazioni che si susseguono ciclicamente, il release planning game è stato caratterizzato da una sola iterazione, poiché si era concordato con gli stakeholder il rilascio di un'unica release del software, come specificato in precedenza.

L'esito del planning game è l'insieme delle seguenti user stories:

1. Modalità Ordinaria
2. Gestione Mappa (Lato Client)
3. Gestione Mappa (Lato Server)
4. Modalità Emergenza

5. Interfaccia Amministrazione Server

6. Sicurezza Server

Le due fasi di *Exploration* e *Commitment* hanno prodotto le storie relative alla modalità ordinaria, quella di emergenza e una storia sulla gestione delle mappe. In fase di *Steering* si è provveduto a scorrere la terza di queste, perché troppo lunga, nelle storie numero due e tre del precedente elenco; si è inoltre aggiunta la sesta storia relativa alla sicurezza del server.

1. Modalità Ordinaria

Priorità: 4

Implementazione delle funzionalità client che riguardano la gestione dell'applicazione in modalità ordinaria:

1. Scaricare dati dal server per il database offline.
2. Interfaccia per modalità ordinaria.
3. Bottone per cambiare modalità manualmente.
4. Calcolare percorso per una destinazione selezionata online.
5. Calcolare percorso per una destinazione selezionata offline.

Implementazione delle funzionalità server che riguardano la gestione dell'applicazione in modalità ordinaria:

1. Restituire dati al Client.
2. Restituire percorso per una destinazione selezionata.

2. Gestione Mappa (Lato Client)

Priorità: 1

Implementare tutte le funzionalità che riguardano la gestione della mappa sul client:

1. Avviare la scansione tramite bluetooth, il quale si conterà al beacon più vicino.
2. Se il client è connesso ad internet scaricare dai server i dati dell'edificio in cui si trova l'utente, i piani, i tronchi, le aule e i beacon, altrimenti leggere dal database di backup.
3. Visualizzare la mappa relativa al piano, in cui si trova l'utente.
4. Visualizzare sulla mappa la posizione dell'utente tramite simbolo grafico.

3. Gestione Mappa (Lato Server)

Priorità: 1

Implementare tutte le funzionalità del server relative alla gestione delle mappe:

1. Restituire tutti i dati memorizzati nel database, a cui è connesso, relativi all'edificio, le mappe, i tronchi e le aule di ogni piano e i beacon.
2. Aggiornare numero di utenti collegati a beacon.
3. Restituire la posizione del beacon a cui l'utente è collegato.

4. Modalità d'emergenza

Priorità: 3

Implementare tutte le funzionalità client che riguardano la gestione dell'emergenza:

1. Ricezione della notifica di emergenza.
2. Richiesta percorso verso l'uscita.
4. Richiesta percorso in caso di variazioni delle condizioni dell'edificio.
5. Controllo uscita edificio.
6. Aggiornamento della posizione dell'utente e del percorso durante il movimento.
7. Visualizzare sulla mappa il percorso verso l'uscita.
8. Ricezione rientro emergenza.

Implementare tutte le funzionalità server che riguardano la gestione dell'emergenza:

1. Notificare emergenza.
2. Calcolo percorso verso l'uscita più vicina.
3. Restituire condizioni edificio.
4. Rilevazione rientro emergenza nel database e invio notifica.

5. Interfaccia amministrazione Server

Priorità: 3

Implementare un'interfaccia grafica per l'inserimento dei dati nel database.

Visualizzazione di un form in cui selezionare le cose da aggiungere:

1. possibilità di aggiungere edifici, piani, tronchi e beacon al database con i relativi dati
2. possibilità di caricare l'immagine della mappa se non presente

Visualizzazione della tabella dei parametri dei tronchi e possibilità di modificarli.

Possibilità di importazione di dati tramite file csv.

Possibilità di modificare i pesi dei parametri dei tronchi.

6. Sicurezza Server

Priorità: 4

Implementare un sistema di sicurezza che permetta la comunicazione con il server solamente a chi scarica e l'utilizza l'applicazione.

Figura 2: UserStories

Come è possibile notare dalla figura 2, sono state attribuite priorità alle storie e si è dunque passati alla fase successiva. Di seguito è riportato l'elenco delle storie ordinate in base alla priorità.

1. Gestione Mappa (Lato Client) (priorità 1)
2. Gestione Mappa (Lato Server) (priorità 1)
3. Modalità Emergenza (priorità 3)
4. Interfaccia Amministrazione Server (priorità 3)
5. Modalità Ordinaria (priorità 4)
6. Sicurezza Server (priorità 4)

4.2.2 Iteration Planning Game

In questa fase, ogni storia è stata suddivisa in task, riportati nelle apposite card. Nell'assegnare i task, si è fatto in modo di bilanciare il carico di lavoro tra tutti gli sviluppatori. Le fasi di scrittura di test case e pair programming saranno descritte più in dettaglio nelle sezioni relative a ciascuna storia utente.

4.2.3 Iteration to Release

Questa fase viene descritta nelle sezioni successive per ciascuna storia utente.

4.2.4 Interfacce Utente

Nello studio e progettazione delle interfacce utente, si è cercato di renderle il più essenziali possibile, limitando le interazioni che un utente ha con l'applicazione. Ciò è conseguenza dalla natura consultiva dell'applicazione: la sua funzione principe è quella di mostrare il percorso verso una destinazione, che sia un'aula (in modalità ordinaria) o l'uscita più vicina e/o sicura (quando è attiva la modalità di emergenza). Nel far ciò, molte procedure del sistema sono state automatizzate, rendendo molto snelle e minimali le interfacce.

Prototyping

Nel presentare agli stakeholder le schermate che si aveva in mente di costruire, si sono utilizzati mockup di "medium fidelity". L'utilizzo di mockup più sofisticati, che permettessero l'inserimento di vere e proprie funzionalità, è parso inutile per le motivazioni precedentemente scritte. Allo stesso tempo, mockup a bassa fedeltà, sebbene più rapidi da produrre, non sembravano troppo esplicativi.

In figura 3 sono riportati i mockup prodotti.



Figura 3: "Medium Fidelity" Mockup

Per completezza, si riporta in figura 4 altri mockup prodotti relativi a funzionalità dell'applicazione che non riguardano questa release. Nello specifico rappresentano un'idea di interfaccia per le funzioni di registrazione di un nuovo utente, autenticazione di un utente precedentemente registrato, modifica della password e segnalazione di una emergenza.

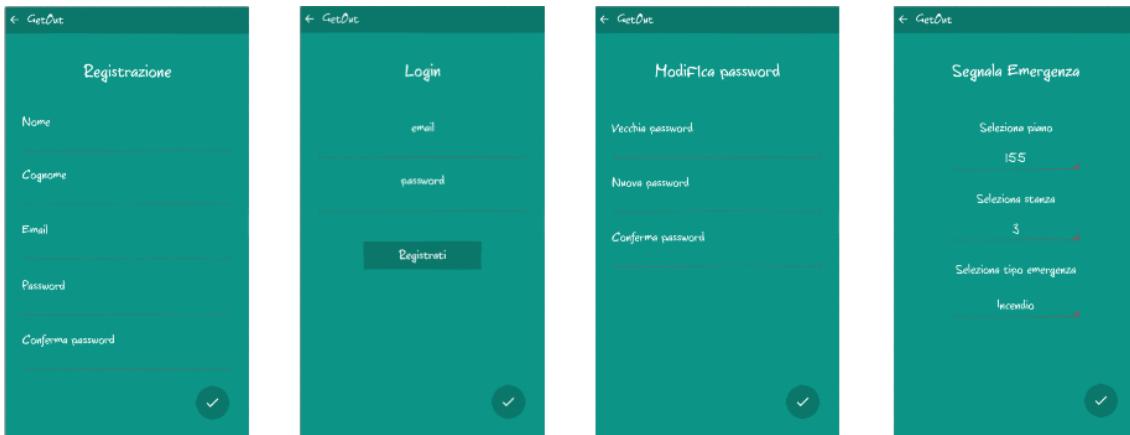


Figura 4: Mockup aggiuntivi

è

5 Tecnologie scelte

5.1 Sistema Android per l'applicativo e Android Studio

Lato client, come da specifiche, l'applicativo deve girare su smartphone con Android (da Android 5 in avanti). Di conseguenza, l'ambiente di sviluppo utilizzato è Android Studio.



(a) Logo Android



(b) Logo Android Studio

Figura 5

5.2 Sistema Linux per il server

Il server, scritto in Java, è stato pensato per lavorare su macchine con Linux, vista la sua grande diffusione tra gli ambienti server e la sua caratteristica open-source. In particolare le due distribuzioni che si possono adoperare sono Ubuntu-Server o CentOS.



Figura 6: Logo di Linux

5.3 IntelliJ e Datagrip

IntelliJ è un IDE per scrivere codice Java. È stato scelto per implementare il Server visto che fornisce molte funzionalità automatiche: analizza il codice, fornisce assistenza con il code completion, esegue analisi degli errori e refactoring.

DataGrip è un IDE per i database. Se JDBC supporta il DBMS utilizzato allora è possibile connettersi al database. Fornisce vari strumenti di DDL (data definition language) e DML (data manipulation language) per la creazione, la modifica e l'analisi della struttura del database.



(a) Logo di IntelliJ



(b) Logo di Data-Grip

5.4 Java e JDK 10

La versione java utilizzata è Java 10.



Figura 7: Logo Java

5.5 Database SQLite e MySQL

SQLite è la tecnologia ufficiale utilizzata da Android per il salvataggio dei dati in formato relazionale. MySQL è la tecnologia utilizzata per il database lato server.

5.6 SensorTag T.I. con B.L.E.

I SensorTag sono sensori IoT utilizzati per lo sviluppo di questo progetto. Essendo sensori, essi hanno integrati al proprio interno diversi moduli per la rilevazione e misurazione di altrettanti parametri e grandezze: barometro, magnetometro, accelerometro, giroscopio, termometro, solo per nominarne alcuni. Essi utilizzano la tecnologia Bluetooth Low Energy (BLE) per la comunicazione con altri dispositivi.

Ai fini dell'applicazione, tuttavia, questi device sono stati utilizzati solamente come beacon: non avviene una connessione vera e propria con conseguente scambio di dati, piuttosto, dopo aver eseguito una scansione Bluetooth filtrata attraverso l'UUID dei device, il client ottiene l'indirizzo MAC dei dispositivi raggiungibili. Conoscendo l'id del beacon più vicino, l'utente saprà, a seguito di elaborazioni del sistema, quale sia la sua posizione all'interno della struttura.

5.7 Bash

Abbiamo utilizzato la Bash Linux per implementare gli script di installazione.

5.8 LaTeX

Per la stesura di questa relazione e di tutti i manuali prodotti si è utilizzato LaTeX.

6 User Story 2: Gestione Mappa (Lato Client)

Questa storia comprende l'implementazione di tutte le funzionalità necessarie per far sì che l'utente possa visualizzare a schermo la mappa del piano dell'edificio in cui si trova e la sua posizione sulla mappa. Come precedentemente specificato, questa era in prima battuta unificata con la storia numero tre (analizzata nella sezione successiva). Tuttavia, essendo troppo onerose l'analisi, la progettazione e l'implementazione delle funzionalità del sistema (client più server) in questo contesto, si è provveduto a separare la parte back-end da quella front-end.

6.1 Storia utente

In figura 8, è riportata la story card con la descrizione delle caratteristiche da implementare.



Figura 8: Storia Utente n. 2

Per riassumere: si vuole che l'applicazione ottenga il MAC address del beacon più vicino tramite Bluetooth (scelto tra gli altri sulla base della potenza del segnale ricevuto (RSSI)); successivamente discriminare il caso online da quello offline e quindi scaricare le informazioni utili al corretto funzionamento dell'app. Per ultimo, si deve mostrare la schermata con mappa e posizione dell'utente. In questa storia viene anche gestita la navigazione dell'utente all'interno dell'edificio e il conseguente hand over tra un beacon e l'altro.

6.2 Divisione in task

In un secondo momento, il team di sviluppo ha provveduto a definire i task necessari per completare la storia, riportati in figura 9.

Release: 1 Story ID: 2 Task ID: 2.1 Task Tag: Schede CRC	Release: 1 Story ID: 2 Task ID: 2.2 Task Tag: Analisi delle UI	Release: 1 Story ID: 2 Task ID: 2.3 Task Tag: Diagramma delle classi
Stesura schede Class-Responsability-Collaboration. Eliminare eventuali schede ridondanti.	Selezionare le schede CRC relative agli utenti e decidere quali responsabilità necessitano di interfaccia. Preparare i mockup. Preparare modelli di dialogo, navigazione e presentazione.	Stesura diagramma delle classi e organizzazione delle stesse in packages.
1 Software Engineer: Jacopo Carloni	2 Software Engineer: Enrico Corradini	3 Software Engineer: Enrico Corradini
Release: 1 Story ID: 2 Task ID: 2.4 Task Tag: Diagrammi dei comportamenti	Release: 1 Story ID: 2 Task ID: 2.5 Task Tag: Visualizzazione mappa	Release: 1 Story ID: 2 Task ID: 2.6 Task Tag: Implementazione bluetooth
Scegliere i diagrammi da produrre tra: -diagramma collaborazioni -diagramma sequenze -diagramma stato -diagramma attività Procedere alla stesura.	Implementare la visualizzazione della posizione dell'utente sulla mappa. Studiare e scegliere le tecnologie necessarie a: far visualizzare una mappa sul dispositivo; evidenziare la posizione dell'utente.	Studiare una modalità di scansione e collegamento dei beacon attraverso l'utilizzo del bluetooth Implementare un metodo di filtro dei dispositivi che non siano beacon
4 Software Engineer: Jacopo Carloni - Andrea Chiorrini	5 Software Engineer: Enrico Corradini	6 Software Engineer: Alessandro Montuccia
Release: 1 Story ID: 2 Task ID: 2.7 Task Tag: Scaricamento dati	Release: 1 Story ID: 2 Task ID: 2.8 Task Tag: Scrittura codice	
Implementazione procedure per l'ottenimento dei dati necessari.	Implementazione classi descritte nel diagramma delle classi.	
7 Software Engineer: Alessandro Montuccia - Edoardo Baldacci - Enrico Corradini	8 Software Engineer: Alessandro Montuccia - Jacopo Carloni	

Figura 9: Task Storia 2

6.3 CRC

L'analisi nome-verbo della story card ha prodotto l'insieme delle schede CRC mostrato in figura 10. Ciascuna di queste classi ha un certo numero di responsabilità, perciò sono state tradotte in

Class: Client Description: attore	Class: Bluetooth Description: non serve per l'interfaccia	Class: Edificio Description: non serve per l'interfaccia	
Responsibility Collaboration	Responsibility Collaboration	Responsibility Collaboration	
Avviare scansione bluetooth	Bluetooth	Ricerca piani edificio	Server,Database
Gestire posizione dell'utente	Posizione,Mappa	Conoscere piani edificio	Piano
Class: Server Description: solo se online, non serve per l'interfaccia	Class: Database Description: solo se offline, non serve per l'interfaccia	Class: Piano Description: non serve per l'interfaccia	
Responsibility Collaboration	Responsibility Collaboration	Responsibility Collaboration	
Scaricamento dati edificio	Lettura dati edificio	Ricerca tronchi del piano	Server,Database
Scaricamento dati piani	Lettura dati piani	Conoscere tronchi del piano	Tronco
Scaricamento dati aule	Lettura dati aule	Ricerca aule del piano	Server,Database
Scaricamento dati tronchi	Lettura dati tronchi	Conoscere aule del piano	Aula
Scaricamento dati beacon	Lettura dati beacon	Conoscere la mappa (immagine) del piano	
Scaricamento mappe	Lettura mappe	Ricerca mappa (immagine) del piano	Server,Database
Scaricamento posizione utente	Lettura posizione utente	Conoscere la posizione del beacon	
Class: Tronco Description: non serve per l'interfaccia	Class: Aula Description: non serve per l'interfaccia	Class: Beacon Description: non serve per l'interfaccia	
Responsibility Collaboration	Responsibility Collaboration	Responsibility Collaboration	
Ricerca beacon del tronco	Conoscere il beacon d'entrata dell'aula	Conoscere la posizione del beacon	
Conoscere beacon del tronco	Ricerca aule del piano		
Class: Mappa Description: serve per l'interfaccia	Class: Posizione Description: non serve per l'interfaccia		
Responsibility Collaboration	Responsibility Collaboration		
Visualizzare la mappa (immagine) del piano	Conoscere beacon connesso		
Disegnare la posizione dell'utente	Conoscere edificio relativo alla posizione		
	Conoscere il piano relativo alla posizione		

Figura 10: CRC Storia 2

classi di analisi. Una nota spetta alla classe *Beacon*: sebbene essa abbia un'unica responsabilità, potrebbe per tanto essere eliminata e considerata come attributo. Si è convenuto mantenerla come classe in quanto essenziale, insieme alle classi *Tronco, Piano, Aula, Edificio*, per modellare un'intera struttura gestita dal sistema.

6.4 Mockup

Per quanto riguarda i mockup prodotti, si rimanda il lettore alla sottosezione 4.2.4

6.5 Diagrammi di dialogo e navigazione

Nei diagrammi riportati in figura 11, si può notare la sequenza di apparizione delle varie schermate (i *fragment* di Android) e le azioni che l'utente può compiere su ciascuna di esse.

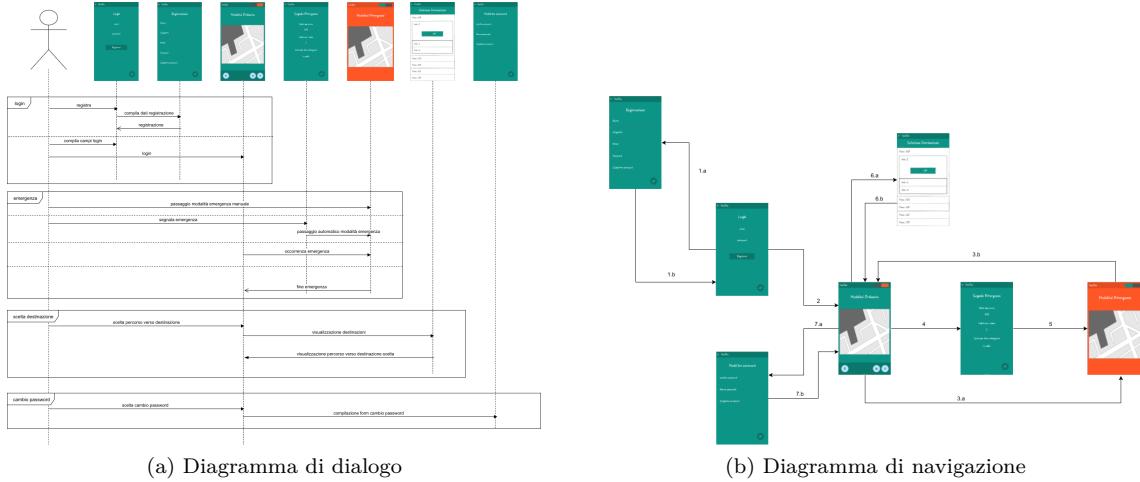


Figura 11

6.6 Diagramma delle classi di analisi

In figura 12 è riportato il diagramma delle classi di analisi derivato dalla riorganizzazione delle schede CRC.

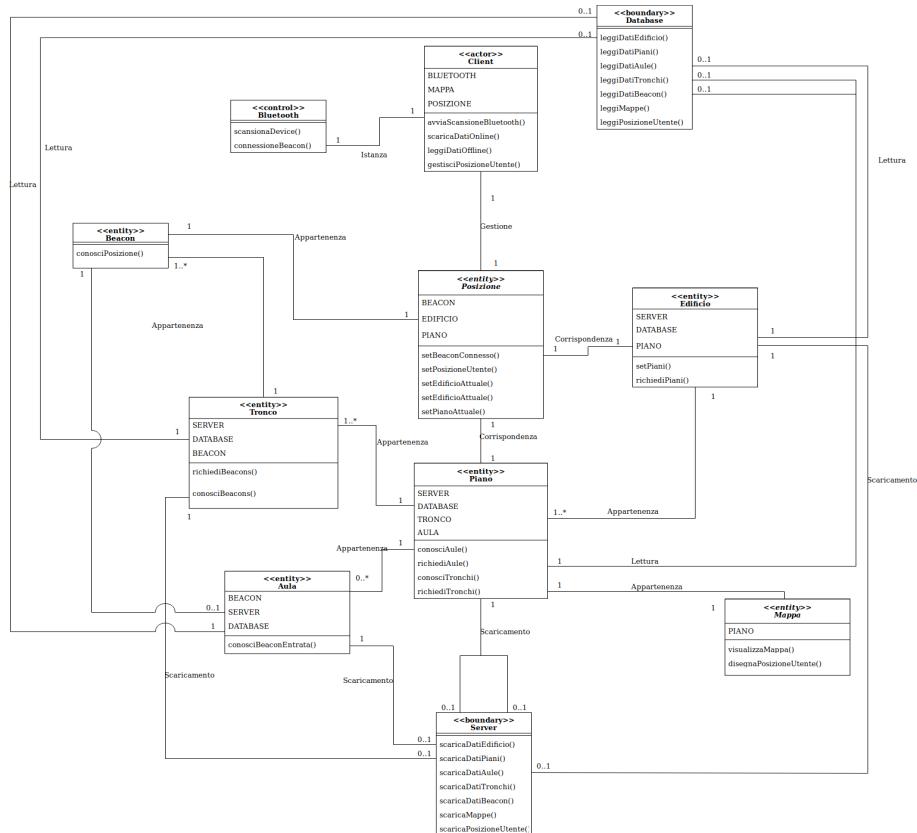


Figura 12: Diagramma classi analisi storia 2

Le classi sono state etichettate con degli stereotipi per specificare la loro funzione:

- Actor: in questo caso solamente la classe *Client*;
- Entity: *Beacon*, *Aula*, *Tronco*, *Piano*, *Edificio*, *Mappa*, *Posizione*. Queste classi mappano i dati che devono essere gestiti per implementare le funzionalità relative alla storia.
- Boundary: *Server*, *Database*. Tali classi sono di interfaccia rispettivamente con il server e con il database locale dello smartphone.
- Control: in questo contesto solamente la classe *Bluetooth* è stata così stereotipata perché utile a descrivere il comportamento del sistema.

Le classi *Server* e *Database* di fatto hanno le stesse responsabilità ma i loro metodi saranno eseguiti in mutua esclusione. Infatti, in base ai requisiti di progetto, l'applicazione lato client deve di norma funzionare connessa alla rete LAN cui è connesso il server. In tal caso lo scaricamento dei dati e il calcolo della posizione dell'utente avviene tramite comunicazioni con il server. In aggiunta a ciò, è previsto che l'app riesca a funzionare anche offline: al primo avvio scarica il contenuto del database remoto e lo inserisce all'interno di quello locale (SQLite). Quando l'applicazione è utilizzata senza connessione a Internet, i dati usati sono quelli del database locale (sebbene possano non essere completamente aggiornati).

Un edificio è modellato tramite le classi *Edificio*, *Piano*, *Tronco*, *Aula* e *Beacon*. Ad un edificio appartengono uno o più piani, un piano è costituito da un insieme di tronchi e aule, un tronco è costituito da uno o più beacon. Ad un aula è associato esattamente un beacon: quello posto all'entrata. Se un'aula è distribuita su più piani, ad ogni entrata corrisponde un beacon.

Posizione è una classe statica composta da: edificio in cui si trova l'utente, piano dell'edificio e beacon del particolare piano. La posizione è in relazione uno a uno con la classe *Client*, che rappresenta l'unica *Activity* (quindi la *Main*) dell'applicazione Android. Ad un client è associato una ed una sola classe *Bluetooth* utilizzata per la scansione dei dispositivi nell'area circostante ed il conseguente rilevamento dell'indirizzo MAC del beacon più vicino.

6.7 Diagramma dei package

In figura 13, è riportato lo schema che rappresenta il modo in cui le classi sono state ordinate e raggruppate a formare i package.

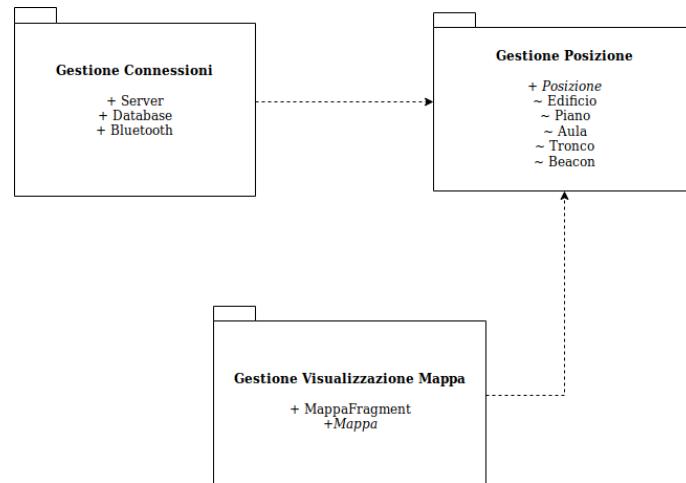


Figura 13: Diagramma dei package storia 2

Più in dettaglio, il package *Gestione Connessioni* comprende le classi che gestiscono la connessione al server, al database locale e con i beacon tramite Bluetooth.

Nel package *Gestione Posizione* si trovano le classi necessarie per il calcolo della posizione all'interno dell'edificio, il package *Gestione Visualizzazione Mappa* contiene le classi utili per l'istanziazione e disegno della mappa a schermo.

6.8 Diagramma delle classi di progettazione

In questa sezione verranno analizzate le modalità con cui si è passati dal diagramma delle classi di analisi a quello più dettagliato delle classi di progettazione, rappresentato in figura 14.

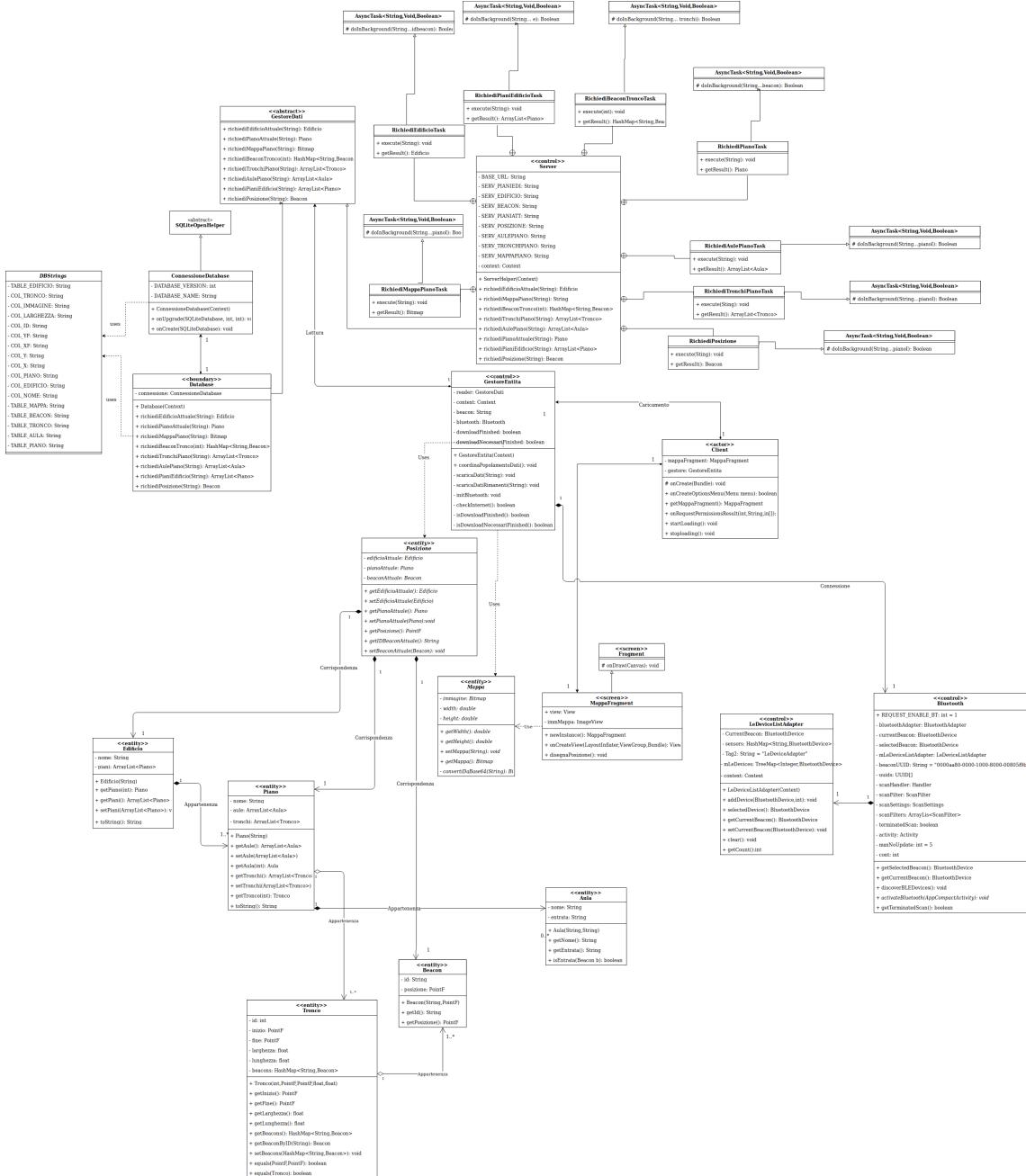


Figura 14: Diagramma delle classi di progettazione storia 2

La classe *Gestore Entità* è la classe principale di questo schema di progettazione in quanto è quella che supervisiona l'esecuzione di tutta l'applicazione: inizializza la scansione Bluetooth, che viene ripetuta ad intervalli regolari in background finché non viene ucciso il processo dell'applicazione stessa; una volta ottenuto l'ID del beacon più vicino, gestisce l'istanziazione delle componenti dell'edificio (al primo avvio) e il rilevamento della posizione dell'utente. Per svolgere le sue funzioni, essa instanzia oggetti di altre classi.

Un primo aspetto interessante da analizzare riguarda l'implementazione della classe astratta *Gestore Dati*. Poiché, come ampiamente descritto in precedenza, l'applicazione può funzionare sia online che offline, essa definisce i metodi di scaricamento dei dati, ma, come ogni classe astratta, delega l'implementazione di questi alle sue sottoclassi, in questo caso *Server* e *Database*, a seconda

che lo smartphone sia connesso o meno in rete. Perciò, all'interno della classe *Gestore Entita*, tra le altre cose, si controlla se c'è connessione: nel caso online verrà istanziata la classe *Server*, nel caso offline la classe *Database*. Di fatto si è implementato il design pattern comportamentale chiamato *State*.

Nel caso online, la classe *Server* implementa tutti i metodi di scaricamento dei dati dal server tramite gli *AsyncTask*, i quali permettono l'esecuzione di attività in background e non sono bloccanti. La classe *Database*, invece, esegue lo scaricamento per mezzo di *Connessione Database*, definita come sottoclasse della classe astratta di Android *SQLiteOpenHelper*.

Per quanto riguarda la connessione Bluetooth, *Gestore Entita* istanzia un oggetto della classe *Bluetooth*: quest'ultima si occupa di avviare una scansione molto rapida dei dispositivi BLE ed al termine costruisce una struttura dati in cui inserisce i device rilevati, sfruttando la classe *LeDeviceListAdapter*. Tra questi viene restituito quello con RSS più alto (e di conseguenza più vicino allo smartphone dell'utente). Quindi viene calcolata la posizione, scaricati i dati e definiti l'edificio, il piano ed il beacon della classe *Posizione* e contemporaneamente scaricata la mappa del piano per poi essere mostrata a schermo con la posizione.

6.9 Diagramma delle architetture

Essendo una storia incentrata esclusivamente sulle funzionalità del sistema lato client, non sono stati prodotti diagrammi delle architetture che mostrassero le relazioni e connessioni tra client e server.

6.10 Diagramma delle sequenze

In figura 15 è riportato il diagramma delle sequenze elaborato per la storia corrente. Riassumendo il flusso degli eventi, l'applicazione svolge le seguenti operazioni:

1. Controllo connessione Internet e istanziazione della classe Server o Database;
2. Avvio scansione Bluetooth e ottenimento dell'indirizzo MAC del beacon più vicino;
3. Scaricamento iniziale, cioè al primo avvio, dei dati da server/ database locale sulla base del beacon passato e istanziazione del piano, tronchi e aule di tale piano;
4. Disegno della mappa del piano e della posizione su di essa;
5. Scaricamento e istanziazione dei dati rimanenti (gli altri piani dell'edificio e i loro tronchi e aule).

Come descritto in precedenza, la scansione Bluetooth viene ripetuta a intervalli regolari finché rimane in esecuzione il processo dell'applicazione in modo da gestire la navigazione dell'utente all'interno dell'edificio. Qualora l'app si colleghi ad un altro beacon, viene rieseguita l'elaborazione per l'ottenimento delle coordinate del beacon e ridisegnata la mappa con la nuova posizione.

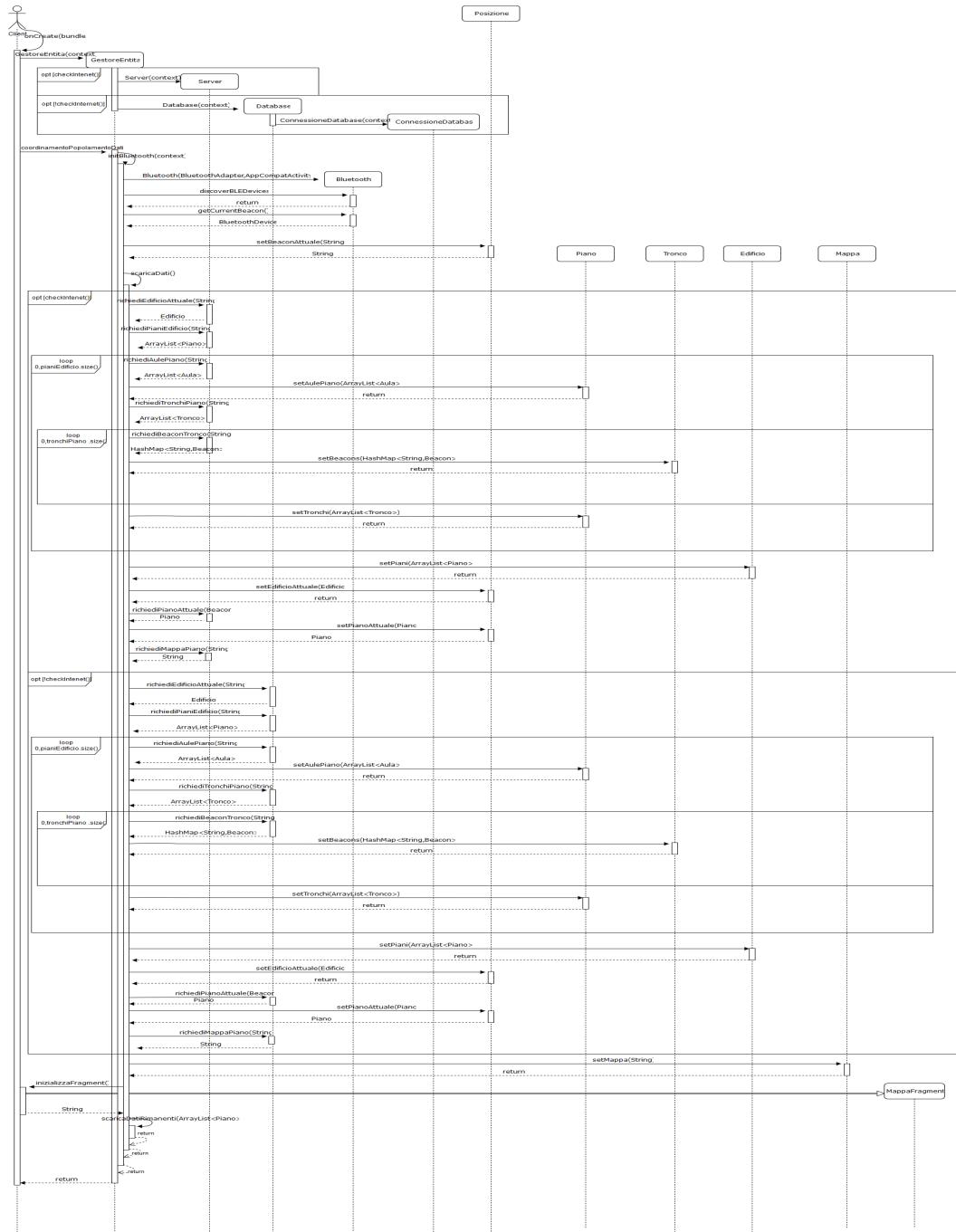


Figura 15: Diagramma delle sequenze storia 2

6.11 Casi di Test

Questa storia utente risulta particolarmente collegata alla storia "Gestione Mappa (Lato Server)", perciò la correttezza delle funzionalità di questa storia in parte dipendono dalla correttezza di ciò che viene implementato nella storia tre. Poiché inoltre in questa storia non è previsto l'input di dati da parte dell'utente, i test si riducono ai due casi di funzionamento dell'app online e offline. Per testare il funzionamento dell'applicazione in modalità offline, tuttavia, è necessaria l'implementazione della storia uno (sulla modalità ordinaria), perché essa introduce la funzionalità di popolamento del database locale dello smartphone.

- **Test Case:** Caso Online: visualizzazione mappa e posizione.

1. Avviare il Server.
2. Popolare il database con dati di prova.
3. Avviare l'applicazione.
4. Se al primo avvio, concedere il permesso all'app di accedere alla posizione e attivare il Bluetooth se non attivo.
5. Attendere il processo di elaborazione e verificare che l'app visualizzi correttamente l'immagine della mappa con la posizione.

7 User Story 3: Gestione Mappa (Lato Server)

7.1 Storia utente

3. Gestione Mappa (Lato Server)

Priorità: 1

Implementare tutte le funzionalità del server relative alla gestione delle mappe:

1. Restituire tutti i dati memorizzati nel database, a cui è connesso, relativi all'edificio, le mappe, i tronchi e le aule di ogni piano e i beacon.
2. Aggiornare numero di utenti collegati a beacon.
3. Restituire la posizione del beacon a cui l'utente è collegato.

Figura 16: Storia Utente n. 3

Come si nota dalla figura 16, la storia numero 3 si occupa della progettazione e dell'implementazione delle funzionalità del server. Le principali funzionalità sono le seguenti:

1. la restituzione di tutti i dati presenti nel database, relativi all'edificio attuale, che saranno richiesti al Client.
2. l'aggiornamento automatico del numero di utenti collegati a un dato beacon.
3. la restituzione della posizione di un utente a partire dal beacon a cui è collegato.

7.2 Divisione in task

Riportiamo in figura 17 la suddivisione in task della storia utente.

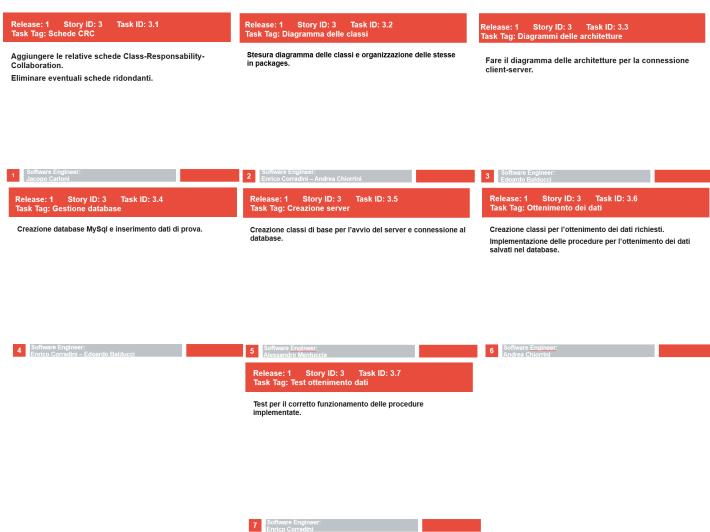


Figura 17: Risultato dell'iteration planning game Storia 3

7.3 CRC

A questo punto abbiamo compiuto l'analisi nome-verbo nella storia utente per ottenere le schede CRC mostrate in figura 18. Si noti come alcune CRC (**Dati**, **Posizione** e **Utente**) non hanno né responsabilità né collaboratori e pertanto vanno eliminate. In riferimento alle classi non eliminate possiamo dire che:

- **Server** ha parecchie responsabilità e molti collaboratori diversi pertanto abbiamo ritenuto di considerarlo un actor.
- **Database** si occupa di gestire la connessione con il database e perciò abbiamo ritenuto di considerarlo un boundary.
- tutte le altre non selezionate per essere eliminate abbiamo notato che si occupavano esclusivamente della lettura o della scrittura dei dati e perciò le abbiamo considerate entity.

Class Server		Class Edif		Class Database	
Description: attore		Description: da eliminare perché non ha responsabilità		Description: non serve per l'interfaccia	
Responsability	Collaboration	Responsability	Collaboration	Responsability	Collaboration
Riistruzione dell'edificio	Edificio	-	-	Gestire connessione al database	-
Riistruzione mappa	Mappa	-	-	-	-
Riistruzione del tronco	Tronco	-	-	-	-
Riistruzione del piano	Piano	-	-	-	-
Riistruzione posizione beacon	Beacon	-	-	-	-
Aggiornamento numero utenti	Beacon	-	-	-	-
Class Mappa		Class Edifici		Class Tronco	
Description: non serve per l'interfaccia		Description: non serve per l'interfaccia		Description: non serve per l'interfaccia	
Responsability	Collaboration	Responsability	Collaboration	Responsability	Collaboration
Lettura mappa nel database	Database	Lettura edifici nel database	Database	Lettura tronchi nel database	Database
Class Aula		Class Piano		Class Beacon	
Description: non serve per l'interfaccia		Description: non serve per l'interfaccia		Description: non serve per l'interfaccia	
Responsability	Collaboration	Responsability	Collaboration	Responsability	Collaboration
Lettura aule nel database	Database	Lettura piani nel database	Piano	Lettura beacon nel database	Database
				Lettura posizione utente nel database	Database
				Aggiornamento nel database del numero utenti collegati al beacon	Database
Class Utente		Class Posizione			
Description: da eliminare perché non ha responsabilità		Description: da eliminare perché non ha responsabilità			
Responsability	Collaboration	Responsability	Collaboration		
-	-	-	-		

Figura 18: Schede CRC ottenuti dall'analisi Storia 3

7.4 Diagramma delle classi di analisi

Dopo l'analisi nome-verbo e la definizione delle schede CRC abbiamo steso il diagramma delle classi di analisi, visibile in figura 19

Nel diagramma delle classi di analisi troviamo nuovamente le classi definite dall'analisi CRC. Sono state identificate tre categorie all'interno di questo diagramma: **actor** (Server), **entity** (Edificio, Piano, Aula, Tronco, Mappa, Beacon) e **boundary** (Database). Abbiamo ottenuto i metodi di ciascuna classe dalle responsabilità e le varie associazioni dalle collaborazioni.

7.5 Diagramma dei package

A partire dalle categorie ottenute in precedenza, abbiamo stilato il diagramma dei package della storia visibile in Figura 20. In questo diagramma è inoltre possibile individuare dipendenze tra vari package. In particolare si nota come sia il package Connessione a dipendere da Entità essendo Connessione a necessitare dei dati conservati nel Data Layer gestito da Entità per preparare i dati che gli vengono richiesti dall'app.

7.6 Diagramma delle classi di progettazione

Dopo aver applicato le varie regole, controllato le classi ed i metodi già esistenti, abbiamo pensato le classi di progettazione, come descritto in Figura 21. In primo luogo è interessante notare come abbiam provveduto a suddividere l'attore Server in Server e JSONServer per garantire il principio

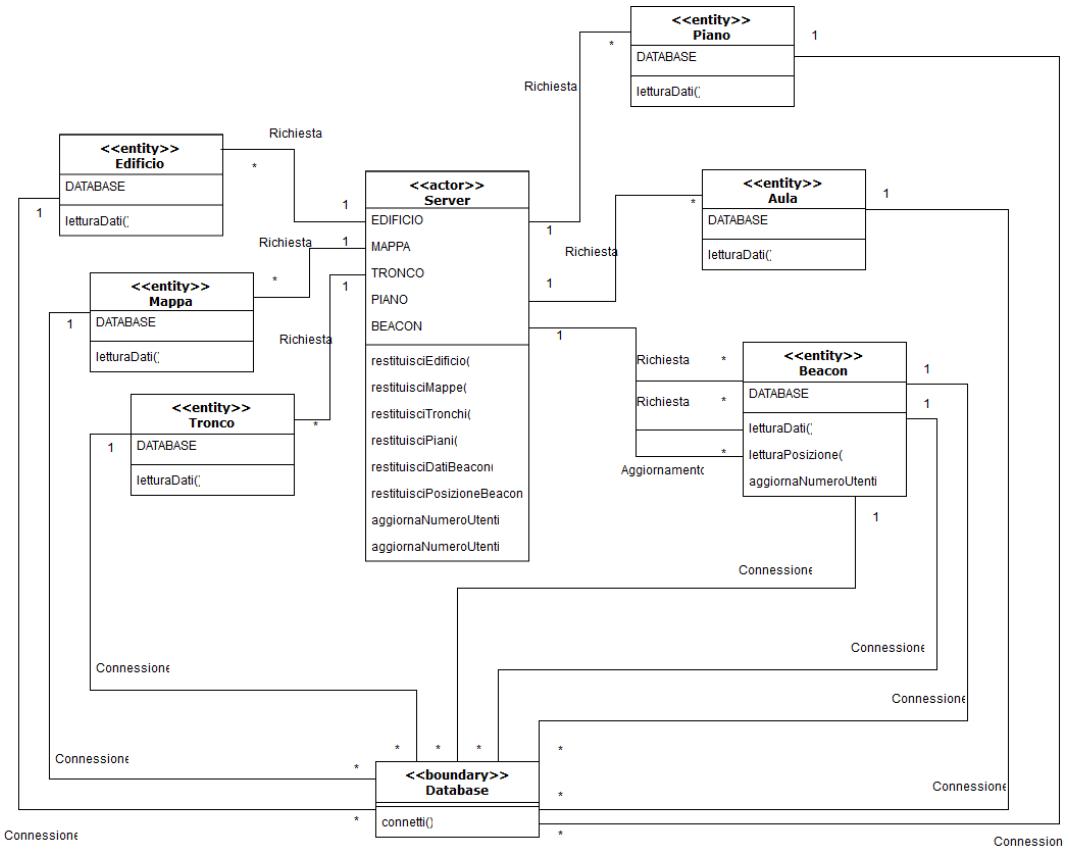


Figura 19: Diagramma delle classi di analisi Storia 3

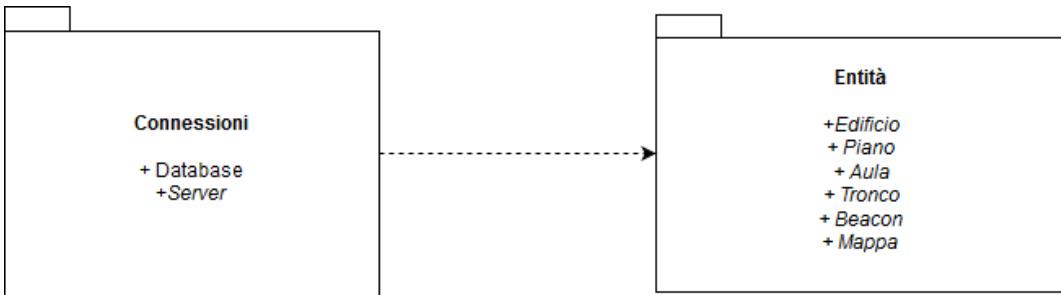


Figura 20: Diagramma dei Package della Storia 3

di singola responsabilità. L’ altro aspetto, interessante da notare, è che abbiamo adottato il DAO come pattern architettonale per questa parte. L’uso del DAO ci ha permesso di stratificare e isolare l’accesso ad ogni tabella ponendo le query all’interno dei metodi della classe: in questo modo il data layer rimane separato dalla business logic creando un maggiore livello di astrazione ed una più facile manutenibilità.

Più nel dettaglio:

- **DAO** su ogni classe dao abbiamo implementato come metodi le query necessarie al funzionamento dell’ applicativo, la maggior parte delle quali sono delle select secondo un certo campo specificato in input. Per esempio **sumUser(String)** si occupa di ritornare la somma di tutti gli utenti connessi a un beacon, fatto che sarà fondamentale per calcolare poi il Los nelle storie successive.
- **Server** è la classe principale richiamata all’avvio iniziale, col metodo main: inizia la connessione al database, manda un broadcast nella rete con l’indirizzo IP e avvia il server JSON per accettare le richieste dai client.

- **JSONServer** si occupa di gestire le varie richieste http che può ricevere da parte dei client, rimane in ascolto per le richieste attraverso un thread di lavoro in background.
- **Database** è la classe di interfacciamento con il database, attraverso il metodo `getConn()` ritorna la connessione al database utilizzata da tutti i DAO.

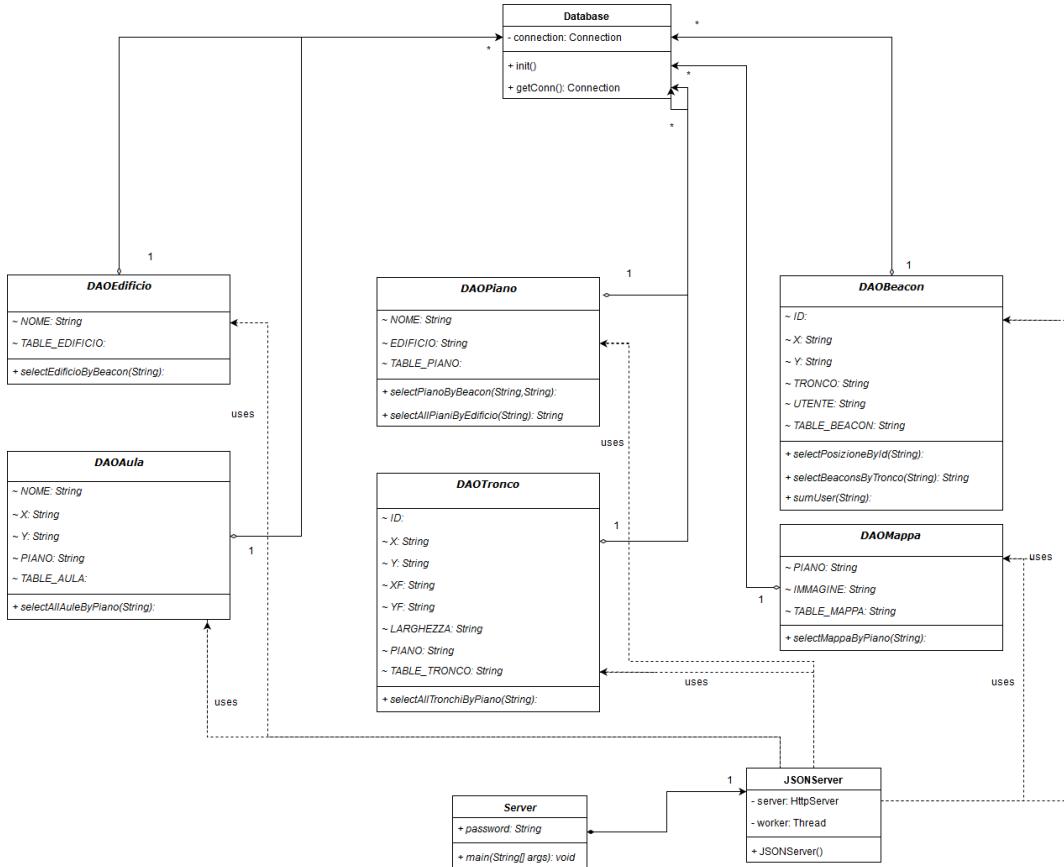


Figura 21: Diagramma delle classi di progettazione della Storia 3

7.7 Diagramma delle architetture

Riportiamo in figura 22 il diagramma delle architetture relativo a questa storia e la storia 2 essendo le due strettamente collegate a livello di funzionamento. Si fa notare che la parte di diagramma strettamente relativa a questa storia ha fondamentalmente due componenti software, **connessioni** che si occupa di gestire le comunicazioni col client e **entità** che invece si occupa di ottenere e gestire i dati nel database: come abbiamo già detto nella progettazione si è cercato di mantenere il massimo disaccoppiamento possibile tra i componenti attraverso l'uso dei DAO.

7.8 Diagramma delle sequenze

Per questa storia utente non si è ritenuta necessaria la stilatura di un diagramma delle sequenze.

7.9 Casi di Test

Si sono svolti i seguenti casi test, si noti che questa storia risulta strettamente collegata alla storia 2 soprattutto inerentemente al funzionamento.

- **Test Case 1:** Accettazione di richieste da parte del Server

1. Avviare il Server.

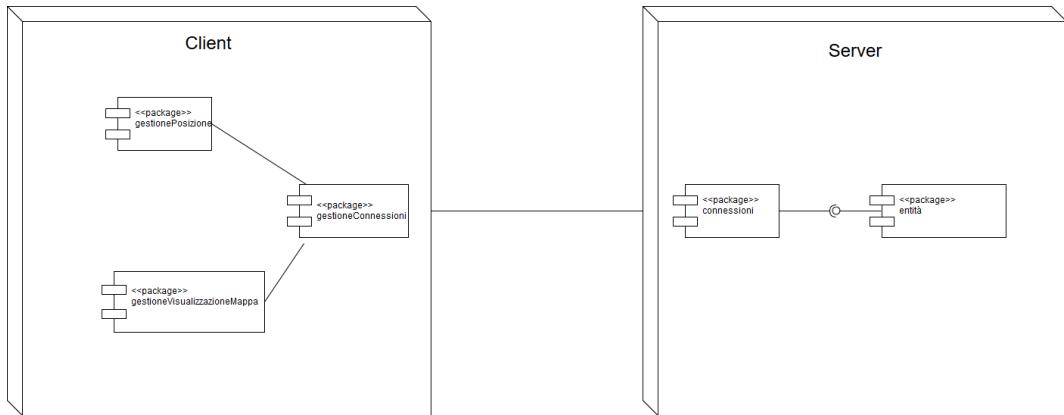


Figura 22: Diagramma delle architetture della Storia 3

2. Inviare da browser una richiesta al Server.
3. Verificare che il Server risponda.

- **Test Case 2:** Richiesta dati da parte del Client

1. Avviare il Server.
2. Inserire dei dati di prova nel database del Server.
3. Lanciare la parte di app sviluppata nella storia 2.
4. Attendere il caricamento.
5. Verificare che i dati siano correttamente visualizzati dall'app.

- **Test Case 3:** Aggiornamento numero di utenti collegati a un beacon

1. Avviare il Server.
2. Lanciare la parte di app sviluppata nella storia 2.
3. Allontanare lo smartphone dal beacon sconnettendosi.
4. Verificare l'aggiornamento sul database del Server del numero di utenti collegati a ciascun beacon coinvolto.

- **Test Case 4:** Visualizzazione mappa

1. Avviare il Server.
2. Inserire dei dati di prova nel database del Server.
3. Lanciare la parte di app sviluppata nella storia 2.
4. Attendere il caricamento.
5. Verificare che l'app visualizzi correttamente l'immagine della mappa.

8 User Story 4: Modalità di emergenza

8.1 Storia utente



Figura 23: Storia Utente n. 4

Come si può notare dalla figura 23, la story card numero 4 descrive la progettazione e l'implementazione delle funzionalità che vengono richiamate nel momento in cui si verifica una emergenza. La storia utente si divide in due fasi. Nella prima vengono descritte le funzionalità lato Client:

1. visualizzazione della notifica di emergenza ricevuta;
2. dopo aver premuto sulla notifica, si avvia la richiesta del percorso dalla posizione dell'utente verso l'uscita più vicina;
3. il percorso viene richiesto ogni qualvolta si verificano variazioni nelle condizioni dell'edificio, tra cui ostruzioni nelle vie d'uscita;
4. si effettuano controlli per conoscere se l'utente è uscito dall'edificio;
5. la posizione dell'utente e il percorso vengono aggiornati durante lo spostamento;
6. il percorso e la posizione dell'utente vengono visualizzate sulla mappa;
7. alla fine dell'emergenza viene ricevuta una notifica di rientro del pericolo.

Mentre nella seconda parte della storia utente si va a considerare il lato Server:

1. si invia la notifica di emergenza a tutti gli utenti che fanno parte del sistema;
2. per ogni utente viene calcolato il percorso verso l'uscita più vicina;
3. aggiornamento del percorso in base alle più recenti condizioni dell'edificio;
4. rilevazione del rientro dell'emergenza e la segnalazione tramite notifica agli utenti interessati.

8.2 Divisione in task

Riportiamo in figura 24 la suddivisione in task della storia utente.

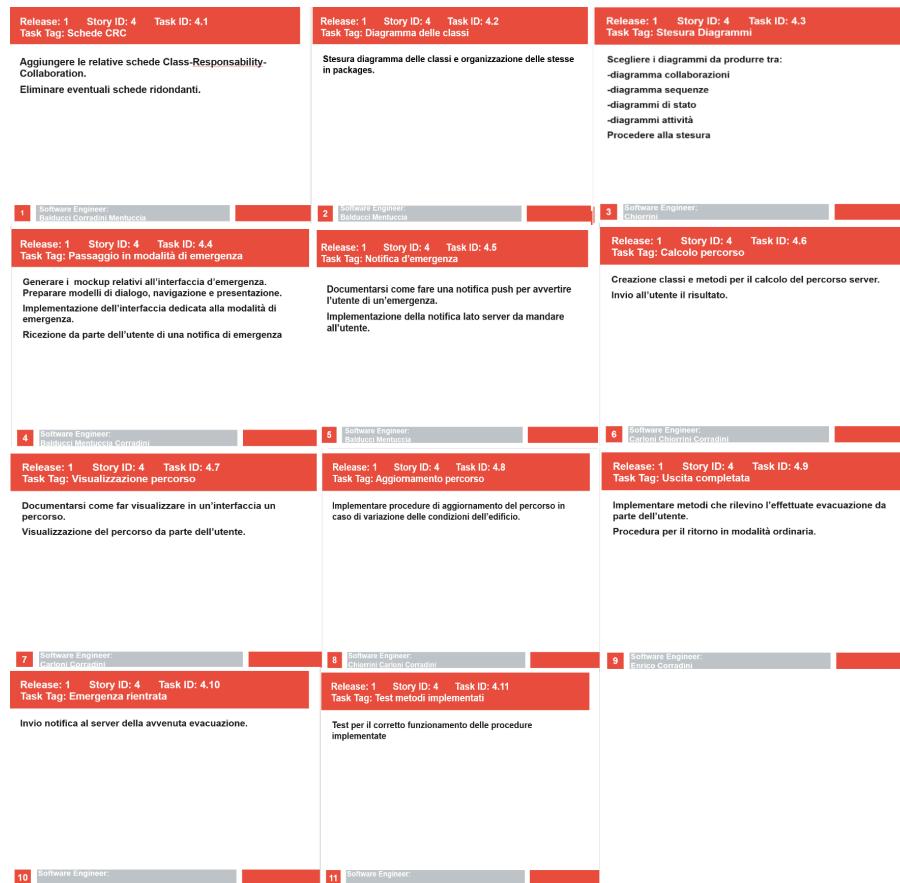


Figura 24: Risultato dell'iteration planning game Storia 4

8.3 CRC

Dopo aver ottenuto le task card, abbiamo effettuato l'analisi nome/verbo della storia utente. Il risultato è rappresentato nella figura 25: Come si può osservare, l'analisi CRC ha prodotto, in particolare, due classi che contraddistinguono gli attori (**Client**, **Server**) e inoltre una classe di tipo **Screen** che rappresenta l'interfaccia grafica, utile alle funzionalità della modalità emergenza. Analizzando invece nel dettaglio il risultato di questa fase abbiamo:

1. due classi attori:
 - **Client** rappresenta l'attore che implementa le funzionalità del cliente, ovvero la ricezione della notifica, la visualizzazione del percorso e la gestione dell'emergenza.
 - **Server** fa riferimento al attore di sistema che gestirà il calcolo del percorso, la notifica dell'emergenza ed il suo rientro;
2. **Database** è la classe che si occuperà della connessione al database perciò verrà considerata un boundary;
3. **Notifica** è la classe che si farà carico della creazione della notifica di inizio emergenza, l'invio al client, la creazione della notifica di rientro emergenza e il relativo invio;
4. **Emergenza** è la classe che si occuperà della visualizzazione delle funzionalità che caratterizzano la modalità emergenza, come la visualizzazione della mappa;
5. **Mappa** rappresenta la classe che avrà il compito di disegnare il percorso e la posizione utente sulla mappa dell'edificio;

Class: Client		Class: Server	
Description: attore		Description: attore	
Responsability	Collaboration	Responsability	Collaboration
Ricevere notifica emergenza	Notifica	Notificare inizio emergenza	Notifica
Gestire Emergenza	Emergenza	Calcolo percorso	Database
Visualizzazione percorso	Mappa	Restituire condizioni edificio	Database
		Rilevazione rientro emergenza	Database
		Notificare rientro emergenza	Notifica
Class: Database		Class: Notifica	
Description: non serve per l'interfaccia		Description: non serve l'interfaccia	
Responsability	Collaboration	Responsability	Collaboration
Apri connessione database		Creazione notifica	
Chiudi connessione database		Invio notifica emergenza	Client
		Invio notifica rientro emergenza	Client
Class: Emergenza		Class: Mappa	
Description: serve per l'interfaccia		Description: non serve per l'interfaccia	
Responsability	Collaboration	Responsability	Collaboration
Transitare in modalità di emergenza		Disegnare percorso	Percorso
		Disegnare posizione	Posizione
Class: Posizione		Class: Percorso	
Description: non serve per l'interfaccia		Description: non serve per l'interfaccia	
Responsability	Collaboration	Responsability	Collaboration
Conoscere la posizione dell'utente		Conoscere il percorso verso l'uscita	
Richiedere aggiornamento posizione	Server	Richiedere percorso	Server
Class: Edificio		Class: Uscita	
Description: non serve per l'interfaccia		Description: non serve per l'interfaccia	
Responsability	Collaboration	Responsability	Collaboration
Richiedere condizioni edificio	Server	Controllare collegamento beacon	Posizione
Conoscere condizioni edificio		Conoscere ultimo beacon attraversato	
		Controllare se ultimo beacon è beacon di uscita	Server

Figura 25: Schede CRC ottenute dall'analisi Storia 4

6. **Posizione** è la classe che tiene traccia della posizione utente
7. **Percorso** si occupa di richiedere il percorso al Server e di conseguenza ha l'informazione riguardante l'uscita più vicina
8. **Edificio** è la classe che possiede le condizioni dell'edificio
9. **Uscita** conosce la posizione del beacon di uscita e si occupa di controllare se l'utente c'ha transitato o no.

8.4 Diagramma delle classi di analisi

Dopo aver fatto l'analisi nome/verbo e definito le classi dell'analisi CRC, siamo passati a modellare il diagramma delle classi di analisi. Per avere una rappresentazione migliore del risultato, abbiamo suddiviso l'analisi in due diagrammi distinti: il primo riguardante le classi definite per il Client ed il secondo per le classi del Server. In figura 26 e figura 27 vengono riproposti i due diagrammi ottenuti.

Nel diagramma delle classi di analisi lato Client ritroviamo le classi definite dall'analisi CRC che riguardavano il Client. Sono state identificate due categorie all'interno di questo diagramma: classi di tipo **control** (Server, Edificio, Posizione, Percorso, Emergenza, Mappa, Notifica, Uscita) e **actor** (Client). Inoltre, dalle responsabilità abbiamo ottenuto i metodi di ciascuna classe e dalle collaborazioni le varie associazioni. Riguardo l'analisi lato server, abbiamo ottenuto dall'analisi CRC cinque classi in grado di rappresentare la logica della User Story 4, ovvero le classi:

1. **Edificio** «control»: servirà per aggiornare le condizione dell'edificio;
2. **Posizione** «control»: si occuperà di aggiornare la posizione dell'utente;

3. **Notifica** «control»: gestirà le comunicazioni di inizio e fine emergenza. Sarà l'unica classe aggiunta che si interfacerà col Client;
4. **Database** «Boundary»: si occuperà di aprire e chiudere la comunicazione col Server;
5. **Server** «actor»: sarà la classe che coordinerà le attività del server.

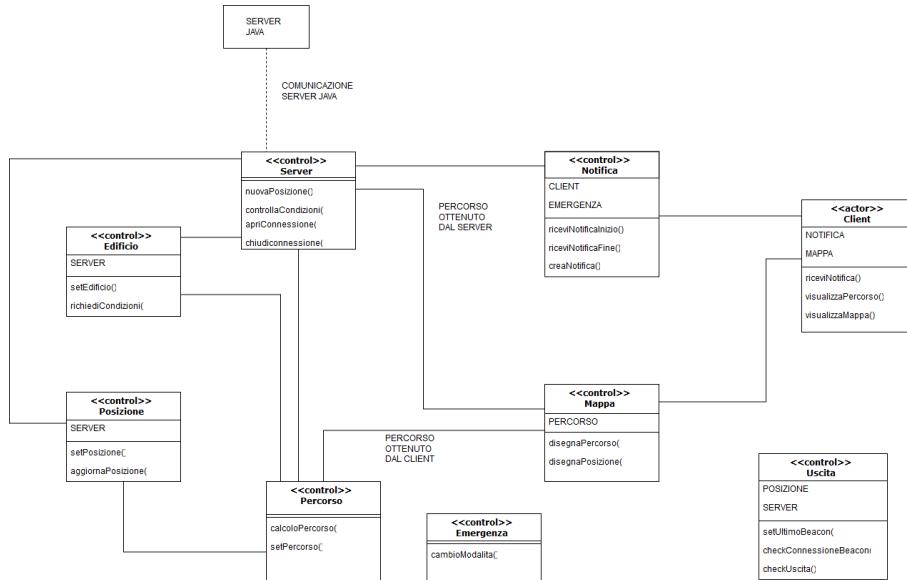


Figura 26: Diagramma delle classi d'analisi lato Client

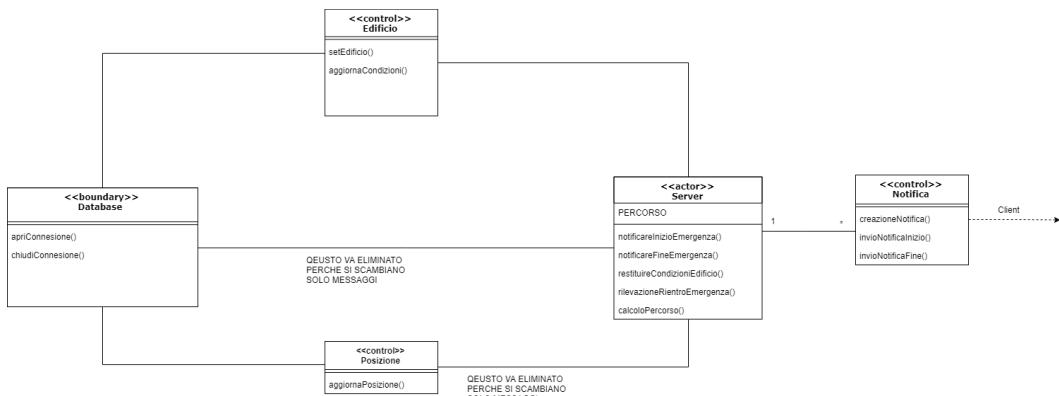


Figura 27: Diagramma delle classi d'analisi lato Server

8.5 Diagramma dei package

In questa sezione analizzeremo il diagramma dei package, in figura 28, che abbiamo deciso di adottare. Come si può notare, abbiamo raggruppato le nostre classi in 4 package e suddivise in base alla categoria che rappresentavano. L'obiettivo di tale scelta era di ottimizzare la coesione interna tra le classi ed avere la minima interdipendenza tra i package.
In particolare, le dipendenze che risultano dal diagramma sono:

1. la dipendenza tra il package **UI**, che si occupa dell'interfaccia grafica, ed il package **Actor**;
2. la dipendenza tra il package **Control**, che definisce la logica della storia 4, ed il package **UI**;
3. la dipendenza tra il package **Actor** ed il package **Control**;
4. la dipendenza tra il package **Boundary**, che fornisce le funzioni per accedere ai dati del database, ed il package **Actor**.

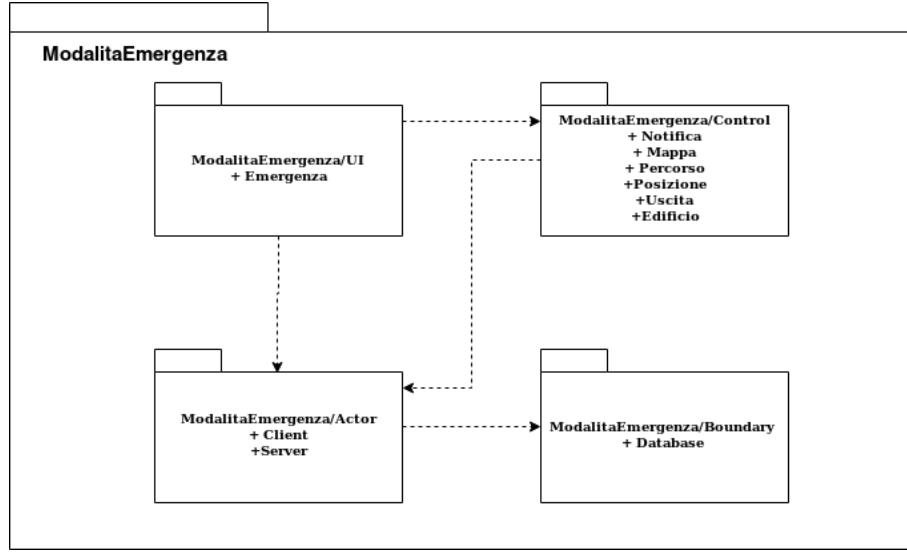


Figura 28: Diagramma dei package

8.6 Diagramma delle classi di progettazione

In questa fase, come per la fase dell'analisi delle classi, abbiamo diviso il lavoro in progettazione delle classi lato client e progettazione delle classi lato Server. Il risultato è riportate nelle figure 29 e 30.

8.6.1 Progettazione lato Client

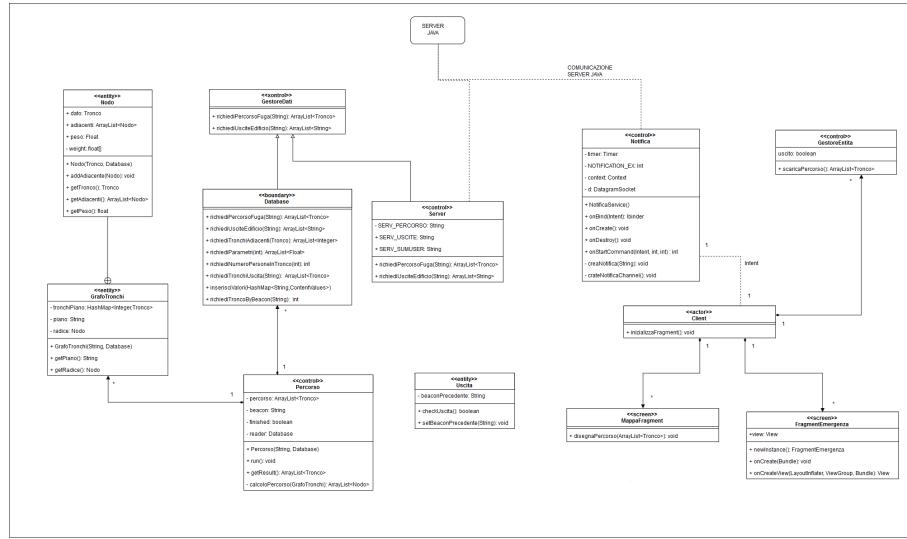


Figura 29: Diagramma delle classi di progettazione lato Client

In questa fase, partendo dal diagramma delle classi realizzato durante l'analisi, abbiamo applicato le varie regole, controllato le classi ed i metodi già esistenti, così da avere come risultato finale il diagramma delle classi di progettazione. In primo luogo, si può osservare come il diagramma sia composto da classi per l'interfaccia grafica (**MappaFragment**, **FragmentEmergenza**), classi Control(**GestoreDati**, **Percorso**, **Server**, **Notifica**, **GestoreEntita**), classi Actor (**Client**), classi Boundary (**Database**) e classi Entity (**Uscita**, **GrafoTronchi**, **Nodo**).

Dato che questa storia descrive la gestione della modalità di emergenza, abbiamo ideato delle classi che permettono la ricezione della notifica a lato client e il calcolo del percorso verso l'uscita più vicina. Le classi che si occupano di queste funzionalità sono:

1. **FragmentEmergenza:** classe che estende Fragment. Il suo scopo è di contenere l’interfaccia grafica che verrà mostrata durante la modalità emergenza. È composta da un bottone switch posizionato in alto a destra e da un ulteriore Fragment, la classe MappaFragment.
2. **MappaFragment:** classe che estende Fragment. È contenuta all’interno di FragmentEmergenza e si occupa esclusivamente della visualizzazione della mappa, il percorso che attraverso un canvas viene sovrapposto ad esso e la posizione dell’utente.
3. **Notifica:** estende Service, una classe dell’API che ci permette di utilizzare dei servizi in background. Tramite il Service e la classe DatagramSocket, facciamo rimanere in ascolto la porta 9601 (porta da noi unicamente utilizzata a questo fine) per ricevere eventuali messaggi di inizio o fine emergenza. Una volta ricevuto comunicazioni dal Server, l’ultimo compito della classe sarà quello di creare la notifica e di visualizzarla su schermo, inserendoci dunque il testo del messaggio precedentemente ricevuto.
4. **Percorso:** il funzionamento della classe percorso è stato ricavato dalla omonima classe del Server. Di conseguenza verrà illustrata nella progettazione lato Server.
5. **GrafoTronchi, Nodo:** come la precedente, le seguenti classi verranno spiegate nella sezione successiva.
6. **Uscita:** la classe, attraverso i suoi metodi, effettua un controllo se l’utente ha completato il percorso ed è uscito dall’edificio. Questa funzionalità è stata implementata andando a verificare se l’utente si è connesso all’ultimo beacon del suo percorso, e cioè il beacon che fa riferimento all’uscita.
7. **GestoreEntità:** oltre alle funzionalità descritte nelle user stories precedenti, abbiamo aggiunto il compito di richiamare la classe GestoreDati e di richiedere il percorso al server.
8. **GestoreDati:** alla classe sono stati aggiunti i metodi relativi alla richiesta del percorso e alla richiesta delle uscite dell’edificio. Come precedentemente descritto, il seguente metodo richiede le informazioni al database locale o al server a seconda se il dispositivo è connesso alla rete locale o no.

8.6.2 Progettazione lato Server

Come nel caso precedente, la progettazione delle classi lato Server è stata svolta per passi. Il primo è stato quello di sviluppare le classi ottenute dalla fase di analisi e prima ancora dall’analisi CRC. Dopodichè siamo andati a definire dei pattern da applicare alle nostre classi ed infine siamo andati a soddisfare i requisiti funzionali che venivano richiesti. Il risultato di questi passaggi verrà descritto qui di seguito:

1. **ConsoleDiComando:** implementa un Runnable ed è la classe che possiede il metodo testaEmergenza per avviare la modalità emergenza;
2. **Emergenza:** questa classe si occupa di interpellare il database e scoprire se ci sono Tronchi che presentano emergenze. Si fa uso di un metodo controllo() che avvia un thread effettuando delle verifica periodiche. In caso positivo istanza la classe NotificaServer.
3. **NotificaServer:** estende Thread ed all’interno del metodo run(), comunica l’emergenza a tutti gli IP connessi.
4. **Percorso, PercorsoConCosto:** la classe Percorso si occupa di calcolare un percorso verso l’uscita o verso una destinazione. Istantiando la classe percorso viene richiamato uno dei due metodi calcoloPercorso(GrafoTronchi.Nodo) o calcoloPercorso(GrafoTronchi.Nodo, GrafoTronchi.Nodo). L’algoritmo mantiene una lista dei nodi da attraversare in una frontiera. Tali nodi sono quelli adiacenti al nodo preso in considerazione. L’algoritmo, scorrendo come un BFS aggiunge ogni volta un nodo al suo nodo padre grazie alla classe PercorsoConCosto. In questo modo in frontiera si mantengono istanze di PercorsoConCosto con l’insieme dei nodi adiacenti da attraversare e il relativo costo di quel percorso. Ad ogni iterazione viene selezionato il percorso con costo minore tra i presenti. Non appena viene selezionato un percorso con l’ultimo nodo che corrisponde all’uscita, l’algoritmo termina e restituisce tale percorso calcolato.

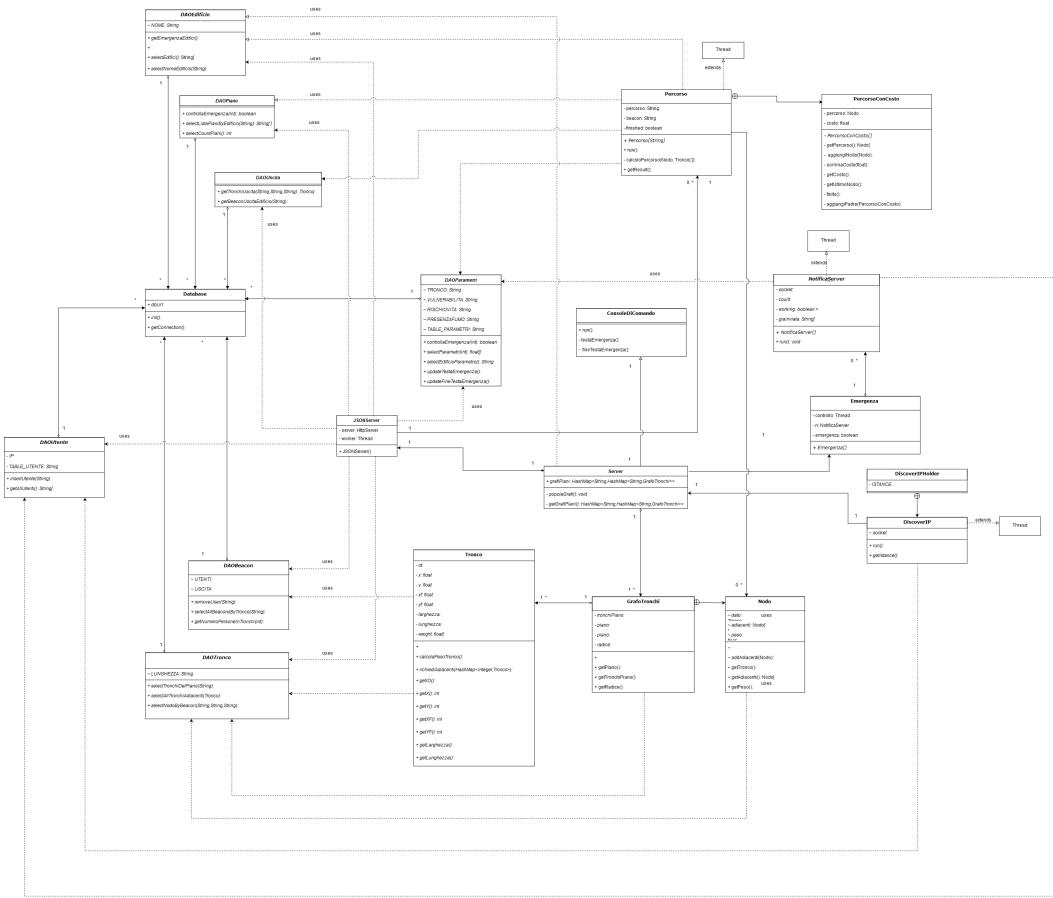


Figura 30: Diagramma delle classi di progettazione lato Server

5. **GrafoTronchi, Nodo:** la classe GrafoTronchi rappresenta il grafo costruibile rispetto alla pianta di un piano. Ogni nodo corrisponde ad un tronco, due nodi sono collegati se i tronchi sono adiacenti. La classe GrafoTronchi contiene informazioni sul nodo radice del grafo. La classe Nodo invece rappresenta un tronco nel grafo, contiene il tronco a cui fa riferimento e mantiene la lista dei tronchi adiacenti a questo.
6. **Tronco:** la classe Tronco è stata creata per modellare al meglio l'entità tronco, da poter utilizzare nelle classi adibite al calcolo del percorso.
7. **DiscoverIP:** la classe implementa un Runnable e ci permette di effettuare una connessione al Client senza aver bisogno di conoscere fin dal principio il suo indirizzo IP.
In sostanza, il metodo run() invia un messaggio di broadcast all'interno della rete LAN e su una porta nota ai Client. Nell'istante in cui il ricevente riconosce che il messaggio è a lui destinato, riinvia al Server un messaggio noto con cui si filtreranno le comunicazioni e si salva il suo indirizzo IP nella sua lista degli utenti.

8.7 Diagramma delle architetture

Riportiamo in figura 31 il diagramma delle architetture relativo alla storia 4. Si può notare che all'interno del seguente diagramma si vengono a sviluppare due rami di comunicazione tra Client e Server. La presenza del secondo ramo è stata introdotta per gestire i messaggi di emergenza. Dunque, a differenza dalle storie precedenti, in cui il client richiedeva informazioni al server, in questo caso abbiamo che è il Server ad inviare al Client informazioni relativi all'inizio o al rientro di un'emergenza. Riguardo invece all'architettura analoga alle storie precedenti (**Server->JsonServer**), in questo caso viene utilizzata per richiedere il percorso al Server e scaricarlo sul Client.

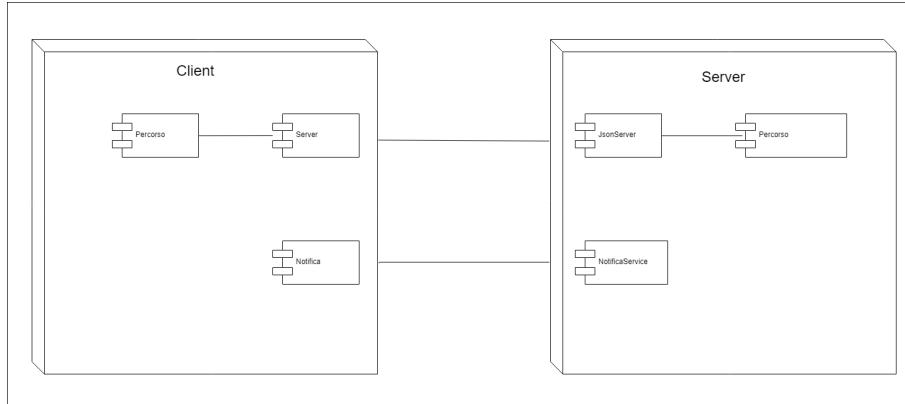


Figura 31: Diagramma delle architetture

8.8 Diagramma delle sequenze

In questa sezione andremo a mostrare, in figura 32, la sequenza del dialogo che intercorre tra il Server che avvia l'emergenza e il Client.

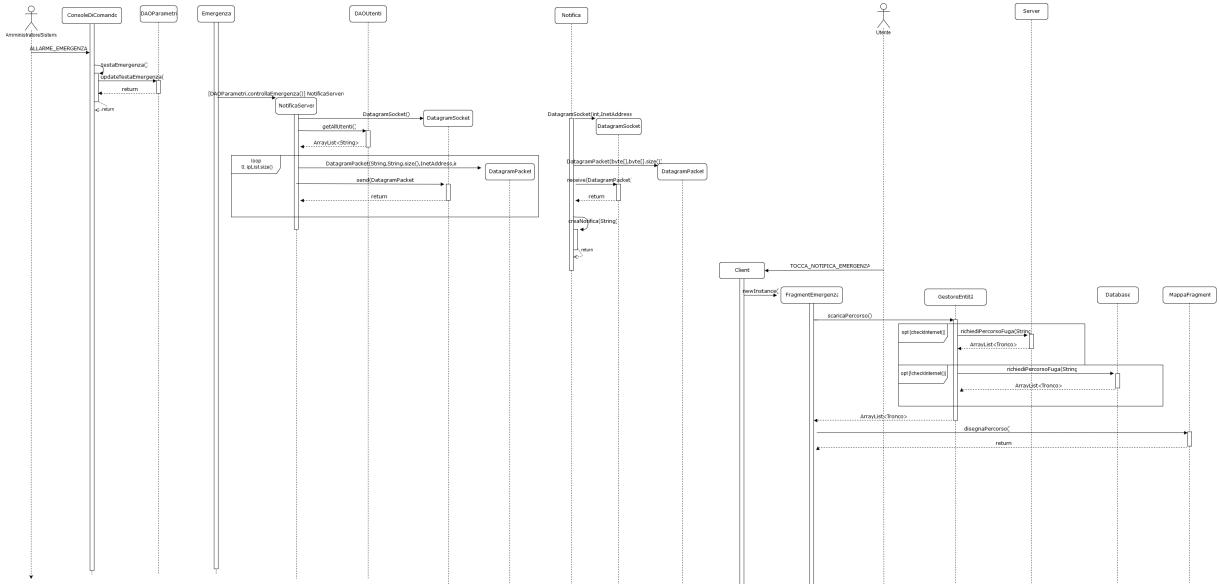


Figura 32: Diagramma delle sequenze

La sequenza di dialogo è composta dalle seguenti fasi:

- Un responsabile, attraverso la console di comando del Server, avvia l'allarme comunicando l'emergenza su un tronco associato all'edificio. In seguito il metodo **testaEmergenza()** della classe **ConsoleDiComando** richiede alla classe **DaoParametri** di aggiornare il dato **RV** (RischioVita) della tabella **Parametri** a 1.
- In questa seconda fase, il Thread **Emergenza**, effettuando periodicamente una verifica sulle informazioni della tabella parametri, controlla il parametro **RV**. Se rileva un'emergenza allora istanzia la classe **NotificaServer**. Quest'ultima ricava gli indirizzi IP di ogni utente connesso al servizio GetOut! e invia ad ognuno di essi una notifica di emergenza. Per avviare ogni singolo messaggio di emergenza vengono istanziate le classi **DatagramSocket** e **DatagramPacket**. La prima ha il compito di preparare il canale di comunicazione col Client alla porta 9601, e **DatagramPacket** per incapsulare il messaggio all'interno di una struttura predefinita per la tipologia di comunicazione.
- A lato client, il messaggio viene rilevato dal Service **Notifica** che rimane in ascolto ugualmente sulla porta 9601. Una volta ricevuto il messaggio, la classe preleva il messaggio dalla sua

struttura **DatagramPacket** e viene inserito in una notifica Push per poi essere visualizzato sul dispositivo.

4. Il seguente step inizia nel momento in cui l'utente preme sulla notifica, facendo così partire l'applicazione in modalità emergenza. All'utente verrà mostrato il **fragmentEmergenza** con internamente **MappaFragment**, ovvero la mappa dell'edificio, il percorso e la sua posizione.
5. Per riuscire a visualizzare il percorso relativo all'utente, il **FragmenteEmergenza** ne richiede lo scaricamento alla classe **GestoreEntità**. Se il dispositivo è connesso alla rete locale, allora si procederà attraverso una richiesta al Server, altrimenti domandando le informazioni al database locale. Una volta che **FragmenteEmergenza** ha ricevuto i dati relativi ai tronchi che compongono il percorso, comunica a **MappaFragment** di disegnare il percorso utilizzando le informazioni ricavate in precedenza.

8.9 Casi di Test

Si sono svolti i seguenti casi test, si noti che questa storia risulta strettamente collegata alla storia 2 soprattutto inerentemente al funzionamento.

- **Test Case 1:** Accettazione di richiesta avvio e fine emergenza
 1. Avviare il Server;
 2. Inviare da console di comando il messaggio di inizio emergenza;
 3. Attendere e verificare il messaggio di conferma;
 4. Ripetere lo stesso procedimento con il comando di rientro emergenza.
- **Test Case 2:** Calcolo percorso a livello server
 1. assicurarsi che venga calcolato il percorso più breve.
- **Test Case 3:** Visualizzazione della notifica
 1. avviare l'applicazione e inviare dal server il messaggio di inizio emergenza;
 2. controllare la corretta ricezione e visualizzazione della notifica push;
 3. verifica del corretto avvio dell'applicazione;
- **Test Case 4:** Scaricamento e visualizzazione del percorso verso uscita più vicina
 1. verifica avanzamento posizione utente sul tracciamento del percorso;
 2. verifica corretto ricalcolo percorso dopo la connessione ad un nuovo beacon;
 3. verifica esatto ricalcolo del percorso dopo aggiornamento delle condizioni edificio;
 4. ricalcolo percorso in modalità offline;
 5. ricalcolo percorso dopo cambio del piano;

9 User Story 1: Modalità Ordinaria

9.1 Storia utente



Priorità: 4

Implementazione delle funzionalità client che riguardano la gestione dell'applicazione in modalità ordinaria:

1. Scaricare dati dal server per il database offline.
2. Interfaccia per modalità ordinaria.
3. Bottone per cambiare modalità manualmente.
4. Calcolare percorso per una destinazione selezionata online.
5. Calcolare percorso per una destinazione selezionata offline.

Implementazione delle funzionalità server che riguardano la gestione dell'applicazione in modalità ordinaria:

1. Restituire dati al Client.
2. Restituire percorso per una destinazione selezionata.

Figura 33: Storia Utente n. 1

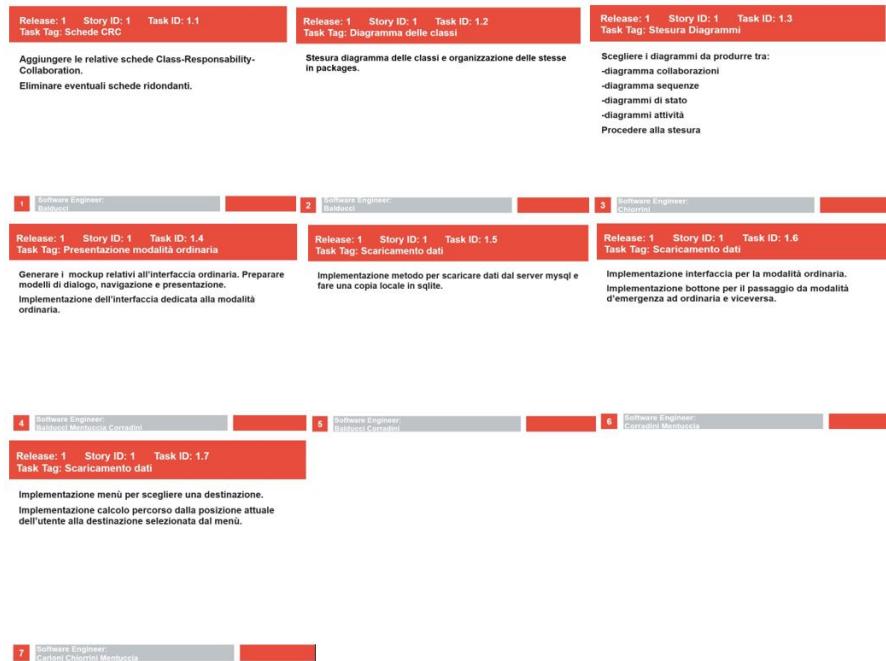


Figura 34: Risultato dell'iteration planning game Storia 1

Come si può notare dalla figura 33, la storia utente numero 1 consiste nella progettazione ed implementazione della modalità ordinaria. Consideriamo questa modalità da due prospettive differenti: lato Client e lato Server.

1. Client

Abbiamo implementato un'interfaccia grafica relativa alla modalità ordinaria. All'utente viene visualizzata la sua posizione attuale nella mappa e può scegliere una destinazione tramite una lista di aule oppure può attivare manualmente la modalità di emergenza.

All'avvio dell'applicazione, se siamo connessi ad internet, viene eseguita in automatico lo scaricamento delle informazioni tramite il server. Completato il download, possiamo disconnettere il telefono e continuare ad usare le funzionalità principali dell'applicazione.

2. Server

Il Server è in grado di rispondere alla richiesta del Client per il download di tutti i dati. Restituisce il percorso dalla posizione attuale del client ad un'aula da esso selezionata.

In figura 34 sono riportati i task ottenuti dalla storia utente.

9.2 CRC

Class: Client	Class: Server
Description: attore	Description: attore
Responsibility	Collaboration
Cambiare modalità	Modalità ordinaria
	Inviare dati per scaricamento
	Database
Class: Database	Class: Modalità Ordinaria
Description: non serve per l'interfaccia	Description: serve interfaccia
Responsibility	Collaboration
Inserire dati ricevuti da server	Server
	Visualizza il percorso
	Percorso
Class: Percorso	
Description: non serve per l'interfaccia	
Responsibility	Collaboration
Calcolare il percorso online	
Calcolare il percorso offline	Posizione

Figura 35: Schede CRC ottenute dall'analisi Storia 1

Successivamente abbiamo compiuto l'analisi nome-verbo nella storia utente ottenendo le schede de CRC presenti in figura 35. Tutte le classi ottenute hanno responsabilità e collaboratori. In particolare otteniamo:

- una classe screen, che andrà poi a definire l'interfaccia grafica.

Modalità Ordinaria: classe che presenta l'interfaccia in modalità ordinaria.

- altre classi che si occupano invece dell'inserimento dei dati nel database e del calcolo del percorso.

Database: classe che si occupa dell'inserimento dei dati nel database offline.

Percorso: classe che si occupa del calcolo del percorso verso una destinazione precisa sia offline che online.

Client: classe che modella il Client, implementa il passaggio in modalità ordinaria.

Server: classe che modella il Server, gestisce il download dei dati.

Dall'analisi CRC nessuna classe risulta eliminabile.

9.3 Diagramma delle classi di analisi

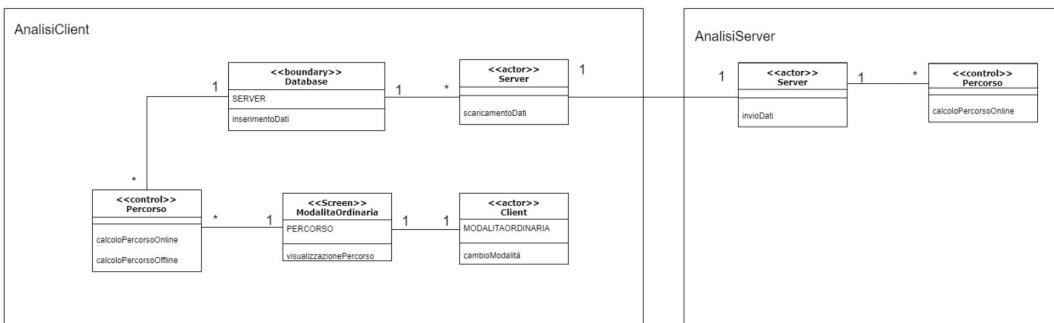


Figura 36: Diagramma delle classi di analisi Storia 1

Successivamente all'analisi nome-verbo e alla definizione delle schede CRC siamo passati alla stesura del diagramma delle classi di analisi. Il diagramma ottenuto è visibile in Figura 36. Nel

diagramma delle classi di analisi ritroviamo le classi definite dall'analisi CRC. Sono state identificate quattro categorie all'interno di questo diagramma: **screen** (ModalitaOrdinaria), **actor** (Server, Client, Server) **control** (Percorso,Percorso) e **boundary** (Database), quando troviamo due nomi uguali significa che una classe è relativa al diagramma di analisi del Client e l'altra per il diagramma di analisi del Server. Dalle responsabilità abbiamo ottenuto i metodi di ciascuna classe, dalle collaborazioni le varie associazioni.

9.4 Diagramma dei package

In figura 37, è stato riportato lo schema che rappresenta il modo in cui le classi sono state ordinate e raggruppate a formare i package. In questo diagramma è inoltre possibile individuare le dipendenze tra i vari package.

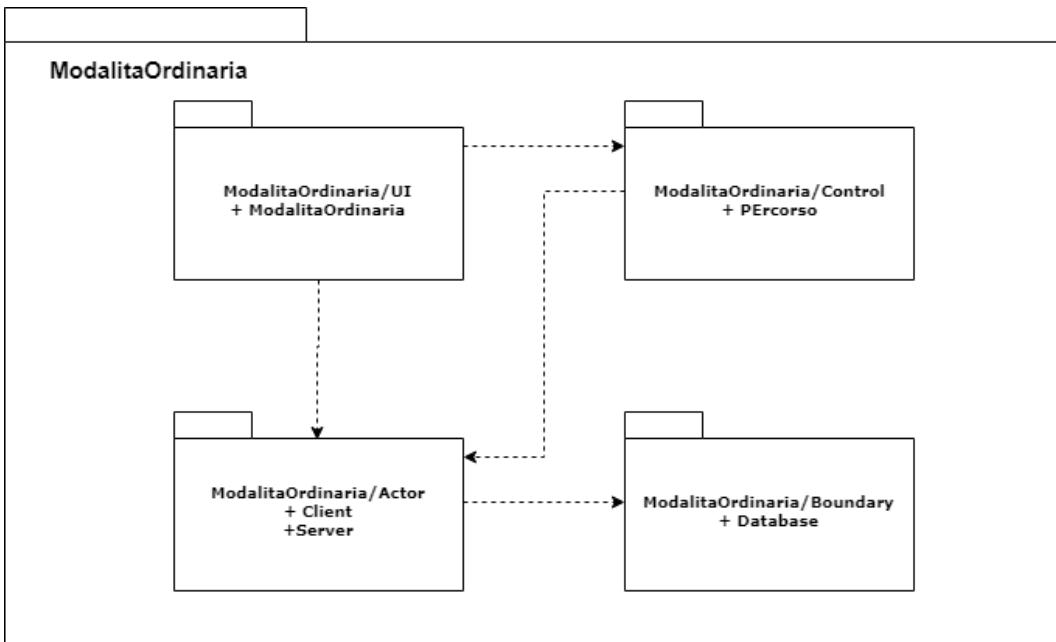


Figura 37: Diagramma dei package Storia 1

9.5 Diagramma delle classi di progettazione

Dopo aver applicato le varie regole, controllato le classi ed i metodi già esistenti, abbiamo pensato le classi di progettazioni, divisi in due schemi uno per il Client l'altro per il Server 38 39.

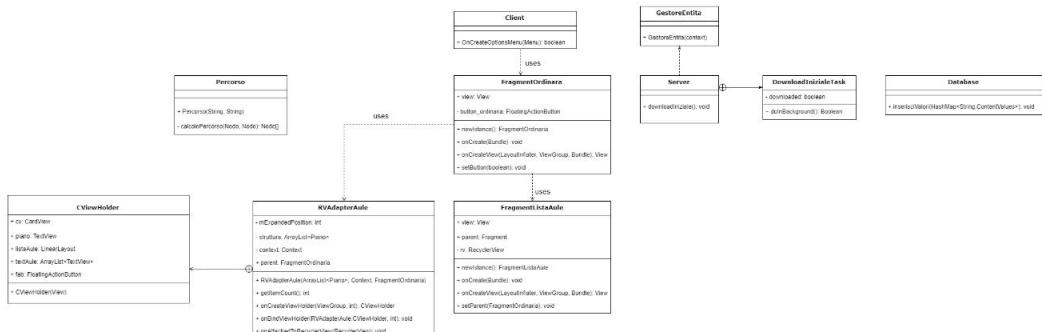


Figura 38: Diagramma delle classi di progettazione client Storia 1

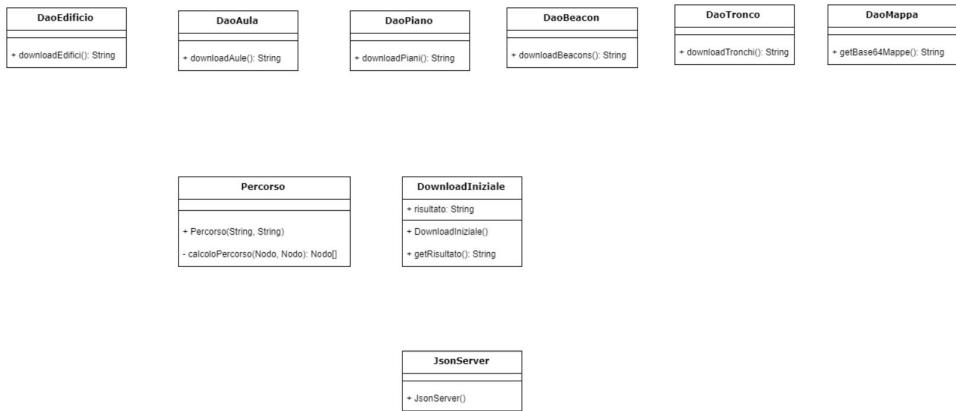


Figura 39: Diagramma delle classi di progettazione server Storia 1

- Per quanto riguarda la progettazione Client abbiamo deciso di trasformare le classi Server e Database, le quali si occupavano del download iniziale, in **GestoreEntita**, **Server**, **Database** e **DownloadIniziale**. Le prime tre erano già state implementate in storie precedenti, quindi sono stati aggiunti solo i metodi necessari. La classe **DownloadIniziale**, implementata in questa storia, elabora la stringa json arrivata dal server per farla diventare un hashmap. L'hashmap viene poi passato alla classe Database che inserirà i dati all'interno del file sqlite. Riguardo l'interfaccia la classe ModalitaOrdinaria è stata trasformata in **FragmentOrdinaria**. Questa classe si occupa della visualizzazione della mappa e della posizione attuale dell'utente
Il calcolo del percorso invece viene gestito dalle classi **FragmentListaAule**, **RVAdapterAule** e **Percorso**. Le prime due servono per far visualizzare un menu attraverso il quale il Client può visualizzare l'elenco delle aule raggiungibili. Alla classe Percorso, già esistente, è stato aggiunto un metodo simile al calcolo del percorso in modalità emergenza, con l'unica differenza che il punto di arrivo non sarà un'uscita ma un'aula selezionata dal Client.
- Per la parte di progettazione relativa al Server, abbiamo ripreso e modificato classi già implementate, l'unica classe nuova è **DownloadIniziale**. Le classi già implementate a cui sono state aggiunte un metodo, per la gestione del dump della tabella a loro associata, sono: **DAOEdificio**, **DAOAula**, **DAOPiano**, **DAOBeacon**, **DAOTronco** e **DAOmappa**. Il risultato del metodo aggiunto, per ogni Dao, è una stringa json contenente il risultato della query 'SELECT * FROM' per ogni signola tabella. La nuova classe **DownloadIniziale** raccoglie i singoli dump dei Dao così da costruire il json finale, il quale poi verrà inviato al Client. Alla classe **Percorso** sono stati aggiunti dei metodi per il calcolo del percorso dalla posizione dell'utente ad un'aula. Funzionamento identico alla classe Percorso lato Client spiegata prima. Le funzionalità lato Server vengono gestite tramite scambio di messaggi tra le classi.

9.6 Diagramma delle architetture

Nel diagramma delle architetture, figura 40, mostriamo come abbiamo diviso in package i nostri due sistemi, Client e Server.

1. Client

Abbiamo quattro packages: **gestioneconnessioni**, **gestioneposizione**, **fragment** e **utilità**. All'interno del primo troviamo le classi che gestiscono il download dei dati iniziali (GestoreEntita, Server, Database, DownloadIniziale). Nel secondo troviamo le classi che gestiscono il calcolo del percorso (Percorso). All'interno di fragments troviamo FragmentOrdinaria che gestisce la visualizzazione dell'interfaccia in modalità ordinaria e FragmentListaAule che gestisce l'elenco delle aule. All'interno dell'ultimo package troviamo la classe RVAdapterAule la quale gestisce la visualizzazione e le funzionalità dei bottoni una volta selezionata un'aula.

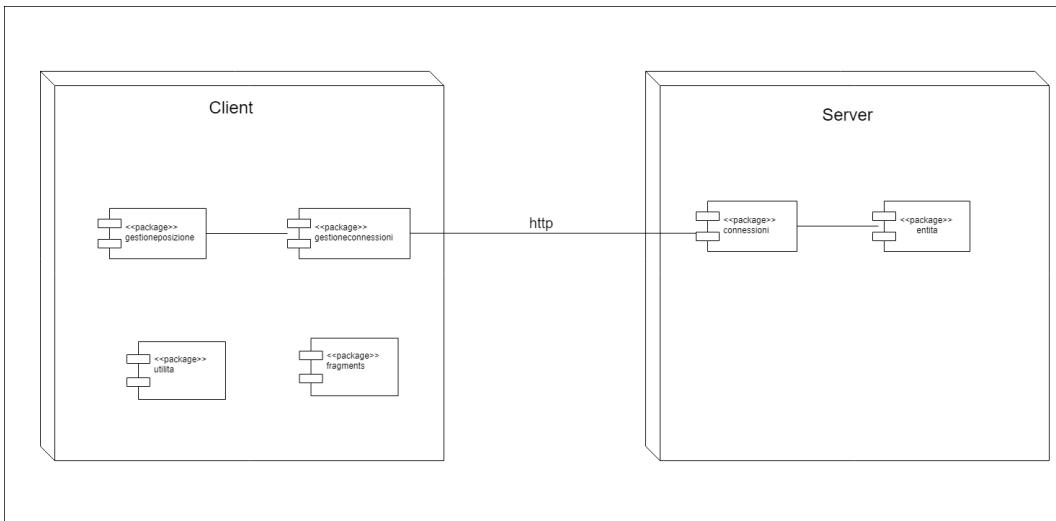


Figura 40: Diagramma delle architetture Storia 1

2. Server

Abbiamo due packages: **connessioni** e **entita**. Il primo contiene la classe `JsonServer` la quale gestisce la richiesta del Client e in seguito invia il json, ricevuto da `DownloadIniziale`, contenente tutti i dati online che il Client dovrà scaricare offline. All'interno di `connessioni` troviamo anche la classe `DownloadIniziale`. Il secondo package invece contiene i singoli Dao (`DAOEdificio`, `DAOPiano`, `DAOAule`, `DAOTronco`, `DAOBeacon`, `DAOMappa`).

9.7 Diagramma delle Sequenze

Nella figura 41 viene mostrato lo scambio dei messaggi per effettuare il download iniziale dei dati se il Client è connesso ad internet.

9.8 Casi di Test

Per la storia presa in esame sono stati definiti i seguenti casi di test:

- **Test Case 1:** Interfaccia modalità ordinaria
 1. Avviare l'applicazione.
 2. Controllare il funzionamento del pulsante per il cambio di modalità.
 3. Controllare il funzionamento del pulsante per aprire il menù contenente la lista delle aule.
- **Text Case 2:** Download iniziale
 1. Controllare la richiesta del download inviata dal Client al Server.
 2. Controllare la preparazione del json da parte del Server da inviare al Client
 3. Controllare l'invio della risposta da parte del Server.
 4. Controllare la ricezione del json da parte del Client.
 5. Controllare la trasformazione del json ricevuto in query sql.
 6. Verificare il corretto inserimento dei dati nel file sqlite.
- **Test Case 3:** Percorso
 1. Selezionare un'aula.
 2. Controllare che il punto di arrivo sia quello selezionato.
 3. Controllare che il percorso restituito sia il più veloce.

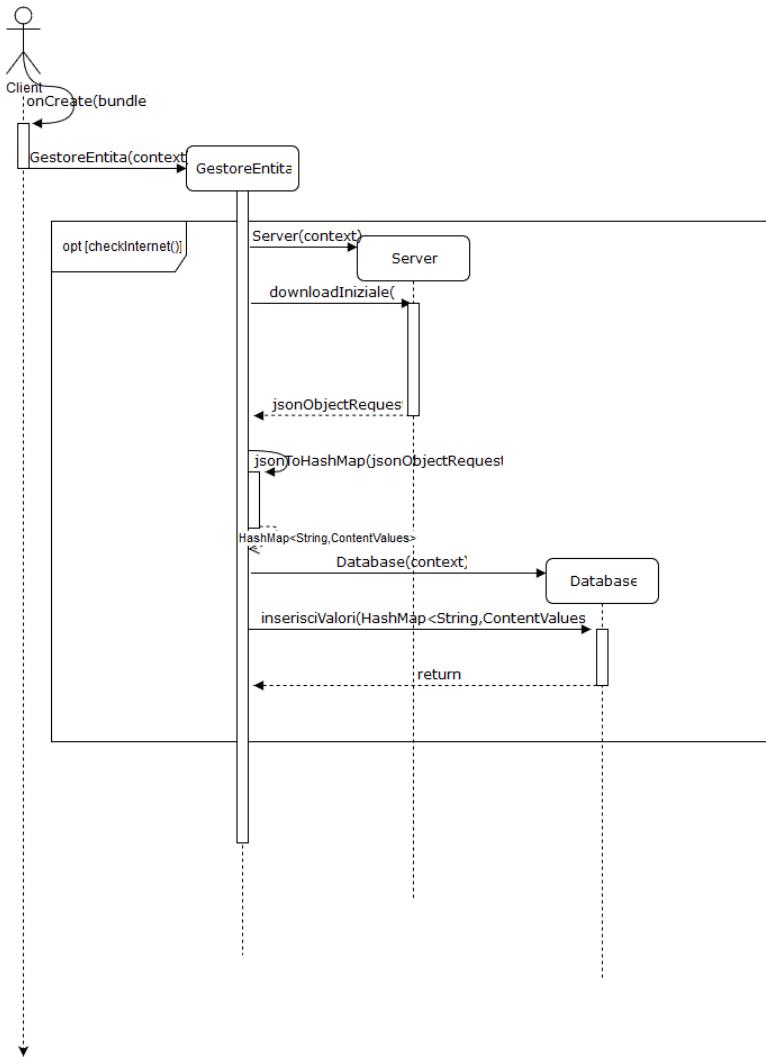


Figura 41: Diagramma delle sequenze Storia 1

10 User Story 5: Interfaccia amministrazione Server

10.1 Storia utente

Come si può notare dalla Figura 42, la storia utente numero 5 consiste nella progettazione ed implementazione di un’interfaccia grafica e di comandi utili all’inserimento e alla gestione dei dati del sistema.

Sostanzialmente, sono previste quattro diverse funzionalità:

1. un’interfaccia grafica che permette l’inserimento grafico dei dati da parte dell’amministratore di sistema, in particolare tutto ciò che riguarda l’edificio comprese le mappe;
2. una tabella per la visualizzazione e la modifica dei parametri attuale dei tronchi dell’edificio;
3. comando di importazione di file CSV per il popolamento rapido del database;
4. comando per la modifica dei pesi associati ai parametri dei tronchi, per il calcolo del costo del tronco.

10.2 Divisione in task

In Figura 43 sono riportati i task ottenuti dalla storia utente. Ad ogni task è associato il relativo Software Engineer che si è occupato del suo sviluppo

5. Interfaccia amministrazione Server

Priorità: 3

Implementare un'interfaccia grafica per l'inserimento dei dati nel database.

Visualizzazione di un form in cui selezionare le cose da aggiungere:

1. possibilità di aggiungere edifici, piani, tronchi e beacon al database con i relativi dati
2. possibilità di caricare l'immagine della mappa se non presente

Visualizzazione della tabella dei parametri dei tronchi e possibilità di modificarli.

Possibilità di importazione di dati tramite file csv.

Possibilità di modificare i pesi dei parametri dei tronchi.

Figura 42: Storia Utente n. 5

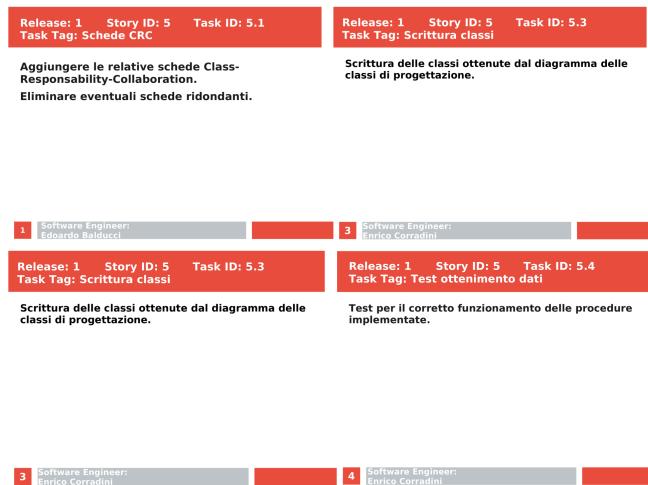


Figura 43: Risultato dell'iteration planning game Storia 5

10.3 CRC

Successivamente abbiamo compiuto l'analisi nome-verbo nella storia utente ottenendo le schede CRC presenti in Figura 44. Tutte le classi ottenute hanno responsabilità e collaboratori. In particolare otteniamo:

- tre classi screen, che andranno poi a definire l'interfaccia grafica.

Amministrazione: classe che presenta l'interfaccia di aggiunta dei dati per le varie entità che compongono l'edificio.

Tabella Parametri: classe che presenta l'interfaccia di visualizzazione e modifica dei parametri per ogni tronco.

Importa CSV: classe che si occupa dell'importazione del CSV.

- altre classi che si occupano invece dell'inserimento dei dati nel database.

Database: classe che si occupa della connessione al database.

Edificio: classe che modella un edificio, rappresenta la tabella del database.

Piano: classe che modella il piano dell'edificio, rappresenta la tabella del database.

Tronco: classe che modella il tronco di un piano, rappresenta la tabella del database.

Aula: classe che modella l'aula di un piano, rappresenta la tabella del database.

Beacon: classe che modella un beacon, rappresenta la tabella del database.

Class: Amministrazione Description: screen Responsability Collaboration Applungere nuovo edificio Applungere nuovo piano Applungere nuovo tronco Applungere nuova aula Applungere nuovo beacon	Class: Tabella Parametri Description: screen Responsability Collaboration Visualizzare parametri tronchi Modificare parametri tronchi	Class: Importa CSV Description: screen Responsability Collaboration Importazione del CSV
Class: Database Description: Responsability Collaboration Connettere al database	Class: Edificio Description: Responsability Collaboration Inserisci nuovo edificio nel database	Class: Piano Description: Responsability Collaboration Inserisci nuovo piano nel database
Class: Tronco Description: Responsability Collaboration Inserisci nuovo tronco nel database	Class: Aula Description: Responsability Collaboration Inserisci nuova aula nel database	Class: Beacon Description: Responsability Collaboration Inserire nuovo beacon nel database
	Class: Parametri Description: Responsability Collaboration Leggere parametri tronchi Modificare parametri tronchi	

Figura 44: Schede CRC ottenuti dall’analisi Storia 5

Parametri: classe che modella l’insieme dei parametri dei vari tronchi, rappresenta la tabella del database.

Dall’analisi CRC nessuna classe risulta eliminabile.

10.4 Diagramma delle classi di analisi

Successivamente all’analisi nome-verbo e alla definizione delle schede CRC siamo passati alla stesura del diagramma delle classi di analisi. Il diagramma ottenuto è visibile in Figura 45. Nel diagramma delle classi di analisi ritroviamo le classi definite dall’analisi CRC. Sono state identificate tre categorie all’interno di questo diagramma: **screen** (TabellaParametri, Amministrazione, ImportaCSV), **entity** (Edificio, Piano, Aula, Tronco, Beacon, Parametri) e **boundary** (Database). Dalle responsabilità abbiamo ottenuto i metodi di ciascuna classe, dalle collaborazioni le varie associazioni.

10.5 Diagramma dei package

Grazie alle tre categorie ottenute in precedenza, siamo riusciti a stilare il diagramma dei package della storia visibile in Figura 46. In questo diagramma è inoltre possibile individuare dipendenze tra vari package. In particolare:

1. il package riferito all’interfaccia grafica (UI) dipende dalle entity che rappresentano i dati nel database
2. il package entity invece dipende direttamente dal boundary, che è quello che contiene le classi necessarie alla lettura e scrittura dei dati nel database.

10.6 Diagramma delle classi di progettazione

Dopo aver applicato le varie regole, controllato le classi ed i metodi già esistenti, abbiamo pensato le classi di progettazioni, come descritto in Figura 47. In primo luogo, le classi entity sono state trasformate nelle classi DAO già implementate durante lo sviluppo della Storia 3. A queste sono stati aggiunti solamente i metodi necessari in questa Storia, ovvero quelli di inserimento dati. Per i restanti passaggi fare riferimento al diagramma della Storia 3.

Le classi screen sono state progettate come **GUIAmministrazione**, che al suo interno contiene la definizione della classe ImagePanel, utile solamente a questa classe, **GUIParametri** e **ImportaCSV**. Le istanze di queste classi sono tutte contenute nella classe **ConsoleDiComando**, vista in precedenza, dalla quale è possibile, tramite opportuni comandi digitati nella console del server, avviare le funzionalità descritte in questa storia. Analizziamo in dettaglio la funzione di tali classi. La classe **GUIAmministrazione** si occupa della creazione di un’interfaccia grafica per l’inserimento dei dati nel database. Oltre ad alcuni metodi funzionali alla creazione dei vari elementi grafici in JAVA, si definiscono metodi per l’inserimento dei dati: aggiungiEdificio(), aggiungiPiano(), aggiungiAula(), aggiungiTronco(), aggiungiBeacon().

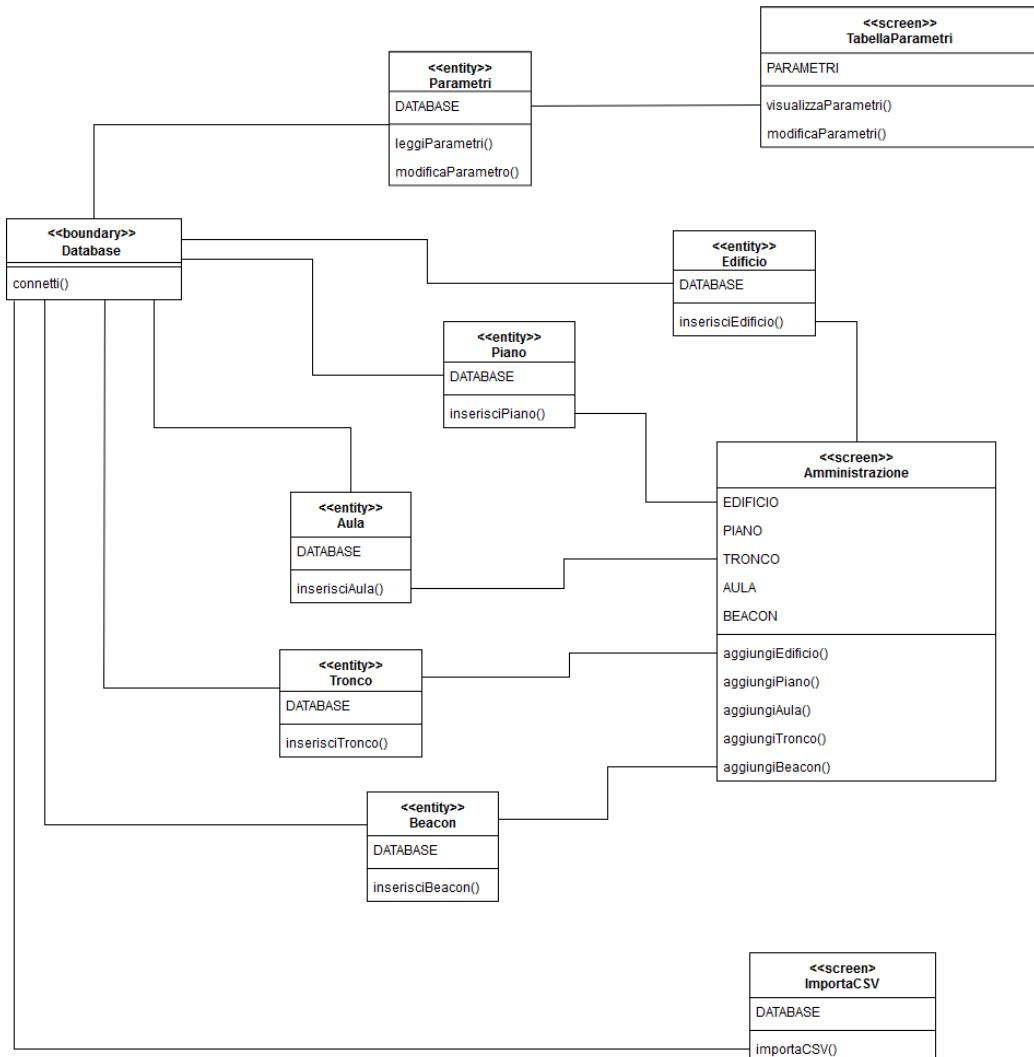


Figura 45: Diagramma delle classi di analisi Storia 5

La classe **GUIParametri** si occupa della creazione di una tabella per la visualizzazione dei parametri attuali di tutti i tronchi salvati nel database. Il metodo aggiungiTabellaParametri(), oltre che a visualizzare i dati ottenuti dal database, definisce anche le procedure di modifica di tali parametri e il loro successivo aggiornamento nel database.

La classe **ImportaCSV** è composta dal solo costruttore. In base al tipo di CSV da importare, la classe andrà ad inserire i vari dati inseriti nel file nella relativa tabella. La distinzione della tabella avviene in base al tipo di comando chiamato nella console, identificato dal parametro String in ingresso al costruttore.

In tutte le classi DAO sono stati aggiunti i metodi necessari alla modifica dei valori nel database.

10.7 Diagramma delle architetture

Questo diagramma non è stato previsto per la storia presa in esame.

10.8 Diagramma delle sequenze

Questo diagramma non è stato previsto per la storia presa in esame.

10.9 Casi di Test

Per la storia presa in esame sono stati definiti i seguenti casi di test.

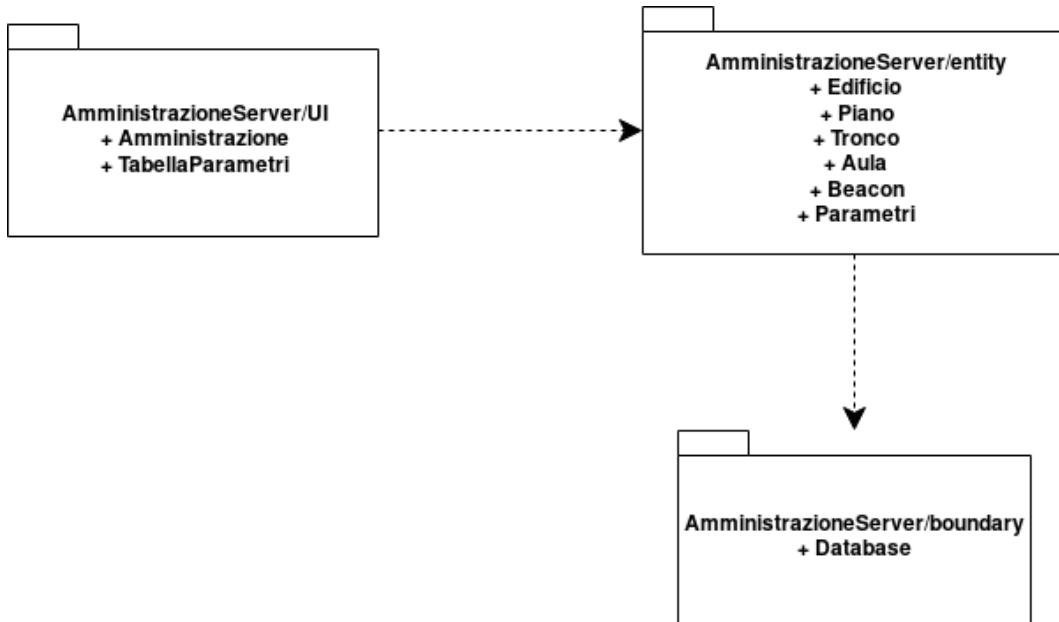


Figura 46: Diagramma dei package Storia 5

- **Test Case 1:** Inserimento dati di prova

1. Avviare l'interfaccia grafica di amministrazione tramite il comando 'amm'.
2. Inserire un nuovo edificio.
3. Inserire un piano per l'edificio.
4. Inserire un tronco per il piano.
5. Inserire un'aula per il piano.
6. Inserire un beacon per il tronco.
7. Controllare che l'inserimento sia avvenuto con successo.

- **Test Case 2:** Caricamento e visualizzazione mappa

1. Selezionare un edificio e un piano precedentemente inseriti.
2. Cliccare sul bottone per il caricamento della mappa e scegliere l'immagine della pianta del piano selezionato.
3. Terminare il caricamento della mappa e controllare il corretto salvataggio dell'immagine e la successiva visualizzazione a schermo.

- **Test Case 3:** Visualizzazione e modifica dei parametri dei tronchi

1. Avviare l'interfaccia di gestione dei parametri tramite il comando 'par'.
2. Controllare la corretta visualizzazione della tabella.
3. Effettuare un doppio click su uno dei parametri di un tronco e modificarlo.
4. Controllare l'avvenuta modifica del parametro.

- **Test Case 4:** Modifica dei pesi dei parametri

1. Lanciare il comando: 'pesi 0.5 0.5 0.5 0.5 0.5' e controllare la modifica dei pesi.

- **Test Case 5:** Importazione CSV

1. Creare cinque file CSV per ciascuna tabella da importare (Edificio, Piano, Tronco, Aula, Beacon).
2. Lanciare il comando 'csv edificio' e importare il file CSV appropriato.

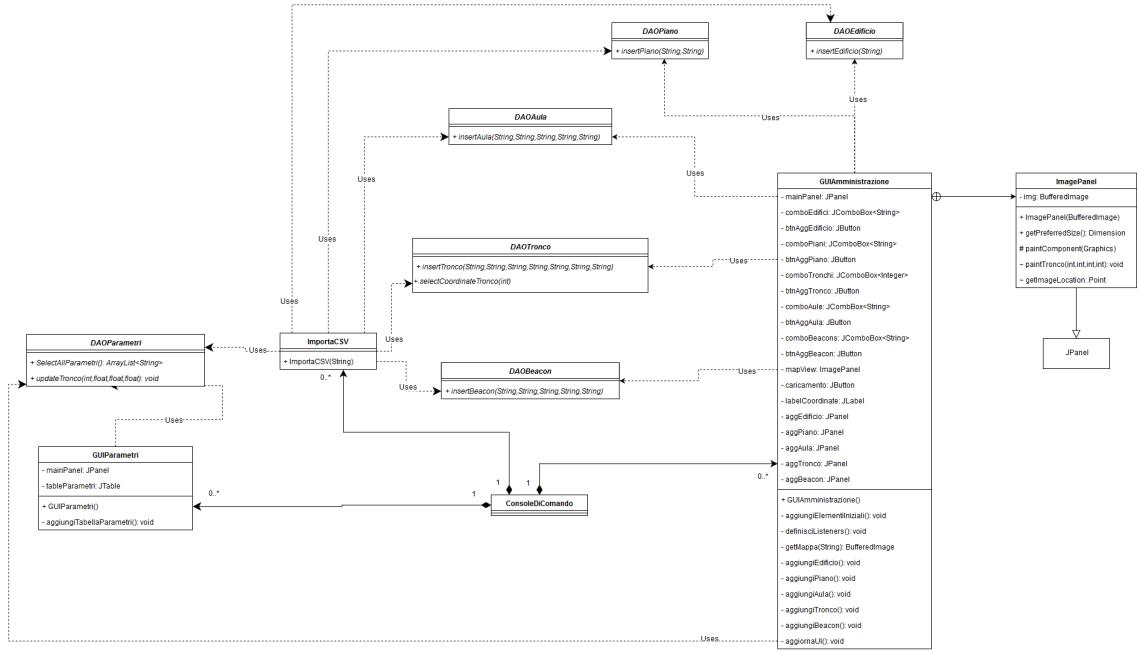
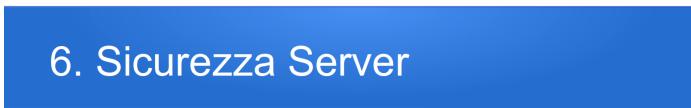


Figura 47: Diagramma delle classi di progettazione Storia 5

3. Lanciare il comando 'csv piano' e importare il file CSV appropriato.
4. Lanciare il comando 'csv tronco' e importare il file CSV appropriato.
5. Lanciare il comando 'csv aula' e importare il file CSV appropriato.
6. Lanciare il comando 'csv beacon' e importare il file CSV appropriato.
7. Controllare l'effettiva importazione dei dati.

11 User Story 6: Sicurezza Server

11.1 Storia utente



Priorità: 4

Implementare un sistema di sicurezza che permetta la comunicazione con il server solamente a chi scarica e l'utilizza l'applicazione.

Figura 48: Storia Utente n. 6

Come si può notare dalla Figura 48, nella storia 6 ci si è occupato della sicurezza informatica del server. La storia è nata dopo aver notato la possibilità di compiere richieste al server da qualsiasi device connesso alla rete LAN dell'edificio.

Attraverso un'analisi della rete è possibile infatti risalire all'indirizzo IP del server e a questo punto inondarlo di richieste per creare un attacco di tipo Denial of Service.

Si è deciso quindi di implementare un protocollo di sicurezza in modo tale che le richieste possano essere fatte solamente dall'applicazione. In questo modo si è cercato di isolare il server in modo fisico dalla rete internet e mantere possibili solamente le connessioni necessarie nella rete LAN del complesso gestito dal sistema.

11.2 Divisione in task

In Figura 49 sono riportati i task ottenuti da questa storia utente, che risulta più snella rispetto alle precedenti.

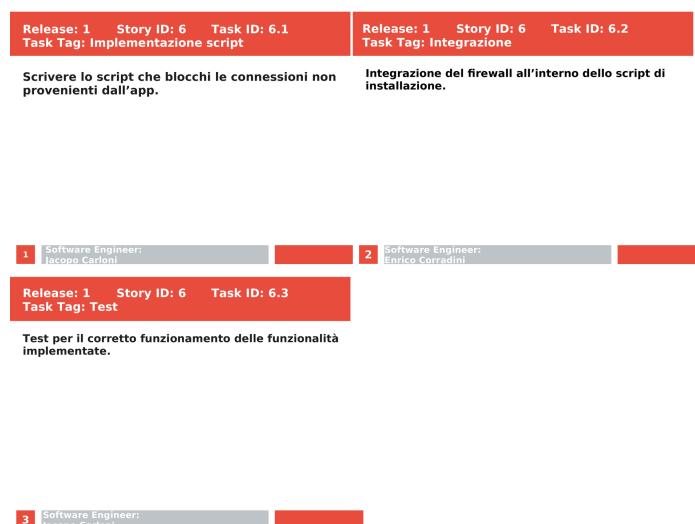


Figura 49: Risultato dell'iteration planning game Storia 6

11.3 IPTABLES

Per configurare la restrizione degli accessi al server, la scelta è ricaduta su iptables, il firewall, filtro di pacchetti, utilizzato nei sistemi Linux. Il ragionamento di fondo adottato consiste nel fatto che le richieste vengono fatte da client con determinati indirizzi IP all'interno della rete del complesso. Come visto nelle storie precedenti, ogni volta che un utente si collega per la prima volta utilizzando l'app, il suo indirizzo IP viene registrato all'interno della lista dei client. L'idea di base è quindi quella di bloccare tutte le connessioni che non provengano da questi indirizzi.

Innanzitutto, tramite uno script di configurazione, sono state definite le seguenti regole:

```
iptables -policy INPUT DROP  
iptables -policy OUTPUT DROP  
iptables -policy FORWARD DROP
```

In questo modo è stata bloccata qualsiasi connessione in input, output o forward. Tuttavia è necessario mantenere libera la porta 9605 che si occupa dell'invio del broadcast dell'indirizzo IP nella rete. Questo è stato possibile con i seguenti comandi, successivi ai precedenti:

```
iptables -A INPUT -p udp -dport 9605 -j ACCEPT  
iptables -A OUTPUT -p udp -dport 9605 -j ACCEPT
```

Inoltre, è necessario mantenere aperte le connessioni in localhost, visto che il server necessita di connettersi al database presente nella stessa macchina. Questo è possibile attraverso i seguenti comandi:

```
iptables -A INPUT -p tcp -s 127.0.0.1 -j ACCEPT  
iptables -A OUTPUT -p tcp -s 127.0.0.1 -j ACCEPT  
iptables -A FORWARD -p tcp -s 127.0.0.1 -j ACCEPT
```

Questi comandi sono stati inseriti in uno script di configurazione del firewall.

A questo punto ogni volta che un nuovo client si connette è necessario accettare connessioni da e verso il suo indirizzo IP. Per fare questo, innanzitutto apriamo la porta dove il client comunica il suo indirizzo IP al server tramite il seguente comando:

```
iptables -A OUTPUT -p udp -dport PORTA_TEMPORANEA -j ACCEPT
```

Tale connessione poi non sarà più necessaria. Non appena il server invia il pacchetto di risposta eliminaremo tale regola:

```
iptables -D OUTPUT -p udp -dport PORTA_TEMPORANEA -j ACCEPT
```

Infine, quando viene registrato l'indirizzo IP del client nella lista degli utenti, viene anche lanciato il seguente comando:

```
iptables -A INPUT -s IP_CLIENT -j ACCEPT  
iptables -A OUTPUT -s IP_CLIENT -j ACCEPT
```

In questo modo qualsiasi connessione al di fuori dei client autorizzati viene respinta.

11.4 Ulteriori cause di DoS

Sebbene iptables blocchi tutti gli indirizzi IP che non sono stati registrati dall'applicazione, la soluzione non è sufficiente per impedire ulteriori attacchi.

Infatti una volta registrato l'indirizzo IP questo ha completo accesso sia in input che in output: l'attaccante potrebbe quindi utilizzare l'indirizzo IP registrato per perpetrare attacchi DoS.

Sebbene non sia stato implementato in questa release, una prima soluzione potrebbe essere quella di creare un hash per l'applicazione che possa fungere da firma digitale. La richiesta a questo punto viene accettata dal server solamente se proviene dall'applicazione con la giusta firma, evitando richieste da indirizzi IP registrati ma non provenienti dall'applicazione. Una seconda soluzione invece consiste nel prevedere una possibile autenticazione dell'utente tramite l'app, in modo da essere sempre sicuri che le richieste non provengano da fonti terze.

11.5 Casi di Test

Per la storia presa in esame sono stati definiti i seguenti casi di test.

- **Test Case 1:** Configurazione del firewall
 1. Lanciare da shell lo script di configurazione del firewall.
 2. Controllare che la macchina non sia più in grado di inviare o ricevere connessioni.

3. Avviare il server 'GetOut!'.
4. Avviare da un client l'applicazione 'GetOut!'.
5. Controllare che la connessione avvenga senza problemi.
6. Ripetere dal punto 4 una seconda volta.
7. Controllare che il tutto avvenga senza problemi.