

RELAZIONE DI PROGETTO DI "LABORATORIO DI  
SISTEMI SOFTWARE"

---

# Intelligent Backpack

---

*Componenti del gruppo:*

- Brighi Andrea
- Di Lillo Daniele
- Lazzari Matteo

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Processo di Analisi</b>	<b>5</b>
2.1	Interviste . . . . .	5
2.2	Dalle idee alla schematizzazione . . . . .	9
2.3	Use Cases del sistema . . . . .	10
2.4	Storytelling in collaborazione con gli esperti . . . . .	12
2.5	Sottodomini . . . . .	13
2.6	Ubiquitous Language . . . . .	13
<b>3</b>	<b>Requisiti</b>	<b>16</b>
<b>4</b>	<b>Tactical Design</b>	<b>18</b>
4.1	Applicazione smartphone . . . . .	19
4.2	Raspberry . . . . .	26
4.3	Sistema di backend . . . . .	31
<b>5</b>	<b>Plugin Gradle</b>	<b>36</b>
5.1	gradle-python-testing . . . . .	36
5.1.1	Configurazioni disponibili . . . . .	38
5.2	Gradle-Git-Sensitive-Semantic-Versioning-Plugin-for-Android . . . . .	39
5.2.1	Configurazioni disponibili . . . . .	40
<b>6</b>	<b>Sviluppo e pratiche DevOps</b>	<b>41</b>
6.1	DVCS e workflow . . . . .	41
6.2	Controlli pre-commit . . . . .	42
6.3	Utility utilizzate . . . . .	43
6.4	Build automation . . . . .	43
6.4.1	Firmware Python per Raspberry . . . . .	44
6.5	Submodules . . . . .	45
6.6	Testing . . . . .	45
6.6.1	Microservizi . . . . .	45
6.7	Continuous integration . . . . .	46
6.8	Versioning delle release . . . . .	50

6.8.1	Microservizi . . . . .	50
6.9	Documentazione . . . . .	51
6.9.1	Applicazione smartphone . . . . .	51
6.9.2	Raspberry Pi 4 . . . . .	52
6.9.3	Microservizi . . . . .	52
6.10	Licenze . . . . .	52
<b>7</b>	<b>Implementazione</b>	<b>54</b>
7.1	Dispositivo e Device Twin . . . . .	55
7.2	Database . . . . .	56
7.3	Microservizi . . . . .	56
7.4	Python + Gradle . . . . .	57
7.5	Applicazione . . . . .	57
<b>8</b>	<b>Timeline di lavoro</b>	<b>59</b>
<b>9</b>	<b>Conclusioni</b>	<b>60</b>
9.1	Sviluppi futuri . . . . .	60

# 1 Introduzione

Lo scopo finale del progetto è lo sviluppo di un sistema per la gestione di uno zaino smart, ovvero intelligente, e la sua interazione con il calendario scolastico. Questo in modo da fornire agli studenti uno strumento per poter gestire gli oggetti e/o libri che devono essere inseriti nello zaino per le lezioni del giorno successivo e sapere sempre cosa si è inserito all'interno. Questo prodotto è considerabile come dispositivo Wearable IoT, in quanto consiste in un oggetto indossabile con intelligenza, potenza di calcolo e connettività.

Il problema alla base del progetto è quello di ottimizzare la gestione del materiale scolastico da portare, in quanto ci possono essere errori di comunicazione tra professore/studenti, oltre alla possibilità che uno studente potrebbe dimenticarsi del materiale a casa.

Il sistema sviluppato sarà composto da 3 principali componenti per l'invio dei dati:

- *Servizi in cloud*: Tutta la parte che riguarda il server e il database sarà gestita in cloud, tramite l'utilizzo della piattaforma Microsoft Azure Portal;
- *Applicazione smartphone*: L'applicazione per dispositivi mobili offre all'utente un'interfaccia che permette di ottenere tutte le informazioni (a richiesta) del calendario, delle successive lezioni e degli oggetti/libri che sono stati inseriti all'interno dello zaino;
- *Raspberry PI 4*: Componente hardware general purpose del progetto che viene utilizzato per la lettura del codice RFID di un libro/oggetto inserito nello zaino dall'utente, aggiornare in tempo reale il database remoto con gli oggetti inseriti/rimossi e rimanere in ascolto di eventi inviati da Hub IoT di Azure Portal.

Per quanto riguarda la parte dei servizi gestiti in cloud, si è deciso di optare per lo sviluppo di un'architettura basata su microservizi, in quanto offre una miglior scalabilità dei singoli servizi, se necessario (miglior gestione del carico di lavoro).

Viene usato un database relazionale, in particolare PostgreSQL, usando un server SQL hostato su Azure visto che è perfetto per la gestione di tutte le informazioni che sono collegate tra di loro.

Per quanto riguarda l'ambito tecnologico dello zaino, viene usato un Raspberry PI 4. Questo è stato collegato ad un lettore RFID, che viene utilizzato per la lettura

del tag del libro e un modulo WiFi, che permette la comunicazione del componente hardware con il database remoto per informarlo sull'aggiunta/rimozione di libri (la comunicazione verrà poi mostrata successivamente nella sezione dedicata alla Progettazione e Implementazione).

## 2 Processo di Analisi

Seguendo l'approccio di sviluppo Domain Driven, si è partiti dalla fase fondamentale di analisi del dominio, dettagliando e snocciolando la domain knowledge per ridurre il rischio di incomprensioni, lack of knowledge e requisiti errati. Siamo partiti dall'analisi dell'ambiente scolastico, semplice nella sua generalità dal punto di vista dello studente ma non banale nei meccanismi, dal punto di vista del professore.

### Richiesta committente

*Salve, vorremmo commissionare un progetto che riguarda la creazione di un sistema integrato allo zaino, con l'obiettivo di poter aiutare studenti e professori nella gestione del materiale scolastico da portare a lezione. In particolare l'idea è quella di adattare un qualsiasi zaino con un dispositivo che possa leggere il materiale che viene inserito, come comunemente vengono letti alla cassa i prodotti acquistati al supermercato, affinché lo zaino intelligente possa sapere cosa è stato inserito e, tramite servizi web e applicazione smartphone sviluppata ad hoc, possa ricordare all'utente quale materiale manca per le lezioni del giorno successivo. Tramite l'applicazione mobile un professore deve poter gestire le lezioni delle proprie classi mentre uno studente deve poter visualizzare il materiale da portare alle proprie lezioni. Una cosa essenziale è il fatto di poter visualizzare realtime il contenuto dello zaino dall'applicazione ed esser notificato in qualche modo nel caso di qualche libro o oggetto mancante.*

### 2.1 Interviste

Per analizzare ed esplorare i vari aspetti sconosciuti del dominio, sono state utilizzate diverse riunioni con l'esperto di dominio per effettuare vere e proprie interviste, con domande mirate a colmare i buchi di conoscenza (lack of knowledge), a volte anche piuttosto precise.

#### Ambiente scolastico

1. Ciao! Raccontaci un po' dell'ambiente scolastico, chi organizza le lezioni?

Le lezioni vengono di norma gestite dalla segreteria, che ha la responsabilità di gestire l'orario delle lezioni durante l'anno. Per effettuare ciò, si avvale di diverse azioni. Vorremmo includere in questa attività anche i professori, in modo che siano in grado di spostare lezioni o modificarne in totale autonomia, in modo veloce e facile e cosicché gli studenti possano saperlo in tempi brevi.

**2. Perfetto, quali meccanismi prevedono la gestione delle lezioni?**

Un professore per definire una lezione deve specificare la materia di interesse e il giorno della settimana in cui si svolge, ripetuta nelle settimane del ciclo scolastico annuale, l'orario d'inizio, quello di fine e la classe che dovrà frequentarla.

**3. Chiarissimo, per quanto riguarda invece modifiche al calendario, quali sono ammesse e come dovrebbero esser gestite?**

Diciamo che in generale le operazioni che si vogliono effettuare sono le classiche, ovvero aggiungere o rimuovere una o più lezione, che sia solo di un giorno o di un interavallo. Inoltre è possibile che le lezioni vengano spostate da un determinato giorno e orario ad una nuova posizione nel calendario, anche in questo caso può essere per un solo giorno o per più gioni (periodo).

**4. Interessante, stessa cosa vale anche per il materiale necessario in una determinata lezione?**

Assolutamente, si vuole lasciare piena libertà lato professore, esso deve poter cambiare il materiale disponibile per una data lezione in caso di errore di inserimento o cambi di programma.

**5. Quindi, se ho capito bene, c'è differenza tra giorno settimanale e data di una lezione**

Un calendario settimanale mostra i giorni della settimana e gli orari in cui si svolgono lezioni in quei giorni, ripetute quindi durante l'anno ogni settimana. Un giorno è definito come l'etichetta del giorno settimanale in questione e un orario, invece una data rappresenta un giorno specifico dell'anno ben formattato seguendo le tipiche convenzioni di rappresentazione. Una lezione è riferita ad un giorno settimanale, quindi è ripetuta, ecco perché è presente il periodo di riferimento con inizio e fine.

6. **Quindi, quando si chiede le lezioni in giorno bisogna sia considerare le lezioni che sono presenti in quell'giorno della settimana che quelle in quella data, giusto?**

Esattamente, bisogna anche tenere conto dei cambiamenti apportati all'orario. Come detto prima le lezioni possono essere aggiunte, cancellate o spostate.

7. **Perfetto, passiamo ad un aspetto un po' più interessante per gli studenti: come deve esser gestito il materiale da portare a lezione?**

Uno studente, accedendo al proprio profilo, vedrà le lezioni dell'anno scolastico o corso, insieme al materiale assegnato da portare ripetutamente per quella lezione, ovvero ad esempio nelle lezioni di mercoledì alle ore 9:00. Nel caso generale si tratta di libri e quindi gli studenti dovranno portare la propria copia del libro.

8. **Quindi, dato un titolo di libro, o magari meglio ancora un codice univoco come l'ISBN lo studente deve portare la propria copia?**

Esatto, ovviamente la copia è personale per ogni studente.

9. **È possibile identificare una copia di un libro di uno studente?**

Alcuni studenti mettono il nome all'interno, ma in genere le copie sono identiche tra di loro.

10. *Considerazioni*

Dall'intervista precedente risulta evidente che sia necessario identificare in qualche modo le copie dei libri degli studenti. Per questo motivo si decide di usare la tecnologia RFID mettendo un tag univoco all'interno della copia del libro in modo da identificarla dalle altre.

## Tipi di utenti

1. **Cambiando tematica, quali sono i tipi di utenti che vorreste differenziare?**

Sono due i tipi di utenti finali del servizio che vorremmo offrire, ovvero lo studente e il professore, entrambi potranno accedere alla piattaforma tramite una semplice applicazione mobile.

**2. Immagino abbiano ruoli e quindi funzionalità diverse all'interno del sistema**

Esattamente, uno studente, essendo iscritto in un istituto e in una classe, potrà vedere il suo calendario di lezioni programmato e il materiale necessario da portare per una data lezione. Un professore, al contrario, avrà la possibilità di gestire i calendari delle proprie classi e aggiornare il materiale necessario per una data lezione.

**3. Avranno anche funzionalità in comune...**

Si, è esatto. Sia studente che professore dovrebbero avere uno zaino intelligente, quindi entrambi i tipi di utenti possono gestire il proprio zaino dall'applicazione mobile, inserire oggetti e vedere il contenuto realtime, nonché esser notificato in qualche modo in caso manchi qualche materiale per la lezione del giorno dopo.

**Peculiarità del sistema che si vuole realizzare**

**1. Ci spieghi la sua idea di zaino che vorrebbe**

Uno zaino intelligente lo immagino proprio come uno classico, sarebbe l'ideale avere tecnologia presente ma non visibile e non fastidiosa per lo studente (o professore) che ne fa utilizzo.

**2. Quali interazioni si aspetta tra lo zaino e lo smartphone?**

Qualsiasi operazione naturale venga fatta con uno zaino classico, ovvero aggiungere e rimuovere oggetti o libri, in modo più naturale possibile, anche se un minimo di abitudine nell'inserirli con cautela è necessario, per ovvie ragioni di lettura del materiale inserito. Infine l'applicazione mobile dovrebbe mostrare in tempo reale, o quasi, il contenuto dello zaino e fornire tutte le funzionalità già discusse.

**3. Quindi, se ho capito bene, il cloud ha un aspetto fondamentale. L'applicazione mobile invece? Quali funzionalità pensa che dovrebbe avere?**

Oltre a quelle elencate nella richiesta e nelle precedenti domande, una bella comodità sarebbe rappresentata dal poter registrare uno zaino per un de-

terminato studente sfruttando l'applicazione, senza interazioni dirette con lo zaino, come ad esempio la scannerizzazione di un codice, veloce e intuitivo.

## 2.2 Dalle idee alla schematizzazione

Per avere un'iniziale chiara visione del dominio su cui il sistema opererà, si è scelto di cominciare nelle prime riunioni a fare mente locale con un piccolo brainstorming collettivo, trascrivendo su una mappa concettuale tutte le idee che sorgevano.

In particolare, dalla mappa si riesce già a delineare quali saranno le attività principali che rappresenteranno le principali milestone del progetto, quali sono le attività più corpose (interne, riassumibili nel completamento di una milestone) e quelle più basilari (esterne alla rete, rappresentano funzionalità base) e quali sono i requisiti che vincolano le varie attività. Inoltre si raccolgono i concetti analizzati dalle interviste con i committenti e si realizza una mappa di Event Storming in modo da poter raggruppare e temporizzare le azioni e i concetti. ([link mappa](#))

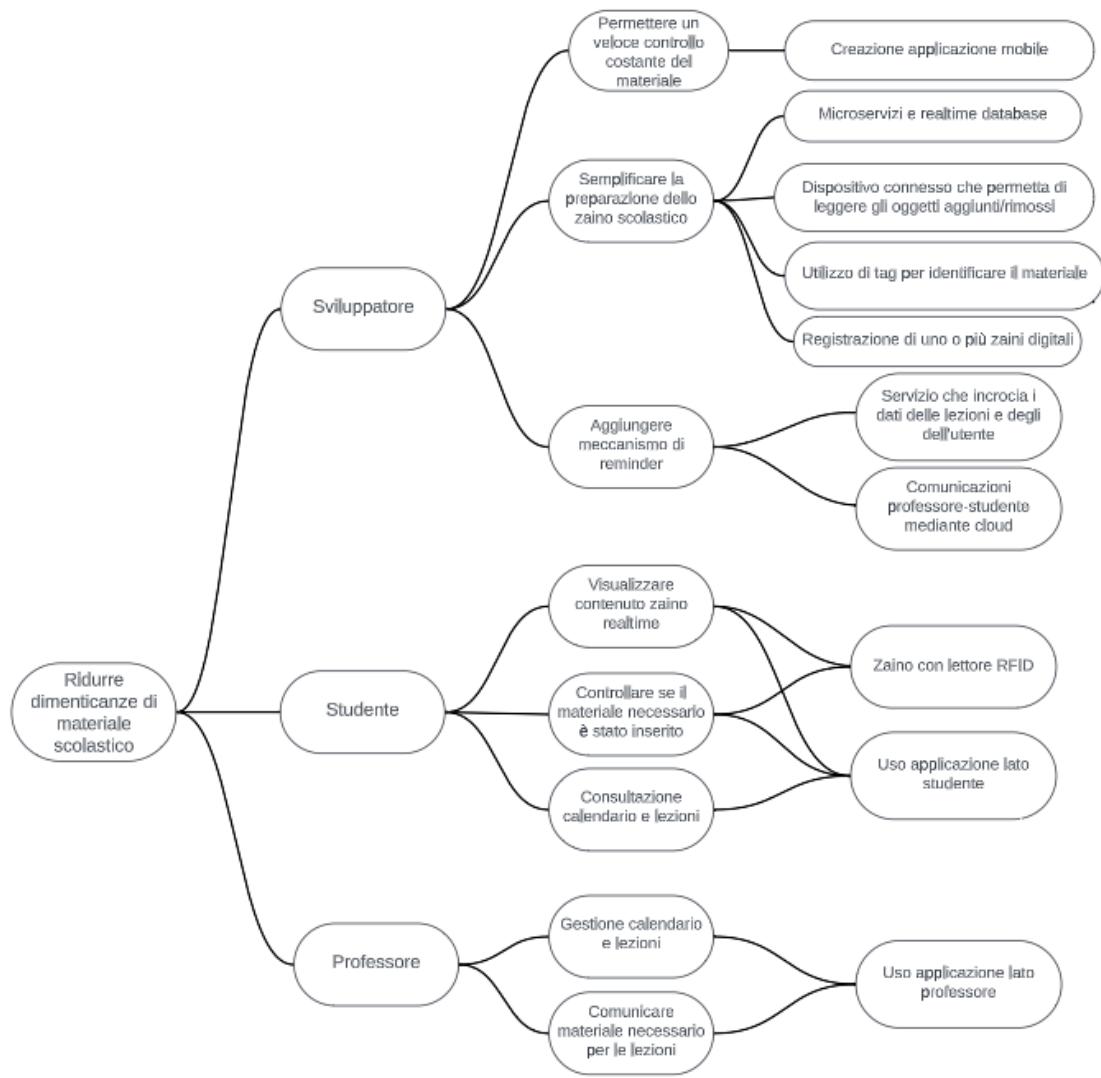


Figura 1: Impact map derivante dalla richiesta e ulteriori interviste

### 2.3 Use Cases del sistema

Come punto di partenza per l’analisi del dominio, si è deciso di partire dai casi d’uso del sistema che si vuole progettare, per capire al meglio quali sono le operazioni che gli utenti vedono e vogliono eseguire e quali attori sono coinvolti. L’importanza del loro utilizzo sta nell’identificare (a caratteri generali) quali attori sono coinvolti nelle varie operazioni e quali sono le principali azioni che devono essere sviluppate e/o sulle quali avere maggior riguardo. Per realizzare i casi d’uso c’è stata una

forte collaborazione del team con i committenti, sfruttando svariate riunioni, che si sono poi ripetute abitualmente. Nel seguito verranno presentati una sottoporzione dei casi d'uso principali, mentre l'analisi completa è presente sulla piattaforma di supporto GitHub (link) con la documentazione completa e più precisa.

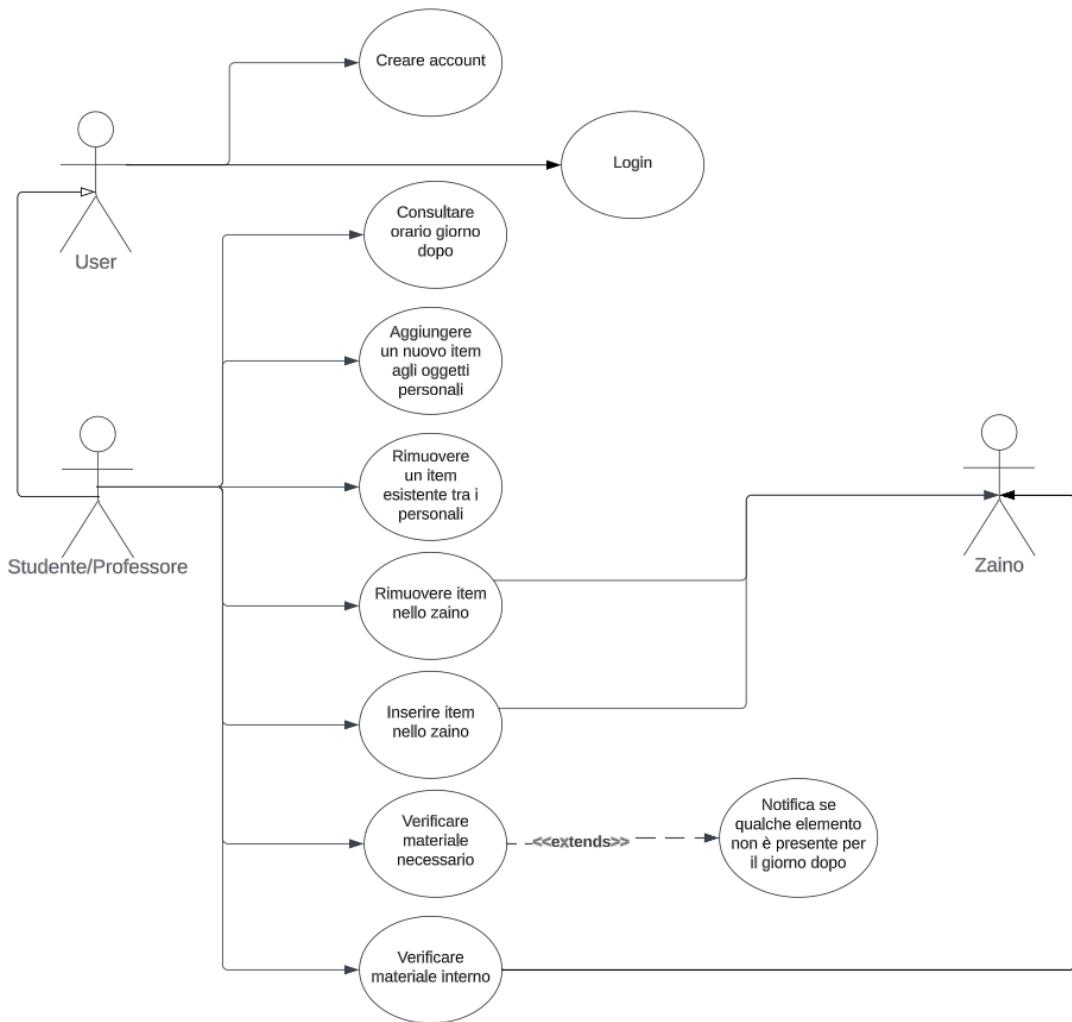


Figura 2: Use case per professore e studente

Come possiamo notare, studente e professore hanno molte operazioni in comune, o più precisamente, lo studente possiede un sottoinsieme di funzionalità concesse al professore, ovvero la gestione degli elementi all'interno dello zaino, esclu-

dendo quindi tutti i casi d'uso relativi alla gestione delle lezioni presenti sulla documentazione completa, perciò si rimanda ad essa per approfondimenti.

## 2.4 Storytelling in collaborazione con gli esperti

Una delle prime analisi effettuate è stata la raccolta delle User Stories, che sono uno strumento utilizzato nello sviluppo per descrivere le esigenze degli utenti in modo semplice, rappresentando una descrizione di quello che gli utenti vogliono ottenere dall'utilizzo del sistema, il tutto in una sessione di riunione in cui si discutono gli aspetti fondamentali del sistema, in linguaggio informale ma preciso.

Sfruttando una serie di riunioni con gli esperti di dominio che ne faranno utilizzo, abbiamo quindi avviato numerose discussioni che hanno portato alla stesura delle User Stories, che descrivono ad alto livello delle azioni eseguite sul sistema, in modo informale, schematico e seguendo una logica di storytelling. Innanzitutto ci siamo focalizzati sulla questione della creazione di un utente, quali azioni devono essere intraprese e quali entità partecipano.

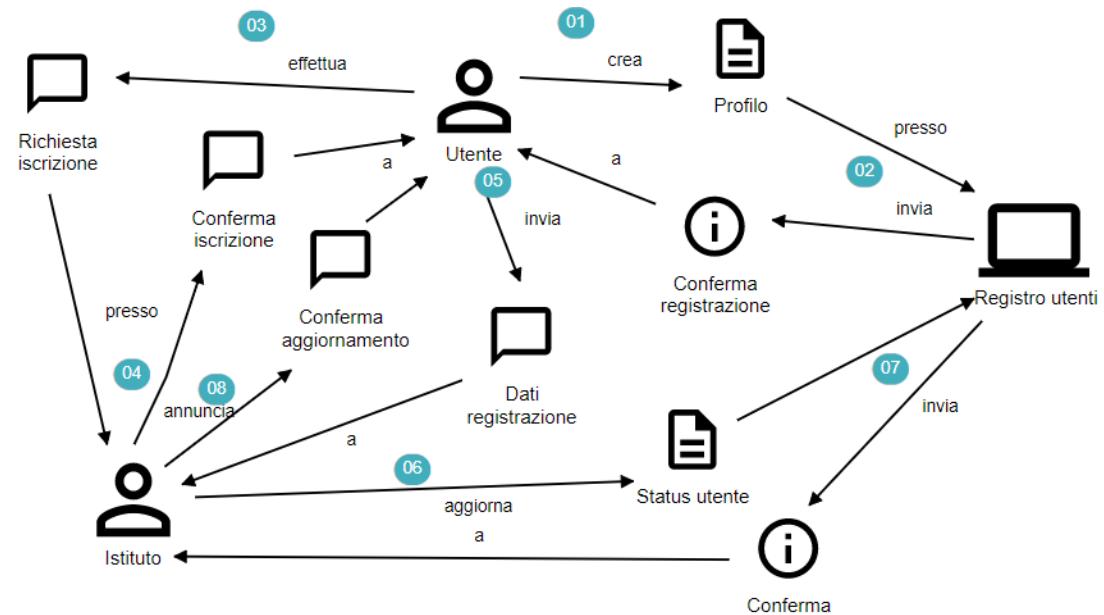


Figura 3: Creazione di un utente

Anche per le User Story si rimanda alla documentazione completa, con tutte le discussioni e user story realizzate.

## 2.5 Sottodomini

I sottodomini individuati, a seguito di una approfondita analisi, risultano i seguenti, con etichetta che ne determina la tipologia nell'obiettivo di business che si vuole raggiungere:

- Gestione libri GENERIC
- Gestione utenti GENERIC
- Gestione calendario SUPPORTING
- Reminder Engine CORE
- Zaino intelligente CORE
- Creazione etichette (TAG) GENERIC
- Fabbricazione zaini GENERIC
- Gestione materiale per lezioni CORE
- Applicazione mobile CORE

Questi sottodomini rispecchiano l'analisi preliminare svolta durante riunioni di Event Storming, per i cui dettagli si rimanda alla documentazione di dettaglio al seguente [link](#).

## 2.6 Ubiquitous Language

Durante la fase iniziale di analisi del dominio, si è definito l'ubiquitous language da usare durante lo sviluppo. Ciò è dato dal fatto che il committente (esperto del dominio) ha un linguaggio e terminologia diversa rispetto a quella degli sviluppatori, quindi è importante definire la differenza tra i vari concetti che potrebbero essere interpretati in un modo non opportuno.

L'ubiquitous language serve per eliminare la confusione e incomprensioni durante la discussione, dovuta ad una diversa terminologia, andando perciò a creare un linguaggio condiviso, che entrambe le parti possono capire. Ciò è possibile andando a definire una serie di vocaboli specifici nel dominio del problema, con una loro

descrizione per massimizzare la comprensione. Infine, questi vocaboli dovranno poi essere utilizzati durante la fase di sviluppo/documentazione.

Nella tabella sotto sono riportati un sottoinsieme dei termini identificati che, anche se sinonimi nel significato, nel dominio del problema rappresentano concetti diversi.

Tabella 1: Ubiquitous language

<b>Termine del dominio</b>	<b>Significato del termine riportato</b>
Giorno	Giorno della settimana.
Data	Rappresenta giorno specifico della settimana.
User	Rappresenta un utente che ha un ruolo generico, non ancora assegnato.
Studente	Uno user che frequenta un istituto in una classe.
Professore	Uno user che insegna ad un istituto e in una o più materie.
Classe	È composta da più studenti, di un determinato anno.
Materia	È un insieme di lezioni svolte da un singolo professore.
Lezione	Insegnata da un professore, ad una classe, in un giorno.
Copia	Copia fisica del libro.
Libro	Informazioni su un preciso libro pubblicato.
Orario di un giorno	Insieme di tutte le lezioni che sono presenti in un giorno.
Oggetto	Materiale scolastico di vario tipo che non comprende i libri.
Materiale	Ci si riferisce ad un insieme di oggetti e copie.
Istituto	User con privilegi speciali che permette di modificare i ruoli e gestire le lezioni.
Zaino	zaino intelligente dello user.
Orario di data	Insieme di tutte le lezioni che sono presenti in una precisa data.

Calendario	Insieme di orari di data.
Calendario scolastico	Insieme di orari di giorni.
Ruolo	Ruolo assunto dall'utente all'interno del sistema, come professore o studente.
Evento	Un qualsiasi evento assegnabile in un orario di data o di giorno. Una lezione è un evento particolare.

### 3 Requisiti

Durante la fase iniziale, sono stati definiti anche quali sono i requisiti che il progetto dovrà soddisfare.

- **Requisiti di Business:** Specificano quali sono le caratteristiche che il sistema dovrà avere per poter essere considerato corretto. Tra questi sono definiti:
  - Il sistema deve essere compatto per far sì che sia adattabile a quanti più zaini possibili e non modifichi il meccanismo umano dell'utente nell'inserire l'oggetto nello zaino;
  - Il sistema non deve andare a modificare le meccaniche di base dell'inserimento dei libri all'interno dello zaino;
  - Il sistema deve avere un basso consumo di energia, massimizzando però l'efficacia dei sensori all'interno di esso;
  - Verificare il corretto inserimento/rimozione dei libri dallo zaino.
- **Requisiti utente:** Specificano quali sono i bisogni dell'utente e descrivono quali azioni l'utente può effettuare quando interagisce col sistema. Tra questi sono definiti:
  - Visualizzare il calendario scolastico della propria classe, con le relative lezioni e i libri/oggetti da portare;
  - Registrare e rimuovere nuovi libri nel sistema;
  - Inserire libri/oggetti all'interno dello zaino;
  - Essere notificato nel caso non venga inserito un oggetto/libro nello zaino per la lezione del giorno successivo.
- **Requisiti funzionali:** Specificano quali sono le funzionalità che il sistema deve mettere a disposizione all'utente. Questi sono ricavati dai requisiti utente identificati precedentemente e dall'analisi delle user stories.

## **Applicazione smartphone**

- Accesso alla propria area;
- Registrazione/rimozione di un libro dalla propria libreria digitale;
- Visualizzazione del calendario scolastico;
- Visualizzazione degli oggetti/libri da portare per una determinata lezione;
- Modifica degli orari delle lezioni e dei libri/oggetti da portare.

## **Raspberry**

- Lettura degli RFID delle copie.
- **Requisiti non funzionali:** Specificano quali vincoli sono imposti al sistema.
    - L'applicazione mobile deve essere in grado di funzionare su diversi dispositivi Android, con differenti caratteristiche hardware;
    - L'interfaccia dell'applicazione deve essere il più semplice e reattiva possibile;
    - Il sistema deve essere in grado di scalare in base alle necessità del carico di lavoro;
    - Il lettore deve essere in grado di ottenere gli RFID con facilità.

## 4 Tactical Design

Come stato definito nella sezione dei requisiti, i componenti del sistema che si vuole realizzare sono tre: Applicazione mobile, sistema di lettura dell'RFID, sistema di backend. Infatti, questi tre sistemi andranno a fornire l'implementazione per i vari bounded context identificati.

Per la realizzazione del sistema di backend, si è deciso di adottare un'architettura basata su microservizi, identificandone 3 come principali:

- **Access:** Si occupa dell'accesso degli utenti al sistema, andando anche a verificarne i privilegi per l'accesso a determinate aree del sistema che sono riservate a solo determinati ruoli;
- **Book:** Si occupa della gestione della gestione delle librerie dei vari utenti, permettendo la registrazione/rimozione di eventuali libri;
- **Calendar:** Si occupa della gestione di tutti gli aspetti che riguardano le lezioni e il calendario scolastico, oltre a fornire la possibilità di modificare orari e materiale per le specifiche lezioni.

Il sistema a microservizi necessita che i vari servizi necessitino della comunicazione con altri servizi, in quanto ogni microservizio gestisce solo il proprio scope.

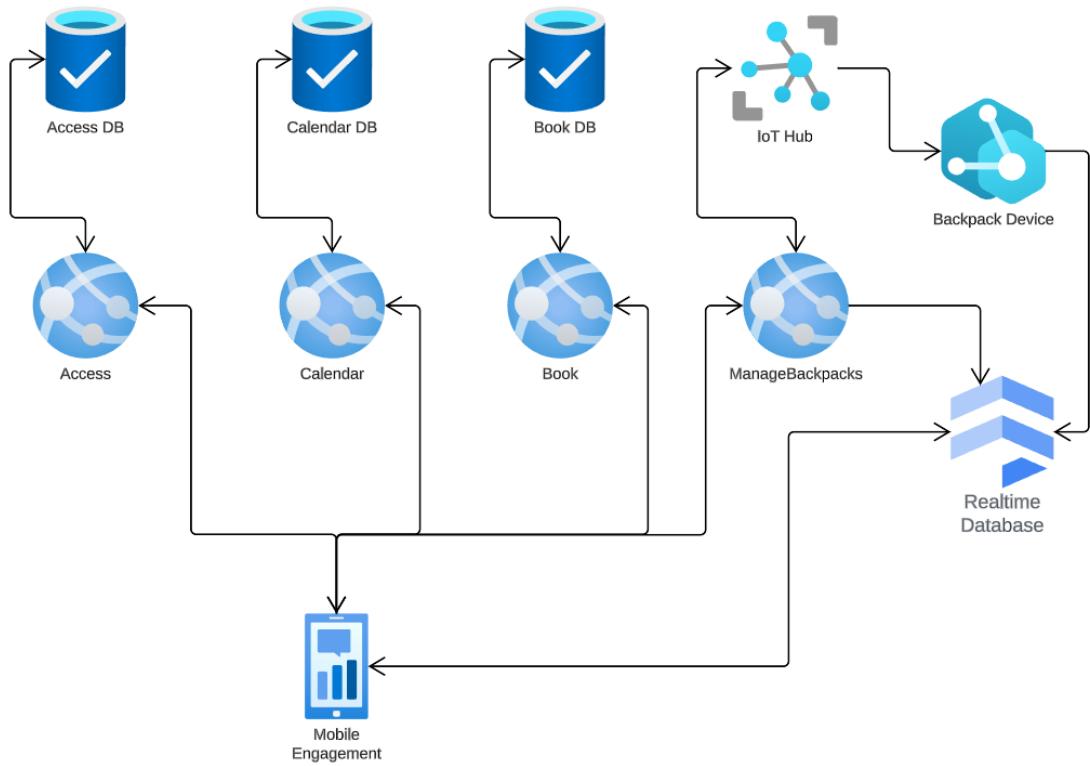


Figura 4: Servizi del backend.

Come si può visualizzare dall’immagine, i vari componenti del sistema si interfacciano direttamente con le API dei microservizi presenti in cloud, che accederanno poi al rispettivo database. In base alle informazioni che sono state raccolte durante la fase di analisi, si è passati alla fasi di progettazione, andando a definire la struttura delle diverse componenti.

#### 4.1 Applicazione smartphone

L’architettura dell’applicazione mobile si basa sulla Clean Architecture, rispettano le buone prassi per la progettazione Android (pagina android sulla struttura architettonica). In particolare l’app si divide in 4 parti per il dominio (uno per ogni bounded context del problema):

- access: per gestire l’accesso
- desktop: per gestire i libri e il materiale scolastico

- school: per gestire la scuola e il suo calendario
- reminder: comprende intersezione tra school e desktop in modo da sapere quali libri portare a lezione

L'applicazione è organizzata a livelli in modo da dividere la logica, per comunicare tra livelli vengono realizzati degli adapter, che convertono una classe in un'altra e viceversa. In caso di interazione con altri bounded context vengono usati adapter (se necessario l'uso come elemento di dominio).

### **Domain Logic o Domain Layer**

Questo rappresenta il nucleo, la logica di dominio, il Domain Model, tutte le altre parti dell'applicazione convertono gli elementi presenti in questo livello in quella usato nella loro astrazione (DB, rete, UI). In questo livello vengono create le policy e l'api del modello del dominio.

### **Application Layer**

Questo livello, rappresentato da una classe normalmente, permette di mostrare a codice le operazioni possibili nel bounded context, gestisce gli errori e l'interazione con gli altri bounded context. Inoltre, tramite il repository, che incapsula la logica, permette di gestire la permanenza locale e la gestione remota dei dati.

### **Interface Adapters**

Questo livello, presente nei moduli esterni al domino, converte gli elementi di domino delle corrispettive rappresentazione usato per mostrare i dati all'utente (UI), salvarli nel database o inviarli in rete.

### **Infrastructure Layer**

Questo livello contiene i dettagli tecnici dell'applicazione android e quindi dipende dal framework usato, dal db e dai formati dei DTO. In particolare la parte che gestisce la grafica utilizza il pattern Model-View-ViewModel (rivisitazione del pattern MVC) e viene usato un service locator che crea i repository e li fornisce come dipendenze all'application layer. La sezione di dati viene divisa in locale, che è un'astrazione del DB, e remote, astrazione delle comunicazioni con i servizi.

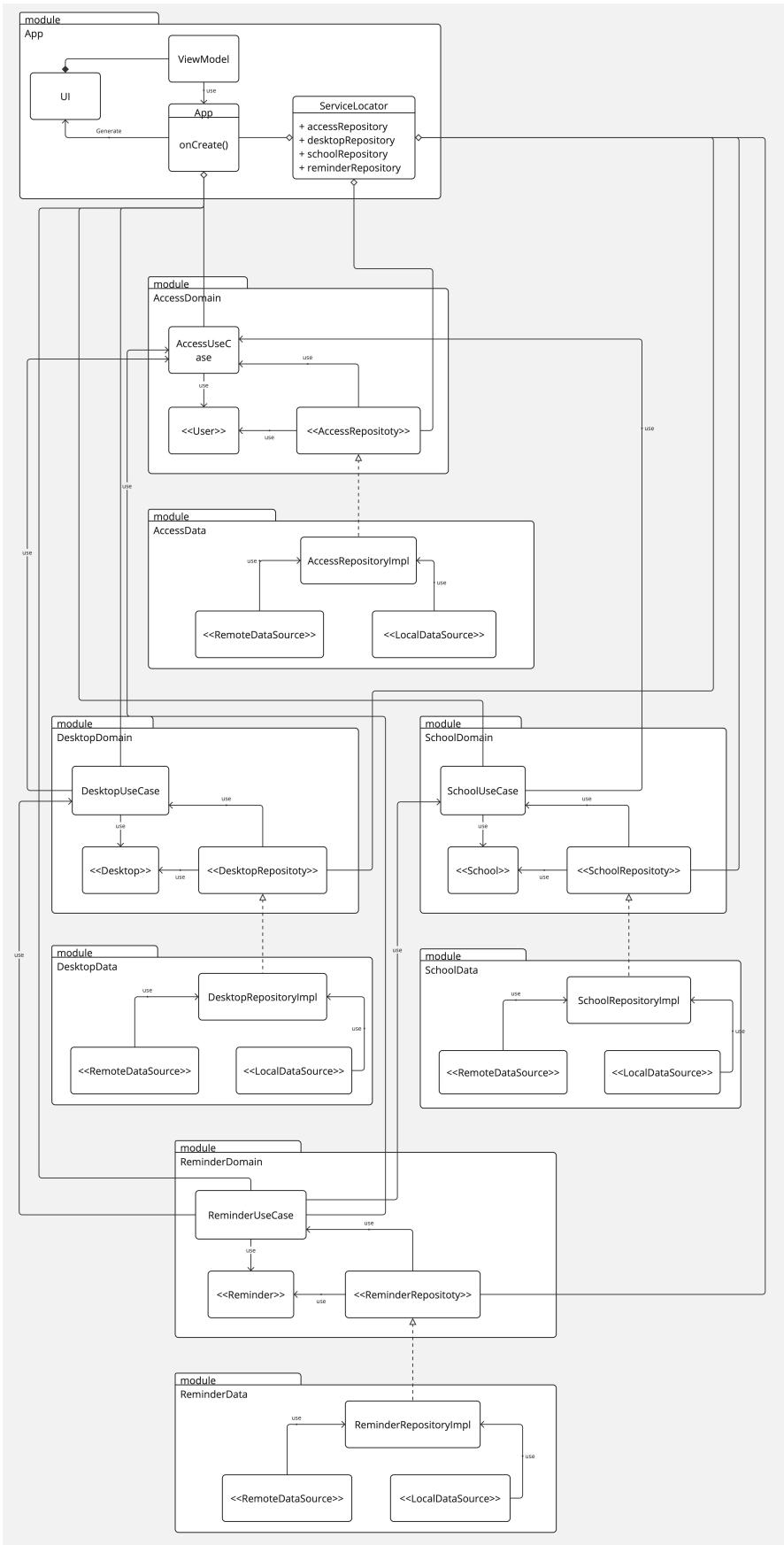


Figura 5: Design ad alto livello dell’architettura dell’applicazione

Il diagramma mostra in modo compatto l'architettura descritta in precedenza e le dipendenze presenti.

### Access Domain

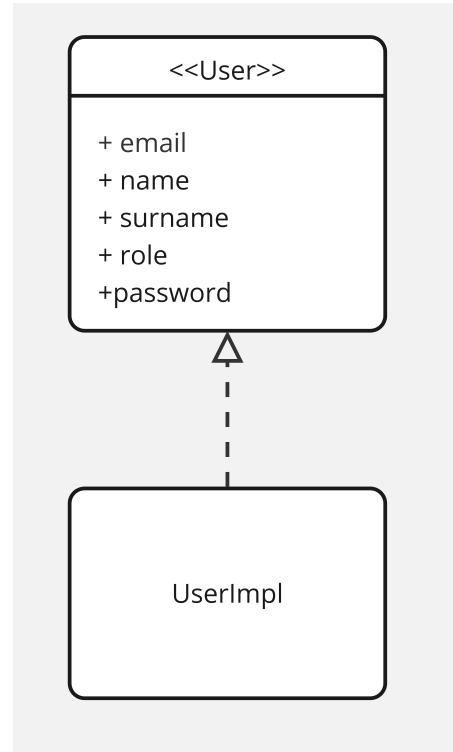


Figura 6: Design del dominio del componente access

La parte di dominio che riguarda le funzioni degli user e di accesso dell'utente (access) è molto semplice come struttura, come si vede dal diagramma. Infatti è composto da una sola interfaccia e da una classe, ma per gestire la creazione e i controlli dei dati dell'utente vengono implementate diverse policy (controlli sulla password, email e sul formato dei nomi e cognomi).

## Desktop Domain

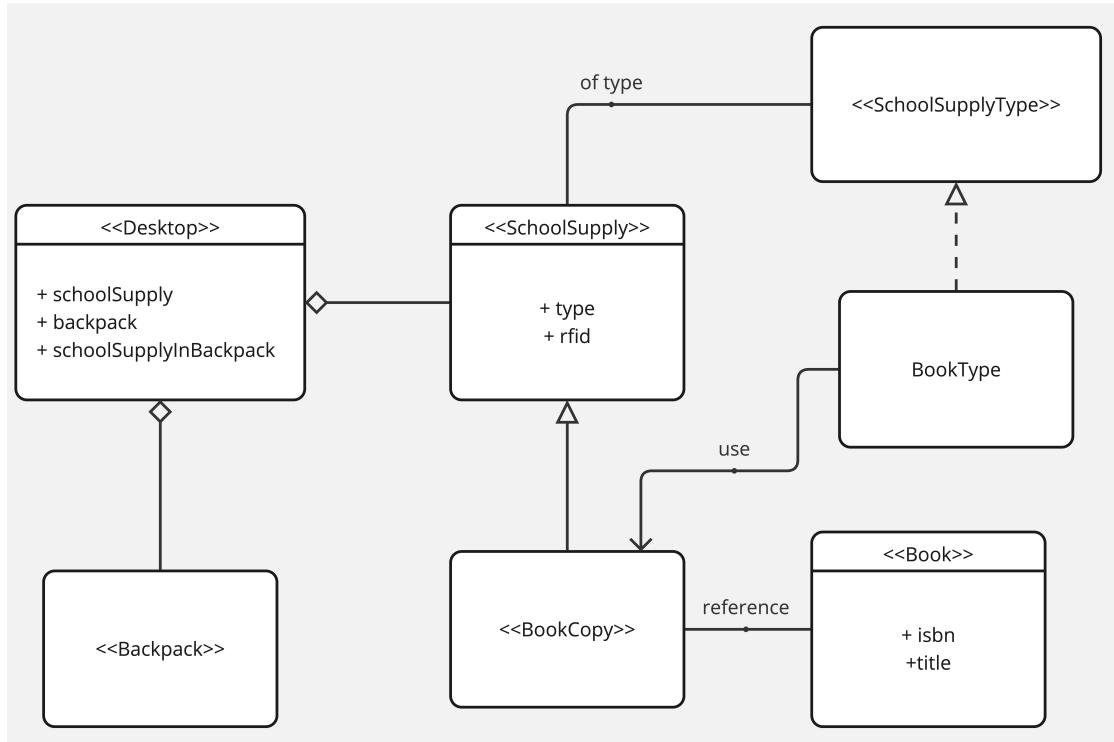


Figura 7: Design del dominio del componente desktop

La parte di dominio Desktop gestisce lo zaino e le copie dei libri dell’utente e viene pensato come la scrivania dello user. Ogni utente, al momento della creazione, crea anche la sua personale scrivania, ogni volta che sia aggiunge una copia, ossia un rfid, questo viene aggiunto alla scrivania. La copia fa riferimento ad un libro che viene identificato tramite isbn ( a 13 cifre). Anche in questo caso vi sono delle policy per controllare che il codice sia valido ((informazioni)). La classe desktop tiene anche le informazioni di quali elementi sono nello zaino e quali no. Per semplificare l’implementazione di sviluppi futuri viene creato un tipo di materiale (school supply) per gestire anche la creazione di altri tipi di materiale oltre ai libri.

## School Domain

La parte di dominio di dominio School gestisce la parte dell’app dedicata alla gestione delle lezioni. Per ogni utente viene considerata la scuola e il calendario.

Se è uno studente l'orario è riferito alla classe, mentre per i professori l'orario è personale. Per rappresentare al meglio la relata si dividono le lezioni in ordinarie, ossia quelle che sono definite sdal normale orario scolastico e quelle che altrarno l'orario (nuove lezioni, cambiamenti o cancellazioni). Come detto delle interviste vi sono due tipi di lezioni, quello in data e quelle di settimana, stessa casa vale per i cambiamenti, possono avvenire solo un giono o una serie di giorni. Per quanto rigurda le policy si è conseiderato il fatto che una classe o un professore non possano avere due lezioni contemporanemente.

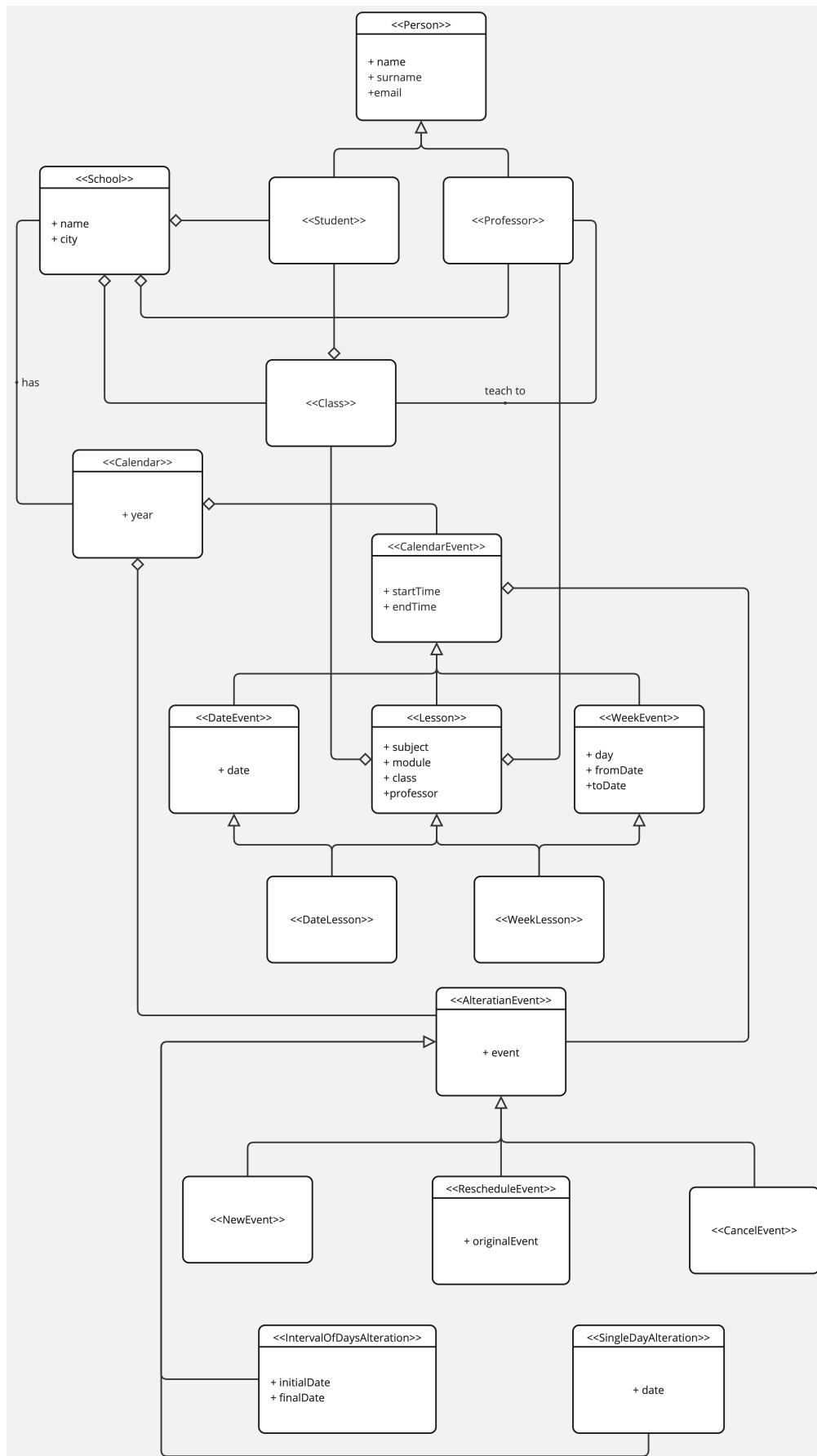


Figura 8: Design del dominio del componente school  
25

## Reminder Domain

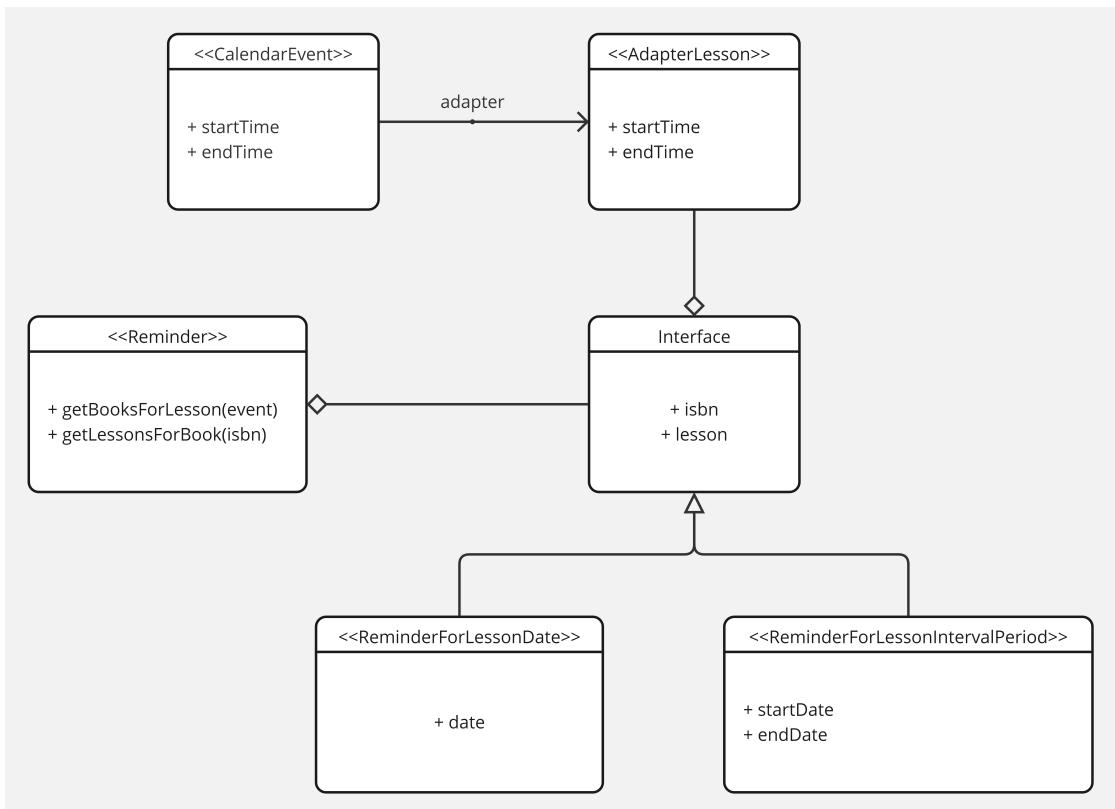


Figura 9: Design del dominio del componente reminder

La parte di dominio reminder definisce il collegamento tra school e desktop, in modo da assegnare un libro ad un evento. Necessita di un adapter per gli eventi del dominio school e l'ISBN di un libro dal dominio desktop. Anche in questo caso il promemoria può essere per un solo giorno o un periodo di tempo. Per quanto riguarda le policy, un promemoria deve avere un intervallo minore o uguale a quello di un evento.

## 4.2 Raspberry

L'architettura del software che andrà a governare il dispositivo presente all'interno dello zaino è basata sul pattern esagonale, Ports&Adapter e Clean Architecture, per isolare al meglio la logica di dominio e renderla indipendente da implementa-

zioni e tecnologie esterne o repository per l'accesso ai dati. All'interno del modello esagonale, possiamo notare un sistema diviso in tre livelli.

## Infrastructure Layer

Questo livello contiene tutti i dettagli tecnici relativi al sistema software, indipendenti dalla logica di business, come le tecnologie di lettura di tag RFID e comunicazioni in rete. In questo livello troviamo gli Inbound Adapter riguardanti le comunicazioni dal servizio Azure Hub IoT e i segnali riguardanti la lettura di un tag RFID, inviando un unico messaggio, in base all'evento, ad un middleware di gestione eventi, il quale, facendo parte dell'Application Layer, reindirizza l'evento alla logica di dominio. Un altro componente di questo livello è il database locale, utilizzato qualora la connessione non sia disponibile per memorizzare il contenuto attuale dello zaino, che viene poi sincronizzato con il database remoto realtime, attraverso il Network Module, ovvero l'Outbound Adapter.

## Application Layer

Rappresenta in un certo senso i casi d'uso e comportamenti del sistema, ma operando ad un livello superiore al domain model, espone un set di servizi che nascondono i dettagli di dominio, con il quale interagisce mediante le *Inbound o Outbound ports*. In questo livello troviamo due middleware molto importanti: un gestore eventi e un repository gateway. Il primo viene usato per intercettare i messaggi di eventi scaturiti ed eseguire un'opportuna operazione sulla logica di dominio in base all'evento in questione, che può essere la registrazione dello zaino presso uno studente, o l'aggiunta/rimozione di un oggetto dallo zaino. Queste ultime due componenti solo le Inbound Ports, operano da interfaccia tra il livello applicativo e quello logico. Oltre a ciò è presente un Outbound Ports, per comunicare all'esterno le modifiche apportate al contenuto dello zaino per sincronizzazione con il database remoto in cloud. Un middleware, chiamato repository gateway, si occupa di gestire la modifica di un elemento (mediante interfaccia Repository) sia sul database locale sia su quello remoto, incapsulando la gestione della doppia base di dati su cui attingere e la loro sincronizzazione, che avviene, come detto, mediante cooperazione tra gateway e il modulo di rete.

## Domain Logic o Domain Layer

Questo rappresenta il nucleo, la logica di dominio o di business, il cosiddetto Domain Model. C'è isolamento tra complessità a livello di modello del dominio e complessità tecnica dovuta ad aspetti e applicazioni di tipo tecnico. In questo livello viene gestita tutta la logica di business, quindi gli use case, le policy, entità e value objects.

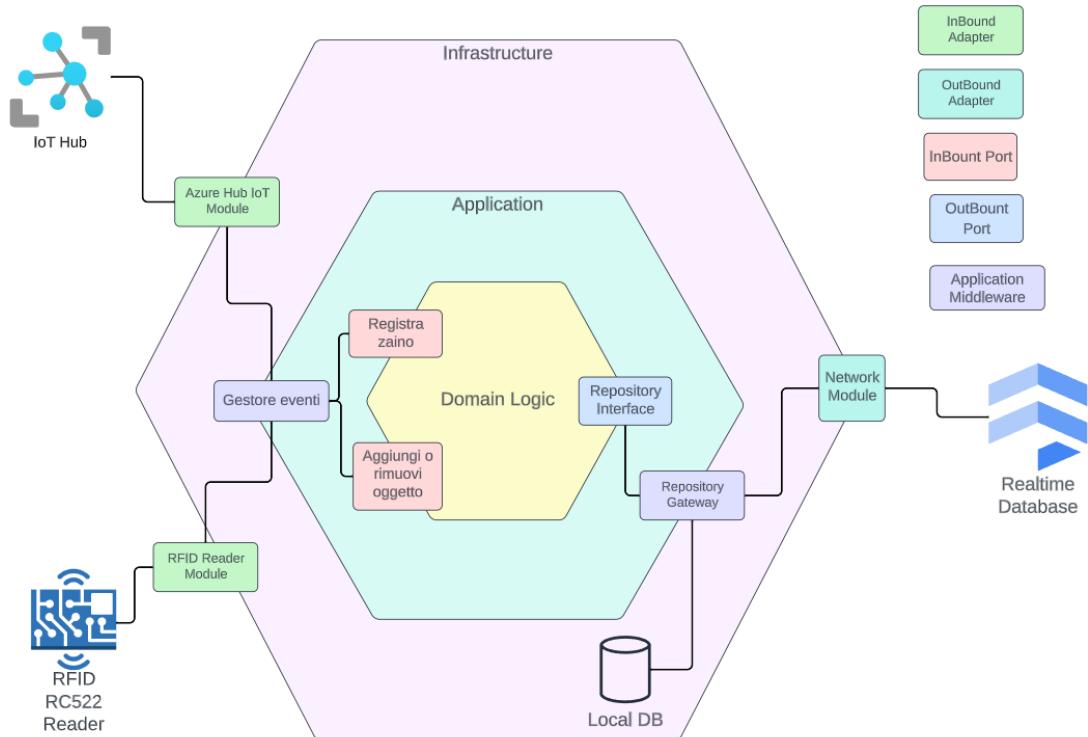


Figura 10: Design ad alto livello dell'architettura del dispositivo

Fondamentale importanza ha avuto il pattern Dependency Inversion, ovvero inversione di dipendenza, mirato a dare massima indipendenza e autonomia nei vari componenti, a fronte di sostituzioni o aggiornamenti di porzioni di codice o tecnologie. Un determinato layer è dipendente solamente dal livello immediatamente sotto:

- Layer infrastrutturale dipendente dal livello applicativo: esso si deve adeguare alle API e adapter forniti dal livello Applicazione

- Allo stesso modo il livello applicativo deve adeguarsi alle API fornite dal livello di dominio per cambiarne lo stato

In questo modo non ci sono altre dipendenze a catena, la gestione degli use case viene delegato al Domain Layer, il quale ne ha la responsabilità.

La struttura interna del sistema software da sviluppare per il dispositivo è stata pensata per soddisfare il modello esagonale, separando ogni layer in package differenti, i quali contengono tutti i componenti necessari inerenti ad uno specifico layer. L'organizzazione dei moduli ad alto livello è visibile nella seguente immagine 11:

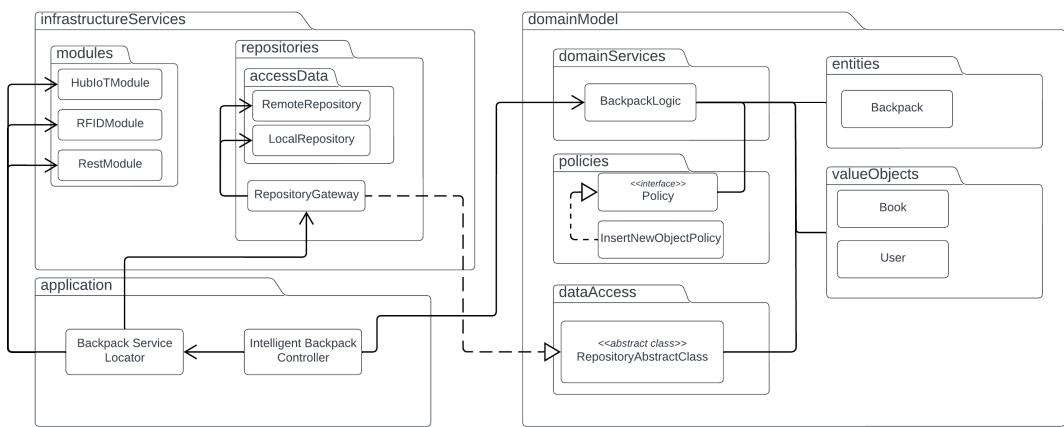


Figura 11: Design ad alto livello dell'architettura

I moduli identificati sono in relazione 1:1 con i livelli del modello esagonale, ovvero **infrastructureServices** rappresenta il livello infrastrutturale, il **domainModel** equivale al livello di Domain Business, stessa cosa ovviamente per il modulo **application**.

## InfrastructureServices

Sono tutti i servizi di infrastruttura, che contengono i dettagli relativi alle effettive tecnologie implementative e dettagli di tipo tecnico. Il package **modules** contiene le definizioni dei thread che gestiscono determinati compiti relativi a tecnologie diverse:

- HubIoTModule: modulo pensato per la comunicazione con la rappresentazione digitale del dispositivo (Device Twin sulla piattaforma Azure Hub IoT, come vedremo in seguito)
- RFIDModule: modulo di gestione del componente reader/writer di tag RFID
- RestModule: modulo che gestisce le richieste HTTP in uscita, ovvero tramite il modulo di rete prende in ingresso una serie di richieste HTTP e le inoltra, attendendone eventualmente il risultato

Per quanto riguarda **repositories**, questo modulo contiene le definizioni delle basi di dati su cui sincronizzare il proprio contenuto interno:

- RepositoryGateway: concentratore di query, questo componente prende in ingresso le query da effettuare e le smista al database locale e al database remoto (se impostato), mantenendo entrambi sincronizzati, rappresenta quindi un unico punto di accesso ai dati
- Local e Remote Repository: rispettivamente modellazione del repository locale al dispositivo e quello remoto situato su qualche servizio in cloud

## Application

Opera ad un livello più alto d'astrazione rispetto ai servizi di dominio, definendo una serie di servizi utili all'esecuzione globale del sistema come una *facade* del modello di dominio: infatti in questo modulo è presente il controller (ovvero il middleware Gestore Eventi presente nella figura 10), che riceve messaggi dai moduli thread citati nel livello infrastrutturale, indirizzando le relative opportune richieste al livello di dominio, come lettura di un nuovo tag RFID o ricezione di un messaggio dal cloud. In questo modulo troviamo anche il Service Locator, concentratore delle dipendenze in modo da applicare il pattern Dependency Injection, per rendere il sistema più modulare.

## DomainModel

In quest'ultimo livello troviamo la logica di business, quindi il servizio di dominio **BackpackLogic**, il quale gestisce i casi d'uso del sistema, interfacciandosi alle

entità (il Backpack) e ai value objects definiti (Book e User). Le policy invece che determinano le condizioni presenti all'interno del dominio sono contenute nel package **policies**, con una interfaccia generica e la sua implementazione nella policy di inserimento di un nuovo oggetto. Infine il metodo di accesso ai dati da parte della logica di dominio è rappresentato dall'interfaccia **RepositoryAbstractClass**, che contiene i casi d'uso relativi alla persistenza dei dati propri dello zaino. Questa interfaccia è implementata dal RepositoryGateway già visto nel livello infrastrutturale: la sua implementazione viene istanziata con dependency injection all'interno della logica di dominio, cosicché la responsabilità e implementazione è estranea al livello di dominio. Nell'immagine seguente è possibile vedere nel dettaglio uno schema UML delle classi facenti parte del domain model, ovvero le policy usate, la logica di business, entità e value object.

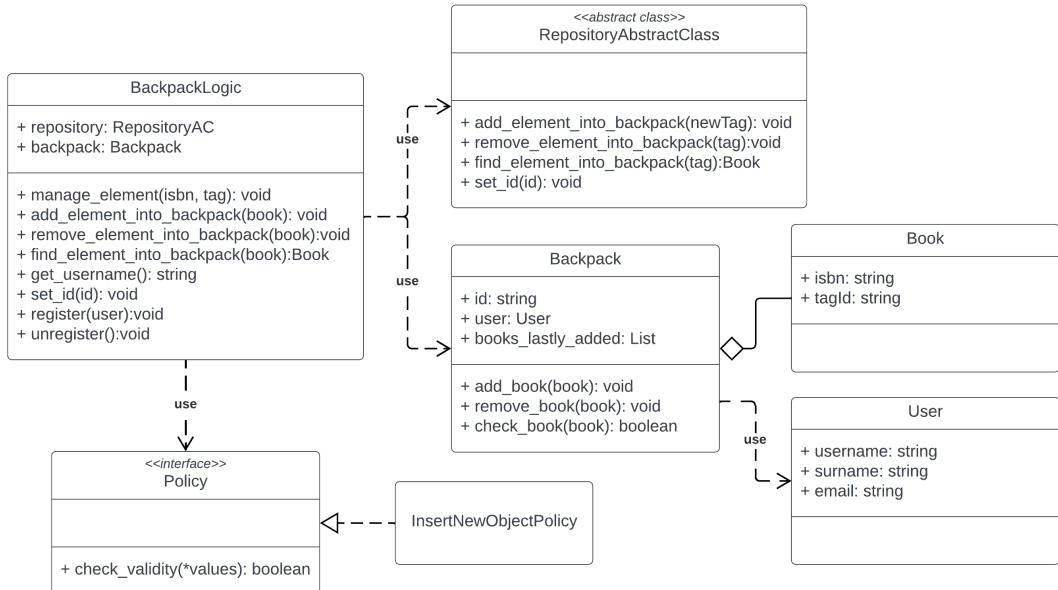


Figura 12: UML del Domain Model

### 4.3 Sistema di backend

Una volta che si sono identificati i bounded context presenti per il sistema di backend, è stato necessario individuare una strategia di integrazione che permette di rendere i bounded context i più autonomi possibili. Questo è stato possibile

mediante l'utilizzo di una architettura a microservizi, la quale garantisce anche isolamento da singoli errori del sistema e una facile scalabilità del sistema (requisito richiesto).

Essendo stato utilizzato un approccio di analisi DDD con microservizi, ogni servizio è stato analizzato e modellato per far sì che corrispondesse ad un concetto del dominio specifico. In questo modo, ogni servizio può essere sviluppato, versionato, testato e distribuito autonomamente senza causare problemi ad altri servizi.

Inoltre, l'utilizzo dei microservizi facilita la comunicazione, in quanto ciascun servizio ha la propria interfaccia API ben definita e documentata. Ciò favorisce la separazione delle responsabilità, permettendo di evitare dipendenze complesse tra i servizi.

Un altro aspetto importante nell'utilizzo dei microservizi in un approccio DDD è la flessibilità nella scelta delle tecnologie. Ogni servizio può essere sviluppato utilizzando la tecnologia più adatta alle sue specifiche esigenze, senza dover rinunciare alla coerenza globale del sistema.

In particolare, come detto precedentemente, una volta identificati i bounded context si è analizzato e modellato il sistema in funzione di essi. Una delle decisioni prese durante questa fase è stata quella di far coincidere i microservizi direttamente con il bounded context. Questo è dato dal fatto che il servizio Azure non permette la creazione di più di 1 server gratuito per servizio. Questo avrebbe comportato la presenza di numerosi servizi dislocati in varie regioni del mondo, andando ad aumentare complessità e latenza. Questo ha portato alla decisione di ridurre il numero di servizi da sviluppare e di svilupparli facendo sì che ricoprono il proprio bounded context.

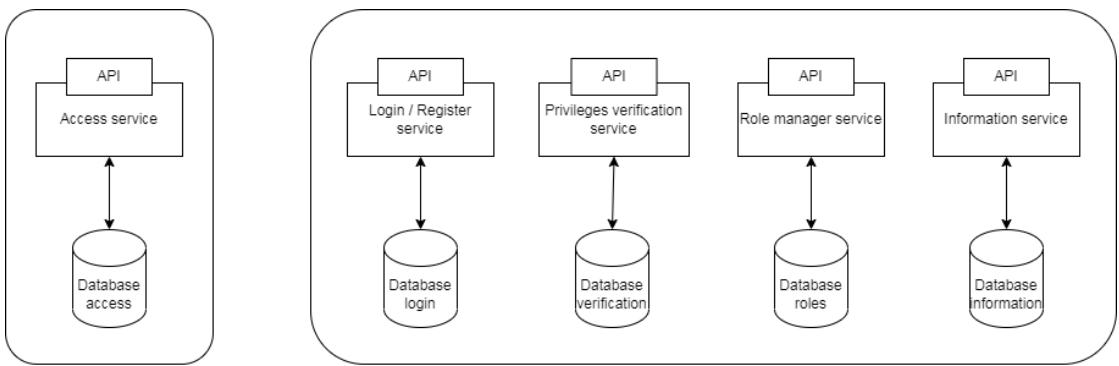


Figura 13: A sinistra: architettura attuale dettata dalle limitazioni del servizio Azure gratuito; A destra: architettura con divisione dei servizi nello stesso bounded context

Tutti i microservizi sono stati realizzati con le stesse caratteristiche:

- Connessione al database: Viene effettuata solamente prima di eseguire la query per ottenere i dati dal database, utilizzando il config predefinito di Azure. Inoltre, la comunicazione viene immediatamente chiusa una volta ottenuto il risultato della query, riducendo la possibilità di possibili accessi non desiderati;
- Configurazione del server: Viene creata una istanza di Express chiamata app dove vengono definite le rotte del sistema e impostata la ricezione di dati JSON da parte del server (necessari per la comunicazione via protobuf);
- Standardizzazione dei messaggi: Le richieste/risposte inviate al/dal server utilizzano tutte il formato di comunicazione protobuf, che permette una standardizzazione della comunicazione tra parti sviluppate con tecnologie diverse;
- Utilizzo delle route: All'interno di ogni servizio vengono definite e usate rotte specifiche per avere una gestione meglio divisa di tutte le chiamate;
- Avvio del server: È stato definito per default che i servizi vengano avviati sulla porta 80, oppure sulla porta definita di default direttamente all'interno della macchina.

## Interazione tra i microservizi

I diversi microservizi, per poter svolgere le proprie funzionalità, potrebbero avere la necessità di comunicare e interagire tra di loro, per verificare possibili vincoli nelle chiamate o identificare informazioni non disponibili nel microservizio attuale.

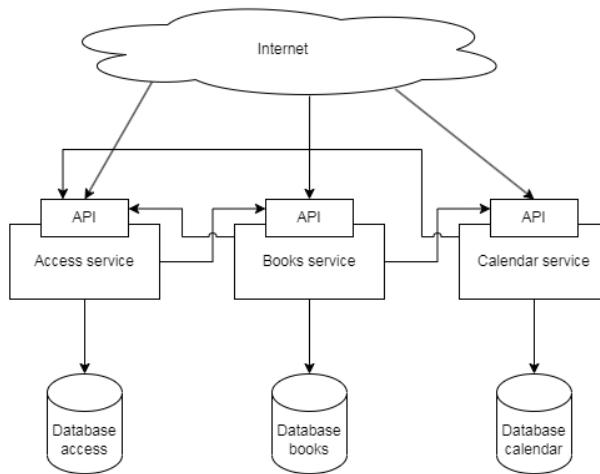


Figura 14: Visualizzazione ad alto livello delle interazioni tra i microservizi.

Come si può vedere dalla figura sopra riportata, le interazioni tra i microservizi avvengono in modo unidirezionale, andando a comunicare direttamente con le API degli altri servizi. Il servizio che ha aperto la comunicazione, dovrà poi gestire la risposta da parte dell'altro servizio in modo sincrono o asincrono. Inoltre, le API dei microservizi vengono direttamente utilizzate mediante chiamate HTTP sulla rete, richiedendo perciò verifiche sui messaggi in arrivo.

Un'altra decisione che è stata presa durante la fase di progettazione è la presenza di una comunicazione diretta tra i client e i microservizi, invece di utilizzare un intermediario tra le 2 entità, chiamato gateway API. Questa decisione è stata presa basandosi sulle seguenti motivazioni:

- Complessità aggiuntiva: l'utilizzo di un gateway API implica l'introduzione di un altro componente nel sistema, che può aumentare la complessità complessiva dell'architettura. Questo può avere un impatto negativo sulla manutenibilità e sulla scalabilità del sistema (uno dei requisiti principali richiesti);

- Perdita di flessibilità: l'utilizzo di un gateway API può limitare la flessibilità nell'aggiungere o modificare i microservizi all'interno del sistema. Visto che il gateway potrebbe richiedere determinate politiche o strutture di routing, potrebbe risultare difficile effettuare modifiche ai servizi senza dover modificare anche il gateway;
- Ridondanza delle funzionalità: i gateway API spesso offrono funzionalità di sicurezza, autenticazione e gestione del traffico. Tuttavia, questi stessi servizi possono già essere forniti da altri componenti all'interno dell'architettura, facendo sì che l'utilizzo di un gateway API potrebbe comportare la duplicazione di alcune funzionalità già esistenti.

## 5 Plugin Gradle

In questa sezione mostreremo lo sviluppo di due plugin risultati necessari per lo sviluppo della build automation di alcune componenti del sistema, sviluppate in Gradle, in particolare per il software Python destinato al Raspberry Pi 4 e per il versioning delle applicazioni Android, in modo da versionare in modo coerente il software mobile.

### 5.1 gradle-python-testing

Questo Plugin per Gradle è nato dall'idea di voler semplificare l'esecuzione dei test e della coverage per un progetto Python ma in un'ambiente di build automation come Gradle, quindi fornendo task per:

- Eseguire i test e fallire nel caso almeno uno non esegua correttamente
- Eseguire un test con riscontro dettagliato sui metodi che hanno riscontrato errore
- Calcolare la coverage in base ai test eseguiti, fallendo nel caso in cui il valore in percentuale della coverage sia più basso di un determinato threshold
- Eseguire i test in ambiente virtuale (virtualenvironment di Python) o in ambiente User

Il grafo delle dipendenze si può riassumere nella seguente immagine, dove la freccia rappresenta la semantica di *dependsOn*.

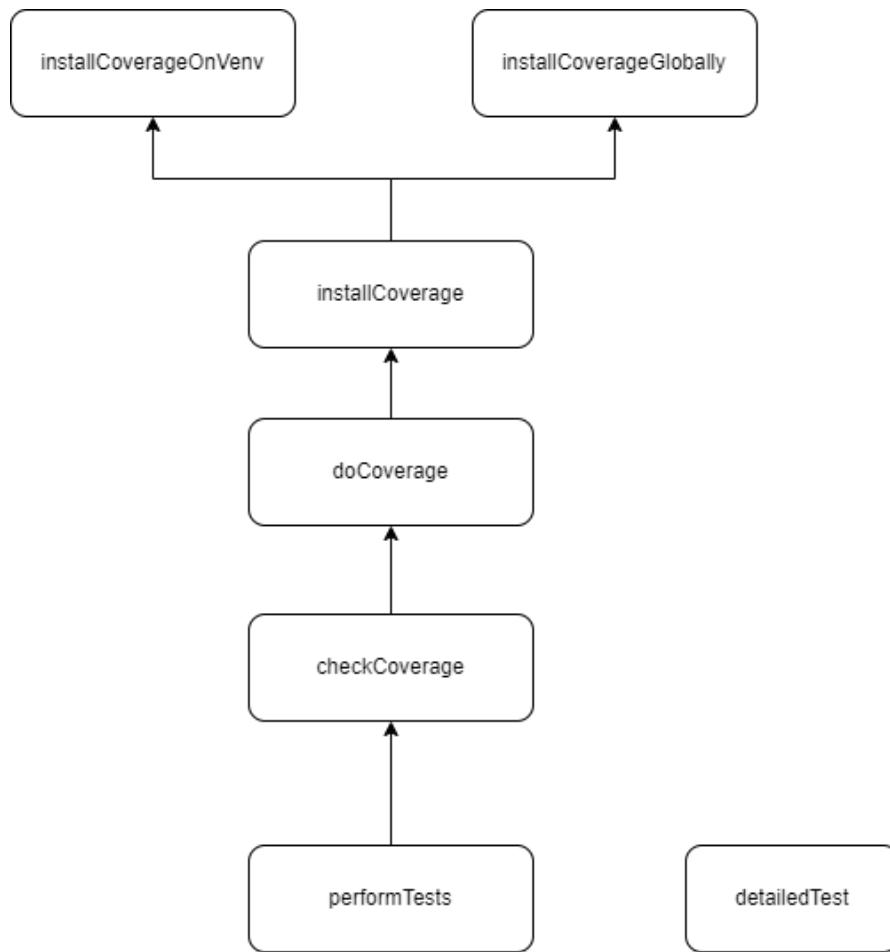


Figura 15: Task Dependency Tree

Il plugin è stato sviluppato in linguaggio Kotlin sfruttando le api offerte da Gradle per definire task. Più in particolare, i task definiti sono i seguenti:

- `installCoverageOnVenv`: task che installa il modulo coverage di Python sul virtual environment indicato nella configurazione (deve pertanto esser già esistente quando si usa il plugin)
- `installCoverageGlobally`: task che installa il modulo coverage a livello di User, quindi globalmente, se non già installato
- `installCoverage`: task che non ha funzionalità, dipende o da `installCoverageOnVenv` o da `installCoverageGlobally`, in base a come è stato configurato il plugin, ovvero se in modalità di virtual environment o in modalità global

- doCoverage: task che dipende da *installCoverage*, in quanto attende che la coverage sia correttamente installata
- checkCoverage: task che controlla se la coverage calcolata è al di sopra di un determinato livello percentuale, fallendo se minore; dipende da *doCoverage* in quanto necessita la coverage già calcolata
- performTests: task che esegue i test senza dettagli su eventuali errori, pensato per essere aggiunto come dipendenza del task *test* nella propria build automation in Gradle; dipende da *checkCoverage*, quindi se questo task passa correttamente significa che la coverage è soddisfatta e i test hanno dato esito positivo
- detailedTest: task che effettua solo i test in modo dettagliato, mostrando i dettagli dei test che hanno prodotto errori

### 5.1.1 Configurazioni disponibili

Le configurazioni disponibili danno la possibilità di impostare determinati valori per adattare il funzionamento del plugin. I valori configurabili sono i seguenti:

- testSrc: indica la directory dove sono contenuti i file di test in python
- minCoveragePercValue: indica il valore minimo in percentuale di coverage che si vuole soddisfare, in modo da far fallire la build nel caso in cui il valore sia minore di questo threshold
- useVirtualEnv: valore booleano che indica la volontà di usare un virtual environment o meno su cui utilizzare o installare il modulo di coverage
- virtualEnvFolder: nel caso si utilizzi il virtual environment, è necessario specificare con questo parametro la directory dove è definito l'ambiente virtuale di python
- coverageAutoInstall: parametro booleano che indica la volontà di installare in automatico il modulo di coverage se non già correttamente installato; se impostato a false, il task di controllo della coverage fallirà nel caso non trovi il modulo

```

pytest {
    testSrc.set("src/test/python") // test folder that contains
        all python tests
    minCoveragePercValue.set(80) // min 0 - max 100 acceptable
        percentage of coverage
    useVirtualEnv.set(true) // true if you use a virtual
        environment or global libraries
    virtualEnvFolder.set(".gradle/python") // virtual env folder
        if you use it
    coverageAutoInstall.set(true) // if coverage module is not
        installed, install it
}

```

## 5.2 Gradle-Git-Sensitive-Semantic-Versioning-Plugin-for-Android

Questo plugin nasce dall'esigenza che android non gestisce la versione solo con un classico nome, chiamata in android version name, (ad esempio usando semantic versioning), ma usa anche un intero, chiamato version code, per gestire quando aggiornare un applicazione. Per automatizzare il rilascio si è scelto di creare un nuovo plugin gradle per gestire anche la generazione del version code. Questo plugin è un wrapper del plugin Gradle-Git-Sensitive-Semantic-Versioning-Plugin realizzato dal prof. Pianini con in più la gestione del version code. In particolare ha due modalità di funzionamento:

- Semantic Code: si divide in tre gruppi il numero di version code e si converte l'attuale semantic version (calcolato dal plugin tramite chiamata), esempio 1.2.3 come semantic version diventa 0001002003 come semantic code. Questo sistema ha il problema che si ha un numero massimo di release definito in base a quante cifre si dedicano per major minor e patch.
- Incrementale: si contano il numero di commit eseguiti fino a questo punto e quello diventa il numero di versione, è quello usato in automatico. Non ha particolari problemi legati al numero di release e rispecchia maggiormente il concetto di versionCode, inoltre risolve il problema per cui un apk nuovo non sovrascriva quello vecchio perché hanno lo stesso version code (questo problema è implicitamente risolto quando si carica un apk tramite android studio, ma si presenta quando passato in altro modo).

### **5.2.1 Configurazioni disponibili**

Essendo un wrapper di Gradle-Git-Sensitive-Semantic-Versioning-Plugin questo plugin permette le stesse configurazioni e permette in più di definire come gestire il versionCode, in particolare se averlo incrementale o meno, e quante cifre assegnare per major minor e patch.

## 6 Sviluppo e pratiche DevOps

L'intero progetto è stato realizzato sfruttando una Build Automation sviluppata ad Hoc per ogni componente, con l'obiettivo di automatizzare il ciclo di sviluppo del software e una Continuous Integration/Deployment per automatizzare integrazione delle modifiche apportate, distribuzione e deployment, senza tralasciare continuous testing e quality assurance.

### 6.1 DVCS e workflow

Prima di tutto è utile partire mostrando come è stato configurato il sistema di versioning del codice e di tutti i repository appartenenti al progetto globale e la loro automatizzazione nelle procedure di DevOps, fondamentali per ridurre al minimo il tempo di testing, deploy e meccanismi facilmente automatizzabili.

Per quanto riguarda il sistema di versioning è stato scelto GitHub, per ovvie ragioni di comodità e funzionalità offerte. Si è scelto di creare un'organizzazione multi-repository per raggruppare tutte le componenti di sistema, il relativo codice e la relativa configurazione di build. Si è scelto inoltre di rispettare la convenzione del workflow GitFlow, ovvero un flusso di lavoro preciso basato su alcune regole di sviluppo su un repository Git, per facilitare lo sviluppo di nuove funzionalità in parallelo senza perdere o creare danno alla linea corrente di distribuzione. Più in particolare, regola l'utilizzo di determinati branch per determinati obiettivi, come possibile vedere nella seguente figura 16.

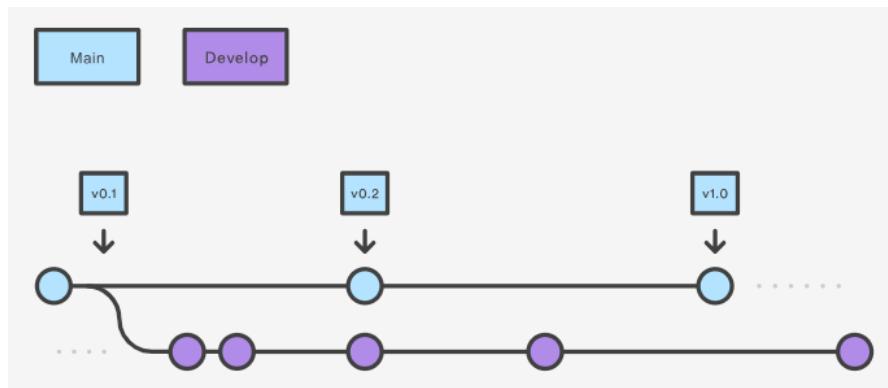


Figura 16: Flusso di sviluppo adottato

Il branch *main*, talvolta chiamato anche *release*, è la linea di sviluppo di distribuzione, mostrando la cronologia di rilascio ufficiale, quella effettivamente in produzione, mentre uno o più branch *develop* o *feature-XXX* vengono creati per aggiungere/testare una nuova funzionalità, senza intaccare il branch di produzione. Per mantenere la storia di progetto il più pulita possibile, si è fatto uso in rari casi di *Squashing* di determinati commit non particolarmente importanti dal punto di vista informativo (fix della CI chain o documentazione).

## 6.2 Controlli pre-commit

Per dare una migliore leggibilità alla cronologia del progetto, quindi alle descrizioni delle varie modifiche apportate al sistema, si è scelto di utilizzare controlli pre-commit per accettare modifiche ben descritte e rifiutare quelle non conformi. Più nello specifico, la convenzione adottata è quella del Conventional Commit, ovvero includere nel messaggio di commit un prefisso significativo che descriva la natura delle modifiche apportate al sistema con quel preciso commit, ovvero il *type*:

- chore: qualsiasi modifica generica che non sia una nuova funzionalità o un fix di qualche problema, manutenzione generale o modifica di README
- feat: messaggio che indica l'aggiunta di una nuova funzionalità
- build: una qualsiasi modifica appartenente alla build automation, ad esempio Gradle o NPM
- ci: rappresenta una modifica apportata ai workflow della Continuous integration/deployment
- docs: indica una modifica o l'aggiunta di documentazione relativa al codice sviluppato
- fix: risoluzione un problema, bug non voluto o code smell

Il messaggio ad esempio, dovrà essere formato nella seguente maniera:

<type>[optional scope]: <description>

Il controllo viene effettuato con Git Hooks sul messaggio di commit. Una volta definita la forma del messaggio adottato, abbiamo definito controlli pre-commit, ovvero dei check effettuati sul codice per accettare il commit o rifiutarlo:

- Check di quality code, ovvero che la qualità del codice sia in linea con le regole definite (convenzioni)
- Check dei test che eseguano senza errori

### 6.3 Utility utilizzate

Sono state utilizzate due utility aggiuntive ai repository:

- Renovate: applicazione che permette una migliore gestione delle dipendenze e il loro aggiornamento, con pull request automatiche del bot qualora trovi una nuova versione per una dipendenza utilizzata nel progetto
- Semantic Release: bot utilizzato per generare una nuova release con la versione indicata dall'utility semantic-release-preconfigured-conventional-commits su Gradle, basato sulla Semantic Release che crea etichette basate sulla semantica dei commit
- SonarCloud: applicazione integrata con il sito web che permette di analizzare il codice ed individuare code smell, ripetizioni, bug e altre statistiche.

### 6.4 Build automation

Una volta che si è deciso quale sistema di versioning utilizzare, si è proseguito andando a definire l'organizzazione della struttura su GitHub. All'interno dell'organizzazione GitHub sono stati realizzati una serie di repository, come anticipato, uno per ogni componente del sistema:

- **IntelligentBackpackApp**: Tratta tutti gli aspetti della gestione dell'applicazione smartphone;
- **IntelligentBackpack**: software Python per il Raspberry Pi 4, che rappresenta lo zaino intelligente
- **Microservice Book**: microservizio di gestione dei libri e tutto ciò che ne concerne
- **Microservice Access**: microservizio di gestione degli accessi, ruoli e utenti

- **Microservice Calendar:** microservizio di gestione del calendario, orari, lezioni
- **ManageBackpackService:** microservizio di gestione degli zaini registrati su Azure IoT Hub
- **documentation:** repository contenente la documentazione di fino riguardo alcuni aspetti dell'analisi DDD
- **DTO Data Transfer Object:** strutture che sono passate tra server e cliente tramite body HTTP

Per ognuno di questi sotto progetti è stata realizzata una Build Automation ad Hoc, utilizzando NPM per i microservizi e Gradle per Android (Kotlin) e Python, per automatizzare tutte le procedure manuali ma automatizzabili, come:

- Esecuzione test
- Quality code assurance
- Generazione documentazione
- Firma e creazione bundle nel caso dei Plugin sopra citati
- Altre funzionalità custom come comandi Python
- Gestione delle dipendenze
- Gestione subproject e relative dipendenze
- download del compilatore protobuf per il linguaggio usato e compilazione dei file
- Tutte le configurazioni di progetto

#### 6.4.1 Firmware Python per Raspberry

Sono stati realizzati task custom in Gradle ad Hoc per il firmware in Python, destinato al Raspberry, in modo da automatizzare determinati aspetti specifici, come gestione di un progetto Python, quality assurance e creazione della documentazione.

## 6.5 Submodules

Come già detto si è organizzato il progetto creando un organizzazione con al suo interno i diversi sistemi. I 3 DTO (AccessCommunication, BookCommunication e CalendarCommunication) servono per gestire la serializzazione del body delle chiamate HTTP del server. Essendo scritti in Protocol Buffers (linguaggio neutro di Google) possono essere importati in un progetto, tramite submodule, e poi compilati nel linguaggio usato nel progetto. Questo permette di importare la stessa struttura sia sul server sia sul client, riducendo problemi di serializzazione e permette di mantenere aggiornato lo stato di entrambi a costo praticamente nullo. In questo modo, si traccia anche la storia dell’evoluzione dei formati dei messaggi di client e server.

## 6.6 Testing

Il testing è stato un punto fondamentale per i microservizi e l’applicazione mobile, mentre un po’ marginale nelle parti di software relativi all’hardware del Raspberry Pi 4, in particolare il sensore RFID e il modulo di rete, i quali hanno portato ad un valore di coverage finale non troppo alto. Per ogni componente si sono prodotti Unit Test mentre per l’applicazione mobile anche test d’integrazione tramite AndroidJunit.

### 6.6.1 Microservizi

Per quanto riguarda il testing dei microservizi, si sono adottate diverse modalità:

- Locale: Tramite l’utilizzo di un database locale (replica del database remoto) e dei comandi npm, venivano testate tutte le funzionalità implementate e la coverage, direttamente dalla macchina su cui veniva sviluppato il codice;
- Remoto (fase development): È stata creato un workflow custom che eseguisse i test una volta che veniva effettuato un push sul branch di develop. Questo è stato fatto per far verificare che tutti i test eseguiti in locale (e le relative funzionalità), funzionassero anche in un ambiente remoto. Ciò permetteva di effettuare direttamente dei fix durante la fase di develop, invece di dover aspettare per una release alla prima versione utile, dovendo

poi andare a sistemare funzionalità rotte, richiedendo tempo. Questo workflow successivamente è stato disabilitato per costi dovuti alla pratica di test continui.

- Remoto (fase release): È stato utilizzato un workflow di Azure per il deploy del microservizio sul server. Questo workflow comprendeva una fase iniziale di installazione dei moduli e testing del codice. Nel caso in cui i test fossero falliti, la build si sarebbe stoppata e non avrebbe proseguito con la fase di deploy.

Inoltre, eseguire i test durante la fase di release risulta essere più lento rispetto alla fase di deploy, in quanto è necessario che prima vengano effettuati i download dei moduli e poi il test. Oltre tutto, il server è impostato per lavorare con una versione ben definita di NodeJS, quindi effettuare test in fase di release per verificare anche altre versioni (diverse da quella impostata nel server) risulta inutile e dispendioso. Per questo motivo, durante la fase di development, i test venivano eseguiti su altre versioni di NodeJS, andando così a verificare, direttamente durante la fase di sviluppo, fino a quale versione il codice fosse retrocompatibile. Ciò permette anche di poter verificare quali funzionalità sviluppate vanno ad escludere versioni precedenti, facendo sì che si possano apportare tempestivamente modifiche per far sì che ciò non accada (molto difficile/costoso da fare se i test vengono solo effettuati in fase di release).

## 6.7 Continuous integration

Per semplificare la lettura della cronologia del software, avere una chiara idea sul suo ciclo di sviluppo e avere più potere d'azione in caso di problemi e necessità di revert su vecchi stati del progetto, si è scelto di adottare un approccio ad integrazione continuativa, quindi integrazione di piccole quantità di modifiche apportate e continue. Questo approccio ha portato alla creazione di diverse release nel corso dello sviluppo, alcune non complete, specialmente le primissime versioni, ma ha semplificato enormemente la capacità di resilienza in caso di bug introdotti o problemi di integrazione. A questo scopo si è scelto di utilizzare le pipeline di integrazione di GitHub, ovvero le GitHub Actions, che ci permettono di integrare il codice svolgendo controlli e test su macchine virtuali configurate appositamente. Nel seguito verranno mostrate le pipeline di integrazione utilizzate.

## Workflow per i plugin

Per i plugin di Gradle sviluppati, è stata realizzata una pipeline di continuous integration ad hoc, per poter testare in più ambienti l'esecuzione del plugin e gestire il rilascio sulle varie piattaforme. In particolare, nella seguente immagine possiamo avere un'idea dei job scritti.



Figura 17: Pipeline dei job dei plugin

Tra i job sviluppati, oltre a compile, troviamo:

- jvm-version: job che, mediante comando in ruby e comunicazione tra job, crea una lista di versioni numeriche, associandolo alla variabile d'ambiente GITHUB\_OUTPUT e raggiungibile tramite l'output *versions*
- test: job che instanzia una macchina virtuale per ogni versione e sistema operativo, realizzando una matrice prendendo in considerazione 3 sistemi operativi (Windows, macOS e Linux Ubuntu) e varie versioni della JVM per Java, definite dal job jvm-version. In ognuna di queste macchine virtuali viene eseguito il task gradle *test*, attendendone l'esito, che può essere positivo se esegue correttamente o negativo in caso di qualche errore intercettato. Questo task prevede un setup differente nei due plugin sviluppati:
  - Per il plugin di testing per progetti Python si è fatto uso di *actions/setup-python@v4* per aggiungere alla macchina virtuale (una tra tutte le occorrenze nella matrice delle VM) la distribuzione di Python, con versione 3.8, per poter effettuare correttamente i test sul plugin
  - Semantic Versioning Android: per testare il plugin su diverse versioni di jvm si usa una matrice con le principali versioni di java (8, 11, 17, 19), con distribuzione temurin e si sono effettuati gli unit Test su ogni versione.

- release: job che si occupa del rilascio di una nuova versione su GitHub e la relativa pubblicazione su Maven Central e Gradle Plugin Portal, recuperando tutti i segreti necessari dal repository in questione, come chiavi api ecc...

La stessa pipeline viene usata per pubblicare la libreria RetrofitProtobufJsonConverter che serve per convertire un json in una classe generata da un file protocol buffer.

## Applicazione Mobile

Per l'applicazione, la pipeline prevede 7 job, uno dei quali a matrice, come mostrato nell'immagine:

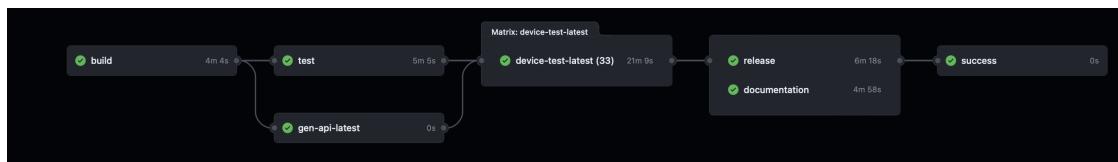


Figura 18: Pipeline dei job di IntelligentBackpackApp

- build: job che compila il codice, usa il task assemble di gradle
- test: job per eseguire gli unit test del modello (scritti in kotest) e dipende da build, usa il task test
- gen-api-latest: job per generare un lista di api android per gli instrument test, per le limitazioni del profilo (si hanno 2000 minuti al mese in comune), si è deciso di eseguire solo l'ultima versione (33).  
Dipende da build
- device-test-latest: job per eseguire gli instrument test scritti in jUnitAndroid. Per essere eseguiti è necessario creare un'emulatore e lanciare i test su di esso, per questo è abbastanza pesante (circa 25 minuti), inoltre richiede una macchina con MacOS per avere maggiori prestazioni.  
Risulta instabile e spesso fallisce per problemi legati all'emulatore.  
Richiede gen-api-latest e prende come input la lista di api.

- documentation: job che crea e carica la documentazione (usando dokka) sul branch docs, che è poi associato per github-pages.
- Richiede test e device-test-latest

- release: job per aggiornare il numero di versione dell'app e pubblicare l'apk e l'aab.

Richiede test e device-test-latest, inoltre al secret GITHUB-TOKEN per poter fare push dell'etichetta di rilascio.

Per generare l'apk vengono usati diversi secret, in particolare viene creato un file la chiave assegnata nel secret SIGNING\_KEY.

Per simulare il rilascio sullo store, che non è stato possibile non avendo un account android developer, viene pubblicato apk e aab sulla release di github.

Per tutte questi job viene usato java 17, unica versione supportata, e si necessita la creazione del file google-services.json in app il cui valore è salvato nei secret GOOGLE\_SERVICES e che contiene i dati per l'accesso a crashlytics.

## Firmware Python

Per il firmware in Python dello zaino, la pipeline prevede quattro job, dei quali due paralleli, come mostrato nella seguente immagine:

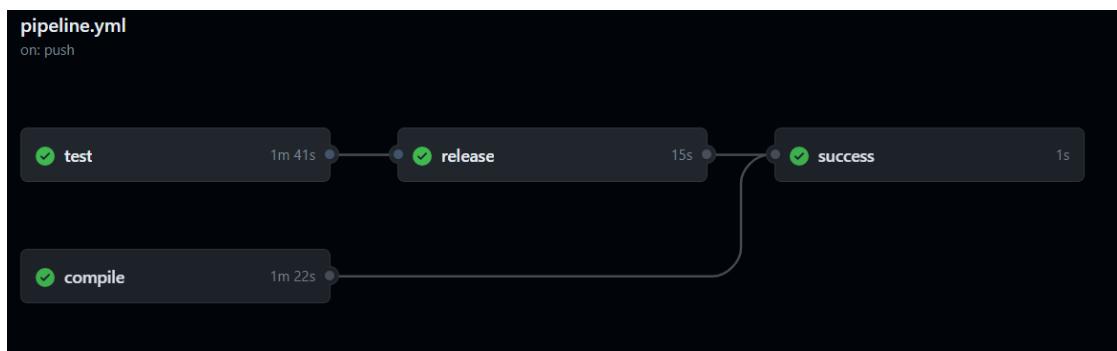


Figura 19: Pipeline dei job di IntelligentBackpack

- test: è il job che instanza una macchina virtuale sul quale lanciare i test sviluppati, che avrà successo se tutti i test andranno a buon fine
- compile: job che compila semplicemente il software mediante Build Automation, quindi dipendenze e configurazioni

- release: job che dipende da test e compile, che crea una release nel caso in cui entrambi i job precedenti passano con successo. Sfrutta semantic-release e accede al secret GITHUB-TOKEN per poter creare un etichetta di rilascio
- success: job che ha successo se tutti i job precedenti passano correttamente, altrimenti lancia le rispettive failure

La seconda pipeline usata è *pages build and deployment*, la quale aggiorna l’arte-fatto per la GitHub Pages, utilizzata per hostare la documentazione del codice.

## 6.8 Versioning delle release

Come accennato, per gestire le etichette delle versioni delle release si è utilizzato il modulo NPM *semantic-release*, il quale permette, in base alla storia dei commit, di calcolare l’etichetta della versione successiva, basato sul contenuto del messaggio di commit, ovvero il suo formato (type, come precedentemente visto):

- MAJOR: viene creata una major se il tipo di messaggio e lo scope terminano con !, causando una *breaking change*
- MINOR: viene rilasciata una minor se il tipo di messaggio del commit è **chore(api-deps)** o **feat**
- PATCH: una patch invece viene rilasciata qualora il tipo di messaggio sia **chore(core-deps)**, **fix**, **docs**, **perf** e **revert**

### 6.8.1 Microservizi

Per gestire le versioni dei microservizi non si è fatto uso del modulo precedentemente citato, ma è stato sviluppato un workflow custom in grado di ottenere automaticamente la versione del progetto e di creare una release.

La versione viene gestita all’interno del package.json. Infatti, ogni progetto NodeJS comprende questo file, il quale al suo interno è strutturato come un file JSON e presenta un campo ”version”. Questo campo deve essere aggiornato (manualmente o via comando) per garantire che rimanga aggiornato con la versione che si sta sviluppando.

Durante lo sviluppo su VSCode, ogni qualvolta si aveva la necessità di effettuare un commit sul branch develop, veniva verificata l'entità della modifica effettuata al codice:

- Patch: fix di bugs, compatibile con versioni precedenti;
- Minor: aggiunta di funzionalità, compatibile con versioni precedenti;
- Major: modifica di API non compatibili con versioni precedenti.

Determinata l'entità, grazie al gestore di pacchetti npm è stato possibile cambiare automaticamente la versione semplicemente con un comando nella commandline (*npm version patch/minor/major*).

Una volta effettuato il commit sul branch di release, viene avviato il workflow di release, che automaticamente ottiene la versione dal package.json e ne crea un tag all'interno della repo, andando poi ad associarlo al pacchetto di quella versione.

Il workflow custom sviluppato è composto da 2 parti fondamentali:

- Generazione del tag in base alla versione del package.json. Questo è stato possibile mediante l'utilizzo dello script esterno *Klemensas/action-autotag@stable*;
- Creazione di una release, utilizzando il tag precedentemente creato. Questo è stato possibile mediante l'utilizzo dello script esterno *marvinpinto/action-automatic-releases@latest*.

## 6.9 Documentazione

Durante tutta la fase di sviluppo, è stata scritta (e mantenuta aggiornata) la documentazione relativa alle funzionalità del sistema. La documentazione non prevedeva solamente l'utilizzo di tool esterni per effettuare una descrizione, ma anche dell'utilizzo di commenti per rendere di facile comprensione il codice anche agli altri sviluppatori. Ogni componente del gruppo ha deciso il tool migliore per documentare il proprio codice, in base alle proprie necessità/preferenze.

### 6.9.1 Applicazione smartphone

Per quanto riguarda il codice dell'app in kotlin viene usato il formato KDoc che poi verrà convertito in formato HTML da dokka.

Come mostrato nella sezione di Continuous integration ogni volta che si fa un commit su main viene pubblicata la versione aggiornata della documentazione che viene mostrata sul web tramite github pages ed è contenuta nel branch docs del repository.

### 6.9.2 Raspberry Pi 4

Per quanto riguarda il codice del firmware in Python, si è deciso di sfruttare le docstrings per definire dei commenti in codice per generare in modo automatico la documentazione. A questo scopo è stato utilizzato il modulo *pdoc3*, il quale permette di generare documentazione in formato HTML, da caricare e facilmente consultabile come documentazione di repository, tramite GitHub Pages. Come vedremo in Implementazione, è stato sviluppato un task specifico custom per generare in modo corretto la documentazione e posizionarla nella corretta locazione all'interno del repository. Nella documentazione sono riportati i package, le classi e le porzioni di codice inerenti a ciascun metodo scritto, con una descrizione breve ma esplicativa.

### 6.9.3 Microservizi

Per la descrizione dei microservizi si è adottato Swagger come tool esterno. Grazie a questo tool, è possibile utilizzare un linguaggio simile a Json, che tramite un apposito parser genera una pagina HTML con tutte le informazioni inserite.

La scelta di questo tool è stata motivata dal fatto che permette di effettuare una visualizzazione suddivisa in base alle chiamate API del server. Per ogni chiamata API è stata definito il tipo di messaggio protobuf da utilizzare, oltre che a tutti i tipi possibili di codice errore utilizzati, con il rispettivo messaggio di errore.

Al termine dello sviluppo, si è verificato che la documentazione corrispondesse con l'effettiva implementazione descritta e, una volta confermata, la documentazione è stata pubblicata online sul repository GitHub del rispettivo microservizio.

## 6.10 Licenze

Nel setup dei repository su github si è deciso di utilizzare la licenza MIT, in quanto fornisce agli sviluppatori un'ampia libertà nel modo in cui vogliono utilizzare,

modificare e distribuire il loro software. In particolare, la licenza MIT è una licenza open source che consente la modifica, la distribuzione e l'utilizzo commerciale del software rilasciato con questa licenza.

La licenza MIT definisce alcune condizioni di licenza:

- L'autore del software concede a chiunque la licenza per utilizzare, copiare, modificare e distribuire il software, sia in forma originale che modificata;
- La licenza richiede che l'utente mantenga l'avviso di copyright dell'autore originale in tutte le copie del software;
- Questa licenza esclude qualsiasi forma di garanzia o responsabilità da parte dell'autore;
- La licenza MIT consente agli utenti di sottoporre a licenza il software, incluse le modifiche apportate, sotto una licenza diversa.

Questa licenza va ad apportare numerosi benefici agli sviluppatori del software, quali:

- La licenza MIT offre agli sviluppatori la possibilità di utilizzare, modificare e distribuire il proprio software liberamente;
- Essendo che la licenza MIT permette di modificare e distribuire il software, gli sviluppatori hanno la possibilità di condividere i propri progetti con la community;
- La licenza MIT permette di utilizzare il software per scopi commerciali senza alcuna restrizione;
- La licenza esonera l'autore da qualsiasi forma di garanzia o responsabilità, riducendo il rischio e i potenziali problemi legali per gli sviluppatori.

## 7 Implementazione

Parlare delle tecnologie, librerie, tool, architetture... robe usate che sono state usate durante lo sviluppo. Non parlare di CI/CD nello sviluppo, per quello c'è la sezione precedente. qui si parla più di tecnologie.

Sotto saranno discusse brevemente le fasi di implementazione del progetto con le relative tecnologie adottate per il successo.

Dato che il progetto è fortemente orientato all'utilizzo del cloud, i servizi utilizzati in ambiente Azure Platform sono i seguenti:

- Web Application per realizzare i microservizi di tipo RESTful in Typescript
- Azure Hub IoT come centro di raccolta della digitalizzazione dei vari zaini intelligenti, in modo da avere una controparte digitale (con tutti i benefici che vedremo) di ogni dispositivo
- Database relazionali per avere persistenza di dati statici o poco dinamici quali informazioni sugli utenti, lezioni, materiale necessario
- Database real time su piattaforma Firebase, ovvero un database in JSON non relazionale, basato su approccio child-parent e elementi chiave-valore, per sfruttare i vantaggi di un database remoto per notificare real time i cambiamenti effettuati

L'idea alla base del progetto è quello di realizzare uno zaino intelligente, quindi unito ad un dispositivo programmabile con potenza di calcolo e connettività, fondamentale per sfruttare i servizi in cloud progettati. Un esempio di funzionamento è proposto nella seguente immagine 20.

L'architettura finale risulta la seguente:

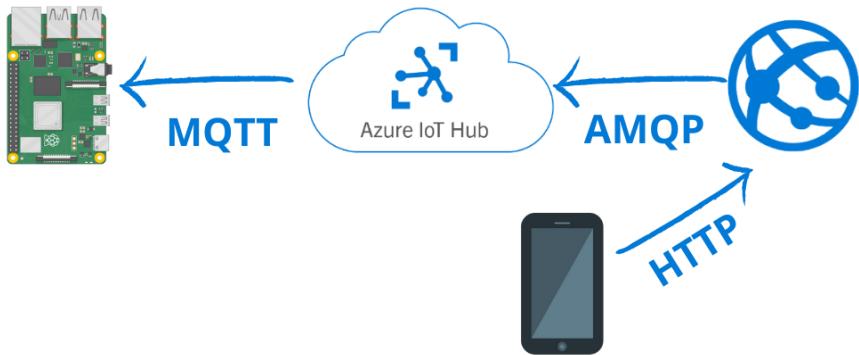


Figura 20: Collegamento delle componenti

## 7.1 Dispositivo e Device Twin

Il dispositivo scelto per il prototipo dello zaino intelligente è il Raspberry Pi 4, in grado di unire leggerezza, potenza di calcolo e connettività insieme, oltre al fatto di poter sfruttare un modulo RC522. Questo'ultimo infatti, è il modulo **RFID-RC522 Funduino** con 8 pin, compatibile con qualsiasi Raspberry Pi P1 Header, in grado di leggere appunto tag di tipo RFID, quelle utilizzate per marcare ogni libro e oggetto. Per quanto riguarda invece il cloud, si è deciso di sfruttare la piattaforma Azure Hub IoT, che consente comunicazioni con sicurezza e affidabilità elevate tra l'applicazione IoT e i dispositivi gestiti da tale applicazione. Tramite questa piattaforma infatti, sono offerti canali di comunicazione MQTT ad hoc per ogni dispositivo IoT connesso, insiem anche ad un piccolo stato persistente per ogni dispositivo chiamato **Device Twin**, molto simile al modello Digital Twin ma più orientato a mantenere solo lo stato e la configurazione del dispositivo, non una rappresentazione digitale completa, che prevede ad esempio anche il punto di vista funzionale. Il Raspberry, infine, è stato programmato in Python, sfruttando, come già accennato, una Build Automation completamente sviluppata in Gradle, ad hoc.

## 7.2 Database

Per quanto riguarda la memorizzazione dei dati in cloud, è stato utilizzato SQL Azure Database, ovvero un database di tipo relazionale, scegliendo il piano tariffario Piano Standard-series (Gen 5). Su questo database in cloud vengono memorizzate le seguenti informazioni:

- Calendario e lezioni;
- Libri, Copie e libri per lezioni;
- Utenti, Ruolo e dispositivi registrati.

Per quanto riguarda invece il contenuto dello zaino, esso è molto più dinamico, rendendo l'utilizzo di un database relazionale classico una scelta non ottimale dal punto di vista delle performance. Neanche il Device Twin è pensato per operare real time, a differenza del Digital Twin. La soluzione è stata trovata in Firebase RealTime Database, che offre un singolo database di tipo chiave-valore (JSON), che permette di notificare l'applicazione mobile in real time ad ogni modifica apportata.

## 7.3 Microservizi

Per lo sviluppo dei microservizi, si è utilizzato il servizio Azure per poter eseguire l'hosting in cloud dei microservizi. Questa scelta è stata favorita sia dalla presenza di un tool chiamato Pipeline in grado di eseguire la CI/CD direttamente con il servizio Azure, sia dalla presenza di credito gratuito usufruibile per il servizio. Per via di problemi con il servizio Pipeline, si è deciso infine di utilizzare i workflow di GitHub.

I microservizi sono stati sviluppati con NodeJS, mediante l'utilizzo del linguaggio di programmazione Typescript (in quanto in grado di semplificare la comunicazione con altri applicativi che utilizzano dati tipizzati). Si è deciso di sviluppare il servizio mediante l'utilizzo della libreria Express, che permette la creazione di servizi REST che rispondono a chiamate HTTP. Per la comunicazione si è utilizzato la tecnologia Protobuf di Google per far sì che lo scambio di messaggi tra le applicazioni avvenisse in modo standardizzato senza la necessità di effettuare una serializzazione/deserializzazione dei dati inviati (l'utilizzo di Typescript ha aiuta-

to nello sviluppo, in quanto in grado di gestire direttamente i dati tipizzati che venivano ricevuti).

Per effettuare il test dei microservizi si è utilizzato la libreria Jest, in quanto permette di testare codice JS/TypeScript. In particolare, l'utilizzo di essa insieme alla libreria Supertest ha permesso di testare direttamente le chiamate HTTP del microservizio, come avverrebbe nel caso di effettivo utilizzo.

L'utilizzo della CI/CD è stato sviluppato tramite workflow su GitHub, in questo modo si avrà il tool per il versioning che gestisce anche il testing e il deploy dell'applicativo. In particolare, si è utilizzato il workflow già definito da Azure per il deploy automatico sulla piattaforma (presente anche di testing prima del deploy, per verificare la correttezza della release) ed è stato anche utilizzato un workflow custom per verificare l'effettivo testing durante il development della soluzione (successivamente disabilitato per problemi di costi con la piattaforma Azure). Inoltre, è stato sviluppato anche un workflow custom per la release automatica di nuovi package all'interno della repository.

## 7.4 Python + Gradle

Per quanto riguarda Python è stato utilizzato un plugin ([link](#)) di terze parti che gestisce le dipendenze per un progetto Python in Gradle, esteso poi con il plugin (sempre per Gradle) personale sviluppato ad Hoc e mostrato precedentemente, per gestire in modo totalmente automatico l'esecuzione di test e verifica della coverage in un progetto Python e Gradle. Questa cosa di sfruttare la build automation offerta da Gradle e un linguaggio ad oggetti versatile come Python, ha permesso di velocizzare e semplificare tutto il processo di sviluppo, ponendo l'attenzione maggiormente sul design del sistema.

## 7.5 Applicazione

Per realizzare l'applicazione si usa lo sviluppo in nativo, usando in kotlin, in particolare per android sdk 29 o superiore. Per la grafica viene usato la libreria Jetpack Compose che rende l'interfaccia grafica più personalizzabile e, essendo dichiarativa, più semplice da usare. Per la gestione dei flussi viene usato il concetto di coroutine fornito da kotlin tramite libreria e che si integra perfettamente con l'architettura MVVM e con i cicli di vita dei componenti android. Per gestire l'implementazione

del database viene usata la libreria Room che permette di creare un'istanza SQLite sul dispositivo android e di usare delle classi kotlin per creare il modello. Per la connessione al sistema di Realtime database viene usata la libreria di firebase, che permette di ottenere un evento quando si effettua una modifica. Un altro componente utile sempre di firebase è crashlytics che permette di avere una segnalazione quando si verifica un problema nell'applicazione su un dispositivo e permette di avere anche lo stack di chiamata, utile quando si ha a che fare con diverse persone con diversi dispositivi e senza contratto diretto.

## 8 Timeline di lavoro

Lo sviluppo del progetto è iniziato con una prima fase estensiva di analisi dei requisiti, definizione del linguaggio, interviste con il personale, definizione dell'ubiquitous language... andando così ad ottenere tutte le informazioni sul dominio e sul linguaggio necessarie.

Successivamente alla fase di analisi iniziale, sulla base dei requisiti ottenuti, si è andato ad analizzare possibili soluzioni (a livello architettonico), per poter definire quale soluzione sia più corretta per la risoluzione del problema, rispettando tutti i vincoli imposti dal dominio. Oltre all'utilizzo di criteri importanti quali complessità, manutenibilità, scalabilità per la decisione dell'architettura finale, sono stati impiegati anche mockup per la visualizzazione della UI nell'applicazione mobile, poi rifiniti per garantire il requisito di una interfaccia semplice.

Una volta definita anche l'architettura, è stato avviato lo sviluppo del progetto seguendo tutte le informazioni ottenute durante le fasi di analisi precedenti, per rispettare il dominio del progetto.

Al termine dello sviluppo, è stata verificata la documentazione del progetto, per far sì che corrispondesse con le effettive implementazioni nel codice, per far sì che future manutenzioni/ampliamenti del codice siano facilitati e risultino in una riduzione del tempo richiesto.

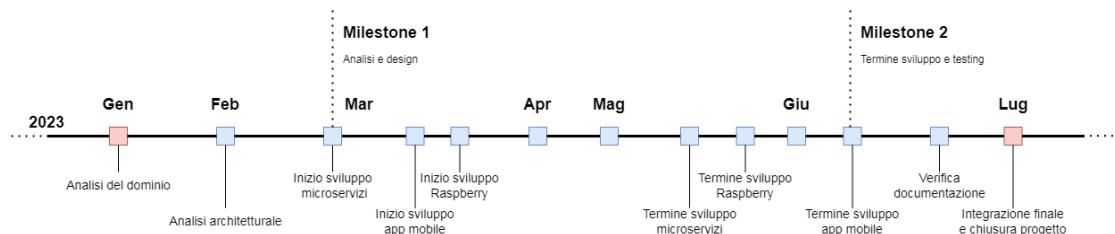


Figura 21: Timeline del progetto.

## 9 Conclusioni

Si voleva realizzare un sistema che fornisse all’utente le funzionalità di uno zaino smart, facendo sì che l’utente potesse tenere traccia di tutti i libri/oggetti delle lezioni e di poterli inserire nello zaino, facendo sì che il sistema lo notificasse nel caso qualche cosa non era stata inserita.

Per realizzare questo obiettivo si è partiti con un’analisi sul dominio del problema, andando ad identificare i requisiti del sistema e l’ubiquitous language da adottare durante tutte le fasi dello sviluppo e documentazione. Una volta identificato il dominio, si è passata all’analisi dei sottodomini, andando ad identificare il componente hardware/embedded (raspberry pi), l’applicazione smartphone e il sistema di backend.

Al termine della fase di analisi, si è passati alla fase di progettazione e implementazione delle diverse componenti del progetto, gestite separatamente dai vari membri del team. Una volta terminato lo sviluppo, sono stati verificati con successo che i requisiti richiesti sono stati rispettati e verificati sul prodotto finale.

Si ritiene che la realizzazione del progetto abbia contribuito a migliorare le competenze e professionalità, in quanto:

- Ha consentito, per la prima volta, di lavorare con la strategia Domain Driven Design;
- Ha permesso di approfondire la strategia di DevOps e alcune tecnologie di interesse, utilizzate dai componenti del gruppo;
- Ha migliorato la coordinazione e comunicazione all’interno del team.

### 9.1 Sviluppi futuri

Al termine del progetto sono stati individuati alcuni aspetti che potrebbero essere migliorati per poter rendere ancora migliore il servizio (o che sono stati semplificati per via del tempo/budget a disposizione).

1. Inserimento ed utilizzo di un modulo GSM per avere connettività mobile indipendente e un modulo GPS per ottenere la posizione geolocalizzata del dispositivo;

2. Implementazione di un Reminder Engine, in grado di poter notificare l'utente oggetti non inseriti necessari alla lezione/evento del giorno successivo, in totale autonomia;
3. Implementazione di un sistema di aggiornamento automatico di tutti i dispositivi Raspberry nel momento di rilascio di una nuova versione del firmware;
4. Inserimento di un salt value negli utenti, per rendere ancora più efficace l'hash delle password e la sicurezza degli utenti (in questo modo, l'hash diventa univoco, facendo sì che anche 2 password identiche hanno un valore di hash differente).