

Documentazione progetto "Smart calendar for business"

Filippo Lazzari [1047143],
Mauro Riva [1053644],
Marco Rodolfi [1040347]

A.A. 2021/2022

Indice

1	Iterazione 0	5
1.1	Introduzione e utilizzo del sistema	5
1.1.1	Il server dell'applicativo	5
1.1.2	L'applicativo Android	5
1.1.3	L'algoritmo	5
1.2	Requisiti funzionali e casi d'uso	6
1.3	Requisiti non funzionali	8
1.3.1	Usabilità	8
1.3.2	Manutenibilità	8
1.3.3	Efficienza	8
1.4	Topologia	9
1.5	Toolchain utilizzati	10
2	Iterazione 1	11
2.1	Introduzione	11
2.2	UC1: Registrazione nuova azienda	12
2.3	UC2: Login manager	13
2.4	UC3: Logout	13
2.5	UC4: Gestione ruoli	14
2.5.1	UC 4.1: Inserimento nuovi ruoli	14
2.5.2	UC4.2: Modifica ruoli esistenti	15
2.5.3	UC4.3: Cancellazione ruolo	16
2.6	UC5: Gestione sedi	17
2.6.1	UC5.1: Inserimento nuove sedi	17
2.6.2	UC5.2: Modifica sedi esistenti	18
2.6.3	UC5.3: Cancellazione sede	19
2.7	UC6: Gestione dipendenti	20
2.7.1	UC6.1: Inserimento nuovi dipendenti	20
2.7.2	UC6.2: Modifica dati dipendenti	21
2.7.3	UC6.3: Cancellazione dipendenti	21
2.8	UC7: Algoritmo allocazione dipendenti su turni	23
2.9	Algoritmo	24
2.9.1	Inizializzazione	25
2.9.2	Cancellazione vecchi turni dello stesso mese e creazione dei nuovi	26
2.9.3	Allocazione dei dipendenti	29
2.9.4	Allocazione dei turni rimasti scoperti tramite straordinari e trasferte	33
2.10	Entity-Relationship Diagram	37

2.11	UML Class Diagram	38
2.12	UML Component Diagram	39
2.13	UML Deployment Diagram	40
2.14	Documentazione API	41
2.14.1	/registrazione	41
2.14.2	/login	42
2.14.3	/logoutPagina	42
2.14.4	/indirizzoSedi	42
2.14.5	/nomeRuoli	43
2.14.6	/checksession	43
2.14.7	/aggiungiSede	43
2.14.8	/checkCalcoloTurni	44
2.14.9	/getStatusGiorni	44
2.14.10	/ricalcolaTurni	44
2.14.11	/getTurniSedeGiorno	45
2.15	Testing	46
2.15.1	Test dell'API del database	46
2.15.2	Analisi dinamica	48
3	Iterazione 2	50
3.1	Introduzione	50
3.2	UC8: Visualizzazione Calendario	50
3.2.1	UC8.1 Visualizzazione Turni Dipendente	50
3.2.2	UC8.2 Visualizzazione Turni Manager	50
3.3	UC9: Login e logout dipendenti	52
3.4	UC10: Inserimento malattia	53
3.5	UC11: Inserimento ferie	54
3.6	UC12: Valutazione richieste di ferie	55
3.7	UML Component Diagram	56
3.8	UML Deployment Diagram	57
3.9	Documentazione API	58
3.9.1	Introduzione	58
3.9.2	API per login	58
3.9.3	API per logout	58
3.9.4	API per get turni	59
3.9.5	API per richiesta malattia	59
3.9.6	Testing App Android	60

4	Guida per l'uso dell'applicazione	61
4.1	Applicazione web - Manager	61
4.2	Applicazione Android - Dipendenti	68

Elenco delle figure

1	Casi d'uso pensati	6
2	Topologia	9
3	Diagramma Entity-Relationship	37
4	Diagramma delle classi	38
5	Diagramma delle componenti	39
6	Deployment diagram	40
7	Diagramma delle componenti	56
8	Deployment diagram	57
9	Finestra mostrata all'apertura dell'applicazione web	61
10	Finestra login	61
11	Finestra registrazione 1	62
12	Finestra registrazione 2	62
13	Finestra registrazione 3	63
14	Finestra registrazione 4	63
15	Finestra registrazione 5	64
16	Finestra registrazione 6	65
17	Inserimento di una nuova sede	66
18	Visualizzazione calendario sede	66
19	Visualizzazione turni di un giorno	67
20	Visualizzazione dipendenti per turno	67
21	Pagina iniziale, login	68
22	Pagina iniziale dopo aver effettuato il login	69
23	Visualizzazione dei turni	70
24	Menu per scelta funzione	71
25	Calendario per la selezione delle date	72
26	Messaggio di accettazione della richiesta di malattia	72

Elenco delle tabelle

1	Alta priorità	7
2	Media priorità	7
3	Bassa priorità	8

1 Iterazione 0

1.1 Introduzione e utilizzo del sistema

Il progetto si pone l'obiettivo di organizzare i turni di un'azienda, specialmente per quelle attività che devono gestire turni a rotazione e che devono essere sempre presenti per il servizio clienti, come i servizi di ristorazione, per esempio. Nel progetto non è prevista la gestione di attività molto flessibili, tipicamente coordinate direttamente dai project manager.

L'idea è di realizzare tre componenti distinte all'interno dell'applicativo:

1. Una parte lato server che gestirà i client tramite chiamate REST e farà girare l'algoritmo per ottimizzare l'allocazione dei dipendenti sui turni.
2. Una parte web che gestisca la registrazione dell'azienda al servizio e fornisca un'interfaccia per il manager.
3. Un'applicativo per dispositivi Android che permetta ai dipendenti dell'azienda di visualizzare i propri turni e inserire malattie/ferie.

1.1.1 Il server dell'applicativo

Il server deve gestire più di una singola azienda e verrà amministrato direttamente da chi gestisce il servizio. Mette inoltre a disposizione le interfacce per consentire agli utenti di modificare i dati e visualizzare lo stato dell'azienda.

1.1.2 L'applicativo Android

Per permettere una maggiore flessibilità di gestione da parte dei dipendenti che lavorano, è stato pensato di creare un'applicativo Android che permetta di gestire il proprio stato all'interno dell'azienda, richiedere malattie e verificare i propri straordinari. La comunicazione verso il server è quindi gestita tramite chiamate REST.

1.1.3 L'algoritmo

L'algoritmo deve allocare dinamicamente i turni e gestire quelli scoperti verificando se dipendenti della stessa sede possono coprire il turno con degli straordinari oppure se personale di altre sedi della stessa azienda che possono svolgere il ruolo ha la possibilità di effettuare la trasferta.

1.2 Requisiti funzionali e casi d'uso

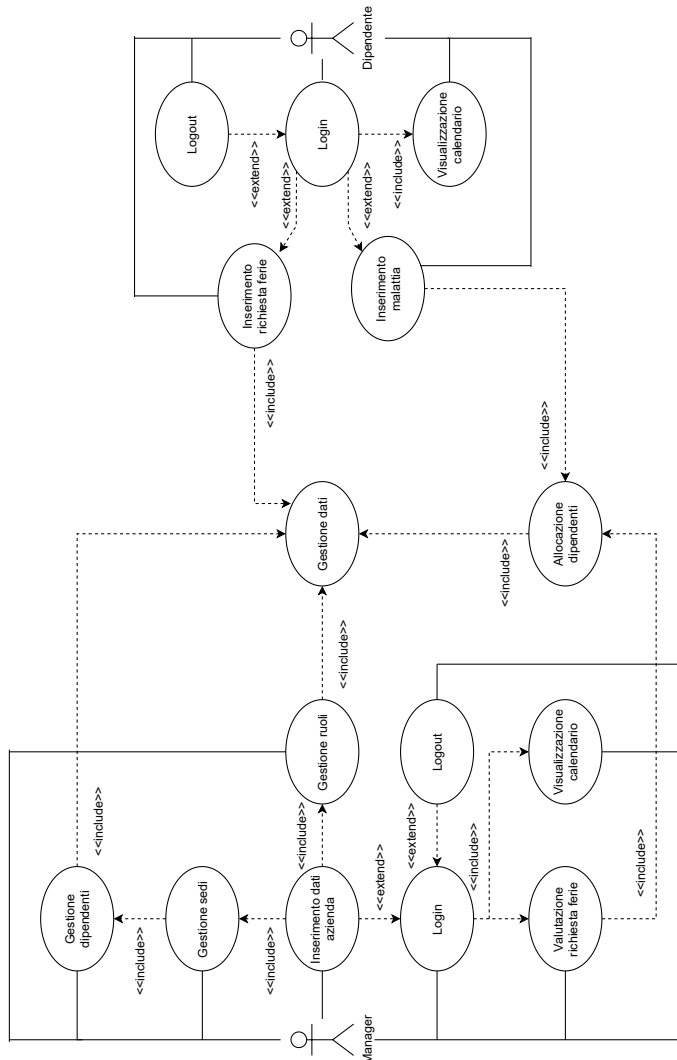


Figura 1: Casi d'uso pensati

La seguente tabella rappresenta i casi d'uso iniziali che sono stati ritenuti importanti da includere già all'inizio del progetto:

IDENTIFICATIVO	NOME FUNZIONE
UC1	Inserimento dati azienda
UC2	Login
UC3	Logout
UC4	Gestione ruoli
UC5	Gestione sedi
UC6	Gestione dipendenti
UC7	Algoritmo di allocazione dei dipendenti

Tabella 1: Alta priorità

Questa seconda tabella invece rappresenta i casi d'uso con priorità minore, cioè quelli da cui non dipendono i casi d'uso fondamentali.

IDENTIFICATIVO	NOME FUNZIONE
UC8	Visualizzazione calendario
UC9	Login e logout dipendenti
UC10	Inserimento malattia
UC11	Inserimento richiesta di ferie
UC12	Valutazione richiesta di ferie

Tabella 2: Media priorità

Infine questi sono casi d'uso che possono essere valutati in versioni future dell'applicazione:

IDENTIFICATIVO	NOME FUNZIONE
UC13	Personalizzazione calendario
UC14	Visualizzazione avanzata della situazione dei dipendenti
UC15	Limitazioni nell'inserimento delle ferie
UC16	Modifica credenziali di accesso

Tabella 3: Bassa priorità

1.3 Requisiti non funzionali

1.3.1 Usabilità

Per facilitare l'usabilità dell'applicazione è stato deciso di sviluppare un'app Android per i dipendenti e una web app per il manager dell'azienda. Entrambe le tipologie di client si connetteranno ad un server remoto.

1.3.2 Manutenibilità

Sono stati implementati nel codice diversi espedienti per facilitare una futura modifica dello stesso. Per esempio tramite attributi statici impiegati per rappresentare vari parametri sparsi per l'algoritmo oppure funzioni al momento molto simili tra di loro che non sono accorpate in vista di eventuali cambiamenti nelle prossime versioni.

1.3.3 Efficienza

L'efficienza dell'allocazione dei dipendenti sui turni è uno degli obiettivi del progetto. Anche l'efficienza dell'algoritmo è stata presa in considerazione ma nello stato attuale del progetto è di importanza secondaria. La progettazione dell'algoritmo ha comunque tenuto in considerazione la manutenibilità anche in ottica di miglioramenti di efficienza.

1.4 Topologia

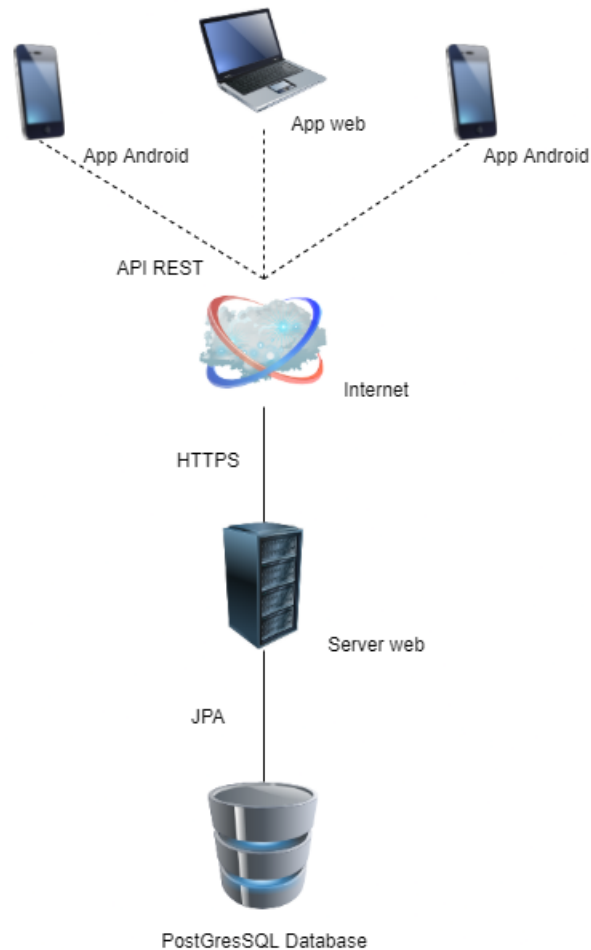


Figura 2: Topologia

La topologia dell'applicazione si divide in tre livelli:

- Il front-end che espone l'interfaccia grafica all'utente e acquisisce i dati in input. Fanno parte di questo livello l'app Android per i dipendenti, accessibile da smartphone e tablet, e l'app web per i manager, accessibile da qualsiasi browser.
- Il server web che espone le interfacce ai client consentendo l'interazione degli stessi con i dati memorizzati in remoto.
- Il database che si occupa di fornire la struttura dati persistente e garantire la consistenza e l'integrità dei dati.

1.5 Toolchain utilizzati

NOME TOOLCHAIN	UTILIZZO
IntelliJ IDEA Ultimate	Utilizzato come IDE di sviluppo per backend.
Android Studio	Utilizzato per lo sviluppo dell'applicativo Android.
Git & Gitlab	Software e piattaforma per gestire la distribuzione del codice sorgente.
CoCalc	Piattaforma utilizzata per la scrittura collaborativa della documentazione \LaTeX .
Java 17	Linguaggio di programmazione usato per il server e l'applicativo Android.
jQuery	Libreria JavaScript per applicazioni web.
Bootstrap	Framework CSS per lo sviluppo di pagine web.
Draw.io	Piattaforma utilizzata per creare diagrammi UML.
Postman	Software per il testing dinamico e la documentazione delle API.
Spring Boot	Il framework utilizzato lato server per il suo funzionamento.
Spring Data JPA	Interfaccia per l'astrazione della base di dati.
JGraphT	Libreria per i grafi utilizzata per lo sviluppo dell'algoritmo.
PostgreSQL	Il DB scelto per gestire la base di dati stessa.
H2	Il DB scelto per l'unit testing dell'applicativo.

2 Iterazione 1

2.1 Introduzione

Nella prima interazione sono stati previsti i seguenti casi d'uso:

- UC1: Registrazione nuova azienda
- UC2: Login manager
- UC3: Logout manager
- UC4: Gestione ruoli
 - UC4.1: Inserimento nuovi ruoli
 - UC4.2: Modifica ruoli esistenti
 - UC4.3: Cancellazione ruolo
- UC5: Gestione sedi
 - UC5.1: Inserimento nuove sedi
 - UC5.2: Modifica sedi esistenti
 - UC5.3: Cancellazione sede
- UC6: Gestione dipendenti
 - UC6.1: Inserimento nuovi dipendenti
 - UC6.2: Modifica dati dipendenti
 - UC6.3: Cancellazione dipendente
- UC7: Algoritmo allocazione dipendenti su turni

2.2 UC1: Registrazione nuova azienda

Descrizione: Il manager dell'azienda inserisce le proprie credenziali registrando l'account con il quale andrà a gestire l'azienda. Per registrare l'azienda è richiesto l'inserimento della sede centrale, dei ruoli dei dipendenti, dell'orario di apertura e dei dipendenti della sede centrale.

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Click del pulsante *registrati* da parte del manager

Postcondizione: Se la registrazione è avvenuta con successo, ossia se i dati sono stati caricati nel database, la pagina viene ricaricata e il manager può procedere con il login

Procedimento:

1. Il manager inserisce l'email, la password, il nome e il cognome
2. Dati azienda e sede centrale
3. Inserimento dei ruoli (almeno un ruolo)
4. Inserimento dei dipendenti della sede centrale (almeno un dipendente), un dipendente può avere al massimo 3 ruoli ed essere part-time
5. Selezione orario settimanale azienda

2.3 UC2: Login manager

Descrizione: Questa funzione è utilizzata per il login del manager all'interno dell'applicativo web. Per effettuare il login è necessario fornire email e password. Il server gestisce il login mediante la creazione di una sessione http.

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Click del pulsante *login* da parte del manager dall'interfaccia web

Postcondizione: Se il manager ha inserito le credenziali corrette nel form di login accederà alla pagina principale dell'applicazione

Procedimento:

1. Click dell'utente sul tasto *login*
2. Inserimento email e password
3. Se le credenziali sono corrette il server crea la sessione http
4. Viene comunicato l'esito dell'operazione all'utente

2.4 UC3: Logout

Descrizione: Questa funzione è utilizzata per effettuare il logout del manager dalla pagina web. Il server distrugge la sessione http.

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Click sul pulsante *logout*

Postcondizione: Sessione http distrutta

Procedimento:

1. Click dell'utente sul tasto *logout*
2. Distruzione della sessione
3. Aggiornamento della pagina

2.5 UC4: Gestione ruoli

Descrizione: L'applicazione deve permettere al manager di inserire, modificare ed eliminare i ruoli dal database. Un ruolo è caratterizzato dal nome e dal numero minimo di dipendenti necessari.

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Selezione da parte del manager della voce nel menù *Ruoli*

Postcondizione: Un alert segnala all'utente l'avvenuta modifica del database

2.5.1 UC 4.1: Inserimento nuovi ruoli

Descrizione: Il manager inserisce il nome del ruolo e il numero richiesto di dipendenti.

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Nella pagina *gestione ruoli* click su *aggiungi*

Postcondizione: Se il nuovo ruolo è stato inserito correttamente nel database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Ruoli* dal menu
3. Selezione della modalità *aggiunta*
4. Inserimento del nome del ruolo e del numero minimo di dipendenti
5. Click su *conferma*
6. Segnalazione del risultato dell'operazione di inserimento

2.5.2 UC4.2: Modifica ruoli esistenti

Descrizione: Il manager seleziona uno dei ruoli già presenti nel database e modifica il nome o il numero dei dipendenti. Il nuovo nome non deve causare conflitti con quelli degli altri ruoli dell'azienda. La variazione del numero minimo di dipendenti causa l'invalidazione di tutti i turni dell'azienda calcolati per date future.

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Nella pagina *gestione ruoli* click su *modifica*

Postcondizione: Se il ruolo è stato modificato correttamente nel database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Ruoli* dal menu
3. Selezione della modalità *modifica*
4. Scelta del ruolo da modificare
5. Inserimento del nome del ruolo e del numero minimo di dipendenti
6. Click su *conferma*
7. Segnalazione del risultato dell'operazione di modifica

2.5.3 UC4.3: Cancellazione ruolo

Descrizione: Il manager seleziona uno dei ruoli già presenti nel database e lo elimina. Questa operazione deve tenere in considerazione le relazioni su database dei ruoli con i turni e con i dipendenti.

Attori coinvolti: Manager, Dipendenti, Sistema

Condizione di attivazione: Nella pagina *gestione ruoli* click su *cancella*

Postcondizione: Se il ruolo è stato eliminato correttamente dal database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Ruoli* dal menu
3. Selezione della modalità *cancella*
4. Scelta del ruolo da eliminare
5. Click su *conferma*
6. Segnalazione del risultato dell'operazione di cancellazione

2.6 UC5: Gestione sedi

Descrizione: L'applicazione deve permettere al manager di inserire, modificare ed eliminare le sedi dell'azienda dal database. Una sede è identificata dal proprio indirizzo e ha dei dipendenti assegnati (almeno uno). Una sede può inoltre avere indicata la sede a lei più vicina e la relativa distanza.

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Selezione da parte del manager della voce nel menu *Sedi*

Postcondizione: Un alert segnala all'utente l'avvenuta modifica del database

2.6.1 UC5.1: Inserimento nuove sedi

Descrizione: Il manager inserisce l'indirizzo della nuova sede ed eventualmente seleziona la sede più vicina. La distanza tra le sedi è espressa in chilometri. Alla sede sono associati uno o più dipendenti

Attori coinvolti: Manager, Sistema

Condizione di attivazione: Nella pagina *gestione sedi* click su *aggiungi*

Postcondizione: Se la nuova sede è stata inserita correttamente nel database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Sedi* dal menu
3. Selezione della modalità *aggiungi*
4. Inserimento dell'indirizzo ed eventualmente della sede vicina con la distanza tra le due
5. Inserimento dei dipendenti con nome, cognome, ruoli ed eventualmente se il contratto è di tipo part-time
6. Click su *conferma*
7. Segnalazione del risultato dell'operazione di inserimento

2.6.2 UC5.2: Modifica sedi esistenti

Descrizione: Il manager seleziona una delle sedi già presenti nel database e ne modifica l'indirizzo o la sede più vicina. La variazione della sede vicina comporta l'invalidazione dei turni in trasferta calcolati per date future.

Attori coinvolti: Manager, Dipendenti, Sistema

Condizione di attivazione: Nella pagina *gestione sedi* click su *modifica*

Postcondizione: Se la sede è stata modificata correttamente nel database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti.

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Sedi* dal menu
3. Selezione della modalità *modifica*
4. Scelta della sede da modificare
5. Selezione di una nuova sede vicina (eventualmente anche nessuna) e inserimento della distanza
6. Click su *conferma*
7. Segnalazione del risultato dell'operazione di modifica

2.6.3 UC5.3: Cancellazione sede

Descrizione: Il manager seleziona una delle sedi già presenti nel database e la elimina. Questa operazione deve tenere conto di eventuali trasferte pianificate per date future.

Attori coinvolti: Manager, Dipendenti, Sistema

Condizione di attivazione: Nella pagina *gestione sedi* click su *cancella*

Postcondizione: Se la sede è stata eliminata correttamente dal database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Sedi* dal menu
3. Selezione della modalità *cancella*
4. Scelta della sede da eliminare
5. Click su *conferma*
6. Segnalazione del risultato dell'operazione di cancellazione

2.7 UC6: Gestione dipendenti

Descrizione: L'applicazione deve permettere al manager di inserire, modificare ed eliminare i dipendenti dislocati in tutte le sedi dell'azienda dal database. Un dipendente è identificato dalla propria email e può svolgere da uno fino al massimo di tre ruoli diversi.

Attori coinvolti: Manager, Dipendenti, Sistema

Condizione di attivazione: Selezione da parte del manager della voce nel menu *Dipendenti*

Postcondizione: Un alert segnala all'utente l'avvenuta modifica del database

2.7.1 UC6.1: Inserimento nuovi dipendenti

Descrizione: Il manager inserisce il nome, il cognome, i ruoli e il tipo di contratto del dipendente (se full-time o part-time) e seleziona la sede a cui assegnarlo. L'email viene generata automaticamente dal server nel formato nome.cognome (ed eventualmente un numero, in caso di omonimie) seguito dal suffisso aziendale. La password è creata dal server e comunicata tramite l'email all'utente che dovrà poi provvedere a modificarla.

Attori coinvolti: Manager, Dipendente, Sistema

Condizione di attivazione: Nella pagina *gestione dipendenti* click su *aggiungi*

Postcondizione: Se il nuovo dipendente è stato inserito correttamente nel database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Dipendenti* dal menu
3. Selezione della modalità *aggiunta*
4. Inserimento del nome, del cognome, dei ruoli e della modalità del contratto
5. Click su *conferma*
6. Segnalazione del risultato dell'operazione di inserimento

2.7.2 UC6.2: Modifica dati dipendenti

Descrizione: Il manager seleziona un dipendente presente nel database e ne modifica il tipo di contratto o i ruoli che può svolgere. La variazione di questi parametri comporta l'invalidazione dei turni associati all'utente in date future.

Attori coinvolti: Manager, Dipendente, Sistema

Condizione di attivazione: Nella pagina *gestione dipendenti* click su *modifica*

Postcondizione: Se i dati del dipendente sono stati modificati correttamente sul database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Dipendenti* dal menu
3. Selezione della modalità *modifica*
4. Scelta della sede per visualizzarne tutti i dipendenti o ricerca diretta tramite email
5. Modifica dei ruoli e del contratto
6. Click su *conferma*
7. Segnalazione del risultato dell'operazione di modifica

2.7.3 UC6.3: Cancellazione dipendenti

Descrizione: Il manager seleziona un dipendente presente nel database e lo elimina. Questa operazione invalida i turni associati all'utente in date future

Attori coinvolti: Manager, Dipendente, Sistema

Condizione di attivazione: Nella pagina *gestione dipendenti* click su *cancella*

Postcondizione: Se il dipendente è stato eliminato correttamente dal database comunica all'utente che l'operazione è avvenuta con successo, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Dipendenti* dal menu
3. Selezione della modalità *cancella*
4. Scelta della sede per visualizzarne tutti i dipendenti o ricerca diretta tramite email
5. Click su *conferma*
6. Segnalazione del risultato dell'operazione di cancellazione

2.8 UC7: Algoritmo allocazione dipendenti su turni

Descrizione: L'obiettivo dell'algoritmo è allocare i dipendenti ai turni in modo da non lasciare turni scoperti. Il calcolo avviene sull'intera azienda per un determinato mese. Se il mese dato in input è quello attuale verranno calcolati solo i turni dalla data odierna fino alla fine del mese, non è possibile modificare i turni nel passato.

Attori coinvolti: Manager, Dipendente, Sistema

Condizione di attivazione: Nella pagina *calendario* selezionare il mese e l'anno e click su *ricalcola*

Postcondizione: Se i turni sono stati calcolati correttamente comunica all'utente se tutti i turni sono stati assegnati o se ne è rimasto almeno uno scoperto, errore altrimenti

Procedimento:

1. L'utente ha effettuato il login
2. Accesso alla pagina *Calendario* dal menu
3. Selezione del mese e dell'anno
4. Click su *ricalcola*
5. Segnalazione del risultato dell'esecuzione dell'algoritmo

2.9 Algoritmo

L'algoritmo alloca i dipendenti ai turni di tutta l'azienda, per il mese dato in input dal manager. L'azienda è caratterizzata dall'orario di apertura, da i ruoli e dalle sedi. L'algoritmo è suddiviso in diverse sezioni:

1. Inizializzazione
2. Cancellazione vecchi turni dello stesso mese e creazione dei nuovi
3. Allocazione dei dipendenti
4. Allocazione dei turni rimasti scoperti tramite straordinari e trasferte

2.9.1 Inizializzazione

In questa prima fase sono inizializzate le variabili ed è calcolato il periodo temporale di riferimento: se il mese su cui l'algoritmo dovrà operare è nel futuro allora la data di inizio è il giorno uno e la data di fine è l'ultimo giorno di quel mese. Se invece il mese è quello attuale la data di inizio è il giorno corrente, così che i turni dei giorni precedenti non vengano modificati. Per ogni dipendente dell'azienda sono inizializzate a zero due `HashMap` che mettono in relazione con i dipendenti le ore lavorate quel mese e le ore di straordinario effettuate.

Inizializzazione

```
46 public boolean calculateBestTurni(Azienda az, int anno, int mese){
47     List<Sede> sedi = sedeRep.findSedeByAzienda(az);
48     List<Ruoli> ruoli = ruoliRep.getRuoliByRuoloID_Azienda(az);
49     Orario orario = orarioRep.findOrarioByAzienda(az);
50     boolean assegnamentoCompleto = true;
51
52     HashMap<Sede, List<Turno>> turniPerSede = new HashMap<>();
53     HashMap<Utente, Integer> utentiOreMensili = new HashMap<>();
54     HashMap<Utente, Integer> utentiOreStraordinario = new
55         ↳ HashMap<>();
56
57     //Inizializzo le ore lavorate questo mese
58     List<Utente> uteList = utRep.findUtenteByAzienda(az);
59     for(Utente u : uteList){
60         utentiOreMensili.put(u, 0);
61         utentiOreStraordinario.put(u, 0);
62     }
63
64     //Calcolo periodo temporale
65     LocalDate startTime = LocalDate.of(anno, mese, 1);
66     //Se devo calcolare solo la parte restante del mese attuale
67     ↳ (non posso modificare il passato!)
68     if(startTime.isBefore(LocalDateTime.now().toLocalDate()))
69         startTime = LocalDate.now();
70
71     LocalDate endTime = startTime.plusMonths(1);
72     endTime = endTime.withDayOfMonth(1);
```

Complessità $O(U)$, con U numero di dipendenti dell'azienda.

2.9.2 Cancellazione vecchi turni dello stesso mese e creazione dei nuovi

Nella seconda fase è necessario rimuovere dal database tutti i turni e gli straordinari che sono stati calcolati precedentemente per quel mese, a partire dalla data di inizio. Fatto ciò si può procedere a ricreare turni "vuoti", senza quindi dipendenti allocati: per ogni giorno compreso tra la data di inizio e la data di fine l'orario di apertura è suddiviso in turni da quattro ore (o meno, se le ore di apertura non sono un multiplo di quattro) per ogni ruolo presente nell'azienda. Ai turni sono assegnate un numero di ore scoperte pari alla durata del turno moltiplicata per il numero minimo di dipendenti necessari per quel ruolo. Quindi su un turno di quattro ore per tre dipendenti le ore da allocare saranno dodici. L'ultima operazione di questa fase è la ricerca nel database di eventuali turni di questo mese precedenti alla data di inizio e l'aggiornamento dei contatori delle ore lavorate e degli straordinari dei dipendenti.

Cancellazione vecchi turni dello stesso mese e creazione dei nuovi

```
73 //Per ogni sede, recupero tutti i turni disponibili.
74 for (Sede sede : sedi) {
75     List <Turno> lista =
76         ↳ turnoRep.findTurnoBySedeAndDataGreaterThanAndDataLessThan
77         ↳ OrderByDataDesc(sede,startTime.minusDays(1),
78         ↳ endTime);
79     //Rimuovo eventuali turni (con annessi straordinari) già
80     ↳ presenti nel database
81     for(Turno t : lista){
82         if((t.getData().isBefore(ChronoLocalDate.from(endTime)) &&
83         ↳ t.getData().isAfter(ChronoLocalDate.from(startTime))) ||
84         ↳ t.getData().isEqual(ChronoLocalDate.from(startTime))){
85             turnoUtenteRep.deleteByTurno(t);
86             straRep.deleteByTurno(t);
87             turnoRep.delete(t);
88         }
89         else if((t.getData().isBefore(ChronoLocalDate.from(end
90         ↳ Time))))
91             break;
92     }
93     //Creo i turni a cui dovranno essere assegnati i dipendenti
94     List <Turno> listaTurni = new ArrayList<>();
95     for(Ruoli r : ruoli) {
96         if (!r.getAdmin()){
```

```

90         LocalDate time = LocalDate.from(startTime);
91         while (time.isBefore(endTime)) {
92             if (orario.isOpenForCurrentDay(time)) {
93                 Giorno giorno =
94                     ↳ orario.getCorrespondingWorkingHours(time);
95                 LocalDate data = LocalDate.from(time);
96                 int oreAperturaRimaste = giorno.getChiusura() -
97                     ↳ giorno.getApertura();
98                 int apertura = giorno.getApertura();
99                 int chiusura;
100                 //FIXME: Eventualmente da aggiungere la pausa
101                 ↳ pranzo
102                 while (oreAperturaRimaste > 0) {
103                     if (oreAperturaRimaste < durataTurno)
104                         chiusura = apertura +
105                             ↳ oreAperturaRimaste;
106                     else
107                         chiusura = apertura + durataTurno;
108                     Turno turno = new Turno(sede,
109                         ↳ r.getRuoloID(), data, apertura,
110                         ↳ chiusura);
111                     //Definisco le ore-dipendente necessarie per
112                     ↳ coprire il turno
113                     turno.setOreScoperte((chiusura - apertura) *
114                         ↳ r.getMinRuolo());
115                     listaTurni.add(turno);
116                     apertura = chiusura;
117                     oreAperturaRimaste -= durataTurno;
118                 }
119             }
120             time = time.plusDays(1);
121         }
122     }
123 }
124
125 //E li salvo in una coppia sede:turni.
126 turniPerSede.put(sede, listaTurni);
127
128 //Cerco le ore già effettuate dai dipendenti questo mese
129 LocalDate st = LocalDate.of(startTime.getYear(),
130     ↳ startTime.getMonth(), 1);

```

```

121 List <Turno> listaTurniQuestoMese =
    ↳ turnoRep.findTurnoBySedeAndDataGreaterThanAndDataLessThan
    ↳ OrderByDataDesc(sede,st.minusDays(1),
    ↳ endTime);
122 for(Turno t : listaTurniQuestoMese){
123     if(t.getData().getMonthValue() == st.getMonthValue()){
124         List<TurnoUtente> turUt =
            ↳ turnoUtenteRep.getTurnoUtenteByTurno(t);
125         for(TurnoUtente tu : turUt){
126             int ore = tu.getOraFine() - tu.getOraInizio();
127             if(!tu.isFuoriOrario()){
128                 Integer oreLav =
                    ↳ utentiOreMensili.get(tu.getUtente());
129                 oreLav += ore;
130                 utentiOreMensili.put(tu.getUtente(), oreLav);
131             }
132             else{
133                 Integer oreLav =
                    ↳ utentiOreStraordinario.get(tu.getUtente());
134                 oreLav += ore;
135                 utentiOreStraordinario.put(tu.getUtente(),
                    ↳ oreLav);
136             }
137         }
138     }
139 }
140 }

```

Complessità $O(S * T * T_u)$, S sedi, T turni, T_u dipendenti per turno.

2.9.3 Allocazione dei dipendenti

Per ogni sede e per ogni ruolo sono create tre liste di dipendenti per gestire la priorità del ruolo: ogni dipendente può essere in grado di svolgere al massimo tre ruoli, si considera l'ordine di inserimento dei ruoli durante la registrazione del dipendente come indicazione della priorità in ordine decrescente. Per esempio, dato un dipendente con ruoli aiuto cuoco, cameriere e cassiere, esso sarà inserito nella lista principale per il ruolo aiuto cuoco, nella secondaria per cameriere e nella terziaria per cassiere. Una volta completate queste liste si procede all'allocazione dei dipendenti per tutti i turni della sede: per ogni turno l'algoritmo prova ad allocare i dipendenti nella lista principale del ruolo, se non ci riesce prova con quelli nella secondaria e infine nella terziaria. Se il turno è rimasto comunque scoperto viene inserito in una lista di turni scoperti che verrà gestita nella prossima fase. Un dipendente per essere considerato valido per un turno deve soddisfare diversi requisiti: Prima di tutto non deve aver lavorato nello stesso giorno più di otto ore (o quattro per i contratti part-time) e le ore rimaste fino al raggiungimento di questa soglia devono essere sufficienti a coprire interamente il turno, ciò dal suo inizio alla sua fine. Se questa prima selezione viene superata si deve controllare la presenza di sovrapposizioni con altri turni già assegnati e con periodi di malattia/ferie presenti nel database.

Allocazione dei dipendenti

```
42 List<Turno> uncovered = new ArrayList<>();
43
44 //Per ogni sede...
45 for (Sede sede: sedi){
46     HashMap<Ruoli, List<Utente>> utentiRuoloPrimario = new
        ↳ HashMap<>();
47     HashMap<Ruoli, List<Utente>> utentiRuoloSecondario = new
        ↳ HashMap<>();
48     HashMap<Ruoli, List<Utente>> utentiRuoloTerziario = new
        ↳ HashMap<>();
49
50     //Creo le liste di priorità per ruolo
51     for(Ruoli r : ruoli) {
52         if (!r.getAdmin()){
53             List<Utente> listaUtenti = new ArrayList<>();
```

```

54      List<RuoliUtente> emailUtenti = ruoliUtenteRep.getEmail(
    ↪ ByAziendaAndNomeRuolo(az.getPartitaIva(),
    ↪ r.getRuoloID().getRoleName());
55      List<Utente> listaPrimaria = new ArrayList<>();
56      List<Utente> listaSecondaria = new ArrayList<>();
57      List<Utente> listaTerziaria = new ArrayList<>();
58
59      for(RuoliUtente e : emailUtenti){
60          Utente ut = utRep.findUtenteByEmail(e.getEmail());
61          if(ut.getSede().getIndirizzo().equals(sede.get
    ↪ Indirizzo()))
62              listaUtenti.add(ut);
63      }
64
65      for (Utente u : listaUtenti) {
66          List<RuoliUtente> lru = ruoliUtenteRep.getRuolo
    ↪ UtenteByEmail(u.getEmail());
67          if (lru.size() == 1)
68              listaPrimaria.add(u);
69          else if (lru.size() > 1) {
70              if (lru.get(0).getNomeRuolo().equals(r.getRuolo
    ↪ ID().getRoleName()))
71                  listaPrimaria.add(u);
72              else if (lru.get(1).getNomeRuolo().equals(r.get
    ↪ RuoloID().getRoleName()))
73                  listaSecondaria.add(u);
74              else if (lru.size() > 2 &&
    ↪ lru.get(2).getNomeRuolo().equals(r.getRuolo
    ↪ ID().getRoleName()))
75                  listaTerziaria.add(u);
76          }
77      }
78
79      utentiRuoloPrimario.put(r, listaPrimaria);
80      utentiRuoloSecondario.put(r, listaSecondaria);
81      utentiRuoloTerziario.put(r, listaTerziaria);
82  }
83  }
84
85  //Per ogni turno...

```

```

86     for(Turno turno : turniPerSede.get(sede)){
87         boolean assegnato = false;
88         turnoRep.save(turno);
89         Ruoli ruolo =
            ↳ ruoliRep.getRuoliByRuoloID(turno.getRuoloID());
90         List<Turno> turniDiOggi =
            ↳ turnoRep.findTurnoBySedeAndData(sede, turno.getData());
91         //Per ogni utente che ha quel ruolo come primario
92         if(utentiRuoloPrimario.get(ruolo) != null) {
93             for (Utente utente : utentiRuoloPrimario.get(ruolo)) {
94                 assegnato = assegnaUtente(utente, turniDiOggi,
                    ↳ turno, utentiOreMensili);
95                 if (assegnato)
96                     break;
97             }
98         }
99         //Se non sono riuscito a completare il turno con gli utenti
100        ↳ che hanno quel ruolo come principale
101        if(!assegnato){
102            if(utentiRuoloSecondario.get(ruolo) != null) {
103                //Per ogni utente che ha quel ruolo come secondario
104                for (Utente utente :
                    ↳ utentiRuoloSecondario.get(ruolo)) {
105                    assegnato = assegnaUtente(utente, turniDiOggi,
                        ↳ turno, utentiOreMensili);
106                    if (assegnato)
107                        break;
108                }
109            }
110            //Se non sono riuscito a completare il turno con gli utenti
111            ↳ che hanno quel ruolo come secondario
112            if(!assegnato){
113                if(utentiRuoloTerziario.get(ruolo) != null) {
114                    //Per ogni utente che ha quel ruolo come terziario
115                    for (Utente utente :
                        ↳ utentiRuoloTerziario.get(ruolo)) {
116                        assegnato = assegnaUtente(utente, turniDiOggi,
                            ↳ turno, utentiOreMensili);
117                        if (assegnato)

```

```

17         break;
18     }
19 }
20 }
21 //Se non sono riuscito a completare il turno con il gli
22 ↪ utenti che hanno quel ruolo come terziario
23 if(!assegnato){
24     uncovered.add(turno);
25 }
26 }

```

Complessità $O(S * T * R * D_s)$, S sedi, T turni, R ruoli, D_s dipendenti della sede.

2.9.4 Allocazione dei turni rimasti scoperti tramite straordinari e trasferte

Questa fase viene eseguita solo se c'è almeno un turno rimasto scoperto nell'intera azienda. Per ogni turno scoperto si prova a completarlo facendo fare straordinari ai dipendenti della stessa sede che possono svolgere quel ruolo. A differenza dell'assegnazione "standard" uno straordinario può avere una durata inferiore a quella del turno. Un dipendente può effettuare al massimo due ore di straordinari al giorno e dieci al mese, per i contratti part-time questi limiti sono dimezzati. Se anche con gli straordinari non è stato possibile coprire il turno scoperto allora si procede con il valutare le trasferte di dipendenti dalle sedi vicine. La vicinanza delle sedi è gestita tramite un grafo pesato non orientato, dove il peso degli archi è la distanza in chilometri tra i nodi (ossia tra le sedi). Per trovare tutte le sedi vicine a quella del turno scoperto viene eseguita una visita in ampiezza del grafo che ritorna una lista con tutte le sedi distanti non più di cinquanta chilometri. Questa lista viene poi ordinata in modo crescente per distanza. Fatto ciò l'algoritmo prova ad assegnare il turno ai dipendenti delle sedi nella lista che possono svolgere il ruolo. Un dipendente per essere idoneo alla trasferta non deve avere turni assegnati lo stesso giorno in sedi diverse da quella di destinazione e deve poter coprire il turno completamente, dall'inizio alla fine.

Allocazione dei turni rimasti scoperto tramite straordinari e trasferte

```
228 if(uncovered.size() > 0){
229     //Creo il grafo delle sedi
230     SimpleWeightedGraph<Sede, DefaultWeightedEdge> grafo = new
        ↳ SimpleWeightedGraph<>(DefaultWeightedEdge.class);
231     HashMap<Sede, List<Sede>> mappaSediVicine = new HashMap<>();
232
233     //Se ci sono più sedi le aggiungo al grafo
234     if(sedi.size() > 1) {
235         List<VicinanzaSedi> listaVicinanza = vicSediRep.findAll();
236         for (VicinanzaSedi vc : listaVicinanza) {
237             SediScambioID sediScambio = vc.getSediDiScambio();
238             Sede s1 = sediScambio.getSedeFrom();
239             Sede s2 = sediScambio.getSedeTo();
240             if (!grafo.containsVertex(s1))
241                 grafo.addVertex(s1);
242             if (!grafo.containsVertex(s2))
243                 grafo.addVertex(s2);
```

```

244         Float distanza = vc.getDistanza();
245         DefaultWeightedEdge e = grafo.addEdge(s1, s2);
246         grafo.setEdgeWeight(e, distanza.doubleValue());
247     }
248     //Calcolo le sedi vicine (entro DistanzaMax) per ogni sede
249     for(Sede sede : sedi){
250         List<Sede> listaSediVicine =
251             ↳ visitaInAmpiezza(Algoritmo.DistanzaMax, sede,
252             ↳ grafo);
253         mappaSediVicine.put(sede, listaSediVicine);
254     }
255     //Per ogni turno scoperto...
256     for(Turno turnoScoperto : uncovered){
257         boolean assegnato = false;
258         Ruoli ruolo =
259             ↳ ruoliRep.getRuoliByRuoloID(turnoScoperto.getRuoloID());
260         List<Turno> turniDiOggi =
261             ↳ turnoRep.findTurnoBySedeAndData(turnoScoperto.getSede(),
262             ↳ turnoScoperto.getData());
263
264         //Trovo tutti quelli che possono svolgere quel ruolo nella
265         ↳ stessa sede
266         List<RuoliUtente> emailUtenti = ruoliUtenteRep.getEmailBy
267             ↳ AziendaAndNomeRuolo(az.getPartitaIva(),
268             ↳ turnoScoperto.getRuoloID().getRoleName());
269         HashMap<Sede, List<Utente>> mappaUtentiRuoloSede = new
270             ↳ HashMap<>();
271
272         for(RuoliUtente e : emailUtenti){
273             Utente ut = utRep.findUtenteByEmail(e.getEmail());
274             mappaUtentiRuoloSede.computeIfAbsent(ut.getSede(), k ->
275                 ↳ new ArrayList<>());
276             mappaUtentiRuoloSede.get(ut.getSede()).add(ut);
277         }
278
279         mappaUtentiRuoloSede.computeIfAbsent(turnoScoperto.get
280             ↳ Sede(), k -> new
281             ↳ ArrayList<>());

```

```

72      //Per ogni utente della sede che può svolgere il ruolo...
73      if(mappaUtentiRuoloSede.get(turnoScoperto.getSede()) !=
74      ↪ null) {
75          for (Utente utente :
76              ↪ mappaUtentiRuoloSede.get(turnoScoperto.getSede())) {
77              assegnato = assegnaUtenteStraordinario(utente,
78              ↪ turniDiOggi, turnoScoperto,
79              ↪ utentiOreStraordinario);
80              if (assegnato)
81                  break;
82          }
83      }
84
85      //se non sono riuscito a completare il turno facendo fare
86      ↪ straordinari ai dipendenti della stessa sede...
87      if(!assegnato){
88          List<Sede> sediVicine =
89          ↪ mappaSediVicine.get(turnoScoperto.getSede());
90          turniDiOggi =
91          ↪ turnoRep.findTurnoByData(turnoScoperto.getData());
92
93          if(sediVicine != null && sediVicine.size() > 0){
94              //Per ogni sede vicina...
95              for(Sede sede : sediVicine){
96                  //Trovo qualcuno che può fare la trasferta
97                  if(mappaUtentiRuoloSede.get(sede) != null) {
98                      for (Utente utente :
99                          ↪ mappaUtentiRuoloSede.get(sede)) {
100                          List<TurnoUtente> turniDiOggiUtente =
101                          ↪ turnoUtenteRep.findTurnoUtenteBy「
102                          ↪ UtenteAndTurnoIn(utente,
103                          ↪ turniDiOggi);
104                          assegnato =
105                          ↪ assegnaUtenteTrasferta(utente,
106                          ↪ turniDiOggiUtente, turnoScoperto,
107                          ↪ utentiOreMensili);
108                          if (assegnato)
109                              break;
110                      }
111                  }
112              }
113          }
114      }

```

```

98         }
99         if(assegnato)
100             break;
101     }
102 }
103 }
104 if(!assegnato)
105     assegnamentoCompleto = false;
106 }
107 }

```

Complessità $O(T_sc * S_v * D_s * R)$, T_sc turni scoperti, S_v sedi vicine.

2.10 Entity-Relationship Diagram

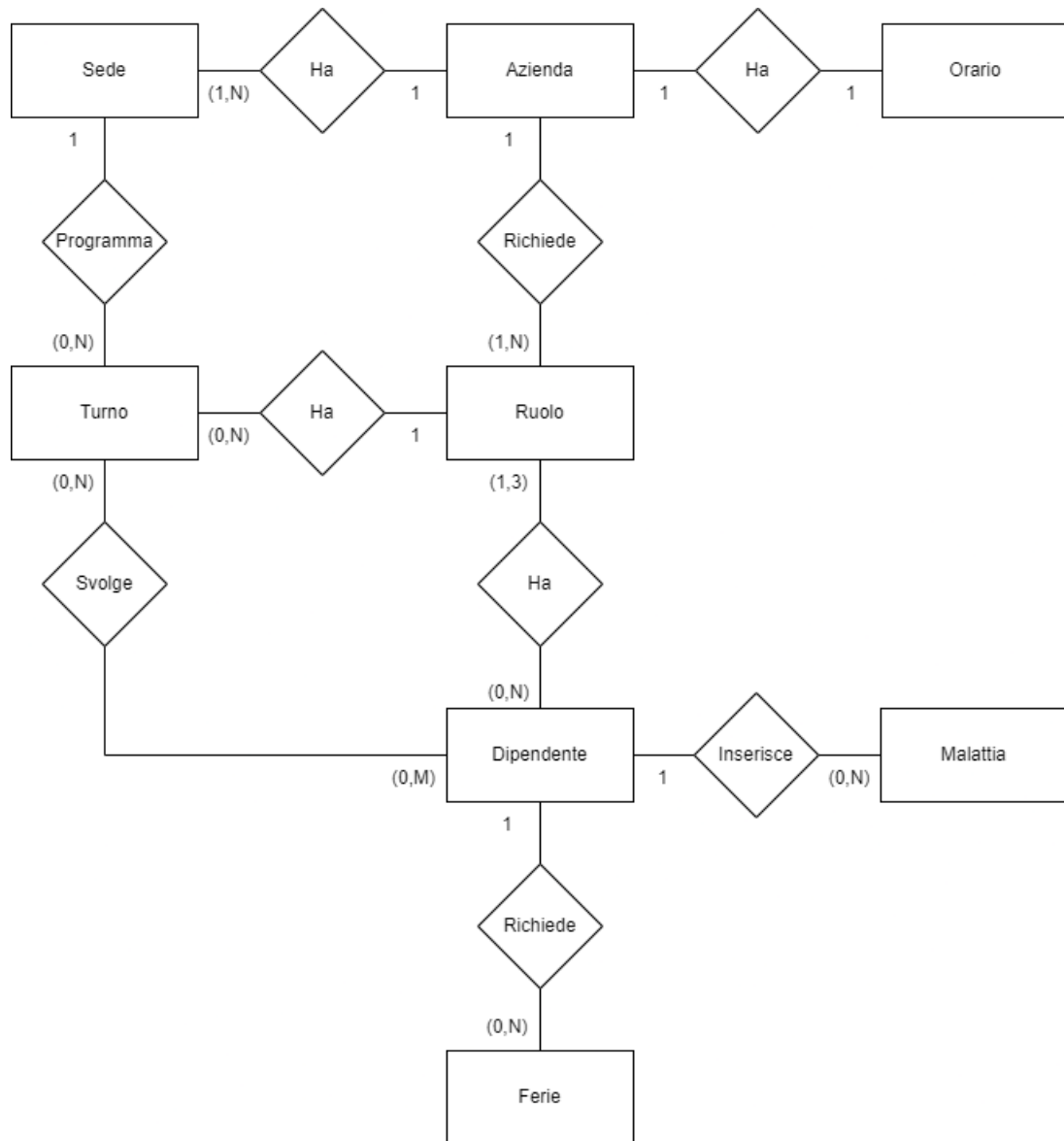


Figura 3: Diagramma Entity-Relationship

2.11 UML Class Diagram

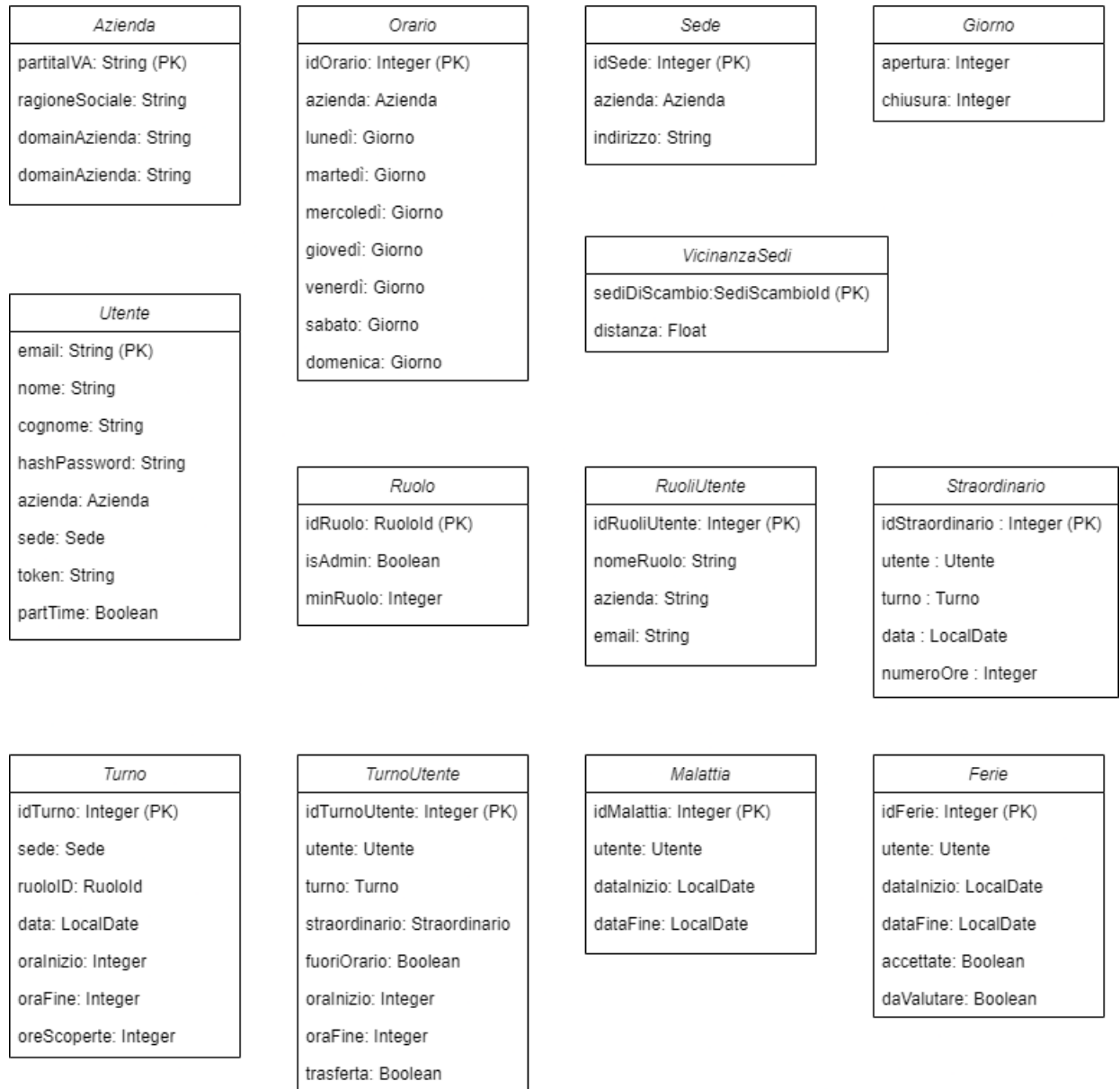


Figura 4: Diagramma delle classi

2.12 UML Component Diagram

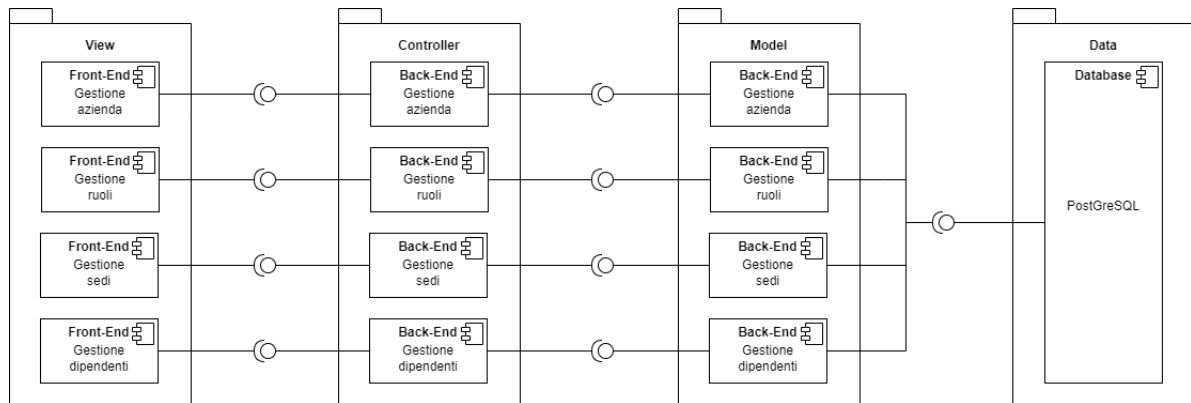


Figura 5: Diagramma delle componenti

A partire dai casi d'uso previsti per questa iterazione sono state previste le componenti in Figura basandosi sul pattern architetturale *MVC*:

- View: interfacce utente nella web app, accessibili tramite un menù.
- Controller: componenti lato server per gestire le interazioni tra l'utente e i dati mediante esposizione di API ai client.
- Model: classi e interfacce per la gestione dei dati e la comunicazione con il database.

2.13 UML Deployment Diagram

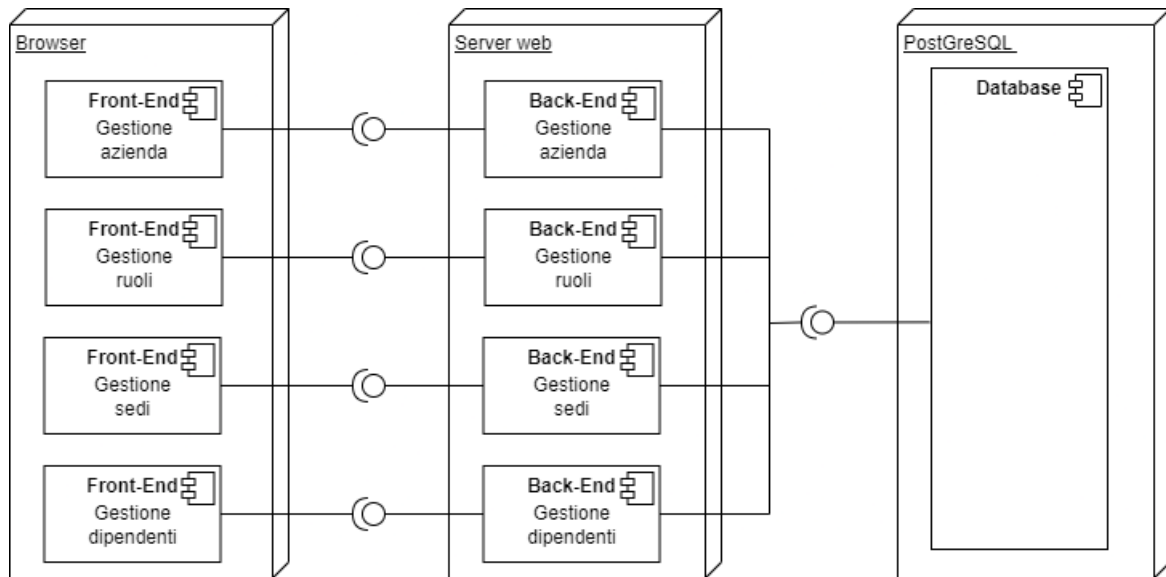


Figura 6: Deployment diagram

Per questa iterazione l'applicazione è distribuita su tre nodi:

- Browser: permette l'interazione tra il manager dell'azienda e le interfacce grafiche dell'applicazione.
- Server web: espone le API per i client che ne fanno richiesta.
- Database: gestisce la memorizzazione dei dati in modo persistente.

2.14 Documentazione API

Di seguito la descrizione delle API che sono state implementate per questa iterazione.

2.14.1 /registrazione

- **Funzione:** il client invia i dati dell'azienda che si vuole registrare al server che, se sono validi, li memorizza nel database.
- **Metodo:** POST
- **Parametri:** Un unico oggetto in notazione JSON con la seguente struttura:
 1. "manager":
 - "email": `String`
 - "password": `String`
 - "nome": `String`
 - "cognome": `String`
 2. "azienda":
 - "partitaIVA": `String`
 - "ragioneSociale": `String`
 - "suffissoEmail": `String`
 - "sedeHQ": `String`
 3. "ruoli": [vettore]
 - "nomeRuolo": `String`
 - "numeroDipendenti": `int`
 4. "dipendenti": [vettore]
 - "nomeDipendente": `String`
 - "cognomeDipendente": `String`
 - "ruoliDipendente": [`String`]
 - "partTime": `boolean`
 5. "orario":
 - "lunedì":
 - * "apertura": `int` (-1 se è un giorno di chiusura)
 - * "chiusura": `int`
 - "martedì": ...
- **Risposta:** `int` (1 se la registrazione è avvenuta con successo, 0 altrimenti)

2.14.2 /login

- **Funzione:** il manager invia le proprie credenziali di accesso (email e password) e il server instaura una sessione HTTP se sono corrette. Se un dipendente prova ad accedere alla web app viene segnalato un errore.
- **Metodo:** POST
- **Parametri:**
 1. "email": String
 2. "password": String
- **Risposta:** **int** (-1 credenziali errate, 1 credenziali dipendente, 2 credenziali manager)

2.14.3 /logoutPagina

- **Funzione:** distruzione della sessione HTTP.
- **Metodo:** POST
- **Parametri:** nessuno, ma ci deve essere una sessione già instaurata.
- **Risposta:** **int** (0 sessione non presente, 1 sessione distrutta)

2.14.4 /indirizzoSedi

- **Funzione:** ritorna l'elenco degli indirizzi di tutte le sedi dell'azienda.
- **Metodo:** POST
- **Parametri:** nessuno, ma ci deve essere una sessione già instaurata (contiene i dati dell'azienda).
- **Risposta:** [String] (**null** se la sessione non è presente)

2.14.5 /nomeRuoli

- **Funzione:** ritorna l'elenco di tutti i ruoli dell'azienda.
- **Metodo:** POST
- **Parametri:** nessuno, ma ci deve essere una sessione già instaurata (contiene i dati dell'azienda)
- **Risposta:** [String] (**null** sessione non presente)

2.14.6 /checksession

- **Funzione:** al caricamento della pagina web il client chiede al server se esiste già una sessione, in caso contrario il manager deve effettuare il login.
- **Metodo:** POST
- **Parametri:** nessuno, ma ci deve essere una sessione già instaurata.
- **Risposta:** **int** (0 sessione non presente, 1 sessione presente)

2.14.7 /aggiungiSede

- **Funzione:** il manager inserisce i dati di una nuova sede nell'interfaccia grafica e li invia al server. Se sono validi viene aggiornato il database.
- **Metodo:** POST
- **Parametri:**
 1. "sede":
 - "indirizzo": **String**
 - "sedeVicina": **String**
 - "distanza": **float**
 2. "dipendenti": [vettore]
 - "nomeDipendente": **String**
 - "cognomeDipendente": **String**
 - "ruoliDipendente": [String]
 - "partTime": **boolean**
- **Risposta:** **int** (-1 indirizzo già presente, 0 sessione non presente, 1 inserimento avvenuto con successo)

2.14.8 /checkCalcoloTurni

- **Funzione:** controlla se, dato un mese e un anno, c'è almeno un turno calcolato dall'inizio alla fine del mese.
- **Metodo:** POST
- **Parametri:** `String` (mese; anno)
- **Risposta:** `int` (0 nessun turno, 1 almeno un turno)

2.14.9 /getStatusGiorni

- **Funzione:** il client richiede al server lo stato di tutti i giorni di un determinato mese per una sede. Può essere un giorno di chiusura e possono esserci o meno turni scoperti.
- **Metodo:** POST
- **Parametri:**
 1. "anno": `int`
 2. "mese": `int`
 3. "indirizzoSede": `String`
- **Risposta:** [`int`] (per ogni giorno: -1 giorno di chiusura, 0 almeno un turno scoperto, 1 tutti i turni coperti)

2.14.10 /ricalcolaTurni

- **Funzione:** comunica al server di ricalcolare i turni di un determinato mese.
- **Metodo:** POST
- **Parametri:** `String` (mese; anno)
- **Risposta:** `int` (0 errore, 1 allocazione turni incompleta, 2 allocazione turni completa)

2.14.11 /getTurniSedeGiorno

- **Funzione:** ritorna l'elenco di tutti i turni di una sede per un determinato giorno.
- **Metodo:** POST
- **Parametri:**
 1. "anno": `int`
 2. "mese": `int`
 3. "giorno": `int`
 4. "indirizzoSede": `String`
- **Risposta:**
 1. "turni": [vettore]
 - "nomeRuolo": `String`
 - "oraInizio": `int`
 - "oraFine": `int`
 - "oreScoperte": `int`
 - "utenti": [vettore]
 - * "nomeUtente": `String`
 - * "cognomeUtente": `String`
 - * "straordinario": `boolean`
 - * "trasferta": `boolean`
 - * "inizio": `int`
 - * "fine": `int`

2.15 Testing

2.15.1 Test dell'API del database

Il testing dell'API del database è stata effettuata con JUnit, utilizzando metodi helper forniti dalla libreria AssertJ e appoggiandosi ai metodi di supporto forniti direttamente da Jpa, la libreria di astrazione usata per la definizione del database. Le classi sono state estese tramite la keyword `@DataJpaTest`. Questo permette di configurare la classe dichiarata con quella specifica decorazione come classe di test di Jpa, permettendoci di usare decorazioni come `@Autowired`, facilitando quindi il collegamento del database alle classi di test. Infine `@ExtendWith(SpringExtension.class)` permette di preconfigurare l'ambiente di JUnit per l'utilizzo del testing di Spring.

Prima di poter fare testing sul database, dobbiamo configurare l'istanza del database di test in modo da permettere a Spring di sapere su cosa deve andare a eseguire i test stessi. Come già visto, abbiamo scelto di usare PostgreSQL come DBMS per l'applicazione a runtime; per facilitare il testing, però, abbiamo invece scelto H2 come database di testing, questo per via di due vantaggi:

1. Nessuna dipendenza a runtime da software applicativo esterno, facilitando quindi la configurazione di batch di test e riducendone la dipendenza dei fallimenti da parte di problemi del database stesso piuttosto che l'API utilizzata
2. Facilità di creazione della base di dati stessi, visto che ad ogni esecuzione il database è interamente contenuto in memoria del calcolatore, permettendoci quindi di iterare rapidamente i test, visto che possiamo semplicemente deallocare i dati precedentemente in memoria e riallocarne un'altro vuoto pronto per il test successivo.

Definizione della classe `AziendaRepositoryTest.java`

```
15 @ExtendWith(SpringExtension.class)
16 @DataJpaTest
17 public class AziendaRepositoryTest {
18     @Autowired private AziendaRepository azRepo;
19     @Autowired private DataSource dataSource;
```

All'interno della classe appena creata vengono definiti dei metodi con nomi che fanno riferimento alla specifica funzionalità da testare. Al loro interno ogni test è composto da tre parti ben definite:

- Given: In questa fase il test viene preparato, precaricando tutte le componenti necessarie per il testing della funzione specifica. L'ideale è quello di utilizzare

meno componenti software possibili, in modo da minimizzare il rischio che il test fallisca per colpa di un altro componente richiesto.

Nel caso del testing su JPA, questo tipicamente si compone caricando il numero minimo di repository di database e definendo i dati di test da salvare nel database.

- When: In questa parte effettuiamo le operazioni effettive sul componente in test (e sue eventuali dipendenze).

Nel caso del testing su JPA, effettuiamo il commit sui database effettivi dei dati preparati precedentemente, in modo da avere uno stato noto all'interno del database e quindi poter verificarne la presenza effettiva nella fase successiva.

- Then: In questa fase finale viene verificato che le operazioni precedentemente effettuate sul componente in test ha avuto l'esito desiderato.

Nel caso del testing su JPA, ciò che viene fatto è quello di effettuare operazioni di lettura e di compararlo con i dati che ci aspettiamo siano presenti. AssertJ ci viene in aiuto fornendoci metodi come `assertThat(metodoCheRitornaQualcosa())` che fornisce un'oggetto del tipo `T ObjectAssert<T>`, che permette di utilizzare metodi accessori associati per verificarne il contenuto.

Vediamone subito un esempio per capirne meglio l'API fornita:

Un test della classe `AziendaRepositoryTest.java`

```
27 @Test
28 void findByOk(){
29     Azienda exists = new Azienda("PI6898", "UniBG", "unibg.it");
30     Azienda unavailable = new Azienda("PI07348", "Inesistente",
31     ↪ "nowhere.com");
32     azRepo.save(exists);
33     //Always identical to exists
34     assertThat(azRepo.findAziendaByPartitaIva(exists.getPartitaIva()
35     ↪ Iva()))
36         .isNotNull()
37         .isEqualTo(exists);
38     //Always null
39     assertThat(azRepo.findAziendaByPartitaIva(unavailable.getPartitaIva()
40     ↪ PartitaIva()))
41         .isNull();
42     //Always identical to exists
```

```

40    assertThat(azRepo.findAziendaByRagioneSociale(exists.getRagione_
    ↪    Sociale()))
41        .isNotNull()
42        .isEqualTo(exists);
43    //Always null
44    assertThat(azRepo.findAziendaByRagioneSociale(unavailable.get_
    ↪    RagioneSociale()))
45        .isNull();
46 }

```

In questo test, possiamo vedere come le chiamate ad `assertThat()` vengono poi verificate tramite i due metodi `.isEqualTo()` e `.isNull()`. Questi quindi restituiranno il risultato del test in base al risultato della chiamata. Questa API permette anche di concatenare più verifiche in un'unica chiamata, effettuando chiamate a catena sull'oggetto ritornato da `assertThat()`

Abbiamo quindi definito una classe Java dedicata per ogni repository creata, caricato le sue dipendenze minime derivate da parte della struttura relazionale del database e quindi utilizzato le API definite per verificarne la funzionalità desiderata.

2.15.2 Analisi dinamica

Sono stati implementati i seguenti casi di test delle API tramite Postman:

Test login

```

1  pm.test("Login corretto manager", () => {
2      const responseJson = pm.response.json();
3      pm.expect(responseJson).to.eql(2);
4  });
5
6  pm.test("Login corretto dipendente", () => {
7      const responseJson = pm.response.json();
8      pm.expect(responseJson).to.eql(1);
9  });
10
11 pm.test("Login errato", () => {
12     const responseJson = pm.response.json();
13     pm.expect(responseJson).to.eql(-1);
14 });

```


Test registrazione

```
1 pm.test("Registrazione corretta", () => {  
2     const responseJson = pm.response.json();  
3     pm.expect(responseJson).to.eql(1);  
4 });  
5  
6 pm.test("Registrazione azienda già presente nel database", () => {  
7     const responseJson = pm.response.json();  
8     pm.expect(responseJson).to.eql(0);  
9 });
```

Test controllo presenza turni

```
1 pm.test("Almeno un turno presente", () => {  
2     const responseJson = pm.response.json();  
3     pm.expect(responseJson).to.eql(1);  
4 });  
5  
6 pm.test("Nessun turno presente", () => {  
7     const responseJson = pm.response.json();  
8     pm.expect(responseJson).to.eql(0);  
9 });
```

3 Iterazione 2

3.1 Introduzione

In questa iterazione sono descritti i casi d'uso riguardanti le azioni che hanno a loro disposizione i dipendenti tramite l'*applicazione Android* dedicata ad essi. Le azioni principali implementate fanno riferimento in particolare alla *visualizzazione dei turni* e alle richieste che si suddividono in base alla loro natura; *richiesta di malattia* e *richiesta di ferie*.

3.2 UC8: Visualizzazione Calendario

3.2.1 UC8.1 Visualizzazione Turni Dipendente

Descrizione: Il dipendente tramite applicazione Android può visualizzare i propri turni di un range di date da lui selezionato dalla pagina apposita.

Attori coinvolti: Dipendente

Condizione di attivazione: Login all'app, scelta di date corrette e click sul bottone "Visualizza".

Postcondizione: Se le date selezionate sono corrette (inizio minore di fine) vengono mostrati nella lista le informazioni richieste. Nel caso in cui i turni non siano presenti per quelle date un Toast con un messaggio provvede ad informare il dipendente dell'assenza di dati.

Procedimento:

1. Il dipendente ha effettuato il login all'applicazione
2. Accesso alla pagina *Turni* dal menu
3. Dipendente seleziona le date di inizio e fine visualizzazione
4. Click su *Visualizza*
5. Output della lista dei turni con le informazioni associate

3.2.2 UC8.2 Visualizzazione Turni Manager

Descrizione: Il manager ha la possibilità, una volta calcolati i turni di visualizzare un calendario mensile di ogni sede con i giorni che assumono colori diversi a seconda

della composizione dei turni.

Attori coinvolti: Manager

Condizione di attivazione: Login all'applicazione web e calcolo dei turni per il periodo di visualizzazione.

Postcondizione: Visualizzazione del calendario e possibilità di aprire il dettaglio di ogni singolo giorno.

Procedimento:

1. Il manager ha effettuato il login all'applicazione
2. Accesso alla pagina *Calendario* dal menu
3. Il manager seleziona i campi necessari per la visualizzazione: sede, mese e anno.
4. Click su *Visualizza*
5. Output del calendario con associati i turni del singolo giorno.
6. Click sul giorno per visualizzare i turni calcolati ed eventuali turni scoperti.

3.3 UC9: Login e logout dipendenti

Descrizione: Dipendente scarica l'applicazione e inserendo le credenziali accede. Uscita dall'applicazione quando termina le operazioni.

Attori coinvolti: Dipendente, Sistema

Condizione di attivazione: Download dell'app, inserimento credenziali e click sul bottone *Login*. Click sulla voce del menù Logout

Postcondizione: Accesso all'applicazione in caso di login. Uscita dall'applicazione, tornando alla pagina di login, in caso di logout.

Procedimento Login:

1. L'utente apre l'applicazione dal proprio dispositivo mobile
2. Inserisce le proprie credenziali nei campi *Email* e *Password*
3. Click sul bottone *Login*
4. Accesso all'applicazione

Procedimento Logout:

1. L'utente ha effettuato il login all'app
2. Apre il menù laterale e click sulla voce *Logout*
3. Uscita dall'app e viene mostrata la pagina login

3.4 UC10: Inserimento malattia

Descrizione: L'utente inserisce la richiesta di malattia che viene ricevuta dal sistema.

Attori coinvolti: Dipendente, Sistema

Condizione di attivazione: Inserimento periodo malattia e click sul bottone *Invia Richiesta*. Non deve già essere presente una malattia che si sovrapponga alle date selezionate.

Postcondizione: Toast che avvisa l'utente che la richiesta è andata a buon fine.

Procedimento:

1. L'utente ha effettuato il login all'app
2. Sceglie la pagina *Malattia* dal menù laterale
3. Inserisce le date di inizio e fine periodo malattia
4. Click sul bottone *Invia Richiesta*
5. Visualizzazione del risultato tramite Toast

3.5 UC11: Inserimento ferie

Descrizione: L'utente inserisce la richiesta di ferie che viene inserita nel sistema e valutata dal Manager.

Attori coinvolti: Dipendente, Sistema, manager

Condizione di attivazione: Inserimento periodo di ferie e click sul bottone *Invia Richiesta*.

Postcondizione: Toast con avviso per il ricevimento della richiesta.

Procedimento:

1. L'utente ha effettuato il login all'app
2. Sceglie la pagina *Ferie* dal menù laterale
3. Inserisce le date di inizio e fine periodo di ferie
4. Click sul bottone *Invia Richiesta*
5. Visualizzazione del risultato tramite Toast

3.6 UC12: Valutazione richieste di ferie

NOT IMPLEMENTED

Descrizione: A seguito della richiesta di ferie effettuata da un dipendente, il manager deve rispondere all'istanza entro un determinato tempo (almeno prima dell'inizio del periodo di ferie richiesto).

Attori coinvolti: Dipendenti, Sistema, Manager

Condizione di attivazione: Ricezione di richiesta ferie da parte di un dipendente

Postcondizione: Approvazione o rifiuto della richiesta

Procedimento: Il manager per poter valutare la richiesta deve controllare che per i turni del periodo di assenza del dipendente sia in grado di coprire i suoi turni con altri dipendenti.

3.7 UML Component Diagram

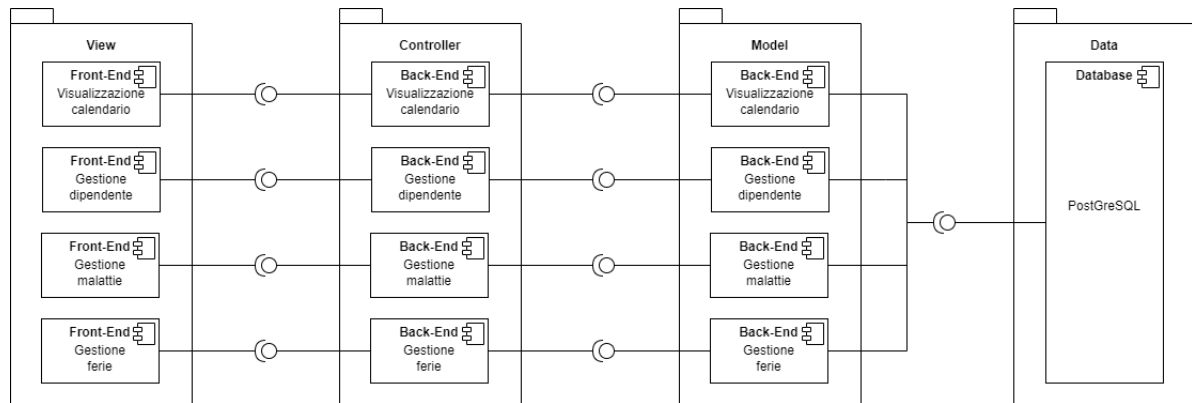


Figura 7: Diagramma delle componenti

A partire dai casi d'uso previsti per questa iterazione sono state previste le componenti in Figura basandosi sul pattern architetturale *MVC*:

- View: interfacce utente nella web app e nell'applicazione Android.
- Controller: componenti lato server per gestire le interazioni tra l'utente e i dati mediante esposizione di API ai client.
- Model: classi e interfacce per la gestione dei dati e la comunicazione con il database.

3.8 UML Deployment Diagram

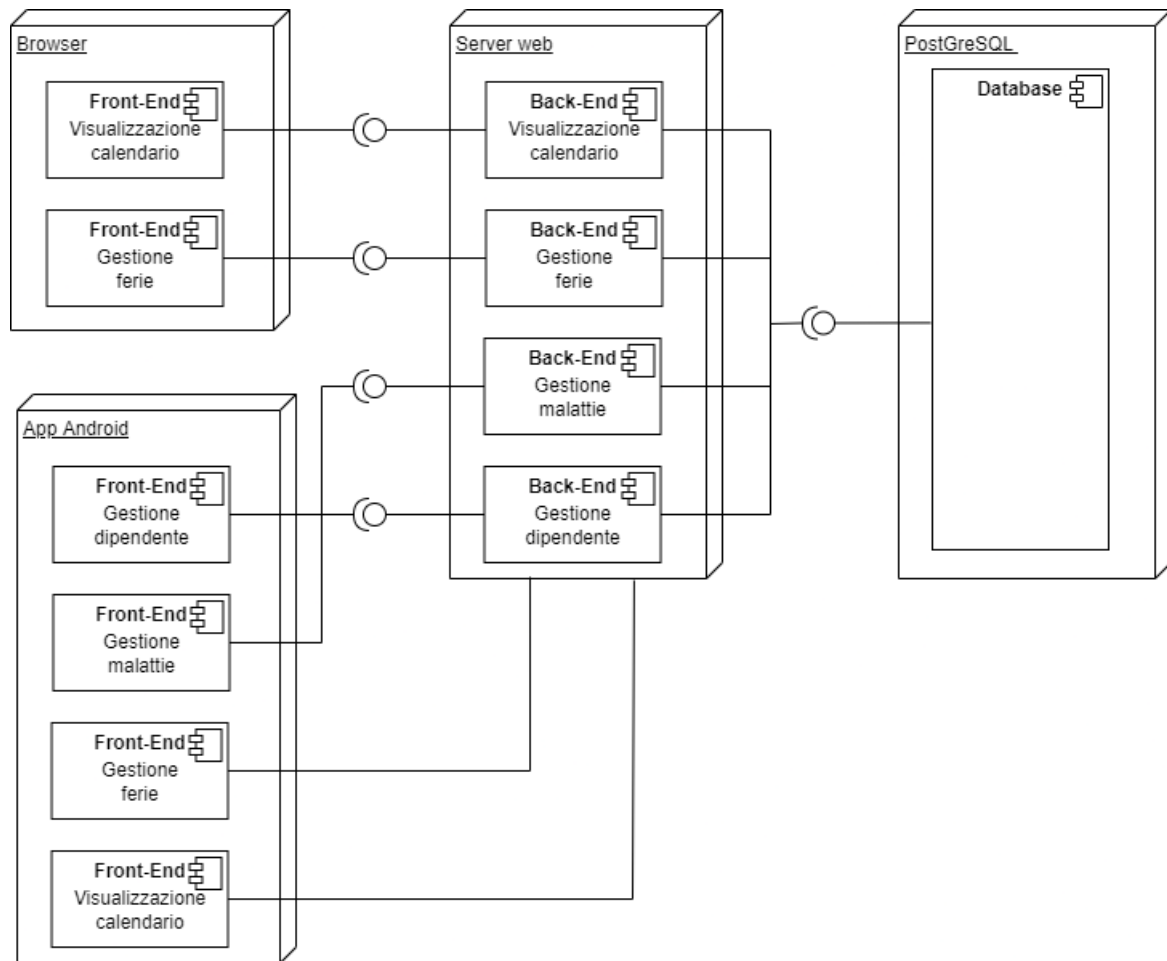


Figura 8: Deployment diagram

Per questa iterazione l'applicazione è distribuita su quattro nodi:

- Browser: permette la visualizzazione del calendario turni al manager dell'azienda e l'accettazione o il rifiuto delle richieste di ferie.
- App Android: consente ai dipendenti di inserire giorni di malattia o richieste di ferie e di visualizzare i turni a cui sono stati assegnati.
- Server web: espone le API per i client che ne fanno richiesta.
- Database: gestisce la memorizzazione dei dati in modo persistente.

3.9 Documentazione API

3.9.1 Introduzione

Per permettere la comunicazione dell'applicazione Android dedicata ai dipendenti con il database contenente i dati di loro interesse e quindi i casi d'uso descritti nell'Iterazione 2 abbiamo implementato un sistema basato sulle richieste REST. L'implementazione delle API è stata inserita nella sotto-directory del progetto */api* per mantenere la parte dedicata all'app Android separata rispetto a quelle della web app.

3.9.2 API per login

Descrizione: L'API per la gestione del login sull'app ha il compito di confermare la presenza di un Utente nel database che abbia i dati specificati nella form di login. Inoltre deve permettere di aprire una sessione di comunicazione che renda valide le richieste effettuate successivamente dall'utente.

URL: */api/login*

Caso d'uso: UC9

Implementazione: Per prima cosa i parametri contenuti nella richiesta sono l'email e la password inserita nella form dall'utente che ha effettuato la richiesta. Il back-end come conseguenza della ricezione della richiesta verifica la presenza dell'utente nel database. In caso positivo viene creato un *token* random e univoco che rappresenta anche la risposta della chiamata. L'applicazione salva questo token nelle *Shared Preferences*, oltre ad altri dati dell'utente, per poterlo riutilizzare nelle chiamate successive.

3.9.3 API per logout

Descrizione: L'API per il logout ha il compito di verificare l'effettiva esistenza dell'utente che sta eseguendo la richiesta per poi cancellare dal database il token ad esso associato, in modo da invalidare la sessione che era in corso.

URL: */api/logout*

Caso d'uso: UC9

Implementazione: I parametri contenuti nella richiesta sono i token e la mail in-

serita nella fase di login, entrambi reperibili dalle Shared Preferences. Il back-end verifica l'effettiva esistenza di un utente dai dati ricevuti; in caso negativo restituisce la Stringa *ERROR*, altrimenti viene cancellato il token associato all'utente nel database e la stringa tornata ha valore *DONE*. A seconda del risultato l'applicazione effettua il logout cancellando i dati contenuti nelle Shared Preferences o meno.

3.9.4 API per get turni

Descrizione: L'API associata alla richiesta dei turni ha il compito di restituire le occorrenze relative al periodo richiesto riguardanti lo specifico dipendente che effettua la richiesta.

URL: */api/richiestaTurni*

Caso d'uso: *UC8*

Implementazione: In questo caso il token è necessario, sia per la verifica della legittimità della richiesta che per identificare l'utente che l'ha effettuata. Se risulta corretto vengono estratti dal database i dati relativi ai turni compresi tra le date di inizio e fine ricevuti anch'esse come parametri. In caso non sia corretta la richiesta, il dipendente non esiste o non sono presenti turni per le date scelte viene restituito *null*.

3.9.5 API per richiesta malattia

Descrizione: L'API per la richiesta della malattia è l'unica che non legge dati da database ma li scrive. In questo caso dopo aver identificato l'utente viene inserito il record della malattia e i turni devono essere ricalcolati.

URL: */api/richiestaMalattia*

Caso d'uso: *UC10*

Implementazione: Come per la richiesta turni il token funge da identificatore per l'utente e verifica la correttezza della richiesta. In caso affermativo viene inserito nel database il record della malattia.

3.9.6 Testing App Android

Per testare l'applicazione Android è stato utilizzato *lint* tramite Android Studio. Il tool *lint* permette di effettuare l'*analisi statica* del codice con l'obiettivo di ottenere migliori prestazioni ed avere un codice leggibile e con un buon livello di manutenibilità.

4 Guida per l'uso dell'applicazione

4.1 Applicazione web - Manager

1. Aprire un browser e connettersi al server sulla porta 8080.
2. Selezionare *login* se si ha già un account, *registrati* per crearne uno nuovo.

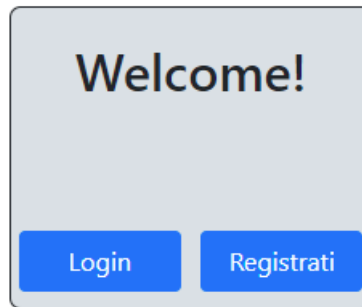


Figura 9: Finestra mostrata all'apertura dell'applicazione web

3. Per effettuare il login è sufficiente inserire email e password.

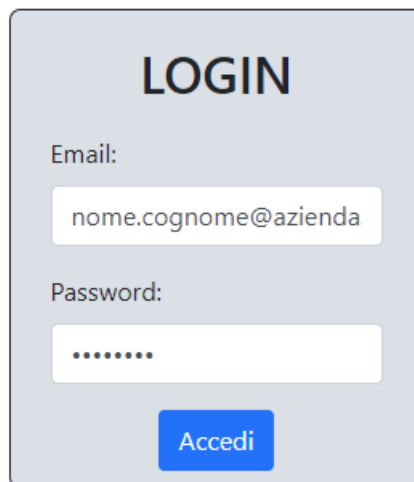
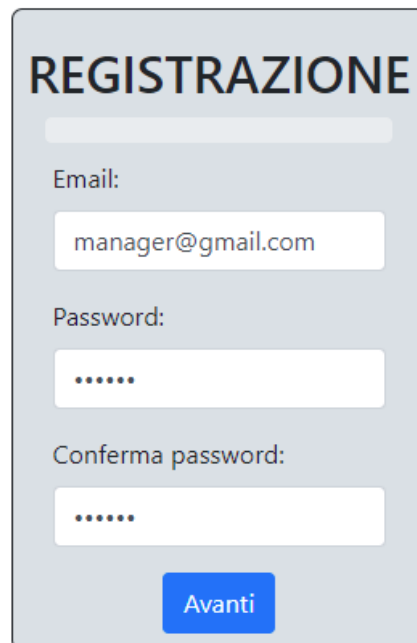


Figura 10: Finestra login

4. Per registrarsi inserire le credenziali del manager.



A registration form titled "REGISTRAZIONE" with a progress bar at the top. It contains three input fields: "Email:" with the value "manager@gmail.com", "Password:" with masked characters "*****", and "Conferma password:" with masked characters "*****". A blue "Avanti" button is at the bottom.

Figura 11: Finestra registrazione 1

5. Inserire i dati personali del manager.



A registration form titled "REGISTRAZIONE" with a progress bar at the top showing "20%". It contains two input fields: "Nome:" with the value "Mario" and "Cognome:" with the value "Rossi". A blue "Avanti" button is at the bottom.

Figura 12: Finestra registrazione 2

6. Inserire i dati dell'azienda.

REGISTRAZIONE

40%

Partita IVA:

P10T57V42

Ragione sociale:

Azienda S.r.l.

Suffisso email aziendale:

@ azienda.com

Indirizzo sede centrale:

Bergamo

Avanti

Figura 13: Finestra registrazione 3

7. Inserire i ruoli (click sul pulsante + per aggiungere altri ruoli).

REGISTRAZIONE

60%

Ruolo 1:

Programmatore

Numero dipendenti necessari:

4

+ -

Avanti

Figura 14: Finestra registrazione 4

8. Inserire i dipendenti della sede centrale (click sul pulsante + per aggiungere altri dipendenti).

The image shows a mobile application registration screen titled "REGISTRAZIONE". At the top, there is a blue progress bar indicating 80% completion. Below the title, the form is for "Dipendente 1:". It includes input fields for "Nome:" (filled with "Lucia") and "Cognome:" (filled with "Bianchi"). Under "Ruoli:", there are three dropdown menus; the first is set to "Programmatore", and the other two show a hyphen. A "Part time:" checkbox is present and is currently unchecked. At the bottom of the form, there are two large buttons: a green one with a "+" sign and a red one with a "-" sign. Below these is a blue button labeled "Avanti".

Figura 15: Finestra registrazione 5

9. Inserire l'orario settimanale di apertura dell'azienda.

REGISTRAZIONE

100%

Orario di apertura:

Lunedì	8	▼	16	▼
Martedì	8	▼	16	▼
Mercoledì	8	▼	16	▼
Giovedì	8	▼	16	▼
Venerdì	8	▼	16	▼
Sabato	Chiuso	▼	Chiuso	▼
Domenica	Chiuso	▼	Chiuso	▼

Avanti

Figura 16: Finestra registrazione 6

10. Effettuato il login, selezionando *Sedi* nel menù posto nella parte superiore della pagina, è possibile aggiungere nuove sedi, inserendo l'indirizzo ed eventualmente la distanza dalla sede più vicina (l'inserimento dei dipendenti della sede è identico a quello visto in fase di registrazione).

Profilo Azienda Sedi Ruoli Dipendenti Calendario Richieste ferie Logout

Gestione sedi

Aggiungi Modifica Cancella

Indirizzo:
Bergamo

Indirizzo sede più vicina:
Dalmine

Distanza (km):
15

Avanti

Figura 17: Inserimento di una nuova sede

11. Selezionando invece *Calendario* e scegliendo il mese e la sede è possibile visualizzare il calendario: in bianco i giorni di chiusura (o senza alcun turno calcolato), in verde i giorni con tutti i turni coperti e in rosso quelli con almeno un turno scoperto.

Profilo Azienda Sedi Ruoli Dipendenti Calendario Richieste ferie Logout

Calendario

Sede:
Dalmine

Mese:
Aprile

Anno:
2022

Visualizza Ricalcola

Aprile 2022						
Lunedì	Martedì	Mercoledì	Giovedì	Venerdì	Sabato	Domenica
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Figura 18: Visualizzazione calendario sede

12. Tramite un click su un giorno nel calendario si possono visualizzare tutti i turni di quella giornata. In bianco i turni completi, in rosso quelli scoperti.

Profilo Azienda Sedi Ruoli Dipendenti Calendario Richieste ferie	Logout
--	--------

<p>Turni del 5/4/2022</p> <p>Ruolo: Amministratore Ora inizio: 8 Ora fine: 12 Ore scoperte: 0</p> <p>Ruolo: Amministratore Ora inizio: 12 Ora fine: 16 Ore scoperte: 0</p> <p>Ruolo: Ingegnere Ora inizio: 8 Ora fine: 12 Ore scoperte: 0</p> <p>Ruolo: IT Ora inizio: 8 Ora fine: 12 Ore scoperte: 0</p> <p>Ruolo: IT Ora inizio: 12 Ora fine: 16 Ore scoperte: 0</p> <p>Ruolo: Ingegnere Ora inizio: 12 Ora fine: 16 Ore scoperte: 3</p>	Indietro
---	----------

Figura 19: Visualizzazione turni di un giorno

13. Tramite un click su un turno si possono vedere tutti gli utenti assegnati a quel turno. In verde i dipendenti della stessa sede in orario ordinario, in giallo i dipendenti della stessa sede in straordinario e in blu quelli in trasferta.

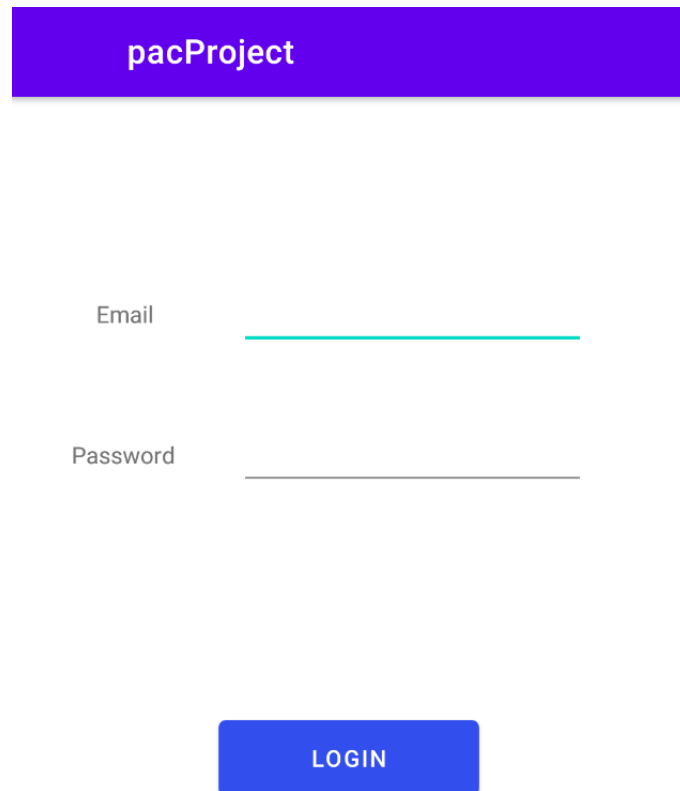
Profilo Azienda Sedi Ruoli Dipendenti Calendario Richieste ferie	Logout
--	--------

<p>Lire scoperte: 0</p> <p>Ruolo: Ingegnere Ora inizio: 12 Ora fine: 16 Ore scoperte: 3</p> <p>Nome: Enrico Cognome: Mantana Straordinario: false Trasferta: false Inizio: 12 Fine: 16</p> <p>Nome: Marco Cognome: Da Milano Straordinario: false Trasferta: false Inizio: 12 Fine: 16</p> <p>Nome: Paolo Cognome: Celata Straordinario: false Trasferta: false Inizio: 12 Fine: 16</p> <p>Nome: Sergio Cognome: Bechis Straordinario: false Trasferta: false Inizio: 12 Fine: 16</p> <p>Nome: Lucia Cognome: Annunziata Straordinario: true Trasferta: false Inizio: 12 Fine: 13</p>	
---	--

Figura 20: Visualizzazione dipendenti per turno

4.2 Applicazione Android - Dipendenti

1. Aprire il progetto tramite Android Studio, buildare e run dell'app su emulatore.
2. Apertura dell'applicazione sull'emulatore



The screenshot shows the initial login screen of an application named "pacProject". At the top, there is a purple header bar with the text "pacProject" in white. Below the header, there are two input fields: "Email" and "Password". The "Email" field has a light blue underline, and the "Password" field has a light purple underline. At the bottom of the screen, there is a blue button with the text "LOGIN" in white.

Figura 21: Pagina iniziale, login

3. Inserire credenziali dipendente e click sul bottone *LOGIN*

4. Accesso alle funzionalità dell'applicazione, viene mostrata la pagina per la richiesta di visualizzazione dei turni.



pacProject

LISTA TURNI


Seleziona primo giorno periodo

Seleziona ultimo giorno periodo

VISUALIZZA TURNI

Figura 22: Pagina iniziale dopo aver effettuato il login

5. Inserire le date di inizio e fine periodo per cui si vogliono visualizzare i turni.
6. Se le date selezionate sono corrette e sono presenti turni vengono mostrate le occorrenze.

 **pacProject**

LISTA TURNI

07/03/2022

15/03/2022

VISUALIZZA TURNI

2022-03-07

INIZIO Turno: 8 FINE Turno: 12

RUOLO: Amministratore

Via Portici Manarini, 41/A 24060 Chiuduno (BG)

2022-03-07

INIZIO Turno: 12 FINE Turno: 16

RUOLO: Amministratore

Via Portici Manarini, 41/A 24060 Chiuduno (BG)

Figura 23: Visualizzazione dei turni

7. Cambio funzione tramite menu, click su hamburger per apertura menu.

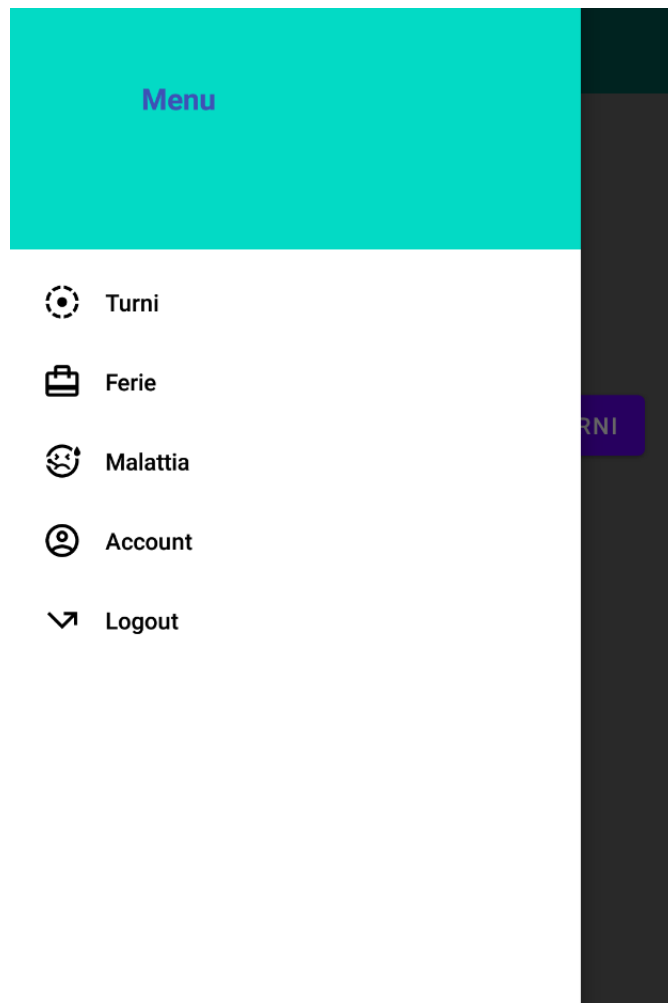


Figura 24: Menu per scelta funzione

8. Click sulla voce *Malattia*
9. Selezione delle date di inizio e fine periodo malattia

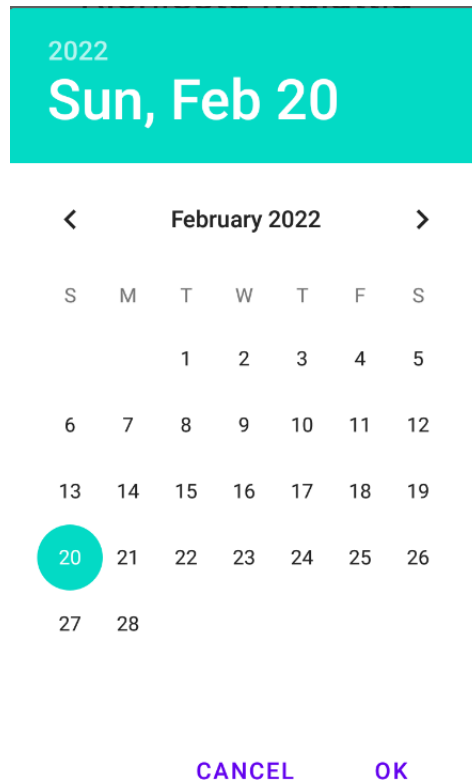


Figura 25: Calendario per la selezione delle date

10. Click sul bottone *INVIA RICHIESTA*
11. Viene mostrato l'esito della richiesta; in caso venga accettata:



Figura 26: Messaggio di accettazione della richiesta di malattia

12. Per effettuare il logout aprire il menu e cliccare sulla voce *Logout*. L'applicazione torna alla schermata di Login.