Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Assignment 3

Corso: Processo e Sviluppo del Software

Relazione di:
Davide Bucci 816067
Alessandro Capelli 816302

Anno Accademico 2019-2020

1 Link Repository

`https://gitlab.com/capelli.alessandro/2019_assignment3_softwaredevelopment`

2 Introduzione al sistema

Il focus del progetto è posto sullo sviluppo di un ipotetico back-end (è stata scelta l'implementazione di Java JPA-based back-end tramite Spring Data JPA) e di un front-end (tramite SpringMVC) di un sistema esistente, da collocare all'interno di un'organizzazione che necessita di tale infrastruttura. Entrando nello specifico, si è pensato di implementare un sistema back-end correlato da un semplice front-end a fronte di una richiesta esplicita da parte di un cliente, in questo caso la Croce Rossa Italiana (CRI), che ha la necessità di introdurre un sistema di gestione dei turni dei volontari. In particolare, si assume che la richiesta sia quella di sviluppare un sistema back-end (con un front-end per visualizzare i risultati) che permetta di gestire una serie di figure chiave (quali ad esempio i volontari e i veicoli coinvolti nei vari servizi, ecc...) presenti nella CRI, per semplificare l'attività di pianificazione dei turni e garantire un servizio continuo assegnando i veicoli appropriati alle relative sedi avendo la certezza di coprire tutte le zone interessate. Il dominio applicativo interessato è quindi quello relativo ai servizi gestionali erogati ed utilizzati internamente dalla CRI per ottimizzare il dispiegamento di risorse sul territorio.

Le entità coinvolte sono le seguenti e verranno trattate tecnicamente nella sezione 5:

- Persona: rappresenta un volontario che svolge dei turni all'interno dell'organizzazione
- Pilota: rappresenta un volontario che ricopre il ruolo di pilota, ha associato una patente speciale per la guida di mezzi di soccorso
- Soccorritore: rappresenta un volontario che ricopre il ruolo di soccorritore generico
- Medico: rappresenta un volontario che ha l'abilitazione di medico e che è specializzato in un settore specifico
- Sede: assumendo che ogni città abbia diverse sedi di soccorso, questa entità rappresenta una sede specifica
- Veicolo: rappresenta un generico mezzo di soccorso

Le richieste dell'organizzazione sono quelle di progettare e sviluppare un sistema back-end che permetta la gestione di oggetti in modo persistente, permettendo inoltre di effettuare le operazioni CRUD e di ricerca sugli stessi. Per la visualizzazione dei risultati è stato implementato un front-end che interagisce con il back-end grazie alle API messe a disposizione.

3 Implementazione

Per lo sviluppo del progetto è stato utilizzato il linguaggio Java e l'ambiente di sviluppo Eclipse. Data la necessità di garantire la persistenza delle informazioni, è stato scelto l'utilizzo di un database relazionale, gestito dal DBMS H2 che supporta SQL come linguaggio di interrogazione. Sono state utilizzate una serie di librerie e framework per l'implementazione efficiente delle varie componenti, in particolare:

- Maven: utilizzato per la gestione delle dipendenze (tramite il file pom.xml), per la distribuzione del pacchetto finale e per semplificare la condivisione del progetto su GitLab
- SpringBoot (con SpringWeb): include il web server Tomcat
- Spring Data JPA
- H2: utilizzato per la gestione del database relazionale
- Hibernate: è una delle implementazioni della specifica JPA (per l'ORM), quindi permette di mappare gli oggetti Java e le entità del database semplificando le attività necessarie a garantire la persistenza. Viene utilizzato implicitamente da Spring Data JPA
- Thymeleaf: per implementare il front-end
- JUnit 5: per l'implementazione dei tests

Il progetto consiste in un sistema back-end che ha alla base un server Tomcat (in ascolto sulla porta 8080 di localhost) ed un database relazionale (generato automaticamente da Hibernate grazie alle annotazioni). Il sistema offre delle API (descritte nella sezione 6) che permettono di effettuare le operazioni CRUD e di ricerca sulle entità gestite da parte del front-end; si ha quindi un sistema che utilizza il protocollo HTTP (tramite richieste GET/POST) per ricevere istruzioni ed, interpretando le richieste che vengono effettuate, è in grado di sfruttare Hibernate (tramite Spring Data JPA) per seguire la specifica JPA e garantire la persistenza. Data la natura del sistema, si è scelto di testare le componenti con richieste HTTP (scritte ad hoc) effettuate tramite comando "curl" e tramite l'implementazione di un front-end sfruttando Thymeleaf.

4 Organizzazione Progetto

Il progetto segue l'approccio MVC per la separazione dei compiti. L'organizzazione seguita è la seguente:

- file src/main/java/it/CroceRossaItaliana/ApplicazioneCRI.java: contiene il main
- cartella src/main/java/it/CroceRossaItaliana/model: contiene i modelli degli oggetti
- cartella src/main/java/it/CroceRossaItaliana/repository: contiene le interfacce necessarie a garantire la persistenza
- cartella src/main/java/it/CroceRossaItaliana/service: contiene le classi (implementazioni delle interfacce presenti in repository) necessarie a garantire la persistenza
- cartella src/main/java/it/CroceRossaItaliana/exception: contiene le eccezioni personalizzate
- cartella src/main/java/it/CroceRossaItaliana/controller: contiene le classi che si occupano di gestire le richieste HTTP e che permettono di esporre le API al front-end

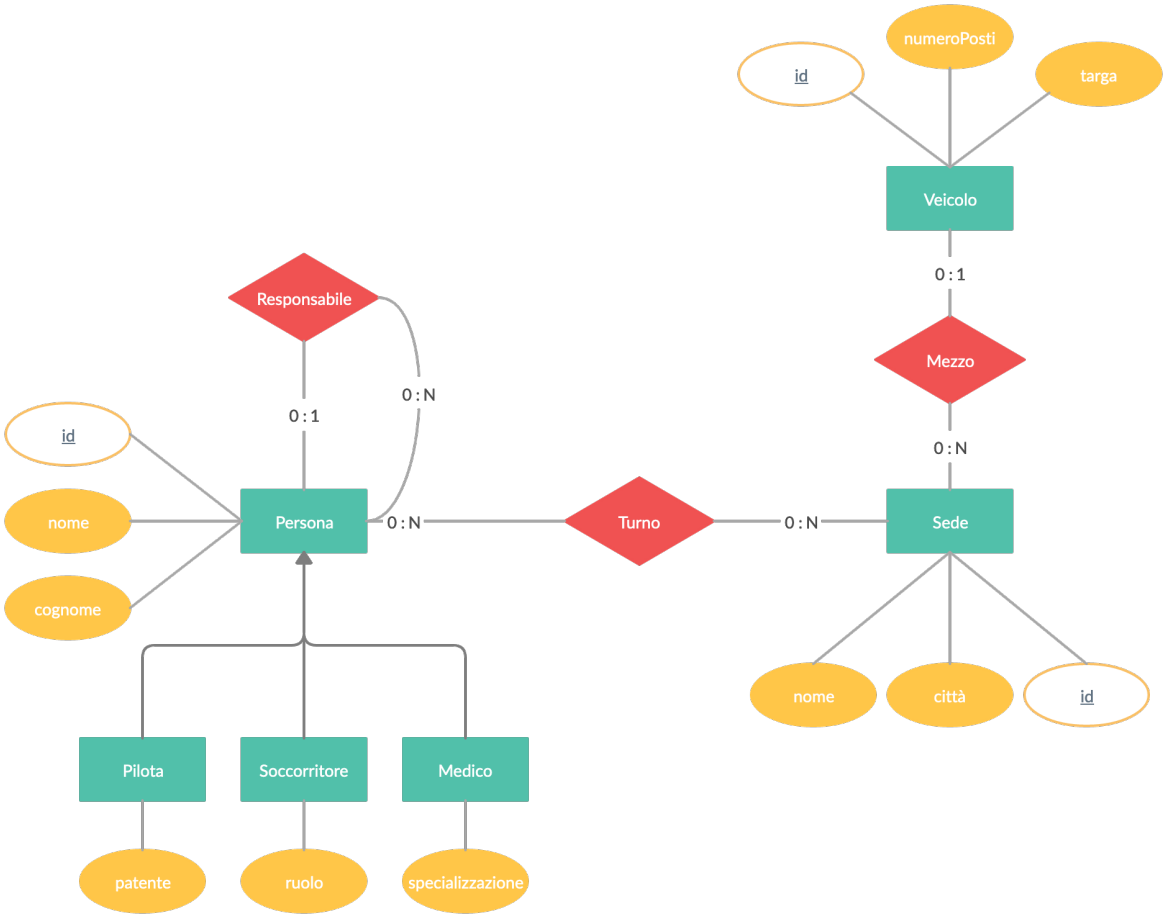
- cartella src/main/resources/templates: contiene il front-end scritto in HTML che utilizza le annotazioni Thymeleaf
- file application.properties: contiene le informazioni di configurazione del database H2 e del web server
- cartella src/test/java/it/CroceRossaItaliana: contiene le classi java utilizzate per il test dei services

5 Modello ER

Per lo sviluppo del modello ER associato al progetto sono state effettuate una serie di assunzioni:

- esistono tre differenti tipologie di persone all'interno dell'organizzazione modellata: Pilota, Soccorritore e Medico
- le persone possono avere una relazione con se stessi chiamata "Responsabile" che rappresenta una relazione in cui una persona può avere un responsabile e una persona può avere differenti persone subordinate (così da assegnare un ruolo). Si noti come è possibile non avere nessun responsabile assegnato
- le persone possono avere una relazione con le sedi chiamata "turno", che rappresenta il fatto che una persona presenzi all'interno di una o più sedi (viceversa, una sede può avere diverse persone che hanno dei turni nella sede stessa)
- i veicoli possono avere una relazione con le sedi chiamata "mezzo", che rappresenta il fatto che un veicolo sia il mezzo di al più una sede e che una sede può avere a disposizione più veicoli di soccorso

Il fatto che le cardinalità abbiano la possibilità di essere nulle, rappresenta il fatto che le entità modellate possono esistere anche in modo indipendente.



6 Mapping API

Le API messe a disposizione dal back-end sono le seguenti:

- Persona:
  - CREATE: tramite POST a /persona/create, richiede un form con le seguenti informazioni: nome, cognome. Viene restituita la pagina relativa alla persona appena creata.
  - READ:
    - tramite GET a /persona/readAll per leggere tutte le persone presenti nel DB. Viene restituita una pagina contenente tutte le persone.
    - tramite GET a /persona/read/{id} per leggere una persona in particolare, specificando l'id nella richiesta. Viene restituita la pagina relativa alla persona richiesta.
  - UPDATE: tramite POST a /persona/update, richiede un form con le seguenti informazioni: nome, cognome. Viene restituita la pagina relativa alla persona aggiornata.
  - DELETE: tramite GET a /persona/delete/{id}. Viene restituita una pagina delle persone presenti nel DB.
  - SEARCH:

    * tramite GET a /persona/search/nome?nome={nome} per cercare una persona in particolare, specificando il nome nella richiesta. Viene restituita una pagina contenente le persone cercate.
    * tramite GET a /persona/search/cognome?cognome={cognome} per cercare una persona in particolare, specificando il cognome nella richiesta. Viene restituita una pagina contenente le persone cercate.
  - READ della relazione responsabile: tramite GET a /persona/readAllResponsabili per leggere tutte le relazioni responsabile presenti nel DB. Viene restituita una pagina contenente tutte le relazioni di responsabile.
  - CREATE della relazione responsabile: tramite POST a /persona/create/responsabile, richiede un form con le seguenti informazioni: id_responsabile, id_subordinato. Viene restituita una pagina relativa alle relazioni di responsabile presenti nel DB.
  - DELETE della relazione responsabile: tramite GET a /persona/delete/responsabile/{id_subordinato} per eliminare la relazione di responsabile da una persona (viene inviato l'id della persona subordinata). Viene restituita una pagina contenente le relazioni di responsabile presenti nel DB a seguito della cancellazione.
  - READ della relazione turno: tramite GET a /persona/readAllTurni per leggere tutte le relazioni turno presenti nel DB. Viene restituita una pagina contenente tutte le relazioni di turno.
  - CREATE della relazione turno: tramite POST a /persona/create/turno, necessita di un form con le seguenti informazioni: id_persona, id_sede. Viene restituita una pagina relativa alle relazioni di turno presenti nel DB.
  - DELETE della relazione turno: tramite POST a /persona/delete/turno per eliminare la relazione di turno tra una persona e una sede, richiede un form con le seguenti informazioni: id_persona, id_sede. Viene restituita una pagina contenente le relazioni di turno presenti nel DB a seguito della cancellazione.

- Medico:
  - CREATE: tramite POST a /medico/create, richiede un form con le seguenti informazioni: nome, cognome, specializzazione. Viene restituita la pagina relativa al medico appena creato.
    * tramite GET a /medico/readAll per leggere tutti i medici presenti nel DB. Viene restituita una pagina contenente tutti i medici.
    * tramite GET a /medico/read/{id} per leggere un medico in particolare, specificando l'id nella richiesta. Viene restituita la pagina relativa al medico richiesto.
  - UPDATE: tramite POST a /medico/update, richiede un form con le seguenti informazioni: nome, cognome, specializzazione. Viene restituita la pagina relativa al medico aggiornato.
  - DELETE: tramite GET a /medico/delete/{id}. Viene restituita una pagina dei medici presenti nel DB.
  - SEARCH:

        * tramite GET a /medico/search/nome?nome={nome} per cercare un medico in particolare, specificando il nome nella richiesta. Viene restituita una pagina contenente i medici cercati.
        * tramite GET a /medico/search/cognome?cognome={cognome} per cercare un medico in particolare, specificando il cognome nella richiesta. Viene restituita una pagina contenente i medici cercati.
        * tramite GET a /medico/search/specializzazione?specializzazione={specializzazione} per cercare un medico in particolare, specificando la specializzazione nella richiesta. Viene restituita una pagina contenente i medici cercati.

- Pilota:
    - CREATE: tramite POST a /pilota/create, richiede un form con le seguenti informazioni: nome, cognome, patente. Viene restituita la pagina relativa al pilota appena creato.
        * tramite GET a /pilota/readAll per leggere tutti i piloti presenti nel DB. Viene restituita una pagina contenente tutti i piloti.
        * tramite GET a /pilota/read/{id} per leggere un pilota in particolare, specificando l'id nella richiesta. Viene restituita la pagina relativa al pilota richiesto.
    - UPDATE: tramite POST a /pilota/update, richiede un form con le seguenti informazioni: nome, cognome, patente. Viene restituita la pagina relativa al pilota aggiornato.
    - DELETE: tramite GET a /pilota/delete/{id}. Viene restituita una pagina dei piloti presenti nel DB.
    - SEARCH:
        * tramite GET a /pilota/search/nome?nome={nome} per cercare un pilota in particolare, specificando il nome nella richiesta. Viene restituita una pagina contenente i piloti cercati.
        * tramite GET a /pilota/search/cognome?cognome={cognome} per cercare un pilota in particolare, specificando il cognome nella richiesta. Viene restituita una pagina contenente i piloti cercati.
        * tramite GET a /pilota/search/patente?patente={patente} per cercare un pilota in particolare, specificando la patente nella richiesta. Viene restituita una pagina contenente i piloti cercati.

- Soccorritore:
    - CREATE: tramite POST a /soccorritore/create, richiede un form con le seguenti informazioni: nome, cognome, ruolo. Viene restituita la pagina relativa al soccorritore appena creato.
        * tramite GET a /soccorritore/readAll per leggere tutti i soccorritori presenti nel DB. Viene restituita una pagina contenente tutti i soccorritori.
        * tramite GET a /soccorritore/read/{id} per leggere un soccorritore in particolare, specificando l'id nella richiesta. Viene restituita la pagina relativa al soccorritore richiesto.

  - UPDATE: tramite POST a /soccorritore/update, richiede un form con le seguenti informazioni: nome, cognome, ruolo. Viene restituita la pagina relativa al soccorritore aggiornato.
  - DELETE: tramite GET a /soccorritore/delete/{id}. Viene restituita una pagina dei soccorritori presenti nel DB.
  - SEARCH:
    * tramite GET a /soccorritore/search/nome?nome={nome} per cercare un soccorritore in particolare, specificando il nome nella richiesta. Viene restituita una pagina contenente i soccorritori cercati.
    * tramite GET a /soccorritore/search/cognome?cognome={cognome} per cercare un soccorritore in particolare, specificando il cognome nella richiesta. Viene restituita una pagina contenente i soccorritori cercati.
    * tramite GET a /soccorritore/search/ruolo?ruolo={ruolo} per cercare un soccorritore in particolare, specificando il ruolo nella richiesta. Viene restituita una pagina contenente i soccorritori cercati.

- Veicolo:
  - CREATE: tramite POST a /veicolo/create, richiede un form con le seguenti informazioni: numeroPosti, targa. Viene restituita una pagina contenente le informazioni sul veicolo creato.
  - READ:
    * tramite GET a /veicolo/readAll per leggere tutti i veicoli presenti nel DB. Viene restituita una pagina contenente tutti i veicoli.
    * tramite GET a /veicolo/read/{id} per leggere un veicolo in particolare, specificando l'id nella richiesta. Viene restituita una pagina contenente le informazioni sul veicolo richiesto.
  - UPDATE: tramite POST a /veicolo/update, richiede un form con le seguenti informazioni: numeroPosti, targa. Viene restituita una pagina contenente le informazioni sul veicolo aggiornato.
  - DELETE: tramite GET a /veicolo/delete/{id}. Viene restituita una pagina contenente i veicoli presenti nel DB a seguito della cancellazione.
  - SEARCH: tramite GET a /veicolo/search/targa?targa={targa} per cercare un veicolo in particolare, specificando la targa nella richiesta. Viene restituita una pagina contenente i veicoli cercati.
  - READ della relazione mezzo: tramite GET a /veicolo/readAllMezzi per leggere tutte le relazioni mezzo presenti nel DB. Viene restituita una pagina contenente tutte le relazioni di mezzo.
  - CREATE della relazione mezzo: tramite POST a /veicolo/create/mezzo, necessita di un form con le seguenti informazioni: id_veicolo, id_sede. Viene restituita una pagina relativa alle relazioni di mezzo presenti nel DB.
  - DELETE della relazione mezzo: tramite GET a /veicolo/delete/mezzo{id_vei colo} per eliminare la relazione di mezzo tra un veicolo e una sede. Viene restituita una pagina contenente le relazioni di mezzo presenti nel DB a seguito della cancellazione.

- Sede:
  - CREATE: tramite POST a /sede/create, richiede un form con le seguenti informazioni: nome, città. Viene restituita una pagina contenente le informazioni sulla sede creata.
  - READ:
    * tramite GET a /sede/readAll per leggere tutte le sedi presenti nel DB. Viene restituita una pagina contenente tutte le sedi.
    * tramite GET a /sede/read/{id} per leggere una sede in particolare, specificando l'id nella richiesta. Viene restituita una pagina contenente le informazioni sulla sede richiesta.
  - UPDATE: tramite POST a /sede/update, richiede un form con le seguenti informazioni: nome, città. Viene restituita una pagina contenente le informazioni sulla sede aggiornata.
  - DELETE: tramite GET a /sede/delete/{id}. Viene restituita una pagina contenente le sedi presenti nel DB a seguito della rimozione.
  - SEARCH:
    * tramite GET a /sede/search/nome?nome={nome} per cercare una sede in particolare, specificando il nome nella richiesta. Viene restituita una pagina contenente le sedi cercate.
    * tramite GET a /sede/search/citta?citta={citta} per cercare una sede in particolare, specificando la città nella richiesta. Viene restituita una pagina contenente le sedi cercate.

7 Esecuzione

7.1 Packaging

Il packaging dell'applicazione viene effettuato tramite il seguente comando:
./mvnw clean package spring-boot:repackage

7.2 Esecuzione

L'esecuzione dell'applicazione è effettuata tramite il seguente comando:
java -Djava.security.egd=file:/dev/./urandom -jar target/CRI.jar

L'applicazione in esecuzione è disponibile all'indirizzo locale: http://localhost:8080