POLITECNICO DI BARI

PROGETTO DI INGEGNERIA DEL SOFTWARE

“Comune Pulito”

A cura degli studenti: IS44

- Colia Giuseppe
- Ciccolella Giuseppe
- Deflorio Andrea
- Misceo Lorenzo

Docenti: Poliba

- Mongiello Marina
- Ing. Fiore Marco

Azienda: Accenture

- Castagna Ivan Damien
- Bertolini Stefania
- Orritos Nicola
- Meloni Marco
- Ciaccia Alessio

FASE DI IDEAZIONE

Abstract

“Nell’ambito dei fondi stanziati per lo sviluppo digitale sul territorio, un’azienda che opera nel settore della raccolta rifiuti, vuole realizzare un’applicazione per il caricamento di segnalazioni sull’abbandono dei rifiuti, incuria e problematiche generali di tema ambientale nel proprio comune.”

Inizio Progettazione

Seguendo i consigli della traccia, abbiamo individuato i seguenti requisiti funzionali:

1) Pubblicazione delle segnalazioni dei rifiuti da parte degli utenti, specificandone la posizione e dettagli.
2) Visualizzazione delle segnalazioni dei rifiuti in tempo reale.
3) Gestione delle segnalazioni, effettuata dall’azienda commissionatrice del progetto.

Successivamente, abbiamo deciso di includere alcune funzionalità aggiuntive:

1) Raggruppamento delle segnalazioni all’interno di Cluster, per semplificare la gestione di quest’ultime.
2) Visualizzazione su mappa delle segnalazioni.
3) Sistema di reward per premiare gli utenti che utilizzano l’applicazione.(Attualmente incompleto)
    - Questo implica l’inserimento di un sistema di penalizzazione per gli utenti che si comportano in maniera scorretta(segnalazioni false, spoofing, ecc…).

Strutturazione Tech Stack

Sulla base delle funzionalità individuate, abbiamo scelto di strutturare il tech stack come segue:

- DB Relazionale con MySQL
- API-RESTful con Spring Boot
- Client Web basato su Angular

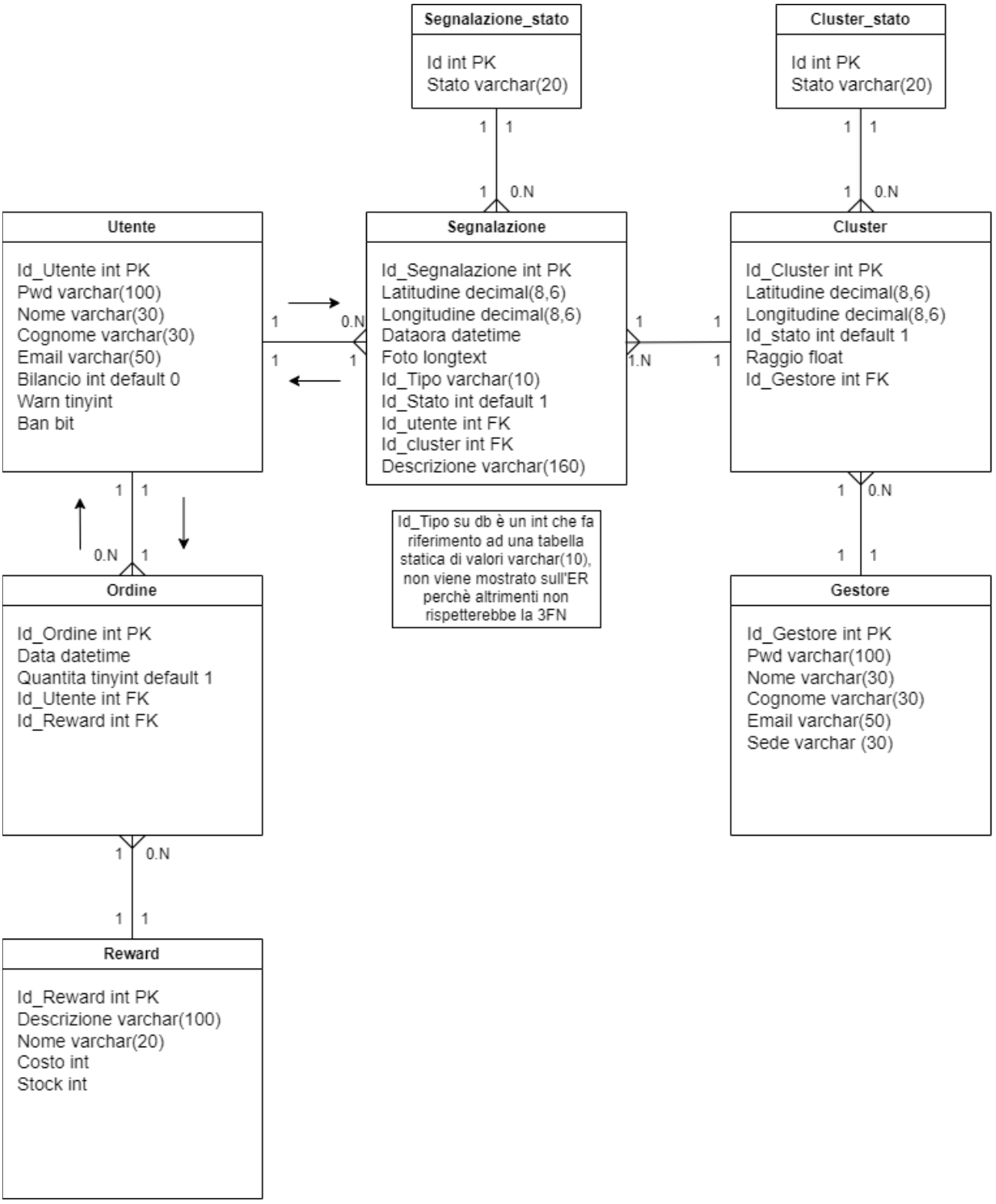
Dividendoci i compiti come segue:

- Colia Giuseppe: Fullstack Developer, lavora principalmente sulla parte di logica del frontend. Project Manager.
- Ciccolella Giuseppe: Fullstack Developer, lavora principalmente alla parte grafica del FE.
- Deflorio Andrea: Backend Developer, Designer del logo.
- Misceo Lorenzo: Backend Developer, Referente all'azienda.

Organizzati secondo questi tempi:

ID	Descrizione	Responsabile	Stima *	Inizio	Fine
PRO01	Diagramma delle Classi	Tutti	10	1-nov	10-nov
PRO02	Diagramma ER	Tutti	10	1-nov	10-nov
PRO03	Grafico delle Pagine	Tutti	10	3-nov	17-nov
REF01	Spring Security	Giuseppi	2	18-nov	25-nov
REF02	Query SQL JPA	Giuseppi	2	21-nov	23-nov
FUN01	Hash Password in FE	Giuseppe Co	0	24-nov	24-nov
REF03	Login con POST(FE/BE)	BE(AL),FE(GG)	7	25-nov	1-dic
FUN02	Controllo Fallimento Login e Signup	Giuseppe Co	1	8-dic	9-dic
EST01	Layout Pagina login e Signup (Form + Sfondo)	Giuseppe Ci	1	10-dic	11-dic
EST02	Mappa in FE	Giuseppi	2	12-dic	14-dic
FUN03	Gestione Segnalazioni in BE	Andrea e Lorenzo	2	15-dic	17-dic
FUN04	Gestione sessione utente	Gluseppi	1	18-dic	19-dic
FUN05	Pagina Segnalazioni	Giuseppe Co	2	20-dic	22-dic
EST03	Layout pagina mappa (Offcanvas + Info)	Giuseppe	3	27-dic	30-dic
FUN06	Gestione Cluster in BE	Andrea	3	2-gen	5-gen
FUN07	Gestione Reward in BE	Lorenzo	2	3-gen	5-gen
FUN08	Gestione Ordine BE	Andrea	1	6-gen	6-gen
REF04	Creazione entità sul database	Lorenzo	1	8-gen	8-gen
FUN09	Cluster FE	Giuseppi	2	7-gen	9-gen
EST04	Layout pagina segnalazioni	Giuseppe Co	1	7-gen	9-gen
EST05	Layout pagina Cluster	Giuseppe Ci	1	8-gen	10-gen
FUN10	Pagina Reward (DEMO)	Gluseppi	1	9-gen	11-gen
EST06	Ultimi ritocchi	Tutti	7	12-gen	19-gen

Diagramma E-R

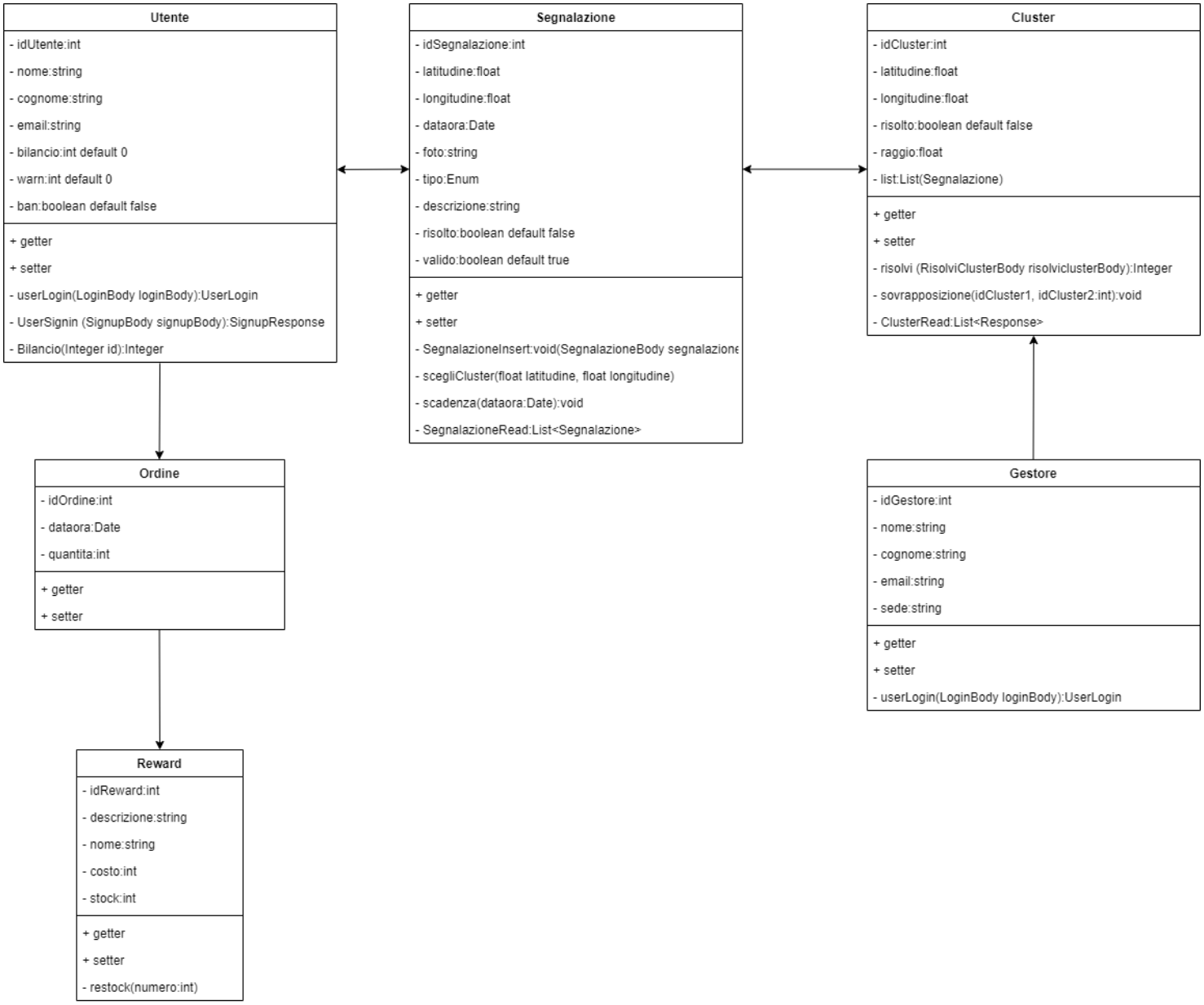


Entità individuate

1) **Utente:** Effettua le segnalazioni inserendo tipo, foto ,descrizione ed in seguito alla loro risoluzione, ottiene punti scambiabili nella sezione reward.
   - **Attributi:** Id_utente, email, nome, cognome, password, bilancio, warn, ban.
2) **Segnalazioni:** Contengono le informazioni utili inserite dall'utente (foto, tipo rifiuto e descrizione) per pulire l'area segnalata, una volta effettuate si aggregano nel cluster più vicino.
   - **Attributi:** Id_segnalazione, posizione, Foto, tipo, descrizione, data, Id_utente, id_stato, id_cluster.
3) **Cluster:** Contengono le segnalazioni effettuate dall'utente, saranno visibili e risolvibili dal gestore.
   - **Attributi:** Id_cluster, posizione, raggio, Id_stato, IdGestore.
4) **Gestore:** Il responsabile della gestione dei cluster, verifica l'attendibilità delle segnalazioni.
   - **Attributi:** Id_Gestore, password, nome, cognome, email, sede.
5) **Ordine:** Entità utile ad evitare problemi di transazione. (Su DB non è stata implementata, verrà aggiunta in versioni future.)
   - **Attributi:** Id_Ordine, data, quantità, Id_Utente, Id_Reward.
6) **Reward:** Ricompensa che l'utente può riscattare utilizzando i punti ricevuti dalle segnalazioni.
   - **Attributi:** Id_Reward, descrizione, nome, costo, stock.

Classi

In questa sezione saranno spiegate più nel dettaglio le classi utilizzate per la realizzazione del progetto, rappresentate all'interno del Diagramma di Classi:



Classi relative alla comunicazione con il database:

Le classi che terminano in "-Repository", fanno uso dell'interfaccia "CrudRepository" di Spring JPA, che permette di interpretare delle tabelle su DB come se fossero delle strutture dati in Java, velocizzando molto il processo di creazione delle Query, che viene automatizzato da Spring con l'utilizzo dell'Annotation "@Query", che implementa il funzionamento della Query all'interno del metodo. L'interfaccia permette inoltre di applicare modifiche al DB in maniera automatica. Le classi che implementano questa funzionalità sono:

- **UtenteRepository:** Classe utile alla comunicazione tra utente e DB. Contiene le Query di:
    - Ricerca dell'Utente in base alla sua Email:
      Optional<Utente> findByEmail(String email);
- **SegnalazioneRepository:** Classe utile alla comunicazione tra Segnalazione e DB. Contiene le Query di:
    - Ricerca della Segnalazione in base al suo Id_Segnalazione:
      Optional<Segnalazione> findById(Integer utente);
- **ClusterRepository:** Classe utile alla comunicazione tra cluster e DB. Contiene le Query di:
    - Ricerca del Cluster in base al suo Id_Cluster:
      Optional<Cluster> findById(Integer id);
    - Ricerca dei Cluster gestiti da un determinato Gestore(Id_Gestore):
      @Query(value="SELECT c FROM Cluster c WHERE c.Id_gestore=?1")
      List<Cluster> findClusterByIdGestore(Integer idGestore);
    - Ricerca di tutte le Segnalazioni relative ad un Cluster(Id_Cluster):
      @Query(value="SELECT s FROM Segnalazione s LEFT JOIN Cluster c ON s.Id_cluster=c.Id_cluster WHERE c.Id_cluster=?1")
      List<Segnalazione> findSegnalazioneCluster(Integer idCluster);

- **GestoreRepository:** Classe utile alla comunicazione tra Gestore e DB. Contiene le query di:
    - Ricerca del Gestore in base alla sua Email:
      Optional<Gestore> findByEmail(String email);
    - Ricerca dei Gestori ordinati in base al numero di Cluster assegnati:
      @Query(value="SELECT g.Id_gestore FROM Gestore g LEFT JOIN Cluster c ON g.Id_gestore=c.Id_gestore GROUP BY g.Id_gestore ORDER BY COUNT(*)")
      List<Integer> findGestoreLibero();

Classi di comunicazione tra FE e BE:

Queste classi permettono di interpretare i dati ricevuti nel body delle richieste POST inviate dal FE.

- **SignUpBody:** Utilizzata per interpretare il body della richiesta di registrazione di un nuovo utente. Contiene gli attributi:
    - private String email;
    - private String password;
    - private String nome;
    - private String cognome;
- **LoginBody:** Utilizzata per interpretare il body della richiesta di accesso da parte di un utente registrato. Contiene gli attributi:
    - private String email;
    - private String password;
    - private boolean isGestore; (indica se l'utente che sta tentando l'accesso è un gestore o no)
- **SegnalazioneBody:** Utilizzata per interpretare il body della richiesta di pubblicazione di una segnalazione da parte di un determinato utente. Contiene gli attributi:
    - private String foto;
    - private String descrizione;
    - private Integer tipo_rifiuto;
    - private Float latitudine;
    - private Float longitudine;
    - private Integer utente;

- **ClusterIdBody:** Utilizzata per interpretare il body della richiesta di visualizzazione di tutti i cluster relativi ad un determinato Gestore. Contiene gli attributi:
    - private Integer gestore;
- **RisolviClusterBody:** Utilizzata per interpretare il body della richiesta di Risoluzione di un determinato cluster con le relative segnalazioni. Contiene gli attributi:
    - private Cluster cluster;
    - private List<Segnalazione> segnalazioni;
- **UserLogin:** Utilizzata per codificare la risposta di una richiesta di login da parte del FE. Contiene gli attributi:
    - private boolean valido;
    - private String token;(è l'Id_Utente dell'utente che si è loggato)
    - private String nome;
    - private String cognome;
- **SignupResponse:** Utilizzata per codificare la risposta ad una richiesta di signup dal FE. Contiene gli attributi:
    - private boolean SignedUp;
    - private UserLogin userLogin;

Classi relative al funzionamento della web app:

Queste classi allestiscono i servizi API RESTful in ascolto sulle porte del backend. Svolgono la maggior parte delle funzionalità dell'applicazione, ed in particolare sono:

- **UserLoginController:** Ascolta sull'entry point "/login". Riceve nel body della richiesta POST un dato di tipo LoginBody, e controlla che l'Utente (o Gestore) sia presente su DB, e che la password inserita sia corretta. Se ha successo, restituisce come risposta un dato di tipo UserLogin sotto forma di JSON. In caso contrario restituisce un errore "401_UNAUTHORIZED"
- **UserSignUpController:** Ascolta sull'entry point "/signup". Riceve nel body della richiesta POST un dato di tipo SignupBody, e controlla che l'Utente (o Gestore) non sia presente su DB, quindi lo registra sul DB. Se ha successo, restituisce come risposta un dato di tipo SignupResponse sotto forma di JSON. In caso contrario restituisce un errore HTTP "401_UNAUTHORIZED"
- **SegnalazioneAddController:** Ascolta sull'entry point "/segnalazione". Riceve nel body della richiesta POST un dato di tipo

SegnalazioneBody,. Se ha successo, restituisce come risposta un dato di tipo UserLogin sotto forma di JSON.

- **GeoJSONParserGetter:** Ascolta sull'entry point "/geojson". Recupera le segnalazioni dal DB e li consegna al FE in formato GeoJSON(Richiesta GET).
- **ClusterReadController:** Ascolta sull'entry point "/mappaGestore". Legge i dati dei Cluster relativi ad un determinato Id_Gestore, e li restituisce al FE sotto forma di lista.
- **RisolviClusterController:** Ascolta sull'entry point "/convalida". Riceve una richiesta POST, di tipo RisolviClusterBody, che Convalida un determinato Cluster, e gestisce le relative segnalazioni.
- **BilancioController:** Ascolta sull'entry point "/bilancio". Restituisce il bilancio dell'utente tramite richiesta GET con parametro id, che è l'Id_Utente dell'utente che sta richiedendo il bilancio.