TEAM
COMMITTED

Università degli Studi di Padova

Definizione di prodotto V1.0

Informazioni sul documento

Nome documento	Definizione di prodotto
Data documento	GG/MM/AAAA
Redattori	Giorgio Maggiolo
Verificatori	xxxxxx
Approvazione	xxxxxx
Uso documento	Interno
Lista distribuzione	• *Team Committed* • *Prof. Tullio Vardanega*

Sommario

xxxxxxxxxxxx

Diario delle modifiche

Modifica	Autore	Data	Versione
Redatto il capitolo 2	Giorgio Maggiolo	2012/02/16	V0.2
Creati i sorgenti per il file. Redatto il capitolo 1	Giorgio Maggiolo	2012/02/15	V0.1

Indice

1 Introduzione

1.1 Scopo del documento

Il documento di *Definizione di Prodotto* descrive tutte le componenti del sistema e il modo in cui esse collaborano. Per ogni componente vengono illustrati gli attributi con il loro significato e i medoti con il loro comportamento.

Gli aspetti più delicati sono rappresentati anche attraverso diagrammi di sequenza, in modo da mostrare in modo chiaro l'ordine delle operazioni e l'interazione delle compoenenti.

Lo scopo principale del documento è quello di fornire ai programmatori una solida e precisa guida alla codifica, in modo che essi possano lavorare in modo autonomo attenendosi alle scelte progettuali ed evitando soluzioni personali ed improvvisate.

1.2 Scopo del prodotto

Il prodotto denominato **SafetyGame** si propone di fornire uno strumento informatico per la gestione delle pratiche di sicurezza sul lavoro in modo dinamico, evitando corsi di formazione che spesso si dimostrano inutili per la poca attenzione prestata dai partecipanti.

Lo strumento si basa sul concetto di **gamification** che comporta competizione tra i dipendenti all'interno delle aziende creando un sano interesse per un argomento delicato come la sicurezza sul luogo di lavoro.

Il sistema è pensato sia per lavoratori che hanno una postazione fissa dotata di PC, sia per quelli che hanno la necessità di spostarsi e che quindi sono forniti di dispositivi mobili. Ad essi verranno poste periodicamente domande, di varia tipologia, le cui risposte comporteranno l'assegnazione di un punteggio generando una classifica aziendale.

1.3 Ambiguità

Al fine di evitare ogni ambiguità relativa al linguaggio e ai termini utilizzati nei documenti formali, il glossario viene incluso nel file `Glossario-V3.0.pdf`, dove vengono definiti e descritti i termini marcati da una sottolineatura.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Norme di Progetto, v 3.0** (allegato `Norme_di_Progetto_V3.0.pdf`)
- **Analisi dei Requisiti, v 3.0** (allegato `Analisi_dei_Requisiti_V3.0.pdf`)

1.4.2 Riferimenti informativi

- **Elementi di Gamification, V1.0** (allegato `Elementi_Gamification_V1.0.pdf`)

2 Standard di progetto

2.1 Standard di progettazione architetturale

Per gli standard di progettazione architetturale, si veda il documento *Specifica Tecnica - V 2.0*.

2.2 Standard di documentazione del codice

La documentazione del codice sarà prodotta come Javadoc. Per ulteriori informazioni si vedano le *Norme di Progetto - V 3.0*.

2.3 Standard di denominazione di entità e relazioni

Entità e relazioni devono avere nomi chiari, coincisi ed esplicativi. Per ulteriori informazioni si faccia riferimento alle *Norme di Progetto - V 3.0*.

2.4 Standard di programmazione

Per gli standard di programmazione si vedano le *Norme di Progetto - V 3.0*.

2.5 Strumenti di lavoro

Per gli strumenti di lavoro si faccia riferimento alle *Norme di Progetto - V 3.0*.

3 Specifica Back-end

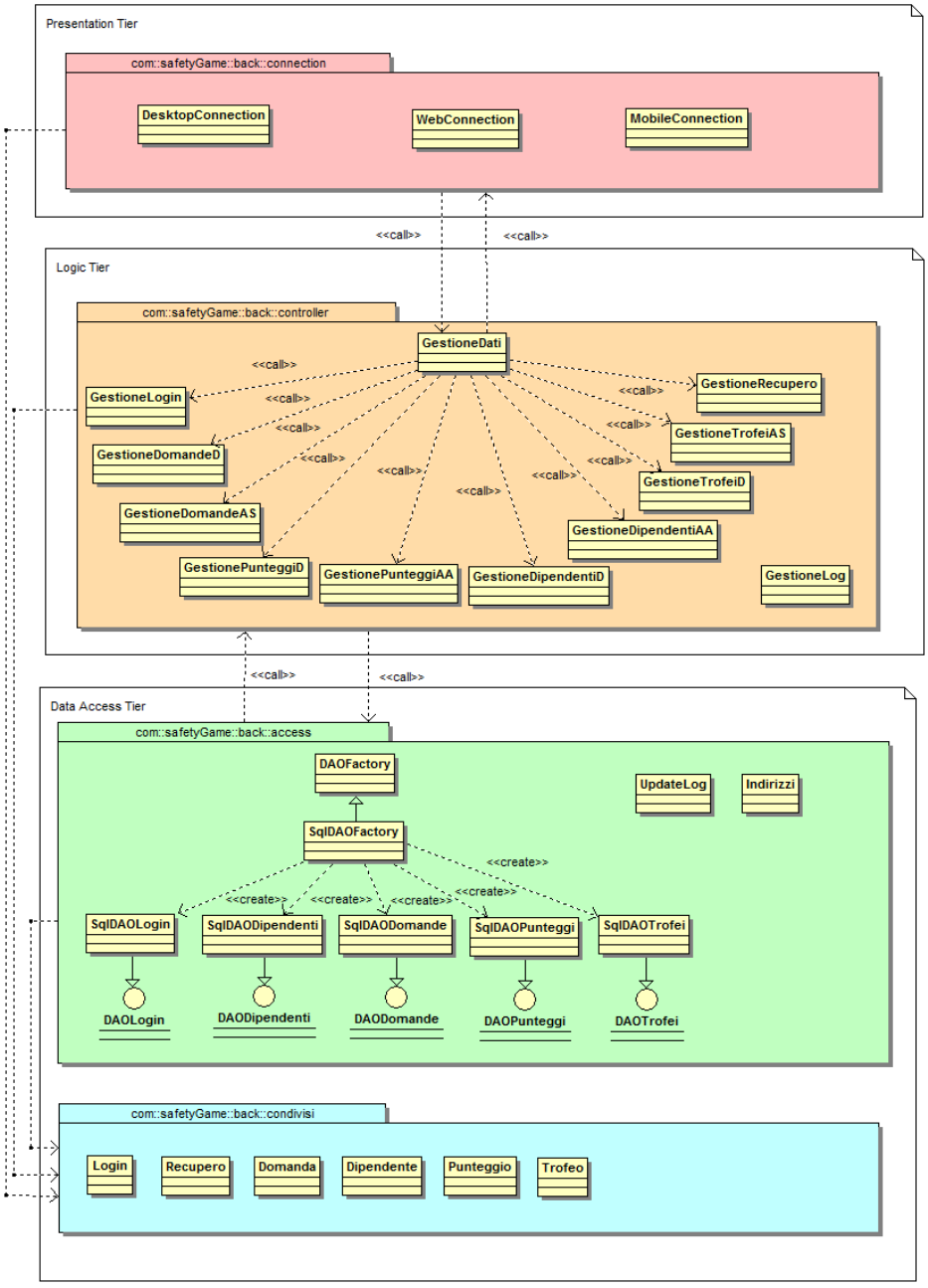

Figura 1: *Back-end, diagramma dei package*

3.1 Package com.safetyGame.back.connection

Figura 2: *Package* ***com.safetyGame.back.connection*** *e le sue relazioni*

Tipo, obiettivo e funzione del componente: contiene gli oggetti deputati all'interfacciamento dei vari front-end con il back-end
Relazioni d'uso di altre componenti: utilizza la classe **com.safetyGame-.back.controller.GestioneDati** per inoltrare le richieste che provengono dai vari front-end verso gli strati inferiori del back-end
Interfacci con e relazioni d'uso da altre componenti: viene utilizzato dalla classe **com.safetyGame.back.controller.GestioneDati** per inoltrare le elaborazioni compiute dagli strati inferiori del back-end verso i vari front-end
Attività svolte e dati trattati: fa da connettore fra i vari front-end e il back-end, divenendo il componente *Presenter* del design pattern *MVP*

Il seguente diagramma delle classi fornisce una panoramica di tutte le classi che compongono il package.

Figura 3: *Diagramma delle classi*

3.1.1 com.safetyGame.back.connection.WebConnection

Funzione

Questa classe si occuperà di presentare al front-end web tutte le funzionalità decise in fase di Progettazione Architetturale a cui ha accesso

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- GestioneDati dati

Metodi

+ WebConnection(GestioneDati d)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ boolean loginDip(String username, String password)

Metodo per il login dei dipendenti.

Dovrà creare un oggetto *Login* a partire dai parametri passati e quindi chiamare la funzione della classe Façade deputata alla gestione del login.

+ boolean loginAdmin(String username, String password)

Metodo per il login degli utenti amministratori.

Dovrà creare un oggetto *Login* a partire dai parametri passati e quindi chiamare la funzione della classe Façade deputata alla gestione del login.

+ Dipendente getDati(Login l)

Metodo che consente di reperire le informazioni di un dipendente a partire dal suo login.

Dovrà chiamare la funzione della classe Façade deputata al recupero dei dati di un *Dipendente*

+ Punteggio getPunteggio(Login l)

Metodo che consente di recuperare le statistiche di un determinato dipendente.

Dovrà chiamare la funzione della classe Façade deputata al recupero del punteggio di un *Dipendente*

+ Punteggio getStat(Login l)

Metodo che consente di recuperare le statistiche globali

Dovrà chiamare la funzione della classe Façade deputata al recupero delle statistiche dell'intera azienda

+ ArrayList<Badge> getBadge(Login l, int n)

Metodo per ottenere i dati dei badge per un dato utente
Dovrà chiamare la funzione della classe Façade deputata al recupero dei primi n *Badge* guadagnati da un *Dipendente*

+ **boolean modPassD(Dipendente dip)**
Metodo che consente la modifica della password da parte di un dipendente
Dovrà chiamare la funzione della classe Façade deputata alla modifica della password di un *Dipendente*

+ **boolean modPassA(Dipendente dip)**
Metodo che consente la modifica della password da parte di un amministratore
Dovrà chiamare la funzione della classe Façade deputata alla modifica della password di un *Amministratore*

+ **boolean modMail(Dipendente d, String mail)**
Metodo che consente la modifica della mail da parte di un dipendente
Dovrà chiamare la funzione della classe Façade deputata alla modifica dell'indirizzo email di un *Dipendente*

+ **boolean resetPassD(Recupero r)**
Metodo che consente la rigenerazione della password per un dipendente
Dovrà chiamare la funzione della classe Façade deputata alla rigenerazione della password di un *Dipendente*

+ **boolean resetPassA(Recupero r)**
Metodo che consente la rigenerazione della password per un amministratore
Dovrà chiamare la funzione della classe Façade deputata alla modifica della password di un *Amministratore*

+ **Domanda mostraDomanda(Login l)**
Metodo che consente di recuperare una domanda
Dovrà chiamare la funzione della classe Façade deputata al recupero di una nuova domanda da sottoporre ad un *Dipendente*

+ **boolean setRisposta(Login l,Domanda risposta)**
Metodo che si occupa di comunicare la risposta data
Dovrà chiamare la funzione della classe Façade deputata al controllo della correttezza della risposta data ad una *Domanda* da un *Dipendente*

+ **boolean posticipa(Login l, Domanda d)**
Metodo per posticipare una domanda
Dovrà chiamare la funzione della classe Façade deputata al posticipo della *Domanda* sottoposta ad un *Dipendente*

+ **void logout(Login l)**

Metodo per segnalare al sistema il logout di un utente
Dovrà chiamare la funzione della classe Façade deputata alla gestione della richiesta di logout da parte di un utente

+ **ArrayList<Domanda> getElencoDomande()**
Metodo per ottenere la lista di tutte le domande
Dovrà chiamare la funzione della classe Façade deputata alla generazione dell'elenco di tutte le domande selezionate dall'Amministratore Sicurezza per l'azienda

+ **boolean aggiungiDomanda(Domanda d)**
Metodo per inserire una domanda dal server domande al server dell'azienda
Dovrà chiamare la funzione della classe Façade deputata all'inserimento di una nuova domanda all'interno dell'elenco delle domande selezionate dall'Amministratore Sicurezza per l'azienda

+ **boolean cancellaDomanda(Domanda d)**
Metodo per eliminare una domanda dal server dell'azienda
Dovrà chiamare la funzione della classe Façade deputata alla rimozione di una domanda all'interno dell'elenco delle domande selezionate dall'Amministratore Sicurezza per l'azienda

+ **ArrayList<Dipendente> getElencoDipendenti()**
Metodo per ottenere i dati dei dipendenti dell'azienda
Dovrà chiamare la funzione della classe Façade deputata al recupero dell'elenco di tutti i *Dipendenti* registrati nel sistema

+ **boolean setTrofei(Dipendente d, int n)**
Metodo per modificare il numero di trofei di un dipendente
Dovrà chiamare la funzione della classe Façade deputata alla modifica del numero di trofei di un *Dipendente*

+ **boolean aggiungiDipendente(Dipendente d)**
Metodo per aggiungere un dipendente
Dovrà chiamare la funzione della classe Façade deputata all'aggiunta di un *Dipendente* all'interno del database

+ **boolean cancellaDipendente(Dipendente d)**
Metodo per eliminare un Dipendente
Dovrà chiamare la funzione della classe Façade deputata alla rimozione di un *Dipendente* all'interno del database

+ **boolean modDipendente(Dipendente dOld, Dipendente dNew)**
Metodo per modificare i dati di un dipendente
Dovrà chiamare la funzione della classe Façade deputata alla modifica di un *Dipendente* all'interno del database

+ **ArrayList<Badge> getBadgesAS()**
Metodo per ottenere tutti i badge
Dovrà chiamare la funzione della classe Façade deputata al recupero di tutti i *Badge* inseriti nel database

+ **boolean assegnaBadge(Domanda D, Login l)**
Metodo per assegnare un badge
Dovrà chiamare la funzione della classe Façade deputata all'assegnazione di un *Badge* ad un *Dipendente*

+ **ArrayList<Dipendente> getPunteggi()**
Metodo per ottenere i punteggi medi dell'azineda e i punteggi di tutti i dipendenti
Dovrà chiamare la funzione della classe Façade deputata al recupero dei punteggi di tutti i *Dipendenti*, oltre che ai punteggi medi dell'azienda.

3.1.2 com.safetyGame.back.connection.MobileConnection

Funzione

Questa classe si occuperà di presentare al front-end mobile e desktop tutte le funzionalità decise in fase di Progettazione Architetturale a cui ha accesso

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- GestioneDati dati

- Parser parser

Metodi

+ ApplicazioniConnection(GestioneDati d, Parser p)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ boolean login(String username, String password)

Metodo per il login.

Dovrà creare un oggetto *Login* a partire dai parametri passati e quindi chiamare la funzione della classe Façade deputata alla gestione del login. Poichè l'uso del front-end mobile e desktop è riservato ai *Dipendenti*, questo metodo dovrà chiamare il metodo per il login dei *Dipendenti*

+ Dipendente getDati(Login l)

Metodo che consente di reperire le informazioni di un dipendente a partire dal suo login.

Dovrà chiamare la funzione della classe Façade deputata al recupero dei dati di un *Dipendente*

+ Punteggio getStat(Login l)

Metodo che consente di recuperare le statistiche globali

Dovrà chiamare la funzione della classe Façade deputata al recupero delle statistiche dell'intera azienda

+ void modPass(Dipendente d)

Metodo che consente la modifica della password da parte di un dipendente

Dovrà chiamare la funzione della classe Façade deputata alla modifica della password di un *Dipendente*

+ void modMail(Dipendente d, String mail)

Metodo che consente la modifica della mail da parte di un dipendente

Dovrà chiamare la funzione della classe Façade deputata alla modifica dell'indirizzo email di un *Dipendente*

+ **void resetPass(Recupero r)**
Metodo che consente la rigenerazione della password per un dipendente
Dovrà chiamare la funzione della classe Façade deputata alla rigenerazione della password di un *Dipendente*, visto che l'uso del front-end mobile e desktop è riservato a questi ultimi.

+ **Domanda mostraDomanda(Login l)**
Metodo che consente di recuperare una domanda
Dovrà chiamare la funzione della classe Façade deputata al recupero di una nuova domanda da sottoporre ad un *Dipendente*

+ **void posticipa(Login l, Domanda d)**
Metodo per posticipare una domanda
Dovrà chiamare la funzione della classe Façade deputata al posticipo della *Domanda* sottoposta ad un *Dipendente*

+ **boolean rispondi(Login l, Domanda d)**
Metodo che si occupa di comunicare la risposta data
Dovrà chiamare la funzione della classe Façade deputata al controllo della correttezza della risposta data ad una *Domanda* da un *Dipendente*

+ **void logout(Login l)**
Metodo per segnalare al sistema il logout di un utente
Dovrà chiamare la funzione della classe Façade deputata alla gestione della richiesta di logout da parte di un utente

+ **ArrayList<Badge> getBadge(Login l, int n)**
Metodo per ottenere i dati dei badge per un dato utente
Dovrà chiamare la funzione della classe Façade deputata al recupero dei primi n *Badge* guadagnati da un *Dipendente*

3.2 Package com.safetyGame.back.controller

Figura 4: *Package* ***com.safetyGame.back.controller*** *e le sue relazioni*

Tipo, obiettivo e funzione del componente: contiene gli oggetti deputati al controllo dei dati immessi nei vari front-end e del loro eventuale indirizzamento verso l'*Access-Tier* per la memorizzazione. Inoltre si occupa di chiedere all'*Access-Tier* di recuperare dati richiesti e di poi inviarli al *Presentation-Tier* affinché vengano spediti al front-end di competenza
Relazioni d'uso di altre componenti: utilizza la classe **com.safetyGame-.back.access** per memorizzare i dati da lui elaborati
Interfacce con e relazioni d'uso da altre componenti: viene utilizzato dal package **com.safetyGame.back.connection** per elaborare i dati in arrivo ed in invio da e per i vari front-end
Attività svolte e dati trattati: è deputato alla computazione su tutti i dati da e per i front-end

Il seguente diagramma delle classi fornisce una panoramica di tutte le classi che compongono il package.

Figura 5: *Diagramma delle classi*

3.2.1 com.safetyGame.back.controller.GestioneBadgeAS

Funzione

Questa classe si occuperà di gestire le operazioni che l'Amministratore Sicurezza può effettuare sui Badge. In particolare potrà recuperare la lista di tutti i badge disponibili all'interno dell'azienda.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- back.controller.GestioneDati

Inoltre utilizzerà le seguenti classi:

- back.access.DAOBadge
- back.condivisi.Badge

Attributi

- DAOBadge accessB

Metodi

+ GestioneBadgeAS(DAOBadge accessB)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ ArrayList<Badge> getBadgesAS()

Metodo per ottenere tutti i badge possibili.

Dovrà interrogare l'*Access-Tier* affinché ritorni un array con gli oggetti *Badge* che rappresenteranno tutti i badge presenti nel database.

3.2.2 com.safetyGame.back.controller.GestioneBadgeD

Funzione

Questa classe si occuperà di gestire le operazioni che i Dipedenti possono effettuare sui propri Badge. In particolare:

- **Ottenere** la lista dei badge fino ad ora acquisiti
- **Controllare** che, rispondendo ad una domanda, si abbia diritto ad acquisire uno o più nuovi badge

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`
- `back.controller.GestioneDomandeD`

Inoltre utilizzerà le seguenti classi:

- `back.access.DAOBadge`
- `back.access.DAODipendenti`
- `back.access.DAODomande`
- `back.controller.GestioneLog`
- `back.controller.GestioneLogin`
- `back.condivisi.Badge`
- `back.condivisi.Login`
- `back.condivisi.Dipendente`
- `back.condivisi.Domanda`

Attributi

- DAOBadge accessB

- DAODipendenti accessDip

- DAODomande accessDom

- GestioneLog log

- GestioneLogin login

Metodi

public GestioneBadgeD(DAOBadge accessB,DAODipendenti accessDip, DAODomande accessDom, GestioneLog log, GestioneLogin login)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

public ArrayList<Badge> getBadgeD(Login l, int n)

Metodo per ottenere i dati dei badge per un dato utente.

Dovrà verificare che esista un *Dipendente* collegato alla *Login* passata come parametro. Se esiste, allora dovrà recuperare l'oggetto *Dipendente* e la lista dei suoi badge, quindi seleziona i primi *n* Badge e li restituisce. Nel caso non esistesse il *Dipendente* o non avesse alcun *Badge*, la funzione dovrà ritornare **null.**

public boolean assegnaBadge(Domanda D, Login l)

Metodo per controllare se l'utente ha soddisfatto dei requisiti per ottenere un badge.

Dovrà verificare che esista un *Dipendente* collegato alla *Login* passata come parametro. Se esiste, allora dovrà controllare tutti i possibili *Badge* assegnabili al *Dipendente* e, nel caso ne trovasse almeno uno, dovrà chiamare la funzione deputata all'assegnazione di tale *Badge* nella classe *DAOBadge* e chiamare la classe *GestioneLog* deputata alla scrittura sul file di log di tale assegnazione.

Se dovesse aver assegnato almeno un *Badge*, la funzione dovrà ritornare **true**, altrimenti **false**.

3.2.3 com.safetyGame.back.controller.GestioneDati

Funzione

Questa classe è la componente principale del Design Pattern Façade. Contiene tutti i prototipi dei metodi pubblici delle altre classi contenute nel package e la funzione principale della classe *GestioneDati* è quella di reindirizzare le chiamate alle classi competenti.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.connection.ApplicazioniConnection`
- `back.connection.WebConnection`

Inoltre utilizzerà le seguenti classi:

- `back.controller.GestioneRecupero`
- `back.controller.GestioneLogin`
- `back.controller.GestioneDomandeD`
- `back.controller.GestioneDomandeAS`
- `back.controller.GestioneDipendentiD`
- `back.controller.GestioneDipendentiAA`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneBadgeAS`
- `back.controller.GestionePunteggiD`
- `back.controller.GestionePunteggiAA`
- `back.condivisi.Badge`
- `back.condivisi.Login`
- `back.condivisi.Dipendente`

Attributi

- GestioneRecupero gestioneRecupero

- GestioneLogin gestioneLogin

- GestioneDomandeD gestioneDomandeD

- GestioneDomandeAS gestioneDomandeAS

- GestioneDipendentiD gestioneDipendentiD

- GestioneDipendentiAA gestioneDipendentiAA

- GestioneBadgeD gestioneBadgeD

- GestioneBadgeAS gestioneBadgeAS

- GestionePunteggiD gestionePunteggiD

- GestionePunteggiAA gestionePunteggiAA

Metodi

+ GestioneDati(GestioneRecupero gr,GestioneLogin gl, GestioneDomandeD gdd, GestioneDomandeAS gda, GestioneDipendentiD gdip, GestioneDipendentiAA gdipa, GestioneBadgeD gbd, GestioneBadgeAS gba, GestionePunteggiD gpd, GestionePunteggiAA gpaa)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ ArrayList<Badge> getBadgesAS()

Metodo per ottenere tutti i badge possibili.

Dovrà interrogare la classe *GestioneBadgeAS* affinché ritorni un array con gli oggetti *Badge* che rappresenteranno tutti i badge presenti nel database.

+ ArrayList<Badge> getBadgeD(Login l, int n)

Metodo per ottenere i dati delle badge per un dato utente.

Dovrà interrogare la classe *GestioneBadgeD* affinché ritorni un array con gli oggetti *Badge* che rappresenteranno tutti i badge guadagnati da un *Dipendente* rappresentato dall'oggetto *Login*.

+ boolean assegnaBadge(Domanda D, Login l)

Metodo per controllare se l'utente ha soddisfatto dei requisiti per ottenere un badge.

Dovrà interrogare la classe *GestioneBadgeD* affinché assegni un nuovo badge al *Dipendente* data la *Domanda* in oggetto.

+ ArrayList<Dipendente> getElencoDipendenti()

Metodo per ottenere i dati dei dipendenti dell'azienda.

Dovrà interrogare la classe *GestioneDipendentiAA* affinché ritorni un array contente oggetti *Dipendente* rappresentazione di tutti i Dipendenti presenti in database

+ boolean aggiungiDipendente(Dipendente Dip)

Metodo per aggiungere un dipendente.

Dovrà chiamare la funzione contenuta in *GestioneDipendentiAA* atta ad inserire il *Dipendente* in oggetto all'interno del database.

\+ **boolean cancellaDipendente(Dipendente Dip)**

Metodo per eliminare un dipendente.

Dovrà chiamare la funzione contenuta in *GestioneDipendentiAA* atta a eliminare il *Dipendente* in oggetto all'interno del database.

\+ **boolean modDipendente(Dipendente newDip, Dipendente oldDip)**

Metodo per modificare i dati di un dipendente.

Dovrà chiamare la funzione contenuta in *GestioneDipendentiAA* atta a modificare i dati del *Dipendente oldDip* con quelli contenuti in *newDip* all'interno del database.

\+ **Dipendente getDati(Login l)**

Metodo che consente di reperire le informazioni di un dipendente a partire dal suo login.

Dovrà chiamare la funzione contenuta in *GestioneDipendentiD* atta a reperire i dati del *Dipendente*, identificato dall'oggetto *Login* passato come parametro, all'interno del database.

\+ **boolean modificaPass(Dipendente dip)**

Metodo che consente la modifica della password da parte di un dipendente.

Dovrà chiamare la funzione contenuta in *GestioneDipendentiD* atta a modificare la password del *Dipendente* all'interno del database.

\+ **boolean modificaEmail(Dipendente dip, String nEmail)**

Metodo che consente la modifica della mail da parte di un dipendente.

Dovrà chiamare la funzione contenuta in *GestioneDipendentiD* atta a modificare l'indirizzo email del *Dipendente* all'interno del database.

\+ **ArrayList<Domanda> getElencoDomande()**

Metodo per ottenere la lista di tutte le domande.

Dovrà interrogare la classe *GestioneDomandeAS* affinché ritorni un array contente oggetti *Domanda* rappresentazione di tutte le Domande scelte dall'Amministratore Sicurezza all'interno del database centrale per l'azienda.

\+ **boolean addDomanda(Domanda Dom)**

Metodo per inserire una domanda dal server domande al server dell'azienda.

Dovrà chiamare la funzione contenuta in *GestioneDomandeAS* atta ad inserire una nuova *Domanda* fra quelle proponibili ai *Dipendenti* dell'azienda.

\+ **boolean remDomanda(Domanda Dom)**

Metodo per eliminare una domanda dal server dell'azienda.

Dovrà chiamare la funzione contenuta in *GestioneDomandeAS* atta a rimuovere una *Domanda* fra quelle proponibili ai *Dipendenti* dell'azienda.

+ **Domanda getDomandaD(Login l)**
Metodo che consente di recuperare una domanda.
Dovrà chiamare la funzione contenuta in *GestioneDomandeD* atta a recuperare una domanda a cui il *Dipendente*, identificato dall'oggetto *Login*, o non ha ancora visualizzato, o ha chiesto di posticiparla.

+ **boolean setRisposta(Login l,Domanda risposta)**
Metodo che si occupa di controllare la risposta data da un dipendente ad una domanda e tenta di scrivere tali informazioni sul database. Se la risposta è corretta assegna il punteggio al dipendente.
Dovrà chiamare la funzione contenuta in *GestioneDomandeD* atta a controllare che la risposta data dal *Dipendente*, identificato dall'oggetto *Login* una domanda e tenta di scrivere tali informazioni sul database. Se la risposta è corretta assegna il punteggio al dipendente.

+ **boolean loginAdmin(Login login)**
Metodo per il login degli utenti amministratori.
Dovrà "passare la chiamata" al metodo deputato alla gestione dei tentativi di login da parte degli *Amministratori* nella classe *GestioneLogin*.

+ **boolean loginUser(Login login)**
Metodo per il login dei Dipendenti.
Dovrà "passare la chiamata" al metodo deputato alla gestione dei tentativi di login da parte dei *Dipendenti* nella classe *GestioneLogin*.

+ **ArrayList<Dipendente> getPunteggi()**
Metodo per ottenere i punteggi medi dell'azienda e i punteggi di tutti i dipendenti.
Dovrà "passare la chiamata" al metodo deputato al recupero di tutti i punteggi dei *Dipendenti* nella classe *GestionePunteggiAA*.

+ **boolean setTrofei(Dipendente Dip, int n)**
Metodo per modificare i trofei di un dipendente.
Dovrà "passare la chiamata" al metodo deputato alla modifica del numero di trofei di un *Dipendente* nella classe *GestionePunteggiAA*.

+ **Punteggio getStatisticheD(Login l)**
Metodo che consente di recuperare le statistiche di un determinato dipendente.
Dovrà "passare la chiamata" al metodo deputato al recupero delle statistiche di un *Dipendente* nella classe *GestionePunteggiD*.

+ **boolean posticipa(Login l, Domanda d)**

Metodo che si occupa di controllare quando una domanda viene posticipata.

Dovrà "passare la chiamata" al metodo deputato alla posticipazione di una domanda sottoposta ad un *Dipendente* nella classe *GestionePunteggiD*.

+ **void logout(Login l)**

Metodo per segnalare al sistema il logout di un utente.

Dovrà "passare la chiamata" al metodo deputato alla gestione dei logout nella classe *GestioneLogin*.

+ **Punteggio getStatistiche Glob(Login l)**

Metodo che consente di recuperare le statistiche globali.

Dovrà "passare la chiamata" al metodo deputato al recupero delle statistiche dell'intera azienda all'interno di *GestionePunteggiD* a partire dal *Dipendente* identificato tramite l'oggetto *Login* passato come parametro.

+ **boolean recuperoD(Recupero dip)**

Metodo che consente ad un Dipendente di resettare la propria password.

Dovrà "passare la chiamata" al metodo deputato al reset della password di un *Dipendente* all'interno della classe *GestioneRecupero*

+ **boolean recuperoA(Recupero amm)**

Metodo che consente ad un amministratore di resettare la propria password.

Dovrà "passare la chiamata" al metodo deputato al reset della password di un *Amministratore* all'interno della classe *GestioneRecupero*.

+ **boolean modPassA(Dipendente admin)**

Metodo per modificare la password di un amministratore

Dovrà chiamare la funzione contenuta in *GestioneDipendentiD* atta a modificare la password del *Dipendente* all'interno del database.

3.2.4 com.safetyGame.back.controller.GestioneDipendentiAA

Funzione

Questa classe si occuperà di gestire tutte le operazioni che un Amministratore Azienda potrà effettuare per gestire gli account sia Dipendente che Amministratore

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`

Inoltre utilizzerà le seguenti classi:

- `back.access.DAODipendenti`
- `back.condivisi.Dipendente`

Attributi

- DAODipendenti accessDip

Metodi

+ GestioneDipendentiAA(DAODipendenti accessDip)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ ArrayList<Dipendente> getElencoDipendenti()

Metodo per ottenere i dati dei dipendenti dell'azienda.

Dovrà interrogare l'*Access-Tier* affinché restituisca un array contente tutti gli oggetti *Dipendente* relativi ai Dipendenti registrati nel sistema.

+ boolean aggiungiDipendente(Dipendente Dip)

Metodo per aggiungere un dipendente.

Dovrà interrogare l'*Access-Tier* affinché inserisca il contenuto dell'oggetto *Dipendente* passato come parametro all'interno del database.

+ boolean cancellaDipendente(Dipendente Dip)

Metodo per eliminare un dipendente.

Dovrà interrogare l'*Access-Tier* affinché elimini il *Dipendente* passato come parametro dall'interno del database.

+ boolean modDipendente(Dipendente newDip, Dipendente oldDip)

Metodo per modificare i dati di un dipendente.

Dovrà interrogare l'*Access-Tier* affinché modifichi i dati del *Dipendente* all'interno del database.

\+ **boolean modPassA(Dipendente admin)**

Metodo per modificare la password di un amministratore.

Dovrà interrogare l'*Access-Tier* affinché modifichi i dati dell'*Amministratore* all'interno del database.

3.2.5 com.safetyGame.back.controller.GestioneDipendentiD

Funzione

Questa classe si occuperà di gestire tutte le operazioni che un Dipendente può effettuare sul suo account.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`

Inoltre utilizzerà le seguenti classi:

- `back.access.DAODipendenti`
- `back.controller.GestioneLog`
- `back.condivisi.Login`
- `back.condivisi.Dipendente`

Attributi

- DAODipendenti daoDipendenti

- GestioneLog gestioneLog

Metodi

+ GestioneDipendentiD(DAODipendenti d, GestioneLog g)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ Dipendente getDati(Login l)

Metodo che consente di reperire le informazioni di un dipendente a partire dal suo login.

Dovrà interrogare l'*Access-Tier* affinché gli ritorni un oggetto *Dipendente* costruito a partire dal *Login* passato come parametro.

+ boolean modificaPass(Dipendente dip)

Metodo che consente la modifica della password da parte di un dipendente.

Dovrà interrogare l'*Access-Tier* affinché modifichi la password del *Dipendente* con quella contenuta all'interno dell'oggetto *Dipendente* passato come parametro. Se non ci sono problemi, dovrà chiamare la classe deputata alla gestione del file di log affinché scriva che il *Dipendente* ha modificato la sua password.

+ boolean modificaEmail(Dipendente dip, String nEmail)

Metodo che consente la modifica della mail da parte di un dipendente.

Dovrà interrogare l'*Access-Tier* affinché modifichi la mail del *Dipendente* con la stringa passata in oggetto. Quindi, se non ci sono problemi, dovrà chiamare la classe deputata alla gestione del file di log affinché scriva che il *Dipendente* ha modificato il suo indirizzo email.

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ **DAODipendenti getDaoDipendenti()**

+ **void setDaoDipendenti(DAODipendenti daoDip)**

+ **GestioneLog getGestioneLog()**

+ **void setGestioneLog(GestioneLog gestioneLog)**

3.2.6 com.safetyGame.back.controller.GestioneDomandeAS

Funzione

Questa classe si occuperà di gestire tutte le operazioni che un Amministratore Sicurezza può effettuare sull'elenco delle domande.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`

Inoltre utilizzerà le seguenti classi:

- `back.access.DAODomande`
- `back.condivisi.Domanda`

Attributi

- DAODomande accessDom

Metodi

+ GestioneDomandeAS(DAODomande accessDom)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ ArrayList<Domanda> getElencoDomande()

Metodo per ottenere la lista di tutte le domande.

Dovrà chiamare l'*Access-Tier* affinché gli restituisca un array di tutte le domande selezionate fino ad ora dall'*Amministratore* fra quelle disponibili nel database centrale.

+ boolean addDomanda(Domanda Dom)

Metodo per inserire una domanda dal server domande al server dell'azienda.

Dovrà chiamare l'*Access-Tier* affinché inserisca la *Domanda* all'interno delle domande selezionate dall'*Amministratore*.

+ boolean remDomanda(Domanda Dom)

Metodo per eliminare una domanda dal server dell'azienda.

Dovrà chiamare l'*Access-Tier* affinché elimini la *Domanda* dalle domande selezionate dall'*Amministratore*.

3.2.7 com.safetyGame.back.controller.GestioneDomandeD

Funzione

Questa classe si occuperà di gestire tutte le operazioni che i Dipendenti possono effettuare sulle domande, in particolare:

- **Rispondere** ad una domanda
- **Ottenere** una nuova domanda
- **Posticipare** una domanda

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`

Inoltre utilizzerà le seguenti classi:

- `back.access.DAODomande`
- `back.access.DAOPunteggi`
- `back.access.DAODipendenti`
- `back.controller.GestionePunteggiD`
- `back.controller.GestioneLog`
- `back.controller.GestioneBadgeD`
- `back.condivisi.Login`
- `back.condivisi.Dipendente`
- `back.condivisi.Domanda`

Attributi

- DAODomande daoDomande

- DAOPunteggi daoPunteggi

- DAODipendenti daoDipendenti

- GestionePunteggiD gestionePunteggiD

- GestioneLog gestioneLog

- GestioneBadgeD gestioneBadge

Metodi

+ GestioneDomandeD(DAODomande d, DAOPunteggi dp, DAODipendenti dd, GestionePunteggiD g, GestioneLog gl, GestioneBadgeD gbd)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ Domanda getDomandaD(Login l)

Metodo che consente di recuperare una domanda.

Dovrà prelevare l'utente caratterizzato dalla *Login* passata come parametro e, nel caso in cui venga trovato, dovrà interrogare il database in modo tale da fornire una *Domanda* candidata ad avere risposta da parte del *Dipendente*. Quindi, nel caso venga trovata una *Domanda*, dovrà chiamare l'*Access-Tier* affinché scriva nel database e la classe deputata alla gestione dei log affinché scriva nel file di log che è stata sottoposta una nuova *Domanda* al *Dipendente*.

+ boolean setRisposta(Login l,Domanda risposta)

Metodo che si occupa di controllare la risposta data da un dipendente ad una domanda e tenta di scrivere tali informazioni sul database. Se la risposta è corretta assegna il punteggio al dipendente.

Dovrà prelevare l'utente caratterizzato dalla *Login* passata come parametro e, nel caso in cui venga trovato, dovrà interrogare il database in modo tale che scriva che alla *Domanda* è stata data risposta. Quindi, nel caso in cui sia stata data risposta corretta, dovrà controllare che nel database esista un *Badge* assegnabile[1]. Infine dovrà scrivere nel file di log che il *Dipendente* ha risposto alla *Domanda*.

+ boolean posticipa(Login l, Domanda d)

Metodo che si occupa di controllare quando una domanda viene posticipata.

Dovrà prelevare l'utente caratterizzato dalla *Login* passata come parametro e, nel caso in cui venga trovato, dovrà interrogare il database in modo tale che scriva che la *Domanda* è stata posticipata. Nel caso non ci siano problemi, dovrà quindi scrivere nel file di log che la *Domanda* è stata posticipata dal *Dipendente*.

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ DAODomande getDaoDomande()

+ void setDaoDomande(DAODomande daoDom)

+ GestionePunteggiD getGestionePunteggiD()

+ void setGestionePunteggiD(GestionePunteggiD gestionePunteggiD)

[1] Ad esempio se si è raggiunto un certo numero di risposte corrette

+ **GestioneLog getGestioneLog()**

+ **void setGestioneLog(GestioneLog gestioneLog)**

+ **DAOPunteggi getDaoPunteggi()**

+ **void setDaoPunteggi(DAOPunteggi daoPunteggi)**

+ **DAODipendenti getDaoDipendenti()**

+ **void setDaoDipendenti(DAODipendenti daoDipendenti)**

3.2.8 com.safetyGame.back.controller.GestioneLog

Funzione

Questa classe si occuperà di creare le stringhe che poi andranno inserite all'interno nel file di log.
La struttura dovrà essere di questo tipo [2]

<TIPO_OPERAZIONE> DATA ORA "usr="USERNAME <VARI>

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDati`
- `back.controller.GestioneDipendentiD`
- `back.controller.GestioneDomandeD`

Inoltre utilizzerà le seguenti classi:

- `back.access.UpdateLog`
- `back.condivisi.Login`
- `back.condivisi.Domanda`
- `back.condivisi.Dipendente`

Attributi

- UpdateLog updateLog
- **String percorso**
- **String log**

Metodi

+ GestioneLog(UpdateLog u)
Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ UpdateLog getUpdateLog()
Metodo che consente di recuperare il riferimento all'oggetto di tipo UpdateLog

+ void setUpdateLog(UpdateLog updateLog)

[2] Gli elementi variabili che variano a seconda del messaggio da inserire sono specificati metodo per metodo

Metodo che consente di impostare il riferimento all'oggetto di tipo UpdateLog

+ **void scriviLogin(Login l)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo un login effettuato correttamente.
TIPO_OPERAZIONE = "LOGIN"

+ **void scriviLogout(Login l)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo un logout effettuato correttamente
TIPO_OPERAZIONE = "LOGOUT"

+ **void scriviDomRic(Login l, Domanda d)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che una domanda viene ricevuta da un Dipendente.
TIPO_OPERAZIONE = "DOMANDA RICEVUTA"
VARI = "id dom=" + <id della domanda>

+ **void scriviDomProp(Login l, Domanda d)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che una domanda viene proposta ad un Dipendente
TIPO_OPERAZIONE = "DOMANDA PROPOSTA"
VARI = "id dom=" + <id della domanda>

+ **void scriviDomPost(Login l, Domanda d)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che una domanda è stata posticipata da un Dipendente
TIPO_OPERAZIONE = "DOMANDA POSTICIPATA"
VARI = "id dom=" + <id della domanda>

+ **void scriviDomRisp(Login l, Domanda d)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che un Dipendente risponde ad una Domanda
TIPO_OPERAZIONE = "DOMANDA RISPOSTA"
VARI = "id dom=" + <id della domanda>

+ **void scriviModPassD(Dipendente d)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che un dipendente modifica la propria password
TIPO_OPERAZIONE = "MODIFICA PASS D"

+ **void scriviModEmailD(Dipendente d)**
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che un dipendente modifica la propria email

TIPO_ OPERAZIONE = "MODIFICA EMAIL D"

+ void scriviOttenimentoBadge(Dipendente d, Badge b)
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che un dipendente ottiene un badge
TIPO_ OPERAZIONE = "OTTENIMENTO BADGE"
VARI = "badge=" + <nome badge>

void scriviAddDip(Dipendente d)
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che l'AA aggiunge un dipendente
TIPO_ OPERAZIONE = "AGGIUNTO DIPENDENTE"

+ void scriviDelDip(Dipendente d)
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che l'AA rimuove un dipendente
TIPO_ OPERAZIONE = "RIMOSSO DIPENDENTE"

+ void scriviModDip(Dipendente d)
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che l'AA modifica un dipendente
TIPO_ OPERAZIONE = "MODIFICATO DIPENDENTE"

+ void scriviAddDomande(Domanda [] d)
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che l'AS aggiunge una o più domande
TIPO_ OPERAZIONE = "AGGIUNTA DOMANDA"
VARI = "id dom=" + <id delle domande>

+ void scriviDelDomande(Domanda [] d)
Metodo che si occupa di inviare alla classe UpdateLog la stringa da inserire nel file di log dopo che l'AS rimuove una o più domande
TIPO_ OPERAZIONE= "RIMOSSA DOMANDA"
VARI = "id dom=" + <id delle domande>

3.2.9 com.safetyGame.back.controller.GestioneLogin

Funzione

Questa classe si occuperà di gestire tutte le richieste di login e logout, sia da parte di Dipendenti che di Amministratori (Azienda o Sicurezza che sia)

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDati`

Inoltre utilizza le seguenti classi:

- `back.access.DAOLogin`
- `back.controller.GestioneLog`
- `back.condivisi.Login`

Attributi

- DAOLogin access

- GestioneLog log

Metodi

+ GestioneLogin(DAOLogin access, GestioneLog log)
Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ boolean loginAdmin(Login login)
Metodo per controllare la correttezza dei dati di un tentativo di autenticazione nel sistema come Amministratore

+ boolean loginUser(Login login)
Metodo per controllare la correttezza dei dati di un tentativo di autenticazione nel sistema come Dipendente

+ void logout(Login l)
Metodo per segnalare al sistema il logout di un utente

3.2.10 com.safetyGame.back.controller.GestionePunteggiAA

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i punteggi, in particolare fornendo metodi per recuperare i punteggi di tutti i Dipendenti iscritti al sistema o per poter modificare il numero di trofei assegnati ad un utente

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`

Inoltre utilizza le seguenti classi:

- `back.access.DAOPunteggi`
- `back.access.DAODipendenti`
- `back.condivisi.Dipendente`
- `back.condivisi.Punteggio`

Attributi

- DAOPunteggi accessP

- DAODipendenti accessDip

Metodi

+ GestionePunteggiAA(DAOPunteggi accessP, DAODipendenti accessDip)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ ArrayList<Dipendente> getPunteggi()

Metodo per ottenere i punteggi medi dell'azienda e i punteggi di tutti i dipendenti. Dovrà costruire un array di oggetti *Dipendente* con al loro interno i dati di ogni Dipendente (per poterlo identificare) e al loro interno un oggetto *Punteggio* con sia i punteggi del Dipendente che dell'intera azienda.

+ boolean setTrofei(Dipendente Dip, int n)

Metodo per modificare il numero di trofei di un dipendente

3.2.11 com.safetyGame.back.controller.GestionePunteggiD

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i Badge, sia per quanto riguarda il loro recupero dal database, che una loro eventuale modifica. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire questo tipo di servizio

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`
- `back.controller.GestioneDomandeD`

Inoltre utilizza le seguenti classi:

- `back.access.DAOBadge`
- `back.access.DAODipendenti`
- `back.access.DAOPunteggi`
- `back.condivisi.Login`
- `back.condivisi.Punteggio`

Attributi

- DAOPunteggi daoPunteggi

- DAOBadge daoBadge

- DAODipendenti daoDipendenti

Metodi

+ GestionePunteggiD(DAOPunteggi dP, DAOBadge dB, DAODipendenti dD)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ Punteggio getStatisticheD(Login l)

Metodo che consente di recuperare le statistiche di un Dipendente identificato dal *Login*.

+ Punteggio getStatisticheGlob(Login l)

Metodo che consente di recuperare le statistiche dell'intera azienda e del Dipendente identificato dal *Login*.

3.2.12 com.safetyGame.back.controller.GestioneRecupero

Funzione

Questa classe si occuperà di gestire le richieste di recupero delle password.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneDati`

Inoltre verranno utilizzate le seguenti classi:

- `back.access.DAODipendenti`
- `back.controller.GestioneLog`

Attributi

- DAODipendenti accessDip

Metodi

+ GestioneRecupero(DAODipendenti accessDip)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ boolean recuperoD(Recupero dip)

Metodo che controlla la validità dei dati inseriti dall'utente e, nel caso questi siano validi, genera una nuova password, andando ad inserirla nel database.

Dovrà prima di tutto controllare che i dati inseriti siano corretti, chiamando l'*Access-Tier* affinché verifichi che *codice fiscale* e indirizzo *mail* appartengano effettivamente all'account identificato dall'*username*. Se i controlli non dovessero segnalare incongruenze, allora il metodo dovrà generare una nuova password e modificare quella che è attualmente presente nel database.

+ boolean recuperoA(Recupero amm)

Metodo che consente ad un amministratore di resettare la propria password.

Dovrà generare la password casuale, quindi dovrà chiamare l'*Access-Tier* affinché modifichi la riga riferita ai dati di accesso dell'Amministratore.

- String generaPassCasuale()

Metodo che genera una nuova password di 16 caratteri di lunghezza. Questa password dovrà contenere almeno un carattere maiuscolo, uno minuscolo, un numero e un carattere non alfa-numerico contenuto fra questi:

- @
- #

- *
- +
- ?
- ^
- %
- &
- /
- $
- !
- +
- -

3.3 Package com.safetyGame.back.access

Tipo, obiettivo e funzione del componente: il package contiene tutte le classi che si occupano di eseguire le operazioni che il *Controller-Tier* chiede che vengano eseguite sui database aziendali e centrali.

Relazioni d'uso di altre componenti: utilizza il package **com.safetyGame.back.condivisi** per trasmettere le informazioni trovate all'interno dei database al *Controller-Tier*

Interfacci con e relazioni d'uso da altre componenti: viene utilizzato dal package **com.safetyGame.back.controller** per riceve tutti i dati dal database di cui ha bisogno per le sue elaborazioni e per essere mostrati agli utenti tramite i *Front-end*

Attività svolte e dati trattati: fa da connettore fra il *Back-end* e i database, divenendo parte del componente *Model* del design pattern *MVP*

Il seguente diagramma delle classi fornisce una panoramica di tutte le classi che compongono il package.

Figura 6: *Diagramma delle classi*

3.3.1 Interfaccia com.safetyGame.back.access.DAOBadge

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i Badge, sia per quanto riguarda il loro recupero dal database, che una loro eventuale modifica. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire questo tipo di servizio

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOBadge`
- `back.access.DAOFactory`
- `back.access.SqlDAOFactory`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneBadgeAS`
- `back.controller.GestioneDipendentiD`

Inoltre utilizzerà le seguenti classi:

- `back.condivisi.Dipendente`
- `back.condivisi.Badge`

Metodi

\+ **ArrayList<Badge> badgeD(Dipendente d)**
Metodo che prende le badge ottenute da un Dipendente dal database

\+ **ArrayList<Badge> badgeAS()**
Metodo che preleva tutti i badge esistenti nel database

\+ **boolean assegna(Dipendente d, Badge b)**
Metodo che assegna una Badge ad un Dipendente

3.3.2 com.safetyGame.back.access.SqlDAOBadge

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i Badge, sia per quanto riguarda il loro recupero dal database, che una loro eventuale modifica.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOFactory`

Inoltre utilizzerà le seguenti classi:

- `back.access.Indirizzo`
- `back.condivisi.Dipendente`
- `back.condivisi.Badge`

Attributi

- Indirizzo serverAzienda;

Metodi

+ SqlDAOBadge(Indirizzo azienda)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ ArrayList<Badge> badgeD(Dipendente d)

Metodo che prende i badge ottenute da un Dipendente dal database.

Nel caso in cui la query dovesse ritornare un oggetto vuoto (ovvero la cui dimensione interna è uguale a zero), la funzione dovrà ritornare **null**.

+ ArrayList<Badge> badgeAS()

Metodo che preleva tutti i badge esistenti nel database.

Nel caso in cui la query dovesse ritornare un oggetto vuoto (ovvero la cui dimensione interna è uguale a zero), la funzione dovrà ritornare **null**.

+ boolean assegna(Dipendente d, Badge b)

Metodo che assegna una Badge ad un Dipendente

3.3.3 Interfaccia com.safetyGame.back.access.DAODipendenti

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i Dipendenti. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire questo tipo di servizio

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAODipendenti`
- `back.access.DAOFactory`
- `back.access.SqlDAOFactory`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDipendentiAA`
- `back.controller.GestioneDipendentiD`
- `back.controller.GestioneDomandeD`
- `back.controller.GestionePunteggiAA`
- `back.controller.GestionePunteggiD`
- `back.controller.GestioneRecupero`

Inoltre utilizzerà le seguenti classi:

- `back.condivisi.Login`
- `back.condivisi.Dipendente`
- `back.condivisi.Recupero`

Metodi

+ Dipendente getInfoD(Login l)

Metodo che prende le informazioni di un Dipendente dal database e le inserisce all'interno di un oggetto Dipendente.

+ Dipendente getInfoA(Login l)

Metodo che prende le informazioni di un Amministratore dal database e le inserisce all'interno di un oggetto Dipendente.

+ boolean resetPassD(Dipendente d)

Metodo che resetta il campo password modificata di un Dipendente nel database.

+ **boolean resetPassA(Dipendente d)**
Metodo che resetta il campo password modificata di un Amministratore nel database.

+ **boolean passD(Dipendente d, String pass)**
Metodo che modifica la password di un Dipendente all'interno del Database secondo la stringa *pass*.

+ **boolean passA(Dipendente d, String pass)**
Metodo che setta il campo password (e il campo data pass) di un Amministratore

+ **boolean mailD(Dipendente d, String mail)**
Metodo che setta il campo mail di un Dipendente

+ **ArrayList<Dipendente> elencoDipendenti()**
Metodo che ritorna l'elenco dei Dipendenti dell'Azienda

+ **boolean aggiungiDipendente(Dipendente d)**
Metodo che aggiunge un Dipendente nel database

+ **boolean cancellaDipendente(Dipendente d)**
Metodo che cancella un Dipendente dal database

+ **boolean modNome(Dipendente d, String nome)**
Metodo che modifica il nome di un Dipendente nel database

+ **boolean modCognome(Dipendente d, String cognome)**
Metodo che modifica il nome di un Dipendente nel database

+ **boolean modCodFis(Dipendente d, String codfis)**
Metodo che modifica il codice fiscale di un Dipendente nel database

+ **boolean modUsername(Dipendente d, String username)**
Metodo che modifica lo username di un Dipendente nel database

+ **boolean modImpiego(Dipendente d, String impiego)**
Metodo che modifica l'impiego di un Dipendente nel database

+ **boolean setTrofei(Dipendente d, int n)**
Metodo che modifica il numero di trofei di un Dipendente nel database

+ **boolean resetD(Recupero r, String p)**

Metodo che resetta la password (casuale) di un Dipendente

\+ **boolean resetA(Recupero r, String p)**
Metodo che resetta la password (casuale) di un Amministratore

3.3.4 com.safetyGame.back.access.SqlDAODipendenti

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i Dipendenti.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOFactory`

Inoltre utilizzerà le seguenti classi:

- `back.access.Indirizzo`
- `back.condivisi.Login`
- `back.condivisi.Dipendente`
- `back.condivisi.Recupero`

Attributi

- Indirizzo serverAzienda;

Metodi

+ SqlDAODipendenti(Indirizzo azienda)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ Dipendente getInfoD(Login l)

Metodo che prende le informazioni di un Dipendente dal database e le inserisce all'interno di un oggetto Dipendente.

+ Dipendente getInfoA(Login l)

Metodo che prende le informazioni di un Amministratore dal database e le inserisce all'interno di un oggetto Dipendente.

+ boolean resetPassD(Dipendente d)

Metodo che resetta il campo password modificata di un Dipendente nel database.

+ boolean resetPassA(Dipendente d)

Metodo che resetta il campo password modificata di un Amministratore nel database.

+ boolean passD(Dipendente d, String pass)

Metodo che modifica la password di un Dipendente all'interno del Database secondo la stringa *pass*.

+ **boolean passA(Dipendente d, String pass)**
Metodo che setta il campo password (e il campo data pass) di un Amministratore

+ **boolean mailD(Dipendente d, String mail)**
Metodo che setta il campo mail di un Dipendente

+ **ArrayList<Dipendente> elencoDipendenti()**
Metodo che ritorna l'elenco dei Dipendenti dell'Azienda

+ **boolean aggiungiDipendente(Dipendente d)**
Metodo che aggiunge un Dipendente nel database

+ **boolean cancellaDipendente(Dipendente d)**
Metodo che cancella un Dipendente nel database

+ **boolean modNome(Dipendente d, String nome)**
Metodo che modifica il nome di un Dipendente nel database

+ **boolean modCognome(Dipendente d, String cognome)**
Metodo che modifica il nome di un Dipendente nel database

+ **boolean modCodFis(Dipendente d, String codfis)**
Metodo che modifica il codice fiscale di un Dipendente nel database

+ **boolean modUsername(Dipendente d, String username)**
Metodo che modifica lo username di un Dipendente nel database

+ **boolean modImpiego(Dipendente d, String impiego)**
Metodo che modifica l'impiego di un Dipendente nel database

+ **boolean setTrofei(Dipendente d, int n)**
Metodo che modifica il numero di trofei di un Dipendente nel database

+ **boolean resetD(Recupero r, String p)**
Metodo che resetta la password (casuale) di un Dipendente

+ **boolean resetA(Recupero r, String p)**
Metodo che resetta la password (casuale) di un Amministratore

3.3.5 Interfaccia com.safetyGame.back.access.DAODomande

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti le domande. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire questo tipo di servizio

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.DAOFactory`
- `back.access.SqlDAODomande`
- `back.access.SqlDAOFactory`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDomandeAS`
- `back.controller.GestioneDomandeD`

Inoltre utilizzerà le seguenti classi:

- `back.condivisi.Dipendente`
- `back.condivisi.Domanda`

Metodi

+ **Domanda getDomanda(Dipendente d)**

Metodo che, dato un Dipendente, fornisce un oggetto Domanda che conterrà una domanda a cui il Dipendente o non ha risposto (possibilmente avendo chiesto di posticiparla), o ha risposto erroneamente o non ha mai visto.

+ **boolean posticipa(Dipendente d, Domanda dom)**

Metodo che dovrà posticipare la Domanda sottoposta al Dipendente, in modo tale che questa possa essere riproposta più tardi allo stesso.

+ **boolean rispondi(Dipendente d, Domanda dom)**

Metodo che permette di verificare la correttezza di una risposta data da un Dipendente ad una Domanda.

+ **ArrayList<Domanda> domandeA()**

Metodo che ritorna un array contenente tutte le Domande che l'Amministratore Sicurezza ha scelto per l'azienda.

+ **ArrayList<Domanda> domande(Dipendente d, Domanda dom)**

Metodo che ritorna un array contenente tutte le Domande a cui un dipendente ha dato risposta.

+ **boolean addDomanda(Domanda d)**
Metodo che aggiunge una Domanda al database aziendale, in modo tale che questa possa essere sottoposta ai Dipendenti

+ **boolean remDomanda(Domanda d)**
Metodo che rimuove una Domanda dal database aziendale, in modo tale che questa non possa più essere sottoposta ai Dipendenti.

+ **boolean scriviSottoposta(Dipendente dip, Domanda dom)**
Metodo che scrive sul database che una Domanda è stata sottoposta ad un Dipendente.

3.3.6 com.safetyGame.back.access.SqlDAODomande

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti le domande.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOFactory`

Inoltre le seguenti classi sono utilizzate:

- `back.access.Indirizzo`
- `back.condivisi.Dipendente`
- `back.condivisi.Domanda`

Attributi

- Indirizzo serverDomande;

- Indirizzo serverAzienda;

Metodi

+ SqlDAODomande(Indirizzo azienda, Indirizzo domande)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

- Domanda prendiCampiDomanda(int id)

Metodo che recupera i campi che compongono un oggetto Domanda tramite il suo id.

Dovrà innanzitutto recuperare la domanda dal server centrale, quindi creare l'oggetto *Domanda* e inserire al suo interno i dati recuperati. Nel caso una delle query dovesse lanciare un'eccezione, la funzione dovrà ritornare un valore **null**.

+ Domanda getDomanda(Dipendente d)

Metodo che, dato un Dipendente, fornisce un oggetto Domanda che conterrà una domanda a cui il Dipendente o non ha risposto (possibilmente avendo chiesto di posticiparla), o ha risposto erroneamente o non ha mai visto.

Tale metodo dovrà controllare in primis che non esistano domande a cui il Dipendente non ha ancora dato risposta. Se questa condizione dovesse essere verificata, la funzione dovrà prelevare una nuova domanda da sottoporre al Dipendente combinando le domande a cui ha risposto erroneamente con quelle a cui non ha mai dato risposta e che non gli sono state sottoposte. Quindi ritornerà l'oggetto passato dalla funzione `prendiCampiDomanda(int id)`

+ boolean posticipa(Dipendente d, Domanda dom)

Metodo che dovrà posticipare la Domanda sottoposta al Dipendente, in modo tale che questa possa essere riproposta più tardi allo stesso.

+ **boolean rispondi(Dipendente d, Domanda dom)**
Metodo che permette di verificare la correttezza di una risposta data da un Dipendente ad una Domanda.
Dovrà confrontare la risposta data con l'id della risposta corretta e, nel caso coincidessero, assegnare i punti previsti al Dipendente. Quindi dovrà segnare in database sia che è stata data risposta, che l'esito, provvedendo eventualmente ad aggiornare il punteggio del Dipendente.

+ **ArrayList<Domanda> domandeA()**
Metodo che ritorna un array contenente tutte le Domande che l'Amministratore Sicurezza ha scelto per l'azienda.
Questo metodo dovrà recuperare gli indici di tutte le domande dal database aziendale, quindi comporre gli oggetti Domanda recuperando le informazioni dal database centrale. Infine inserirle dentro un array, quindi restituirlo alla funzione chiamante.

+ **ArrayList<Domanda> domande(Dipendente d, Domanda dom)**
Metodo che ritorna un array contenente tutte le Domande a cui un dipendente ha dato risposta.
Dovrà associare al Dipendente tutte le possibili domande e da queste estrarre quelle a cui ha dato risposta.

+ **boolean addDomanda(Domanda d)**
Metodo che aggiunge una Domanda al database aziendale, in modo tale che questa possa essere sottoposta ai Dipendenti

+ **boolean remDomanda(Domanda d)**
Metodo che rimuove una Domanda dal database aziendale, in modo tale che questa non possa più essere sottoposta ai Dipendenti.

+ **boolean scriviSottoposta(Dipendente dip, Domanda dom)**
Metodo che scrive sul database che una Domanda è stata sottoposta ad un Dipendente.

3.3.7 Interfaccia com.safetyGame.back.access.DAOFactory

Funzione

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOFactory`
- `back.controller.GestioneDipendentiD`
- `back.controller.GestioneDomandeD`

Inoltre le seguenti classi vengono utilizzate:

- `back.access.DAOLogin`
- `back.access.DAODipendenti`
- `back.access.DAODomande`
- `back.access.DAOBadge`
- `back.access.DAOPunteggi`
- `back.access.Indirizzo`

Metodi

+ DAOLogin creaDAOLogin(Indirizzo azienda)

Metodo che dovrà restituire un oggetto di sottotipo di DAOLogin; più precisamente dovrà esser il sottotipo corretto in relazione alla tipologia di database utilizzato.

+ DAODipendenti creaDAODipendenti(Indirizzo azienda)

Metodo che dovrà restituire un oggetto di sottotipo di DAODipendenti; più precisamente dovrà esser il sottotipo corretto in relazione alla tipologia di database utilizzato.

+ DAODomande creaDAODomande (Indirizzo azienda, Indirizzo domande)

Metodo che dovrà restituire un oggetto di sottotipo di DAODomande; più precisamente dovrà esser il sottotipo corretto in relazione alla tipologia di database utilizzato.

+ DAOBadge creaDAOBadge(Indirizzo azienda)

Metodo che dovrà restituire un oggetto di sottotipo di DAOBadge**;** più precisamente dovrà esser il sottotipo corretto in relazione alla tipologia di database utilizzato.

+ DAOPunteggi creaDAOPunteggi(Indirizzo azienda, Indirizzo domande)

Metodo che dovrà restituire un oggetto di sottotipo di DAOLogin; più precisamente dovrà esser il sottotipo corretto in relazione alla tipologia di database utilizzato.

3.3.8 com.safetyGame.back.access.SqlDAOFactory

Funzione

Relazioni d'uso con altri moduli

Le seguenti classe verranno utilizzate:

- `back.access.SqlDAOLogin`
- `back.access.SqlDAODipendenti`
- `back.access.SqlDAODomande`
- `back.access.SqlDAOBadge`
- `back.access.SqlDAOPunteggi`
- `back.access.Indirizzo`

Attributi

Metodi

+ **SqlDAOFactory()**
Costruttore della classe.

+ **DAOLogin creaDAOLogin(Indirizzo azienda)**
Metodo che dovrà restituire un oggetto di tipo SqlDAOLogin, visto che la classe riguarda Sql.

+ **DAODipendenti creaDAODipendenti(Indirizzo azienda)**
Metodo che dovrà restituire un oggetto di tipo SqlDAODipendenti, visto che la classe riguarda Sql.

+ **DAODOmande creaDAODomande (Indirizzo azienda, Indirizzo domande)**
Metodo che dovrà restituire un oggetto di tipo SqlDAODomande, visto che la classe riguarda Sql.

+ **DAOBadge creaDAOBadge(Indirizzo azienda)**
Metodo che dovrà restituire un oggetto di tipo SqlDAOBadge, visto che la classe riguarda Sql.

+ **DAOPunteggi creaDAOPunteggi(Indirizzo azienda, Indirizzo domande)**
Metodo che dovrà restituire un oggetto di tipo SqlDAOPunteggi, visto che la classe riguarda Sql.

3.3.9 Interfaccia com.safetyGame.back.access.DAOLogin

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i tentativi di autenticazione all'interno del sistema. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire questo tipo di servizio

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.DAOFactory`
- `back.access.SqlDAOFactory`
- `back.access.SqlDAOLogin`
- `back.controller.GestioneLogin`

Metodi

+ boolean loginAmministratore(Login l)

Metodo che controllerà la correttezza dei dati inseriti da un utente per connettersi al sistema come amministratore.

+ boolean loginDipendente(Login l)

Metodo che controllerà la correttezza dei dati inseriti da un utente per connettersi al sistema come dipendente.

3.3.10 com.safetyGame.back.access.SqlDAOLogin

Funzione

Questa classe si occuperà di gestire tutti i dati riguardanti i tentativi di autenticazione all'interno del sistema.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOFactory`

Inoltre vengono utilizzate le seguenti classi:

- `back.access.Indirizzo`
- `back.condivisi.Login`

Attributi

- Indirizzo serverAzienda

Metodi

+ SqlDAOLogin(Indirizzo azienda)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ boolean loginAmministratore(Login l)

Metodo che controllerà la correttezza dei dati inseriti da un utente per connettersi al sistema come amministratore. Dovrà eseguire una query sul database aziendale cercando i dati contenuti nell'oggetto Login all'interno della tabella contenente le credenziali d'accesso degli amministratori e, nel caso non vengano lanciate eccezioni, dovrà ritornare **true**. Altrimenti **false**.

+ boolean loginDipendente(Login l)

Metodo che controllerà la correttezza dei dati inseriti da un utente per connettersi al sistema come dipendente. Dovrà eseguire una query sul database aziendale cercando i dati contenuti nell'oggetto Login all'interno della tabella contenente le credenziali d'accesso dei Dipendenti e, nel caso non vengano lanciate eccezioni, dovrà ritornare **true**. Altrimenti **false**.

3.3.11 Interfaccia com.safetyGame.back.access.DAOPunteggi

Funzione

Questa classe si occuperà di recuperare i dati riguardanti i Punteggi dei Dipendenti e dell'Azienda. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire questo tipo di servizio

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOPunteggi`
- `back.access.DAOFactory`
- `back.access.SqlDAOFactory`
- `back.controller.GestioneDomandeD`
- `back.controller.GestionePunteggiAA`
- `back.controller.GestionePunteggiD`

Metodi

+ Punteggio getStat(Dipendente d)

Metodo che dovrà ritornare un oggetto di tipo Punteggio con informazioni relative al solo Dipendente in oggetto.

+ Punteggio getGlobalStat(Dipendente dip)

Metodo che dovrà ritornare un oggetto di tipo Punteggio contenente le informazioni sia del Dipendente in oggetto, che di quei Dipendenti immediatamente prossimi a lui nella classifica dei punteggi, oltre che alle statistiche dell'azienda (come punteggio medio o numero di risposte corrette totali).

3.3.12 com.safetyGame.back.access.SqlDAOPunteggi

Funzione

Implementa back.access.DAOPunteggi.
Questa classe si occuperà di recuperare i dati riguardanti i Punteggi dei Dipendenti e dell'Azienda da un database Sql.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOFactory`

Inoltre utilizzerà le seguenti classi:

- `back.access.Indirizzo`
- `back.condivisi.Dipendente`
- back.condivisi.Punteggio

Attributi

- Indirizzo serverAzienda;

- Indirizzo serverDomande;

Metodi

+ SqlDAOPunteggi(Indirizzo azienda, Indirizzo domande)
Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ Punteggio getStat(Dipendente d)
Questo metodo dovrà ritornare un oggetto di tipo Punteggio con informazioni relative al solo Dipendente in oggetto. Per cui dovrà effettuare una query al database aziendale per recuperare il punteggio che il Dipendente ha ottenuto fino a quel momento e quindi ritornare l'oggetto Punteggio creato a partire da quanto risulta nel database.

+ Punteggio getGlobalStat(Dipendente dip)
Questo metodo dovrà ritornare un oggetto di tipo Punteggio contenente le informazioni sia del Dipendente in oggetto, che di quei Dipendenti immediatamente prossimi a lui nella classifica dei punteggi, oltre che alle statistiche dell'azienda (come punteggio medio o numero di risposte corrette totali). Dovrà interrogare ogni tabella che contiene questo tipo di informazioni e, una volta recuperate, incapsularle all'interno di un oggetto Punteggio, che dovrà restituire alla funzione chiamante.

3.3.13 com.safetyGame.back.access.UpdateLog

Funzione

Questa classe fungerà da contenitore per i dati riguardanti i Badge che sono stati inseriti all'interno del database aziendale

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.controller.GestioneLog`

Attributi

- PrintWriter out

Metodi

+ UpdateLog(String s)
Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ void finalize()
Distruttore della classe: dovrà chiudere lo stream al file di log

+ void scrivi(String s)
Aggiunge una nuova riga al file di log senza chiudere lo stream verso il file

+ void scriviChiudi(String s)
Aggiunge una nuova riga al file e chiude lo stream al file di log

3.3.14 com.safetyGame.back.access.Indirizzo

Funzione

Questa classe contiene tutti i metodi per l'esecuzione di query all'interno dei database (azienda e centrale)

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOBadge`
- `back.access.SqlDAODipendenti`
- `back.access.SqlDAODomande`
- `back.access.SqlDAOFactory`
- `back.access.SqlDAOLogin`
- `back.access.SqlDAOPunteggi`

Attributi

- Connection conn

- Statement connettore

Metodi

+ Indirizzo(String database, String utente, String password)

Costruttore che imposta il valore degli attributi secondo il valore dei parametri passati.

+ void finalize()

Distruttore della classe Indirizzo: prova a chiudere la connessione al database quando l'oggetto viene distrutto. Nel caso di lancio di eccezioni, dovrà stampare a video un messaggio d'errore

+ boolean inserisciRiga(String tabella, String colonne, String [] valori)

Crea, a partire dall'array *valori*, una stringa (che chiameremo *val*) che contiene tutti i valori contenuti. Dovrà avere la seguente struttura:

(?,?....,?)

ripetendo tanti punti interrogativi quanti sono gli elementi dentro all'array *valori*. Quindi crea la query da eseguire sul database utilizzando:

- **tabella:** nome della tabella su cui eseguire la query

- **colonne:** colonne dove andranno inseriti i dati. Al contenuto di questa variabile dovrà essere applicata una funzione per eliminare eventuali caratteri di spaziatura
- **val:**

Quindi i punti interrogativi saranno sostituiti dai valori contenuti nell'array *valori*. Se la funzione di esecuzione della query non lancerà alcuna eccezione, questo metodo si concluderà ritornando **true**. Altrimenti dovrà ritornare **false**.

+ boolean modificaRiga(String tabella, String colonnevalori, String controlli)

Modifica una tupla di valori all'interno della tabella indicata, selezionandola secondo i parametri di controllo indicati. Nel caso l'esecuzione della query ritornasse un'eccezione, la funzione dovrà ritornare **false**, altrimenti **true**.

+ boolean cancellaRiga(String tabella, String controlli)

Elimina una tupla di valori all'interno della tabella indicata, selezionandola secondo i parametri di controllo indicati. Se la stringa *controlli* dovesse risultare vuota dopo aver applicato una funzione per eliminare eventuali caratteri di spaziatura, la funzione dovrà ritornare **false**, così come se l'esecuzione della query sul database dovesse sollevare un'eccezione. Altrimenti la funzione dovrà ritornare **true**.

+ ResultSet selezione(String tabella, String colonne, String controlli, String extra)

Preleva una tupla di valori all'interno della tabella indicata, selezionandola secondo i parametri di controllo indicati. Dovrà eseguire una query sulla *tabella* indicata, prelevando i valori all'interno delle *colonne* secondo le condizioni indicate in *controlli* e *extra*.

Nel caso dovesse essere sollevata un'eccezione, la funzione dovrà ritornare **null.** Altrimenti dovrà ritornare quanto ritornato dalla query.

3.4 Package com.safetyGame.back.condivisi

Tipo, obiettivo e funzione del componente: il package contiene le classi che conterranno le informazioni che dovranno essere condivise fra i vari strati del *Back-end*

Relazioni d'uso di altre componenti: non utilizza alcun componente del *Back-end*

Interfacci con e relazioni d'uso da altre componenti: viene utilizzato da tutte le classi dei package *controller, access* e *connection* per trasmettere informazioni complesse

Attività svolte e dati trattati: serve a trasferire le informazioni che dovranno essere condivise fra i vari package che compongono il *back-end*. In particolare si occuperà di contenere le informazioni riguardanti:

- **Badge**
- **Dipendenti**
- **Domande**
- **Login**
- **Punteggi**
- **Richieste di recupero password**

3.4.1 com.safetyGame.back.condivisi.Badge

Funzione

Questa classe fungerà da contenitore per i dati riguardanti i Badge che sono stati inseriti all'interno del database aziendale

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOBadge`
- `back.condivisi.Dipendente`
- `back.connection.WebConnection`
- `back.controller.GestioneBadgeAS`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDati`
- `back.controller.GestioneLog`

Attributi

- String nome

Nome del badge

- String descrizione

Descrizione del badge (ex. la motivazione per cui si è guadagnato questo badge)

- Punteggio soglia

Indica il quantitativo di punti necessari all'acquisizione del badge

Metodi

+ Badge(String nome, String d, Punteggio p)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ Badge()

Costruttore di default che dovrà inizializzare ogni attributo a **NULL**

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ String getNome()

+ void setNome(String nome)

+ **String getDescrizione()**

+ **void setDescrizione(String descrizione)**

+ **Punteggio getSoglia()**

+ **void setSoglia(Punteggio soglia)**

3.4.2 com.safetyGame.back.condivisi.DataOra

Funzione

Questa classe fungerà da contenitore per le date e le ore

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOBadge`
- `back.access.SqlDAODipendenti`
- `back.condivisi.badge`
- `back.condivisi.Dipendente`
- `back.condivisi.Login`
- `back.controller.GestioneLog`

Attributi

- int anno

- int mese

- int giorno

- int ora

- int minuti

- int secondi

Metodi

+ DataOra(int a, int me, int g, int o, int mi, int s)

Costruttore che imposta gli attributi dell'oggetto alla data e all'ora specificata

+ DataOra()

Costruttore di default che crea l'oggetto con la data e l'ora corrente

+ static String aggiusta(int parteData)

Funzione statica che deve restituire la parte della data (ex. giorno) scritta con due cifre (ex. se si è al giorno "2", questa funzione deve restituire una stringa con scritto "02")

+ String toString()

Restituisce una stringa dell'oggetto nel seguente formato:
`AAAA/MM/GG HH:MM:SS`

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ **int getAnno()**

+ **void setAnno(int anno)**

+ **int getMese()**

+ **void setMese(int mese)**

+ **int getGiorno()**

+ **void setGiorno(int giorno)**

+ **int getOra()**

+ **void setOra(int ora)**

+ **int getMinuti()**

+ **void setMinuti(int minuti)**

+ **int getSecondi()**

+ **void setSecondi(int secondi)**

3.4.3 com.safetyGame.back.condivisi.Dipendente

Funzione

Questa classe fungerà da contenitore per i dati riguardanti i Dipendenti che sono stati inseriti all'interno del database aziendale

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAOBadge`
- `back.access.SqlDAODipendenti`
- `back.access.SqlDAODomande`
- `back.access.SqlDAOLogin`
- `back.access.SqlDAOPunteggi`
- `back.condivisi.Domanda`
- `back.condivisi.Login`
- `back.condivisi.Punteggio`
- `back.condivisi.Recupero`
- `back.connection.ApplicazioniConnection`
- `back.connection.WebConnection`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDati`
- `back.controller.GestioneDipendentiAA`
- `back.controller.GestioneDipendentiD`
- `back.controller.GestioneDomandeD`
- `back.controller.GestioneLog`
- `back.controller.GestioneLogin`
- `back.controller.GestionePunteggiAA`
- `back.controller.GestionePunteggiD`

Attributi

- int id
Numero univoco che identifica l'utente all'interno del database

- String codFiscale
Contiene il codice fiscale dell'utente

- ArrayList<Badge>
Array contenente tutti i badge guadagnati da un utente

- Punteggio punteggio
Contiene alcune statistiche riguardanti le risposte date e i punti guadagnati dall'utente

- String nome
Contiene il nome dell'utente

- String cognome
Contiene il cognome dell'utente

- String email
Contiene l'indirizzo email dell'utente

- String nickname
Contiene l'username assegnato all'utente

- String password
Contiene la password dell'utente

- String ruolo
Contiene il ruolo che l'utente ha all'interno dell'azienda

- boolean ammAA
True = L'utente è un amministratore azienda; False = L'utente è un amministratore sicurezza

- DataOra dataModPass
Contiene la data e l'ora dell'ultima modifica alla password

- String nuovaPass
Contiene la password che dovrà essere scritta nel database

- int trofei
Contiene il numero di trofei guadagnati

Metodi

+ Dipendente()

+ Dipendente(int id, String cf, String n, String c, String e, String nn, String p, String r, int pu, String np, int trf)

+ Dipendente(boolean aA, DataOra dmp, String np, String mail, String nick, String pass, String codfisc, int i)

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:
+ int getId()

+ void setId(int id)

+ String getCodFiscale()

+ void setCodFiscale(Stirng codFiscale)

+ ArrayList<Badge> getBadges()

+ void setBadges(ArrayList<Badge> badges)

+ void addBadge(Badge badge)

+ Punteggio getPunteggio()

+ void setPunteggio(Punteggio punteggio)

+ String getNome()

+ void setNome(String nome)

+ String getCognome()

+ void setCognome(String cognome)

+ String getEmail()

+ void setEmail(String email)

+ String getNickname()

+ void setNickname(String nickname)

+ **String getPassword()**

+ **void setPassword(String password)**

+ **String getRuolo()**

+ **void setRuolo(String ruolo)**

+ **String toStringID()**

+ **boolean isAmmAA()**

+ **void setAmmAA(boolean ammAA)**

+ **DataOra getDataModPass()**

+ **void SetDataModPass(DataOra dataModPass)**

+ **String getNuovaPass()**

+ **void setNuovaPass(String nuovaPass)**

+ **int getTrofei()**

+ **void setTrofei(int trofei)**

3.4.4 com.safetyGame.back.condivisi.Domanda

Funzione

Questa classe fungerà da contenitore per le domande che dovranno essere trasmesse ai vari front-end e le relative risposte che da questi verranno indirizzate al back-end

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAODomande`
- `back.access.SqlDAOPunteggi`
- `back.condivisi.Punteggio`
- `back.connection.ApplicazioniConnection`
- `back.connection.WebConnection`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDati`
- `back.controller.GestioneDomandeAS`
- `back.controller.GestioneDomandeD`
- `back.controller.GestioneLog`
- `mobile.condivisi.Domanda`
- `mobile.Utils.ConnectionUtils`
- `mobile.View.DashboardActivity`
- `mobile.View.DatiActivity`
- `mobile.View.DomandaActivity`
- `mobile.View.PunteggiActivity`
- `mobile.View.TimerNotifica`

Attributi

- int id

Numero univoco che identifica la domanda all'interno del database

- Punteggio punteggio

Punteggio attribuito alla domanda

- String tipologia

Identifica la tipologia della domanda (ex. Risposta multipla, risposta aperta, ecc...)

- ArrayList<String> risposte
Conterrà tutte le possibili risposte

- int corretta
Identifica all'interno della lista *risposte* quella corretta

- String testo
Contiene il testo della domanda

- int rispostaData
Identifica qual'è stata la risposta selezionata da un dipendente. rispostaData=-1 se non è ancora stata selezionata alcuna risposta

- boolean mobile
Identifica se la domanda è stata proposta o dovrà essere proposta sarà destinata ad un dispositivo mobile

- int tempo
Identifica il tempo permesso per rispondere alla domanda. tempo=-1 se non è stato assegnato un parametro temporale alla domanda (ovvero si potrà rispondere alla domanda impiegando quanto "tempo si vuole")

- String ambito
Identifica il "settore" di appartenenza della domanda (per esempio se la domanda è attinente alle norme anti-incendio oppure a quelle anti-infortunistica)

- boolean internaAzienda
Indica se la domanda può essere proposta ai Dipendenti in quanto selezionata dall'Amminstratore Sicurezza

Metodi

+ Domanda(int i, Punteggio p, String ti, ArrayList<String> r, int c, String te, int rd, boolean m, int sec, String a, boolean inA)
Costruttore con parametri. Dovrà inizializzare i campi dati con le variabili passate

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:
+ int getId()

+ void setId(int id)

+ **Punteggio getPunteggio()**

+ **void setPunteggio(Punteggio punteggio)**

+ **String getTipologia()**

+ **void setTipologia(String tipologia)**

+ **ArrayList<String> getRisposte()**

+ **void setRisposte(ArrayList<String> risposte)**

+ **int getCorretta()**

+ **void setCorretta(int corretta)**

+ **String getTesto()**

+ **void setTesto(String testo)**

+ **int getRispostaData()**

+ **void setRispostaData(int rispostaData)**

+ **boolean isMobile()**

+ **void setMobile(boolean mobile)**

+ **int getTempo()**

+ **void setTempo(int tempo)**

+ **String getAmbito()**

+ **void setAmbito(String ambito)**

+ **boolean isInternaAzienda()**

+ **void setInternaAzienda(boolean internaAzienda)**

3.4.5 com.safetyGame.back.condivisi.Login

Funzione

Questa classe fungerà da contenitore per i dati riguardanti i tentativi di autenticarsi all'interno della piattaforma

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAODipendenti`
- `back.access.SqlDAOLogin`
- `back.connection.ApplicazioniConnection`
- `back.connection.WebConnection`
- `back.controller.GestioneBadgeD`
- `back.controller.GestioneDati`
- `back.controller.GestioneDipendentiD`
- `back.controller.GestioneDomandeD`
- `back.controller.GestioneLog`
- `back.controller.GestioneLogin`
- `back.controller.GestionePunteggiD`
- `desktop.logic.ControlLogin`
- `desktop.logic.DatiLogin`
- `desktop.view.Login`
- `mobile.Utils.ConnectionUtils`
- `mobile.Utils.IntentIntegrator`
- `mobile.View.LoginActivity`
- `mobile.View.TimerNotifica`

Attributi

- String username

Username dell'account che tenta di fare login

- String password

Password dell'account che tenta di fare login

Metodi

+ **Login()**
  Costruttore di default senza parametri. I campi dati dovranno essere inizializzati con valori di default

+ **Login(String u, String p)**
  Costruttore con parametri. Dovrà inizializzare i campi dati con le variabili passate

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ **String getUsername()**

+ **void setUsername(String username)**

+ **String getPassword()**

+ **void setPassword(String password)**

3.4.6 com.safetyGame.back.condivisi.Punteggio

Funzione

Questa classe fungerà da contenitore per i dati riguardanti i punteggi sia dei dipendenti che dell'intera azienda.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAODomande`
- `back.access.SqlDAOPunteggi`
- `back.condivisi.Dipendente`
- `back.condivisi.Domanda`
- `back.connection.ApplicazioniConnection`
- `back.connection.WebConnection`
- `back.controller.GestioneDati`
- `back.controller.GestioneBadgeD`
- `back.controller.GestionePunteggiAA`
- `back.controller.GestionePunteggiD`

Attributi

- int punti

Indica i punti accumulati da un dipendenti O i punti accumulati da tutti i dipendenti dell'azienda

- double mediaPuntiAzienda

Indica il punteggio medio di tutti i dipendenti dell'azienda

- int puntiPrec

Indica il numero di punti ottenuti dal Dipendente direttamente successivo al Dipendente "proprietario" all'interno della classifica dei punteggi (ovvero colui che ha meno punti del proprietario di questo oggetto)

- int puntiSuc

Indica il numero di punti ottenuti dal Dipendente direttamente precedente al Dipendente "proprietario" all'interno della classifica dei punteggi (ovvero colui che ha più punti del proprietario di questo oggetto)

- int nDomRisp

Indica il numero di domande a cui o il Dipendente o tutti i Dipendenti dell'azienda hanno dato risposta

- int nRispCorr
Indica il numero di domande a cui o il Dipendente o tutti i Dipendenti dell'azienda hanno dato risposta corretta

Metodi

+Punteggio()
Costruttore di default senza parametri. I campi dati dovranno essere inizializzati con valori di default

+ Punteggio(int punti)
Costruttore con parametri. Dovrà inizializzare i campi dati con le variabili passate

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ int getPunti()

+ void setPunti(int punti)

3.4.7 com.safetyGame.back.condivisi.Recupero

Funzione

Questa classe fungerà da contenitore per i dati riguardanti le richieste di recupero password.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

- `back.access.SqlDAODipendenti`
- `back.connection.WebConnection`

Attributi

- String email

Indirizzo email a cui si dovrà inviare la mail con la password

- String codFiscale

Codice fiscale dell'account a cui è collegato anche la mail

Metodi

+ Recupero(String email, String codFiscale)

Costruttore con parametri. Dovrà inizializzare i campi dati con le variabili passate

+ Recupero()

Costruttore di default senza parametri. I campi dati dovranno essere inizializzati con valori di default

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ String getEmail()

+ void setEmail(String email)

+ String getCodFiscale()

+ void setCodFiscale(String codFiscale)

4 Specifica Front-end Desktop

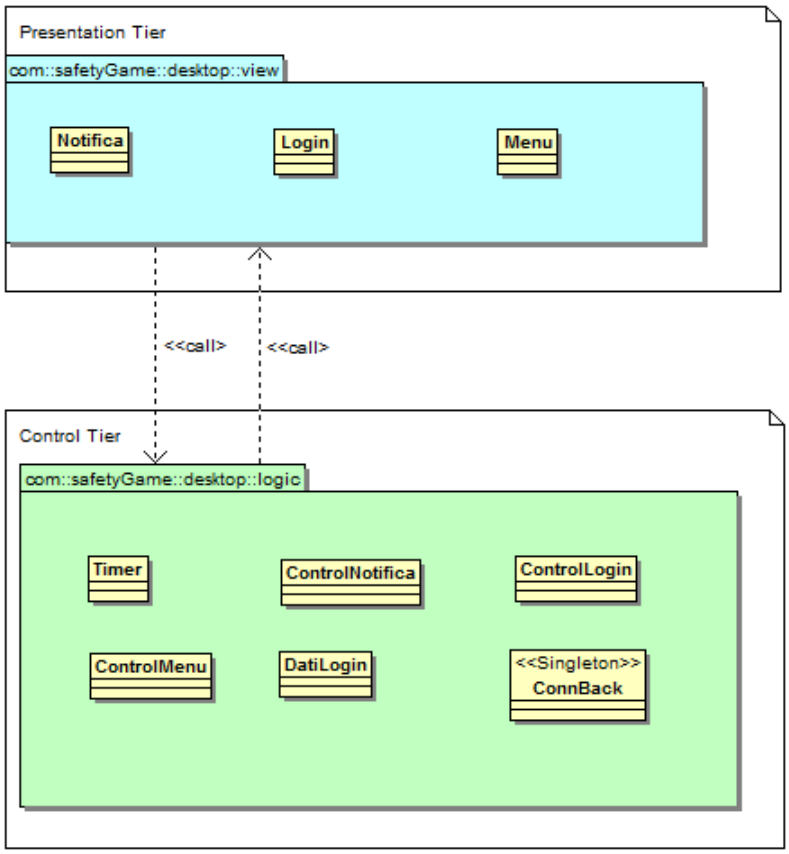

Figure 7: *Front-end Desktop, diagramma dei package*

5 Specifica Front-end Web

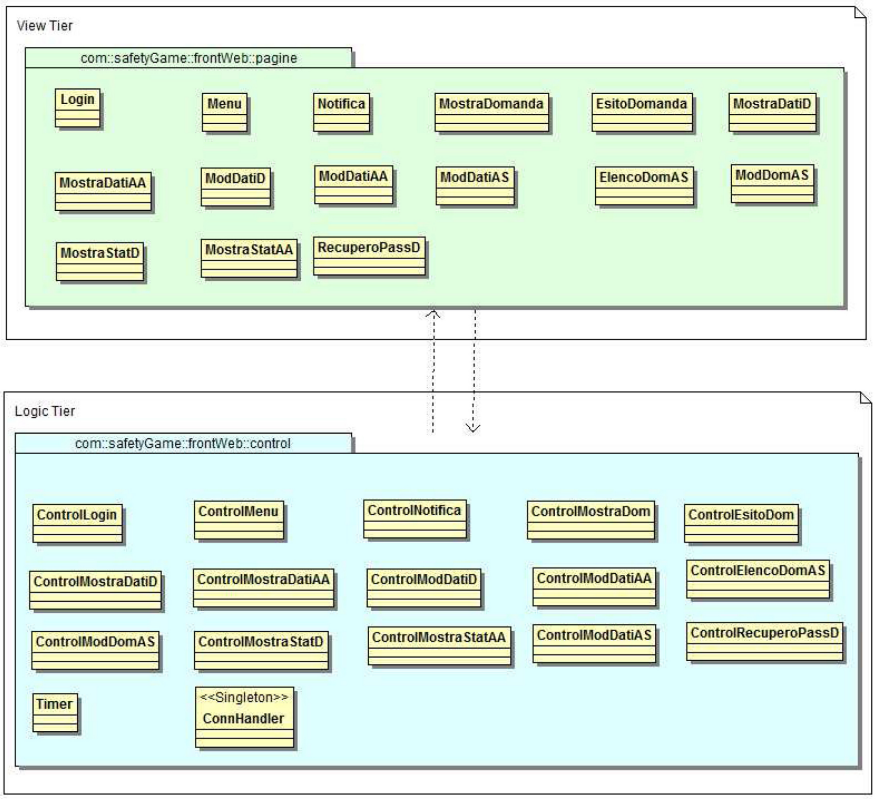

Figure 8: *Front-end Web, Diagramma dei Package*

6 Specifica Front-end Mobile

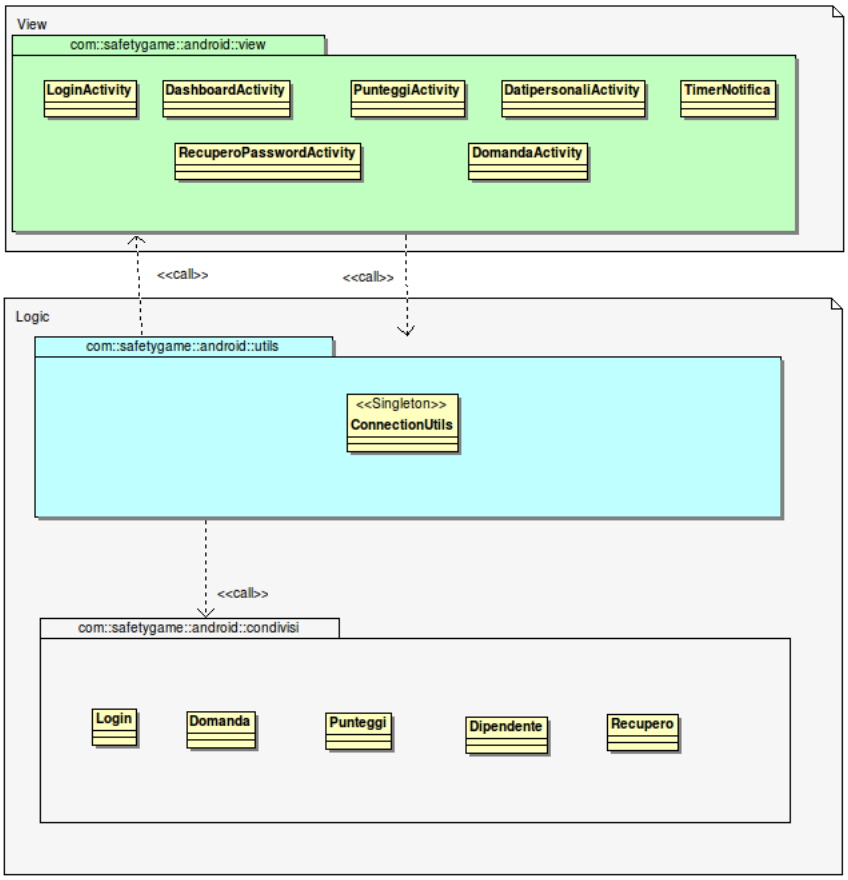

Figura 9: *Front-end Mobile, diagramma dei package*

6.1 Package com.safetyGame.mobile.View

Tipo, obiettivo e funzionamento del componente: Contiene le classi Java che estendono quelle Android "Activity" e che quindi sono inerenti la grafica dell'applicazione. La classe TimenNotifica è parte di questo componente nonostante estenda la classe Java Android "Service".
Relazione d'uso di altre componenti: Vengono utilizzate funzioni del package Utils.
Interfaccie con e relazioni d'uso da altre componenti: Nel momento dell'arrivo dei dati Utils notifica View.

Attività svolte e dati trattati: Definisce l'aspetto dell'applicazione Android definendone l'interfaccia con cui l'utente dovrà interagire.

6.1.1 com.safetyGame.mobile.View.DashboardActivity

Funzione

Permette di raggiungere le altre Activity del programma. Dovrà intercettare gli input dell'utente e lanciare le altre Activity.

Relazioni d'uso con altri moduli

Questa classe utilizzerà le seguenti classi:

- `mobile.View.DomandaActivity`
- `mobile.View.DatiActivity`
- `mobile.View.PunteggiActivity`

Attributi:

- Context context

Metodi

+ void onCreate(Bundle savedInstanceState)

Metodo chiamato alla creazione della classe.

Dovrà prendere il controllo della grafica e disegnare al suo interno i pulsanti per accedere alle seguenti funzionalità:

- **Risposta domanda**
- **Esecuzione quest**
- **Visualizzazione propri punteggi**
- **Visualizzazione propri dati**

  Quindi dovranno essere ridefiniti le azioni di pressione dei pulsanti affinchè creino gli oggetti deputati all'azione richiesta.

6.1.2 com.safetyGame.mobile.View.DashboardLayout

Funzione
Classe che aiuta la visualizzazione della Dashboard all'avvio dell'applicazione.

Relazioni d'uso con altri moduli
Questa classe verrà utilizzata dalle seguenti classi:

Attributi:

- static final int UNEVEN_GRID_PENALTY_MULTIPLIER = 10

- int mMaxChildWidth = 0

- int mMaxChildHeight = 0

Metodi

+ DashboardLayout(Context context)
Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ DashboardLayout(Context context, AttributeSet attrs)
Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

+ DashboardLayout(Context context, AttributeSet attrs, int defStyle)
Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

 # void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
Metodo utilizzato per calcolare la dimensione ideale che la grafica dovrebbe avere a seconda del suo contenuto.
Deve controllare le dimensioni del widget interno più grande, le ricontrolla per essere sicuro che siano quelle, quindi imposta gli attributi *nMaxChildWidth* e *mMaxChildHeight* con quella dimensione.

 # void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
Metodo per determinare la grandezza ideale del layout che dovrebbe impiegare [3].
Dovrà provare calcolando ogni possibile combinazione di righe e colonne, per poi decidere in base al rapporto fra la dimensione degli elementi presenti e lo spazio disponibile rimasto quale sia il miglior rapporto.

[3] Layout calcolato in colonne e righe, come in una tabella

6.1.3 com.safetyGame.mobile.View.DatiActivity

Funzione
Permette ad un Dipendente Autenticato di visionare e modificare i dati personali personali.

Relazioni d'uso con altri moduli
Questa classe verrà utilizzata dalle seguenti classi:

- `mobile.View.DashboardActivity`

Inoltre utilizzerà le seguenti classi:

- `mobile.Utils.ConnectionsUtils`

Attributi

- Context context

Metodi:

+ void onCreate(Bundle savedInstanceState)
Metodo invocato alla creazione dell'oggetto *DatiActivity*.
Dovrà creare un'istanza della classe *DatiTask* e quindi mandarla in esecuzione

+ boolean onOptionsItemSelected(MenuItem item)
Metodo invocato quando si preme un pulsante del menù.
Dovrà controllare quale pulsante è stato premuto tramite il suo id, quindi eseguire l'azione associata.

6.1.3.1 com.safetyGame.mobile.View.DatiActivity.DatiTask

Funzione

Consente ad un Dipendente di effettuare il login di un Dipendente, recuperando i dati immessi dal Dipendente e inviandoli al Back-End per la verifica

Relazioni d'uso con altri moduli

Questa classe utilizzerà le seguenti classi:

- `mobile.Utils.ConnectionUtils`

Attributi

 # ProgressDialog dialog

 # EditText user

 # EditText passw

Metodi

 # void onPreExecute()

Metodo che viene eseguito prima di mostrare la grafica.

Dovrà mostrare all'utente una *ProgressDialog* con scritto sotto "Loading. Please wait..." fino a quando non è stata creata la vista dei campi dati.

 # String doInBackground(Object... params).

Metodo che crea la grafica deputata a raccogliere le credenziali d'accesso al sistema.

Dovrà inserire all'interno dell'interfaccia grafica i due campi dove poter inserire username e password, scrivendone al loro interno l'indicazione di cosa dovrà essere inserito. Quindi, una volta inseriti, dovrà controllare che i campi siano stati compilati; una volta compilati, dovrà interrogare il package mobile.Utils affinchè comunichi con il Back-End per controllare la correttezza dei dati inseriti.

 # void onPostExecute(String status)

Metodo che si occupa di segnalare all'utente la correttezza o meno del login, mostrando a schermo un messaggio d'errore o facendo si che il gestore della grafica mostri tutte le opzioni che erano nascoste all'utente che non abbia fatto accesso all'applicazione.

Dovrà rimuovere dalla grafica il pannello creato con la funzione doInBackground(...) e cotrollare che la stringa passata come parametro non sia vuota e che contenga *"OK"*. Se una di queste due condizioni non dovesse essere verificata, dovrà far comparire un messaggio d'errore all'utente. Altrimenti dovrà notificare al gestore della grafica che il Dipendente è stato identificato, quindi mostrare al Dipendente le opzioni che prima non erano visibili.

6.1.4 com.safetyGame.mobile.View.DomandaActivity

Funzione
Mette a disposizione del Dipendente Autenticato la possibilità di visualizzare, richiedere e rispondere alle domande.

Relazioni d'uso con altri moduli
Questa classe verrà utilizzata dalle seguenti classi:

Attributi:

- Context context

Metodi:

+ **void onCreate(Bundle savedInstanceState)**
Metodo chiamato alla creazione della classe.
Dovrà controllare la tipologia di azione richiesta (Domanda o Quest) e chiamare la sottoclasse corrispondente

+ **boolean onOptionsItemSelected(MenuItem item)**
Metodo invocato quando si preme un pulsante del menù.
Dovrà controllare quale pulsante è stato premuto tramite il suo id, quindi eseguire l'azione associata.

6.1.4.1 com.safetyGame.mobile.View.DomandaActivity.DomandaTask

Funzione
Consente ad un Dipendente di rispondere ad una Domanda

Relazioni d'uso con altri moduli
Questa classe utilizzerà le seguenti classi:

- `mobile.Utils.ConnectionUtils`

Attributi

ProgressDialog dialog

EditText user

EditText passw

Metodi

void onPreExecute()
Metodo che viene eseguito prima di mostrare la grafica.
Dovrà mostrare all'utente una *ProgressDialog* con scritto sotto "Loading. Please wait..." fino a quando non è stata creata la vista dei campi dati.

String doInBackground(Object... params).
Metodo che recupera dal Back-End la domanda che dovrà essere sottoposta al Dipedente.

void onPostExecute(String status)
Metodo che crea la grafica deputata a mostrare la domanda e a ricevere la risposta che il Dipendente ha dato alla suddetta domanda
Dovrà inserire all'interno dell'interfaccia grafica il testo della domanda e le possibili risposte con a fianco un pulsante che permetterà di selezionarne solo una alla volta. Inoltre dovrà essere presente un pulsante che permetterà di inviare la risposta al Back-end per verificare che questa sia corretta

6.1.4.2 com.safetyGame.mobile.View.DomandaActivity.QuestTask

Funzione
Consente ad un Dipendente di effettuare una nuova Quest

Relazioni d'uso con altri moduli
Questa classe utilizzerà le seguenti classi:

- `mobile.Utils.ConnectionUtils`

Attributi

ProgressDialog dialog

EditText user

EditText passw

Metodi

void onPreExecute()
Metodo che viene eseguito prima di mostrare la grafica.
Dovrà mostrare all'utente una *ProgressDialog* con scritto sotto "Loading. Please wait..." fino a quando non è stata creata la vista dei campi dati.

String doInBackground(Object... params).
Metodo che recupera dal Back-End la domanda che dovrà essere sottoposta al Dipedente.

void onPostExecute(String status)
Metodo che crea la grafica deputata a mostrare la domanda e a ricevere la risposta che il Dipendente ha dato alla suddetta domanda
Dovrà inserire all'interno dell'interfaccia grafica il testo della domanda e dovrà integrare un lettore di codici sia QR che a barre. Inoltre dovrà essere presente un pulsante che permetterà di inviare la risposta al Back-end per verificare che questa sia corretta

6.1.5 com.safetyGame.mobile.View.LoginActivity

Funzione

Classe di gestione della parte grafica dell'attività di login, gestendo l'ingresso alla pagina che permette l'inserimento delle credenziali d'accesso e contenendo la classe deputata alla visualizzazione dei campi dati per l'immissione

Relazioni d'uso con altri moduli

Questa classe utilizzerà le seguenti classi:

- `mobile.View.DomandaActivity.LoginTask` *(classe privata interna alla classe LoginActivity)*
- `mobile.Utils.ConnectionUtils`

Attributi

- Context context

Metodi

+ void onCreate(Bundle savedInstanceState)

Metodo per la creazione dell'oggetto LoginActivity. Questo oggetto verrà creato all'avvio dell'applicazione.

Dovrà creare un pulsante che permetta l'accesso alla funzionalità di login, collegando l'azione di pressione del pulsante alla creazione di un oggetto *LoginTask* che permetterà l'inserimento delle credenziali d'accesso al sistema. Il pulsante così creato dovrà poi essere inserito all'interno della grafica dell'applicazione.

6.1.5.1 com.safetyGame.mobile.View.LoginActivity.LoginTask

Funzione

Consente ad un Dipendente di effettuare il login di un Dipendente, recuperando i dati immessi dal Dipendente e inviandoli al Back-End per la verifica

Relazioni d'uso con altri moduli

Questa classe utilizzerà le seguenti classi:

- `mobile.Utils.ConnectionUtils`

Attributi

 # ProgressDialog dialog

 # EditText user

 # EditText passw

Metodi

 # void onPreExecute()

Metodo che viene eseguito prima di mostrare la grafica.

Dovrà mostrare all'utente una *ProgressDialog* con scritto sotto "Loading. Please wait..." fino a quando non è stata creata la vista dei campi dati.

 # String doInBackground(Object... params).

Metodo che crea la grafica deputata a raccogliere le credenziali d'accesso al sistema.

Dovrà inserire all'interno dell'interfaccia grafica i due campi dove poter inserire username e password, scrivendone al loro interno l'indicazione di cosa dovrà essere inserito. Quindi, una volta inseriti, dovrà controllare che i campi siano stati compilati; una volta compilati, dovrà interrogare il package mobile.Utils affinchè comunichi con il Back-End per controllare la correttezza dei dati inseriti.

 # void onPostExecute(String status)

Metodo che si occupa di segnalare all'utente la correttezza o meno del login, mostrando a schermo un messaggio d'errore o facendo si che il gestore della grafica mostri tutte le opzioni che erano nascoste all'utente che non abbia fatto accesso all'applicazione.

Dovrà rimuovere dalla grafica il pannello creato con la funzione doInBackground(...) e cotrollare che la stringa passata come parametro non sia vuota e che contenga *"OK"*. Se una di queste due condizioni non dovesse essere verificata, dovrà far comparire un messaggio d'errore all'utente. Altrimenti dovrà notificare al gestore della grafica che il Dipendente è stato identificato, quindi mostrare al Dipendente le opzioni che prima non erano visibili.

6.1.6 com.safetyGame.mobile.View.PunteggiActivity

Funzione

Permette ad un Dipendente Autenticato di visualizzare i vari punteggi e trofei.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- Context context

Metodi:

+ void onCreate(Bundle savedInstanceState)

Metodo invocato alla selezione dell'azione di visualizzazione dei punteggi.
Dovrà creare un oggetto anonimo di tipo *PunteggiTask* ed eseguirlo.

+ boolean onOptionsItemSelected(MenuItem item)

Metodo invocato quando si preme un pulsante del menù.
Dovrà controllare quale pulsante è stato premuto tramite il suo id, quindi eseguire l'azione associata.

6.1.6.1 com.safetyGame.mobile.View.PunteggiActivity.PunteggiTask

Funzione

Permette ad un Dipendente Autenticato di visualizzare i vari punteggi e trofei.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- Context context

Metodi:

 # void onCreate(Bundle savedInstanceState)

Metodo che viene eseguito prima di mostrare la grafica.

Dovrà mostrare all'utente una *ProgressDialog* con scritto sotto "Loading. Please wait..." fino a quando non è stata creata la vista dei campi dati.

 # Punteggi doInBackground(Object... params)

Metodo che interroga il Back-end per recuperare i punteggi del Dipendente.

Dovrà mandare una richiesta al Back-end interrogando la pagina deputata a restituire un oggetto di tipo *Punteggi* e restituirlo al chiamante

 # void onPostExecute(Punteggi punteggi)

Metodo per la visualizzazione dei punteggi.

Dovrà far visualizzare le statistiche del Dipendente mostrando:

- Risposte date
- Risposte corrrette
- Risposte errate
- Punteggio ottenuto

Nel caso ci siano problemi, dovrà far visualizzare un messaggio d'errore.

6.1.7 com.safetyGame.mobile.View.TimerNotifica

Funzione

Servizio di temporizzazioen che propone al Dipendente Autenticato una domanda secondo quanto impostato tramite le notifiche standard di Android.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi:

- long mStartTime

- Handler mHandler = new Handler()

- NotificationManager mNotificationManager

- Notification notification

- SaredPreferences prefs

Metodi:

+ int onStartCommand(Intent intent, int flags, int startId)

+ IBinder onBind(Intent arg0)

6.1.7.1 com.safetyGame.mobile.View.TimerNotifica.mUpdateTimeTask

Funzione
Questa classe si occuperà di gestire le operazioni che l'Amministratore Sicurezza può effettuare sui Badge. In particolare potrà recuperare la lista di tutti i badge disponibili all'interno dell'azienda.

Relazioni d'uso con altri moduli
Questa classe verrà utilizzata dalle seguenti classi:

Tipo, obiettivo e funzionamento del componente: Servizio di temporizzazioen che propone al Dipendente Autenticato una domanda secondo quanto impostato tramite le notifiche standard di Android.
Relazione d'uso di altre componenti: Viene chiamata la classe DomandaActivity.
Interfaccie con e relazioni d'uso da altre componenti: Nessuna.
Attività svolte e dati trattati: Un timer, in accordo con quanto impostato di default nel dispositivo Android in cui sarà installato il software, farà comparire una notifica standard di Android che permette di visualizzare una nuova domanda.

Attributi:

Metodi:

\+ **void run()**

6.2 Package com.safetyGame.mobile.Utils

Tipo, obiettivo e funzionamento del componente: Vi appartengono le classi che comunicano con le API del server. Queste classi ricevono i dati e notificano la View.
Relazione d'uso di altre componenti: Comunica con il back-end e quando i dati sono pronti invia notifiche al componente View.
Interfaccie con e relazioni d'uso da altre componenti: Viene utilizzato dalla View.
Attività svolte e dati trattati: Invia richieste HTTP al server, il quale gli risponderà inviando i dati richiesti attraverso un XML. In seguito ne estrarrà i dati e li renderà disponibili alla View.

6.2.1 com.safetyGame.mobile.Utils.BootReceiver

Funzione
Questa classe si occuperà di gestire le operazioni che l'Amministratore Sicurezza può effettuare sui Badge. In particolare potrà recuperare la lista di tutti i badge disponibili all'interno dell'azienda.

Relazioni d'uso con altri moduli
Questa classe verrà utilizzata dalle seguenti classi:

Tipo, obiettivo e funzionamento del componente: Classe che estende BroadcastReceiver e viene automaticamente chiamata al completamento del boot del dispositivo.
Relazione d'uso di altre componenti: Fa partire il service com.safetygame.mobile.View.TimerNotifica .
Interfaccie con e relazioni d'uso da altre componenti: Nessuna
Attività svolte e dati trattati: La classe fa avviare il servizio in bacground che propone le domande.

Attributi

Metodi:

+ **onReceive(Context context, Intent arg1)**

6.2.2 com.safetyGame.mobile.Utils.ConnectionUtils

Funzione

La classe è utilizzata per effettuare connessioni fisiche tra il front-end Mobile ed il back-end, ricevere dati, effettuare parser e creare gli opportuni oggetti da notificare alla View.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

Metodi:

- static Element rootXML(HttpResponse response)

+ static String parseXML(Element root, String stringa, int position)

+ static Object HttpCreateClient(String url, List<NameValuePair> nameValuePairs)

6.2.3 com.safetyGame.mobile.Utils.IntentIntegrator

Funzione

Classe che aiuta l'integrazione dell'applicazione Safety Game con l'applicazione Barcode Scanner.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

+ static final int REQUEST_CODE = 0x0000c0de

- static final String TAG = IntentIntegrator.class.getSimpleName()

+ static final String DEFAULT_TITLE = "Install Barcode Scanner?"

+ static final String DEFAULT_MESSAGE = "This application requires Barcode Scanner. Would you like to install it?"

+ static final String DEFAULT_YES = "Yes"

+ static final String DEFAULT_NO = "No"

- static final String BS_PACKAGE = "com.google.zxing.client.android"

+ static final Collection<String> PRODUCT_CODE_TYPES = list("UPC_A", "UPC_E", "EAN_8", "EAN_13", "RSS_14")

+ static final Collection<String> ONE_D_CODE_TYPES = list("UPC_A", "UPC_E", "EAN_8", "EAN_13", "CODE_39", "CODE_93", "CODE_128", "ITF", "RSS_14", "RSS_EXPANDED")

+ static final Collection<String> QR_CODE_TYPES = Collections.singleton("QR_CODE");

+ static final Collection<String> DATA_MATRIX_TYPES = Collections.singleton("DATA_MATRIX")

+ static final Collection<String> ALL_CODE_TYPES = null;

+ static final Collection<String> TARGET_BARCODE_SCANNER_ONLY = Collections.singleton(BS_PACKAGE)
+ static final Collection<String> TARGET_ALL_KNOWN = list(BS_PACKAGE, "com.srowen.bs.android", "com.srowen.bs.android.simple")

- final Activity activity

- String title

- String message

- String buttonYes

- String buttonNo

- Collection<String> targetApplications

Metodi

+ IntentIntegrator(Activity activity)

+ AlertDialog initiateScan()

+ AlertDialog initiateScan(Collection<String> desiredBarcodeFormats)

 # void startActivityForResult(Intent intent, int code)

- AlertDialog showDownloadDialog()

- String findTargetAppPackage(Intent intent)

+ static IntentResult parseActivityResult(int requestCode, int resultCode, Intent intent)

+ void shareText(CharSequence text)

- private static Collection<String> list(String... values)

+ String getTitle()

+ void setTitle(String title)

+ void setTitleByID(int titleID)

+ String getMessage()

+ void setMessage(String message)

+ void setMessageByID(int messageID)

+ String getButtonYes()

+ void setButtonYes(String buttonYes)

+ void setButtonYesByID(int buttonYesID)

+ String getButtonNo()

+ void setButtonNo(String buttonNo)

+ void setButtonNoByID(int buttonNoID)

+ Collection<String> getTargetApplications(Collection<String> targetApplications)

+ void setTargetApplications(Collection<String> targetApplications)

+ void setSingleTargetApplication(String targetApplication)

6.2.4 com.safetyGame.mobile.Utils.IntentResult

Funzione
Classe che aiuta a gestire i risultati dello scan del barcode.

Relazioni d'uso con altri moduli
Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- final String contents

- final String formatName

- final byte[] rawBytes

- final Integer orientation

- final String errorCorrectionLevel

Metodi

+ IntentResult(String contents, String formatName, byte[] rawBytes, Integer orientation, String errorCorrectionLevel)

+ String getContents()

+ String getFormatName()

+ byte[] getRawBytes()

+ Integer getOrientation()

+ String getErrorCorrectionLevel()

+ String toString()

6.3 Package com.safetyGame.mobile.condivisi

Tipo, obiettivo e funzionamento del componente: Package che raggruppa tutti i tipi di dato non nativi usati dall'applicazione.
Relazione d'uso di altre componenti:
Interfaccie con e relazioni d'uso da altre componenti:
Attività svolte e dati trattati:

6.3.1 com.safetyGame.mobile.condivisi.Dati

Funzione

Questa classe si contenere i dati dei Dipendenti

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- String nome

- String cognome

Metodi

public Dati(String nome, String cognome)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ String getNome()

+ String getCognome()

6.3.2 com.safetyGame.mobile.condivisi.Domanda

Funzione

Questa classe fungerà da contenitore per le domande.

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- String type

- String title

- String testo

- String[] risposte

Metodi

public Domanda(String type, String title, String testo)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ String getType()

+ String getTitle()

+ String getTesto()

+ String[] getRisposte()

6.3.3 com.safetyGame.mobile.condivisi.Punteggi

Funzione

Questa classe fungerà da contenitore per i dati riguardanti i punteggi dei dipendenti

Relazioni d'uso con altri moduli

Questa classe verrà utilizzata dalle seguenti classi:

Attributi

- String rispostedate

- String rispostecorrette

- String risposteerrate

- String punti

- String[] badges

Metodi

+ Punteggi(String rispostedate, String rispostecorrette, String risposteerrate, String punti, String[] badge, int numeroBadge)

Costruttore che dovrà impostare il valore degli attributi secondo il valore dei parametri passati.

Vengono quindi definiti i metodi getter/setter per i vari attributi della classe. In particolare:

+ String getRispostedate()

+ String getRispostecorrette()

+ String getRisposteerrate()

+ String getPunti()

+ String[] getBadges()

6.3.4 com.safetyGame.mobile.condivisi.Quest

Funzione
Questa classe si occuperà di gestire le operazioni che l'Amministratore Sicurezza può effettuare sui Badge. In particolare potrà recuperare la lista di tutti i badge disponibili all'interno dell'azienda.

Relazioni d'uso con altri moduli
Questa classe verrà utilizzata dalle seguenti classi:

Variabili:
- String title
- String testo

Metodi:
+ String getTitle()
+ String getTesto()

7 Tracciamento requisiti