PROGETTO-G

Collaboratori

- Casadei Alberto
- La Salvia Marco
- Ricotti Margherita
- Groppi Alberto
- Farina Nicholas
- Delbò Davide

Rooming

Requisiti funzionali

- Un utilizzatore può effettuare la ricerca di una casa in base a diversi parametri.
- Un utilizzatore può inoltre effettuare la ricerca di un possibile coinquilino.
- Un utente può candidarsi come possibile coinquilino.
- Un guest può registrarsi o, in caso di avvenuta registrazione, effettuare login o logout.
- Un utente può creare un annuncio casa.
- Un utente può modificare i dati del proprio profilo e del suo annuncio.

Rooming

Requisiti non funzionali

- Presenza di un interfaccia web.
- Presenza di una base di dati.

Rooming

Requisiti di dominio

- Un utente può creare solo un annuncio
  (viene data per scontata l'appartenenza alla casa del creatore dell'annuncio).
- Un utente registrato non è di default un candidato coinquilino, ma è un'opzione che può essere aggiunta attraverso la modifica del proprio profilo.
- Un utente può creare un annuncio solo inserendo tutti i dati richiesti.
- Ai guest non sono visibili i contatti presenti nei risultati delle ricerche, è richiesto il login.

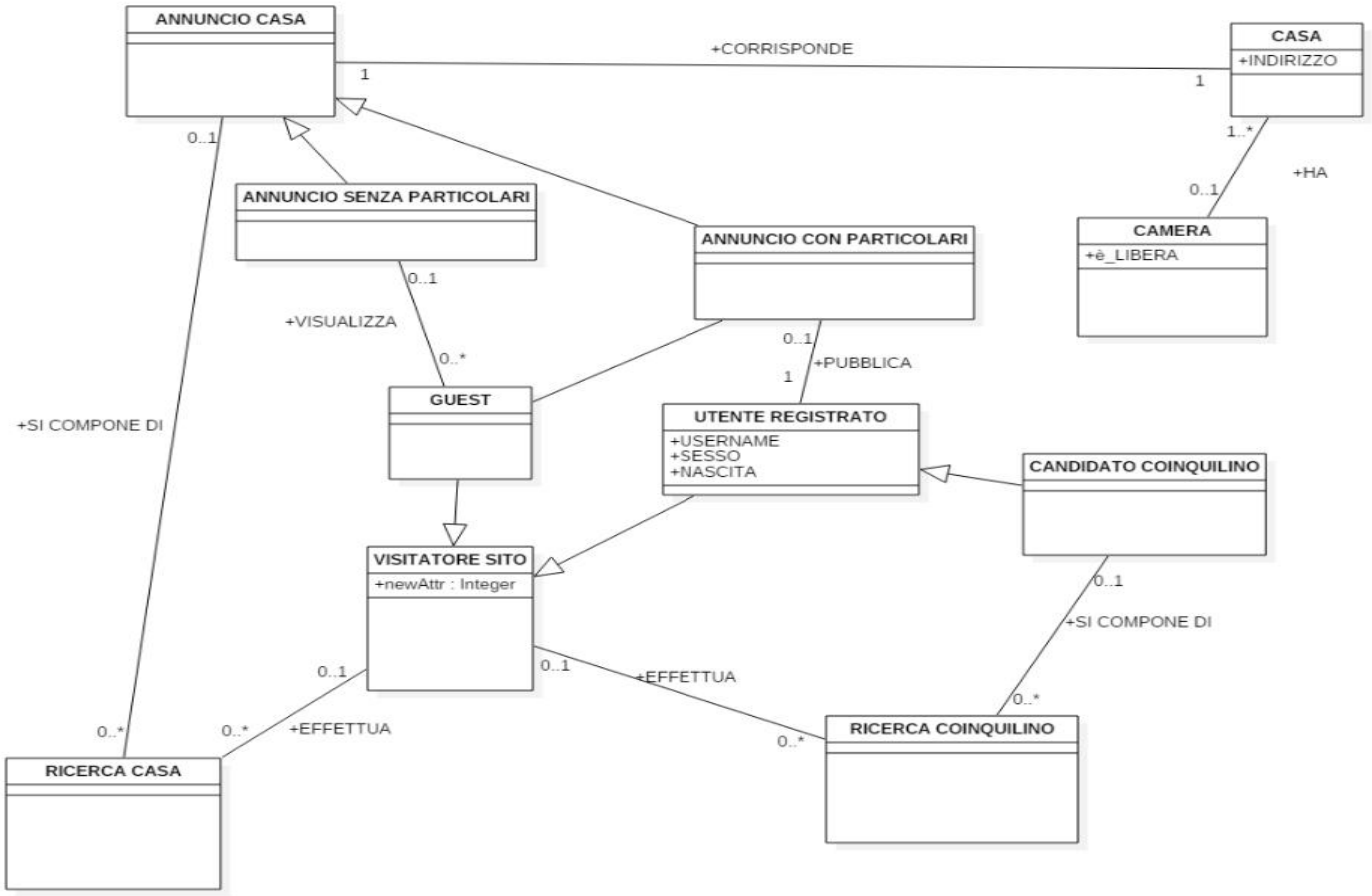
Rooming

Caso d'uso: ricerca coinquilino

- DESCRIZIONE: Utente che cerca un coinquilino
- STAKEHOLDER: Utente che si candida come potenziale coinquilino
- PRECONDIZIONE: Utente loggato
- SCENARIO PRINCIPALE: Accedere alla schermata della ricerca
- Inserire le specifiche obbligatorie della ricerca
- Inserire filtri, dettagli per migliorare la ricerca
- Inserire i pesi relativi ad ogni parametro per il calcolo dell'affinità
- Selezione di un utente tra i risultati
- Visualizzazione del profilo dell'utente
- SCENARI ALTERNATIVI
  - Utente non loggato:
    - Accedere alla schermata di ricerca
    - Inserire parametri della ricerca
    - Selezione di un annuncio tra i risultati
    - Non verranno visualizzati i contatti, ma solo le altre informazioni sul profilo dell'utente trovato
- Nessun coinquilino soddisfa le specifiche della ricerca:
  - Effettua passi 1,2,3 dello scenario principale
  - Se la ricerca non da' risultati, lo scenario fallisce e si ritorna alla schermata di ricerca

Rooming

Modello di dominio



Rooming

Architettura

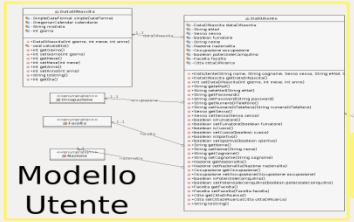
- E' stata utilizzata un'architettura a strati con la tecnica pull from above.

  Poichè il sistema deve essere distribuito, l'iterazione client-server avviene tramite messaggi HTTP.
- Il client (browser) è di tipo Thin Client.
- L'architettura client-server è nascosta dalla libreria Jetty.

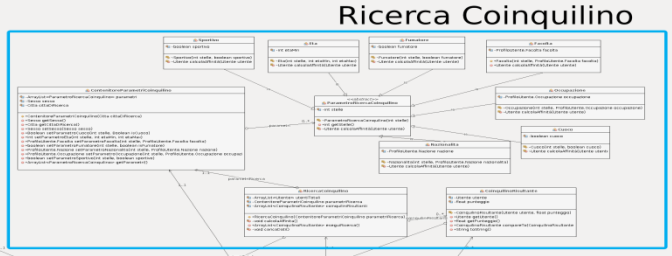
Rooming

UML

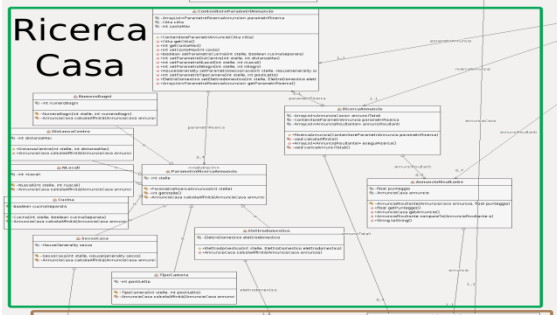
Eccezioni



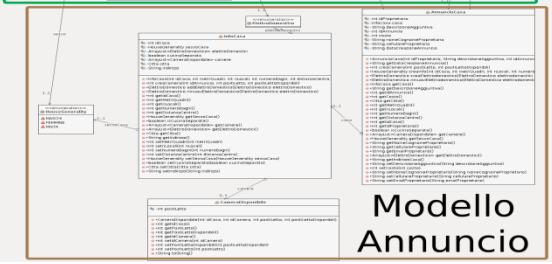
Modello Utente



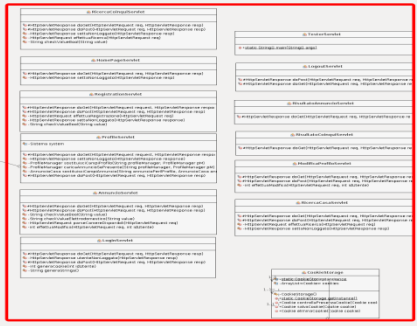
Ricerca Coinquilino



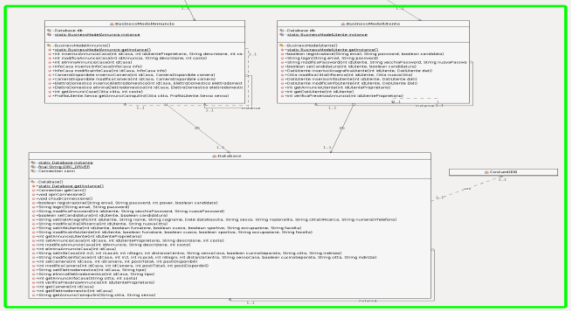
Ricerca Casa



Modello Annuncio



Servlet

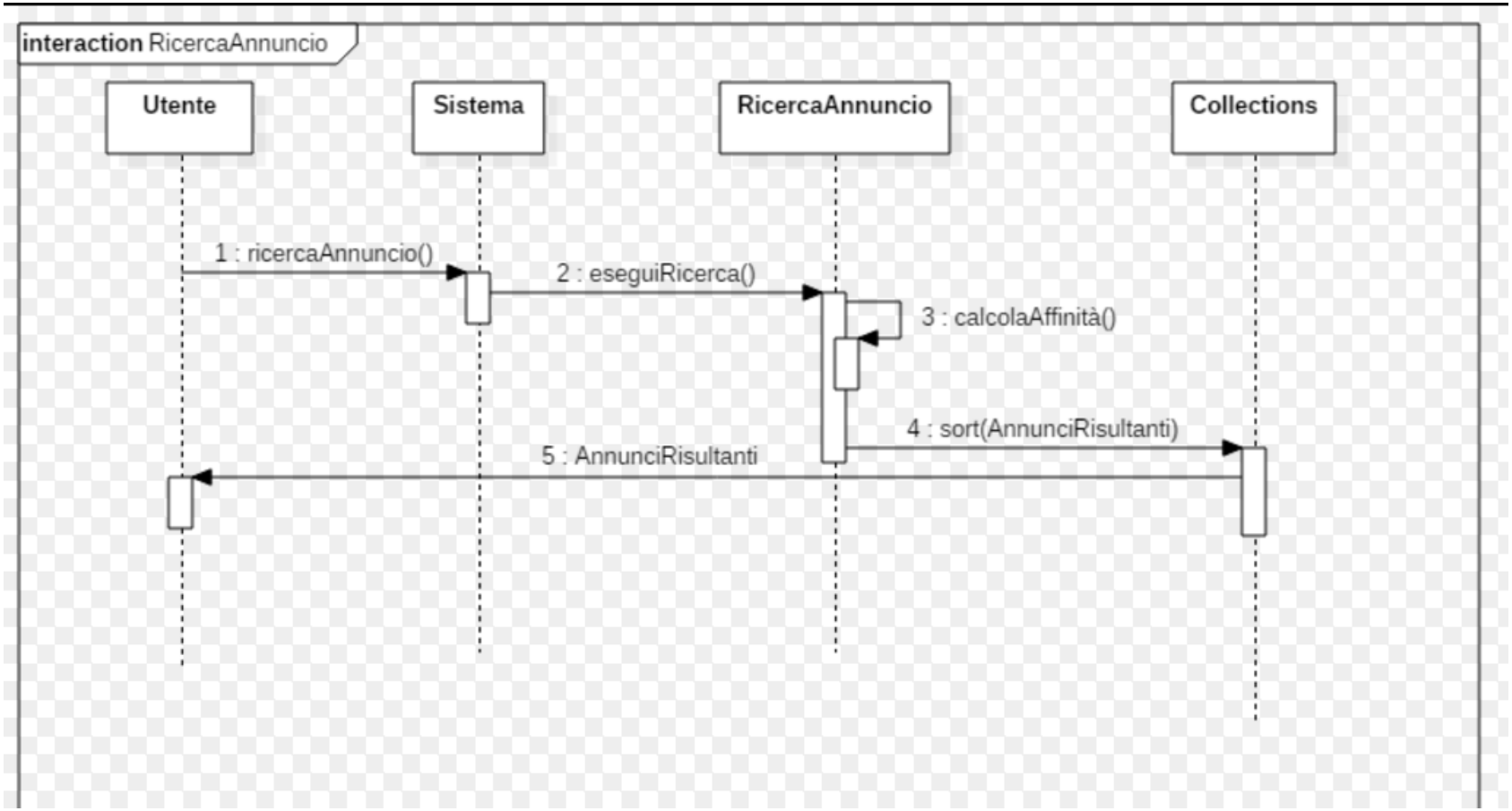


Database

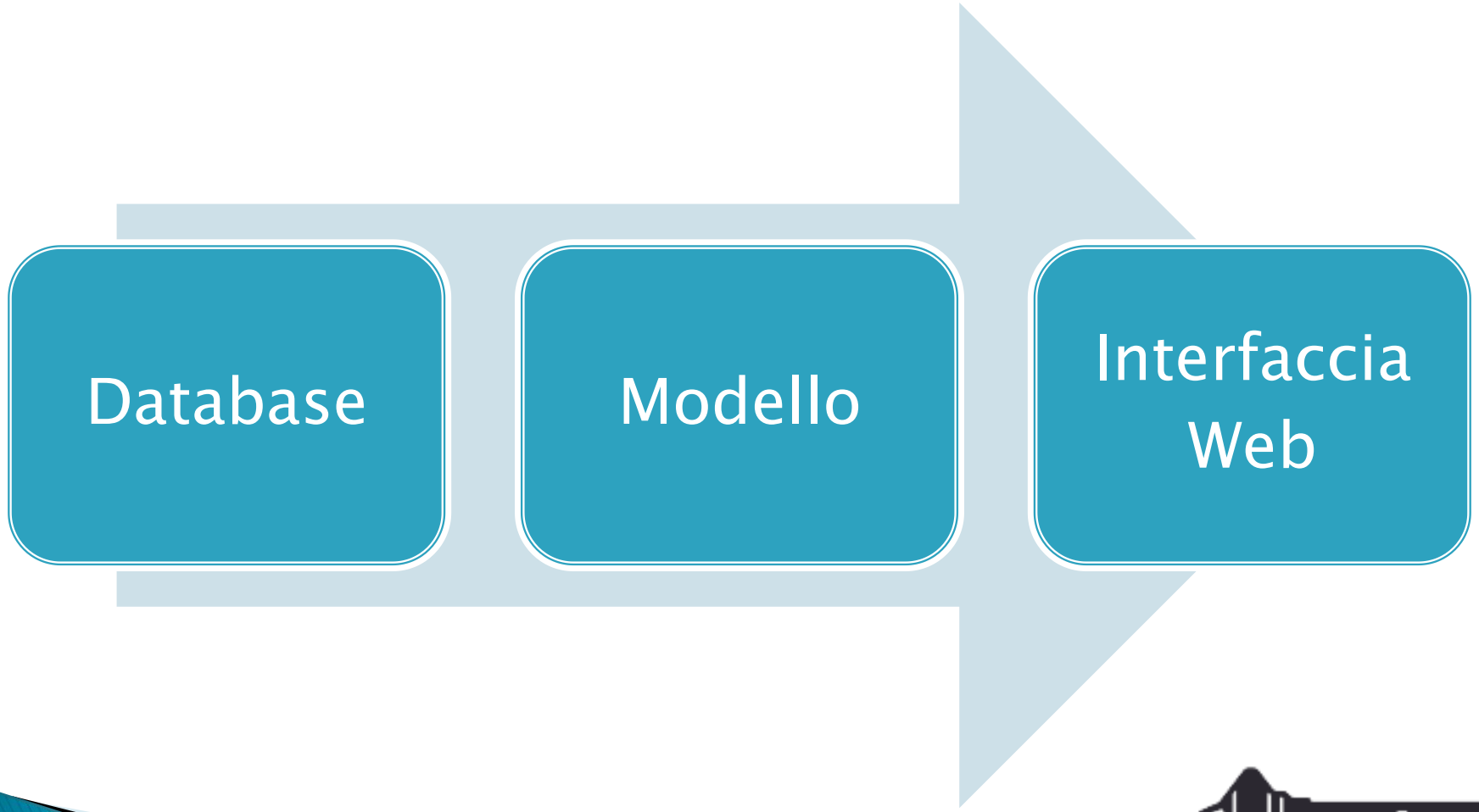
Creatori HTML

Rooming

Diagramma di sequenza



Rooming



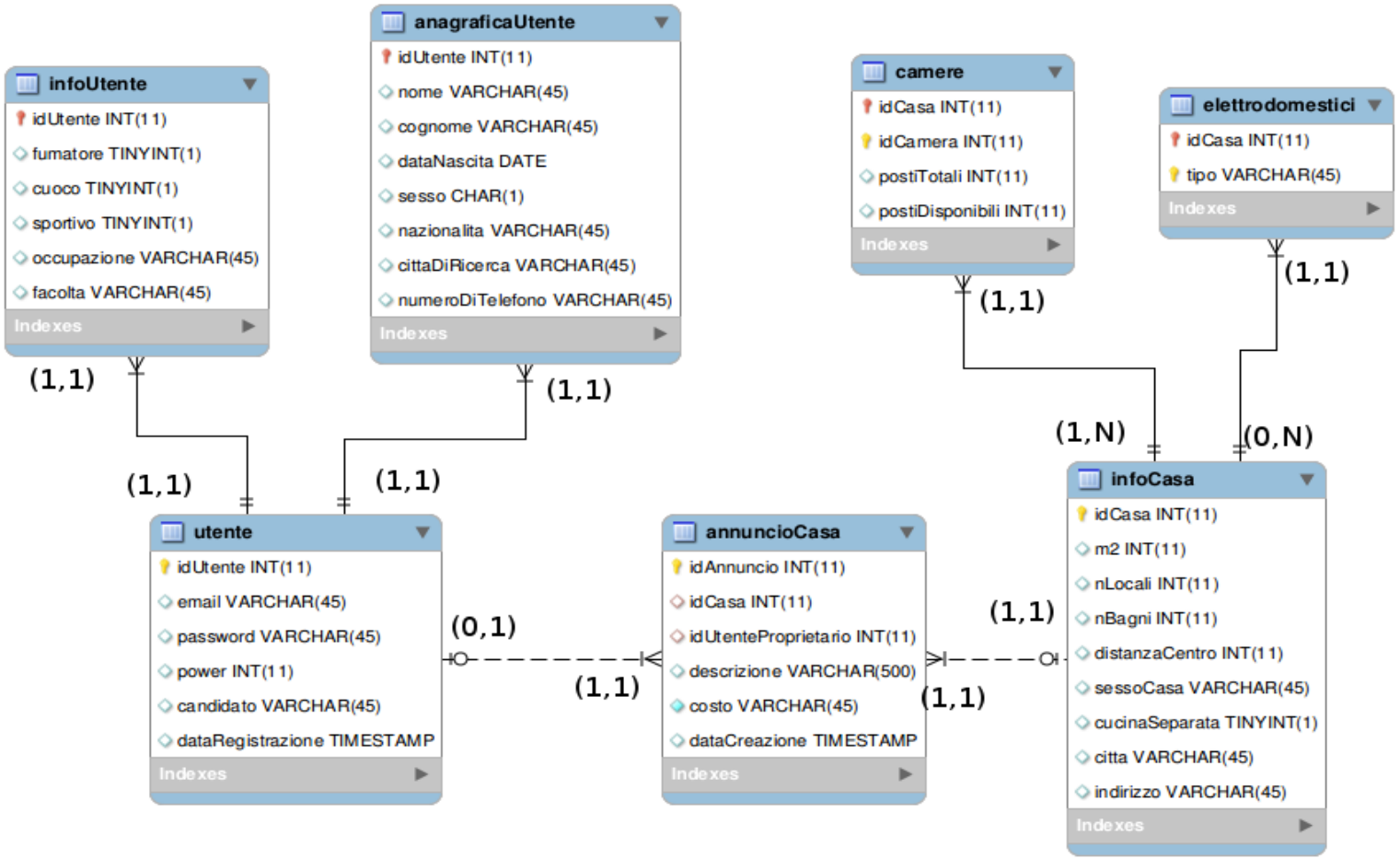
Rooming

Database

- Utilizzo del servizio Amazon Web Services per l'hosting.
- Database di tipo relazionale (MySQL).
- Utilizzato il driver JDBC per la connessione al modello.

Rooming

Database



Rooming

Data Access Layer

- Database:
  - Utilizzato il pattern *Singleton.*
  - Interfaccia diretta tra classe “database” e database stesso tramite l’esecuzione di query.
  - Classe ad hoc per il codice SQL.
- Business Model:
  - Utilizzato il pattern *Singleton.*
  - Elabora i risultati delle query su database e li restituisce al modello in un formato noto.
  - Sono presenti due classi *BusinessModel* : una per gli annunci e una per gli utenti.

Rooming

Esempio: setInfoCasa

```
public int modificaInfoCasa(int idCasa, int m2, int nLocali, int
   nBagni, int distanzaCentro,
          String sessoCasa, boolean cucinaSeparata, String citta,
   String indirizzo) throws SQLException {

      PreparedStatement ps =
   conn.prepareStatement(CostantiDB.modificaInfoCasa);
      ps.setInt(1, m2);
      ps.setInt(2, nLocali);
      ps.setInt(3, nBagni);
      ps.setInt(4, distanzaCentro);
      ps.setString(5, sessoCasa);
      ps.setBoolean(6, cucinaSeparata);
      ps.setString(7, citta);
      ps.setString(8, indirizzo);
      ps.setInt(9, idCasa);
      return ps.executeUpdate();
   }
```

Rooming

Esempio: modificaInfoCasa

```
public ResultSet setInfoCasa(int m2, int nLocali, int nBagni, int
    distanzaCentro,
            String sessoCasa, boolean cucinaSeparata, String citta, String
    indirizzo) throws SQLException {

        PreparedStatement ps =
    conn.prepareStatement(CostantiDB.inserisciInfoCasa,
    Statement.RETURN_GENERATED_KEYS);
        ps.setInt(1, m2);
        ps.setInt(2, nLocali);
        ps.setInt(3, nBagni);
        ps.setInt(4, distanzaCentro);
        ps.setString(5, sessoCasa);
        ps.setBoolean(6, cucinaSeparata);
        ps.setString(7, citta);
        ps.setString(8, indirizzo);
        ps.executeUpdate();
        return ps.getGeneratedKeys();
    }
```

Rooming

Modello

Gestisce :

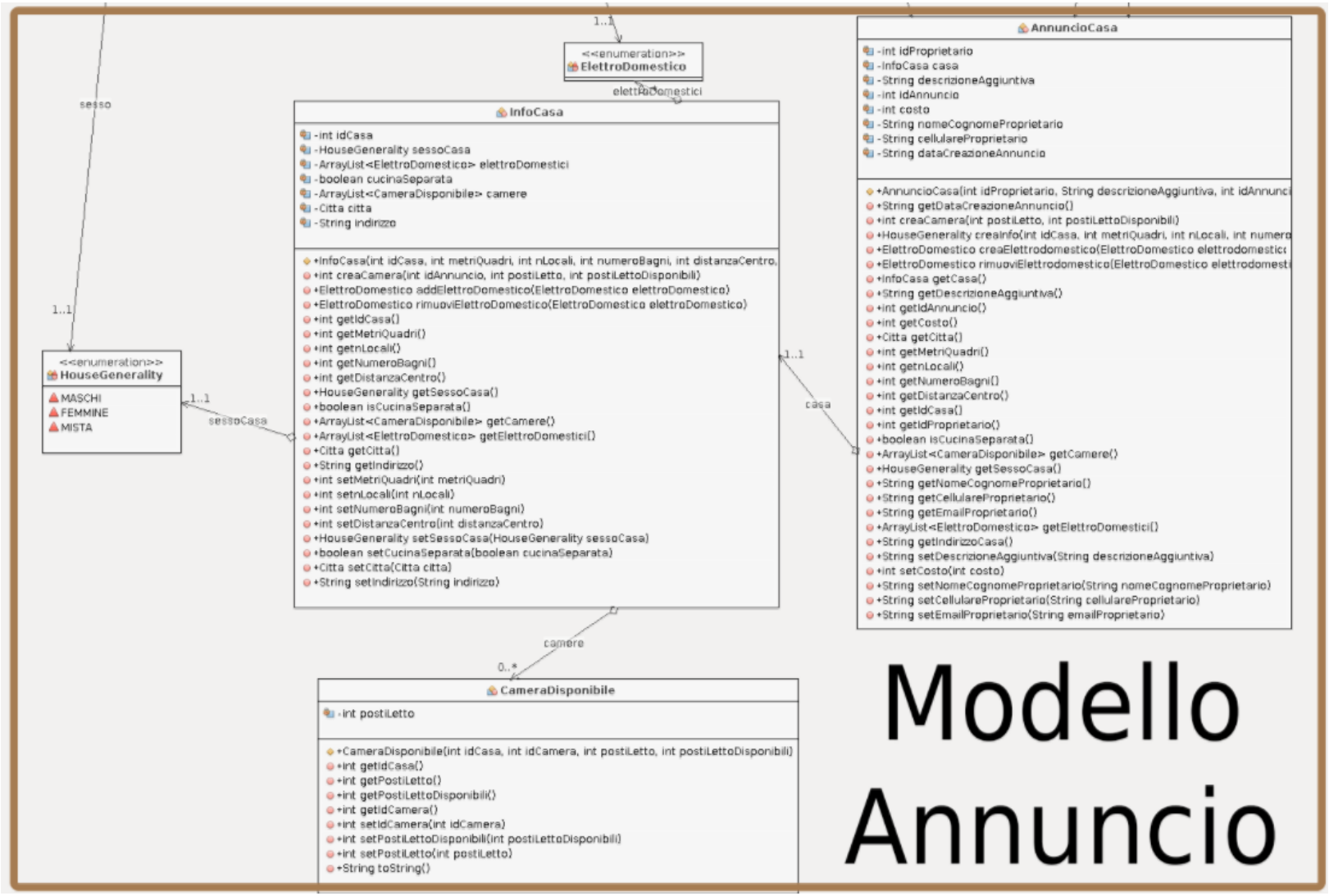
- Registrazione di nuovi utenti.
- Login e Logout.
- Operazioni degli utenti tramite interfaccia web.
- Collegamento al database tramite Business Model.
- Calcolo “affinità” tramite algoritmo ad hoc.

Rooming

Annuncio

- Un utente può creare un solo annuncio tramite il proprio profilo.
- Un annuncio può essere modificato in ogni suo aspetto.
- L'annuncio sarà visibile tra i risultati della ricerca di annunci di altri utilizzatori.

Rooming

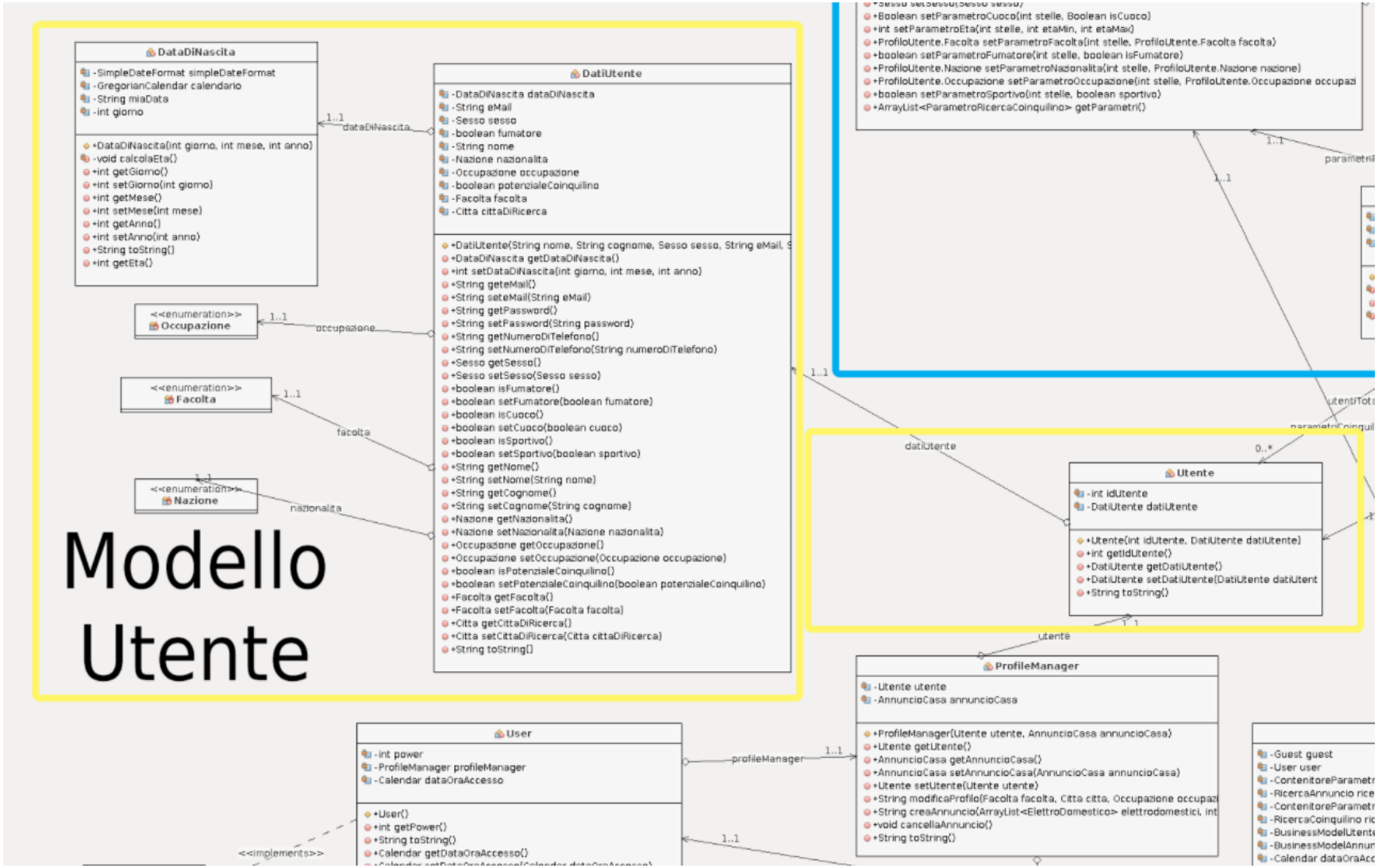

1..1
<<enumeration>>
ElettroDomestico
elettroDomestici
sesso
1..1
<<enumeration>>
HouseGenerality
MASCHI
FEMMINE
MISTA
1..1
sessoCasa
InfoCasa
-int idCasa
-HouseGenerality sessoCasa
-ArrayList<ElettroDomestico> elettroDomestici
-boolean cucinaSeparata
-ArrayList<CameraDisponibile> camere
-Citta citta
-String indirizzo
+InfoCasa(int idCasa, int metriQuadri, int nLocali, int numeroBagni, int distanzaCentro,
+int creaCamera(int idAnnuncio, int postiLetto, int postiLettoDisponibili)
+ElettroDomestico addElettroDomestico(ElettroDomestico elettroDomestico)
+ElettroDomestico rimuoviElettroDomestico(ElettroDomestico elettroDomestico)
+int getIdCasa()
+int getMetriQuadri()
+int getnLocali()
+int getNumeroBagni()
+int getDistanzaCentro()
+HouseGenerality getSessoCasa()
+boolean isCucinaSeparata()
+ArrayList<CameraDisponibile> getCamere()
+ArrayList<ElettroDomestico> getElettroDomestici()
+Citta getCitta()
+String getIndirizzo()
+int setMetriQuadri(int metriQuadri)
+int setnLocali(int nLocali)
+int setNumeroBagni(int numeroBagni)
+int setDistanzaCentro(int distanzaCentro)
+HouseGenerality setSessoCasa(HouseGenerality sessoCasa)
+boolean setCucinaSeparata(boolean cucinaSeparata)
+Citta setCitta(Citta citta)
+String setIndirizzo(String indirizzo)
1..1
casa
AnnuncioCasa
-int idProprietario
-InfoCasa casa
-String descrizioneAggiuntiva
-int idAnnuncio
-int costo
-String nomeCognomeProprietario
-String cellulareProprietario
-String dataCreazioneAnnuncio
+AnnuncioCasa(int idProprietario, String descrizioneAggiuntiva, int idAnnunci
+String getDataCreazioneAnnuncio()
+int creaCamera(int postiLetto, int postiLettoDisponibili)
+HouseGenerality creaInfo(int idCasa, int metriQuadri, int nLocali, int numero
+ElettroDomestico creaElettrodomestico(ElettroDomestico elettrodomestico
+ElettroDomestico rimuoviElettrodomestico(ElettroDomestico elettrodomesti
+InfoCasa getCasa()
+String getDescrizioneAggiuntiva()
+int getIdAnnuncio()
+int getCosto()
+Citta getCitta()
+int getMetriQuadri()
+int getnLocali()
+int getNumeroBagni()
+int getDistanzaCentro()
+int getIdCasa()
+int getIdProprietario()
+boolean isCucinaSeparata()
+ArrayList<CameraDisponibile> getCamere()
+HouseGenerality getSessoCasa()
+String getNomeCognomeProprietario()
+String getCellulareProprietario()
+String getEmailProprietario()
+ArrayList<ElettroDomestico> getElettroDomestici()
+String getIndirizzoCasa()
+String setDescrizioneAggiuntiva(String descrizioneAggiuntiva)
+int setCosto(int costo)
+String setNomeCognomeProprietario(String nomeCognomeProprietario)
+String setCellulareProprietario(String cellulareProprietario)
+String setEmailProprietario(String emailProprietario)
camere
0..*
CameraDisponibile
-int postiLetto
+CameraDisponibile(int idCasa, int idCamera, int postiLetto, int postiLettoDisponibili)
+int getIdCasa()
+int getPostiLetto()
+int getPostiLettoDisponibili()
+int getIdCamera()
+int setIdCamera(int idCamera)
+int setPostiLettoDisponibili(int postiLettoDisponibili)
+int setPostiLetto(int postiLetto)
+String toString()
Modello
Annuncio

Rooming

Profilo utente

- Un utente registrato potrà visualizzare il suo profilo.
- L'utente, dal proprio profilo, potrà creare un annuncio casa che potrà essere poi eliminato o modificato.
- Alcune delle informazioni personali potranno essere modificate tramite Profile Manager.
- Dal proprio profilo un utente può candidarsi come coinquilino, diventando visibile dalle ricerche di altri utenti.

Rooming


DataDiNascita
-SimpleDateFormat simpleDateFormat
-GregorianCalendar calendario
-String miaData
-int giorno
+DataDiNascita(int giorno, int mese, int anno)
-void calcolaEta()
+int getGiorno()
+int setGiorno(int giorno)
+int getMese()
+int setMese(int mese)
+int getAnno()
+int setAnno(int anno)
+String toString()
+int getEta()
1..1
dataDiNascita
DatiUtente
-DataDiNascita dataDiNascita
-String eMail
-Sesso sesso
-boolean fumatore
-String nome
-Nazione nazionalita
-Occupazione occupazione
-boolean potenzialeCoinquilino
-Facolta facolta
-Citta cittaDiRicerca
+DatiUtente(String nome, String cognome, Sesso sesso, String eMail,
+DataDiNascita getDataDiNascita()
+int setDataDiNascita(int giorno, int mese, int anno)
+String geteMail()
+String seteMail(String eMail)
+String getPassword()
+String setPassword(String password)
+String getNumeroDiTelefono()
+String setNumeroDiTelefono(String numeroDiTelefono)
+Sesso getSesso()
+Sesso setSesso(Sesso sesso)
+boolean isFumatore()
+boolean setFumatore(boolean fumatore)
+boolean isCuoco()
+boolean setCuoco(boolean cuoco)
+boolean isSportivo()
+boolean setSportivo(boolean sportivo)
+String getNome()
+String setNome(String nome)
+String getCognome()
+String setCognome(String cognome)
+Nazione getNazionalita()
+Nazione setNazionalita(Nazione nazionalita)
+Occupazione getOccupazione()
+Occupazione setOccupazione(Occupazione occupazione)
+boolean isPotenzialeCoinquilino()
+boolean setPotenzialeCoinquilino(boolean potenzialeCoinquilino)
+Facolta getFacolta()
+Facolta setFacolta(Facolta facolta)
+Citta getCittaDiRicerca()
+Citta setCittaDiRicerca(Citta cittaDiRicerca)
+String toString()
<<enumeration>>
Occupazione
1..1
occupazione
<<enumeration>>
Facolta
1..1
facolta
<<enumeration>>
Nazione
1..1
nazionalita
Modello
Utente
+Boolean setParametroCuoco(int stelle, Boolean isCuoco)
+int setParametroEta(int stelle, int etaMin, int etaMax)
+ProfiloUtente.Facolta setParametroFacolta(int stelle, ProfiloUtente.Facolta facolta)
+boolean setParametroFumatore(int stelle, boolean isFumatore)
+ProfiloUtente.Nazione setParametroNazionalita(int stelle, ProfiloUtente.Nazione nazione)
+ProfiloUtente.Occupazione setParametroOccupazione(int stelle, ProfiloUtente.Occupazione occupazi
+boolean setParametroSportivo(int stelle, boolean sportivo)
+ArrayList<ParametroRicercaCoinquilino> getParametri()
1..1
1..1
1..1
datiUtente
0..*
Utente
-int idUtente
-DatiUtente datiUtente
+Utente(int idUtente, DatiUtente datiUtente)
+int getIdUtente()
+DatiUtente getDatiUtente()
+DatiUtente setDatiUtente(DatiUtente datiUtent
+String toString()
utente
ProfileManager
-Utente utente
-AnnuncioCasa annuncioCasa
+ProfileManager(Utente utente, AnnuncioCasa annuncioCasa)
+Utente getUtente()
+AnnuncioCasa getAnnuncioCasa()
+AnnuncioCasa setAnnuncioCasa(AnnuncioCasa annuncioCasa)
+Utente setUtente(Utente utente)
+String modificaProfilo(Facolta facolta, Citta citta, Occupazione occupaz
+String creaAnnuncio(ArrayList<ElettroDomestico> elettrodomestici, int
+void cancellaAnnuncio()
+String toString()
User
-int power
-ProfileManager profileManager
-Calendar dataOraAccesso
+User()
+int getPower()
+String toString()
+Calendar getDataOraAccesso()
profileManager
1..1
1..1
<<implements>>
-Guest guest
-User user
-ContenitoreParametr
-RicercaAnnuncio rice
-ContenitoreParametr
-RicercaCoinquilino ric
-BusinessModelUtente
-BusinessModelAnnun
-Calendar dataOraAcc

Ricerca

La ricerca della casa e del coinquilino sfruttano di base lo stesso “algoritmo di affinità”.
Un utente per comparire tra i risultati di ricerca deve essersi candidato come tale.
Si compone di:

- Parametri di ricerca
- Peso dei parametri di ricerca
- Affinità risultante

Rooming

Algoritmo di affinità

- Permette all'utente di esprimere il proprio grado di considerazione per i parametri di ricerca attribuendo un valore da 1 a 5 stelle.
- L'affinità risultante si ottiene come:
  - AFFINITA' = ((STELLE ASSEGNATE)/ STELLE CALCOLATE)*100

Rooming

Parametri di ricerca coinquilino

- E’ stata utilizzata una classe astratta per modellizzare i parametri di ricerca facoltativi (sportivo, età ..) .
- Tutti i parametri vengono successivamente gestiti da una classe *ContenitoreParametriCoinquilino* che gestisce inoltre i due parametri obbligatori, ovvero il sesso del coinquilino e la città oggetto della ricerca.

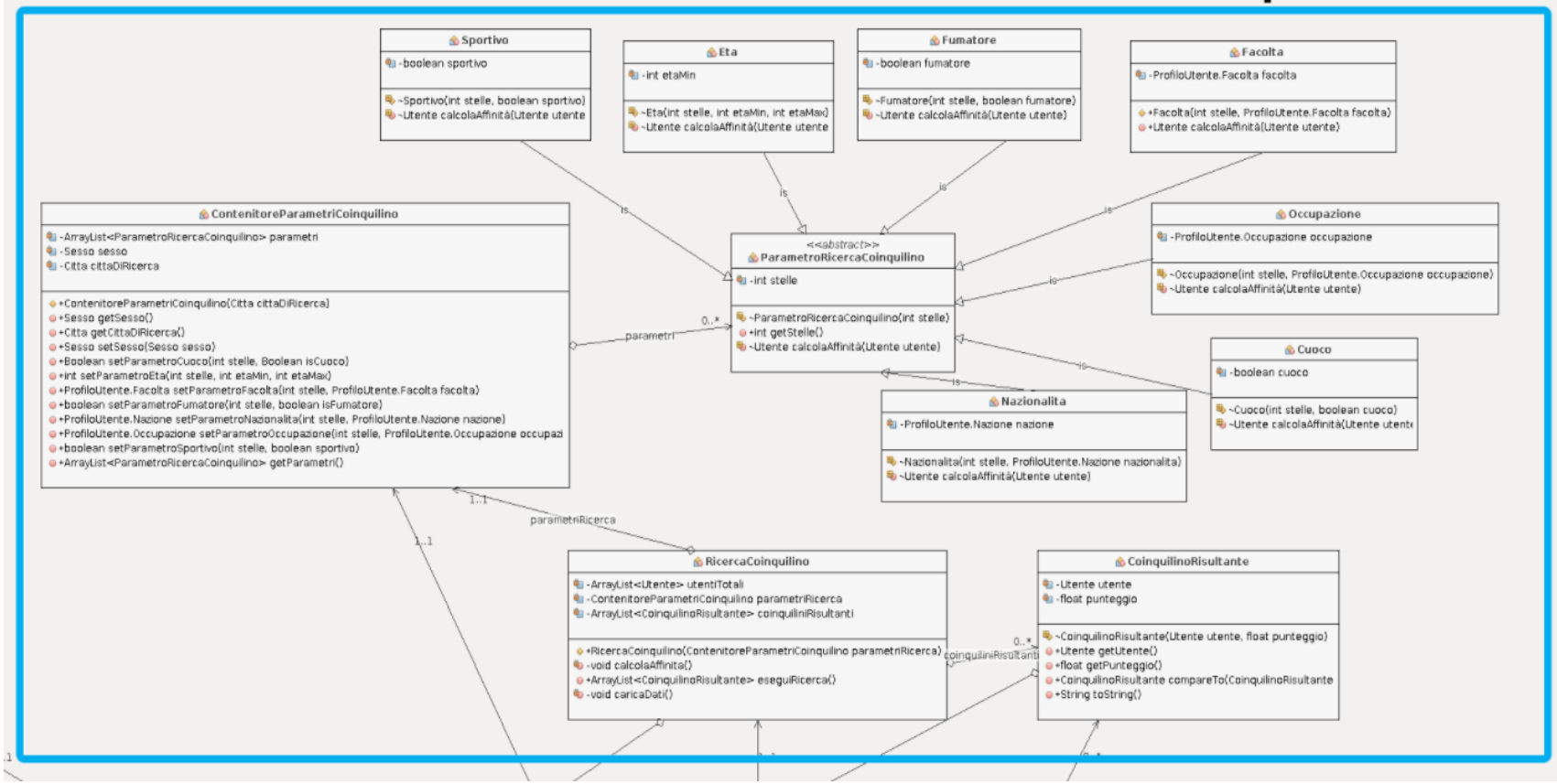
Rooming

Ricerca coinquilino

- Nella classe *RicercaCoinquilino* è definito un attributo di tipo *ContenitoreParametriCoinquilino* sulla base del quale viene successivamente effettuata la ricerca.
- Il metodo *eseguiRicerca()* restituirà i coinquilini risultanti.
- La classe *CoinquilinoRisultante* conterrà un punteggio relativo all'affinità con la ricerca effettuata e i dati dell'utente.

Rooming

Ricerca Coinquilino



Rooming

Parametri ricerca annuncio

- E’ stata utilizzata una classe astratta per modellizzare i parametri di ricerca facoltativi (cucina, tipoCamera, elettrodomestico..) .
- Tutti i parametri vengono successivamente gestiti da una classe *ContenitoreParametriAnnuncio* che gestisce inoltre i due parametri obbligatori, ovvero il costo massimo e la città oggetto della ricerca.

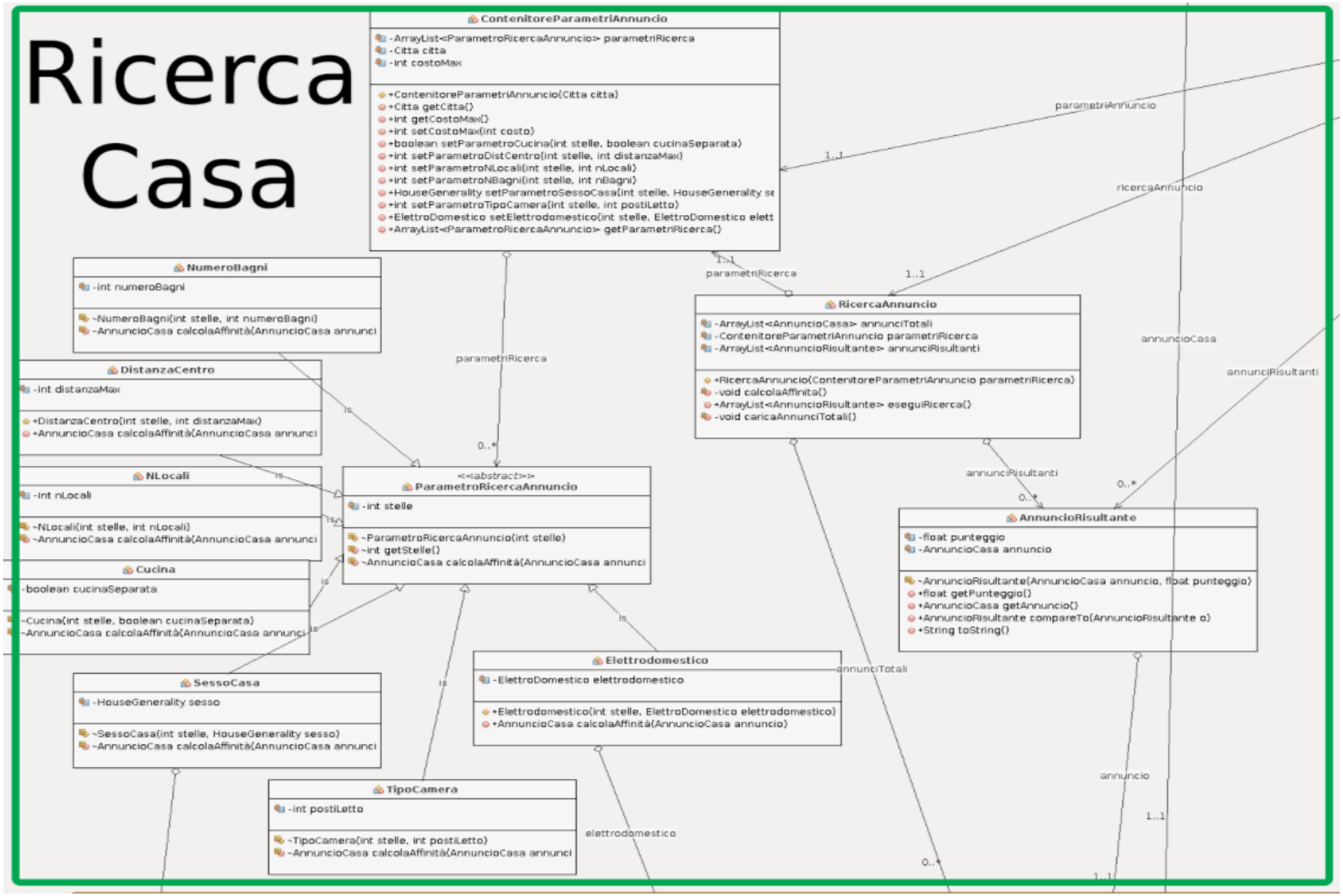
Rooming

Ricerca annuncio

- Nella classe *RicercaAnnuncio* è definito un attributo di tipo *ContenitoreParametriAnnuncio* sulla base del quale viene successivamente effettuata la ricerca.
- Il metodo *eseguiRicerca()* restituirà gli annunci risultanti.
- La classe *AnnuncioRisultante* conterrà un punteggio relativo all'affinità con la ricerca effettuata e i dati dell'annuncio.

Rooming

Ricerca Casa

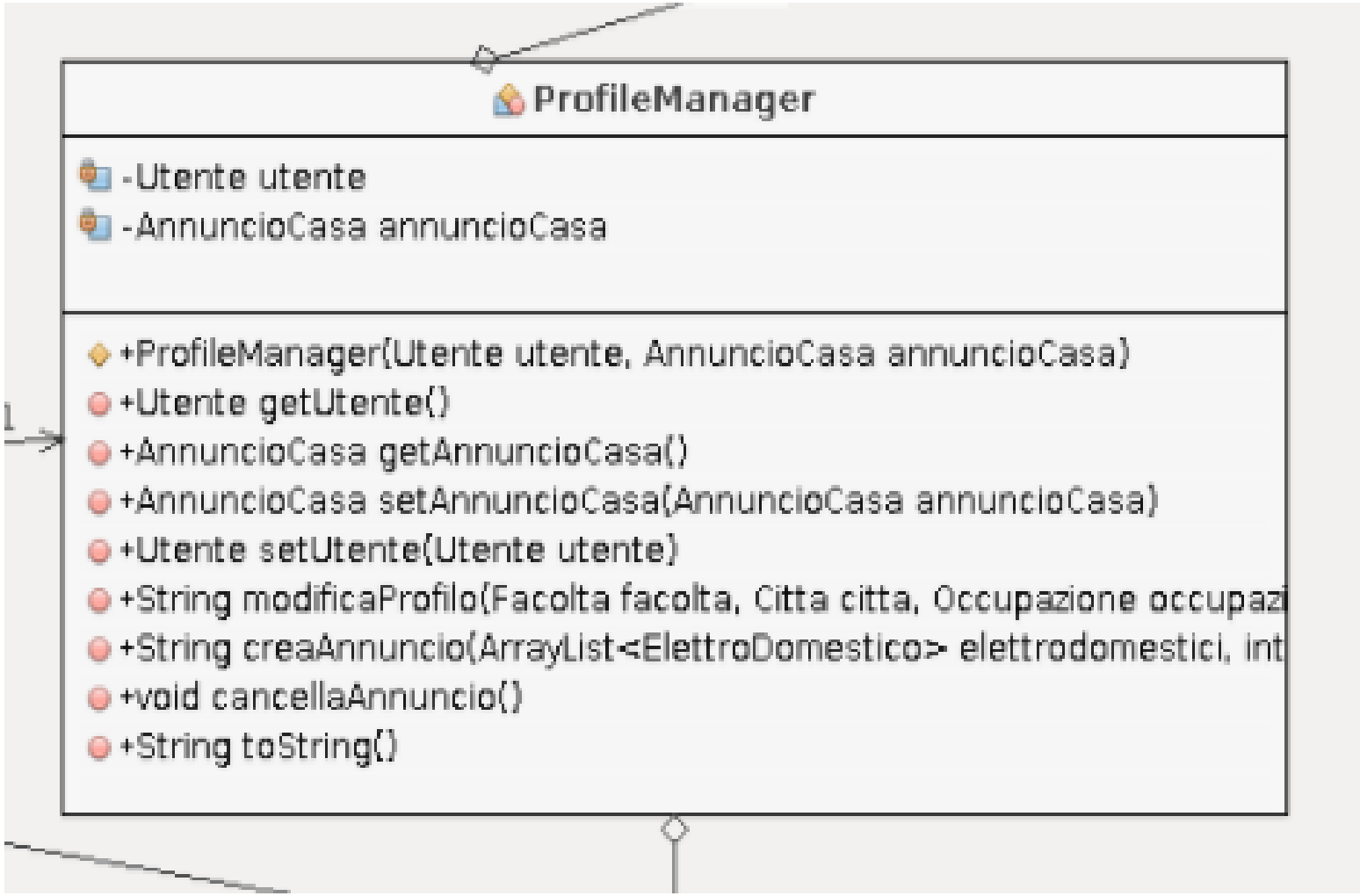


Rooming

Profile Manager

- Utilizzato il pattern *Facade*
- La classe ProfileManager gestisce le operazioni sul profilo dell'utente, come modifica e creazione dell'annuncio o modifica dei dati personali.

Rooming


ProfileManager
-Utente utente
-AnnuncioCasa annuncioCasa
+ProfileManager(Utente utente, AnnuncioCasa annuncioCasa)
+Utente getUtente()
+AnnuncioCasa getAnnuncioCasa()
+AnnuncioCasa setAnnuncioCasa(AnnuncioCasa annuncioCasa)
+Utente setUtente(Utente utente)
+String modificaProfilo(Facolta facolta, Citta citta, Occupazione occupazi
+String creaAnnuncio(ArrayList<ElettroDomestico> elettrodomestici, int
+void cancellaAnnuncio()
+String toString()

Rooming

Classe Sistema

- Nell'implementare la classe *Sistema* è stato utilizzato il pattern *Facade*.
- Si occupa di gestire le operazioni di ricerca (casa e coinquilino), di login e di registrazione.
- E' direttamente collegato alle classi BusinessModel per l'interfacciamento con il database.

Rooming


Sistema
-Guest guest
-User user
-ContenitoreParametriAnnuncio parametriAnnuncio
-RicercaAnnuncio ricercaAnnuncio
-ContenitoreParametriCoinquilino parametriCoinquilino
-RicercaCoinquilino ricercaCoinquilino
-BusinessModelUtente bmUtente
-BusinessModelAnnuncio bmAnnuncio
-Calendar dataOraAccesso
-ArrayList<AnnuncioRisultante> annunciRisultanti
-ArrayList<CoinquilinoRisultante> coinquiliniRisultanti
+Sistema()
+String logIn(String eMail, String password)
+int settaLoggato(int idUtente)
+boolean registrazioneUtente(String nome, String cognome, Sesso sesso, String eMail, String password, i
-void switchToUser()
-void setNullAll()
-void switchToGuest()
+int setCostoMax(int costo)
+Citta iniziaRicercaAnnunci(Citta citta)
+Citta iniziaRicercaCoinquilini(Citta citta)
+boolean setParametroCucina(int stelle, boolean cucinaSeparata)
+int setParametroDistCentro(int stelle, int distanzaMax)
+int setParametroNLocali(int stelle, int nLocali)
+int setParametroNBagni(int stelle, int nBagni)
+HouseGenerality setParametroSessoCasa(int stelle, HouseGenerality sesso)
+int setParametroTipoCamera(int stelle, int postiLetto)
+ElettroDomestico setParametroElettrodomestico(int stelle, ElettroDomestico elettrodomestico)
+Guest getGuest()
+User getUser()
+ContenitoreParametriAnnuncio getParametriAnnuncio()
+RicercaAnnuncio getRicercaAnnuncio()
+ContenitoreParametriCoinquilino getParametriCoinquilino()
+RicercaCoinquilino getRicercaCoinquilino()
+Calendar getDataOraAccesso()
+void ricercaAnnuncio()
+Boolean setParametroCuoco(int stelle, Boolean isCuoco)
+int setParametroEta(int stelle, int etaMin, int etaMax)
+ProfiloUtente.Facolta setParametroFacolta(int stelle, ProfiloUtente.Facolta facolta)
+boolean setParametroFumatore(int stelle, boolean isFumatore)
+ProfiloUtente.Nazione setParametroNazionalita(int stelle, ProfiloUtente.Nazione nazionalita)
+ProfiloUtente.Occupazione setParametroOccupazione(int stelle, ProfiloUtente.Occupazione occupazione)
+boolean setParametroSportivo(int stelle, boolean sportivo)
+Sesso setSesso(Sesso sesso)
+void ricercaCoinquilino()
+ArrayList<AnnuncioRisultante> getAnnunciRisultanti()
+ArrayList<CoinquilinoRisultante> getCoinquiliniRisultanti()
+int verificaPresenzaAnnunci(int idUtenteProprietario)
+int getDatiUtente(int idUtente)
+int getAnnuncioProprietario(int idProprietario)

Rooming

Application Server

- Sito web: sistema distribuito a scambio di messaggi (HTTP)
- È stata utilizzata la libreria Jetty.
- Implementazione di una servlet per ogni pagina web.

  Sono stati utilizzati metodi doGet e doPost per interagire con il client e gestire i dati dinamici.

Rooming

Esempio: Ricerca Casa (1/2)

```
public class RicercaCasaServlet extends HttpServlet  {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)  {

        try {
            Cookie cookie = req.getCookies()[0];
            if(CookieStorage.getInstance().controllaPresenzaCookie(cookie)){
                // utente gia loggato.
                String headerLoggato = HtmlReader.htmlReader("headerLoggato.html");
                String ricercaHtml = HtmlReader.htmlReader("RicercaCasa.html");
                resp.setStatus(200);
                resp.getWriter().println(headerLoggato + ricercaHtml);
            } else {
                cookie.setMaxAge(0); // il cookie non è più valido, dunque lo elimino
                resp.addCookie(cookie);
                settaNonLoggato(resp);
            }
        } catch (NullPointerException ex) {
            settaNonLoggato(resp);
        }
    }
```

Rooming

Esempio: Ricerca Casa (2/2)

```
 @Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) {

    try {
        String risultati = ListaAnnunciCreator.creaListaAnnunci(effettuaRicerca(req),
                    req,resp);
        resp.getWriter().println(risultati);
        resp.setStatus(200);

} catch (NessunAnnuncioException ex) {
        // pagina con avviso nessun annuncio trovato.
        String risultati = HtmlReader.htmlReader("norisultati.html");
        resp.getWriter().println(risultati);
    }
}
```

Rooming

Gestione del Login: Cookie

- Utilizzo dei cookie per il riconoscimento dei client loggati
- Cookie" memorizza idUtente ed una stringa di controllo (16)
- La memorizzazione e la verifica della validità avviene con un *Singleton. (CookieStorage)*

Rooming

Interfaccia Web

- Per la realizzazione dell'interfaccia web, sono stati utilizzati HTML, CSS e Javascript.
- Per il CSS sono state utilizzate varie fonti, sia proprie che reperite online.
- La barra di navigazione viene caricata dinamicamente come anche il footer.
- Per alcune situazioni, sono state create pagine di reindirizzamento automatico temporizzato (Javascript).

Rooming

DEMO