**Object Design**

1. **Introduzione**
   1. **Object design trade-offs**

[Costi vs Mantenimento]

Aggiungeremo della documentazione specifica per porzioni di codice avendo un aumento dei costi, ottenendo però in cambio una maggiore comprensibilità e maggiore facilità di manutenzione.

[Interfaccia vs Easy-use]

L'interfaccia verrà progettata in modo tale da essere intuitiva garantendo facilità d'uso anche ad utenti poco esperti. Il sistema userà font di grandezza adeguata per non appesantire la lettura e la navigazione sarà agevole. L'interfaccia si adatterà a qualsiasi dispositivo e manterrà le caratteristiche sopra descritte.

* 1. **Linee guida per l’interfaccia di documentazione**
* Le query di inserimento andranno su due righe: la prima conterrà la insert ed i parametri formali, la seconda i valori.
* Dopo un ciclo while o for seguirà la parentesi graffa, il blocco delle istruzioni verrà distanziato di un tab alla riga successiva.

Esempio:

while(true) {

Istruzioni

}

* Per la scrittura dei metodi, e per i tag html ci rifacciamo all’indentazione utilizzata per i cicli.

Esempio:

public metodo() {

istruzioni

}

Esempio:

<html>

<body>

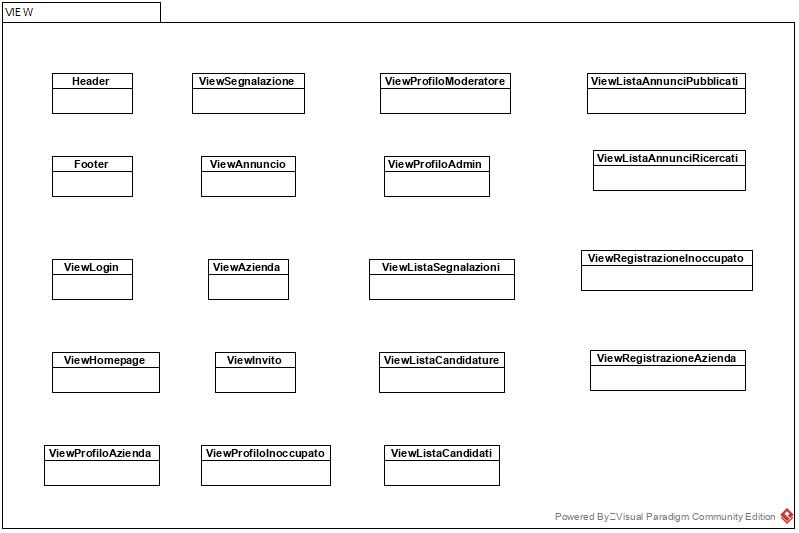
</body>

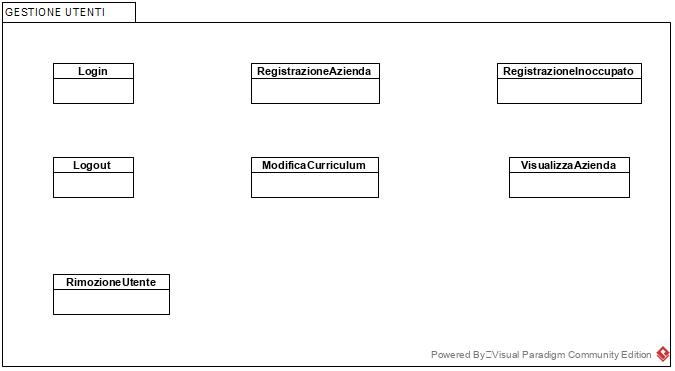
</html>

* Per le condizioni if con una sola istruzione utilizzeremo comunque le parentesi graffe.
* I nomi delle classi sono scritti al singolare e sono significativi rispetto al ruolo della classe.
* Anche i nomi delle variabili rispetteranno i parametri definiti per i nomi delle classi.
  1. **Componenti off-the-shelf**

Un componente che andremo ad utilizzare sarà Bootstrap, un Toolkit open source per lo sviluppo con HTML, CSS e JS. Risulta molto efficiente usare questo framework data la vasta gamma di librerie che mette a disposizione per la creazione dell’interfaccia utente.

* 1. **Definizioni, acronimi e abbreviazioni**
  + RAD: Requirements Analysis Document.
  + SDD: System Design Document.
  + ODD: Object Design Document.
  + DB: Database.
  + Off-the-shelf: componenti hardware/software disponibili sul mercato per l’acquisto da parte di aziende di sviluppo interessate ad utilizzarli.
  + Moderatore annunci: utente che si occupa della gestione degli annunci, della loro rimozione e della segnalazione degli utenti.
  + Amministratore: utente che si occupa della rimozione degli utenti e dell’analisi delle segnalazioni.
  + HTML: linguaggio di mark-up per pagine web.
  + Bootstrap: framework che fornisce un insieme di elementi grafici, stilistici, di impaginazione e di Javascript pronti all’uso.

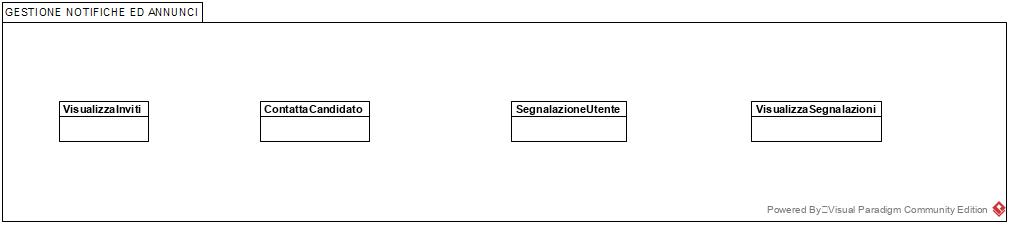
1. **Packages**
   1. **Package Generale**
   2. **Package View**
   3. **Package Control** 
      1. **Package Gestione Utenti**

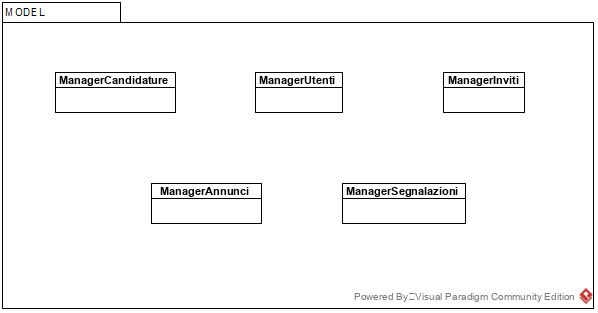


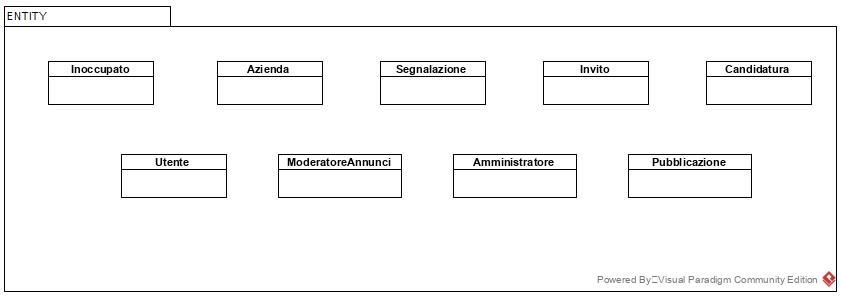
* + 1. **Package Gestione Bacheca**



* + 1. **Package Gestione Notifiche, Inviti E Segnalazioni**

****

* 1. **Package Model**
  2. **Package Entity**



1. **Interfaccia delle classi**
   1. **CandidateManager**

|  |
| --- |
| **Nome Classe**: CandidateManager |
| **Descrizione Classe**: Classe composta da metodi statici che permettono di creare una candidatura e di visualizzare la lista delle candidature effettuate. |
| **Metodi:** + visualizzaCandidature (idInoccupato)  + candidate(IdInoccpato,IdAnnuncio)  + isAlreadyCandidate(idInoccupato,idAnnuncio) |

|  |  |
| --- | --- |
| **Nome Metodo:** +visualizzaCandidature (Inoccupato inocc): List<Candidatura> | |
| **Descrizione metodo:** Dato un id di un inoccupato restituisce, se presente, una lista di candidature effettuate | |
| **PreCondizione** | **Context:** CandidateManager::visualizzaCandidature (int idInoccupato)  **pre**: (inoc != null) && (inoc.getId()>=1) |
| **PostCondizione** | **Context:** Context: CandidateManager::visualizzaCandidature (idInoccupato)  **post:**  // |

|  |  |
| --- | --- |
| **Nome Metodo:** +isAlreadyCandidate (Inoccupato inocc,Annuncio ann): Boolean | |
| **Descrizione metodo:** Dato un id di un inoccupato e un id di un’annuncio, controlla se l’utente si è già candidato per l’annuncio specificato. | |
| **PreCondizione** | **Context:** CandidateManager::isAlreadyCandidate (Inoccupato inocc,Annunncio ann)  **pre**: (inocc != null && inocc.getId()) >=1) && (ann != null && ann.getId()>=1) |
| **PostCondizione** | **Context:** Context: CandidateManager::isAlreadyCandidate(idInoccupato)  **post:**  Se non ha candidature allora: visualizzaCandidature(idInoccupato).isEsmpty() => false  Se ha candidature allora:  ! (visualizzaCandidature(idInoccupato).isEmpty()) && exist (visualizzaCandidature(idInoccupato).getIdAnnuncio() == idAnnuncio) =>true |

|  |  |
| --- | --- |
| **Nome Metodo:** +candidate (Inoccupato inocc,Annuncio ann): Boolean | |
| **Descrizione metodo:** Dato un id di un inoccupato e un id di un’annuncio, memorizza nel database la candidatura, solo se questo non si è già candidato per l’annuncio. | |
| **PreCondizione** | **Context:** CandidateManager::candidate(Inoccupato inocc,Annuncio ann)  **pre**: (inocc != null && inocc.getId()) >=1) && (ann != null && ann.getId()>=1) && !(isAlreadyCandidate) |
| **PostCondizione** | **Context:** Context: CandidateManager::candidate(Inoccupato inocc,Annuncio ann)  **post:**  // |

* 1. **UserManager**

|  |
| --- |
| **Nome Classe**: UserManager |
| **Descrizione Classe**: Classe composta da metodi statici che permettono di: creare nuovi utenti, modificare informazioni su gli utenti e recupreare le loro informazioni |
| **Metodi:** + isPresent (Utente u)  + logIn(String username,string password)  + deleteUser(Utente u)  + modificaCurriculum (idUser)  + getUserById (idUser)  + alreadyBanned (idUser )  + registerUser(Utente u) |

|  |  |
| --- | --- |
| **Nome Metodo:** +isPresent(Utente u): Boolean | |
| **Descrizione metodo:** Dato un utente si controlla che l’utente è presente o meno nel database. | |
| **PreCondizione** | **Context:** UserManager::isPresent(Utente u)  **pre**: (u != null) |
| **PostCondizione** | **Context:** UserManager:: isPresent(Utente u)  **post:**  Se l’utente è presente nel db => true  Se l’utente non è presente nel db => false |

|  |  |
| --- | --- |
| **Nome Metodo:** +logIn(String username,String password): Utente u | |
| **Descrizione metodo:** Dato un utente si controlla che l’utente è presente o meno nel database, se è presente viene effettuato il login. | |
| **PreCondizione** | **Context:** UserManager::logIn(String username,String password)  **pre**: (username != “” && username != null) && (password != “” && password != null) |
| **PostCondizione** | **Context:** UserManager:: logIn(String username,String password)  **post:**  restituisci un utente u, in corrispondenza dei dati passati. |

|  |  |
| --- | --- |
| **Nome Metodo:** +alreadyBanned(String idUser): Boolean | |
| **Descrizione metodo:** Dato un id di un utente si controlla se l’utente è stato bannato. | |
| **PreCondizione** | **Context:** UserManager::alreadyBanned(String idUser)  **pre**: (u.getId()>=1) && isPresent(Utente u) |
| **PostCondizione** | **Context:** UserManager:: alreadyBanned(String idUser)  **post:**  getUserById(int idUser).getBanned() == true => TRUE  getUserById(int idUser).getBanned()==false => FALSE |

|  |  |
| --- | --- |
| **Nome Metodo:** +deleteUser(Utente u): Boolean | |
| **Descrizione metodo:** Dato un id di un utente questo viene rimosso, se presente dal database. | |
| **PreCondizione** | **Context:** UserManager::deleteUser(Utente u)  **pre**: (u.getId()>=1) && !(alreadyBanned(u.getId())) |
| **PostCondizione** | **Context:** UserManager:: deleteUser(Utente u)  **post:**  Restituisce true se è stato eliminarlo  Restituisce false se non è stato eliminato |

|  |  |
| --- | --- |
| **Nome Metodo:** +modificaCurriculum(int idUtente, String newCV): Boolean | |
| **Descrizione metodo:** Dato un id di un utente questo viene rimosso, se presente dal database. | |
| **PreCondizione** | **Context:** UserManager::modificaCurriculum(int idUtente)  **pre**: idUtente>=1 && (newCv != “” && newCv != null) |
| **PostCondizione** | **Context:** UserManager:: modificaCurriculum(int idUtente, String newCv)  **post:**  Restituisce true se è stato modificato  Restituisce false se non è stato modificato |

|  |  |
| --- | --- |
| **Nome Metodo:** +registerUser(Utente u): Boolean | |
| **Descrizione metodo:** Dato un utente se questo non è già presente nel database, allora viene memoriazzato | |
| **PreCondizione** | **Context:** UserManager::registerUser(Utente u)  **pre**: !(isPresent(u)) |
| **PostCondizione** | **Context:** UserManager:: modificaCurriculum(int idUtente, String newCv)  **post:**  isPresent(u) == true |

* 1. **InvitesManager**

|  |
| --- |
| **Nome Classe:** InvitesManager |
| **Descrizione Classe:** Classe che si occupa di gestire l’elenco di inviti di un inoccupato e la memorizzazione degli inviti creati da un’azienda per un inoccupato. |
| **Metodi:** +visualizzaInviti(Inoccupato inoccupato): List<Invito>  +contattaCandidato(Invito invito): boolean |

|  |  |
| --- | --- |
| **Nome Metodo:** +visualizzaInviti(Inoccupato inoccupato): List<Invito> | |
| **Descrizione metodo:** Dato un inoccupato, restituisce tutti gli inviti da lui ricevuti. | |
| **PreCondizione** | **Context:** InvitesManager :: visualizzaInviti(inoccupato)  **pre:** (inoccupato != null) && (inoccupato.getId() >= 1) |
| **PostCondizione** | **Context:** InvitesManager :: visualizzaInviti(inoccupato)  **post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** +contattaCandidato(Invito i): boolean | |
| **Descrizione metodo:** Dato un invito, lo rende persistente. | |
| **PreCondizione** | **Context:** InvitesManager :: contattaCandidato(invito)  **pre:** (invito != null) && (i.getInoccupato().getId() >= 1) |
| **PostCondizione** | **Context:** InvitesManager :: contattaCandidato(invito)  **post:** Se la memorizzazione è andata a buon fine => true  Altrimenti => false |

* 1. **SegnalationManager**

|  |
| --- |
| **Nome Classe:** SegnalationManager |
| **Descrizione Classe:** Classe che si occupa di gestire l’elenco di segnalazioni ricevute dall’amministratore e di gestire la memorizzazione delle segnalazioni create da un moderatore. |
| **Metodi:** +visualizzaElencoSengalazioni(): List<Segnalazione>  +segnalaUtente(Segnalazione segnalazione): boolean |

|  |  |
| --- | --- |
| **Nome Metodo:** +visualizzaElencoSegnalazioni(): List<Invito> | |
| **Descrizione metodo:** Restituisce tutte le segnalazioni effettuate. | |
| **PreCondizione** | **Context:** InvitesManager :: visualizzaElencoSegnalazioni()  **pre:** // |
| **PostCondizione** | **Context:** InvitesManager :: visualizzaElencoSegnalazioni()  **post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** +segnalaUtente(Segnalazione s): boolean | |
| **Descrizione metodo:** Rende persistente una segnalazione. | |
| **PreCondizione** | **Context:** InvitesManager :: segnalaUtente(Segnalazione s)  **pre:** (segnalazione != null) && (s.getAzienda().getId >= 1) |
| **PostCondizione** | **Context:** InvitesManager :: SegnalaUtente(invito)  **post:** Se la memorizzazione è andata a buon fine => true  Altrimenti => false |

* 1. **AdManager**

|  |
| --- |
| **Nome Classe:** AdManager |
| **Descrizione Classe:** |
| **Metodi:** +searchAd(String ricerca): List<Annuncio>  +searchAdAdvanced(String ricerca): List<Annuncio>  +visualizzaElencoAnnunci(Azienda azienda): List<Annuncio>  +pubblicaAnnuncio(Annuncio annuncio): boolean  +filterSearch(Date data): List<Annuncio>  +removeAd(int idAnnuncio): boolean  +readAd(int idAnnuncio): Annuncio |

|  |  |
| --- | --- |
| **Nome Metodo:** +searchAd(String ricerca): List<Annuncio> | |
| **Descrizione metodo:** Restituisce un elenco di annunci in base al tag specificato in input. | |
| **PreCondizione** | **Context:** AdManager :: searchAd(ricerca): List<Annuncio>  **pre:** (ricerca != null) && (formato Ricerca ^[A-Za-z]{1,}$) |
| **PostCondizione** | **Context:** AdManager :: searchAd(ricerca): List<Annuncio>  **post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** +searchAdAdvanced(String ricerca): List<Annuncio> | |
| **Descrizione metodo:** Restituisce un elenco di annunci in base alla città specificata in input. | |
| **PreCondizione** | **Context:** AdManager :: searchAdAdvanced(ricerca): List<Annuncio>  **pre:** (ricerca != null) && (formato Ricerca ^[A-Za-z]{1,}$) |
| **PostCondizione** | **Context:** AdManager :: searchAdAdvanced(ricerca): List<Annuncio>  **post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** +visualizzaElencoAnnunci(Azienda azienda): List<Annuncio> | |
| **Descrizione metodo:** Restituisce l’elenco di annunci pubblicati da un’azienda. | |
| **PreCondizione** | **Context:** AdManager :: visualizzaElencoAnnunci(azienda): List<Annuncio>  **pre:** (azienda != null) && (azienda.getId() >= 1) |
| **PostCondizione** | **Context:** AdManager :: visualizzaElencoAnnunci(azienda): List<Annuncio>  **post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** +pubblicaAnnuncio(Annuncio annuncio): boolean | |
| **Descrizione metodo:** Dato un annuncio, lo rende persistente. | |
| **PreCondizione** | **Context:** AdManager :: pubblicaAnnuncio(annuncio): boolean  **pre:** (annuncio != null) && (annuncio.getAzienda.getId() >= 1) |
| **PostCondizione** | **Context:** AdManager :: pubblicaAnnuncio(annuncio): boolean  **post:** readAd (annuncio.getId()) != null |

|  |  |
| --- | --- |
| **Nome Metodo:** +filterSearch(Date data): List<Annuncio> | |
| **Descrizione metodo:** Restituisce un elenco di annunci pubblicati nella data inserita in input. | |
| **PreCondizione** | **Context:** AdManager :: filterSearch(data): List<Annuncio>  **pre:** data <= Calendar.getInstance() |
| **PostCondizione** | **Context:** AdManager :: filterSearch(data): List<Annuncio>  **post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** +removeAd(int idAnnuncio): boolean | |
| **Descrizione metodo:** Rimuove un annuncio dal database. | |
| **PreCondizione** | **Context:** AdManager :: removeAd(int idAnnuncio): boolean  **pre:** idAnnuncio >= 1 |
| **PostCondizione** | **Context:** AdManager :: removeAd(int idAnnuncio): boolean  **post:** Se la memorizzazione è andata a buon fine => true && readAd(idAnnuncio) == null  Altrimenti => false && readAd(idAnnuncio) != null |

* 1. **RegistrazioneAziendaServlet**

|  |
| --- |
| **Nome Classe:** RegistrazioneAziendaServlet |
| **Descrizione Classe:** Servlet che si occupa di gestire la registrazione dell’azienda. Se tutti i dati sono stati inseriti correttamente e non ci sono omonimie, l’utente viene registrato e risulta loggato. |
| **Metodi:** +doGet (HttpServletRequest request, HttpServletResponse response): void  +doPost (HttpServletRequest request, HttpServletResponse response): void |

|  |  |
| --- | --- |
| **Nome Metodo:** +doPost (HttpServletRequest request, HttpServletResponse response): void | |
| **Descrizione metodo:** Questo metodo si occupa di prendere i dati di input inseriti dall’azienda e memorizzarli al fine di renderla registrata al sito. | |
| **PreCondizione** | **Context:** RegistrazioneAziendaServlet :: doPost(request, response)  **pre:** (request.getParameter(“Nome”) != null) && (formato Nome ^[A-Za-z ]{2,50}$) &&  (request.getParameter(“Cognome”) != null) && (formato Cognome ^[A-Za-z ]{2,50}$) &&  (request.getParameter(“Username”) != null) &&(formato Username ^[A-Za-z0-9]{5,20}$) &&  (request.getParameter(“Password”) != null) && (formato Password ^[A-Za-z0-9-.\_]{8,16}$) &&  (request.getParameter(“Conferma password”) != null) && (request.getParameter(“Conferma password”).equals(request.getParameter(“Password”))) &&  (request.getParameter(“e-mail”) != null) && (formato e-mail ^[A-Za-z0-9\_.]+@[a-zA-Z.]{2,}\.[a-zA-Z]{2,3}$) &&  (request.getParameter(“Città”) != null) && (formato Città ^[A-Za-z' ]{2,20}$) && (request.getParameter(“Indirizzo”) != null) && (formato Indirizzo ^[A-Za-z ]{3,6}[A-Za-z ]{2,35}[,]{1}[0-9 ]{2,5}$) |
| **PostCondizione** | **Context:** RegistrazioneAziendaServlet :: doPost(request, response)  **post:** request.getSession().getAttribute(“Azienda”) != null |