University of Glasgow | Department of Computing Science

D6: SYSTEM DESIGN DOCUMENT

Deliverable ID	D6
Deliverable Title	System Design Document
Project	PSD3 Group Exercise 2
Team	L
Authors	Dan Tomosoiu Peeranat Fupongsiripan Hector Grebbell Michael Kilian Tony Lau
Deliverable Date	30 Jan 2013
File Name	SystemDesign.tex
Version	1.0

1 Introduction

1.1 Identification

This document outlines the proposed design for the Internship Management System.

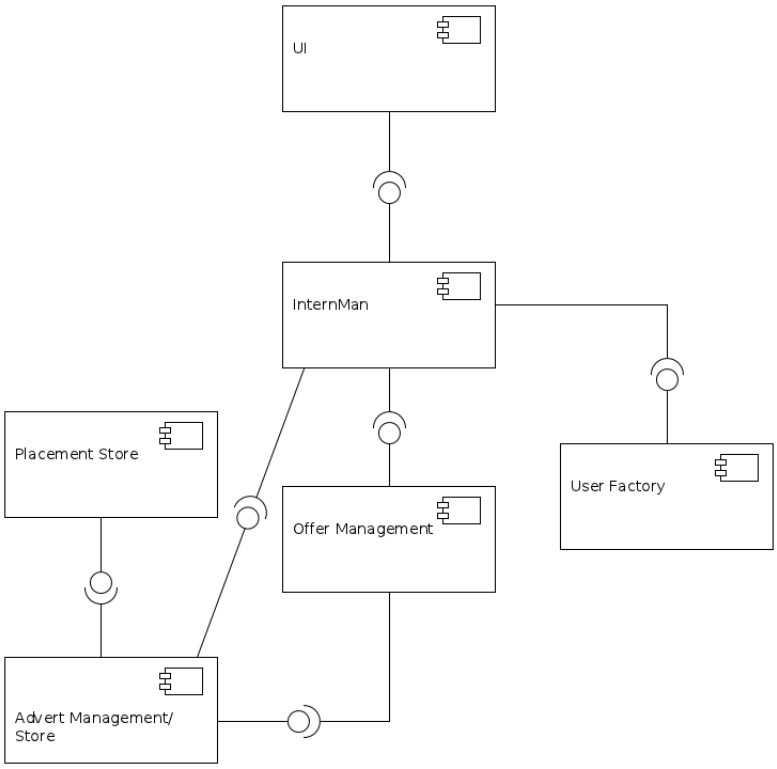
1.2 Related Documentation

- Requirements Specification Document

1.3 Document Status and Schedule

30/01/2013 - First draft

2 System Overview



The proposed system consists of the following components:

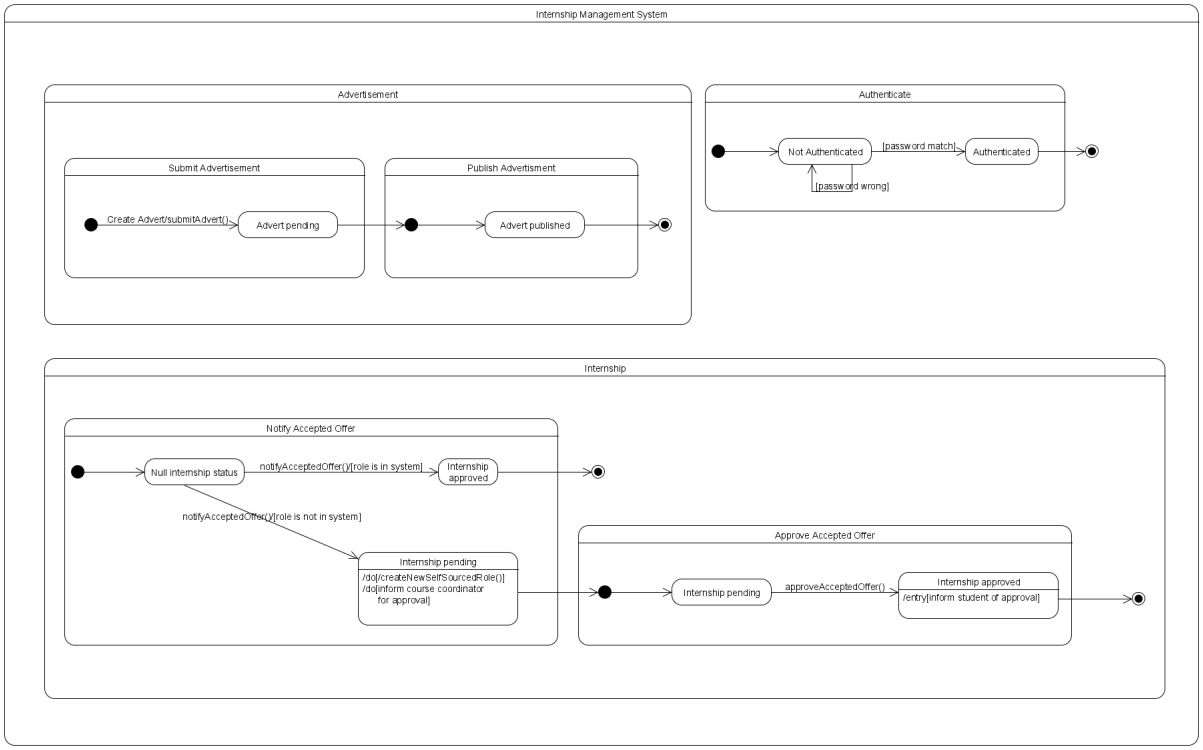
- InternMan: This component contains an implementation of the InternMan facade provided and can be seen as the 'main' component for the system.
- User Factory: Provides methods for creating new users of different types, storing created users and fetching users from this store. This component also provides services for authenticating a user's login credentials using a stored username/password combination.
- Offer Management: Handles offers made to students.

- Advert Management/Store: This component provides the necessary services to create and view adverts, associated these with an employer account, etc. It simultaneously manages the storage and retrieval of adverts from an appropriate source.
- Placement Store: Provides services for manipulating internships and associating these with an approriate role.

3 State Machine Diagram

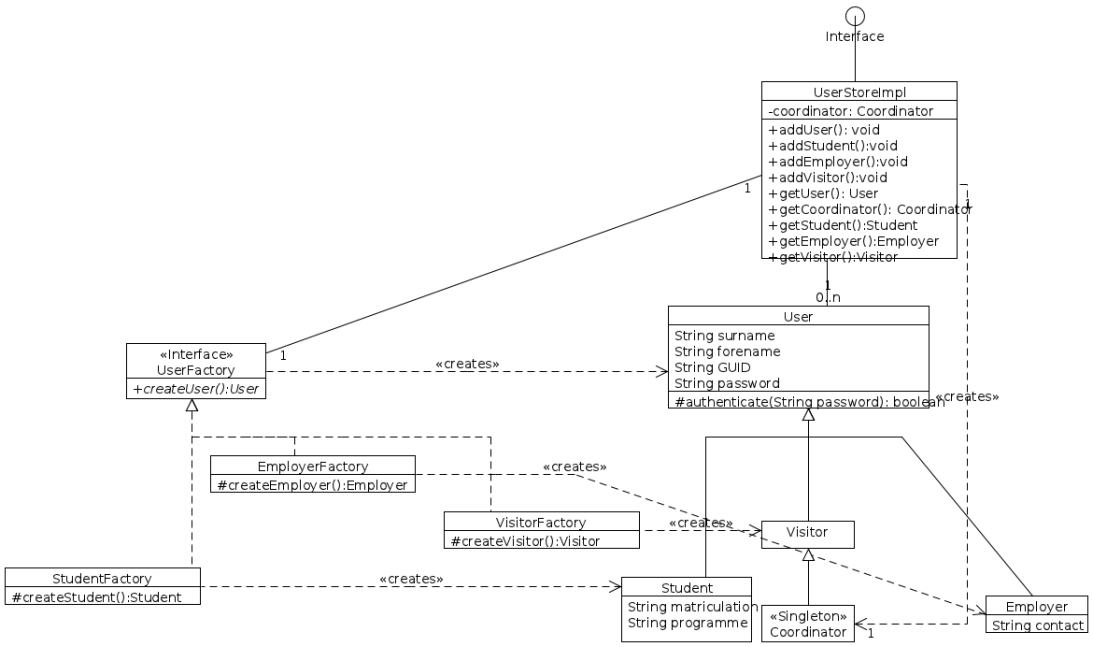
3.1 Description and Rationale

The state machine diagram shows the state of objects within the Internship Management System at any given moment in time.



4 Component Breakdown

4.1 User Factory



4.1.1 Description and Rationale

There is a need to create many different categories of user in the system who share attributes and methods. For example all users must be able to log in using a method to authenticate an input password. In this situation it is appropriate to employ the factory design pattern for the creation of generic users, visitors, students and employers. The coordinator is not instantiated by a factory, but instead is implemented according to the singleton design pattern.
UserStoreImpl groups the methods of these factories and entities together to provide methods for creating and fetching each class of user. For the moment the exact means by which these users will be stored has not been decided, but this should also be handled by UserStoreImpl. The method set for UserStoreImpl is equivalent to the API for the UserFactory component.

4.1.2 API

UserStoreImpl implements UserStore
Methods
public void addUser(String surname, String forename, String GUID, String password)
Adds a new user to the store.
Parameters
String surname
String forename
String GUID
String password

public void addVisitor(String surname,String forename,String GUID, String password)
Adds a new visitor to the store
Parameters
String surname
String forename
String GUID
String password

public void addVisitor(String surname,String forename,String GUID, String password,String matriculation,String programme)
Adds a new student to the store
Parameters
String surname
String forename
String GUID
String password
String matricaulation
String programme

public void addVisitor(String surname,String forename,String GUID, String password,String contact)
Adds a new employer to the store
Parameters
String surname
String forename
String GUID
String password
String contact

public User getUser(String GUID, String password)
Returns a user specified by the GUID, if authentication is successful.
Return the user specified by GUID, if the password is a match
Parameters
String GUID
String password

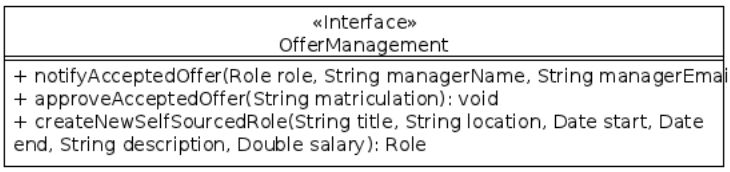
public Visitor getVisitor(String GUID, String password)
Returns a visitor specified by the GUID, if authentication is successful.
Return the visitor specified by GUID, if the password is a match
Parameters
String GUID
String password

public Student getVisitor(String GUID, String password)
Returns a Student specified by the GUID, if authentication is successful.
Return the student specified by GUID, if the password is a match

Parameters
String GUID
String password

public Visitor getVisitor(String name, String password)
Returns a visitor specified by the name if authentication is successful.
Return the user specified by name, if the password is a match
Parameters
String name
String password

4.2 Offer Management



4.2.1 Description and Rationale

This component is needed to allow students to notify the course coordinator that they have accepted an internship. If the internship was advertised in the internship management system, then it is automatically approved. Otherwise, the student can create a new self-sourced role which will send an email to the course coordinator asking for approval. Approval is done through this component.

4.2.2 API

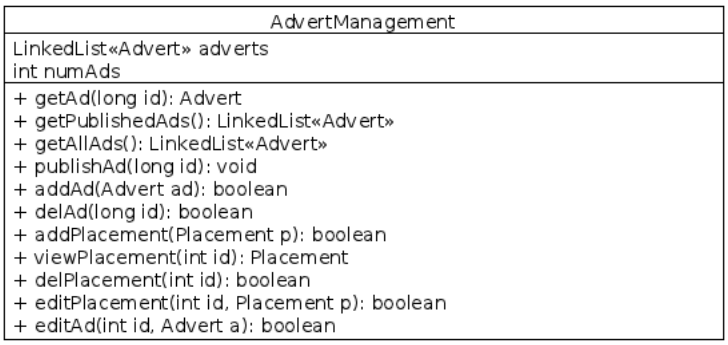
OfferMangementImpl implements OfferManagement
Methods
public void notifyAcceptedOffer(Role role, String managerName, String managerEmail)
Sends email to coordinator informing that a student has accepted an internship
Parameters
Role role
String managerName
String managerEmail

public void approveAcceptedOffer(String matriculation)
Sends email to coordinator informing that a student has accepted an internship
Parameters
String matriculation

public Role createNewSelfSourcedRole(String title, String location, Data start, Date end, String description, Double salary)

Sends email to coordinator informing that a student has accepted an internship
Return
Returns the newly created self-sourced role
Parameters
String title
String location
Data start
Date end
String description
Double salary

4.3 Advert Management/Store



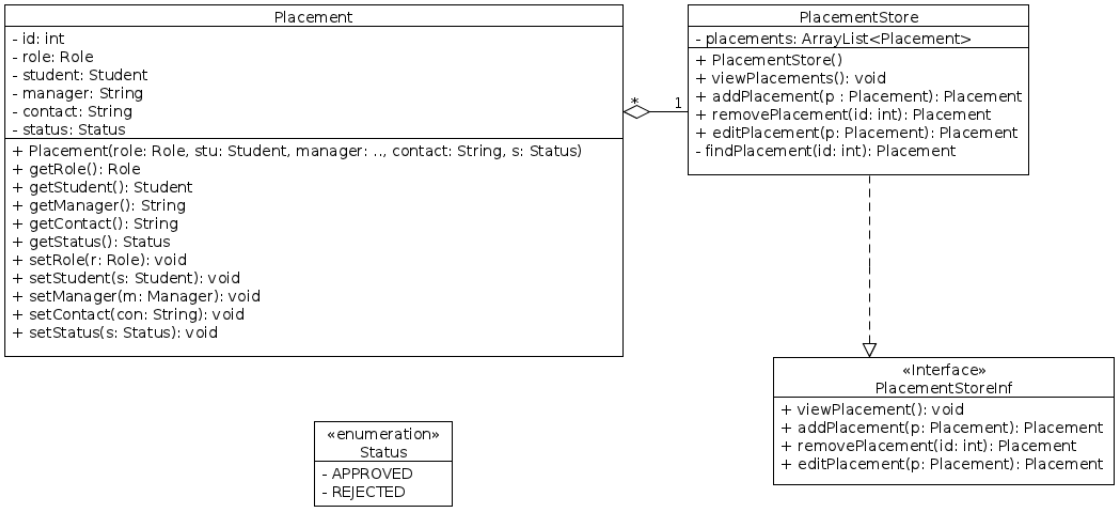
4.3.1 Description and Rationale

Advert Management/Store is needed to handle creation, selection and publishing of all advertisements within the system. It interacts with other components which need information on adverts through the selectAdvertisement method which provides a handle on an advert with a supplied index.

4.3.2 API

TODO

4.4 Placement Store



4.4.1 Description and Rationale

Placement Store contains all information on placements in the system at any given time and is needed to allow other componenets to keep track of which placements are in the system and their status.

4.4.2 API

Interface PlacementStoreInf

Methods
public void viewPlacements()
Prints out all existing placements.

public Placement addPlacement(Placement p)
Add new placement.
Return: Placement p added if successful, otherwise null.
Parameters
Placement p

public Placement removePlacement(int id)
Return: Placement with identification number id removed if successful , otherwise null.
Parameters
int id

public Placement editPlacement(Placement p)
Edits existing placement.
Return: Placement edited if successful, null otherwise.
Parameters
Placement p

Class Placement
Contains details about the internship including the student taking and role involved.

public Placement(Role role, Student student, String manager , String contact, Status status)
Parameters
Role role
Student student
String manager
String contact
Status status

public Role getRole()
Return: Role of placement if exists, null otherwise.

public Student getStudent()
Return: Student secured the placement , null otherwise.

public String getManager()
Return: Manager if exists, null otherwise.

public String getContact()
Return: Contact details, if exists, null otherwise.

public Status getStatus()
Return: returns the status of a particular placement (value can be either APPROVED OR REJECTED)

setRole(Role r)
Return:
Parameters
Role r

setStudent(Student s)
Return:
Parameters
Student s

setManager(String m)
Return:
Parameters
String m
setContact(String c)
Return: Parameters String c

setStatus(Status s)
Return:

Parameters
Status s

A Glossary

- UML - Unified Modelling Language
- API - Application Programming Interface
- PSD/PSD3 - Professional Software Development (3)

Professional Software Development 3

Acceptance Test Plan

Team L

Peeranat Fupongsiripan, Dan Tomosoiu,
Michael Kilian, Tony Lau,
Hector Grebbell

This Document Describes the Acceptance Test Plan for each component within our design of an Internship Management System for the Department of Computing Science at the University of Glasgow. The purpose of this acceptance test is to make sure the system developed meets its system requirements and is suitable for all accepted use cases.

Test Cases Prefixed A can be performed with an automatic test harness. Any Prefixed M must be completed manually.

Authenticator

Test Case	A001
Use Case	The Authenicator is passed a valid username and password for a user registered with the MyCampus System
Input Specification	Multiple known MyCampus Username-Password sets are fed into the authenticator.
Output Specification	For each input the authenticator should return the correct User item associated with the username given.

Test Case	A002
Use Case	The Authenicator is passed a valid username and password for a user registered within the Internship Management System
Input Specification	Multiple known Username-Password sets from the system database are fed into the authenticator.
Output Specification	For each input the authenticator should return the correct User item associated with the username passed in.

Test Case	A003
Use Case	The Authenicator is passed a valid username with incorrect password for a user registered with the MyCampus System
Input Specification	Multiple known MyCampus Usernames are fed into the authenticator with incorrect passwords.
Output Specification	For each input the authenticator should return null.

Test Case	A004
Use Case	The Authenicator is passed a valid username with incorrect password for a user registered within the Internship Management System
Input Specification	Multiple known Usernames from the system database are fed into the authenticator with incorrect passwords.
Output Specification	For each input the authenticator should return null.

Test Case	A005
Use Case	The Authenicator is passed an invalid username with a correct password for a user (with a different username) registered with the MyCampus System
Input Specification	Multiple Usernames known not to exist within the MyCampus System are fed into the authenticator with correct passwords for other users.
Output Specification	For each input the authenticator should return null.

Test Case	A006
Use Case	The Authenicator is passed a valid username with incorrect password for a user registered within the Internship Management System
Input Specification	Multiple Usernames known not to exist within the system database are fed into the authenticator with correct passwords for other users.
Output Specification	For each input the authenticator should return null.

Test Case	A006
Use Case	The Authenicator is passed an invalid username with a password not within the Intern Management or MyCampus systems.
Input Specification	Multiple Username-Password Sets known not to exist within either system are fed into the authenticator.
Output Specification	For each input the authenticator should return null.

UserStore

Test Case	A007
Use Case	A logged in Coordinator wishes to create a new General User Account
Input Specification	addUser() is called with the correct arguments for a specific new user. getUser() (also from this component) and authenticate() (From the authenticator) should then be called with the correct details for the new user.
Output Specification	Both getUser() and authenticate() should return the correct user item (Only valid once tests A002 and A0011).

Test Case	A008
Use Case	A logged in Coordinator wishes to create a new Student User Account
Input Specification	addStudent() is called with the correct arguments for a specific new user. getUser(), getStudent() (also from this component) and authenticate() (From the authenticator) should then be called with the correct details for the new user.
Output Specification	Both getUser() and authenticate() should return the correct user item. getStudent() should return the correct student item (Only valid once tests A002, A011 and A013).

Test Case	A009
Use Case	A logged in Coordinator wishes to create a new Employer User Account
Input Specification	addEmployer() is called with the correct arguments for a specific new user. getUser(), getEmployer() (also from this component) and authenticate() (From the authenticator) should then be called with the correct details for the new user.
Output Specification	Both getUser() and authenticate() should return the correct user item. getEmployer() should return the correct employer item (Only valid once tests A002, A011 and A014).

Test Case	A010
Use Case	A logged in Coordinator wishes to create a new Visitor User Account
Input Specification	addVisitor() is called with the correct arguments for a specific new user. getUser(), getVisitor() (also from this component) and authenticate() (From the authenticator) should then be called with the correct details for the new user.
Output Specification	Both getUser() and authenticate() should return the correct user item. getVisitor() should return the correct visitor item (Only valid once tests A002, A011 and A015).

Test Case	A011
Use Case	The System wishes to retrieve a User item from the UserStore
Input Specification	A valid username is passed to the getUser() function.
Output Specification	The correct User item is returned

Test Case	A012
Use Case	The System wishes to retrieve a Coordinator item from the UserStore
Input Specification	A valid username for the coordinator is passed to the getCoordinator() function.
Output Specification	The correct Coordinator item is returned

Test Case	A013
Use Case	The System wishes to retrieve a Student item from the UserStore
Input Specification	A valid username for the student is passed to the getStudent() function.
Output Specification	The correct Student item is returned

Test Case	A014
Use Case	The System wishes to retrieve an Employer item from the UserStore
Input Specification	A valid username for the Employer is passed to the getEmployer() function.
Output Specification	The correct Employer item is returned

Test Case	A015
Use Case	The System wishes to retrieve a Visitor item from the UserStore
Input Specification	A valid username for the Visitor is passed to the getVisitor() function.
Output Specification	The correct Visitor item is returned

OfferManager

Test Case	A016
Use Case	A student wishes to inform the system of an accepted offer
Input Specification	notifyAcceptedOffer() is called with a range of acceptable variables. getStatus (from the placement store) should then be called on each placement
Output Specification	When getStatus is called on each placement it should return Rejected, not an error

Test Case	A017
Use Case	The Course Co-ordinator wishes to approve a placement
Input Specification	approveAcceptedOffer() should be called on a range of placements. getStatus (from the placement store) should then be called on each placement.
Output Specification	getStatus should return Accepted

Test Case	A018
Use Case	A Student wishes to inform the system of an accepted role from outside of the system.
Input Specification	createNewSelfSourcedRole() is called with a range of acceptable variables.
Output Specification	The function should return the correct role.

PlacementStore

Test Case	A019
Use Case	The user wishes to view all placements
Input Specification	viewPlacements() should be called
Output Specification	The UI should list all placements

Test Case	A020
Use Case	A user wishes to add a placement
Input Specification	addPlacement should be called with a range of placement items
Output Specification	When eviewPlacements() is called, the new placements should also be displayed

Test Case	A006
Use Case	The coordinator wishes to remove a placement
Input Specification	removePlacement is called with the correct placement id
Output Specification	listPlacements no longer displays the placement.