Traceability Matrix

For each of the two use cases, you need to create the following table showing traceability of your designs that you've produced.

Use Case	Detailed use case scenario		Related classes & methods in the class diagram	Description about what to do
	Actor	**System**		
Match job seekers	1. The recruiter choose a created job to search job seekers. 2. The recruiter views the interested seekers' profile.	1.1 extract job keywords of job. 1.2 mark job seekers based on the matching algorithm. 1.3 rank job seekers based on their matching score. 1.4 display a ranking of job seekers. 2.1 display this job seeker's profile.	1. <<Boundary>>RecruiterUI getThisJobId():int 1.1-1.3 <<Controller>>Matching matchByJob(int):ListMultiMap<Seeker,int> <<Controller>> JobDataMaganer getJobByID(int):Job <<Controller>>SeekerDataManager loadAll(): Collection<Seeker> <<Controller>>Matching markJobSeeker(Job,Collection<Seeker>):ListMultiMap<Seeker,int> <<Entity>>Job getRelKeywords():ArrayList<String> <<Entity>> Seeker getSkillSet():ArrayList<String> 1.4 <<Boundary>>RecruiterUI diplayMatchingResult(ListMultiMap<Seeker,int>) 2 <<Boundary>>RecruiterUI getThisSeekerId():int 2.1 <<Controller>>SeekerDataManager getSeeker(int): Json <<Boundary>>RecruiterUI displaySeekerInfo(Json)	1.The recruiter presses "matching" button on a specific created job, activating the button listener to call getThisJobId method, passing the job id to the controller class 1.1 - 1.3 Then the Matching class get the job_id and pass down to JobDataManager to get this job object and also call the SeekerDataManager to load all seekers existing in the database. Then pass down the job object and the collection of seekers to the markJobSeeker function. This function would call job and seeker entity class to get job relevent keywords and seeker's skill set and then calculate the marking scores, following by ranking the result based on the scores. The result will return in the ListMultiMap. 1.4 The boundary class Recuiter UI get this ListMultiMap to display to the actor. 2. When the recruiter press the button to see the seeker's profile, the button listener function would be activated and get this seeker's id and pass to the controller class SeekerDataManager 2.1 And SeekerDataManager class will get the seeker objects and parse it retun json type. The boundary class RecruiterUI receive the json and display to the actor.

Match job seekers	1. The recruiter matches the job and seekers by keywords. 2. The recruiter views the interested seekers' profile.	1.1 validate the keywords. 1.2 mark job seekers based on the matching algorithm. 1.3 rank job seekers based on their matching score. 1.4 display a ranking of job seekers. 2.1 display this job seeker's profile.	1. <<Boundary>>RecruiterUI checkInput(String[]) getKeywords(): String[] 1.1-1.3 <<Controller>>Matching matchByKeywords(String[]):ListMultiMap<Seeker,int> <<Controller>>SeekerDataManager loadAll(): Collection<Seeker> <<Controller>>Matching markJobSeeker(String[],Collection<Seeker>):ListMultiMap<Seeker,int> <<Entity>> Seeker getSkillSet():ArrayList<String> 1.4 <<Boundary>>RecruiterUI diplayMatchingResult(ListMultiMap<Seeker,int>) 2 <<Boundary>>RecruiterUI getThisSeekerId():int 2.1 <<Controller>>SeekerDataManager getSeeker(int): Json <<Boundary>>RecruiterUI displaySeekerInfo(Json)	1. The recruiter inputs the keyword(s) and selects "matching" button to ask the system executes matching function in the Matching class. 1.1-1.3 Then the Matching class calls the SeekerDataManager to load all seekers existing in the database. Then pass down the keywords and the collection of seekers to the markJobSeeker function. This function would call seeker entity class to get seeker's skill set and then calculate the marking scores, following by ranking the result based on the scores. The result will return in the ListMultiMap 1.4 The boundary class Recuiter UI get this ListMultiMap to display to the actor. 2.When the recruiter press the button to see the seeker's profile, the button listener function would be activated and get this seeker's id and pass to the controller class SeekerDataManager 2.1 And SeekerDataManager class will get the seeker objects and parse it retun json type. The boundary class RecruiterUI receive the json and display to the actor.
Search jobs	1. Seeker searches the job using keywords.	1.1 Search jobs based on the skill sets and keywords.	1.1<<Boundary>>SeekerUI getKeywords() <<Controller>>JobController searchJob(Seeker,String):Collection<Job> <<Controller>> JobDataMaganer getJobByKeyworks(String[]):Map<Job>	1.1 The seeker inputs the keyword(s) and selects "Searching" function. 1.2 Then the jobContoller class gets job objects by calling JobDataMaganer class function getJobByKeywords(String[]) Then JobDataMaganer class gets the related job collections and return the map type

	2. Seeker sees the interested jobs' details	1.2 Display a ranking list of job based on the relevant keywords. 2.1 Display the job detail that seeker choose.	<<Controller>>JobController markJobSeeker(Seeker,Collection<Job>):ListMultiMap<Job,int> <<Entity>>Job getRelKeywords():ArrayList<String> <<Entity>> Seeker getSkillSet():ArrayList<String> 1.2 <<Boundary>>SeekerUI diplaySearchResult(ListMultiMap<Job,int>) 2 <<Boundary>>SeekerUI getThisJobId():int 2.1 <<Controller>>JobDataManager GetJobint(): Json <<Boundary>>SeekerUI displayJobInfo(Json)	The JobController class gets the job relevant keywords by calling getRelKeywords() in the Job class and also get seeker details (skillset) by calling the getSkillSet() in the Seeker class. Then each matched job relevant keywords are matched with seeker's skillset respectively. The job will get a number (matched mark) after matching with the seeker's skillset. Then the searching result (a list of matched jobs) is ranked by markJobSeeker(Job,Collection<Seeker>) function according to the jobs' mark. The last, the ranking result is returned to the SeekerUI class. 1.2 If one or more than one job is found, a list of ranked jobs is displayed. If no result, the "No job matched" is displayed. 2. If the seeker wants to see the job details, select the job. 2.1 And the JobDataManager class will get the job objects and return to the SeekerUI.

Note the followings to describe the above table:

- **Use case**: Use case name
- **Detailed use case scenario**: up-to-date use case scenario that shows detailed interactions between actor(s) and the system.
- **Related classes & methods in the class diagram**: Referring to the action in the "System" side in the "Detailed use case scenario", specify what classes and methods in your class diagram are responsible for taking this action.
- **Description about what to do**: Describe what each method does mentioned in the "Related classes & methods in the class diagram".