***Group #3***

***Members: Asma - Zackline - Nina***

***Final Report***

***Due date: 11/1/2016***

***Course#: CSC 621***

***Description:***

*Company information:*

Dayesda (Emergency Management Company) is the company that we will do our project about. Dayesda is a startup company that provides enhanced emergency management *services* and *courses* in Saudi Arabia. Its services and courses are goal-oriented, cost-effective, and efficient. It has expertise and affiliation with U.S emergency companies and expertise from around the world.

*Examples of Service Beneficiaries:*

* Schools, hospitals, industries.
* Civil defense.
* Saudi Red Crescent Authority.
* Saudi Humanitarian Emergency Aid and Response Team (Saudi HEART).
* Ministries.
* Any Private sector who deals with hazards materials.

*Examples of Course Beneficiaries:*

* Civil defense workers.
* Saudi Red Crescent Authority workers.
* Saudi Humanitarian Emergency Aid and Response Team (Saudi HEART) workers.
* EMTs and Paramedics.
* Professionals and healthcare providers who are working in emergency management positions.
* Officials who working the Internal Ministry in positions relating to security and emergency management.
* Individuals who work in Ministry of Hajj as emergency planners and coordinators.
* Ministry of Health managers who are working in the Emergency Management section.
* Workers who deal with hazards materials.

*Services:*

1. Emergency management.
2. EOP(emergency operation plan)
3. Business continuity plans.
4. COOP(continuous of operation plan)
5. Infrastructure protection planning
6. Disaster response planning
7. Hazard mitigation planning
8. Hazard recovery planning
9. Preparedness planning
10. Hazard Vulnerabilities Assessment planning
11. Threat and Hazard Risk Assessment planning
12. Exercises and drills management planning
13. Walk in security assessment
14. Incident Response
15. Exercises and drills management evaluation
16. Infrastructure protection evaluation.
17. Disaster response evaluation
18. Hazard mitigation evaluation
19. Hazard recovery evaluation
20. Damage Assessment.

*Courses:*

1. CERT(community emergency response team)
2. Disaster Life Support (DLS)
3. Public Information Officer (PIO)
4. Disaster Rapid Initial Assessment
5. Counter-terrorism procedures.
6. ICS 300(incident command system)
7. ICS 400(incident command system)
8. Planning for children in disaster
9. FEMA course (10 courses from Fema 1 to Fema 12)

*DB requirements:*

The DB that we want to create for this company will cover the online service requesting process and the online course registration. Therefore, our data base will keep tracks of the company Employees, the services that the company provides for organizations, and the courses that the company provides for individuals, also the organizations’ information and the trainees’ information who benefit from this company Also, it will keep tracks of the course payments that are made by the trainees. We don’t need to track the organization payments since it is done after agreement and paper work, we will just store the price of requesting a service by an organization. So, the users of our data base are (Organizations, Trainees and Employees of our company). We will complete the requirements in details in the ERD section.

*Type of Queries in our DB:*

1. Create queries :

Create queries for all DB relations (employee, organization, trainee, course, service, request, payment, isassignedto, completion)

1. Insert queries:

* All our different users will be able to insert into the different relations

***Employees:*** Will be able to insert a record into the employee relation when they register.

***Organizations:*** Will be able to insert a record into the organization relation when they register.

***Trainees:*** Will be able to insert a record into the trainee relation when they register.

* Insert all services into service relation
* Insert all courses in course relation.
* Insert into is assigned to relation by an employee who is responsible for that.
* When an organization requests a service it will be inserted into the database as a request record.
* When a trainee enrolls in a course it will be inserted into the database as a completion record. Also it will insert its payment transaction in payment relation.

1. Update queries:

Our users will be able to make update in their various relations.

Let’s say an Employee, Trainee, or Organization has a name, phone number or address change, they should be able to make that update which will be captured in the relations. But a trainee will not be able to make any update in the other relations. An employee (CEO, Manager) can make update in the other relations, for example, updating the price of a request that made by a particular organization, according to the agreement with them after visiting.

1. Select Queries:

Our users will have the ability to access only the data that is related to them and nothing else. For example a trainee will be able to access information about himself and the courses he or she is enrolled in. An employee can also access information about both the trainee and the organization. For example:

* Each trainee can check on courses information that he had enrolled (course name, instructor).
* Each employee can check on service request information that he is assigned to.
* Each employee (Instructor) can check on courses information that he is teaching.
* Find all the requests information for specific organization by its name.
* Get all requests information in specific date.

1. Delete Queries:

We will not give all our users the ability to delete records from our relations because we are particular about our data integrity and giving our users too much access will cause us lots of issues. So, just the employees (CEO, Managers) can do delete. For example: Delete any request transaction if the organization didn’t agree with the company for any problem.

*ERD:*



*DB requirements:*

The DB that we want to create for this company will cover the online service requesting process and the online course registration. Therefore, our data base will keep tracks of the company Employees, the services that the company provides for organizations, and the courses that the company provides for individuals, also the organizations’ information and the trainees’ information who benefit from this company Also, it will keep tracks of the course payments that are made by the trainees. We don’t need to track the organization payments since it done after agreement and paper work, we will just store the price of requesting a service by an organization. So, the users of our data base are (Organizations, Trainees and Employees of our company).

* For each Employee, DB keeps track of unique employee identifier, name, address, and phone, his role (job) in the company, salary, and password.
* For each service, DB keeps track of unique service identifier, service name, service description (there are no more important characteristics other than these since the whole service depends on employee checking and paper work.
* For each request of a service, DB keeps track of unique request identifier, date of request and cost of conducting this specific service for this particular organization. The employee will visit the organization to check and do the work (service) according to that.
* For each organization, DB keeps track of unique organization identifier, organization name, and sector (private or public), location, phone, password.
* For each Trainee, DB keeps track of unique trainee identifier, name, and phone, password.
* For each course, DB keeps track of unique course identifier, course name, description, and course tuition, duration (number of weeks).
* For each payment by the trainee for a course, DB keeps track of unique card number, card type, and the billing address.
* Each service is requested via 0 or many request transactions, and each request transaction has exactly one service.
* Each organization requests via 0 or many request transactions. Each request belongs to exactly one organization.
* Each request transaction is assigned to 1 or many employees who work on it, and each employee works on 0 or many requests.
* Each trainee takes 0 or many courses, and each courses has 0 or many trainees. The same trainee can take the same course but in different year to refresh his certificate or to get one if he fails on the previous one (rule of the course maker).
* Each completion contains exactly one payment, and each payment can be included in many completion since the same trainee might be using the same card to pay for different completions.
* Each employee (who works as an instructor) teaches 0 or many courses, and each course is thought by exactly one employee since the company is small and have limited instructors.

*Relational Model:*

**

*How data will be accessed:*

* Data in all of our relations will be accessed using the SQL queries. Of course we will not have our users write and run SQL queries, instead in development, we will create the back end which is the database and write queries to run off the database.
* We will have front end application where the users can enter the required credentials to access the data.
* When the user enters the values, then the service which we using PHP, will link the front to the back end and search in the back end, the relations where it can find a value or values that matches the requirement the user enters.
* If a record or records are found that matches the criteria then the record or records are returned and displayed to the user else and empty set or error is thrown out.
* For each form or forms included in the application we may have to select from either one of our 9 relations or a join of multiple of the relations depending on the data that needs to be accessed.

*Indexing:*

* It is important that we put a mechanism for increasing the speed of our data search and data retrieval in place so as not to allow our users having to wait so long just to access and retrieve the information they need and that is where indexing comes in handy.
* On the other hand, indexes also cost a lot, requires too much of time and resources to maintain, therefore it is wise only to use them when you have a large volume of data to access, and if adding them speeds the data access and retrieval process.
* Dayesda, is a startup company and our data is not yet in large volume to the point of accessing and retrieving it takes a long time. Besides, the time to access the data without an index is approximately equal to the time to access and retrieve the data using an index.
* Also, all our queries now mostly depend on the PKs and FKs, and all are already indexes.
* But in the future, we could add more different queries, which need to add different indexes to improve them.
* To satisfy the requirements, we choose (Organization name, Trainee name, request date) as indexes since these tables will become large in future, and the selectivity of these indexes is acceptable.
  + *ALTER TABLE organization ADD INDEX (orgname);*
  + *ALTER TABLE trainee ADD INDEX (tfname,tlname);*
  + *ALTER TABLE request ADD INDEX (date);*

*FD closure Test:*

All of our relations have full functional dependencies, no partial or transitive FD. There is no need to do any FD test, since all of them are in the 3NF. We have created our relational model from the ERD, which was based on the first description of our company. It was a straightforward process, which meant we didn’t need to normalize any of them.

*Referential integrity constraints and ensure integrity:*

* Each unique request record contains two FKs (both Not Null), one matches one of the values in the PK column of the organization relation, and the other matches one of the values in the PK column of the service relation.
* Each record in a relation (is assigned to), that represents the relationship between request of the service and employee, contains two FKs (both Not Null) one matches one of the values in the PK column of the employee relation, and the other matches one of the values in the PK column of the request relation.
* Each unique course record contains FK (Not Null) matches one of the values in the PK column of the employee relation, which is the employee id who teaches this specific course.
* Since each trainee can takes the same course multiple times in different years. Each record in completion relation contains partial Identifier (year of completion) and two FKs (both Not Null) one matches one of the values in the PK column of the course relation, and the other matches one of the values in the PK column of the trainee relation. All together form the PK of this relation.
* Also, each unique completion record contains FK (Not Null) matches one of the values in the PK column of the Payment relation.
* All FKs in our relations are restricted to delete and all cascaded to update except the course id (crid) and service id (sid), both are restricted to update also, because we use them exactly in our interfaces.
* In all our forms that are related to executing the queries, first, we check on the existing of the value of the (FK) in the (PK) column of the referred relation, before executing the query, to ensure integrity. For example, we will not allow a trainee to enroll in a course until we check if the (FK) trainee id is existing in (PK) column in the Trainee relation, and the (FK) course id is existing in (PK) column in the course relation. If both ok, we can execute the query of inserting a new record in the completion relation that has the both existing (FKs).

*SQL of creation and population relations:*

The document is attached.

*Application usage:*

*According to the Dayesda description we drive our design as the following:*

1. Our users (Organization- trainee- employee) first, need to register to be able to use our services. When they register by using their registration interfaces, their records will be inserted in their respective relation.
2. After registration, the Organization now will be able to request a service which will result to an insert of a record in the request relation per each request.
3. After registration, The Trainee now will be able to enroll in a course resulting to an insert in the completion and payment relations. Also, the trainee can get a report of the courses that he enrolled in.
4. The Employee after registration can get a report of all the courses that he/she is teaching (instructor). Also, the employee (CEO, Manager) have ability to do updates, delete on some relations.
5. We will not allow deleting records from most of our relations, except the request and completion as transactions, because those can be deleted in case an organization doesn’t want the service after discussing the price or for any reason, and that should be before assigning an employee/s to work on applying this specific service for this organization. Also, if the trainee changed his/her mind and doesn’t want to complete the course, we can delete that record from the completion relation. All these deletes can be done just by an employee. We don’t want to give more privileges to our users, because that will cause some problems with our DB constraints especially integrity.
6. Deleting records from any other relations are not allowed because we need all the information in them. We need to know all the information about any organization, trainee, and employee that have dealt with our company especially regarding the service requesting and course registration processes. We can’t delete the record of any employee since they could be in some service request or course records. So, how a trainee can get a report of the courses if that course might contain some deleted instructor? He will not be able to know the name of the instructor who gave him that course. The same problem will be if the organization wants to know who worked on a specific requesting if that employee was deleted.
7. Also, we can’t delete any course or service if they have any related records for the same reasons as above. Our company is a small company and our services and courses all fundamentals and reliable and they will be offering unless the whole company shut off.

*Queries that we created the interfaces for:*

Our Database contains many queries that the end-user can utilize for their work. We built some of them to cover all queries types.

Insert:

1. An organization requests a service.
2. A trainee enrolls in a course.
3. An organization registration.
4. A trainee registration.
5. An employee registration.

Select:

1. Log in for all our users (employee, trainee, organization).
2. Course report for trainee- those he enrolled in.
3. Course report for instructor- those he is teaching.

Update:

1. A *manager* may update the price for a requested service.

Delete:

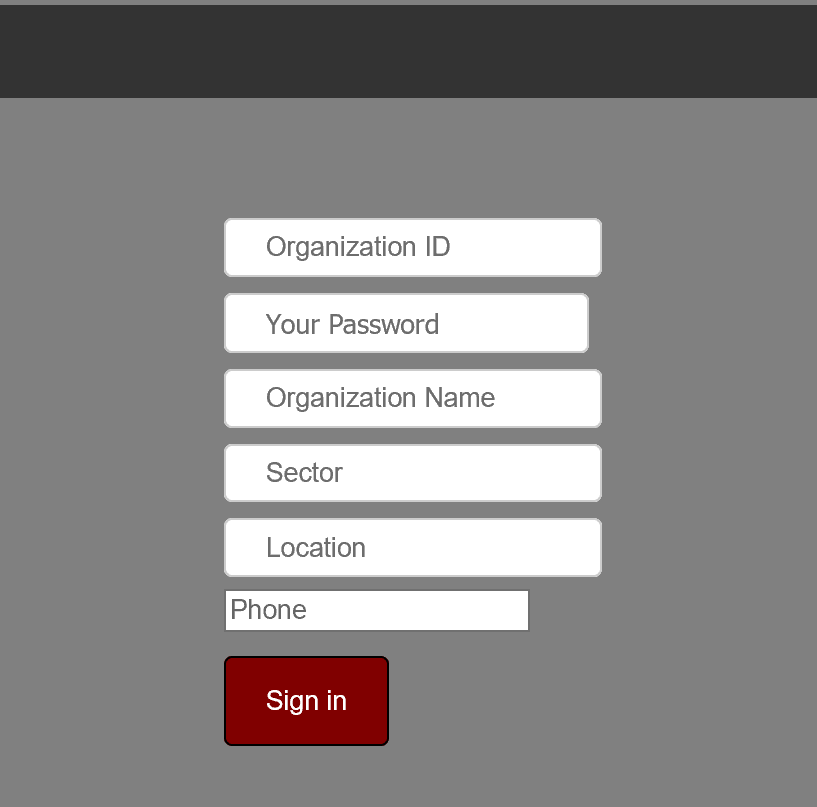
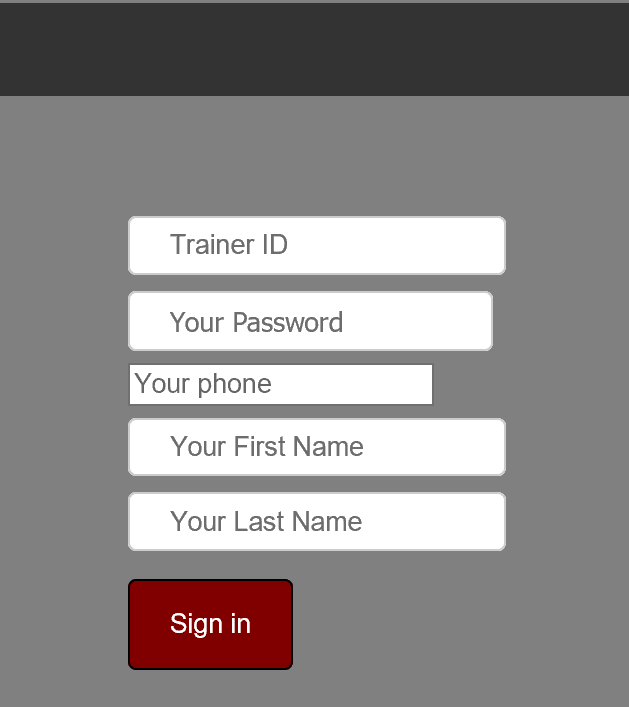
1. A *manager* may delete a course completion.
2. A *manager* may delete a service request.

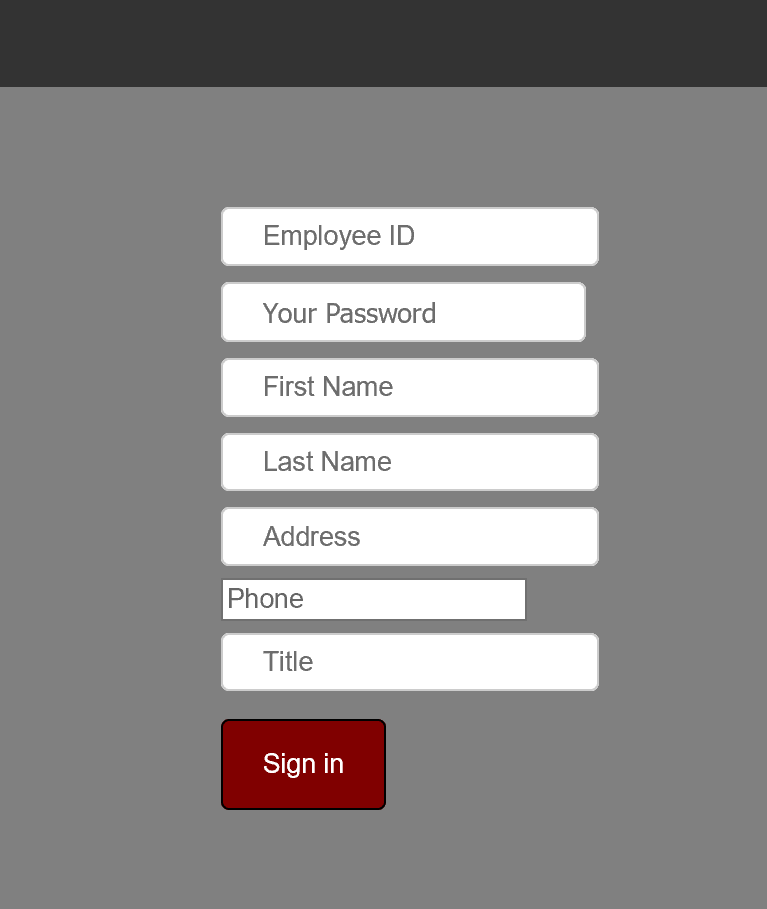
*Interfaces:*

*Home page:*

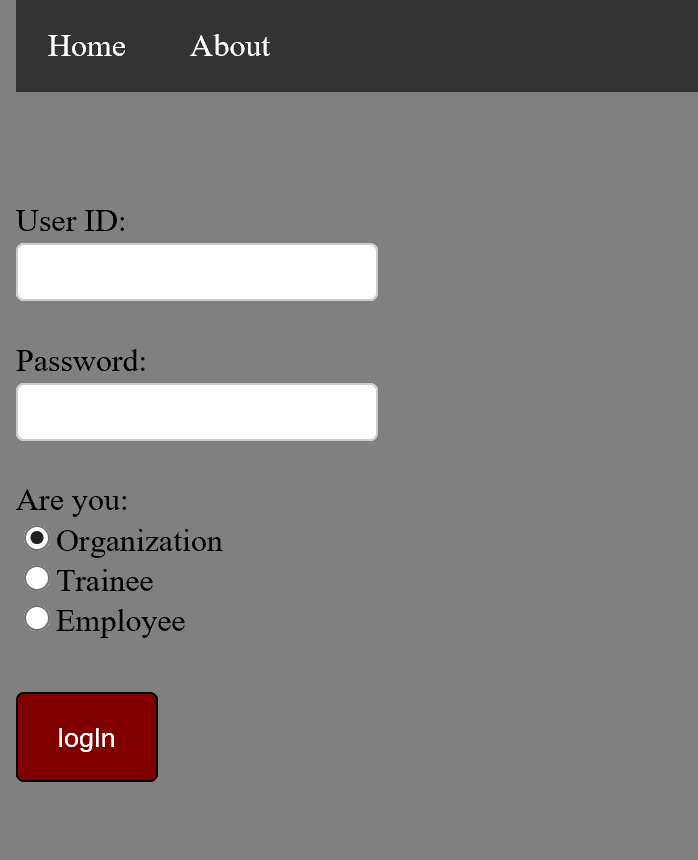


*Registration pages (each type of users will have a registration page):*

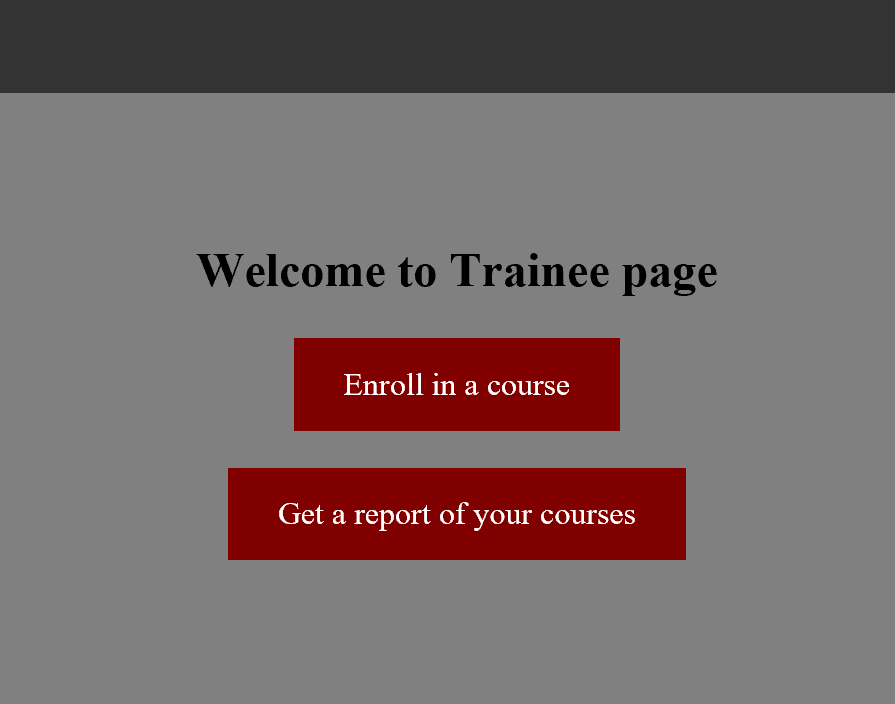
 



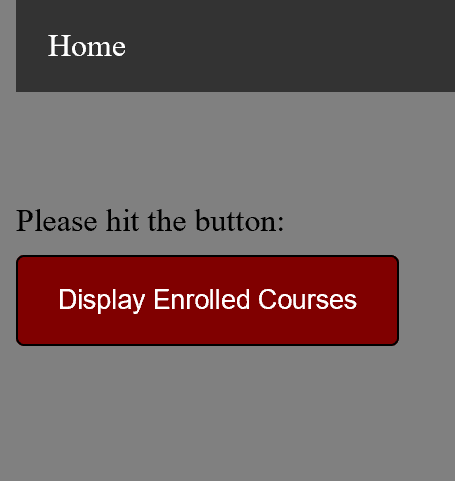
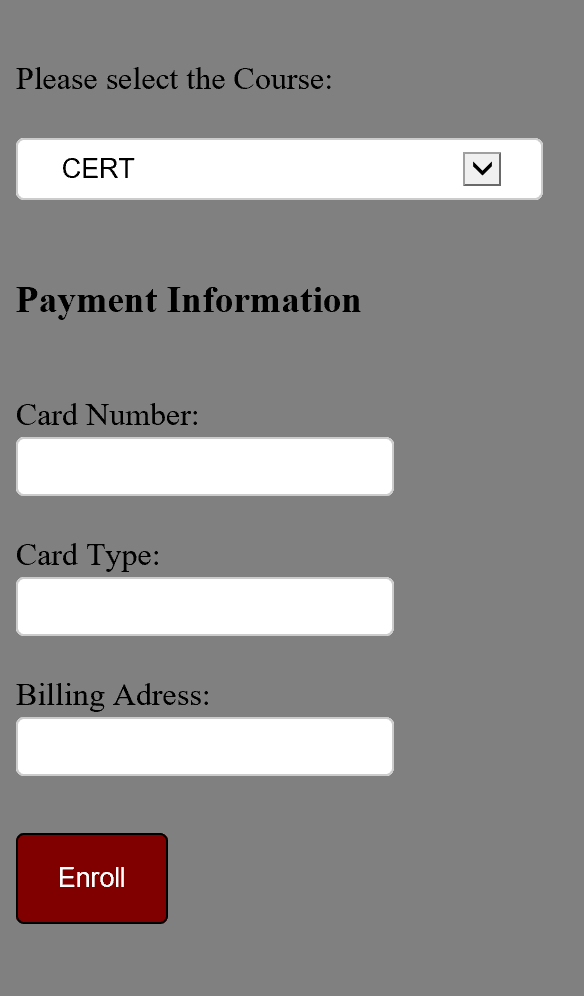
*Login page:*



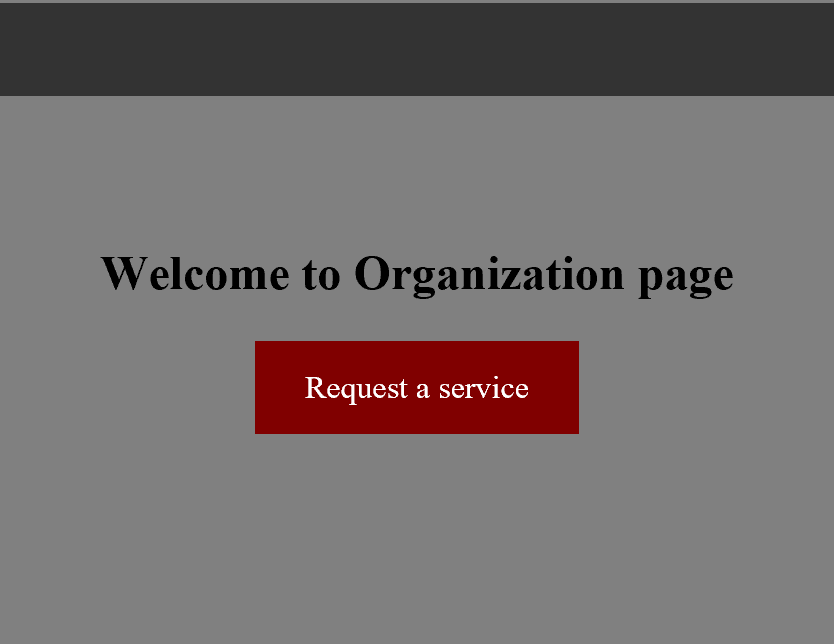
*Trainee interface:*



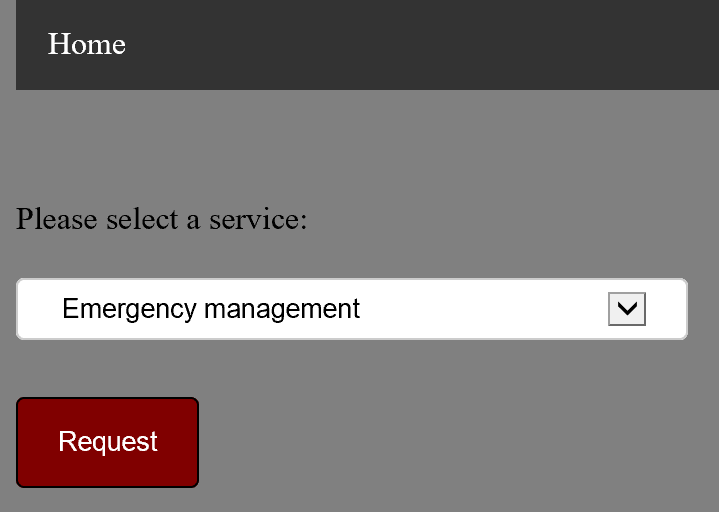
*Enroll in a course page and Display a report of enrolled courses (For trainee):*



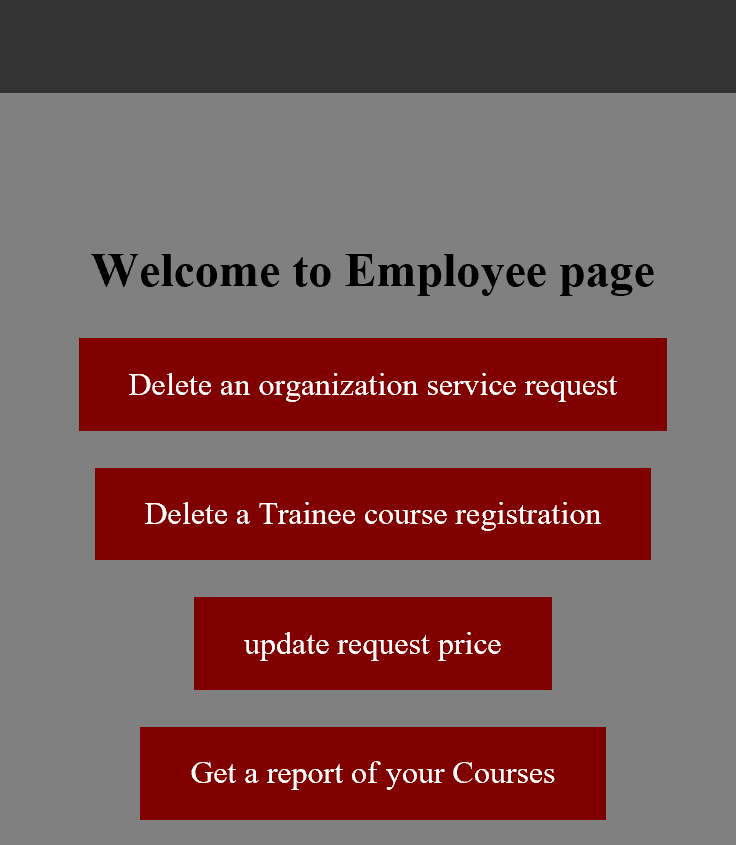
*Organization interface:*



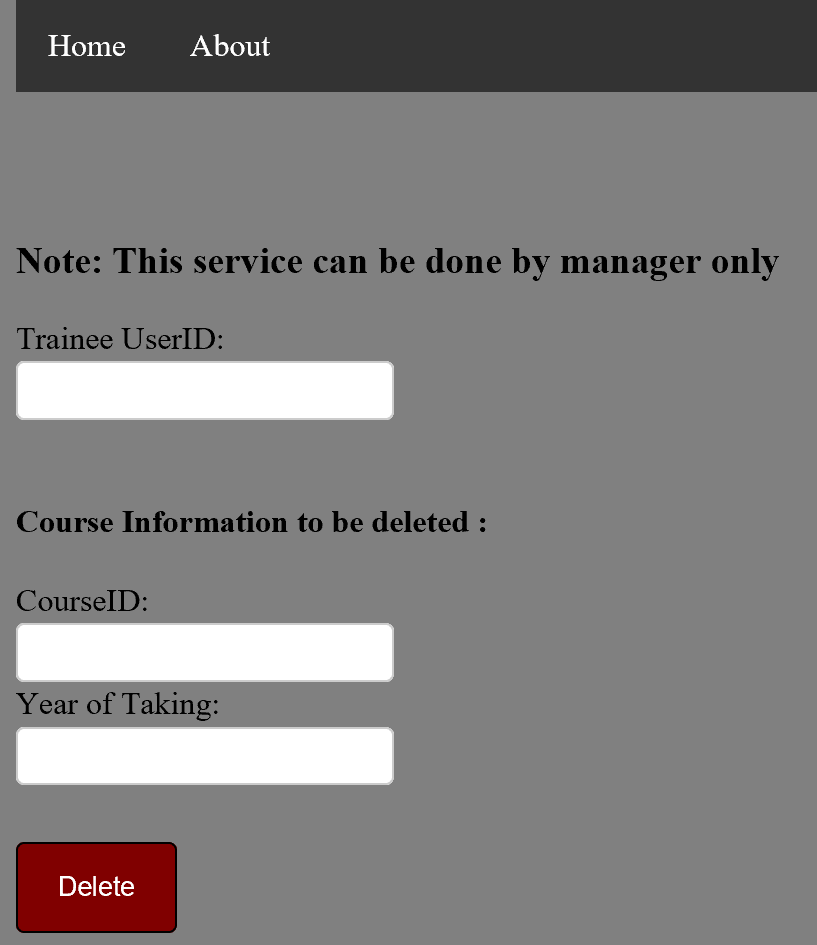
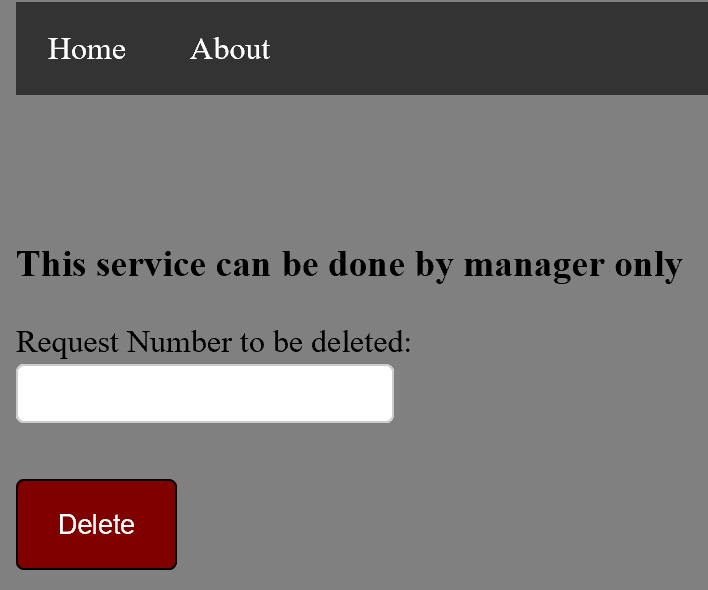
*Request Service (for organization):*



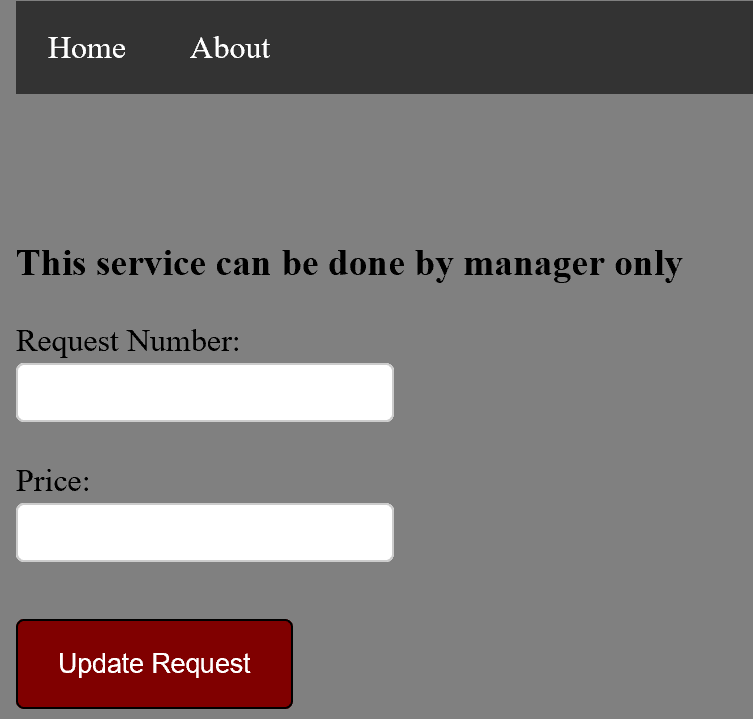
*Employee interface:*



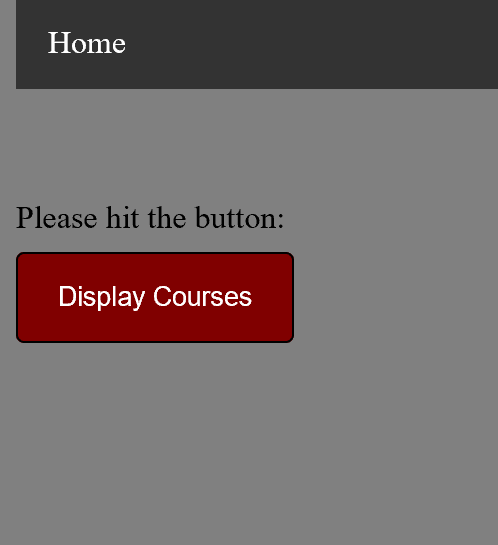
*Delete a service request and Delete a course registration:*



*Update a service request price:*



*Get a report of courses that you teach (for instructor):*



*Transactions:*

* The one update that absolutely requires a transaction to be built is payment. A user cannot be accepted into a course or proceed any further without submitting some portion of the amount owed through a valid card number, whether it be in full or the first payment in a payment plan. Additionally, adding a course requires that an instructor be assigned simultaneously. You cannot have a course with no instructor. Also, adding a service request requires that an organization and a service be assigned simultaneously. You cannot have a request with no organization or service. All other updates can occur independently.

*Error checking:*

* In the login page, we check first if the user actually is registered or not (if he has a unique record in our Database or not), to make sure none of the database constraints will be affected, and the executing of the queries related to him will work smoothly without errors.
* We make all fields in the forms required and checked by default, in that case, we make sure that our queries will run without missing some data, to avoid errors in executing.
* In all our codes (PHP), we check if the session is setting (and has a valid user from the login session) or not, and to make sure that our queries will run without errors in executing.
* Also, in all our codes (PHP), we handled the errors in case if the result of the queries is not correct (according to the need). Error messages or redirecting to the different page will happen.

*Work Schedule (whole team):*

|  |  |
| --- | --- |
| Date | Work |
| Week 1: Oct 20-Oct 31 | Project description- choosing the idea and agree on it. Working on collecting/defining the requirements of our DB/ discuss the vision of our DB to imagine the design/ set up GitHub repository for project. |
| Week 2: Nov 1-Nov 7 | ERD/relational Model/ constraints/ Indexing/Functional dependencies. |
| Week 3: Nov 8-Nov 14 | SQL of creating and population the DB. |
| Week 4: Nov 15- Nov 21 | Working on queries and their interfaces. |
| Week 5: Nov 22- Nov 28 | Error checking/ discuss transactions /Presentation. |
| Week 6: Nov 29- Dec 5 | Final paper/project presentation. |
| Week 7: Dec 6 | Finalize Project paper/ final presentation. |

*Ranking Form:*

*Member-1- Asma:*

My teammates and I agrees I did 35 % of the overall project.  My specific tasks are:

|  |  |
| --- | --- |
| Date | Work |
| Week 1 | * Write the description of the project (database requirements and the different users of DB and their requirements) after discussion with team. |
| Week 2 | After discussion with team:   * Write the Description of the implementing constraints. * Creating the ERD. * Creating the Relational Model. |
| Week 3 | * SQL for creating tables of the DB. * SQL for population 4 tables. * Write the application Usage. |
| Week 4 | * SQL of 5 queries and their interfaces (with screenshots). * Working in log in and sessions. |
| Week 5 | * Doing 40 % of presentation. * Write about Error checking after discussing with team. |
| Week 6 | * Finalize and write the final report. |

*Member-2- Zackline:*

My teammates and I agrees I did 35 % of the overall project.  My specific tasks are:

|  |  |
| --- | --- |
| Date | Work |
| Week 1 | * Write the list of tasks for the project and discuss the definition of the project with team and. |
| Week 2 | After discussion with team:   * Write on index selection and how the data will be accessed. * Write the DB requirements for the ERD. |
| Week 3 | * Revise SQL for creating tables of the DB. * SQL for population 4 tables. * Work on ensuring integrity after discussion that with team. |
| Week 4 | * SQL of 5 queries and their interfaces (with screenshots). * Working in log in and sessions. |
| Week 5 | * Doing 50 % of presentation. * Write about updates that require transactions after discussing with team. |
| Week 6 | * Finalize and revise presentation. |

*Member-3- Nina:*

My teammates and I agrees I did 30 % of the overall project.  My specific tasks are:

|  |  |
| --- | --- |
| Date | Work |
| Week 1 | * Discuss the definition of the project with team and revise the project description. |
| Week 2 | After discussion with team:   * Work on FD closure test. |
| Week 3 | * Revise ERD and Relational Model for any update. * SQL for population 1 table. * Revise the application Usage. |
| Week 4 | * Revise SQL queries and their interfaces. |
| Week 5 | * Writing the presentation outlines. * Doing 20 % of presentation. * Revise presentation. |
| Week 6 | * Finalize and Revise final Project paper. |

*Signed:*

Asma:

Zackline:

Nina: