CS 387 Database and Information Systems Lab
Project Report - ConsumerConnect

Astha Agarwal (110050018)
Anmol Garg (110050020)
Deepali Adlakha (11D170020)

November 16, 2013

1 Introduction - Application Domain

As our CS 387 project, we have designed an online service portal, whereby people can interact with service providers and other customers to seek opinion and decide the appropriate service provider/service they want to use. Here, 'customers' refer to other users on the database, who also interact in a similar manner.
On the other side, service providers will also be able to interact with users to know their potential customers. They can give details about the services they provide, and answer customer queries along with providing bids for services they provide and book appointments for it.
The project involves crowdsourcing, search and social networking.

2 The Functionalities Provided

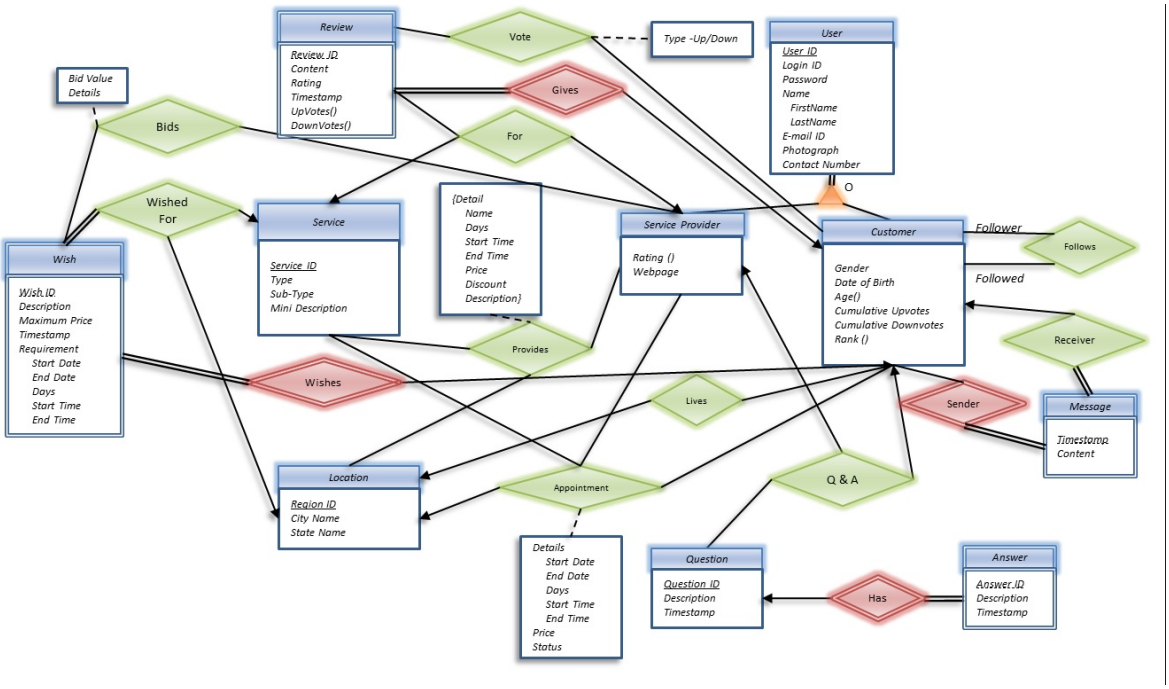
ConsumerConnect is a web application that provides an interface for people to buy and sell the services they need or provide. It is an eBay for services with a Social Networking Quotient which makes it more interesting and informal.

Broadly, there are two types of users of our application - customers and service providers. Here is a brief description of the facilities which will be provided to the users:

1. **Personal Details -** Users can provide their personal information like name, address, age, gender etc.
2. **Reviews -** Users can provide reviews and rate the service providers according to their experience. (Validating the experience can be seen as a further extension of the project.) Also, they can upvote ot downvote others' reviews.
   (a) Users will earn points when their reviews are upvoted. Their rank improves as the number of points increases.
   (b) The rating of Service Providers changes as and when they get reviews from respective customers.
3. **Follow People -** Users can follow other users and see their recent activity as news feed.

4. **Messaging -** Users can send a message to other customers to seek opinion or otherwise.
5. **Wishlist -** Customer can maintain a wishlist of the services required by him for future reference. Service Providers can bid on these wishes and specify the price they want to offer the service at.
6. **Service Details -** Service Provider can add the details of service provided by him such as service type, rates, availability etc.
7. **Appointments -** Customers can book an appointment with a service provider for a particular service. Service Providers will confirm the booking according to their availability.
8. **Bidding -** The wishlist of customers can be viewed by a service provider, and he can post a bid for the corresponding service. Customer can then make a decision by comparing various prices and options depending on the response of service providers.
9. **Search functionality -** There are three types of searches users can perform on the database:
   (a) **User Search -** To find other users by their name or part of name.
   (b) **Service Search -** To find required services based on type subtype and other availability rekated filters, and book appointments accordingly.
   (c) **Wish Search -** To search for wished put up by Customers so that Service providers can bid on them.

3 Final ER Model



4 Relational Model

4.1 For Entity Sets

Assuming E/R Model Approach for hierarchy relationship in Users (Customer, ServiceProvider):

1. User(UserID, LoginID, Password, FirstName, LastName, EmailID, Photograph, ContactNumber)
2. Customer(UserID, Gender, DOB, Age, CumulativePoints, Rank)
3. ServiceProvider(UserID, Rating, Webpage)
4. Service(ServiceID, Name, Type, SubType, MiniDescription)
5. **Many-to-Many Bad Relational Design -**
   Review(ReviewID, Content, Rating, TimeStamp, VotedByCustomerUserID, TypeOfVote)
6. Wish(WishID, Description, MaximumPrice);
7. WishTime(WishID, StartTime, EndTime);
8. Question(QuestionID, Description, TimeStamp);
9. Answer(QuestionID, AnswerID, Description, TimeStamp);
10. Country(CountryName);
11. State(CountryName,StateName);
12. City(CountryName,StateName,City);
13. **Many-to-One Bad Relational Design-**
    Street(CustomerUserID, StreetName, CityName, StateName, CountryName)

Note:

1. We have removed the CustomerID and ServiceProviderID, as they were redundant in providing unique identification to a type of user. In the E/R approach, we are anyway storing the UserID in Customer and ServiceProvider tables, and hence no need for another unique attribute.
2. We assume that there are no two end times for same start time, thus we haven't considered it as a primary key.
3. For the Many to Many relationship Votes from Review to Customer, we have added CustomerId as the attribute of Review, whereas there should be another relation relating Customer and Review for Votes. In this case, Review Information will be repeated and there can be an issue of consistency.
4. For the Many to One relationship Lives from Customer to Street, we have added the CustomerID to Street as an attribute. This will not only lead to repetition of information and hence wastage of resources, but also create another problem. We have a ternary relation from ServiceProvider and Service to Street.
   Now we will have to map one location of a service given by a service provider to all tuples corresponding to every customer living at that location, which not only wastes space, but will also complicate search and other functions. So, in the relation for relationships, we have not considered this model, but the correct model which assumes that there is no CustomerID in Street table.

For **OO Approach**, we have 3 relations again:

1. CustomerUser(UserID, LoginID, Password, Name, EmailID, Photograph, ContactNumber, Gender, DOB, Age, CumulativePoints, Rank)
2. ServiceProviderUser(UserID, LoginID, Password, Name, EmailID, Photograph, ContactNumber, Rating, Webpage)
3. CustomerServiceProviderUser(UserID, LoginID, Password, Name, EmailID, Photograph, ContactNumber, Gender, DOB, Age, CumulativePoints, Rank, Rating, Webpage)

Note:

1. Here, there is no table for only User, because user has total participation in the "is a" relationship, and has at least one role as a Customer and/or a ServiceProvider. We note that in this case as well, separate CustomerID and ServiceProviderID are not required to distinguish entries. So, we have removed the two attributes from the relations.
2. Another point to note here is the problem associated with this structure. Suppose there is a User, who has only a customer profile. He starts offering a service and then he makes a service provider profile associated to the same account. In that case, we have to shift all his details from CustomerUser table to CustomerServiceProviderUser table. This might be a cumbersome task. Also, there is no authority that dictates that UserIDs cannot be same in the three tables.

4.2 For Relationship Sets

1. Friend(CustomerUserID1,CustomerUserID2)
2. Message(SenderCustomerUserID, ReceiverCustomerUserID, Content, Timestamp)
3. Experience(CustomerUserID, ReviewID, ServiceID)
4. Bids(ServiceProviderUserID, WishID, BidValue, Details)
5. Wishes(CustomerUserID, WishID,ServiceID)
6. Provides(ServiceProviderID, ServiceID, StreetName, CityName, StateName, CountryName, Description, Rate, Discount)
7. ProvideTime(ServiceProviderUserID, ServiceID, StartTime, EndTime)
8. Appointment(CustomerUserID, ServiceID, ServiceProviderUserID, StreetName, CityName, StateName, CountryName, Rate, StartTime,EndTime)
9. QAndA(CustomerUserID, ServiceProviderUserID, QuestionID,AnswerID)

5 Normalisation

5.1 1NF

1. The entity and relationship sets have already been normalised to 1NF form.
2. The components of composite attributes like Appointment Details have been flattened when converting to Relational Model.

3. The multivalued attribute like details of a service provided by service provider was stored separately as a table ProvidesDetails. However, in this case, there was no non-primary attribute in table Provides. So, we removed the table Provides and renamed ProvidesDetails as Provides, as it stores all the relevant information.

5.2 2NF

1. User(UserID, LoginID, Password, FirstName, LastName, EmailID, Photograph, ContactNumber)
2. Customer(UserID, Gender, DOB)
3. ServiceProvider(UserID, Webpage)
4. Service(ServiceID, Name, Type, SubType, MiniDescription)
5. In the earlier relation Reviews, attributes CustomerUserID, Content, Rating, Timestamp are dependent only on the subset of the candidate key that is (ReviewID, ServiceID), hence a separate relation corresponding to this is formed.

   Review(ReviewID, ServiceID,CustomerUserID, Content, Rating, TimeStamp)

   Vote(ReviewID, ServiceID, VotedByCustomerUserID, TypeOfVote)
6. Wish(WishID,CustomerUserID, Description, MaximumPrice, StartDate, EndDate, Days, StartTime, EndTime, ServiceID, RegionID);
7. Question(QuestionID, Description, TimeStamp, CustomerUserID, ServiceProviderUserID);
8. Answer(QuestionID, AnswerID, Description, TimeStamp);
9. Message(SenderCustomerUserID, Timestamp, Content, ReceiverCustomerUserID)
10. In the earlier relation Location attributes RegionName, CityName, StateName, CountryName are dependent only on RegionID which is a part of the key, hence a separate relation is formed corresponding to it.

    Location(RegionID, RegionName, CityName, StateName, CountryName)

    Lives(RegionID, CustomerUserID)
11. Follows(FollowerCustomerUserID,FollowedCustomerUserID)
12. Bids(ServiceProviderUserID, WishID, CustomerUserID, BidValue, Details)
13. Provides(ServiceProviderUserID, ServiceID, RegionID, Days, StartTime, EndTime, Price, Discount, Description)
14. Appointment(CustomerUserID, ServiceID, ServiceProviderUserID, RegionID, Price, Status, StartDate, EndDate, Days, StartTime, EndTime)

5.3 BCNF and 3NF

1. All the functional dependencies listed above have already been removed by 2NF normalisation.
2. The one non-trivial dependency which remains is the dependency in User Relation. However, we will not split up the relation for this because both LoginId and EmailID are taken as candidate keys.
3. Hence, we end up with a sufficiently normalised relation model that complies with BCNF anf 3NF rules.

5.4 Denormalisation

For rating and other such derived quantities, we had removed them during normalisation. But, to compute them everytime on the fly was an expensive and undesirable operation. So we included the following attributes back to some entity sets, and used triggers containing multiple updates to update the requisite rating or vote count.

1. Added Upvotes and Downvotes to Reviews, which could otherwise be computed using Vote table.
2. Added Rating to ServiceProvider, which could be calculted as average of all ratings in reviews given to his services.
3. Added CumulativeUpvotes and CumulativeDownvotes to Customer which could be calculated by adding upvotes and downvotes to reviews provided by him.

After normalisation and denormalisation, we have the following tables in our database.

5.5 For Entity Sets

Assuming E/R Model Approach for hierarchy relationship in Users (Customer, ServiceProvider):

1. User(UserID, LoginID, Password, FirstName, LastName, EmailID, Photograph, ContactNumber)
2. Customer(UserID, Gender, DOB, CumulativeUpvotes, CumulativeDownvotes, Gender)
3. ServiceProvider(UserID, Webpage, Rating)
4. Service(ServiceID, Type, SubType, MiniDescription)
5. Review(ReviewID, CustomerUserID,ServiceID, ServiceProviderUserID, Content, Rating, TimeStamp, UpVotes, DownVotes)
6. Wish(WishID,CustomerUserID, Description, MaximumPrice, StartDate, EndDate, Days, StartTime, EndTime, ServiceID, RegionID, Timestamp);
7. Question(QuestionID, Description, TimeStamp, CustomerUserID, ServiceProviderUserID);
8. Answer(QuestionID, AnswerID, Description, TimeStamp);
9. Message(SenderCustomerUserID, Timestamp, Content, ReceiverCustomerUserID)
10. Location(RegionID, CityName, StateName)

5.6 For Relationship Sets

Most of the relations being weak relationships or many to one relationsip, were merged within Entity tables. This, in fact helped us to better manage our queries and results.

1. Follows(FollowerCustomerUserID,FollowedCustomerUserID)
2. Bids(ServiceProviderUserID, WishID, CustomerUserID, BidValue, Details)
3. Provides(ServiceProviderUserID, ServiceID, RegionID, Days, StartTime, EndTime, Name, Price, Discount, Description)
4. Appointment(CustomerUserID, ServiceID, ServiceProviderUserID, RegionID, Price, Status, StartDate, EndDate, Days, StartTime, EndTime)

5.7 Assertions

1. In case of Appointments and Availability of services, the End Time should be ahead of Start Time.
2. A user cannot send message to himself, or vote on his own reviews, or provide reviews for his own services, or follow himself, or book an appointment for his own service, or bid for his own wish or put a question for himself.
3. At the time of booking, the time for appointment should be subset of the time in Availability.
4. Rating should be between 0 to 5.
5. The timestamps in case of Questions, Answers and Reviews should be those of the past.
6. Discount should be between 0-100%, and prices and bids should be positive.
7. Dates, days and months should be apt.
8. DOB should be of the past.
9. Password should be of minimum 8 digits (includes one symbol, one number, one alphabet)
10. Vote type should be +1/-1.
11. Gender should be Male/Female.

5.8 Triggers

1. A review provided to a service provider triggers a change in the rating of the service provider.
2. A vote on an already existing review causes two changes :
   (a) It updates the total upvotes/downvotes for the particular review.
   (b) It updates the total upvotes/downvotes for the customer who has provided the review.

6 Test Plan and Results

6.1 Data

1. For our application, the data was generated randomly by writing PHP scripts for all the schemas. All the numbers were generated using random number generator.
2. We crawled online web databases to create random user profiles with profile pictures.
3. We used Lorem Ipsum for generating texts for messages, questions and answers.
4. We designed a random review genertaor, which provides a meaningful review for services, and rates them accordingly.

To test our application, we have manually written optimized SQL queries corresponding to the above functionalities and test it against the data collected.

6.2 Functionalities Implemented and Tested

Apart from testing the required functionalities mentioned in above sections, we also tested the following:

1. A customer can look out for a service at a particular location, with a particular availability. He can also see the reviews provided by other people for that service.
2. A user can search for other users in the database through a User search bar.
3. A customer can make his wish for a service public. A service provider can search for the customer having a wish for a particular service and the maximum price they can pay for it, therefore he can take selective decisions to increase his customer base.
4. A user can act as both a customer and service provider, so their were different interfaces and accordingly different queries for this interaction.
5. Review must be properly linked to the service provider and his service. There cannot be a review for a service that does not exist.
6. Asking questions to a service provider which are subsequently answered by the service provider, no other people get to interfere with this interaction.
7. Interacting with other customers via personal messages - Message must go to only the cutomer inteded.
8. UpVote/DownVote someone else's review on the basis of his own experience - Now, we have stored who upvoted which answer, and not to allow multiple upvotes on same answer by same user.
9. The customer can view his friend list, and reviews provided by them.
10. Book an Appointment on the basis of the Availability of the Service - A service can be booked only if it is available for the time period required by the customer.
11. Service provider can put out details of the services provided by him. He can also offer discounts.

6.3 Error Handling

1. Login - A user can login only when he provides the correct username and password. Else, he is shown an error.
2. Sign up - On sign up, a user has to provide a username that is not in use. If the username is in use, then he is shown an error message to choose a new user name.
3. Forms - In forms, if a user submits incomplete information, the data is not stored and user is prompted to fill the form again.

7 Performance Tests

For this project, our concentration was more on providing the functionality which we intended to provide. We performed some Query Optimization tests in the beginning where we wrote our queries as different SQL statements and timed their execution(facility provided by phppgadmin). This was important because being a dynamic social networking type of website, there will be millions of users in the database so we need to run the queries very fast. The times were not much different because we didn't have a million users for testing but still we used the SQL statements which had the best timing.

7.1 Performance Improvement - Cursors

We implemented Cursors which facilitated row-wise processing of our resultset. Thus rather than reading the whole query at once we read the queries 10 rows at a time. This improves performance and saves memory when the number of rows in the resultset is very large.

8 Conclusions

We were able to successfully create a system where Users(whether a Customer or Service Provider) can interact with each other.

A customer can look for all the service providers providing a particular service (service search functionality) and then choose the one which fits best according to his/her needs. In the pursuit of finding the best service provider for his required service he can seek help from the other customers who have taken service from the particular service provider through his reviews or can even message him personally.

A service provider can know who are the other service providers providing the same services, This helps the service provider to compete with them accordingly, They can also search for customers who wish for their service (wish search) and place appropriate bids for those wishes. This enables the service provider to interact with the customer directly.

A customer gets benefitted as he will have the best options available to him and thus can choose accordingly. On the other hand, service providers gets benefitted as they have a huge market of customers to showcase their services. It is a healthy platform for interaction between users. Considering the flourishng E-market, it can be a very successful idea being the first of its kind.

8.1 Further Improvements

1. **Customisation based on kind of service -** Depending on the kind of service one provides, he might want to have a customised profile through which he can properly showcase the facilities available to him. But at present, we are keeping a general information area for the same.
2. **Authentication of Customers and Service Providers -** How to know whether a service provider actually exists, or is it fake profile? We will be using Email-Activation for building an authenticated system.
3. **Authentication of Reviews -** Did the customer actually receive a service? Or is he providing fake reviews (Note that this is partly handled by the concept of customer rating which we are including in the project.)
4. **Followers of a Service Provider -** We have also thought of the concept of "following" a service provider, i.e. receive his updates, reviews etc. via notifications.
5. **Blocking a User -** We also wish to provide the facility of bloacking another user which will be helpful in case another user is unecessary troubling you.
6. **Notifications -**We can make the interface more user-friendly by providing a notification icon (like in Facebook) which will tell the user specifically whether his Question has been answered by the service provider, whether he has received any message etc.
7. **Integration with Social Platforms -** We can integrate it with existing social networks like Twitter, Facebook etc.

9 Appendix

We are submitting the Schema creation adn upload scripts along with the report.

9.1 Schema Creation

We have the SQL scipt for creating the database in sql/structure.sql. This includes the schema for all the tables in our database.

9.2 Data Upload

We have the data upload php scripts in data-upload-scripts.

10 Screenshots of the Interface

We have added the screen shots sepertely in a folder since there were many and the size of pdf was becoming very large.