



Order Anyday

Sub-Par Systems :: Jacob Adkins, Brian Galambos, Jon Widner, Brock Justice, Alvah Young

# Database Schema

kAs shown in Figure 1, we designed the database using BCNF reduction wherein redundancy is reduced throughout the design bearing in mind that logical joins between tables can produce the relational tables the application needs to process queries. We started with the user table as its column values are defined in the ASP.NET framework. We then added details to the user such as contact information that the system would need to meet the requirements as defined in the SRS and problem statement. The AspNetUserLogins (1), AspNetUserClaims (2), AspNetUserRoles(3), and AspNetRoles (4), follow in this same convention. (1) provides the system with the location of the user’s credentials. This is so that, in future builds, external login accounts can be added. For example, a user may wish to authenticate with their Gmail credentials. (2) provides tracking for users that have submitted “Forgotten your password claims”, since this alters their password hash and the system needs to be notified to not allow logins on that user. (3) holds a list of all of the possible roles a user can hold within the system. Our system employs only two, user, and admin. Finally, (4) maps each user to one of the two roles in the system.   
  
Now each user can place orders which are also stored in the database. Each order can be uniquely identified by its ID, and holds information about its status and dates relevant to the employee tracking the order. Also associated with each order is a cart. This holds the list of items that are placed in each order. Finally, we have products that can placed in the orders, and vendors to which these products are mapped. Vendors and Products both contain an “inactive” field that tell the system whether or not the information can be used in new orders. This is so that information can be “removed” from the system without corrupted information in old orders.   
  
There is also a notification system requiring a Notifications table that houses the notifications themselves, and a UserNotification table that maps notifications to users.

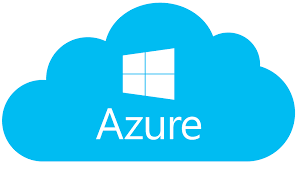
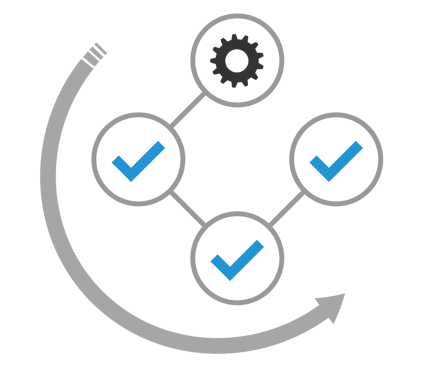
**Figure 1.**

# System Architecture Diagram

LINQ Transaction

HTTP request over ISP network

Perform Business Logic



**HTTP Response**

**Entity**

**Cloud Web Server**

**Web Browser**

**Figure 2.** As shown in Figure 2, a user makes a create, read, update, delete (CRUD) request via  
web browser of their choice. It is sent over their internet service provider’s (ISP’s) network to   
Microsoft Azure where the web app is deployed. On the server, ASP.NET employs the Entity  
framework to update the data within the application before passing it to the database where   
information is stored permanently for recall later. Note that the entity can also pass a READ   
request to pull data to populate the entity framework. The Entity framework on Azure then  
sends the response back to the browser where it is displayed to the user.