# SalonBook.com

## CIS 499 SENIOR PROJECT FINAL REPORT

Joshua Roth
Advisor: Norm Badler, Jeff Nimeroff
University of Pennsylvania

## PROJECT ABSTRACT

SalonBook is a web-based salon management application with appointment scheduling functionality. It connects clients, salons, and stylists in an online community allowing users to browse salons and stylists, and book or cancel appointments. Users can also write and read reviews of salons and particular stylists. Salons can specify the stylists that work at their salons, as well as the services they offer. Salons can also book appointments for customers, and can view and print schedules in convenient formats.

OpenTable.com serves a similar need in the restaurant industry, but nothing quite like this existed to bring clients and beauty salons together online. SalonBook.com fills this void in a way that is on-demand, easy to use, and effective for users and salon managers.

My project will use MySQL and PHP to back the interface with strong database functionality. For appointment scheduling, SalonBook will integrate WebCalendar as a backend database for appointments as well as a front-end scheduling interface. WebCalendar is a PHP-based calendaring application that can be a stand-alone program or integrated into other applications. This project will target the major web browsers as the initial platform for the Beta version.

The final deliverable will be a functioning web application that can handle all specified use cases. Some of the major use cases include user account registration, login/logout, appointment scheduling, schedule viewing and printing, adding stylists and services to a salon account, adding schedules to stylist account, accumulating points in a client account, writing and reading reviews for specified salons and/or services, and creating temporal salon promotions that users can browse and filter.

**Project blog**:
http://jrothdesign.blogspot.com/

# 1. INTROUDUCTION

## *1.1. Significance of Problem or Production/Development Need*

Being a salon client today is inconvenient. You have to remember that you need to make an appointment, then hope you remembered during business hours, and finally scramble to find the phone number and take time out of your busy day to make the call. And never mind finding user reviews of local salons and stylists; tracking those down somewhere on the web is a headache that most don't even attempt. This is the experience of countless salon-goers every day, even as we move well into the twenty-first century. There is a strong, well-articulated need for a "one-stop-shop" online hub that connects clients to salons, putting all the information users need in one convenient place, on demand.

Equally surprising, many salons today are still using paper and pencil systems to schedule appointments and manage customers. These systems are grossly inefficient at sharing and syncing information, and are prone to costly human error. SalonBook promises to end the dependence on paper-pencil systems, and put salons in control with an easy to use interface. SalonBook will be more than just an online scheduler; it will be a salon management tool that will allow salons to manage stylists and services, promote sales to customers, and track customer satisfaction.

## *1.2. Technology*

My project uses MySQL and PHP to back the interface with strong database functionality. To build the data tables I've used PHPMyAdmin, which is a web-based interface for MySQL. For the front-end development, I primarily used HTML and JavaScript widgets to style the site and create a simple, clean interface. To facilitate the process of styling the application, I used Adobe Dreamweaver. However, Dreamweaver is very limited, and the scope of my project quickly outgrew that of Dreamweaver's capabilities. Nonetheless, it was a very helpful tool early on.

For appointment scheduling, I'm going to integrate WebCalendar, a PHP-driven calendaring application, into my SalonBook application. After much research, I decided to implement WebCalendar rather than Google Calendars for a number of reasons. First, WebCalendar is installed locally and runs on MySQL. This means that all of the code running WebCalendar is in a subdirectory of my project. This gives me a great deal of control and customization of the application as I integrate it into my program. It also uses MySQL data tables that nicely sit right next to the tables storing data for SalonBook. Thus, it is a very clean integration, and a powerful tool. Google Calendars, on the other hand, would mean outsourcing functionality to Google. Any future changes to the Google APIs would cause serious problems with my program. Also, viewing and editing Google Calendars requires user authentication as a prerequisite. This means users of my site would need Google accounts or I would personally need to be the

“owner” of all the calendars. Neither of these options offered the neat, compact solution that WebCalendar provided, and therefore were not implemented.

## *1.3. Design Goals*

### 1.3.1 Target Audience.

There are two distinct target audiences for this project. Salon managers and owners make up the first primary audience for this project. They will use the salon side of the web application.

Salon-goers, primarily female, are the second major audience. They will use the client side of the application, primarily using it to browse local salons, view user ratings, and make appointments online, all on demand.

### 1.3.2 User goals and objectives

USE CASES:

**User**

- Create a new account
- Log in
- Search for salons by:
  - location (city, zip)
  - name
- View and Edit Account information
- Make an appointment for:

  Haircut (Male or female)
  - With our without color
  - With or without perm
  - With or without straightening
  - Specify stylist

  Beauty salon services

  **Allows users to make recurring appointments**
- Cancel an appointment
- View Appointments
- Rate salon/stylist (can only rate Salons/Stylists after an appointment)

- View user ratings of salon and stylist (filter by salon location, service)
- View “My Points”
- View Salon Promotions
- View Top-Rated Salons by
    - Hair Service
    - Beauty Service
    - Atmosphere
    - Customer Service

**Salons**

- Create a salon account
- Log in
- Edit Salon Account Information
- Add a Stylist
- Remove a Stylist
- Edit the services a stylist provides
- Add/remove beauty services your Salon offers
- View hair ratings of your salon by stylist
- View beauty ratings of your salon
- View/Print schedules
- Add and publish promotions
- Edit promotion information (start date, expiration date, promotion, services)

### 1.3.3 Project features and functionality

The tab menu on the Home Page allows any visitor to view the top rated salons by four different categories, and filter these results by zip code. This feature finds all salons that meet the zip code criteria (if applicable, otherwise all salons), and for each salon, re-calculates each salons average rating across the four categories every time the page is loaded. This ensures the information is always maximally updated. Then it displays the salons that have the highest average ratings by category. Users can click on the salon’s name in order to get the address and phone number of the salon.

When users sign in, they are directed to their dashboard, their one-stop-shop for all salon-related information and functionality. At the dashboard they are reminded about their points total and upcoming appointments. The JavaScript pull-down menus give the page a clean look while still providing an abundance of information and functionality. The dashboard also defaults to the Salon Search aspect of the application, where users can search salons by name or location.

Salons have a similar dashboard that is the root point for all use cases mentioned above. An important difference, however, is the integration of WebCalendar into their

management experience. WebCalendar is seamlessly embedded into the SalonBook application, and customized to look and feel like part of the SalonBook application. While in the WebCalendar application, salons are provided with a link back to their dashboard, as well as the footer at the bottom of the page with standard links to Home, About Us, etc.

## 2. Prior Work

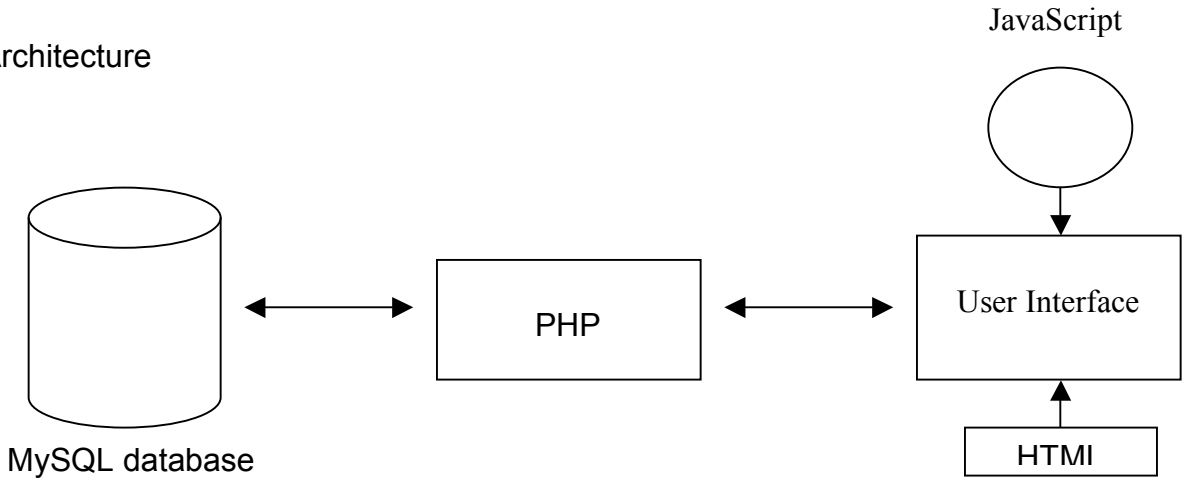
OpenTable.com is the inspiration and the benchmark for this project. OpenTable represents the primary body of relevant prior work. OpenTable developed a Windows-based restaurant management application that allows reservations to be made, cancelled, and viewed locally. Additionally, this application stores customer data and reservations locally. It is also in constant communication with the website, which is designed to allow diners to book reservations in real time. This model of local software combined with a web application is ideal because it maintains all data locally in case the internet fails, yet it still gives customers real time access to the reservation book.

Another noteworthy set of prior work is the use of Google Calendars in third party schedule-related applications. The Google Calendars API is robust, and allows developers to harness the functionality of Google Calendars as a back-end for storing and retrieving calendar events. One example is the Ithaca College school library website, which inputs its fluctuating hours into a Google Calendar, and then daily pulls that information into a MySQL database to then be displayed on the website front-end.

## 3. PROJECT DEVELOPMENT APPROACH

### *3.1. Algorithm Details*

Architecture



## Database Design

Many-to-one relationships are key to sound database design. In designing a database, inadequacies and inefficiencies that are noticed later in the development cycle can be costly, and so a lot of time and thought went into this early on. It is tempting to create database tables like classes in object-oriented programming. This would imply that a set of promotions, or stylists belonged to a salon, and thus are fields in the “salon” table. However, this method would be quite inefficient and limiting. Each entry would have to be unnecessarily large to accommodate many potential promotions and stylists, and yet still be finitely limited. Good database design means finding these redundancies and pulling information apart into separate tables that represent relationships between different tables.

It turns out that in the salon world, it is often the case that stylists work at multiple salons on different days, and sometimes offer a different subset of services at each salon. If stylists are tied to a column in a salon table, none of these dynamic relationships could be represented. However, I created a Salon-Stylist-Services table that stores a salon id, stylist id, and service id. While there may be many entries for each salon and/or stylist, there will only be one unique combination of a salon, stylist and service. Each entry represents the fact that a specific stylist offers a specific service at a specific salon. Further, when a salon removes a stylist, the stylist is not deleted from the stylist table. Rather, only the entries in the salon-stylist-service table that relate that salon to the stylist are removed. This is more natural because the stylist is not ceasing to exist; simply their relationship with that salon is. Thus, all the desired information is represented, and there is no redundancy because no combination of all three will ever be repeated. The same thing was done with promotions; I created a separate promotions table that stores the relationship between that promotion and the salon that created it.

## Displaying Dynamic Data from Many-To-One Tables

Many-to-one relationships are very powerful in representing data, but it does mean making a tradeoff in a different respect. Retrieving and displaying information to the end user becomes much more difficult this way. It means you are one, two, or sometimes three additional steps (database queries) away from the information you need. Here is one example of where I encountered this issue and how I overcame it:

When viewing salon promotions, users can filter promotions by salon name, zip code, or city. They can also specify hair and/or beauty services they want to filter results by. The salon name, zip, and city forms allow me to query the salon table with the user’s constraints. However, this returns an array of salon database entries. If I had stored promotions as columns in the salon table, I would be done, and could simply loop through the resulting salons and display the promotions within each entry. With my design, I need to loop through each of these salons still, but then perform a database query within this loop to find promotions in the promotions table that have the salon id

currently in the loop, as well as the service constraints that the user specified. This query can also yield multiple results (promotion entries). So then I need an embedded loop that goes through each of these results and puts them in a “results” array that exists outside all loops. Once all resulting promotions are put in the “results” array, step through to the next salon in the outer loop and repeat. Clearly, this is quite a bit more difficult as a result of the many-to-one database design. However, it is a tradeoff worth making because these retrieval scripts are one-time inconveniences to solve and write; whereas the enhanced database design continually increases the power and performance of the application, especially as it grows.

Delete in Batches, Insert in Batches

Another algorithmic concept came into play when salons add or remove services that a stylist offers at their salon (Same also applies to beauty services offered at a salon, but slightly more simple because no stylists are involved).

In this case, a salon chooses a stylist in order to edit their services. The first order of business is to properly populate the checkboxes with the services that the stylist currently offers. To do this, the script queries the many-to-one salon-stylist-service table for entries with the salon’s id that is currently logged (session variable) and the stylist id of the chosen stylist to edit (URL variable passed page to page when salon chooses stylist), and each service. If a result is found, the box is checked.

The next step occurs when the salon (un)checks the desired services and submits the form by hitting the submit button. One option is to handle each check box individually; but this would be inefficient and messy. Instead, I first find all entries in the salon-stylist-services table that relate this salon to the stylist at hand, and remove them. Then, in a loop for each checkbox, I add a new entry for this salon, the stylist, and the service tied to that checkbox IF it is checked. This is not only a much cleaner routine, but it also keeps all related data together in the database. This is very important for database management and upkeep as the entries grow over time.

$_POST versus $_GET variables

These two types of variables offer developers two different ways of passing information across pages and sessions, each with pros and cons.

$_POST variables are “posted” on an event, and then become available for use. $_POST variables are a secure way of passing information because they are only stored within a particular session. However, they are generally less stable than $_GET variables, and are also more prone to syntax problems. For example, $_POST variables do not work if the name of the variable has a space in it.

$_GET variables, on the other hand, are passed from page to page via the URL. They are very reliable and stable. Additionally, they offer the important benefit of allowing users to bookmark a page or send a link to a friend and go directly to that point. For a

social application like SalonBook, this can be very beneficial. It means users do not have to repetitively fill out forms again and again to return to a desired part of the site. However, the down side is that URL variables are inherently public. This means $_GET variables are not suitable ways to store sensitive or personal information, like passwords, emails, names, etc. While this did limit my use of $_GET variables, I made a conscious decision to use them as much as possible in order to enable to social book marking and sharing of links to my application. This choice enhances the user experience of SalonBook by limiting frustration and lowering barriers to revisit the site.

## *3.2. Target Platforms*

### 3.2.1 Hardware

Any internet-enabled computer

### 3.2.2 Software

Safari, Mozilla Firefox

## *3.3. Project Milestones*

- **Completed Literature Review***:* Over the past 8 weeks, I thoroughly researched the work and tools that have already been released. OpenTable.com is an exemplar of the application I am aiming to create in terms of functionality. I researched their implementation and platforms, in addition to exploring the actual application for functionality and features. I also reviewed the functionality and usability of Google Calendars, and determined that it can be used as a backend database for appointments as well as a potential front end user interface for appointments. I've also researched and familiarized myself with all the tools mentioned below, which I have gone on to set up and test.
- **Learned PHP and MySQL:** I purchased a self-teaching guide to PHP and MySQL and read the first 300 pages. This familiarized me with the language and the technology, and walked me through a few basic PHP script and MySQL query examples.
- **Set up PHP**: I downloaded and installed PHP on my computer. I wrote a few basic scripts to test my installation and confirmed that it was working.
- **Set up MySQL:** I downloaded and installed MySQL. I encountered problems with the daemon, but was able to change configurations using the Terminal to eventually get my computer to run the database.
- **Set up MySQL Query Browser:** Downloaded and installed this GUI in order to facilitate interaction with MySQL. I got it running and used it to create a new user account on my database, as well as the 'salonbook' database. Unfortunately, it was minimally helpful, and still required me to query MySQL using SQL code.
- **Set up phpMyAdmin:** Downloaded and installed this web-based MySQL interface. I found it to be extremely useful as a graphical interface for MySQL. I

used this tool to build my database tables and in some cases added elements to these tables (i.e. states).

- **Design and Create Database Tables**: Iterative design process to eliminate redundancies and create many-to-one relationships wherever possible
- **Set up ADODB:** I downloaded ADODB; I decided to use ADODB after thoroughly researching the many differing options available, including PEAR DB, Propel, Phing, etc. ADODB is a database abstraction layer that allows me to write portable PHP code not limited to MySQL. (chose not to use it)
- **Set up ActiveRecord:** ADODB implements ActiveRecord, which allows developers to create PHP objects that represent database entries. Get and set database queries can be done in an object-oriented fashion without writing any hard-coded SQL. I had a lot of trouble finding good documentation of ADODB ActiveRecord and getting it to work on my machine. Eventually I was able to debug my test scripts, and I now can get and set database records with customized PHP class objects. (chose not to use it)
- **Set up OpenLaszlo:** I downloaded and installed OpenLaszlo Explorer. I also began familiarizing myself with the declarative syntax it employs and the functionality it offers. (chose not to use it)
- **Download and Learn Dreamweaver**: I downloaded and installed Dreamweaver to help me create the HTML framework of the site. I spent time learning the tools and going through tutorials.
- **Build the Basic Interface:** Using Dreamweaver, JavaScript date picker, and Spry menu bars (JavaScript)
- **Build PHP Functionality:** Build the functionality of all use cases mentioned above except appointment scheduling. This was the largest part of the project, and was done over the course of 3-4 weeks.
- **Research Google Calendars API:** In depth review of API and assessment of its applicability to my project. I also researched other calendaring applications that could be potential options for this project.
- **Download and Install WebCalendar:** Installation included creating all MySQL database tables used by WebCalendar, and syncing it with my local server and MySQL database. I also studied the code (open source) to better understand how it was functioning and where useful functions were located.
- **Integrate WebCalendar into Salon Interface:** Customized the settings of WebCalendar to fit with my application. Also customized the look and feel of WebCalendar to make it seamlessly fit into the SalonBook application.
- **Build User Appointment Scheduling Functionality:** Built on top of the WebCalendar system by querying, inserting, updating, and removing entries in WebCalendar's data tables. This was an enormous step that took a long time to get working. WebCalendar was initially not responding to my PHP scripts as I expected, and so it requires significant studying of WebCalendar's source code and settings.

## 4. WORK PLAN

4.1 Project timeline

**1/30** Learn PHP and MySQL, literature review complete
**2/17** Design data tables and build them with phpMyAdmin
**3/1** Build PHP database objects using ADODB
**3/20** Initial version of user interfaces complete
**3/27** Integrate interfaces with database; start building PHP functionality
**4/5** Build functionality for all use cases except scheduling
**4/24** Full integration of WebCalendar system and build all user scheduling functionality
**5/1** Project and Presentation completed

## 6. REFERENCES


- Sams Teach Yourself: PHP, MySQL, and APACHE by Julie Meloni
- Google Calendar API: http://code.google.com/apis/calendar/docs/2.0/developers_guide_protocol.html#AuthAuthSub
- ADODB official website: http://adodb.sourceforge.net/
- Melonfire Web Articles about using ADODB: http://www.melonfire.com/community/columns/trog/article.php?id=142&page=8, http://www.melonfire.com/community/columns/trog/article.php?id=144&page=7
- Lacorna ADODB ActiveRecord documentation and examples: http://www.melonfire.com/community/columns/trog/article.php?id=144&page=7
- OpenTable.com

## 7. Work Log

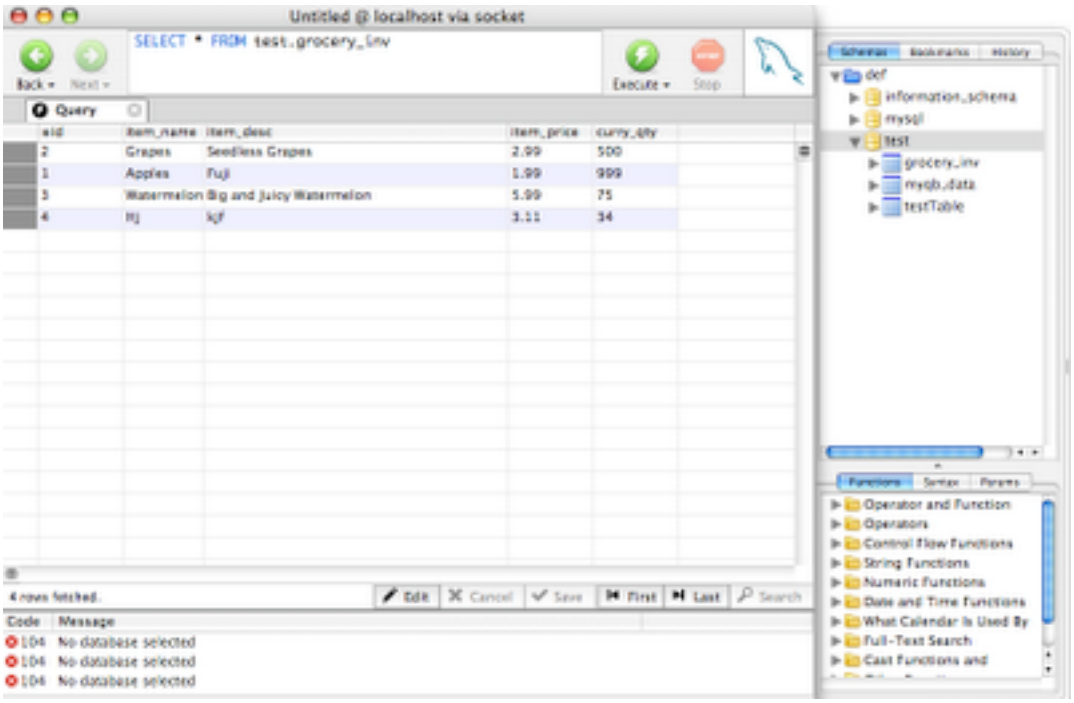
WEDNESDAY, JANUARY 28, 2009

### Design Choices

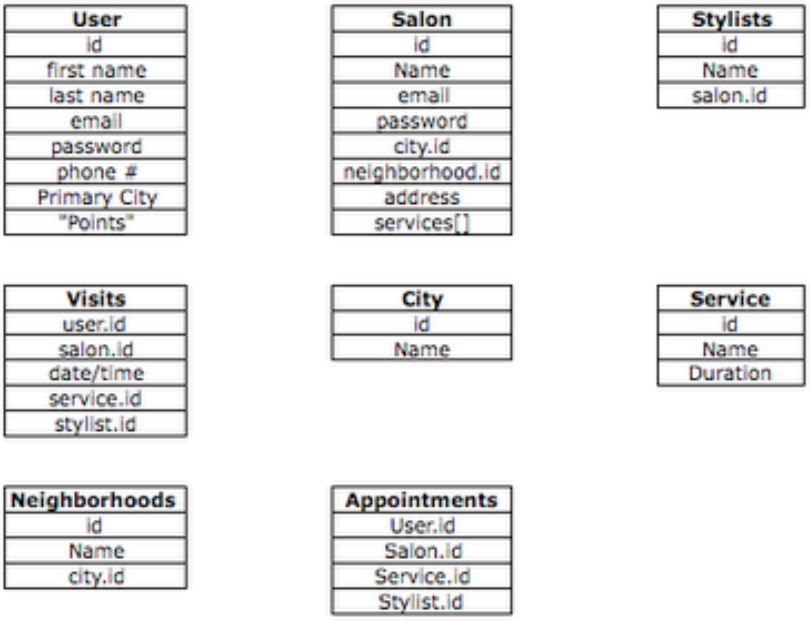
There are two fundamentally different approaches to creating a dynamic web application that is backed with data. The first is to employ relational data tables in a database (MySQL) to store, search, and retrieve data. I would then use PHP to communicate with the database, serving as the link between the web html and the MySQL database. The advantages of this approach are two fold. First, it makes search and sort operations very simple. Secondly, it supports a growing data set in a natural and well-organized fashion. The downside is that this design is fairly rigid once building begins. After the various tables are defined to capture the desired data, it becomes extremely difficult to redesign or augment the data tables to encompass more or different data.

The second option is to use text files and file readers to act as a pseudo-database. The advantages here are essentially the reciprocals of the disadvantages of the database design, and visa versa. While text files can be much more flexible to changes, they make search and sort operations far more costly.
Together with Professor Nimeroff, I've decided to design and implement a true database for this project. This choice means that my focus and time will be primarily geared towards building a functioning database that can be manipulated dynamically by users.

Because I have no experience with databases, PHP, or servers, I am currently studying the theory and syntax of these various aspects on my own. I've installed MySQL, Apache, and PHP on my laptop and have already experimented with these tools. Below is a relational table that I created and manipulated with MySQL Query Browser:




The first step in designing a database is to lay out all of the tables that you'll need and the information that they'll contain. Because of the above-mentioned drawback of databases, this step is critically important. The time spent perfecting these tables up front can save that time 100 fold down the road. There is a 3-step process explained in the MySQL book I bought for designing and flushing out data tables. I went though the first step and I'm currently on the second. These "steps" exist to help identify and eliminate redundancies in the data, making the database maximally efficient. Below are the tables (and their contents) that I have thus far:




The two biggest issues right now are how to handle appointments and how to handle an array of services (manicures, pedicures, massages, etc) beyond simply haircuts. For appointments, it seems natural to create tables that mirror the structure of a schedule. But, this would mean creating tables for each day, for each stylist. Intuitively this doesn't sound like a good thing. So for now, I have appointments modeled as stand alone events that get pushed onto the bottom of a growing table, and contain the critical information. This choice is still very much up for debate.

As for services, I can't figure out how to model them in terms of tables without ending up with one table having to store arrays. I'm not sure if storing arrays as table elements is possible or advisable. I'm going to have to discuss this with Professor Nimeroff and figure out if there is a way to capture this data soundly. If not, I may have to limit the scope of the website to just haircuts, and the basic services that accompany them (like coloring, highlights, perm, etc.).

---

WEDNESDAY, FEBRUARY 4, 2009

# The New Plan

After research, thought, and discussion with Professor Nimeroff, I've formulated a plan that shifts the focus of my work.

The mere scheduling aspect of the site is not novel, and the management of scheduling data in my own database would be extremely tedious and time consuming. So, I decided to look into the offerings of various calendar toolkits already available on the web. It turns out Google Calendars has all of the functionality that I would need for the scheduling aspect of the site. Google Calendars supports creation of new calendars, adding and deleting events (including recurring events), and viewing the schedule. The API for this toolkit is openly available online and will prove to be very helpful in harnessing this technology towards my project.

http://code.google.com/apis/calendar/docs/2.0/developers_guide_protocol.html#AuthAuthSub

While this calendar application will be very useful, my project will extend far beyond the simplicity of a generic calendar application. SalonBook is more than just a scheduler, it is an online salon management tool and salon community. There will be three distinct interfaces customized for three distinct groups of users. Salons, stylists, and clients will all create accounts and experience SalonBook is a different way. Below are the "use cases" that my site will set out to handle, organized by user type:

USE CASES

**User**

- Create a new account
- Log in
- Search for salons by:
    - location (address, neighborhood, city, state, zip)
    - name
    - service
    - price (range)
    - appointment time desired

- Make an appointment for:
  Salon
    - Haircut (Male or female)
        - With our without color
        - With or without perm
        - With or without highlights
        - (etc for all other hair services)

        - Specified stylist
        - Unspecified stylist

        - With or without notes to the stylist/salon

    - Other beauty salon services
        - (request for certain person or not in notes)

    - **Allow users to make recurring appointments**

- Cancel an appointment
- Rate salon/stylist
- View "My Points"
- Redeem "My Points"

**Salons**

- Create a salon account
- Log in
- Add/remove stylists (with the services they do)
- Add/remove hair services (for each stylist)
- Add/remove beauty services
- Add availability schedules (for each beauty service and stylist)
- Edit/Delete availability schedules

- Confirm stylist requests (and their services)
- Confirm stylist schedules
- View clients by:
    - Most recent visits
    - Frequency of visits
    - Total visits
    - Stylist's clients
    - Services received
    - Gender

- View ratings of the salon/stylists
- View/Print schedules by:
    - Day
        - Stylist
        - Hair Services
        - Beauty Services

    - Week
        - Stylist
        - Hair Services
        - Beauty Services

**Stylists**

- Create account
- Log in
- Accept/Reject salon requests
- Accept/Reject services
- Add/remove salons (and services at that salon)
- Add/Remove services (that they do at a particular salon)
- Add/Remove availability schedules
- View schedule by:
    - Day
    - Week

-View clients by
- Most recent visits
- Frequency of visits
- Total visits
- Services received
- Gender

---

After thinking through all of the use cases, I had to revisit my data tables in order to capture all of the data I would need. The use cases dictate what data you will need access to at what points in a user experience, and so as stepped through the data tables as dictated by the use cases, I found many deficiencies in my initial tables. In order to allow for more flexible relationships between salons, services, and stylists, I created tables with "many-to-one" relationships that could represent a wide array of cases without redundancies. I also decided to split "services" into two separate categories: hair services and beauty services. The two aspects are distinctly different when it comes to salon management and schedule organization. It became very messy to try to treat them as different instances of the same "service" object. Below is a screenshot of my new set of data tables as I've designed them so far:

Its clear from this that the scope and functionality of my website will go far beyond a generic scheduler. Google Calendars will assist in back-end management of only a small portion of the data being stored, managed, and queried. Building the three distinct user interfaces and linking all of these pieces and data together will be a substantial undertaking. Thus, Professor Nimeroff suggested Amazon Web Services as a potential way of offloading the grudge work of managing my own database. It turns out Amazon has a service called SimpleDB that is perfect for the needs of this project. SimpleDB allows developers to create and store data tables, then access and manipulate that data with simple function calls from my program.

http://aws.amazon.com/simpledb/

Using SimpleDB will allow me to focus my efforts on the interfaces and pull together a rich, cohesive, and novel application. To do this, I'm going to employ OpenLaszlo. Its syntax is relatively straightforward and is capable of deploying applications in a number of different formats, abstracting away gritty compatibility issues.

So, this is the new plan. The next steps from here are to finalize the data tables and use cases, and to create a features matrix with all the features I want for my application against all the various tools that already exist and are relevant to this project.

WEDNESDAY, FEBRUARY 11, 2009

# Database Tables!

After looking into options for database management interfaces, I found phpMyAdmin and was able to successfully download and integrate it with my Apache server. Using phpMyAdmin, I constructed a new database ("salonbook") and built all of the tables that I designed. The process was tedious but fairly straightforward.

I ran into some issues trying to figure out what exactly the different "indexing" options are. The different "index" options for a given field are: none, primary, unique, index, fulltext. I realize that for tables with unique id's, these are primary indexes into that table. However, the 'index' option is unclear to me. I'm hypothesizing based on research that it is appropriate for storing id's from other tables in a field, in order to link the data together. For now I used 'unique' for fields that should never have repeats appear in the table. One example would be email addresses for users; two users should never be able to have the same email address. I also looked into what 'fulltext' means, and I discerned that it allows you to search and sort data based on string matching. These indexing issues are important, and I plan on teasing out the details with Professor Nimeroff when I discuss this with him.

Another issue was whether to store "notes" as text types or as varchar types. I looked into the definitions and limitations on the two types, and they both have pros and cons. Varchars will truncate padding on the end of input, whereas text types will not do that. Text types have limited size, whereas varchars don't. Still unsure which I'm going to use though.

Date/Time types are also a mystery to me right now. Not sure which type to use between: date, time, datetime, timestamp. For now I used type 'datetime' for appointment times.

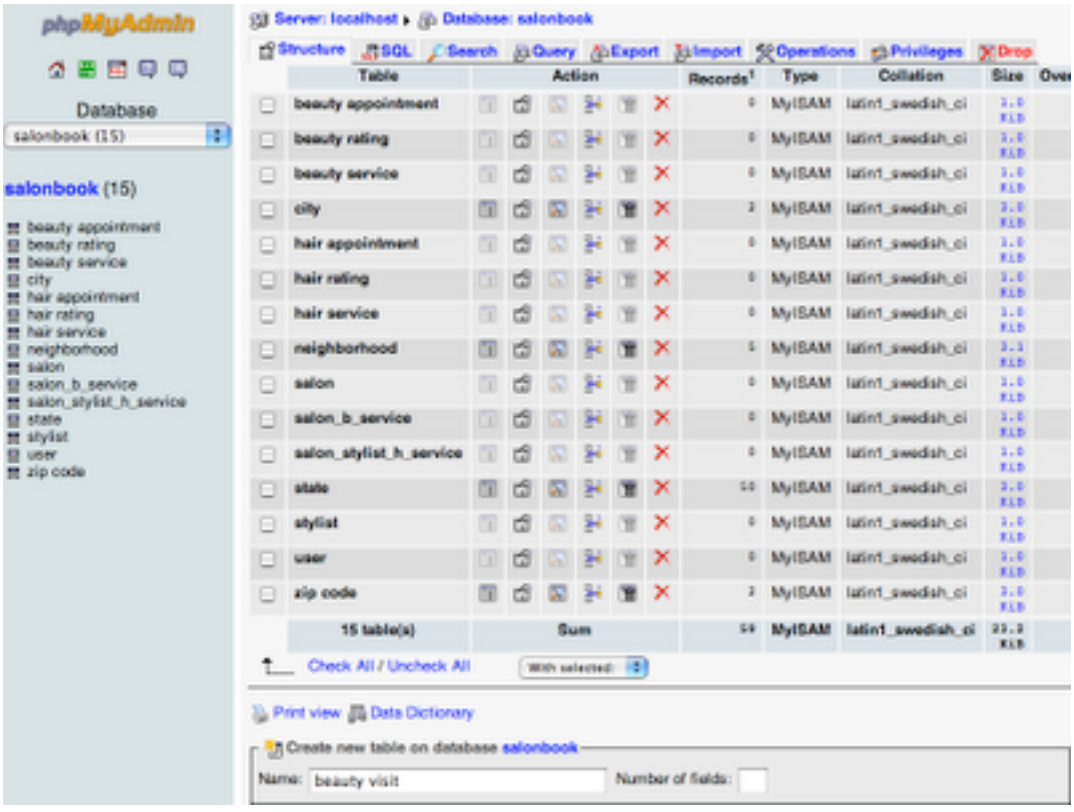
In order to avoid having sets or enums within tables, I established tables that feature many-to-one and many-to-many relationships. Examples are

*salon_stylist_h_service* and *salon_b_service*. This way there may be many table entries with the same stylist and/or salon and/or service, but never a repeat of any combination of these three values. This relationship allows for much more flexibility in terms of the use cases that can be handled. With this design, for example, I can represent the circumstance where a stylist works at various salons, and that he offers a different set of services at each salon.

I'm not sure if I am or am not going to store hair appointment information. Google Calendars can provide me with equivalent back-end functionality. However, I built the tables for now, if nothing else to maintain customer history information (which may prove valuable to salons down the line).

Here are some screenshots of my database and data tables I built:

salonbook database and its tables:




"user" table and its fields:

| | Field | Type | Collation | Attributes | Null | Default | Extra | Action |
|---|---|---|---|---|---|---|---|---|
| ☐ | id | int(255) | | | No | None | | |
| ☐ | first | varchar(15) | latin1_swedish_ci | | No | None | | |
| ☐ | last | varchar(20) | latin1_swedish_ci | | No | None | | |
| ☐ | email | varchar(30) | latin1_swedish_ci | | No | None | | |
| ☐ | password | varchar(15) | latin1_swedish_ci | | No | None | | |
| ☐ | phone | int(10) | | | No | None | | |
| ☐ | city id | int(255) | | | No | None | | |
| ☐ | gender | tinyint(1) | | | No | None | | |
| ☐ | points | int(255) | | | No | None | | |

Check All / Uncheck All *With selected:*

"state" table (with all 50 states entered into the table):

| | id | name |
|---|---|---|
| ☐ | 1 | Alabama |
| ☐ | 2 | Alaska |
| ☐ | 3 | Arizona |
| ☐ | 4 | Arkansas |
| ☐ | 5 | California |
| ☐ | 6 | Colorado |
| ☐ | 7 | Connecticut |
| ☐ | 8 | Delaware |
| ☐ | 9 | Florida |
| ☐ | 10 | Georgia |
| ☐ | 11 | Hawaii |
| ☐ | 12 | Idaho |
| ☐ | 13 | Illinois |
| ☐ | 14 | Indiana |
| ☐ | 15 | Iowa |
| ☐ | 16 | Kansas |
| ☐ | 17 | Kentucky |
| ☐ | 18 | Louisiana |
| ☐ | 19 | Maine |
| ☐ | 20 | Maryland |
| ☐ | 21 | Massachusetts |
| ☐ | 22 | Michigan |
| ☐ | 23 | Minnesota |

"neighborhood" table (with a few familiar neighborhoods entered):

| | | | id | name | city id |
|---|---|---|---|---|---|
| ☐ |  |  | 1 | South Miami | 1 |
| ☐ |  |  | 2 | Coral Gables | 1 |
| ☐ |  |  | 3 | Coconut Grove | 1 |
| ☐ |  |  | 4 | University City | 2 |
| ☐ |  |  | 5 | Center City | 2 |

*Because I store the 'city id' in the neighborhood table, it has indirect knowledge of the city to which the neighborhood belongs, and the city knows its state. So because of this structure, a neighborhood knows what city and state it's in without explicitly storing city or state name in the "neighborhood" table.

The next thing I need to look into and work on is setting up and learning how to integrate ADO DB. It'll play an integral link in passing information between my database and my interface.

---

SUNDAY, MARCH 1, 2009

## ADODB

ADODB is a database abstraction layer that allows developers to write portable code that is not tied down to one particular type of database. It also has an ActiveRecord feature that turns database rows into objects. That lets me work with object-oriented programming rather than tediously writing and rewriting MySQL queries. I've spent the last 10 days familiarizing myself with these tools and experimenting with them.

The first thing I did was research and learn to connect to my database with the tools built into PHP. I've successfully written a couple scripts that can connect to my database, create a PHP object, and perform an insert into the database, using the properties of that object as the field values for the table. One script looked like this:

```
<?php
class stylist{
```

```
public $first = 'not set';
public $last = 'not set';
public $email = 'not set';
public $password = 'not set';

function stylist($firstname, $lastname, $emailaddress, $pass){
$this->first = $firstname;
$this->last = $lastname;
$this->email = $emailaddress;
$this->password = $pass;
}

function insert(){
mysql_query("INSERT INTO stylist (first, last, email, password)
VALUES ('$this->first', '$this->last', '$this->email', '$this->password')");
}
  }

$con = mysql_connect("localhost","rjroth","aZsXdCf");
if (!$con)  {
 die('Could not connect: ' . mysql_error());
 }
mysql_select_db("salonbook", $con);

//Create new stylist objects
$bob = new stylist('Bob', 'Vance', 'Bob@Bob', 'BV');
$jane = new stylist('Jane', 'Dance', 'Jane@Bob', 'JD');
//Call the function that executes the MySQL query
$bob->insert();
$jane->insert();
mysql_close($con);

?>
```

This was great, but because the insert() function uses a hard-coded MySQL query, this code would be useless if my data were moved to a different

database. That's where ADODB comes in. Useful ADODB tutorials and examples were not easy to come by on the internet, but I was able to piece together a cursory understanding that allowed me to use ADODB to perform some simple scripts that can get and set data to MySQL. Here is one example of a script that SELECTS * from my "state" table and prints each element on a separate line and numbers them:

```
<?php
include("/Library/WebServer/Documents/adodb/adodb.inc.php");
 $db = NewADOConnection('mysql');
 $db->Connect("localhost", "rjroth", "aZsXdCf", "salonbook");
 $result = $db->Execute("SELECT * FROM state");
 if ($result === false) die("failed");
 while (!$result->EOF) {
 for ($i=0, $max=$result->FieldCount(); $i < $max; $i++)
     print $result->fields[$i].' ';
 $result->MoveNext();
 print "
\n";
 }

?>
```

This is a step in the right direction, but it still doesn't save me much tedium as a coder and it’s only a small step in the direction of object-oriented coding. ADODB implements a version of ActiveRecord that can make the entire process of getting and setting data object-oriented. Unfortunately, there are even less examples of code that use this online. Here's the most helpful one I could find:

http://phplens.com/lens/adodb/docs-active-record.htm

I tried to implement this and integrate it with my database. However, I was unable to get anything to work. When I ran the script no INSERTS would actually be made into database table, and thus far my debugging efforts have been futile. Below is the script that mirrors the example from the above link; it is designed to load a row from the "states" database into a newly created "state"

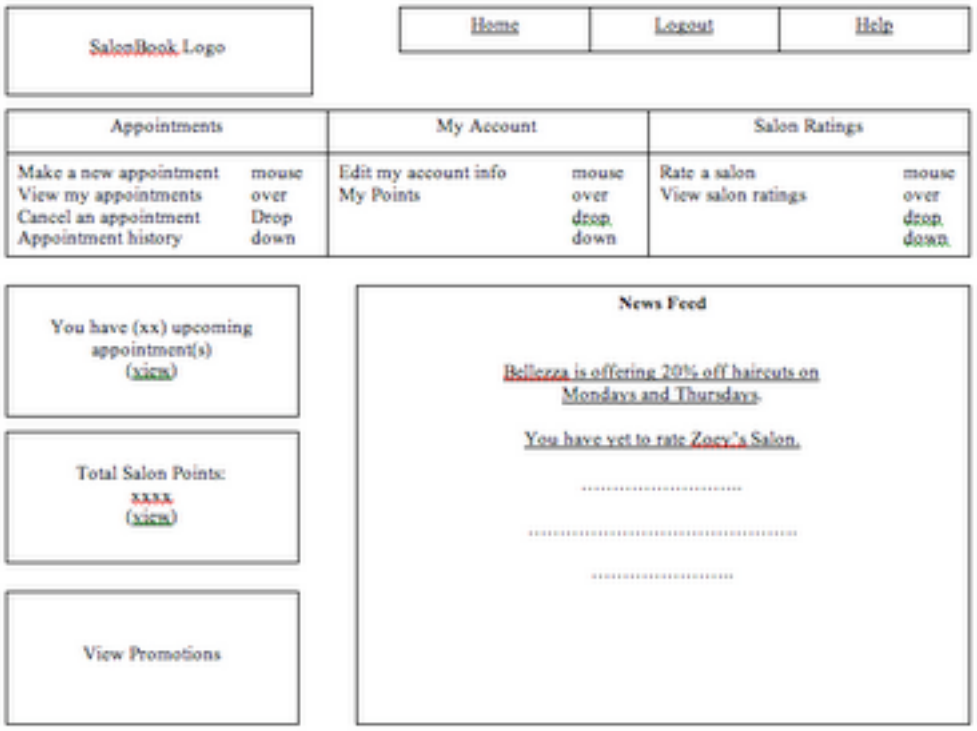
object, and then as a test I want to print out the ID from the state that was just loaded:

```
<?php
include('/Library/WebServer/Documents/adodb/adodb.inc.php');
require_once('/Library/WebServer/Documents/adodb/adodb-active-record.php');

// configure library for a MySQL connection
$db = NewADOConnection("mysql");

// open connection to database
$db->Connect("localhost", "rjroth", "aZsXdCf", "salonbook") or die("Unable to connect!");
echo 'hello world';
class state extends ADOdb_Active_Record
 {
 var $_table = 'state';
 }

$state = new state();
$state->load("id=5");
echo $state->id;

?>
```

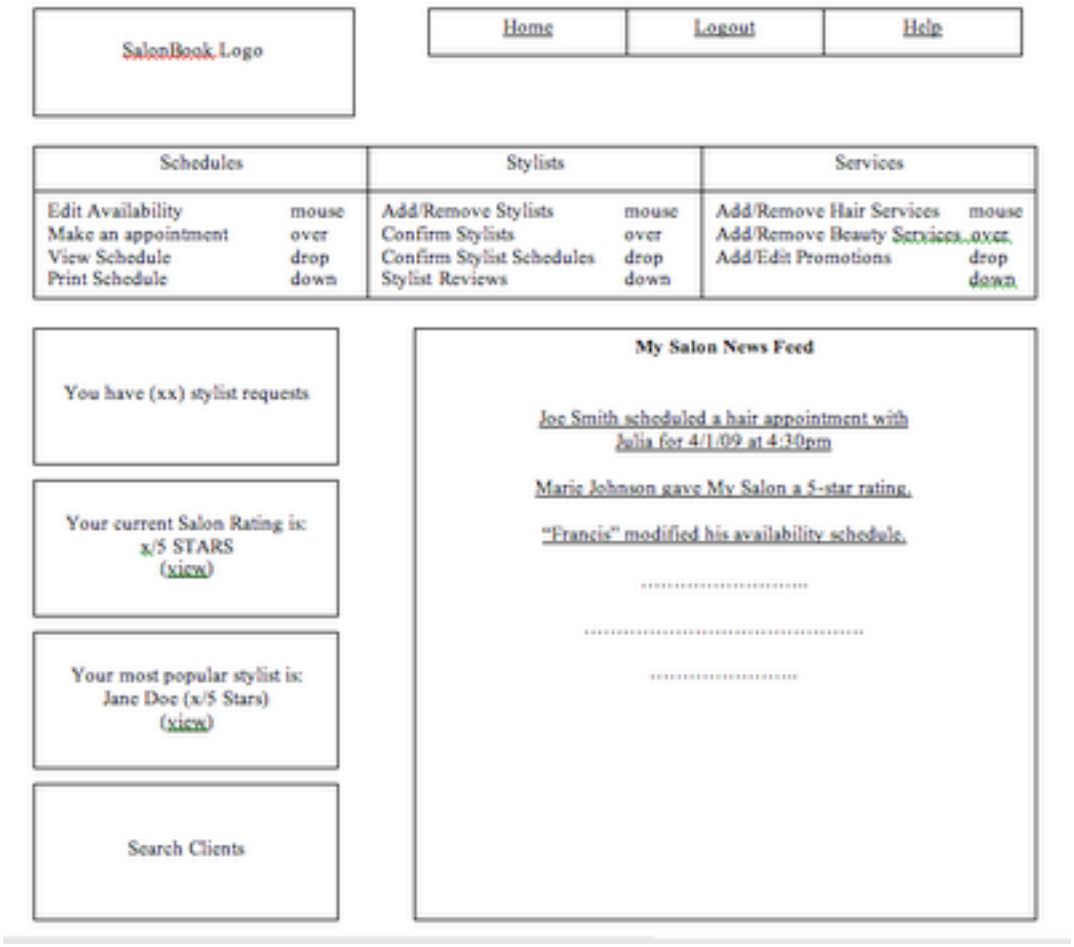
Unfortunately, I can't get this to work so far. Hopefully, with help from Jeff, I will turn the corner this week and go on to build all the objects I will need to represent my database.

SUNDAY, MARCH 22, 2009

# ADODB Working and Interfaces




First and foremost, I was finally able to get ADODB working. Now I can connect to my database using adodb and access my database tables using adodb ActiveRecord. This is important because it abstracts away the messy and database-specific SQL code and allows me to use object-oriented programming. I've run multiple test scripts that add, edit, and remove items from tables using ActiveRecord, and they are all functioning properly.




After getting over this hump, I shifted my focus to the other side of things...the interface. The first step in interface design is to layout a skeleton design for the critical web pages users will see. Thus far I've laid out the interface for the user dashboard and salon (manager) dashboard. In designing these interfaces, I came up with the idea of having a Facebook-style "newsfeed" for salons and users that will write relevant updates and news to the feed by pulling information out of the database based on date. I believe this feature will give my users a more rich, personalized experience. Also, the top navigation bar will feature mouse-over pull down menus so that the interface will be clean and minimal until a user seeks more information. Otherwise the interface will look cluttered. The other major interfaces will be the scheduling interface for the salon and for the user. For the salon, I will use the Google Calendars interface as well as their database functionality to present the full spectrum of calendar viewing and editing. For user scheduling,

I will have a form that asks for the user to specify the neighborhood, date, time, service, and salon/stylist if applicable when searching for an appointment. I will display the results with my own interface rather than Google Calendars for this part. It will simply show the available salon(s) and the time(s) available at and near the requested time. One of the main purposes of designing the interfaces was to make sure that my database would be capturing all of the information my interface would need. Once I designed these skeleton interfaces, I realized I would need a "Promotions" table that kept track of new deals and sales salons decide to offer. This information would be needed for the news feed, which could then pull these deals out of the database based on relevance (location, dates, etc) and serve them to users on the news feed. Thus, I created a new table in my salonbook database to support this information.

FRIDAY, APRIL 3, 2009

## User Interface

For the past two weeks I have shifted my efforts to the front-end interface of SalonBook. I installed Adobe Dreamweaver and Illustrator, and I have been using both to design the look and feel of the application.

I have been learning how to use Dreamweaver on the go, and it includes lots of helpful tools that can create database connection, session variables, and database queries. Of course, the built-in tools are not nearly advanced or complex enough to be sufficient for my needs, but thus far it has provided a good starting place and code infrastructure.

I struggled for a long time trying to figure out how to use session variables directly in SQL queries, but I was eventually able to figure out the syntax and method for accomplishing this.

For now, I am working on trying to build all of the functionality of the site with the exception of the scheduling aspect. This will likely prove to be the most challenging, from a conceptual standpoint as well as implementation

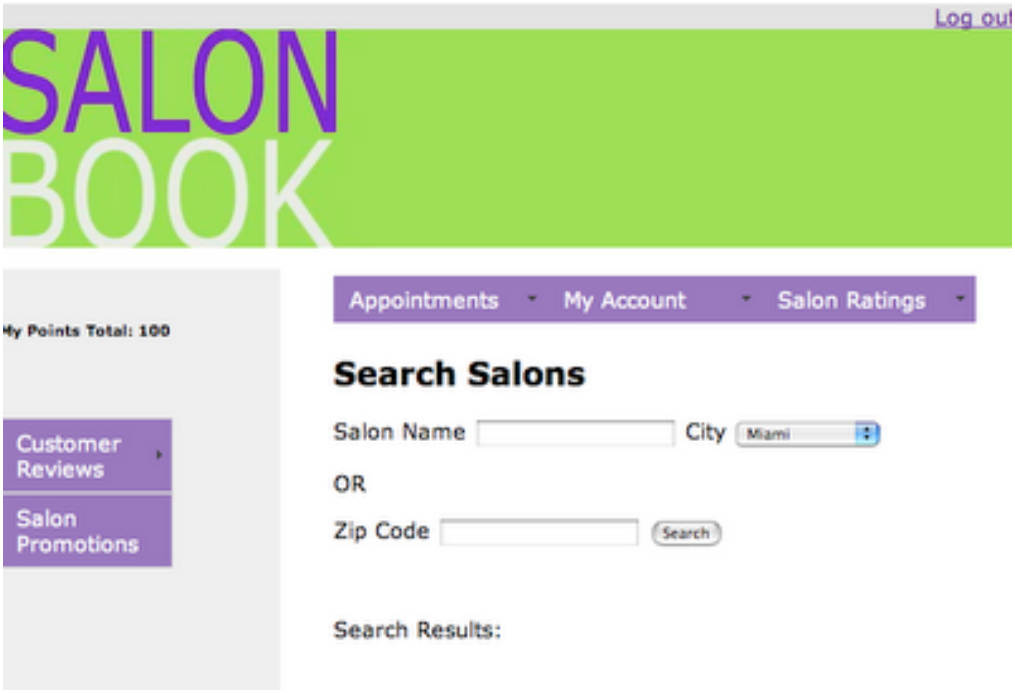
SUNDAY, APRIL 19, 2009

# UI Progress

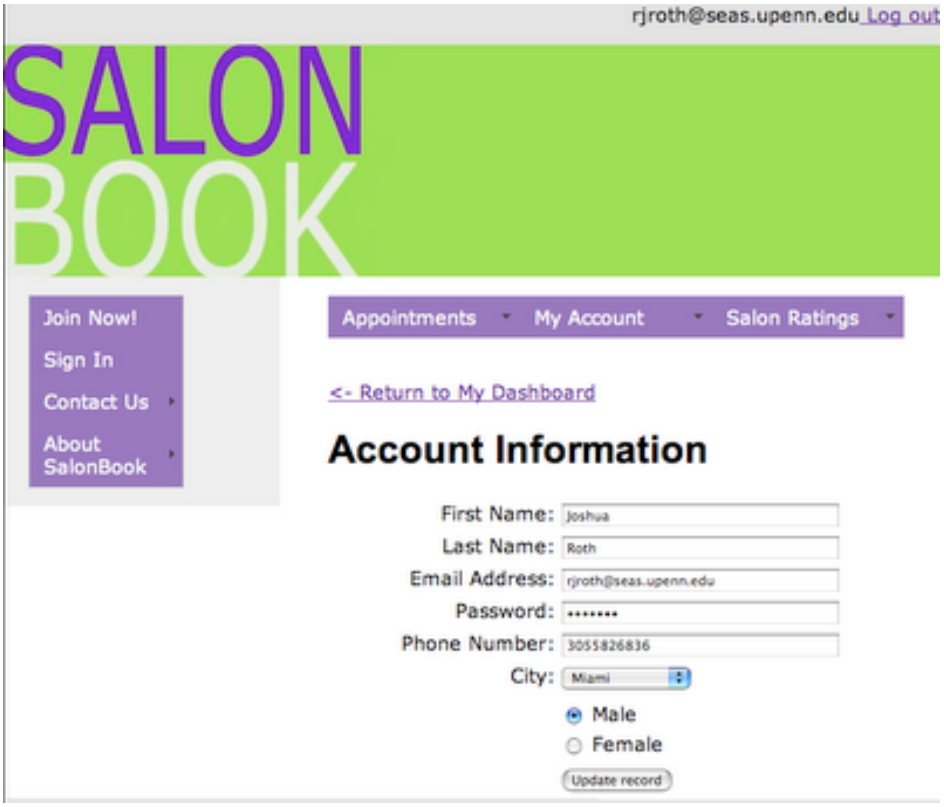
The following use cases are now functional: User login/logout, user create account, user view/edit account information, salon search (with filters by name, city, or zip code), promotions search (with filters by salon name, salon city, salon zip, and by beauty or hair services being promoted/discounted), viewing user point total, salon login/logout, salon create account, salon view/edit account, salon add new stylist, salon remove stylist, salon add/remove services offered by each stylist, salon add/remove beauty services that they offer, salon create promotions, salon view promotions, salon edit promotions (promotion note, start date, expiration date, and services applicable), and salon view customer ratings of stylists at their salon.

Additionally, I've spent time editing the CSS of the page layout and the spry menu bars in order to get a better look and feel, even though the design is still fairly basic, as I have spent much more time wrestling with PHP and SQL queries trying to build the functionality. Here are some screenshots of the interface:

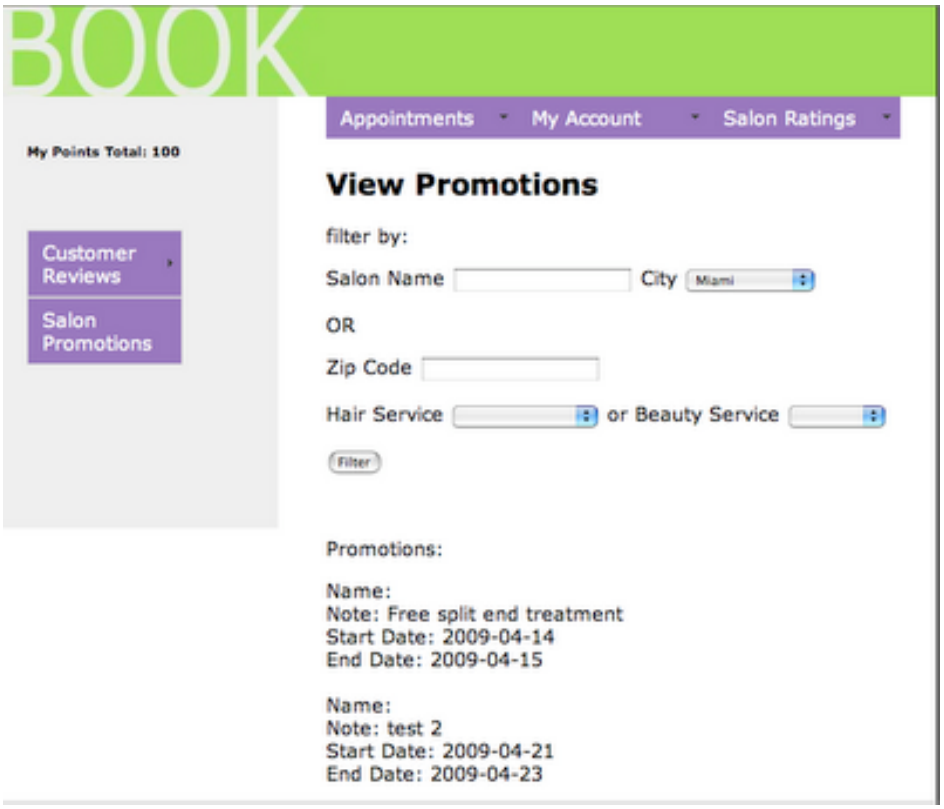
User Dashboard:



View/Edit Account Info:




Promotions Search/Filter:



One interesting problem that arose was the way to retrieve and pass information from the database to and from the client end. I designed my database to be flexible in the salon-stylist relationships it could represent. Essentially, my database is designed to allow for the possibility that a stylist belongs to multiple salons, AND that they offer a different subset of services at each salon. I did this by having a many-to-one relationship table of salon/stylist/service so that while there will be multiple records for each salon, stylist, or service, each combination of stylist-salon-service is unique. While this is great for richness of information, it makes retrieval of information much more difficult. In order to allow salons to add/edit stylists and services they offer, I would have to be looking up each stylist and each service for that salon. The solution was this:

```
$colname_Stylist = "-1";
if (isset($_GET['id'])) {
  $colname_Stylist = $_GET['id'];
}
//find stylist in database from URL variable passed through link on previous page
mysql_select_db($database_test, $test);
$query_Stylist = sprintf("SELECT * FROM stylist WHERE id = %s", GetSQLValueString($colname_Stylist, "int"));
$Stylist = mysql_query($query_Stylist, $test) or die(mysql_error());
$row_Stylist = mysql_fetch_assoc($Stylist);
$totalRows_Stylist = mysql_num_rows($Stylist);


//check to see if stylist currently offers women's haircut
$colname_Services = "-1";
if (isset($_GET['id'])) {
  $colname_Services = $_GET['id'];
}
$womensCut = 1;
mysql_select_db($database_test, $test);
```

```
$query_Services = sprintf("SELECT * FROM salon_stylist_h_service WHERE
`stylist id` = %s AND `salon id` = %s AND `hair service id` = %s" ,
GetSQLValueString($colname_Services, "int"),  GetSQLValueString($salonID,
"int"),   GetSQLValueString($womensCut, "int"));
$Services = mysql_query($query_Services, $test) or die(mysql_error());
$row_Services = mysql_fetch_assoc($Services);
$totalRows_Services = mysql_num_rows($Services);

//check to see if stylist currently offers mens cut
$mensCut = 2;
mysql_select_db($database_test, $test);
$query_Services2 = sprintf("SELECT * FROM salon_stylist_h_service WHERE
`stylist id` = %s AND `salon id` = %s AND `hair service id` = %s" ,
GetSQLValueString($colname_Services, "int"),
 GetSQLValueString($row_Salon['id'], "int"),   GetSQLValueString($mensCut,
"int"));
$Services2 = mysql_query($query_Services2, $test) or die(mysql_error());
$row_Services2 = mysql_fetch_assoc($Services2);
$totalRows_Services2 = mysql_num_rows($Services2);



//delete womens haircut script
$womensCut=1;
  $deleteSQL = sprintf("DELETE FROM salon_stylist_h_service WHERE `stylist
id`=%s AND `salon id`=%s AND `hair service id`=%s",
                       GetSQLValueString($_GET['id'], "int"),
GetSQLValueString($salonID, "int"), GetSQLValueString($womensCut, "int"));

  mysql_select_db($database_test, $test);
  $Result1 = mysql_query($deleteSQL, $test) or die(mysql_error());



//delete mens haircut script

$mensCut=2;
```

```
  $deleteSQL2 = sprintf("DELETE FROM salon_stylist_h_service WHERE `stylist
id`=%s AND `salon id`=%s AND `hair service id`=%s",
                       GetSQLValueString($_GET['id'], "int"),
GetSQLValueString($salonID, "int"), GetSQLValueString($mensCut, "int"));

  mysql_select_db($database_test, $test);
  $Result2 = mysql_query($deleteSQL2, $test) or die(mysql_error());

//insert script

//insert womens cut if checked
 if ($_POST["Services1"] == 1) {
 $insertSQL = sprintf("INSERT INTO salon_stylist_h_service (`salon id`, `stylist
id`, `hair service id`) VALUES (%s, %s, %s)",
                       GetSQLValueString($salonID, "int"),
                       GetSQLValueString($_GET['id'], "int"),
                      // GetSQLValueString(isset($_POST['Services1']) ? "true" : "",
"defined","1","0"));
       GetSQLValueString($womensCut, "int"));

  mysql_select_db($database_test, $test);
  $Result1 = mysql_query($insertSQL, $test) or die(mysql_error());
 }
//insert mens cut if checked
if ($_POST["Services2"] == 2) {
$insertSQL2 = sprintf("INSERT INTO salon_stylist_h_service (`salon id`, `stylist
id`, `hair service id`) VALUES (%s, %s, %s)",
                       GetSQLValueString($salonID, "int"),
                       GetSQLValueString($_GET['id'], "int"),
                      // GetSQLValueString(isset($_POST['Services2']) ? "true" : "",
"defined","1","0"));
      GetSQLValueString($mensCut, "int"));

  mysql_select_db($database_test, $test);
  $Result2 = mysql_query($insertSQL2, $test) or die(mysql_error());
}
```

```
//go to:
$deleteGoTo = "Salon_Dashboard.php";
  if (isset($_SERVER['QUERY_STRING'])) {
    $deleteGoTo .= (strpos($deleteGoTo, '?')) ? "&" : "?";
    $deleteGoTo .= $_SERVER['QUERY_STRING'];
  }
  header(sprintf("Location: %s", $deleteGoTo));
```

Basically, what this script does is for the selected stylist (and logged in salon), populate the forms on the screen to the current state of the world. Then, once the form is submitted, delete all entries in the salon/stylist/service table. Then add new entries into this table for the stylist and salon for the selected services. It is a nice solution because it adds and deletes in batches, thus keeping related information together in the data tables as well, rather than checking and adding/removing each separately.

User viewing and filtering Promotions by Salon name, city, or zip code was also challenging. I could create a recordset that would pull all salons that matched the criteria of the name, city, and zip code fields. I could also pull promotions from the database that matched the beauty/hair service filter. The problem was integrating the salon filter to the promotions filter. The promotions table only stores the salon id of the salon that created a given promotion. So I could not use the filters as direct search criteria into the promotions table. I had to use embedded do-while loops and a new array to store the results in order to later print to screen. Below is most of the code that populates the array with the proper promotions:

```
$colname_city = "-1";
if (isset($_POST['city'])) {
  $colname_city = $_POST['city'];
}
$colname_name = "-1";
if (isset($_POST['name'])) {
  $colname_name = $_POST['name'];
}
```

```
$colname_zip = "-1";
if (isset($_POST['zip'])) {
  $colname_zip = $_POST['zip'];
}
$name="Josh's Salon";
mysql_select_db($database_test, $test);
$query_Salons = sprintf("SELECT * FROM salon WHERE `zip code` = %s OR
(`city id` = %s AND name LIKE %s) ORDER BY name DESC",
GetSQLValueString($colname_zip, "int"), GetSQLValueString($colname_city,
"int"), GetSQLValueString("%" . $colname_name . "%", "text"));
$Salons = mysql_query($query_Salons, $test) or die(mysql_error());
$row_Salons = mysql_fetch_assoc($Salons);
$totalRows_Salons = mysql_num_rows($Salons);

mysql_select_db($database_test, $test);
$query_HairServices = "SELECT * FROM `hair service`";
$HairServices = mysql_query($query_HairServices, $test) or
die(mysql_error());
$row_HairServices = mysql_fetch_assoc($HairServices);
$totalRows_HairServices = mysql_num_rows($HairServices);

mysql_select_db($database_test, $test);
$query_BeautyServices = "SELECT * FROM `beauty service`";
$BeautyServices = mysql_query($query_BeautyServices, $test) or
die(mysql_error());
$row_BeautyServices = mysql_fetch_assoc($BeautyServices);
$totalRows_BeautyServices = mysql_num_rows($BeautyServices);

$colname_beauty = "-1";
if (isset($_POST['beautyservice'])) {
  $colname_beauty = $_POST['beautyservice'];
}
$colname_hair = "-1";
if (isset($_POST['hairservice'])) {
  $colname_hair = $_POST['hairservice'];
}
```

```
$date = date(y-m-d);
$promos = array();
$i=0;
do{
$thisID = $row_Salons['id'];
//$thisID = 2;
mysql_select_db($database_test, $test);
if($colname_hair == 0 && $colname_beauty == 0){
 $query_Promotions = sprintf("SELECT * FROM Promotions WHERE `salon id` =
%s ORDER BY `end date` ASC", GetSQLValueString($thisID, "int"));
}
if($colname_hair != 0 && $colname_beauty != 0){
$query_Promotions = sprintf("SELECT * FROM Promotions WHERE `salon id` =
%s AND (`beauty service id` = %s  OR `hair service id` = %s) ORDER BY `end
date` ASC", GetSQLValueString($thisID, "int"),
GetSQLValueString($colname_beauty, "int"),
GetSQLValueString($colname_hair, "int"));
}
if($colname_hair != 0 && $colname_beauty == 0){
 $query_Promotions = sprintf("SELECT * FROM Promotions WHERE `salon id` =
%s AND `hair service id` = %s ORDER BY `end date` ASC",
GetSQLValueString($thisID, "int"), GetSQLValueString($colname_hair, "int"));
}
if($colname_hair == 0 && $colname_beauty != 0){
 $query_Promotions = sprintf("SELECT * FROM Promotions WHERE `salon id` =
%s AND `beauty service id` = %s ORDER BY `end date` ASC",
GetSQLValueString($thisID, "int"), GetSQLValueString($colname_beauty,
"int"));
}

$Promotions = mysql_query($query_Promotions, $test) or die(mysql_error());
$row_Promotions = mysql_fetch_assoc($Promotions);
$totalRows_Promotions = mysql_num_rows($Promotions);
do{
if($totalRows_Promotions > 0){
```

```
$promos[$i] = $row_Promotions;
$i++;
}
        } while($row_Promotions = mysql_fetch_assoc($Promotions));
              } while($row_Salons = mysql_fetch_assoc($Salons));
```

For each result in the salon recordset based on the name/city/zip filter, the script uses the salon id from each salon and queries the promotions table with the salon id, beauty service id, and hair service id. It's likely that this too will pull multiple matching promotions. So, now a second do-while loop puts each resulting promotion from this query into an array. Then the outside loop moves to the next resulting salon from the salon filters, uses that id and repeats the promotions search, and so on until the result sets are null.

The next big step is to add the functionality of actually scheduling appointments. I've been in a dialogue with Professor Nimeroff about the best way to do this. The goal is to use Google Calendars, but there are conceptual issues with authentication, calendar ownership, and limitations on embedded calendars that need to be worked out. The zend framework and gData libraries are very robust and allow me to create calendars, add events, remove events, query events, and set availability in a fairly straightforward manner. The problem is how and who to authenticate into google (behind the scenes) and also how to enable more than read-only embedded calendars so that the Google Calendars interface can be utilized.

The difficult, important questions that I am currently working through for the scheduling aspect include if/how to package multiple services into one appointment, how to assign appointments for hair and beauty when the stylist is not specified, and how to have appointment blocks pre-specified by the salons. Once I come up with solid answers to these fundamental questions, and after the rest of the UI is working, I will begin implementing Google calendars as a back-end appointment database and pulling that information forward on the client side of my site.

Posted by Josh Roth
at 8:16 AM

## 8. RESULTS

The result of this project was a successful high-fidelity prototype of a Salon management and scheduling application. I successfully implemented all of the above mentioned use cases, and achieved clean interfaces that further enhance the user experience.

The many-to-one database tables successfully represent a robust set of scenarios and relationships that are critical to salon management. WebCalendar was integrated well to provide salons with full calendaring functionality. The aesthetic customization of WebCalendar makes it undifferentiated from the SalonBook application.

For users, scheduling was the heart of the application. And indeed, users can currently select a salon, stylist, date, and service to make an appointment. They can then view or cancel appointments, and rate salons on past appointments. Additionally, I achieved the goal of making SalonBook a one-stop hub for salon-related information, including Salon search and contact information, peer reviews of nearby salons and stylists, and easy viewing of relevant promotions that salons are advertising. Together, all of this functionality gives users an unmatched value proposition for booking salon appointments. Users also accumulate points after attending and rating an appointment, which, upon launch of the site, would be redeemable for salon credit. This is an additional source of incentive to use this application.

For more results, see video screen captures of user and salon interfaces in action.

## 9. CONTRIBUTIONS

The contributions of this project to my knowledge base are extensive. Before starting, I had no previous experience with web development or any of the tools used for this application.

Database Design is an invaluable aspect of the process that I mastered in the early stages of the project. Building many-to-one relationships and robust databases were required in developing SalonBook. Familiarizing myself with SQL and database languages is also very useful going forward.

I also take away the combined understanding of PHP, MYSQL, and HTML, and how they come together to create a rich, database-backed web application.

Finally, web applications can become more than the sum of their parts with the successful integration of open source tools. Under limited time and resources, there is only so much one person can develop. However, by identifying appropriate open source projects and integrating them seamlessly, an application can become much more extensive and functional. Integrating WebCalendar, and widgets like the JavaScript date picker are perfect examples of this. If integrated poorly, these tools are obvious and detract from the user experience. Fortunately, I was able to integrate these tools well and package SalonBook as more than the sum of its parts.

In terms of contributions to the field, this prototype stands as a proof of concept for the idea of web-based appointment scheduling of any kind. Doctors' offices, dentist offices, salons, and many other industries are in great need of online, on demand scheduling. This project proves that it is very possible to build this functionality with the above-mentioned tools at very low cost. All tools used were open source, except Dreamweaver.

## 10. FUTURE DIRECTION

Like I mentioned earlier, full integration of open source tools is critical in creating a seamless overall application. There are some ways in which WebCalendar was not integrated which I would like to pursue given more time.

The first is to automate WebCalendar account creation and login when those tasks are completed for the SalonBook application. This would require inserting a new Salon user into the appropriate WebCalendar database tables when a salon creates a SalonBook account. Of course, this would also ensure that the WebCalendar account information matches their SalonBook information. The same for login would be an important improvement. As it exists now, salons have to log in to the WebCalendar application additionally within the SalonBook system when they navigate to it via the Dashboard. Separate sign in is only required once per session though, as the login is stored throughout a session even as they go back and forth from WebCalendar to SalonBook. Even so, automated account creation and login are important integration features that would be next in line, given more time.

Also, a constraint of WebCalendar was that it only allowed a user to have one calendar. Ideally, a salon would have one calendar for each stylist. This could be achieved using an automated WebCalendar account creation script with a well-defined prefix naming system. Salons would be oblivious to what is actually going on behind the scenes, and

simply be able to navigate to calendars that are tied to each stylist. When a salon would add a stylist, another WebCalendar account (and corresponding calendar) would be created, using a prefix of the stylist name and salon name. This would dramatically increase the power of the application as a schedule visualization and distribution tool. Availability would be more apparent, and schedules could be printed for each stylist at the start of the day.

Additionally, I would like to host the prototype online and run usability tests with surveys to gauge the effectiveness of the interface and functionality. This is a key step to perfecting the front-end interface and the functionality of the tools.

Finally, I would like to explore more dynamic, Web 2.0 tools like JavaScript, jQuery, and OpenLaszlo in more depth with the hope of making the interface richer for the user. Over the course of this project, I was able to get a small taste of these web development tools and their capabilities. They would allow me to give my application a more desktop software-feel on the web, with features like animation and drag-and drop. Unfortunately, I did not have enough time to fully explore them, and so that would be an addition improvement to consider going forward.