Project Snordgrüv:

a work-order system for humans

Team members:

Stefan van Bodegraven
Curtis LayCraft
Tiffany Tang
Matthew Zorkin

Comp199 Final Report

Stefan van Bodegraven, Curtis LayCraft, Tiffany Tang, Matthew Zorkin

Project Specifications

Summary

This application was built to help a user who is operating a small service-industry business manage day-to-day tasks through a simple, elegant interface. It stores information about customers in profiles that can be retrieved and updated, a list of jobs and prices which can be added to work orders, as well as details about work orders such as due date, payment/pickup status, and notes. Payment tracking, integrated credit card processing, and partial payments are built-in. The user also has access to an admin area which allows retrieval of payment or ticket records with the option of filtering by customer or date.

Functional requirements

- User can store customer information in profiles
- User can create and store order tickets with one or more items and associated tasks
- User can set predefined tasks with standard time estimates and prices
- User can check off completed order tickets from a job queue
- User can close an order by marking it as “picked up”
- User can apply a payment or partial payment to a job using a payment processor or cash
- User can look up a customer and their previously completed jobs
- User can print simple reports and perform daily closing activities

Non-functional requirements

- Run platform independent
- Work in both portrait and landscape resolutions
- Work in a browser
- Use MySQL for tracking data
- Use PHP for assembling a HTML UI
- Use CSS to style the UI
- Use JavaScript and JQuery UI elements to make user friendly UX
- The system has to perform actions fast enough that the user doesn't have significant delays

Usability Guide

(The user manual is attached to back page of this document.)

Problems and Challenges

A lack of proper documentation was a key issue early in development. As the code-base grew, team members found it difficult to pick up development in areas where another had begun work. We paused development during week 7 to comment our code. Moving forward, collaboration was smoother.

Initially, with each team member working on a different module, interaction with the database was accomplished through individually coded PHP scripts with tailored SQL statements. Though this method was quick to get working, as the number of scripts grew the structure of the application became disorganized and fragmented. In week 8 we restructured the app and unified all database connectivity into a single file, controller.php. This file provided an access point to query or submit data to the database. It communicated with the originating Javascript calls using JSON objects and selected which statements to execute based on a “flag” member in the incoming JSON. With all of the code in one place, identifying an existing function which could be repurposed for a new task was easy, and adding a new use case was simple as creating a new flag and making changes to existing code to fit the new purpose.

Another major source of delays was using HTML/CSS and the box model to design what was to essentially mimic a desktop app. Though we did accomplish what we set out to do, fiddling with floats, alignment, and nested elements was a huge source of frustration for our front-end developers.

We had some problems with merge conflicts. We resolved this by keeping each other updated on what we were working on.

What Was Learned?

Our team’s familiarity with Javascript and JQuery increased significantly through the course of the project. Included in the technical skills we learned were:

- How to handle the asynchronous nature of AJAX calls
- Avoid AJAX if possible
- Use of JQuery UI widgets: Datepicker, Timepicker, Autocomplete
- Dynamic production of HTML elements based on data from a MySQL database.
- When to use Javascript vs JQuery

Some general lessons we encounter were:

- Leave visual styling and implementation of eye-candy until the end because they can get in the way
- Comment all code right away
- Do not get attached to your code: Sometimes things need to be thrown out
- Commit regularly
- Test thoroughly before pushing
- Time spent planning out the design of an application in the beginning can significantly cut down on development difficulty later on.

We learned the value of spending time to outline and combine systems that perform similar tasks. When the project started we spend less time planning than we did on coding particular tasks that we wanted do. As the project progressed we discovered that spending the time to outline in each of the functional areas and like tasks in the application helped the long-term cohesion. Planning helped us reduce the complexity of the coupling in the app and allowed us to focus on particular tasks within the framework that we outlined. Half way through the project we had to restructure a lot of the code because there were complications between the implementations that we had already built. The finished project has highly modularized code and adding new features is not going to cause conflicts between different parts of the application.

Instead of having user interface elements communicating directly with our database PHP file, we eventually made a controller file that received JSON arrays are routed to the correct functions to communicate with the database access PHP file. The database access then has a reduced set of methods that perform actions on the database. The effects of this change were twofold: increasing security by reducing complexity.

We learned the value of commenting code early in the project. Because after working with your own block of code for a short period of time you become intimately acquainted with how it works in combination with other code. Documentation helps to simplify the readability to other developers who are also working on implementing similar functions into the software. Commenting the code is also very beneficial for long-term maintainability; it was sometimes difficult for a code creator to interpret what was written even a week after it was written.

Because we had minimal experience with the client-server model it was very difficult for us to determine what should be executed locally on the user's machine and what should be executed on the server. We had to learn what asynchronous meant and the implications of asynchronous programming. The client server model comes with latency in the communication between the client machine and the server, and your code would keep executing even if the response hadn’t been received yet. What seems like simple programming tasks, like refreshing a page of tickets, is complicated by the system executing server requests and handling user interactions simultaneously.

When we started building the user interface in JavaScript we didn't realize that JavaScript runs from a single threaded main loop and that it doesn't accommodate multithreading. Timing UI events in AJAX is tricky because the client sends data but does not wait for the ajax request to complete before continuing with the code. In the case where a user clicks three checkboxes one after the other, there shouldn't be a server request for each of the checkboxes; Each click from the user should start a short timer, and each subsequent click should reset the timer so that all the clicks can be handled by one larger Ajax request and all the actions can be handled remotely.

What Would We do Differently?

If given the task to create all of the functional requirements of this project again, the primary change would be abandoning the client-server model.

Our database is accessed remotely. It doesn't make sense for a business to lose connection to their work order management system when they have issues with their network connection. We would replace MySQL with SQLite and rely on copying the entire SQLite database file for backups instead of building a set of tables for each client so that we would have limited responsibility for collecting our customers data and can perform quick and portable database duplication.

Since we would abandon a client-server model, the PHP that is communicating with the database should be replaced with something that doesn't require Apache to interpret. A language like Java is better suited to handle local database queries. This change would speed up the app significantly.

We would also aim to outline the essential functions that we wanted to implement more before deciding on non-functional requirements. Improved specifications and design documentation would give better guidelines for establishing the structure of the project and would help to make informed implementation decisions before beginning the work.

We would focus moreso on the object model early in the project development so that tasks can be split among team members more easily. It is easier for a team to contribute to individual components without understanding the entire workings of the system. One black-box function that performs its task in every situation is better than multiple similar functions. The object model drastically reduces the amount of redundancy in the system and can increase the optimization of code. Our finished project was based on the object model but our initial project was not; if we were to do this again we will start with the object model and get it done in half the time.

Build an app using a cross platform UI builder to cut down time programming conventional user interactions. JavaScript has weaker implementation of the observer pattern and having a language that uses listeners to handle events could significantly simplify the user interactions we intended to make. There're a number of user interface building tools like QT and Unity that create cross platform compatible UI and consistent UX without the need to micromanage user interactions. One of the three prime candidates for accomplishing cross-platform functionality with minimal modification to the software is QT. QT is a cross platform application framework that works with C++, QML, and Python that would have been a very good fit. QT would have required us to learn C++ and QML so it would be a steep learning curve. Another cross-platform application framework that we can use is Unity, an engine that Matthew is very familiar with. With Unity there is an application engine overhead that may be overkill for a simple database driven application. The other candidate for interface building would have been Java Swing which we are all familiar with. Java Swing makes apps that tend to feel like non-native apps. The interface builder can also be cumbersome and tricky to use and is riddled with quirks.

If we had to implement the project using web-technologies again as non-functional requirements, we would start the project as a series of self contained web-forms instead of a monolithic structure. Having individual forms helps to compartmentalize and assist in implementing core functionality before implementing style elements and user interactions. When core methods have been established, the components would be modified and assembled into a unified web project. This application would be a good candidate for REST programming.

Code Samples

This function is called when you check or uncheck a task on an open ticket. The database is updated to contain the current status of the check-box. To prevent errors when double clicking a task, the check-box is disabled until the response is received. Upon success it also checks if all of the tasks have been checked off. If they are all checked and it is marked as “open”, the ticket list is refreshed so that the ticket is moved to the “Completed Tickets” section and vice versa.

```
1.  // Occurs when a ticket checkbox is clicked
2.  function checkEvent(ticketno, jobno) {
3.
4.          // Get the checkbox object and disable it
5.          var checkbox = document.getElementById(ticketno+"-"+jobno);
6.          checkbox.disabled = true;
7.
8.          // Create an array with the ticketno, jobno, and checked status
9.          var complete = (checkbox.checked==true) ? 1 : 0;
10.         var formArray = {0:'taskcheck', 1: ticketno, 2: jobno, 3: complete};
11.
12.         // Send the array to the controller
13.         $.post("php/controller.php", formArray, function(data) {
14.
15.                 // Check if the status has changed
16.                 var oldStatus = getStatus(ticketno);
17.                 var status = checkStatus(ticketno);
18.
19.                 if (status == oldStatus) {
20.                         // Re-enable checkbox
21.                         checkbox.disabled = false;
22.                  } else {
23.                         // Refresh the ticket list if the ticket needs to move to or
    from completed tickets
24.                         startLoading();
25.                         refresh();
26.                 }
27.         });
28. }
```

Each part of the data collection in the UI is done through a consistent collection pattern and adding new fields is as simple as adding a new variable into fieldIDs. The data is handled through controller.php which handles json arrays by case. The json arrays are standardized and adding new cases to controller.php is straightforward. The initial letter of the of the fieldID indicates which form the information is gathered from.

```
1.      // (javascript) place to assemble the KV pairs from the form
2.      var formData = {};
3.      // holds a ticket number if available
4.      var custNum = document.getElementById('ccustno').value;
5.      // the name of the html id tags to grab and turn into kv pairs
6.      var fieldIDs = [ 'cphone', 'cfirstname', 'clastname',
7.          'cext', 'caddress', 'cemail', 'czip'];
8.      // for each field, save a key value pair into formData
9.      for (i=0; i < fieldIDs.length; ++i) {
10.     // collect the value of the field
11.     var myKey = fieldIDs[i].substring(1);
12.     var myValue = document.getElementById(fieldIDs[i]).value;
13.     // add key value pair to form data
14.     formData[myKey] = myValue;
```

We have one extensible clean script that we can modify to boost DB security, filter text input, and fix text input quirks. The code is not as important as what this little module offers: room for expansion.

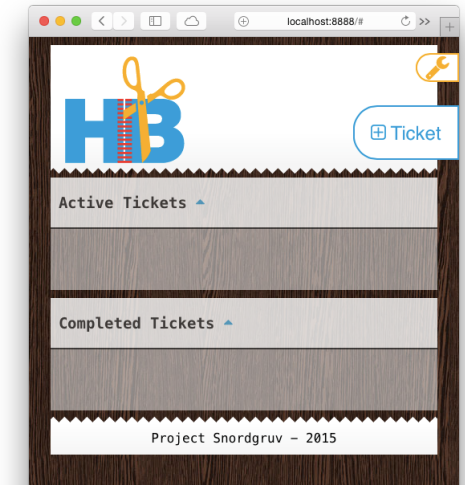
```
1.  <?php
2.  // Sanitization script     // Removes: <, >, \\, ;, /*, */
3.  // Also escapes apostrophes to allow their use in text fields.
4.
5.          function clean($data)
6.          {
7.                  $cleaned = trim(strip_tags($data));
8.                  $Regex = array('(<|>)','(\/\/)','(\;)','(\/\*)','(\*\/)','/\'/');
9.                  $replacement = array('','','','','\\\'');
10.                 $cleaned = preg_replace($Regex, $replacement, $cleaned);
11.
12.                 return $cleaned;
13.         }
14. ?>
```

Project Snordgrüv:

User Manual

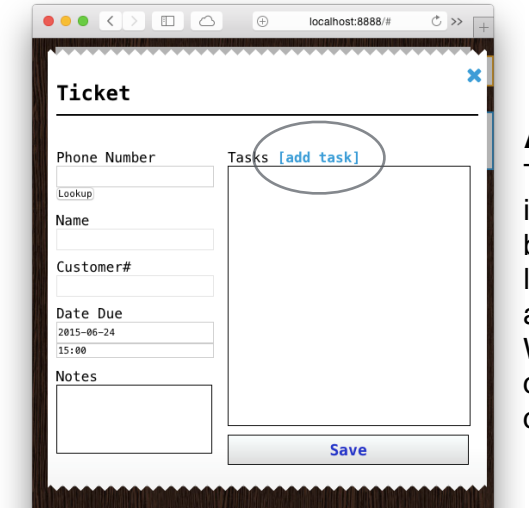
Team members:

Stefan van Bodegraven
Curtis LayCraft
Tiffany Tang
Matthew Zorkin



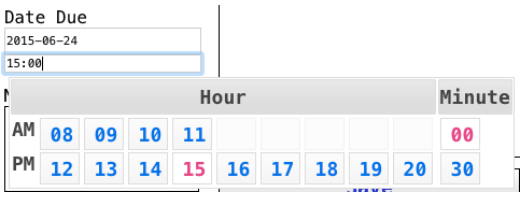
The workspace:

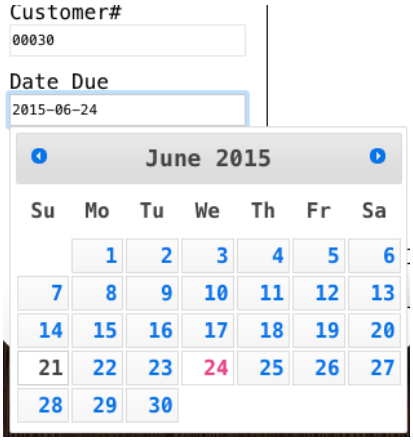
This is the main part of the app, it contains an active and completed ticket section. When all tasks have been checked-off of an active ticket, the ticket will move to the completed tickets area where the ticket is staged for payment or pickup. To modify tasks and see reports click the wrench in the top right hand corner. The active or completed tickets can be collapsed down to a stub by clicking the minimize arrow located beside the text "active tickets" or "completed tickets."



Add or edit a ticket:

This is the window for adding or editing a customer's information. To add a new ticket, click on the +ticket button. To edit an existing ticket, click on the edit ticket link on the ticket you wish to edit. Tasks and notes about service requirements are added to this window. When all information is as it should be, click save. To cancel out of the ticket window, click the "X" in the top corner or alternately click outside of the window.



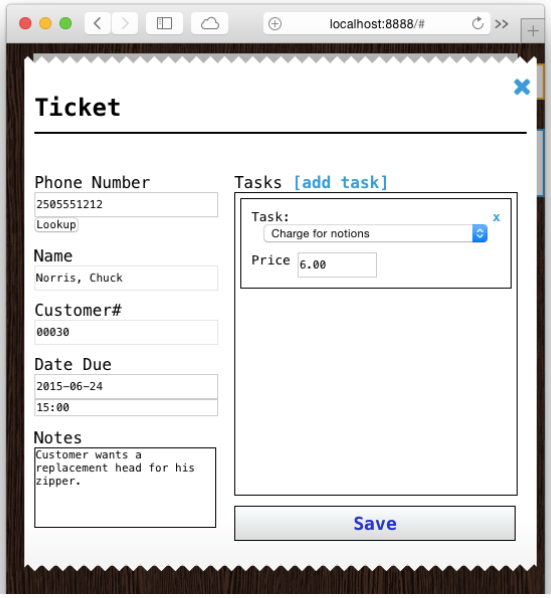


Set a due date:

A date picker and a time picker are available so that the ticket has a due date. This due date is used to prioritize active tickets. Tickets that are due soonest are displayed first.

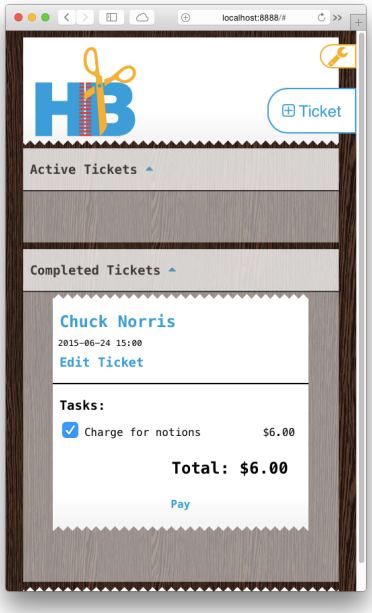
Add tasks to a ticket:

To add a task to the customers ticket, click add task. This will start a new task in the task list. Each task can have the price customized. To add a repeat task, add it through the settings page.

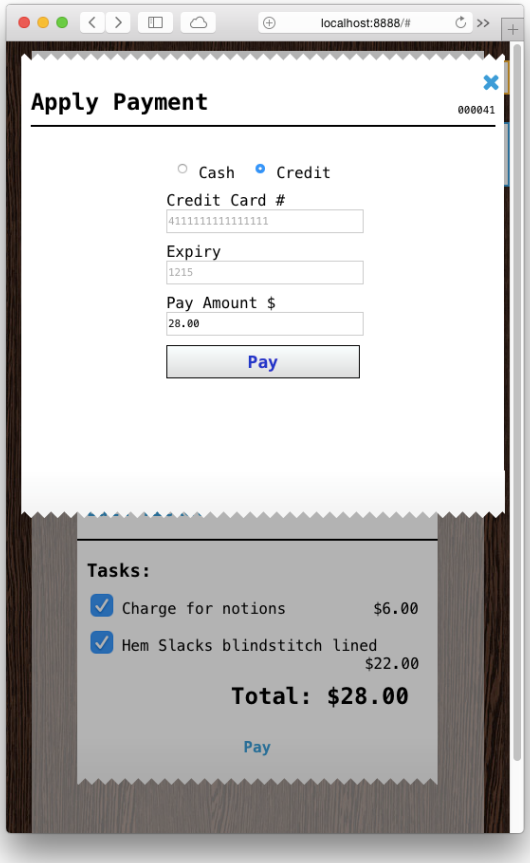


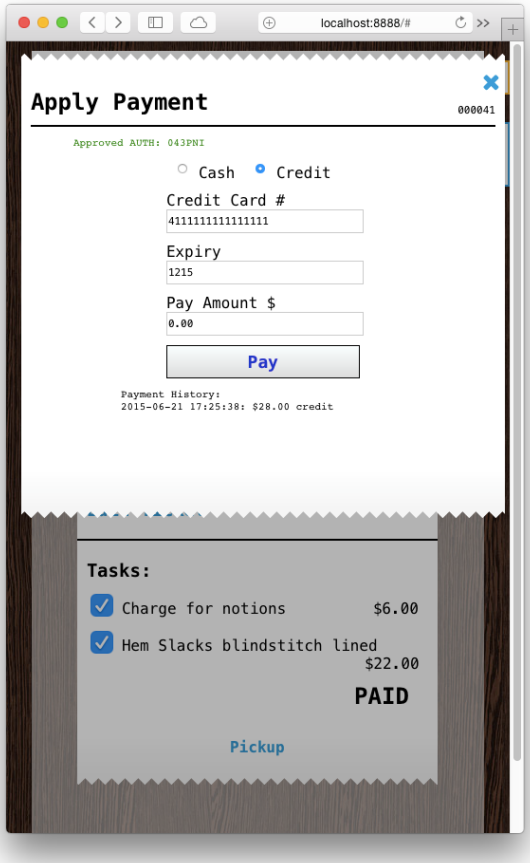
Changing tickets:

A ticket in the active tickets area has a variety of actions that the user can take to modify, pay, or check off tasks on the ticket. Clicking on the name of the customer allows a user to edit that customers' information. Clicking the "edit ticket" button under the due date allows changing tasks and other information on the ticket.



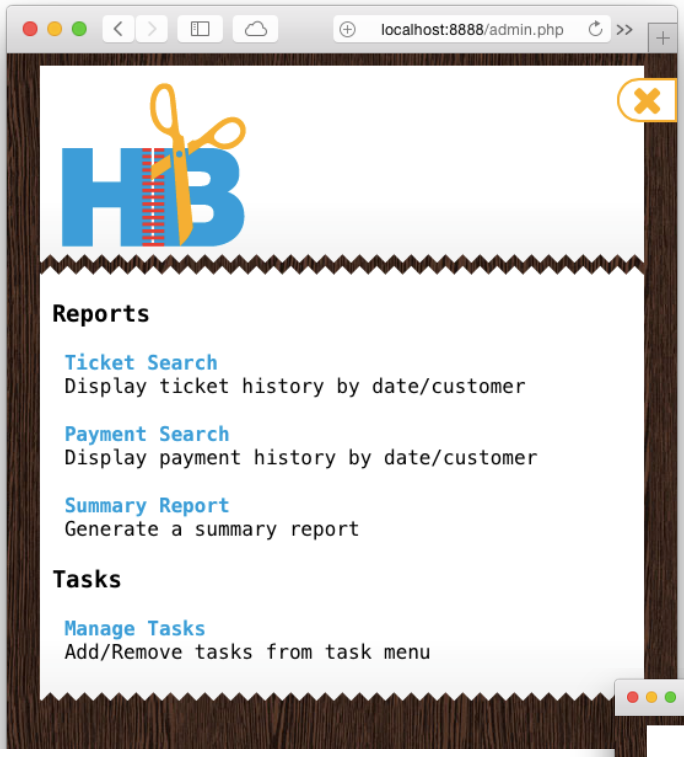
When an active ticket has had all of its tasks checked off, it automatically moves down to the completed ticket area for pickup or payment. If another task is added onto that ticket it moves back up to the active tickets area. A ticket cannot be "picked up" from the active tickets area unless it has been paid in full. To apply a payment click the pay button at the bottom of the ticket. When the ticket has been paid in full the "payment" button changes to a "pick up" button.





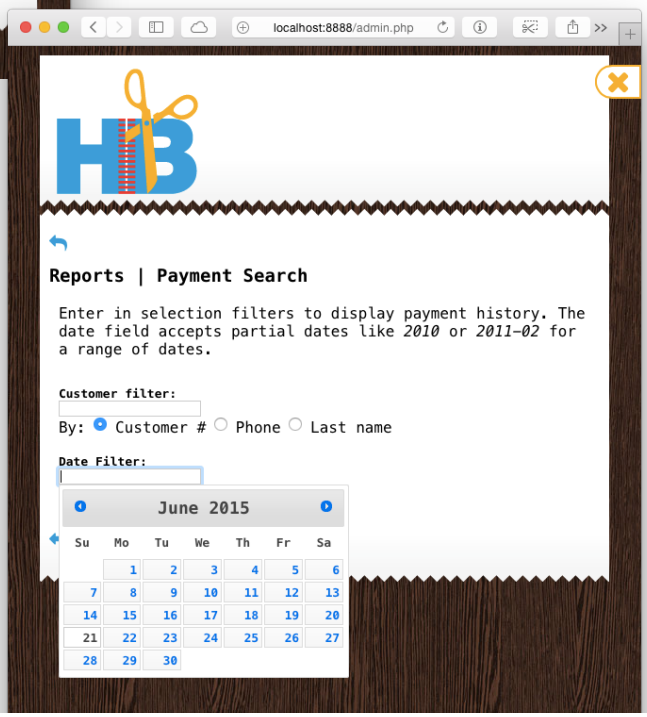
Accept a payment:

The payment window has the choice of cash or credit. If a payment for a partial payment is applied to the ticket the payment will show up below the pay button. A confirmation number for the most recent credit-card payment shows at the top of the window when the payment has been processed by the payment system.



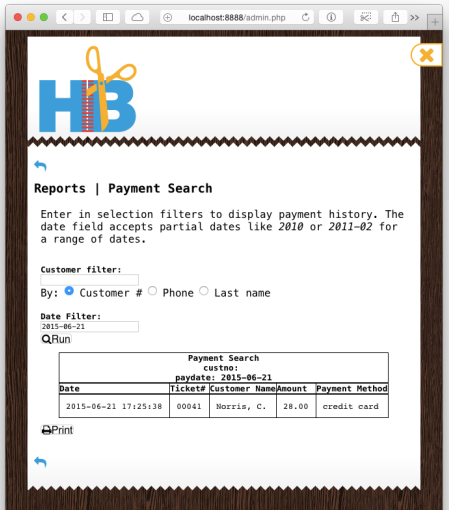
Business stuff:

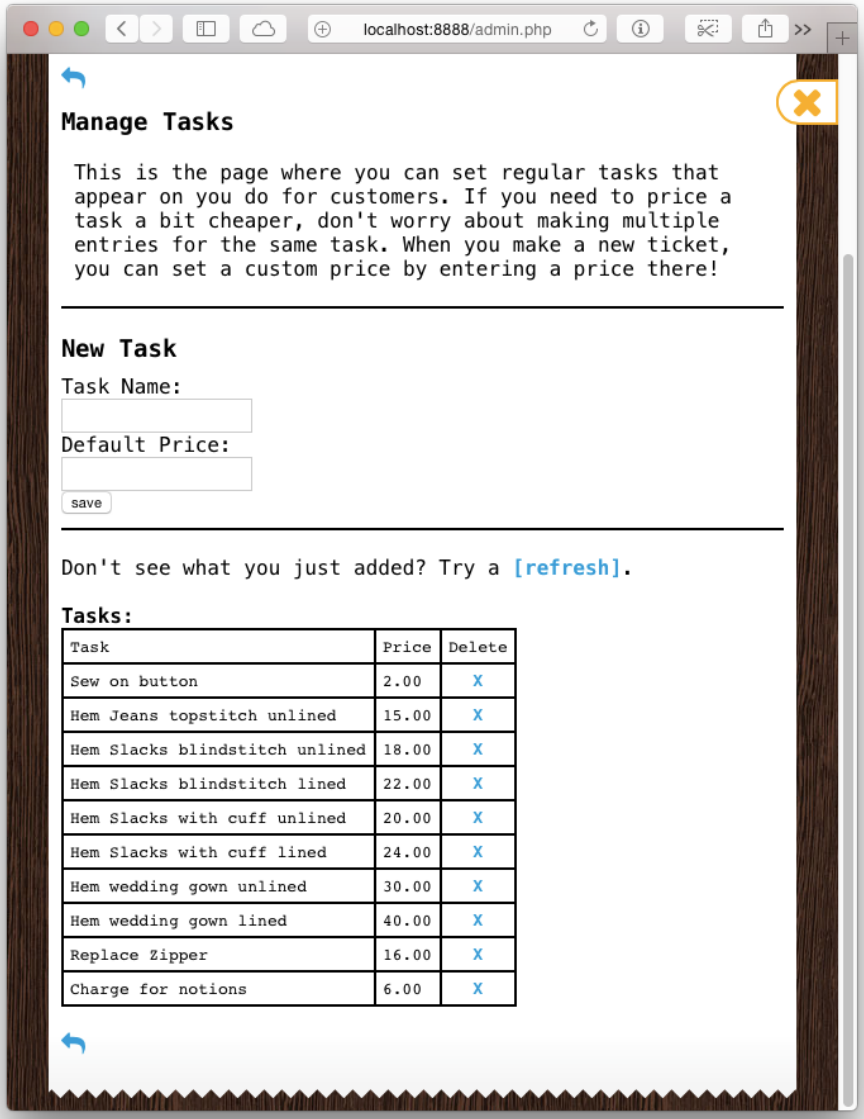
The settings area is used for viewing reports or modifying tasks. The ticket search allows you to search through closed and picked up tickets. The payment search allows you to search through payments that have been taken. Each of these search areas allows you to search by customer and date.



Search previous tickets/payments:

Select to search by customer number, phone number, or last name. Choose to narrow the search by choosing a date. (Either field can be omitted.) Click the print button to print out the generated report.





Manage Tasks:

In the manage tasks area, type a name for a task and set the default value. The new task and its default price should be appended to the list below immediately. You can delete any of the tasks in the list by clicking on the X beside the task. (Note: a task cannot be deleted if it associated with an open ticket.)