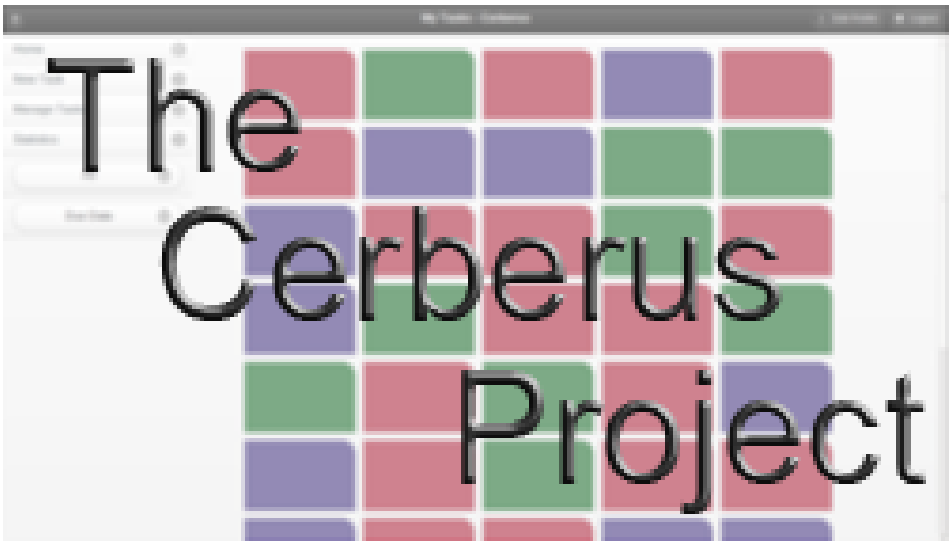


Oasis Digital Solutions Inc.

Summer 2012

Intern Team:
David McNeil, Sam Pepose, Sarah Whelan

Project Overview

Oasis Digital Solutions, Inc. is a software company that develops custom software for companies worldwide. Oasis Digital is primarily based in St. Louis, but has contributors spread throughout the United States. The success of the 2011 internship program led Oasis Digital to expand the program for the summer of 2012. This year the internship program has grown to include three interns in high school, or recently graduated and about to enter college. One of the reasons this program was started was to allow for the developers at Oasis Digital to learn about new technologies they normally would not get a chance to work with. There is a mutual benefit between the interns and the company, as the company will have new products to show existing customers by the end of the summer, while the interns gain a once-in-a-lifetime experience in the industry.

Goals

The primary objective of the Cerberus Project was to create a workflow management system with desktop, tablet, and mobile interfaces.

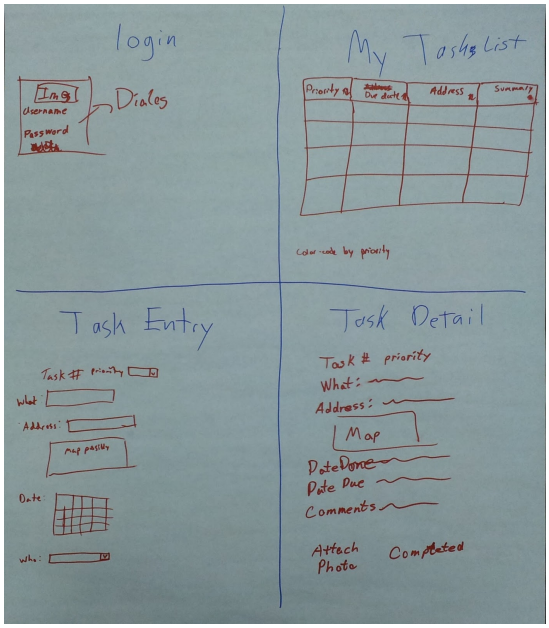
Team Members

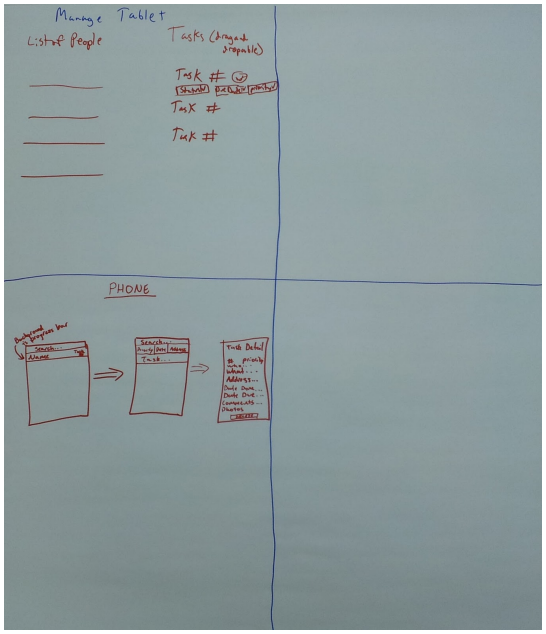
David McNeil is a freshman at Rose-Hulman Institute of Technology in Terre Haute, Indiana.
Sam Pepose is a senior at Whitfield High School in Saint Louis, Missouri.
Sarah Whelan is a senior at Whitfield High School in Saint Louis, Missouri.

Design Overview

The team discussed the global plans for the project on the first day. We discussed which languages and platforms to use that would best fit the project's goals. Node.js was decided upon for the server because of its asynchronous nature and flexibility. JavaScript was decided upon for the client because of its simplicity and gradual learning curve. The team decided that it was beneficial to have the same language for both the client and server.

We discussed the basic layout of the application. It was decided that we will need six different pages for our application: a register page, login page, tasklist page, task entry page, task detail page, manager page. Rough sketches of each page were drawn:





Tools

Node.js

Node.js is a server-side platform built on Chrome's V8 JavaScript engine. Programs for the platform are written in JavaScript, meaning the average programmer does not need to learn a new language. Rather than being multi-threaded, Node.js is single-threaded, which enables event-driven, asynchronous I/O. Most Node.js I/O functions are given callbacks as a parameter which are called when the I/O activity is complete. This allows the app to continue to run smoothly even through intensive I/O activities [1]. The team found the cons of Node.js to outweigh the pros. We decided that Node.js is a good choice for small-scale projects without a lot of server-database interaction. For larger projects, the amount of callback functions that are needed becomes foolish. A long chain of callbacks is made when the server needs to execute many functions linearly. This causes confusion while reading the code because of code-indentation and the sheer amount of callbacks. We had to nest four callbacks within each other in our project, and it was painful. Another problem with Node.js is that it can become difficult to think in an asynchronous manner for complicated processes. Unfortunately, many Node.js functions do not provide a synchronous option; asynchrony is forced upon the developer.

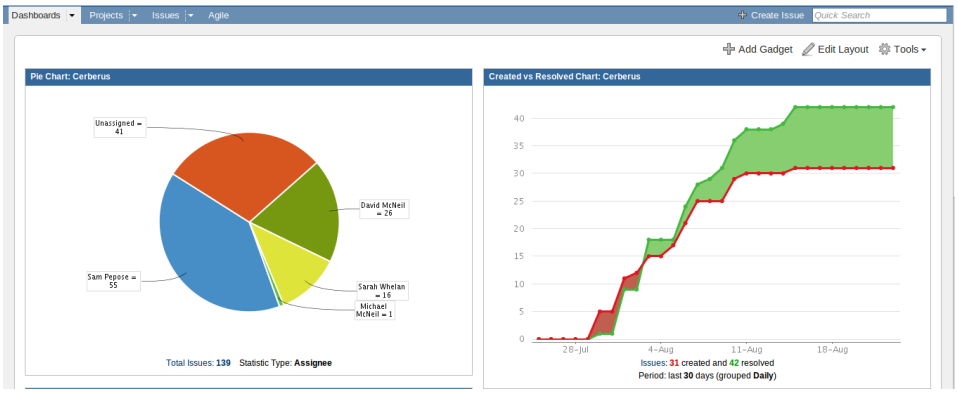
Node Package Manager (NPM)

Node Package Manager is a universal repository for Node.js modules (applications). NPM comes packaged with the Node.js installation. Modules are able to be imported and re-used in applications, creating a web of module dependencies. A JSON file, package.json, is used to record the modules that an application requires to function. NPM is able to download and install all dependencies for a specific module based on the contents of package.json. NPM is a wonderful tool as long as the package.json file is updated whenever a new package is added. A downside of NPM is that packages are not automatically added to the file when installed. The repository is hosted on Node's servers and is accessible via command line or their website [2].

JIRA

JIRA is an issue-tracker which made up a large part of the project management aspect of the project. JIRA allows bugs, new features, improvements, and tasks to be created and assigned to members of the development team. JIRA includes a customizable workflow which each issue needs to flow through before completion. This allows for in-depth progress tracking for each issue and the ability to know each developer's role in bringing an issue to completion. JIRA helped to organize daily tasks for the team and removed any chances of duplicate fixes. It provided an excellent interface for watching the progress and

history of our project. JIRA includes a user-customizable dashboard that is able to include graphs or other representations of data regarding the project [3].



The image above shows two examples of JIRA graphs. The first graph displays issue delegation, and the second graph displays whether the team is keeping up with the amount of open issues.

BeeBole

BeeBole is an online timesheet service for small and medium-sized businesses. BeeBole provides a week-by-week calendar for employees to enter the hours they worked per day [4]. Each week, the employee submits their timesheet for approval. A project manager or owner of the business then processes and approves the timesheet on a bi-monthly basis and sends a paycheck through a service called SurePayroll. The BeeBole dashboard offers customizable graphs displaying personalized information regarding your timesheet. These graphs include hours worked per project, hours per day, trends throughout the month, and many other options. Managers and owners have access to charts that show the total hours of the company per project which allows more efficient management of a business.

BitBucket

BitBucket is an hosting site for the distributed version control systems Git and Mercurial. While using Git,

BitBucket allowed the team to view each commit throughout the project and to see the differences made between each commit online. BitBucket is owned by Atlassian, the company that owns JIRA. Conveniently, this allows BitBucket and JIRA to interact. The team was able to simply reference the ID of a JIRA issue in a commit message, and a link was made between JIRA and the commit. The ID in the commit message becomes hyperlinked to the JIRA issue, and the JIRA issue has an option to view commit history for each issue [5].

Git

Git is a distributed version control system. In our project, it was specifically used as a source code management tool which allowed many people to collaborate on the same project. It was originally developed by Linus Torvalds for the Linux kernel which is a firm testament to the software's efficiency and stability [6]. Our basic use case for Git is as follows: developers are assigned specific coding tasks, the developer pulls the current source code (in our case it was stored on BitBucket), makes changes to complete their given task, commits that specific changeset, and pushes these changes to the central repository. Other team members are then able to pull these changes and continue working on their specific tasks. An important item to note here is that, the majority of the time, Git automatically handled merging conflicts between versions of code which significantly automated the collaboration process.

JavaScript

JavaScript is one of the most prolific programming languages used in the world of software. JavaScript (JS) has become the standard scripting language of the web, creating widespread support from all modern browsers. The language was originally developed by Brendan Eich for use in Netscape. Despite its name, it bears no direct correlation with Java, but was named as an attempt to share the hype of Java. It possesses many deficiencies such as a global namespace, numerous idiosyncrasies in predication, and internal qualms pertaining to the use of prototypical or traditional class inheritance. However, at its heart, it has the ability to posses an elegant functional style. In JS, functions have first order status, and variables are dynamically typed which allows for a declarative programming style.

Heroku

Heroku is a cloud platform as a service (PaaS). It provides hosting for applications written in Java, Node.js, Clojure, Python, Ruby, and more. We utilized Heroku to publicly host our project. A Git repository and domain name is assigned when you create a new Heroku project. Code is easily pushed to the repository, and Heroku handles the rest. It automatically downloads and compiles the necessary modules listed in the package.json file and then deploys the application to the server. Heroku provides free minimal hosting that includes a domain name with a Hiroko extension. There is an option to use a domain name that has already been purchased [8]. We took advantage of the free hosting for this project and kept a free account up-to-date with our source code. You can find the Cerberus project hosted on Heroku at http://oasisdigital-cerberus.herokuapp.com/. Heroku was very easy to set-up and

keep up-to-date with our current commits.

PostgreSQL

PostgreSQL is a relational database that uses the SQL database query language. The basic structure of a relational database is the table. A table consists of an arbitrary number of columns, each having a name and specified datatype to be stored within the corresponding cell. Users then inject rows into this table with values correlating to the various columns. The SQL protocol provides an intuitive interface for adding, manipulating, and retrieving information from within these tables [9]. In this project we had two tables: one to store the users and another for the tasks.

```
CREATE TABLE tasks (
        task_id          integer          PRIMARY KEY,
        user_id          integer          NOT NULL    DEFAULT 2,
        summary          varchar(100)     NOT NULL,
        description      varchar(255)     NULL,
        address          varchar(100)     NOT NULL,
        priority         integer          DEFAULT 2,
        due_date         timestamp with time zone        NOT NULL,
        finished_date    timestamp with time zone        NULL,
        comments         varchar(255)     NULL,
        workflow         integer          DEFAULT 1
);
```

Socket.io

Socket.io is a JavaScript library that provides the ability to update data between clients in realtime. There are two parts to socket.io: the server and the client. The server half of the library is a Node.js module. Socket.io is event-driven. Socket.io uses the WebSockets protocol to keep a constant connection open between the server and client. If WebSockets is unavailable in the clients browser, socket.io seamlessly falls back to using either Adobe Flash sockets or long polling. The code does not need to be changed to support all of these methods [10]. The team used socket.io to update task and user data in realtime throughout the application. If any attributes of a task or user are edited on one client, all other clients will see the change instantly. The server sends an event after it changes the data in the database, and all clients receive the event. Although we only send events from the server, socket.io allows sending events from the client and receiving events on the server [10]. For our application, it is up to the client to determine whether the event is relevant to itself. For instance, non-managers are unable to get to the manager page in the application. Therefore there would be no need to update user information for an instance of a non-manager client. We also used socket.io to update the number of open and in progress tasks for each user, the manager order of the tasks, and whether a user is logged in or not. Socket.io was very easy to implement and use. It is very easy to comprehend the event-driven core of socket.io

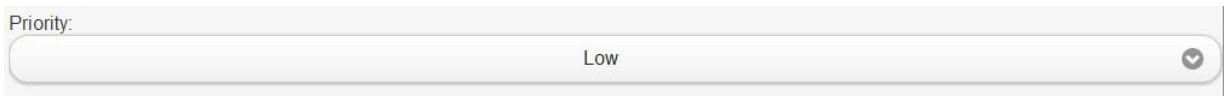
because it is made simple, and it closely represents the essence of the Node.js event system.

jQuery

jQuery is an extremely popular JavaScript library that aims to simplify complicated DOM manipulation with a very simple API. For example, instead of using getElementById() to obtain a DOM element, jQuery allows for the use of the jQuery convention $(selector) or the less conventional JQuery(selector). Selectors are typically by id, "#id," or by class, ".class." DOM elements are able to be manipulated by obtaining the DOM object, and then calling any one of various functions including .css() to change CSS of the element, .val() to change the value of the element, .html() to change the element content, .append() to append an element after the selected element, and .replaceWith() to replace the element with another. Other helpful features include binding, calling events, and setting animations on elements. Useful utilities such as each and isArray prove to be very useful. jQuery also has the ability to detect the user's browser and features backwards browser compatibility [11]. This library is also a requirement for other plugins we used including jQuery Mobile, jQuery UI, Mobile Datebox, jQuery UI Touch Punch, jQuery Password Strength Meter, jQuery Mobile SimpleDialog2, and Moment.js.

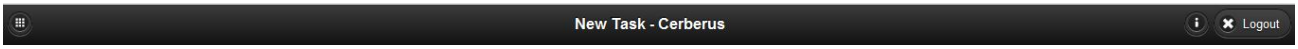
jQuery Mobile

jQuery Mobile is a plugin for jQuery that provides convenient and easily-implementable mobile-designed web content. It is touch-optimized and strives to make web content easily interactable for users with a touch device. JQM is compatible with all major mobile platforms as well as most desktop browsers. The developers of jQuery host a web-interface called Themeroller which allows the user to customize the colors and styling of the mobile HTML elements [12]. When JQM is included in an HTML page, all compatible HTML elements are automatically converted into mobile styling. JQM also provides additional components to developers by providing easy-to-use dialog boxes and pop-ups, although this is not easily compatible with Backbone.js. The framework provides a client-side routing system very similar to Backbone. For this project, we chose to use Backbone's routing system, but we could have converted the project over to use JQM's routing system. We felt that the styling of mobile elements was too forced in some situations and could be very hard to restyle differently. JQM does not simply add CSS classes to elements: sometimes it nests content in layers of divs, spans, and other elements. It can become very annoying to decipher how JQM formats an element, but it is necessary in order to customize the styling in any major way. Currently, the formatting horrors of jQuery Mobile seem to be a necessary evil. Besides those problems, JQM is a very fast and easily implementable tool to aid in the field of mobile web development. The following images display our usages of jQuery Mobile in the project.
Select box:

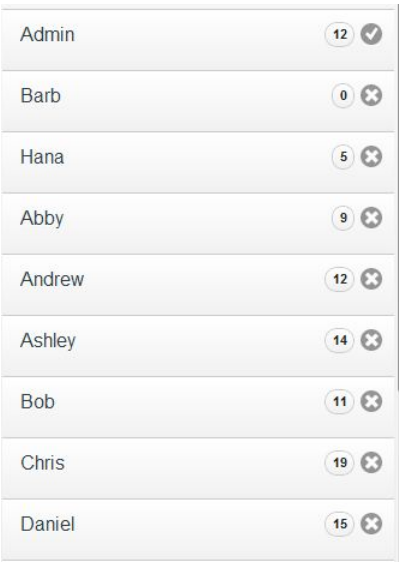


Input field:

Header:



Listview:



jQuery UI

jQuery UI is a plugin for jQuery that attempts to improve the aesthetics of the user interface by using elements that respond to user input. Three of the parts of this library that were explored during this project were “draggables,” “droppables,” and “sortables.” A group of HTML elements can be created into a “sortable,” which allows the elements to be dragged into different orders. Draggables are the same concept, except that the HTML element can be dragged anywhere within the container element, window or body, depending on settings. Typical settings for both draggable and sortable include: revert, an option to force the element to return to its original position along with the duration of the animation, appendTo, the HTML element that the element can be manipulated in, and helper, an option to move the original element on drag or make a copy of the element to move, leaving the original. Events include create, drag, start, and stop. During the entire process of making the tasks drag and droppable, we tried having them all be draggables. We discovered later that we wanted to keep all of the tasks organized and switched to “sortables.” Later in the summer, we saw an example of the other elements drifting to fit around the one being dragged, causing us to change the entire way the task elements move. Instead of making all of the task divs draggables, the individual div becomes a draggable object on click. The other elements move around it when it is dragged, and, when the dragging has stopped, the draggable is destroyed. Droppables were also used as the user list on the manage page to allow the tasks to be reassigned to a different user by dragging and dropping. All that

was required to make the user list “droppables” after calling droppable() on the HTML element was to define which elements can be dropped onto them, what should happen when an accepted draggable is over a droppable, and what should happen when an element was actually dropped, i.e reassign the task [13]. JQuery UI caused problems when we tried to make all of the elements drift around.

Ember.js

Ember is a client side JavaScript framework. The goal of the framework is to provide helpful tools and a basic workable organization to developers while also allowing for developer flexibility. Ember.js branched out and can also be used as a server side framework adding to the complexity when using it. It does, however, come with Handlebars.js, a templating language, prepackaged. Handlebars.js allows Ember.js and the screen to talk to each other through two way bindings. This means that when anything is updated on screen, it is automatically updated within Ember’s reference to that model, and vice-versa [14]. While this feature would have been helpful, we decided against Ember.js as an option for two reasons. The team had already decided to use Express.js for our server-side framework, which would mean extra work to disable Ember’s server-side code. Ember.js is also a very recent framework, causing new patches to be released each week. Some of the main functions were annoyingly changed with each update. The documentation did not reflect the new changes, and the community took a long time to adjust to the change. This is a framework worth looking at again once it is in a more stable, longer-lasting release.

Batman.js

Batman.js is a framework intended for the creation of single page web apps. It is heavily dependent upon the use of CoffeeScript, but it is still usable with regular JavaScript. Instead of using a templating tool like many other frameworks, Batman.js can be directly inserted into HTML as attributes. Batman.js focuses on speed, easy DOM manipulation, and reduction of redundant code. Our experience with it, however, proved that the project is still very much in its infancy. Following the suggested tutorials or getting an actual example to work was nearly impossible due to unforgivable discrepancies in various versions of the API [15].

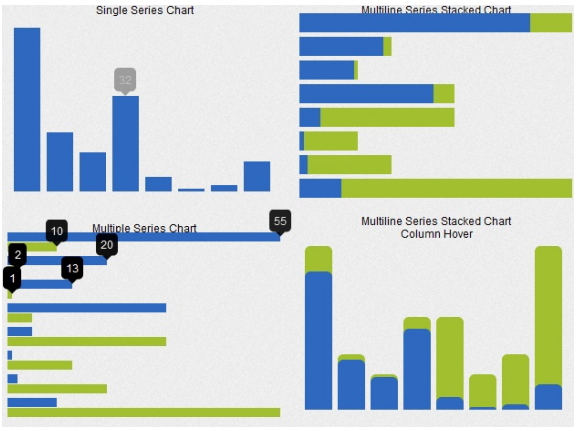
Backbone.js

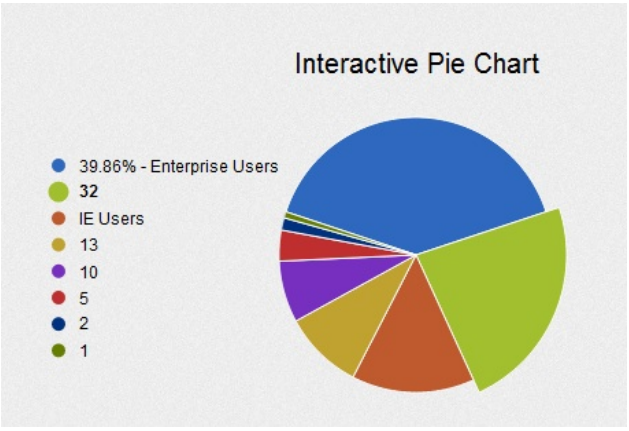
Backbone.js is a JavaScript framework that is loosely based on the model-view-controller (MVC) design. Backbone.js utilizes a RESTful interface when interacting with the server. It is lightweight and is designed for developing single-page applications [16]. There are three main entities used in Backbone: models, collections, and views. Models are objects with attributes that generally represent the data on the server. Collections are arrays of models with added functionality such as filtering and re-ordering. Views represent the UI and hold data containing what should be displayed in the DOM. Views are given either plain HTML or Underscore.js templates. The templates are then filled with data for the model associated with the specific view. It is possible and very common to nest views. Functions can be bound to models or

collections to interact with a view when a certain event is called. An example of this in Cerberus is re-rendering the TaskView when the task collection is reset. Backbone.js provides a client-side routing system which uses a hashtag in the URL when navigating pages, opposed to redirecting to a new URL. The DOM is replaced with new elements generated from templates in a view. The DOM is replaced with each route function which gives a fluid look when navigating pages. This is efficient because the developer can update in the DOM he or she wants and does not have to waste resources reloading preexisting content. Backbone.js is a very useful tool for creating simple single-page applications. For applications with a lot of dynamic data it is probably not the best solution. It leaves much to be wanted in terms of model-view interactions. The developer is forced to do more work than should be necessary to tie all of the aspects of the application together. Once everything is tied together, Backbone works exceptionally well. In the future we might explore Angular as an alternative.

gRaphael

gRaphael.js is an extension to a JavaScript drawing library called Raphael. gRaphael.js has limited documentation which makes it hard to use in the beginning. After looking at the available examples, we tweaked some base examples which taught us how to use the extension. Since gRaphael is an extension of Raphael, a Raphael object must first be made [17]. A Raphael object is an HTML element which uses SVG drawings on a canvas to create a surface on the screen that the rest of the plugin can render within. There are four different types of graphs supported by gRaphael: line, pie, bar, and dot. To use one of the types of graphs the corresponding JavaScript file has to be imported in addition to Raphael and gRaphael. To use a graph, we called the graph's method on the newly created Raphael object and specified the location within the canvas, size, and data to use. There are also specific hover functions that work out of the box with gRapheal [18]. All of the graphs look relatively nice at default settings, but if the end goal is to have different axis labels or highly customized graphs, this library is not the answer. Example graphs from demos:

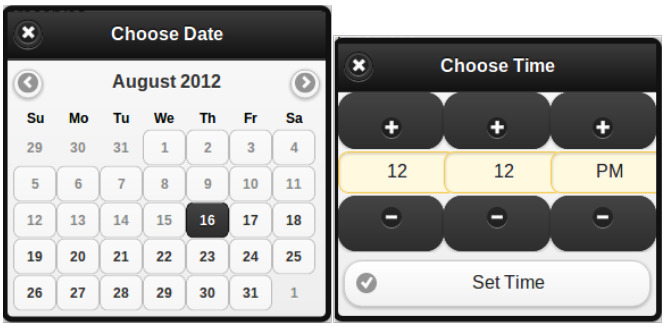




jQuery Mobile Datebox

jQuery Mobile Datebox is a plugin for jQuery Mobile that provide simple ways for users to interact with

dates and times. They provide nine different options for date and time selectors. For our project, we used CalBox and TimeBox on the add and edit task pages [19].



JQM Datebox was very easy to implement into our project. The plugin comes with numerous options that can be set to meet your exact needs. For example, we configured the date-picker to not allow users to select dates in the past. For our project, we envisioned having just one combined input for the date and time. This was surprisingly easy to implement using this plugin because it accepts callback functions for different events. We gave the plugin a callback function for when each dialog is closed. The callback for the date-picker will open the time-picker; the callback for the time-picker will save the user-input into the Backbone model and update the database. It is very easy to format the dates that the user selects. For our project, we formatted the date and time into ISO format for easy compatibility with the database.

jQuery UI Touch Punch

jQuery UI Touch Punch is a JavaScript library that allows jQuery UI widgets to work with touch events on mobile devices. It works by simulating a mouse event when the touch event equivalent is triggered. The library only needs to be included in the head of the HTML page and it will automatically intercept touch events and trigger the correct mouse events. jQuery UI will then use those mouse events just as it would in a normal desktop environment [20]. The core functionality of dragging task divs would not be possible on mobile devices without jQuery UI Touch Punch.

Moment.js

Moment.js is a JavaScript library which provides the capability to format and parse dates. Any JavaScript Date or JSON data can be made into a “moment.” Moment.js allows manipulation, parsing, formatting, or validation of the date. In our project, the date on each task div is formatted by Moment.js to display as text similar to “in twelve days” or “three hours ago,” in order to show when a task is due [21]. Moment.js is also used on the Task Edit page to display a more comprehensive format of the due dates show on the task divs. The Moment.js dates are used in conjunction with unformatted dates that are saved and retrieved from the PostgreSQL database.

Zepto.js

Zepto.js is a JavaScript library that attempts to provide a majority of the functionality of jQuery in a much smaller package. The minified production version of JQuery is 92 KB, and the development version is a much larger 256 KB. Compare this to Zepto.js, which boasts a 24 KB minified size and a mere 48 KB for the full version [22]. One of the goals of the project was to create an application that loads quickly in any browser. With this in mind, we tried to implement Zepto.js first. However, as the project grew, we decided to use the jQuery Mobile plugin, which requires jQuery. Zepto.js is very good for smaller projects or projects that do not require all of the features or plugins of jQuery.

Underscore.js

"Underscore is a utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Prototype.js (or Ruby), but without extending any of the built-in JavaScript objects. It's the tie to go along with jQuery's tux, and Backbone.js's suspenders" [23]. Underscore.js contains many functions that simplify the usage of collections and arrays such as forEach, map, filter, every, and indexOf. Underscore is backwards compatible with browsers; it will fall back to native implementations of its functions when requirements are not present in the browser. Underscore comes prepackaged with Backbone.js [23]. We used Underscore's templating capabilities in our Jade files to display a model attribute directly from the Backbone.js model. The team wrote templates for each Backbone view which were loaded using Underscore's template function. It would require a fair amount of work to get Underscore to update every time a model was changed, so we settled on re-rendering views when model or collections attributes are changed. Underscore is also used in our filtering/sorting of tasks on both the home and manage pages.

Python

Python is a general purpose scripting language with a very simple rational syntax, allowing for the quick creation of readable programs. Much of its usability is derived from its indentation based syntax, interpreted run time environment, and ability to support numerous programming styles. Despite it being generally considered a scripting language, it supports a vast array of libraries and can easily be packaged into a standalone application [24]. For our use case, Python was exclusively used to create a large number of randomly generated tasks to be inserted into the database. This helped extensively in testing the robustness and reliability of the application.

BCrypt

BCrypt is a Node.js module that can be used to hash passwords for security. BCrypt appends a salt to the password which is a randomly generated string of characters. We used this module to protect all user passwords in the database. BCrypt is a hashing functional, meaning passwords are unable to be decrypted [25]. When we need to check if a user-input matches the password in the database, on the login page for example, we hash the user-inputted text and compare the hash to what is stored in the database. The Node.js BCrypt module requires a C++ compiler, which made it unavailable when we tried

to host the project on Windows Azure. BCrypt worked very well other than the prerequisites needed for compiling.

Express.js

Express.js is a Node.js module that creates a web application framework which creates more simple and less redundant server code. There are three main aspects of Express: setting up the application, requesting information, and responding to requests. When setting up the application, the developer can set a port and configure the view engine and routing preferences. The team set the view engine to use Jade, rather than HTML, for our project. The requests to the server determine what page is shown or if a query to the database needs to occur. Requests contain information about the requester such as what they are requesting, data sent with the request, and the URL the request was sent to. The server can then respond with the correct information, sending or displaying information in the form of a response [26].

Jade.js

Jade is a templating engine heavily influenced by Haml. Jade is a different way of writing HTML. It makes creating HTML content easier to write and read. Like Python, Jade uses indentation to denote nested data, instead of curly braces like in JavaScript. For our project, Jade drastically reduced the amount of code we needed to write for our HTML. It provides inline JavaScript which helps to create dynamic web pages. Jade is implemented via a Node.js module. This means that all of the client-side HMTL is compiled on the server and is then sent back to the client. Jade also supports mixins and includes, functions that output HTML and imports of other .jade files, respectively. This does not always have to be the case because Jade supports client-side compiling [27].

```
script#desktop-tmplt-manage(type='text/template')
  <% var selectBar = true; %>
  mixin sidebar
  div(data-role="content")
    div#userSection
      h2 Users
      input#userSearch.userSearch(type='text', placeholder='Search for users...', name='userSearch')
      br
      ul#users.manage(data-role='listview')
    div#taskSection
      h2#tasksHeader Tasks
      div#tasks
        br
        | Click a user to view their tasks.
```

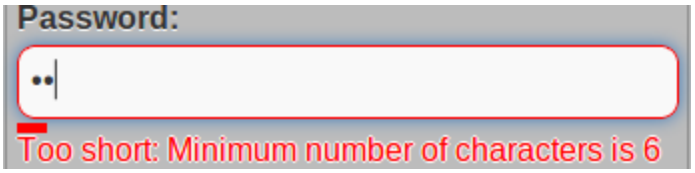
CSS

CSS is the third major pillar to web-development, along with HTML and JavaScript. CSS is used for the styling of the website. CSS is an acronym for Cascading Style Sheets. HTML elements are styled based on their classes or id. Some of the more difficult styling with this project came from the use of jQuery Mobile, a JavaScript library that adds classes to HTML elements before the page is rendered. This

hindered our ability to change the styling of certain elements for our own needs. The main parts of our custom css are used to make the task divs colorful with rounded edges and to override part of jQuery Mobile in order to get a look closer to what we wanted.

jQuery Password Strength Meter

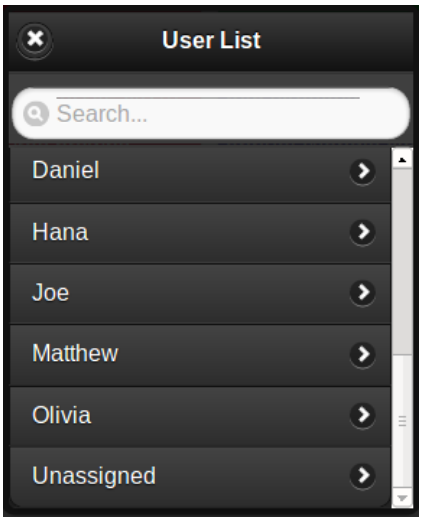
jQuery Password Strength Meter is a jQuery plugin that provides realtime client-side evaluation of a password, providing a visual indication regarding the strength of the password. The plugin can be used for any password input field and can be configured to have different tiers of security. It displays a simple, yet informative, progress bar showing the relative security of your entered password.



For this project we used the plugin for all of our password input fields. This plugin provides no real addition to the functionality of our application, it simply helps to encourage users to choose stronger passwords.

jQuery Mobile SimpleDialog2

jQuery Mobile provides excellent support for displaying information as a modal dialog. This is a pop-up dialog box that the user must interact with before returning to the main page content. JQM utilizes a routing type system for this, creating the new modal dialog as a "separate page" with a hashtag in the URL, which means the user does not actually leave the page. In our project, this was a major inconvenience. We had to disable jQuery Mobile's routing system because we chose to use Backbone's routing system instead. This meant that the modal dialogs would not work with our setup. After many hours of research, we finally found SimpleDialog2, a jQuery Mobile plugin. It provides all of the same functionality of a normal JQM dialog, except it doesn't use a hashtag in the URL. In fact, it doesn't need to change the URL at all! Surprisingly enough, it worked on all devices we tested it on. The developer is able to choose one of three options for configuration: button list mode, button input mode, and freeform mode. Button list mode is a simple way of listing input buttons, button input mode is a way of having text input fields, and freeform mode is a way to insert your own HTML into the dialog [28]. We used freeform mode to create a jQuery Mobile listview of users whenever user selection is needed.



The list items are generated from options of a hidden select menu which is generated from the Backbone collection of users.

System Design

Server

The team decided to implement a Node.js server. Node.js is a trending web server that runs off of the Google V8 engine. Node's asynchronous, non-blocking I/O has been a source of its rising popularity. There is a full analysis of Node.js in the Tools section. We used Express.js for the organization of the server. The group used PostgreSQL as the database system for the project. PostgreSQL is a popular, free database system, rivaling MySQL and SQL. We wrote different API files to organize the server into server calls to the database relating to users, tasks, and graphs. There is an utility file with helper functions including the PostgreSQL module to allow a connection with the database. Interestingly, the client-side HTML is rendered on the server from Jade syntax into proper HTML.

The following code displays the modules that were loaded and used by the server. Obtaining the library files was as simple as including the module names in the Node.js package.json file, and Node.js handles the rest. We only needed to require each module in our main server file, as shown below. We were also able to include our own server files, which housed the routing information for the server calls to the database.

```
var express = require('express');
var http = require('http');
var app = express();
var server = http.createServer(app);
var fs = require('fs');
var bcrypt = require('bcrypt');
var user_api = require('./user_api').api;
var task_api = require('./task_api').api;
var dashboard_api = require('./dashboard_api').api;
var util = require('./util');
util.io = require('socket.io').listen(server);
```

The main purpose of the server is to take a request from the client and respond with the requested information. In order to do this, the server is largely made up of routes. Each route has a String associated with it which is needed to be appended to the base URL in order for the function to be called on the server. Below is an example of our main route on the server. “Req” stands for the client-request, and “res” stands for the server response. Notice that the server responds to the client by rendering our main Jade file.

```
app.get('/main', function (req, res) {
    if (req.session.user_id)
        res.render('main');
    else
        res.redirect('/');
});
```

A large part of the server was to find and display information from the database. The task, user, and graph server files were composed of functions that interacted with the database using a specific SQL call. For example, the function below displays the attributes of a certain task in the database.

```
getTask:function (id, res) {
    var qs = "SELECT * FROM tasks WHERE task_id=$1";
    queryResponse(qs, [id], res);
},
```

The function below is a main part of the server. It interacts directly with the PostgreSQL database. Most other functions in the server utilize this function.

```
//Used to send information from database query back to client
var queryResponse = function (qs, data, res, func, err) {
    var query;
    if (data) {
        query = pgClient.query(qs, data);
    } else {
        query = pgClient.query(qs);
    }
    log("Database Query:", query.text);
    var affected_rows = [];
    query.on('row', function (row) {
        affected_rows.push(row);
    });
    query.on('end', function (result) {
        if (func) {
            func(res, affected_rows, data);
        }
        else if (affected_rows.length > 0) {
            res.writeHead(200, {'Content-Type':'text/plain'});
            if (affected_rows.length == 1)
                res.write(JSON.stringify(affected_rows[0]));
            else
                res.write(JSON.stringify(affected_rows));
            res.end();
        } else {
            res.end();
        }
```

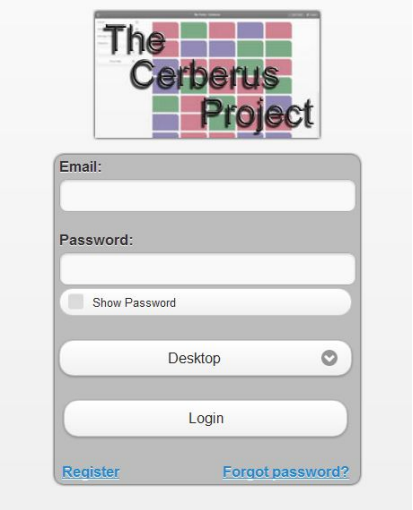
Client

We used Backbone.js to organize the client into views, models, and collections which were handled by a built-in routing system. We created four main Jade files: desktop, tablet, phone, and main. The main Jade file loads all of the CSS and JavaScript. We then implement Jade's "include" feature to include each Jade file for the different layouts. Each Backbone view has a template of HTML associated with it, with a different version of the template being stored in each layout Jade file. There are Task and User models with collections for each type of model. We decided to use two different collections for tasks: filtered and unfiltered. The filtered task collection has a cut-off size which is specified in the config.js file. The filtered task collection is filtered by the user's selection of the input fields on the sidebar of the home or manage pages. The unfiltered collection contains all of the tasks in the database that the user has access to view. In order to use jQuery Mobile, we had to disable its routing system. Each time the user clicks to change pages, the body of the DOM is replaced with new HTML that is loaded using Underscore from one of the templates from the Jade files. Backbone utilizes a hashtag in the URL, making Cerberus a single-page application. Backbone includes a RESTful interface for interacting with the server which is handled behind the scenes, minimizing our usage of manual AJAX calls using jQuery. The client code is largely the same between the layouts of the application.

Desktop

Login:
The login screen includes a logo with a background of the home page of the app. The email input field checks for a valid email. The password field includes a button to toggle the visibility of the password, which makes typing passwords easier on mobile devices. A select menu allows the user to select which type of interface they would like to use. The user's device type is automatically detected and the most suitable layout is pre-selected. The width is optimized to be about a quarter of the screen in the middle on a tablet or desktop and fill up the whole screen on a phone.

Login:



Register:

Home Page:
On the left side of the screen is the navigation pane with a button to toggle visibility in the upper-left corner. The pane includes links to all of the pages, the filtering/sorting controls for the current task collection, and a legend for the group of tasks located in the main part of the screen. Each task div is placed on a corresponding underlying grid of invisible slots composed of divs of the same shape and size as the tasks.

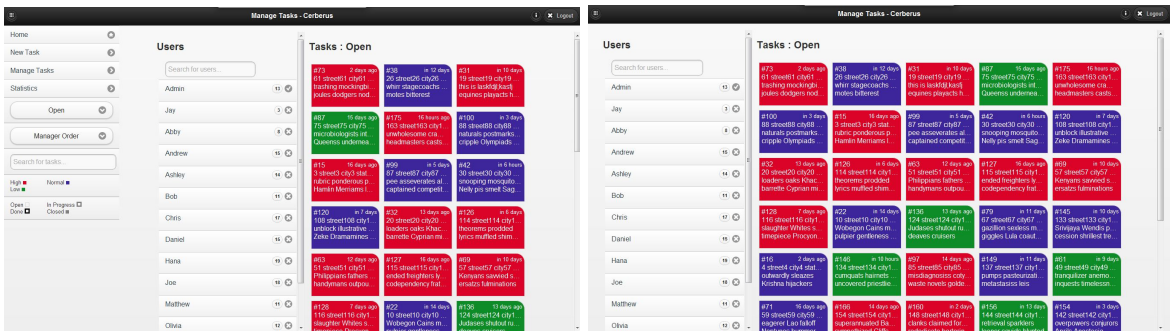
Here is part of the source code that makes the tasks drift when a task is being dragged:

```
driftPositions:function () {
    if (!this.drift)
        return;

    var stillDrift = false;
    for (var i = 0; i < this.collection.length; i++) {
        var taskDiv = $("#" + this.collection.at(i).id);//get task DOM object
        if (taskDiv.length > 0) {
            if (!taskDiv.data('draggable')) {//if taskDiv is not being dragged
                var slotDiv = $("#slot_" + i);
                var taskPos = taskDiv.position();
                var slotPos = slotDiv.position();
                var newX = this.closerCoord(taskPos.left, slotPos.left);//gets an x pos closer to the destination
                var newY = this.closerCoord(taskPos.top, slotPos.top);//gets a y pos closer to the destination
                if (Math.abs(newY - taskPos.top) >= 0.1 || Math.abs(newX - taskPos.left) >= 0.1) {//if pos is changed
                    setTranslate(taskDiv, newX, newY);
                    stillDrift = true;
                }
            }
        }
    }
    if (this.drift && !stillDrift) {
    }
    this.drift = stillDrift;
},
```

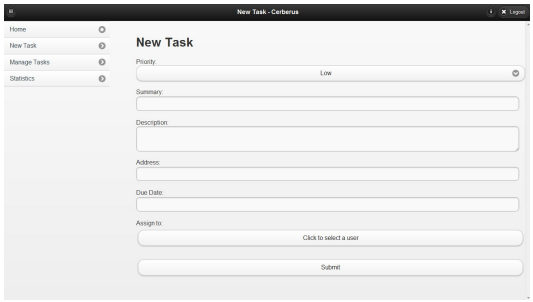
Manage Page:

The manage page allows managers to reassign a task to a different user by dragging and dropping a task onto a user in the list on the left. The count bubble indicates the number of combined open and in-progress tasks for each user, and the check or x next to each name indicates if the user is logged in. A manager may also click on a user from the list to edit the attributes of the user. Hiding the left sidebar is extremely helpful on this page because it allows the user to see more tasks. This layout along with the home screen had to be changed drastically to keep the functionality on the phone.



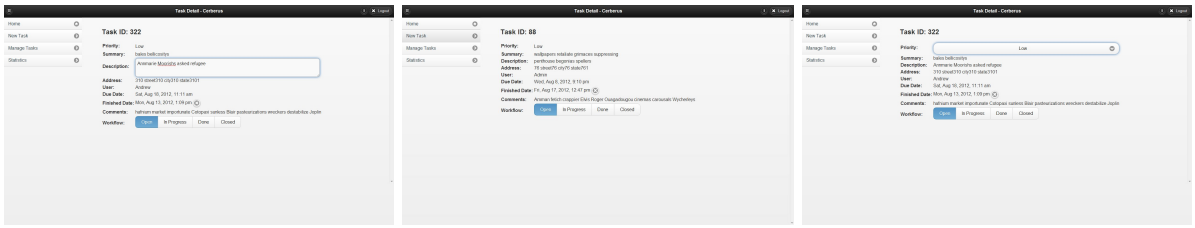
Add Task Page:

The add task page is a typical submission form styled with JQuery Mobile.



Task Edit Page:

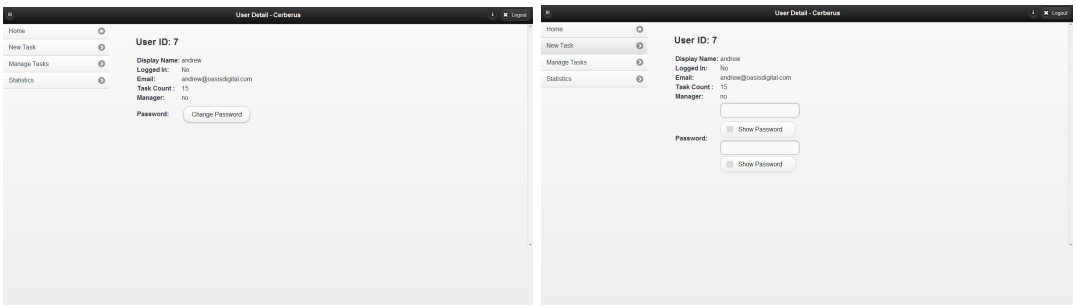
The edit task page begins as a table with labels on the left and content on the right. Once a user clicks a detail of the task, the data becomes an input box or some other user input such as a select or checkbox as appropriate. When a user clicks out of a field or presses enter, another function is called to validate the model, changes the display of the screen with the new information, and saves the changes to the database.



Edit User Page:

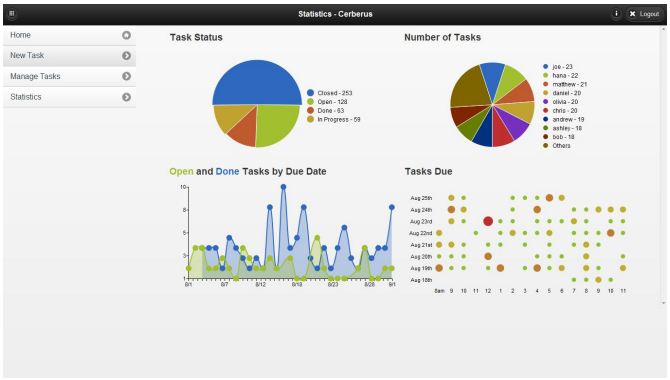
This page is similar to the task edit page because each field changes on click, validates, and saves on loss of focus. One of the differences is that the edit password field displays two entries to require the user to enter their password twice for verification. These fields also use the same requirements and validation as the password fields on the register page. In addition to being able to reach this page for

each user from the manage page, each individual user can change their information by clicking the “i” button in the upper right corner next to the logout button.



Statistics Page:

This is a page of graphs inspired by JIRA and the Beebole dashboard to show the practical usability of allowing managers to see how their company as a whole is functioning and keeping up with or falling behind demand. The upper left pie chart shows the number of tasks based on a status of open, in progress, done, and closed. The upper right pie chart shows the distribution of “todo” tasks, all tasks either open or in progress, assigned to a given user. The bottom left line-chart shows the number of tasks due on a given date both 15 days in the past and in the future. The green line denotes tasks with an open status and the blue line indicates a done status. The bottom right “dot” chart shows the number of tasks that are going to be due at a given hour within the next seven days. Some difficulties included timezones for displaying the dot and line chart and rendering well across all platforms. This uses gRaphael.js and renders the graphs on canvas elements. This page is the same on a tablet and the charts are in a column instead of a table on a phone in order to effectively using screen space.



Tablet & Phone

Our main goal for the mobile versions of the Cerberus Project was to make the application look and feel as native as possible. To do this, we tested the application on many devices. These devices include the iPhone 3GS, iPhone 4, iPhone 4S, iPad 2, iPad (3rd generation), Samsung Galaxy Nexus, Samsung Galaxy (1st generation), and LG Optimus. We found the application to run noticeably faster and smoother when

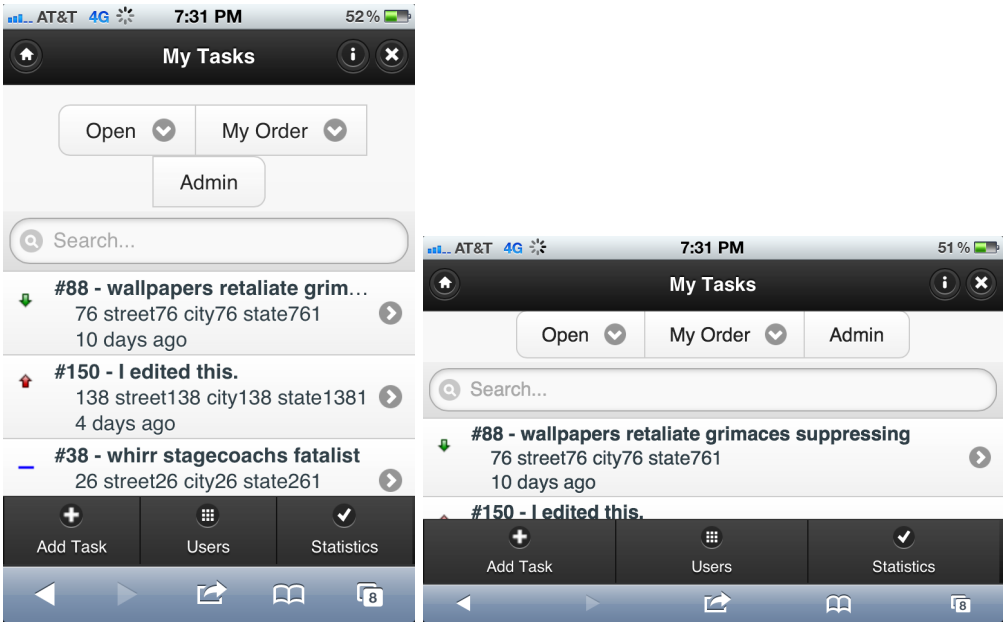
using the Chromium browser. In order to facilitate part of the mobile experience, we attempt to detect the user's device using a JavaScript library called DeviceDetection by Daniel Pötzinger, and then set the select box on the login page to be the most suitable layout for the user's device.

Tablet:

There were very few changes required to make our original layout work well on a tablet. Words were cut off on the sidebar on seven-inch tablets and as a remedy we made the width of the side bar marginally higher and the width of the content div marginally smaller. One unresolved problem with certain seven-inch tablets is the content is sometimes displayed below the sidebar. This is a problem with the width of the sidebar div and content divs combined being too wide for the screen resolution.

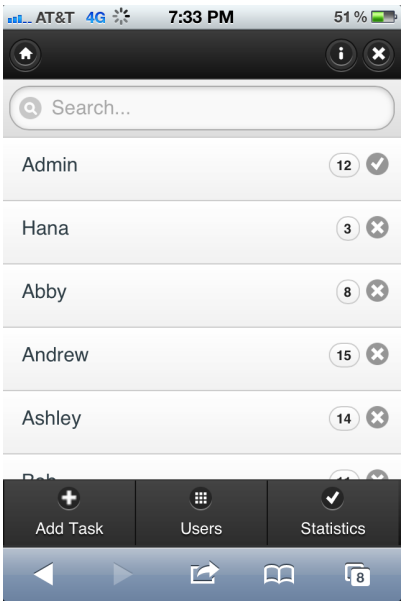
Phone:

In order to maximize the usage of small screen sizes of mobile devices, we needed to change how our home tasklist page was displayed. The team decided to adopt a native mobile-themed approach, so we implemented a jQuery Mobile listview to display the tasks.

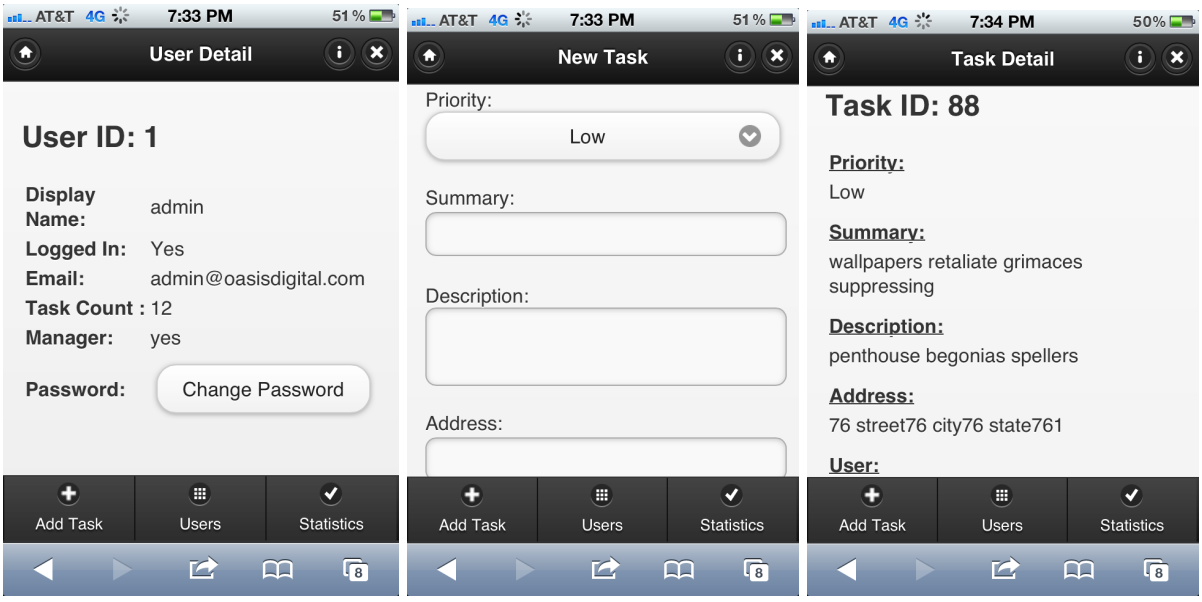


The only major difference in the page-organization that is unique to the mobile layout is the addition of a userlist page and removal of the manager page. The team felt that there was not enough screen space to efficiently implement a manager screen for mobile devices. Instead, a manager can access a list of all users. From this page, the manager is able to see the number of open or in progress tasks assigned to each user. The task reassignment feature that is available from the manager page on a desktop is

unavailable for mobile devices. Instead, a manager must manually edit the user from the task edit page.



The rest of the mobile layout is almost exactly the same as the desktop version. There are minor changes in the styling and layout of content on the task detail and edit pages.



What We Learned - List at least 3 main concepts. Describe your perspective prior to the project and the difference now. List some things that you did not even consider important previously that will help you in the future.

Frameworks

JavaScript frameworks and libraries allow developers to focus primarily on accomplishing only what they need to do. The framework sets a skeleton for the developer to work with, and libraries accomplish specific tasks without the developer having to re-write already available code. Model-View-Controller (MVC) frameworks allow for easy control of data and how the user views data. Backbone is not the only MVC framework option. Many frameworks should be considered when starting a project because there are certain features and limitations of each one. It is also something that should be decided very early on because each framework will entirely change how the application is structured. Backbone.js is a MVC framework that we used to organize the client.

Models:

Each model has a set of defaults and functions that can be applied to it but when data is returned from the server the defaults are overridden.

```
var User = Backbone.Model.extend({
    initialize:function () {
        this.formatDisplayName(this);
        this.id = this.attributes.user_id;
    },
    formatDisplayName:function (model) {
        var old = model.get("display_name");
        var pretty_display_name = old.capitalize();
        model.set({"pretty_display_name":pretty_display_name});
    },
    defaults:{
        user_id:"",
        email:"",
        display_name:"",
        password:"",
        task_count:"",
        manage:"",
        logged_in:"",
        pretty_display_name:""
    },
    idAttribute:"user_id",
    urlRoot:base_url + '/user'
});
```

Collections:

A basic collection is just an ordered set of models with its own functions and events.

```
var UserCollection = Backbone.Collection.extend({
    model:User,
    url:base_url + "/user",
    comparator:function (User) {
        return (User.get('logged_in') ? "0" : "1") + User.get('display_name');
    }
});
```

Views:

Views control what is input in as HTML to the DOM. When a view is initialized, the initialize function is called. When render is called, the HTML is rendered from the template into the DOM.

```
var MobileUserListView = Backbone.View.extend({
    template:function () {
        if (layout === "phone")
            return _.template($("#phone-tmplt-userListPage").html());
        else
            return "";
    },
    initialize:function () {
        new UserTableView({
            collection:UserColl
        });
        UserColl.fetch();
    },
    render:function () {
        $(this.el).html(this.template());
    }
});
```

and a Controller:

The router contains functions for each page and manages everything required to switch between views.

```
var AppRouter = Backbone.Router.extend({
    routes:{
        "":"homePage",
        "addTask":"addTaskPage", // #addTask"
        "task":"taskPage", // #task
        "manage":"managePage", //#manage
        "user":"userPage", //#user
        "statistics":"statisticsPage", //#statistics
        "users":"userListPage" //#users
    },
```

Database

We used a traditional relational database for this project and learned about using a schema to easily set up tables and update databases for each developer. We also learned about using SQL to get information from both tables, gaining information like counts, and how to use groupings and orderings. Time zones were also a factor because we hosted our application on a web hosting service in a different time zone.

Some of the more complicated queries were required with the addition of the statistics page. In addition to needing numbers from groups of data in one table, we needed information from both tables, and the data to be grouped by days and number of tasks. Here is one of the queries from that page:

Here is one of the queries from that page:

```
CREATE VIEW dotGraphData AS
SELECT
  CAST(date_part('hour', due_date) AS integer) AS x,
  date(date_trunc('day', due_date))-current_date AS y,
  COUNT (*) AS n
FROM tasks
WHERE due_date >= now()
AND due_date <= now() + interval '7 days'
GROUP BY x,y
ORDER BY 1,2;
```

Here is the JSON that it produces:

```
[{"x":8,"y":5,"n":2},{"x":8,"y":7,"n":2},{"x":9,"y":1,"n":3},{"x":9,"y":2,"n":1},{"x":9,"y":3,"n":3},{"x":9,"y":4,"n":1},{"x":9,"y":5,"n":2},{"x":9,"y":7,"n":1},
{"x":10,"y":1,"n":1},{"x":10,"y":3,"n":2},{"x":10,"y":5,"n":1},{"x":11,"y":1,"n":1},{"x":11,"y":2,"n":1},{"x":11,"y":4,"n":3},{"x":11,"y":5,"n":1},{"x":11,"y":6,"n":1},
{"x":12,"y":3,"n":1},{"x":12,"y":5,"n":1},{"x":13,"y":0,"n":1},{"x":13,"y":2,"n":4},{"x":13,"y":3,"n":1},{"x":13,"y":4,"n":1},{"x":13,"y":5,"n":1},{"x":13,"y":6,"n":2},
{"x":14,"y":0,"n":1},{"x":14,"y":1,"n":2},{"x":14,"y":2,"n":2},{"x":14,"y":4,"n":1},{"x":14,"y":5,"n":2},{"x":15,"y":1,"n":2},{"x":15,"y":2,"n":2},{"x":15,"y":3,"n":1},
{"x":15,"y":4,"n":2},{"x":15,"y":5,"n":1},{"x":16,"y":0,"n":1},{"x":16,"y":5,"n":2},{"x":16,"y":6,"n":1},{"x":17,"y":0,"n":2},{"x":17,"y":1,"n":1},{"x":17,"y":3,"n":3},
{"x":18,"y":0,"n":1},{"x":18,"y":3,"n":1},{"x":18,"y":4,"n":1},{"x":18,"y":5,"n":1},{"x":18,"y":6,"n":1},{"x":19,"y":0,"n":1},{"x":19,"y":1,"n":2},{"x":19,"y":3,"n":1},
{"x":19,"y":4,"n":4},{"x":20,"y":0,"n":1},{"x":20,"y":1,"n":1},{"x":20,"y":3,"n":1},{"x":20,"y":6,"n":1},{"x":21,"y":1,"n":1},{"x":21,"y":2,"n":1},{"x":21,"y":3,"n":1},
{"x":22,"y":0,"n":2},{"x":22,"y":1,"n":1},{"x":22,"y":2,"n":3},{"x":22,"y":4,"n":1},{"x":22,"y":6,"n":2},{"x":23,"y":2,"n":1},{"x":23,"y":3,"n":2},{"x":23,"y":6,"n":1}]
```


Another complicated query we tried to use was a recursive query to get the tasks in a certain order. We eventually gave up on this and ordered the tasks with an array stored in the database.

```
CREATE VIEW recursiveTasks AS
WITH RECURSIVE rqName (task_id, user_id, summary, description, address,
priority, due_date, finished_date, comments, workflow, prior_task, sort_order) AS
(SELECT task_id      ,
        user_id      ,
        summary      ,
        description  ,
        address      ,
        priority     ,
        due_date     ,
        finished_date,
        comments     ,
        workflow     ,
        prior_task   ,
        0
FROM    tasks
WHERE   prior_task IS NULL

UNION ALL

SELECT tn.task_id      ,
       tn.user_id      ,
       tn.summary      ,
       tn.description  ,
       tn.address      ,
       tn.priority     ,
       tn.due_date     ,
       tn.finished_date,
       tn.comments     ,
       tn.workflow     ,
       tn.prior_task   ,
       tp.sort_order + 1
FROM   rqName tp,
       tasks tn
WHERE  tp.task_id = tn.prior_task
)
SELECT *
FROM   rqName;
```

Javascript

With very minimal JavaScript and even less Node.js experience between the intern developers, it was very impressive that we were able to build an entire app with JavaScript on both the client and the server. With the decision to use jQuery, some of the more complicated, or at least more redundant in terms of code, parts of using JavaScript, especially for manipulation the DOM, went away. As a group there was some background in Java making the understanding of basic functions, variables and other general practices transfer easily to JavaScript. This left only syntax and quirks of JavaScript including global name spacing and Boolean true/false values as part of the learning curve.

Socket.io

In this day and age, information demands to be displayed instantaneously. Individuals and companies heavily rely on the fact that data is instantly available for access. If there was a way to make the process of obtaining this new data more efficient and easier for the user, we knew we had to implement it. We eventually stumbled across a JavaScript package named Socket.io which was exactly what we needed to add a sleek finish to our project. Socket.io is used to instantly push data to clients without user-interaction. Full technical details regarding socket.io can be found in the Tools section. Socket.io taught the team that we can extend product-simplicity throughout all aspects of a project. We did not know that we could make it easier for the user by making them never need to refresh the page. The need to refresh the page to receive new data is just another aspect of the technological zeitgeist we are transitioning from. Below is code from the application that displays the server-side code alongside the client-side code. The server-side code sends an event to all connected clients which interpret the event based on conditions unique to the user.

Server-side:

```
var old_user;
var x = "SELECT * FROM tasks WHERE task_id=$1";
queryResponse(x, [id], res, function (res, data) {
    old_user = data;
});
queryResponse(qs, data, res, function (res, data) {
    var getUserTaskCount = function (id, res) {
        var qs = "SELECT COUNT(*) FROM tasks WHERE user_id=$1 AND workflow < 3";
        queryResponse(qs, [id], res, function (res, data) {
            util.io.sockets.emit('task_count_changed', {owner:id, count:data[0]['count'] });
        });
    };
    if (fields['user_id'] && old_user[0]['user_id'] && fields['user_id'] !== old_user[0]['user_id']) {
        getUserTaskCount(old_user[0]['user_id'], res);
        getUserTaskCount(fields['user_id'], res);
    }
    queryResponse("select * from tasks WHERE task_id=$1", [id], res, function (res, data) {
        util.io.sockets.emit('task_changed', {id:id, task:data[0]});
    });
    res.end();
});
```

Client-side:

```
socket.on('task_count_changed', function (data) {
    if (manage) {
        if (UserColl.get(data.owner)) {
            var model = UserColl.get(data.owner);
            model.set('task_count', data.count);
        }
    }
});
```

Using Documentation and Resources

The Oasis Digital Internship taught the team the importance of using available documentation and resources. Our project can be thought of as a puzzle; we put together many different pre-existing modules and packages, tinkering each one in order for them to all fit together perfectly. This would be very difficult without documentation for each of the modules. Some packages had very limited documentation. This proved to be very difficult to bypass while building the application. In some instances, such as with Ember.js and Batman.js, we were unable to work efficiently with the limited resources. The team had to disregard those packages and find an alternative, Backbone.js. When we did not understand something with the documentation, we sometimes asked peers online in discussion boards. A popular one that we used was StackOverflow. It is never dishonest, cheating, or immoral to get help from others to help finish a project. The best software engineers utilize the resources available to them. Lastly, the team learned to never underestimate the power of search engines. With the right search terms, a powerful search engine will make finishing a project so much easier. We spend just as much time on Google researching how to fix a problem as we did actually programming the fix. Google was definitely at the top of our list of online resources.

Works Cited

[1] Joyent, Inc. “Node.js.” Internet: http://nodejs.org/, [Sep. 3, 2012].
[2] Joyent, Inc. “Node Package Modules.” Internet: http://npmjs.org/, [Sep. 3, 2012].
[3] Atlassian. “JIRA.” Internet: http://www.atlassian.com/software/jira/overview, Sep. 5, 2012 [Sep. 11, 2012].
[4] BeeBole. “The Timesheet you were looking for.” Internet: http://beebole.com/, [Sep. 11, 2012].
[5] Atlassian. “BitBucket.” Internet: http://www.atlassian.com/software/bitbucket/overview, [Sep. 11, 2012].
[6] “Git.” Internet: http://git-scm.com/about, [Sep. 11, 2012].
[7] ECMA International “ECMAScript Language Specification.” Internet: http://ecma-international.org/ecma-262/5.1/, [Sep. 16, 2012].
[8] “Heroku.” Internet: http://www.heroku.com/, [Sep. 14, 2012]
[9] The PostgreSQL Global Development Group. “PostgreSQL.” Internet: http://www.postgresql.org/about/, [Sep. 14, 2012].
[10] Guillermo Rauch. “Socket.io.” Internet: http://socket.io/, [Sep. 14, 2012].
[11] The jQuery Foundation. “jQuery.” Internet: http://jquery.com/, [Sep. 14, 2012].
[12] The jQuery Foundation. “jQuery Mobile.” Internet: http://jquerymobile.com/, [Sep. 14, 2012].
[13] The jQuery Foundation, jQuery UI Team. “jQuery UI.” Internet: http://jqueryui.com/, [Sep. 14, 2012].
[14] Tilde, Inc. “Ember.” Internet: http://emberjs.com/documentation/, [Sep. 15, 2012].
[15] Shopify. “Batman.js.” Internet: http://batmanjs.org/, [Sep. 15, 2012]/
[16] DocumentCloud. “Backbone.js.” Internet: http://backbonejs.org/, [Sep. 15, 2012].
[17] Dmitry Baranovskiy. “raphael.” Internet: http://raphaeljs.com/, [Sep. 15, 2012].
[18] Dmitry Baranovskiy. “g.raphael.” Internet: http://g.raphaeljs.com/, [Sep. 15, 2012].
[19] Jonathan Sage. “jQueryMobile - Datebox.”http://dev.jtsage.com/jQM-DateBox/, [Sep. 15, 2012].
[20] David Furfero. “jQuery UI Touch Punch.” http://touchpunch.furf.com/, [Sep. 15, 2012].
[21] Tim Wood. “Moment.js.” Internet: http://momentjs.com/, [Sep. 15, 2012].
[22] Thomas Fuchs. “Zepto.js” Internet: http://zeptojs.com, [Sep. 15, 2021].
[23] DocumentCloud. “Underscore.js.” Internet: http://underscorejs.org/, [Sep. 13, 2012].
[24] Phython Software Foundation. “Python Programming Language - Official Website.” Internet: http://www.python.org/, [Sep. 15, 2012].
[25] “BCrypt.” Internet: https://github.com/ncb000gt/node.bcrypt.js/, [Sep. 14, 2012].
[26] Visionmedia, TJ Holoqaychuck “express.” Internet: http://expressjs.com/api.html, [Sep. 15, 2012].
[27] Visionmedia. “Jade.” Internet: https://github.com/visionmedia/jade#readme, [Sep. 14, 2012].
[28] Jonathan Sage. “jQueryMobile SimpleDialog2.” Internet: http://dev.jtsage.com/jQM-SimpleDialog/demos2/index.html, [Sep. 10, 2012].

Further Reading

http://coenraets.org/blog/2012/03/using-backbone-js-with-jquery-mobile/
http://www.nodebeginner.org/
http://diveintohtml5.info/index.html
http://www.emberist.com/
http://www.adobe.com/devnet/html5/articles/flame-on-a-beginners-guide-to-emberjs.html
http://www.knockmeout.net/2011/06/10-things-to-know-about-knockoutjs-on.html
http://www.chromeweblab.com/