TritonPARK, a UCSD assistive parking application

GREGORY MARCHESE, University of California, San Diego
YASMINE KOTTURI, University of California, San Diego
JAYSEN PARMAR, University of California San Diego

Parking is a real issue, and one that knows no boundaries. UC San Diego is no different -- parking is a sore spot for many students, faculty, and staff. Given that many people depend on a car to get around San Diego, a sprawling and geographically diverse region, simply not owning a car is a poor solution for many. Furthermore, the public transport system is slow, and does not reach many areas of San Diego further exacerbating the issue of parking. From personal experience, finding a parking spot at UCSD is a fine art. From knowing when peak hours are, to allocating an extra half-hour of travel time to your schedule to account for driving around the parking structure multiple times until you find a spot, the issues with parking stem from a lack of solid information as to increase efficiency of this system. This doesn't just stop with students. Faculty and staff are not only forced to buy parking permits for their jobs, but due to the sparse parking situation, they have much trouble finding a spot too.

Categories and Subject Descriptors: **H.5.2 [Information Interfaces and Presentation]**: User Interfaces—*Evaluation/methodology*; **H.1.2 [Models and Principles]**: User/Machine Systems—*Human Information Processing*;

General Terms: Human Computer Interaction

Additional Key Words and Phrases: Parking, Visualization, UCSD

1. INTRODUCTION

Parking is a real issue, and one that knows no boundaries. UC San Diego is no different -- parking is a sore spot for many students, faculty, and staff. Given that many people depend on a car to get around San Diego, a sprawling and geographically diverse region, simply not owning a car is a poor solution for many. Furthermore, the public transport system is slow, and does not reach many areas of San Diego further exacerbating the issue of parking. From personal experience, finding a parking spot at UCSD is a fine art. From knowing when peak hours are, to allocating an extra half-hour of travel time to your schedule to account for driving around the parking structure multiple times until you find a spot, the issues with parking stem from a lack of solid information as to increase efficiency of this system. This doesn't just stop with students. Faculty and staff are not only forced to buy parking permits for their jobs, but due to the sparse parking situation, they have much trouble finding a spot too.

1.1 Summary of Requirements

Any given solution to these issues requires that the surprisingly detailed information from the UCSD Transportation Office be used to create a prediction of parking availability. Further, the possibility of providing a means to correct these predictive models based on a real-time component would be helpful and increase the model accuracy while increasing the helpfulness to the user. Finally, given the nature of parking, it is imperative that this information be accessible quickly, easily, and without extraneous info, as it is more than likely that a solution will be used in a (possibly moving) vehicle.

1.2 Presentation of Solution

To these requirements, Group 13 presents, TritonPARK, a responsive, real-time, and location enabled web application, implementing HTML5 geolocation, predictive modeling from historical traffic data, web socket based real-time parking availability notifications, and schedulable notifications, all hosted on a bare-metal server cluster.

Publication date: June 2014

1.3 Report Organization

This report has been organized into several sections, intended to follow the design, development, and future of TritonPARK. Each section focuses on one aspect of the TritonPARK project.

(1) Introduction	(Sections 1.1-1.3)
(2) Motivation & Background	(Section 1.4)
(3) Application Design & Realization	(Sections 1.5-1.10)
(4) Technical Design & Implementation	(Sections 1.11-1.12)
(5) Human Computer Interaction Design	(Sections 1.13-1.14)
(6) Project Management & Cross-Focus Team Collaboration	(Sections 1.15-1.18)
(7) Application Status & Future	(Sections 1.19-1.21)

1.4 Motivation & Background

As the world's population continues to grow and an increasing number of vehicles start to roam the roads, it's apparent that parking will become a problem for many. People tend to spend a lot of time searching for parking in densely populated areas by going up and down rows in parking lots or checking the parking garages for vacancy in the given area. This results in wasted time and increases frustration. To help alleviate this issue, there are currently applications being developed such as Streetline that help with the parking problem. Data from these applications is gathered through street-level sensors and parking analytics. Applications like these are useful for planning and helping people get to where they need to be without wasted time or effort. However, solving a problem like this is not easy because there are an astronomical number of parking spaces to keep track of within a densely populated area, and expanding the service to reach as many people as possible will take a lot of time as the appropriate hardware and infrastructure starts to become more available.

On a smaller scale, there are many UCSD students who own a car and it does not help that UCSD sells more parking permits than there are spaces available in parking lots and parking structures. As a result, it can be nearly impossible to find a parking space on campus and people usually need to plan their day around parking availability. By simply observing traffic flow at these parking spaces and lots, one will quickly gather the current methods students use to try and locate parking. One of them involves arriving early enough only to not find any available spots, then waiting in the car in the parking lot until a space opens up. This method requires a lot of time and patience and it is extremely frustrating when another student is quick to grab the same spot someone has been waiting for. Simply put, parking at UCSD is a nightmare, and there is a need for a reliable way to find parking when time is tight and where there may be a few free spots.

Our fellow collaborators of COGS 102C conducted interviews in order to conduct need-finding studies. Examples of these questions are below:

- How long have you been parking at UCSD?
- How much time do you usually spend on parking?
- Where do you usually park?
- What is your knowledge of all of rules in regards to parking?
- What are some financial or emotional investments you make toward parking?

- What are your personal frustrations with parking?
- What are some strategies you use in regards to parking?
- What's the most negative worst experience you've had with parking? Best?

From this data, a user profile would be generated. An example response is below.

- *USER PROFILE:*
    - 3rd year female Village resident
    - New user, first time with a car & parking experience;
    - S permit
- *EXPERIENCE*
    - Must move car after 2AM
    - Checks Village, then Hopkins, then Pangea
    - Wasted gas and money trying to find parking (from circling around campus)
    - Financial investment to buy parking permit
    - Emotional frustration at seeing other people get parking
    - Best experience: finding parking quickly in Hopkins or Pangea on a weeknight (20/30%)
    - Always uses permit to park on campus
    - Considers herself as a new user
    - Learned knowledge from old users
    - Used to go to Pangea regularly but now mainly goes to Hopkins

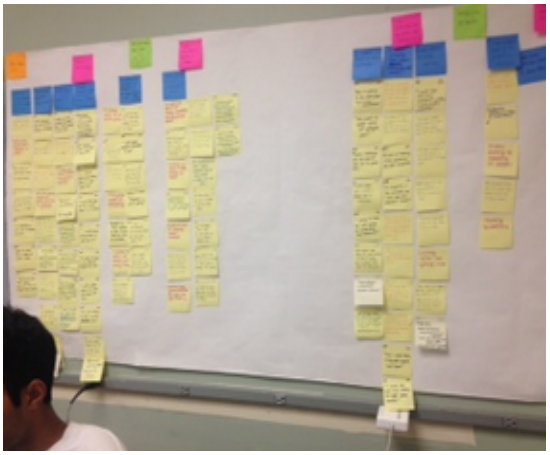

Fig. 1. User Affinity Diagram produced to visualize specific trends across various users, users needs, and desires.

After correspondents completed their interviews and need finding, the 102C group came up with an affinity diagram in order to summarize their findings (Figure 1). Using an affinity diagram, we were able to group issues that users experienced and then see which issues were the main problems and which issues were sub-problems. From the affinity diagram, we were able to understand that the main problems related to parking on campus: price, availability, and location. While our app cannot

necessarily reduce the price of parking permits, we decided that one area that we could help alleviate stress in was availability and location.

The next step was to conduct market research and to determine presence of other solutions. While the UCSD Transportation Office publishes all data of availability and inventory, a highly detailed tabular format was determined to be an inefficient and ineffective method to convey parking probability for a user that is likely looking at our app while in the car. However, a similar app was found, called ParkUCSD, and is located at http://parkucsd.net was found late in the development cycle, it was determined that this app, was not particularly adept at display data effectively. This version, while optimized for mobile platforms, is still text heavy, and lacks integration with Google Maps thereby forcing a user to leave the application and jump into another. This jumping, which occurs rapidly, and without warning, is jarring to the user experience; this point was especially considered in our version of the app. This app is discussed in further detail later in the report.

1.5 Design Summary

Our idea is designed to be centralized and mobile in order to help the on-the-go user find parking fast. Our goals for the design included making the information accessible and easy to view for the user. We wanted the user to be able to choose the most optimal parking location based on the probability of open spaces per permit type and time of day. We also wanted the user to visualize the parking space data based on his or her mental model of the campus space.

1.6 White boarding

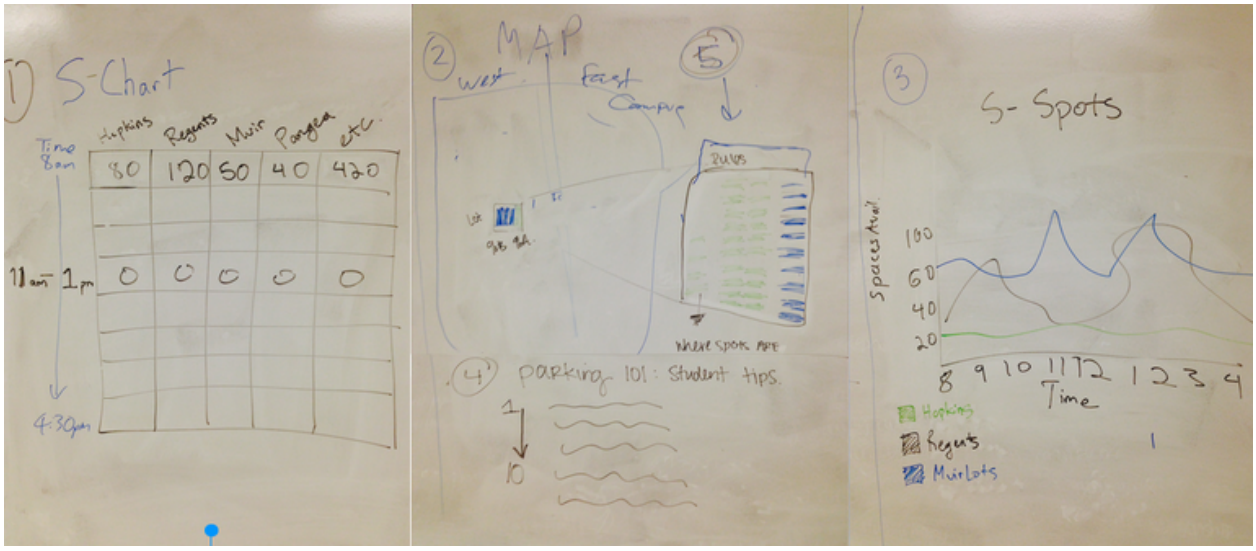

Fig. 2. Whiteboard diagram of the TritonPARK app.

We used white boards to initially layout and share our vision for the application, what information it would display, and design the user-flow (Figure 2). White boarding proved to be a useful large format to quickly sketch out ideas, edit them, and present them in a group setting.

In these designs, we focused on how to present the data we gathered from UCSD Parking Services to the user. Our various methods of presentation included very dense tables and graphs as pictured above.

With this design it is apparent that we can display a lot of information to the user at once in an organized fashion; however, this can be overwhelming to the user if he or she wants a quick reference to a particular time and view the number of spots available at a certain time. Sifting through all the data to view a particular piece of information can be very time consuming for the user this way and can make the user frustrated, so we needed to refine our design further and be very particular about what we wanted to display at which point in the user flow.

1.7 Paper Prototyping

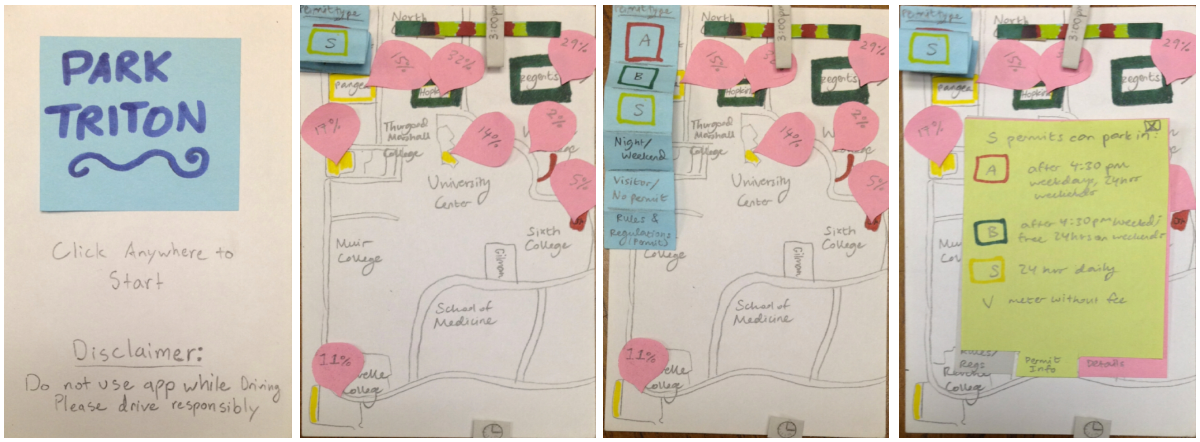

Fig. 3. These paper prototypes were created by the 102c group to help them communicate their ideas for the applications and what it would look like to us.

Starting with paper prototypes is low cost way of getting ideas out fast. If they are made with cut out elements then things can easily be moved around and adjusted. In addition, new screens do not have to be made because the overlay elements will change and it's easy to make changes by simply creating new paper elements. In addition, paper prototypes are tangible and closely replicate the interaction the user will have with the display (Figure 3).

With this design the user is able to pick up on visual cues more easily to assess how likely they are to find parking at the different parking lots and structures around campus because each location is labeled with a percentage the indicates the likelihood of finding parking. In addition, all of this information is displayed over a map so that the user is able to locate him or herself geographically. This assists the user with his or her mental model of the campus and helps them get oriented and accustomed to the user interface more quickly.

1.8 Wireframing

Fig. 4. We also used Balsamiq to create a wireframe of the user interface for our app.

This was our original design. The map with the data overlay infographic was a popular design element and has been a part of the application's development since the initial prototyping stages because of its effectiveness in displaying data and orienting the user. Since the original design, the interface has changed to a one-page application for the sake of simplicity and after taking into consideration that most of our users will probably be operating a vehicle while looking for parking. The advantage to the design featured above is that the user would be able to systematically filter his or her settings on the first screen. After clicking on a particular parking lot or structure for more information, the user can then view more detailed statistics. We tried to put relative information together and create a separation of concerns by having multiple screens that way the user could have better sense of organization when navigating the application's screens. We later swapped this idea for a simplified one-page approach.

In our final design we chose to implement the ability to have the user choose his or her type of permit and then have the parking information display the corresponding parking information over a map. We felt like this would be the most effective way to communicate appropriate information to the user so that he or she would be able to reference it at a glance and make an informed parking decision.

1.9 Timeline

We all wanted to be a part of making the parking application because it is an issue that is close to our hearts. In the beginning, we had a slight delay due to miscommunications of grouping between our classes at COGS 102C. Once that was sorted out, we were able to move forward with our COGS 102C partners. We started with a meeting once a week where we were able to communicate with our partner group and help in part of the design process. The first week, we helped them with their affinity diagram, which was used in order to conduct need finding. After the main needs were established in relation to parking on campus, we moved forward to start addressing them. We were able to proactively tackle the problems and accomplish our goals. Towards the end, our schedules

started to get discordant, but we were able to accomplish our project through social media and Google documents.

Throughout this process, the idea behind the parking application changed. With the excitement of thinking that we could possibly contribute to a solution, even be the solution, to this sore spot for many, we had ideas of making a real-time app that could be synched up to video cameras or sensors that could give precise parking availability. Of course, in a month, this is not possible and the tools just aren't there (cameras/sensors/money). So instead we had to use what we could work with, and that was the hundreds of data sheets that monitor the fluctuations of parking availability. Converting these sheets, as mentioned, was a rigorous and time-consuming task, so once again we had to be realistic with how many aspects of the parking issues we could tackle. In the end, our application changed drastically when we decided to hone in on a "prediction application" that used previous collected data of parking availability that would give a number indicating how likely that the user would find parking in a particular structure or lot. In short, throughout the course of this past month, our application's design and implementation has been an iterative process that has left us with a fully functioning and intuitive application.

1.10 Final Design

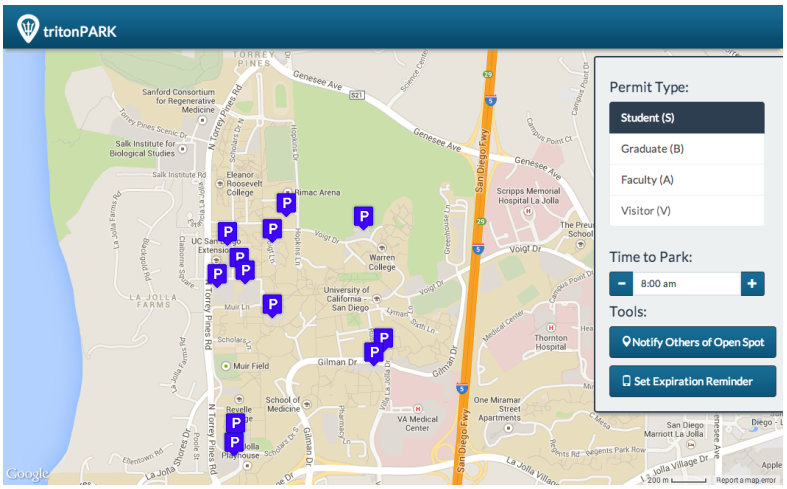

Fig. 5. Our final design is mobile, tablet, and web browser friendly. We ended up with a simple color scheme, blue, that ties into UC San Diego's school colors.

A stretch goal of ours was to color code the permit types (Student, yellow, Graduate, green; Faculty, red; Visitor, white) However, as we approached the deadline we thought our time would be better spent elsewhere, to ensure the functionality of the app.

1.11 System Development

TritonPARK is a highly modularized system. Given a 5 week development cycle, every component of our system is realized as it's own "vertical slice" in the overall system. This allowed our development team to work on each component in parallel. Simple mockups of the data were created to allow for components that needed to interface with the database or other components to be tested against something, even though the actual component may not have been ready for use.

Located below is a diagram of system architecture (Figure 6). TritonPARK is implemented with a dual server setup, and a completely detached front-end interface. This architecture design was implemented with a focus on future expandability, and for utility in the event of loss of network connection. We can easily replace the database endpoints with a more in-depth datastore with further historical data, or should a real-time parking spot availability system ever exist at UCSD, with live data from that system.

With this architecture design, loss of network communications does not render the app useless. The app caches data locally if it's ready for display. Further, the parking lots remain cached as well, allowing the user to still locate a parking spot. With the loss of network connection, the app will not be able to schedule a text message notification for parking permit expiration, nor send real-time broadcasts to other users of TritonPARK.

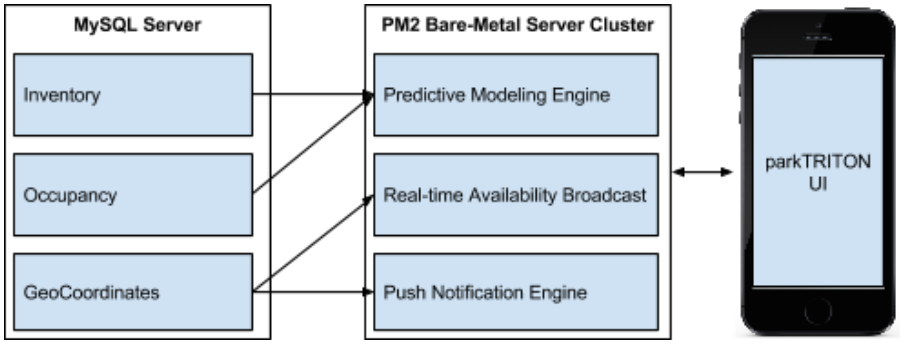

Fig. 6. Our Application Architecture Diagram

Our architecture design means that we expose many endpoints for our app to use. Further, we have designed our various API endpoints to be highly flexible in terms of client access, so that our API does not limit our client nor any others artificially. If we have the data, we provide a means to access it with varying degrees of information specificity. A diagram of these endpoints is shown below (Figure 7).

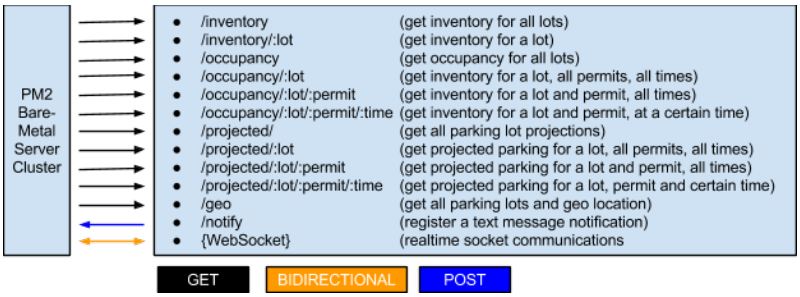

Fig. 7. Endpoint Directivity Diagram

TritonPARK is the culmination of several hundred lines of code, many of which are wrappers around external API's, data sources or technologies. Our architecture and implementation makes TritonPARK a highly modular system, and one that exposes many API endpoints. In industry, this concept is called "dogfooding". We're implementing a client on top of our own API, and would expose the same API to others should we ever decide to release a UCSD Parking API.

MySQL Database

A MySQL database was used to store data about each lot and availability. Separate tables were created for parking lot inventory, parking lot occupancy, and geolocation information for each lot. MySQL was selected because it is a table-based, relational database. The original data was organized in a table-based format, so storage of data in this system was much faster, and logical for programming than a NoSQL, document-based database such as Mongo, CouchDB or Redis. Further, the requirement of a strongly typed, and static database schema forces a coherent organization of data.

MySQL also has an extremely powerful and time-proven system of "JOINS", which allow us to combine the tables of data we have in various formats for use in the application. This ability to JOIN tables minimizes the amount of extraneous data returned in an API call; while allowing the ability to combine multiple data sources if need be. By moving data combination and processing into the backend database server, we can keep our front end user interface responsive, and minimize data communications. Both of these abilities are imperative for TritonPARK users, who will most often be located on mobile devices, with limited processing abilities, and limited data plans.

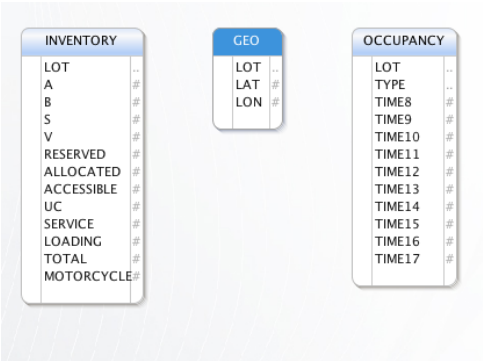

Fig. 8. MySQL Schema Diagram

To the right, is a visual diagram of the MySQL Schema. As one can see, there are three easily dividable databases. This division, along with a strict MySQL schema, forces data to remain cleanly organized, and allows for easy database normalization. While not originally implemented, it would be trivial to write a maintenance script for the database to normalize the data if we were to expend the effort to import data older than three years.

Twilio

Twilio API integration is used to power our user text message / scheduled notifications. This system was used, as Twilio is the largest cloud communications provider, and provided simple integrations for Node.JS. We were also provided with a dedicated phone number for TritonPARK, which allows users to save the number as a dedicated contact point for TritonPARK.

Twilio was selected over other providers due its proven stability and solid API. A simple HTTP request is all that's need to send a message to a user. During the TritonPARK development cycle, other providers were tested, but were shown to be unstable, did not provide API endpoints that were functional. One provider had their entire system crash for the span of three days during our testing and evaluation period. We otherwise would have gone with this provider, but the inability to provide a stable environment, and apparent lack of a proper development, staging and production pipeline for their product compelled us to look into other solutions.

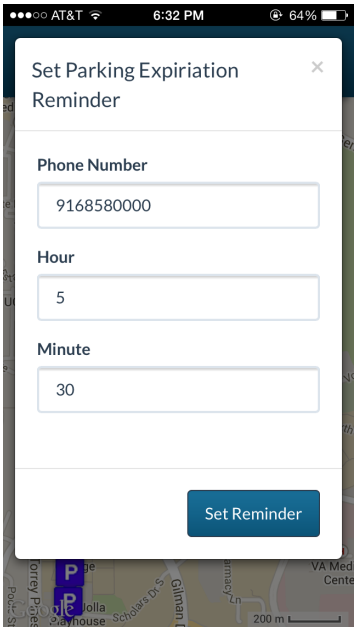

Fig. 9. In- App usage of the Twilio API

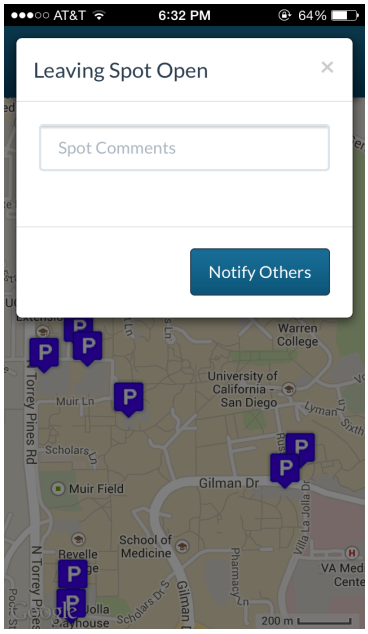

Fig. 10. In-App Usage of the Socket.IO API.

Socket.IO

Socket.IO was chosen as a transport layer for our real-time data component. While the majority of devices on the market today ship with, or can download, an HTML 5 compliant browser, the use of Socket.IO as a wrapper around HTML5 Web Sockets ensures that all devices still in use today can receive real-time open spot notifications. For users whose devices don't support HTML5, Sockets.IO will automatically fallback to a long-polling HTML socket / AJAX polling. Further, Socket.IO provides a comphrensive set of methods for directing messages, allowing us to control how to distribute real-time information. For example, we chose to only notify users other than the person who is leaving a spot open about the availability of a spot. In-app usage of this feature is shown above, in Figure 10.

Node.JS

Our backend server is written in Node.JS. This was chosen over other languages, such as PHP, or pure JavaScript, which are languages that we were otherwise familiar with, and debated using. Ultimately, the decision to go with Node.JS came down to four factors; programmer familiarity, non-blocking communications, high concurrency, speed, and memory usage.

Primarily, all three of our team members had in-depth familiarity with Node.JS. This was from both personal projects, this class, and from past experience. Node provided us with the least steep learning curve to traverse. The other languages considered, pure JavaScript, and PHP, had varying levels of familiarity, so would present differing levels of training required. We wanted to avoid time wasted on education, and prevent issues with uneven workload distribution. While pure JS and Node.JS are both JavaScript based, programmer knowledge of Handlebars, Express, and other Node.JS libraries promoted usage of Node.JS.

The factors of non-blocking communications, high concurrency, and speed are all imperative in terms of the final app. TritonPARK is an application intended for use by a number of users in peak traffic (in both terms of web communications and in vehicular terms). Therefore, it was important to ensure that all features were rock-solid and worked as quickly as possible. Node.JS is excellent for the non-blocking communications and high concurrency for running the Socket.IO communications used for real-time availability broadcasts, which are programmed in a way to be able to handle tens or hundreds of users hitting the server concurrently. Node.JS uses the V8 JavaScript engine from Google, so we know that it's built on a solid, lightweight, and high-speed foundation.

Finally, Node.JS' low memory footprint is critical in the choice of a server for a real-time web socket application. Other languages, such as PHP, would require a new PHP + Apache (or Nginx/other web server) instance for every long-polling client connected. This would require dedicated servers with a massive amount of memory. By using Node.JS, the overall memory usage is much lower, as requests are non-blocking, and concurrent. So while memory spikes may be greater for a Node.JS server, overall a Node.JS server will be able to handle many more connected clients per bare-metal machine.

Scheduler
The TritonPARK team wrote a custom event based scheduler for TritonPARK. This system, is used in conjunction with the text message parking permit expiration notification to send text messages at the correct time. Currently, this is the only functionality implemented, but it would be trivial to allow the app to schedule other functions as well. For example, we currently reset the real-time pins every three minutes client side, but with the scheduler, it would not be difficult to implement per pin based expiration in the application, and have it removed from all users at the same time. Functionality for times further than the current day are also implemented, but are not publicly exposed in the TritonPARK API.

UCSD Parking Data
UC San Diego maintains highly accurate inventory and occupancy counts for all of its parking lots. However, these datasets are maintained in an obscure location (http://parking.ucsd.edu/survey/surv.html), and stored in .PDF and .XLS formats. Furthermore, these datasets use complex formatting, with frozen rows, multiple sheets, and several layers of variable data selectability before actually getting to a number that might hold meaning to someone. All of these factors make for a less than satisfying experience in attempting to locate factual data for UCSD Parking. However, as developers, we were able to get this data converted into a format that is digestible programmatically, and from this, create a data view for users that is functional and easy to use.

Bootstrap
Bootstrap was selected as a front end UI framework. This was selected due it's powerful CSS library, and intrinsic design around responsive interfaces. This app needed to work across a wide range of devices, with varying screen types, resolution and sizes. Further, a CSS implementation of a user interface will be a less data intensive requirement than one implemented heavily with images. In our app, we use two images in addition to the Google Map tiles to display our interface. Everything else is produced purely in CSS. This design keeps network traffic from the PM2 bare-metal server cluster low, and minimizes data transfer over cellular networks, which is often bandwidth and transfer limited. Therefore, it is important to ensure that as little as possible required data hungry resources.

Seen below are screenshots of the app in desktop, tablet, and mobile views (Figure 11). Please notice how the user interface self organizes into its' optimal configuration based on the device used.

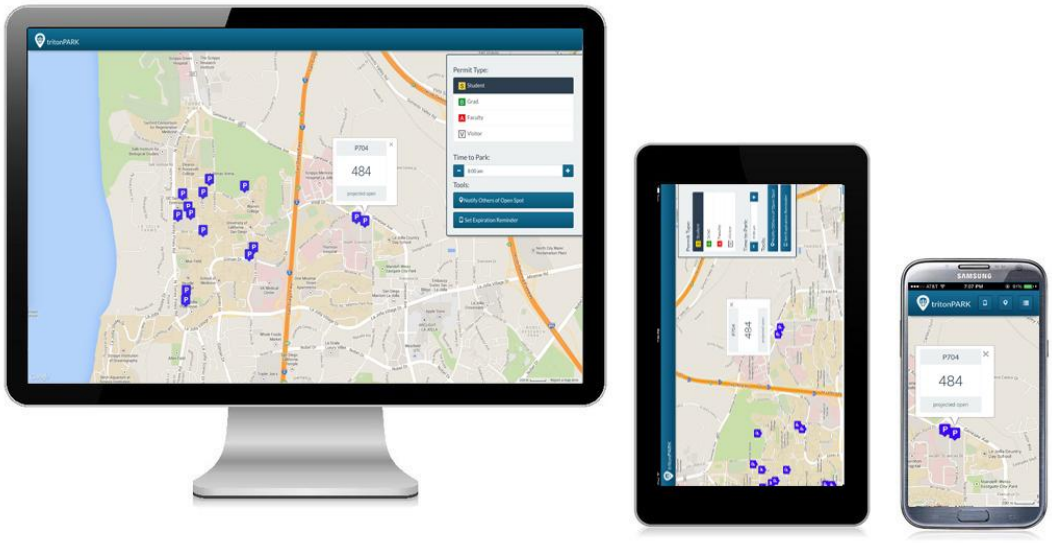

Fig. 11. TritonPARK presents a responsive UI, for all device types.

1.12 Features

Highly Detailed Map

One of our main features that the user will be interacting with the most is the highly detailed map that gives the user a bird's eye view of UCSD and its various parking locations. It improves on the existing system of simply displaying a Google Maps view of campus because we have integrated data and overlaid visual elements about the physical location of the parking facilities. The tags of the parking locations overlaid on the map help the user block out extraneous information so that they can solely focus on the task of finding parking around campus.

Set Expiration Reminder

Another one of our features is a text message alert system that sends the user a text message at the user's choosing. This was implemented to help users keep track of when their visitor's pass is about to expire, a useful feature since many people often forget to keep track of the expiration time on their pass. It improves on an existing yet rudimentary system of either keeping a receipt or mental note detailing the pass's expiration time. It acts as a type of assistant to user by sending him or her a friendly message since he or she is already carrying a mobile device.

Leaving Spot Open

The Leaving Spot Open feature allows users to drop a pin at their current location. Their current location is given by geolocation, which gets access to the latitude and longitude. If the user spots a available parking spot or leaves an available spot, then he or she can choose to notify other users of the location by dropping a pin. This pin's location will then be broadcasted through the application for other users to view for a brief period. Currently there are no other applications similar to ours that contain this type of functionality. Viewing how this feature works in real-time is a treat and can very useful to other users searching for parking in a similar area because the posting of the notification instantaneous.

Time to Park
The time to park feature is a useful feature for predicting parking availability around campus at a particular hour. Based on the user input the map portion of the application updates all of the parking locations pins to reflect how many spots are predicted to be available at the user's indicated time for a particular permit type. In other systems, this is akin to selecting different inputs and updating the system to reflect the changes. In this regard, our application is not doing anything special or improved, it is just gathering data and outputting a result of a particular query.

1.13 Application of HCI Principles

On a general level, the implementation of the user interface was straightforward since most of the CSS elements were designed for us using Bootstrap. During our implementation, we made use of a GitHub repository so that we could work remotely, collaborate, and make the appropriate changes to our application. We decided to keep things minimal with regard to the user interface. We wanted to hide elements from the user when they were not being used so that he or she would have an uninterrupted yet intuitive experience. The main part of the user interface that that the user interacts with the most is the map of the UCSD campus because it is here that the user will interact with the different pins indicating where parking structures and lots are located. For this we used a combination of Google Maps, geolocation, and JavaScript to update the various pin elements that appeared on the map. We chose these methods because we believed that Google Maps was an excellent tool already available with an extremely easy to use API. Geolocation allowed us to access the user's coordinates for use in the application when he or she would like to indicate that a spot is available. We also chose to use JavaScript in our application making it easy to update various elements that appeared on the page after the user made changes via input. We also believe that TritonPARK does a great job of adhering to the different human computer interaction principles as well.

Visibility
Upon starting the application the user is immediately presented with a large, detailed map, depicting the location of various parking locations around the UCSD campus. The map is a great way of making sure that the application orients the user geographically, and it helps frame the utility of the application. Because the application was designed to function within one page, the top navigation bar has all the tools the user needs to make modifications to his or her query or perform other tasks. When the different options are toggled, they appear on screen when they are needed and can be dismissed after the user is done using them. The navigation bar is kept on the page at all times for accessibility and it indicates the possible actions that can be performed. If the user needs to change his or her permit type, the system notifies them of the currently selected permit type by highlighting it. If the user needs to modify the time at which he or she wishes to park, there is an unambiguous interface, which lets the user view the current selected time and make modifications via the increment or decrement buttons. It is also clear that if the user needs to broadcast his or her locations indicating that he or she found or left an empty parking spot then the action can be done by inputting text and clicking a button that will notify other users. This is the right method of doing so because it gives the user the freedom to provide extra comments or details about a given space if needed while

performing its main notification function. Finally, parking locations are indicated by pins on the map that use the largely recognizable blue parking icon. By keeping the number of textual elements minimal and having a large focused uncluttered working area, we are able to give the user a very clear and uninterrupted experience.

Constraints

With regard to constraints, the user is not able to enter a specific time to as to when they plan on parking around campus because the data we've gathered is based on hourly estimates. This is why we've chose to have a simple time picker that allows the user to increment or decrement the hour and then have the map reflect the updated changes. The user is constrained to a few well-defined functions, but if an error were to arise in the case of misoperation, the system status would change to display an error page. Given these constraints, the complexity of the user experience is decreased immensely and the user cannot stray too far from the various functions we have implemented for them. The user experience is thereby improved since he or she will encounter very few errors if any during normal operation of the application.

Mapping

With regard to our button choices, the mapping is straightforward. If a user wishes to modify the time to park, first he or she will click on the clock icon at the top of the navigation page. Then he or she will use the increment or decrement buttons to increase or decrease the time by an hour. These buttons are indicated by a plus and minus sign respectively. The set expiration reminder button has an icon of a smartphone because notifications will be sent via text message to the user's phone. Finally, notifying others of an open spot occurs when the user clicks on the button with a location pin on it because after the user broadcasts a message, a location pin with appear on the map to other users notifying them of the broadcast location. The user experience is improved by making it as simple as possible for the user to recognize what buttons perform what actions.

Mental/Conceptual Models

Having the user be presented with a large map of the UCSD campus with parking pin overlays immediately gives the sense to the user of what he or she is viewing. It helps that because we've implemented this feature using Google Maps, we also have access to labels for different streets and buildings. This helps the user with his or her mental model of campus so that they can spatially visualize the relative distance and locations of the various landmarks around campus. In addition, the user is able to interact with this map by zooming in and inspecting various elements if needed. This makes the user experience very enjoyable and interactive so that locating one's self in the environment is never ambiguous and the element of frustration is eliminated from the experience.

Affordance

When presented to users for the first time, it is clear that our web-based application functions like any other iOS or Android application since it is self contained and includes input elements such as buttons and text fields. Simply by interacting with the map and moving it around on a mobile device through touch, the user gets a sense of how easy it is to manipulate the viewing space. Again, the buttons in the top navigation menu look like buttons and give an indication of what functions they perform.

Although the menus the buttons access are hidden from the user initially, it is clear that clicking on one of these buttons will cause the system to change in a similar manner to how any iOS or Android application would. The various touch input actions such as tapping, touching and dragging, and pinching to zoom, perform flawlessly on the mobile platforms giving the user a truly mobile experience. This was also included in one of our main goals since we expect users to be operating this application while in their car driving around searching for parking.

1.14 Testing and Evaluation

Testing our application for development was simply done by checking out the latest build from our repository and making sure that any added functionality was performed correctly. When laying out the various elements on the page we simply edited the frontend HTML and CSS and reloaded the page on our local machines to verify we achieved the layout we wanted. When it came time to test the actual features of our product we needed to make sure that the frontend elements were correctly linked up with the backend data and displaying the appropriate information. Parking pins were placed on the map by verifying their latitude and longitude coordinates using Google Maps. Moreover, the information they display when a user clicks on them was verified with our database schema to ensure they were retrieving the correct data. This was done by logging into MyPHPAdmin, viewing the data tables, and ensuring that the data on the number of parking spaces available at a particular parking location at the user's inputted time matched up to what was being displayed. Testing the user broadcasting system was done by setting up two instances of the latest build on the local host and setting a broadcast on one instance and double checking to see the location pin was displayed on the second instance of the local host. After setting up Twilio in our application, testing was done by simply verifying that text messages were sent to our phones. Further, Drone.IO was used to continuously check and deploy the builds, in addition to manual verification of application functionality.

As far as user testing and evaluation, we conducted a few simple tests with a small group of five students at UCSD who commute to campus. We gave them the application to use for a brief period and instructed them to explore its features. All of the users commented on how the design of the simple user interface was visually appealing. Users quickly gathered how to change input through the top navigation menu and were easily able to change their permit, change the time of day when searching for parking, and broadcast their location of an empty parking space. After changing the time to park field by either incrementing or decrementing the hour, three of the five students were confused as to what effect their input had on the system until they went back to inspect the details of the parking pins again. This was valuable feedback for us as in the future we plan to implement a system status that indicates that the system has been updated and the reflected changes to parking data is ready to be viewed. Moreover, until the users clicked on the third button to change the permit type, it was unclear for what permit the data pertained to. This applied to all of our test users. Having a simple overlay indicating the "current permit type selected" would help us with this issue. The limited number of functions meant that the user did not struggle to recall how to access a particular function. In addition, they described that it was useful that the buttons on the navigation menu gave some hint of their function. Since the web-based application looked like any other mobile application, the users also knew how to operate it without much help. Text message alerts were also successfully

sent to their phone via the application. The users were unclear if this was a real-time model, so we had to describe to them that this application is used for predictive data modeling only; however, the application can be used to broadcast locations of available spots and send permit expiration alerts via text message in real-time. It was in these two features that the users found the most utility, but the application did give them a very general idea of where their best chances for finding parking would be throughout the day.

To expand on user testing we would also need to observe how people with different permit types would operate this application. For example, although we only had a group of students test drive the application, it would be beneficial for us in the future to hand the application over to a graduate student, member of faculty, or visitor to UCSD and view how they might act differently with our project. Due to limited time constraints and the majority of complaints about parking being voiced by students, we decided to focus more on their perspective for this initial round of testing.

1.15 Organizational Tools, Rules and Procedures

As a development team, we are an extremely busy group, so ease of communication and a strong ability to organize was required. To this effect, Yasmine served as our Project Manager, ensuring that we had frequent meetings, and made sure the team was organized and aware of deadlines. Gregory served as the Lead Developer, and guided the software development effort. Jaysen served primarily as the liaison between COGS102 and COGS121. Our team used several tools to better organize our development. We'll shortly describe each of these, and what function is used in our development systems.

Private Facebook Group

The first thing we implemented was a Facebook group. All the members of our development team are hooked into Facebook via smartphone or laptop most of the day, so this ensured the fastest possible communications. Further, the Facebook group facilitated easy file transfers when we just needed to send over graphic resources or spreadsheets and didn't need to add to our Github file repo. This resource was shared with our 102c counterparts, and facilitated communication with them as well.

Group IM messaging

As with any large team, sometimes private communications needed to occur. We used Group and Private IM messaging to communicate. Largely, this was for communications that our 102c counterparts did not need to be apart of, so we spared the full team extraneous notifications.

Drone.IO Continuous Integration and Deployment

We implemented a continuous integration and deployment system using Drone.IO. (http://drone.io). This system allowed for the team to commit new code to the repo, and then have it automatically tested on an Ubuntu virtual machine before we pushed the code live. For a period of time, this deployment pipeline also pushed to our production environment; however, during the ending of our month long sprint, we decided to cut the pipeline after the integration step, and appoint someone in charge of doing the last push live on to our bare-metal server cluster.

GitHub
Like any good software development team, we utilized a centralized version control system. We selected Github due to it's integrated issue tracker, it's ability to hook into our automated deployment pipeline, and the far-superior interface in it's desktop app and online. In addition, all of our developers were familiar with the tool, and it's particular differences from a standard Git setup.

1.16 Issues & Solutions

Lack of market research until late into development cycle.
One of the issues that came up was when Greg discovered that the application that our COGS102C correspondents wanted to create already existed in a web page form. This program, called ParkUCSD, and located online, at http://parkUCSD.net was also a front-end interface for the same data source that our application was to pull from. With the late finding of this application, changing our idea entirely was out of the question. Instead, we took the time to play around with this app, and identify features that we liked and disliked, and user interface considerations that were both excellent and poor.

There were a number of details that we looked at; we liked the fact that this application color-coded its data, but without a uniform scale, the coloring was virtually useless. We, instead, made our app just display the raw number of expected open spots, and made this a prominent piece of info. Our users seemed to be more interested in a definite manner to make a determination on their own.

In addition, we liked that it was mobile optimized, but the application did not scale well to tablets or a personal computer interface, so this was a point we wanted to address. In addition, any information a user may actually want to see is located "below-the-fold", which does not make for an enjoyable user experience. Our application doesn't scroll, and instead uses modal popovers, and the interface drawer in order to keep the interface non-scrolling, and to only show the data that is necessary.

Further, the interface was pulled directly from the mobile theme pack from the UCSD Campus Web Office, and just applied without any real consideration for designing around the data presented. This interface is overly verbose, requires the user to scroll, and to read extra bits of information. For an application that is likely to be used in a vehicle, this was a huge concern, as the goal should be to get the info you need quickly, and to continue driving.

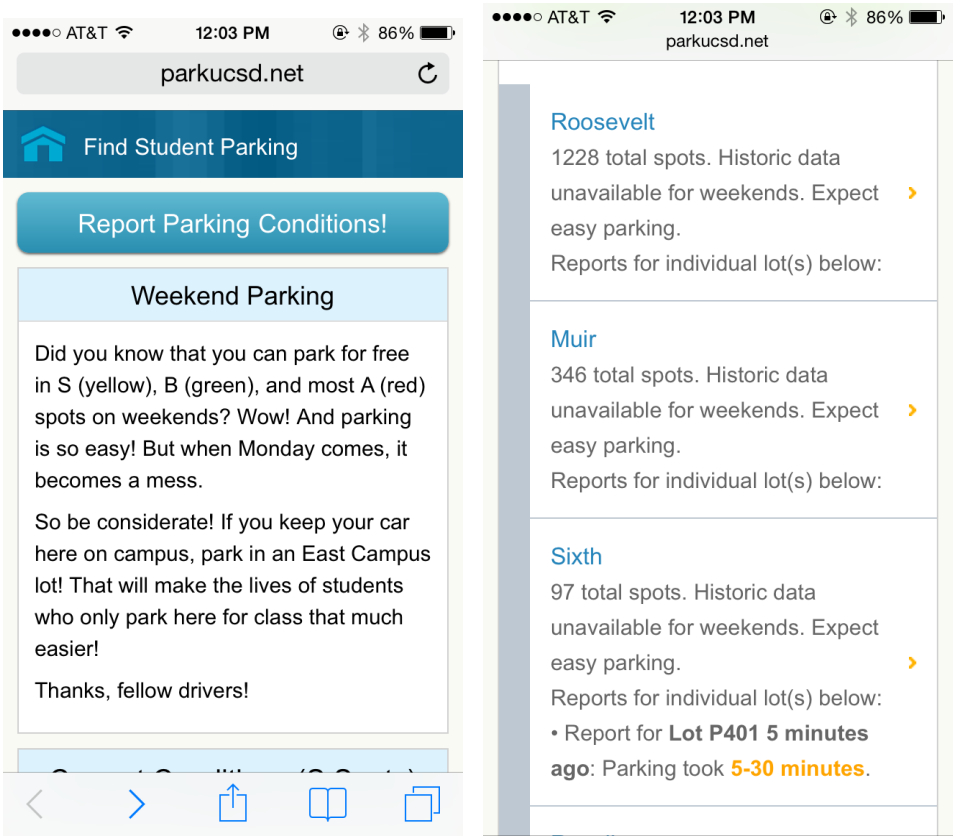

Fig. 12. Presentation of the ParkUCSD App.

In addition, this design uses templates intended only for use by UCSD academic units and organizations; this project, is neither, and falls directly outside of licensing terms. We address this by using a similar color scheme to the UCSD official one, but do not use any UCSD licensed logos, or other branding elements to ensure that we do not falsely identify as a campus sponsored project.

Finally, the application uses Apache and PHP to handle its backend. This setup is resource intensive, and would not scale well to a greater volume of traffic without requiring a more extensive server setup.

Slow communications with 102c group

A small issue with our 102c group occurred where they did not seem to communicate very well. Communications would be short or non-existent, and often times they would be giving data to us past the point it would even be helpful to have. We addressed these issues by setting up a Facebook group, and centering all communications there. This ensured that all team members stayed in the same page, as we are all connected to Facebook on our computers and smartphones. Moreover, the 121 team started setting deliverables for the 102c to fill, and in turn, we helped with their interface design. Ultimately, this issue was solved by the 121 team serving as the steering group for the project.

1.17 COGS121 Group Collaboration & Project Division

Throughout our development cycle, our group kept in constant contact via the Facebook group we set up. In the Facebook group, we discussed meeting times, issues with development, assigned tasks to group members. When it was time to implement certain features the tasks were divided between Greg and Jaysen. Greg worked on the backend portions of the application and this included the time consuming task of getting the database set up by converting the data we obtained from UCSD Parking Services. The data set went back many years so most of it was irrelevant for the application's purposes, so sifting through the data to grab the relevant information took some time. Jaysen laid out the basic framework for the front-end elements according to what was discussed between 121 and 102c groups during the prototyping stages of development. Both Jaysen and Greg would alternate delivering the weekly one slide presentation in class detailing the progress made in development and the collaboration between the two groups. Yasmine was in charge of scheduling meetup times between the 121 and 102c groups in addition to making sure that the 121 group was on track to meet their goals. During meeting times she set the agenda for the day and would detail what would happen next in order to ensure steady progress up until the final delivery of the application. Overall, collaboration went smoothly and efficiently, and it helped that Greg was familiar with many of the technical aspects of implementation because his expertise helped the 121 group implement an application that met the 102c group's requirements and end user's goals.

Gregory Marchese

- Real-Time notification of open parking, both frontend and backend.
- MySQL database schema design, information upload and normalization
- Text Message and Scheduling API
- Predictive Modeling Engine implementation
- Icon Design
- Frontend Interactivity
- Weekly Status Updates

Jaysen Parmar

- Bootstrap UI interface layout
- One slide PowerPoint
- Low Fidelity Mockups
- Weekly Status Updates

Yasmine Kotturi

- Project Management
- Final write up bullet pointing and conversion to ACM format

1.18 Work Summary with COGS102c

Our team worked closely with our 102c counterparts to produce a final product. Our 102c counterparts had formed a strong network of potential users to determine a guiding set of ideas that we were tasked with implementing. Throughout the process, design implementations were shared between both teams, keeping both groups up-to-date in our respective development cycles. This meant that the

various testing procedures that 102c went through were often either directly applicable to or often directly representative of our implementation. This close-knit collaboration ensured that most of our work-product was directly used, and minimized wasted resources, particularly in time and effort, both of which were severely constrained due to our status as students, with full workloads to share with other obligations.

Our workflow was as follows: we would meet once a week with the 102c group on Wednesday evenings for two hours. We would initially work separately on our own assignments for the half an hour finishing up any loose ends before moving on to working together and discussing our ideas for the TritonPARK application. The 102c group would talk a lot about features that they would love to have in the application based on their user research; however, due to time constraints and technical limitations some of their ambitious ideas had to be scaled back so that we could implement an application with a clear focus and well defined user.

Moreover, communications between both groups continued outside of the allotted meeting time using social media and Google documents. The 102c group shared with us a highly detailed document with links to all of their research and findings. This helped us immensely as they provided us with the relevant data we needed to implement the predictive modeling scheme that appears in our application. The user feedback they gathered helped us better understand our end user and address their frustrations with the parking situation on the UCSD campus. If our team had any doubts or questions, we would refer to the 102c group's document to check for any relevant information before proceeding to ask the group members for clarification. Relations between the two groups remained friendly, constructive, and helpful throughout.

1.19 Current Applicability

As the application currently stands, users will gain a better understanding of how many parking spots will be available for their selected parking permit at different parking locations throughout campus. They can also see how the number of these spots varies based on the time of the day. Additional features include the user's ability to broadcast their current location if they wish to notify other users that they are leaving a parking spot open or if they have spotted an empty spot. The user can also remind themselves via our integrated text messaging system when their visitor permit is about to expire. *TritonPARK* is a great tool that helps the frustrated students, graduate students, faculty, and visitors of UCSD help locate their best chances at finding parking, a problem that plagues the campus due to the administration's poor planning and lack of infrastructure.

1.20 Future Applicability

Our app was designed with future expansion and module replacement in mind. Because the system was designed and implemented in a very separated way, and only combined into one application at the end, we could easily implement any additional features, or supplement our databases with further historical data, to increase the accuracy of our predictive modeling.

In terms of future development, we've separated our future roadmap into two categories. One set are ideas that would be easy to implement in terms of time, effort, and material costs, and the second set are those that would require significant investment in any one of these areas.

Tier 1: Current Roadmap Ideas

Include data that goes back further than the last 3 years.

We currently include data from the last 3 years. This was due to time/effort constraints, since the data was poorly layout in .XLSX files. Given more time, a shell script could be written to convert all of the data, and even insert it into the MySQL database.

Include Historical Inventory Data

We currently use only the current parking inventory amounts. However, small amounts of reallocation to certain permit types occur every year, so these numbers introduce a small error into our predictive modeling. However, given the nature of parking, this error is small enough to be negligible when compared to the highly variable nature of locating a parking spot.

Increase the amount of parking locations selectable in our GeoLocation database

Currently, we have only the biggest parking structures in our GeoLocation Database. It would be nice to allow all parking lots to be locatable in our app, as that will greatly influence the ability to find a parking spot. Given additional time, a simple script could be created to lookup each lot using the Google GeoLookup API.

Implement day/night mode CSS.

While we do not condone the usage of our app in a moving vehicle due to obvious safety and legal risks, we do recognize that is highly likely that it will be used in this manner. To this effect, it is our job as UX designers to recognize this. Currently, we have a high contrast theme, with large buttons, and complete reduction of extraneous UI elements to achieve this. We could further this goal by implementing a selectable CSS theme based on whether it is day or night to further increase ease of information visibility.

Tier 2: Tasks that would require significant time, money or effort to implement

Implement GeoFencing, to allow the server to make smarter predictions of open parking, based on the users current direction of travel.

A GeoFencing layer would allow the server to make smarter decisions about open parking. For example, if a user is entering campus at any of the Torrey Pines Road entrances, than the app should not show any of the parking garages near Regents, until the user is nearer to these lots.

Use routed distances vs. point-to-point distance

Currently, we determine the distance to a lot based on the "as the crow flies" distance - a straight line. Given that this campus contains many winding roads, this is a mediocre method of determining the nearest structures. With enough GeoLocation data, we could implement a "routable" distance calculation, which would make the model of nearest garage actually reflect drivable distance, which in turn affects the temporal proximity.

Maintain a Universally Unique ID (UUID) based anonymized database of parking trends.
It would be relatively easy to assign a UUID to every device that uses our website, and to track trends across these UUIDs. If we could determine a users likely time to arrive and leave a spot, the aggregated data would give us an increasingly accurate amount of data with which to model parking patterns. Use of a UUID for every device would allow our server to maintain memory of a user's parking trends. Along with the geofencing, this would allow our server to process usage patterns along with just raw occupancy vs. inventory modeling, forming a better predictive model of open parking, for both new and returning users alike.

1.21 Future Relevance & Summary

This app is likely to be handed off or sold to an interested third party in the future, and provided as a service to those that park at UCSD. Based on this, it is unlikely to see this app provide a monetary return outside of the value of selling the codebase unless ads were to be implemented. Further, the app requires a server with a setup not currently found in any platform as a service setup with a free tier, and will require monetary investment into the technical infrastructure to support the app. While it would be easy to split the app into multiple servers based on the inherent system modularity, providing a stable server pool for an app that provides real-time broadcast to connecting clients would need a server array with serious processing capability during peak times to ensure that real time broadcasts are sent in a timely manner.

TritonPARK was the birth child of a short, highly sprinted development cycle. Our counterparts in 102c identified a need in the UCSD community, and in cohorts with our COGS121 team identified user requirements, system requirements, and the end-goal feature set. Over a 5 week development cycle, we were able to implement a highly-functional, rock-solid application, implementing predictive modeling of historic parking data, along with a real time parking availability broadcast system, a scheduled notifications system, and user GeoLocation; all of this is presented in a responsive, user-focused interface optimized for use in a vehicle. While our application may not do a million things, it was also never the focus. We wanted to focus solely on the problem, and do nothing more, and nothing less, to solve it. TritonPARK accomplished these goals, and in doing so, have alleviated some of the issues with parking at UCSD.

REFERENCES

(2014) *Continuous Integration - drone.io,* Available at: *http://drone.io* (Accessed: 8 June, 2014).
(2014) *CSS - Bootstrap,* Available at: *http://getbootstrap.com/css/* (Accessed: 8 June, 2014).
(2014) *Find Student Parking,* Available at: *http://parkucsd.net/?no_server_init* (Accessed: 8 June, 2014).
(2014) *GitHub Help,* Available at: *https://help.github.com* (Accessed: 8 June, 2014).
(2014) *gmaps.js - Google Maps API with less pain and more fun,* Available at: *http://hpneo.github.io/gmaps/* (Accessed: 8 June, 2014).
(2014) *node.js,* Available at: *http://nodejs.org* (Accessed: 8 June, 2014).
(2014) *Park UCSD - Find Student Parking,* Available at: *http://parkucsd.wordpress.com* (Accessed: 8 June, 2014).
(2014) *Socket.IO,* Available at: *http://socket.io* (Accessed: 8 June, 2014).
Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs (2009) *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 5th edn., : .
Jeff Johnson (2014) *Designing with the Mind in Mind, Second Edition: Simple Guide to Understanding User Interface Design Guideline*, 2nd edn., : .
Nadir Weibel (2014) *W2/Tue - Design Principles* , UC San Diego: .
Nadir Weibel (2014) *W3/Tue - The Art of Web and Visual Design*, UC San Diego: .