Parking Lot Monstrosity Specifications Document

Project Description:
Brown University maintains several parking lots on campus and the over-demand for parking space have resulted in long waiting lines to obtain a parking permit. Each parking lot has a waiting list and it is possible to be on the waiting list for several parking lots at a time. One never knows their status on any of the lines or an estimated time of approval. Recent statistics have shown that the waiting list for certain lots are measured in years.

This project will create a "one stop" centralized secure parking lot system with a graphical web interface. Users can see information about the statistics and locations of each parking lot, size of each waiting list, and their position on the waiting lists. New users can register and even decide to switch waiting lists.

System Model Diagram:

<table>
<thead>
<tr>
<th>Model Component:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Web Interface</td>
<td>The front end web system that the users will log into to access all user features. The user features include viewing current status on all subscribed waiting lists, changing waiting lists, adding/removing waiting lists, parking lot statistics and information, and an Estimate Time of Approval (ETA) calculation for each waiting list subscribed. The system will be written in PHP and connected to a MySQL database.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>CIS Login Security Check</td>
<td>The User Web Interface will send all login attempts to the CIS Security Check. The Security Check should send back a signal to either log the user in or deny access. This particular component is an external component that the CIS will provide to us.</td>
</tr>
<tr>
<td>Database</td>
<td>The database will contain all the parking lot and user data. This includes current waiting lists, parking lot statistics and permissions (user groups, access to certain lots).</td>
</tr>
<tr>
<td>Admin C++ Interface</td>
<td>This program will run on the parking lot administration computers. It will provide live access to the database. The admin can edit waiting lists, set permissions, update parking lot information, mass email users and add/remove users from waiting lists. The admin will also be able to look at statistics and history of each user/parking lot and edit the API System Queue.</td>
</tr>
<tr>
<td>Admin Web Remote Interface</td>
<td>The Admin Web Interface will provide a more basic list of features mentioned above. The purpose of this web interface is so that the administration can log into the system anywhere around the world in case of emergency. Most of the access and work should be done in their office. The web interface will be written with PHP.</td>
</tr>
<tr>
<td>Cron</td>
<td>A cron job must be setup to give the system its “heartbeat”. This will check the API queue in “x” time intervals to see if there are any queued jobs that need to be executed.</td>
</tr>
<tr>
<td>API System Queue</td>
<td>The API System Queue will contain all the jobs that are queued up to be executed including when. The admin can add jobs in the queue such as “Add/Remove this user from this waiting list next week at noon” or “Send out this mass email tomorrow/every week at 3pm”. This system will reside in the database but accessed through C++ or PHP.</td>
</tr>
</tbody>
</table>
**User Interface Diagrams:**

User main page:

```
Title + Big buttons to most popular features

News and Notifications from Parking Lot Administration

Graphs and diagrams of different parking lot statistics

Graphs and diagrams of different waiting list statistics

Side navigation panel

Current User Waiting List Status (Colorgenics)
```

User sub pages:

```
Title + Big buttons to most popular features

Action confirmation

Sub Page Information

Side navigation panel

Current User Waiting List Status (Colorgenics)
```
Admin pages:

Title + Big buttons to most popular features

Side navigation panel

Admin Information

Descriptions:

User Main Page

The box on the top will show the title and popularly used features that will be shown in large easy to read buttons. These buttons would include features such as “Join a waiting list”, “Edit waiting lists”, “View waiting list statistics”, “View parking lot statistics” and “Help/Contact”.

The left side navigation panel will list all the features clients have access to. The list will be divided up into categories. The list will also have the ability expand and collapse individual categories.

The right side will contain the user’s status of his subscribed waiting lists. “Colorgenics” will be the form of data representation. What this means is that we will not only provide the position (number) they are on for each waiting list, we would give a “temperature” as well. For example, if they are high up on the waiting list, the box will be “red”. If they are within only a few spots the box will be “green”. An ETA (Estimated Time of Approval) will also be provided.

The news and notifications located at the center top will display any current news, updates and alerts to the user. Administration should be able to send custom alerts to the
user through this box. The box can change colors accordingly. If there is an important message for them, the box could have a solid red border outline.

The parking lot statistics box will contain basic graphs and diagrams that show the status/statistics of the parking lots. (e.g. how full the parking lot is, how many parking spots, location, average waiting time). The users will be able to customize what data and graphs to show in this box.

The waiting list statistics will be similar to the above parking lot statistics box. Instead the basic graphs and diagrams will show status/statistics of each waiting list. (e.g. how many people in each waiting list, demographics, % of waiting list are professors/students). The users will be able to customize what data and graphs to show in this box.

User sub pages

The user sub pages will be very similar to the main page in terms of template and data. The biggest difference is the center column.

The action confirmation box will alert the user when data has been stored into the database correctly or if it has failed. Just say a user wanted to add himself to a waiting list. The action confirmation box will either say “Subscribed to waiting list successfully” or “Failed to subscribe to waiting list. Reason: XXX”. This confirmation is a re-assurance to the user that an action has been taken.

The sub page box will be the “meat” area. The main information will be placed here depending on which feature the user is accessing.

Admin pages

The admin pages will look similar to the user page template. The pages will only be compartmentalized into 3 boxes. Following the descriptions above, the layout is self explanatory.

Non-functional Requirements:

Performance:
Performance bottleneck is located at the database, since this is going to be a data driven system. Queries to the database must be kept to a minimum. A very diligent database model must be created to handle the traffic and efficiency. The database will be stored in a 3NF structure to allow for scalability and “grab only what is needed” queries to the database. The maximum number of users on at any given time may be at around 100 users. This is grossly over-estimated but since we are trying to develop a system that can be applied to any usage, we should be aiming to create a system that can handle 1000 users simultaneously.
**Testing:**
The system will be tested across the most commonly used browsers. This should be easy enough to test by actually going through each link with every browser to check for discrepancies. To test for performance, we could create bots that will connect to the system simultaneously and execute certain order. This way we could test just how much activity our system can handle before it locks up. To test for usability, we will get actual users with different demographics to test the application and fill out a survey. To check for data quality, we will enter all the data from the parking lot administration into our system and check if the queues and statistics are concurrent.

**Reliability:**
Assuming the above testing works, we ensure reliability by keeping the server that runs the database in good shape. We also must administer frequent backups of the database and find a way to restore a corrupted database as fast as possible. For secure logins, we should add SSL certificates which CIS may provide to us through their login system.

**Ease of use:**
Brainstorming should happen for every piece of information we decide to put into the system for the user and administration. We will pay particular attention to the navigation system, making sure the user can find everything want to access easily. Since this is an information intensive system, all space should be used up on the screen allowing enough room to view spread out navigation links and data. The best way to make sure we have a good ease of use is to ask for feedback from the Brown community. This is administered in testing.

**Portability:**
The system will naturally be portable since the access points are through web interface. Even the administrator’s control panel has remote web access. The most difficult issue would be moving the database. I assume that the database size would be huge and therefore not fit to be portable. The whole system could be moved but the time it will take is proportional to the size of the database. The rest of the system is just small text files.

**Documentation:**
Help links will be provided with every feature of the software. A small “?” icon will be placed next to the features. The user can click on it and learn what that feature does. We will also dedicate a section to do a full documentation of how to use the system and a FAQ. The administrators will get a similar documentation system but provided an in depth manual of how to use/update the system. The code must be commented well since we should expect other coders to be hacking our software to customize it to their needs.

**Dependencies on other systems:**
The dependencies are with the CIS login system. Also we will depend on the format of the MySQL database. The C++ code that we develop must be able to run on the administrator’s computers. This could be in windows, sunlab, or macs.
Updated Requirements:

Features

- Live Statistics (Priority: HIGH)
  - Number of spaces per lot
  - Number of cars allocated per lot
  - Size of waiting list per lot
  - User position on waiting lists
- Secure User Login (Priority: HIGH)
  - Login with Brown Auth-ID
  - Session management (Individual Priority: MEDIUM)
  - New user registration *Optional* (Individual Priority: LOW)
- Interactive User Controls (Priority: HIGH)
  - User can add/remove/swap on authorized waiting lists
- Administrative Control Panel (PRIORITY: HIGH)
  - Access live waiting lists
  - Edit live waiting lists
  - Set permissions on waiting lists (e.g. Students can only pick from...)
- Graphical Representation of Data (PRIORITY: MEDIUM)
  - Intuitive and effective bar/pie charts of "Live Statistics"
  - Use colors to represent magnitude of waiting list sizes
- Estimation tool (PRIORITY: MEDIUM)
  - Calculate estimated waiting time
  - Store waiting line statistics to improve accuracy of estimation
- Notifications and Alerts (PRIORITY: LOW)
  - Users can setup alert conditions to send an e-mail to them when the condition is satisfied
  - Option to email monthly/daily/semester/year on current status waiting line statistics
- User Status Priorities (PRIORITY: LOW)
  - Manipulate waiting lines to give priority to Professors before Students
  - Set balancing mechanism

Hardware and OS Requirements
- Program and database must run on Solaris in the Sun Lab, windows and macs

Web Interface
- Must be compatible across all browsers on all systems. Most common browsers are: Internet Explorer, Netscape, Firefox, Safari, Opera
- Can be accessed over the internet (public)
Risks:
A large risk we have is with the CIS login system. I am not sure if the CIS would allow us to access their system. If we are unable to, we must create our own dummy system that will authenticate users.

Also we must coordinate with the parking lot administration. This could prove to be a hard task. Generally people are not inclined for improvement unless necessary and thus involvement from them may be lacking.

After going through this specifications document, the project seems to be of substantial size! The challenge will be trying to create a complete fully functional perfect package. This project is very divisible since each area needs close attention and dedication. Now I am actually worried that we may not complete this project in the given time frame.

Challenges:
The biggest challenge we have is to try and make this project become usable in all situations that require a waiting list. This project must be usable by restaurants, ticket selling systems, hotel reservations, basically anything!

We can make this happen by:
- Compartmentalize/modulize everything
- Make everything robust and dynamic. Admin can change any settings they want easily. This includes the links of the navigation menu, colors, styles, add/remove boxes to layout, edit boxes.
- Great documentation
- Smart and cutting edge graphs and diagrams
- Good use of “colorgenics”
- A complete installation script and guide
- Able to connect to any security check by providing API to do so

This project is huge but could be do-able with 10 people.