SmartPark

Senior Project Report

Author: David Cosgrove
Advisor: Professor Slivovsky

6/12/13

Introduction and Project Charter

Project Overview

SmartPark is a system designed to supplement and/or replace existing paid parking infrastructure with a low overhead, easy to use system made possible with the use of smart phones and QR code technology. As parking infrastructure gets updated in recent years, municipalities are favoring more high tech, and often costly, solutions to the classic coin operated parking timer. These newer solutions have brought challenges to municipalities seeking to implement these 21st century parking meters, such as the high cost of upgrading to such an infrastructure, and security considerations for citizens, who often are paying for their parking with debit or credit cards, a situation vulnerable to information theft. SmartPark is designed with these challenges in mind and offers a compelling alternative to other modern parking systems.

Citizens that utilize paid parking infrastructure and organizations implementing said infrastructure are the two groups that SmartPark aims to serve. For the average user of the system, SmartPark should provide added convenience through smartphone technology that was simply impossible with previous solutions. Organizations looking to upgrade to a modern system would see a huge cost savings with SmartPark compared to other solutions - the system does not require any electricity or networking service from the side of the parking provider, and could be added to existing parking meters with stickers. The system would save approximately the same amount of money in upkeep costs as any other modern solution, with most savings coming from increased productivity from parking enforcement.

Goals and Objectives

Goals

- Create a system which allows users to pay for parking with their smartphone by simply scanning a QR code
- Provide a website for organizations and users to use to easily set up their own parking lots for pay

Stretch Goals

- Add improvements to the user experience by providing feedback to the user while they are parked
- Integrate location services for parking spots so that they can be placed on online maps
- Create a generalized “locking” infrastructure as a web service that can be utilized in other settings

Objectives

- Build database that can be accessed from a website
- Integrate website and database with the phone application
- Create registration system through website
- Implement QR code solution on mobile
- Test prototype system

Outcomes and Deliverables

Outcomes

The completed project will consist of a fully-featured website that allows users to securely create accounts and manage their own parking infrastructure. This website will allow users to create new managed spaces and automatically generate QR codes for them. It will not implement a payment service in this iteration, and will instead have simulated money amounts. The database will store sensitive user information and therefore will do so in a secure and up to date way. The app will run on Android phones and allow a user to scan a QR code and pay for their parking space.

Deliverables

- Mobile Application for a single platform (Android)
- Web server (Web site)
- User Manual

Duration

The alpha prototype was completed over the course of 20 weeks. The first quarter was used to familiarize myself with more modern web technologies, especially those pertaining to front end web development, and to NoSQL database solutions. Significant time was spent studying modern authentication standards and protocols in order to ensure that sensitive data was treated and stored as responsibly as possible. The second quarter was mostly spent implementing the website and backend from the ground up, starting from the parking space management system and ending with the mobile QR code detection development.

Formal Product Definition

Marketing Requirements

1. Development of a parking system to allow users to pay for parking via smartphone
2. Organizations or individuals can easily create and manage their own parking infrastructure, setting pay rates
3. User's sensitive information is abstracted away from individual transaction process
4. Users can pay for parking in more convenient blocks than traditional coin-based pay systems
5. No additional cost to the user for choosing to pay via smartphone
6. Needs to support a large number of parking spaces - should be appropriate for all parking use cases
7. Clean, modern look to website
8. Extremely slim hardware overhead - should use existing infrastructure

Engineering Requirements

Marketing Reqs	*Engineering Requirement*	*Justification*
1, 2, 3 Functionality	System will use MongoDB for persistent datastore	Lots of data needs to be tracked - fits document paradigm
1, 2, 6 Usability	Parking spaces have unique 12-byte identifier codes	Allows system to work at potentially massive scale, for all use cases
1, 2, 7 Usability	Responsive layout	Site will be used on multiple platforms of varying screen size
4, 5, 6 Functionality and Economic	Parking can be paid for in 1 minute increments	Doesn't cost any more engineering effort, offers major functionality upgrades over current models
4, 7 Usability	Mobile application will have one function and use case.	The process of parking needs to be fast and simple to understand for the user
8 Environmental	No new hardware should be needed other than printed stickers for meters	The system should have no negative impact on the environment
Legal	An intellectual property search will be conducted to ensure that there is no infringement on prior patents	The system shouldn't cause any infringement that could cause damage to the client
1 Functionality	A client constraint is that the system must be implemented on Android	Android is the only hardware I have to test with

Constraints

- System must be designed for use on a mobile device with a camera/QR code scanning capabilities.

Design Criteria

- Real-time parking payment - server updates immediately and is in sync with the user's timer when a spit is paid for.
- Full transaction logging in database so that later on, arbitrary views could be constructed for analysis.
- Compatibility with Android, and with existing internet browsers. The application should be able to communicate through Android as well as browser support.
- Parking space management functionality for admin users, allowing setup and creation of unlimited parking spots on demand, with unique QR codes that will not interfere with any previously existing spaces.

Design Research

Other things we researched involved some of the solutions to problems that came up during the design process. Listed below are the information on these systems and what problem they solve, as well as what drawbacks they include.

ZXing

http://code.google.com/p/zxing/

The goal was to create a custom ZXing-based client to decode QR codes, abstracting the technical knowledge of QR codes away. Ultimately an existing ZXing client was used on the mobile side, Barcode Scanner. The drawback to this approach is that it required a 2nd app to be installed on the user's smartphone before the SmartPark application could be used, but it did provide good results.

The User Experience

Use-Cases

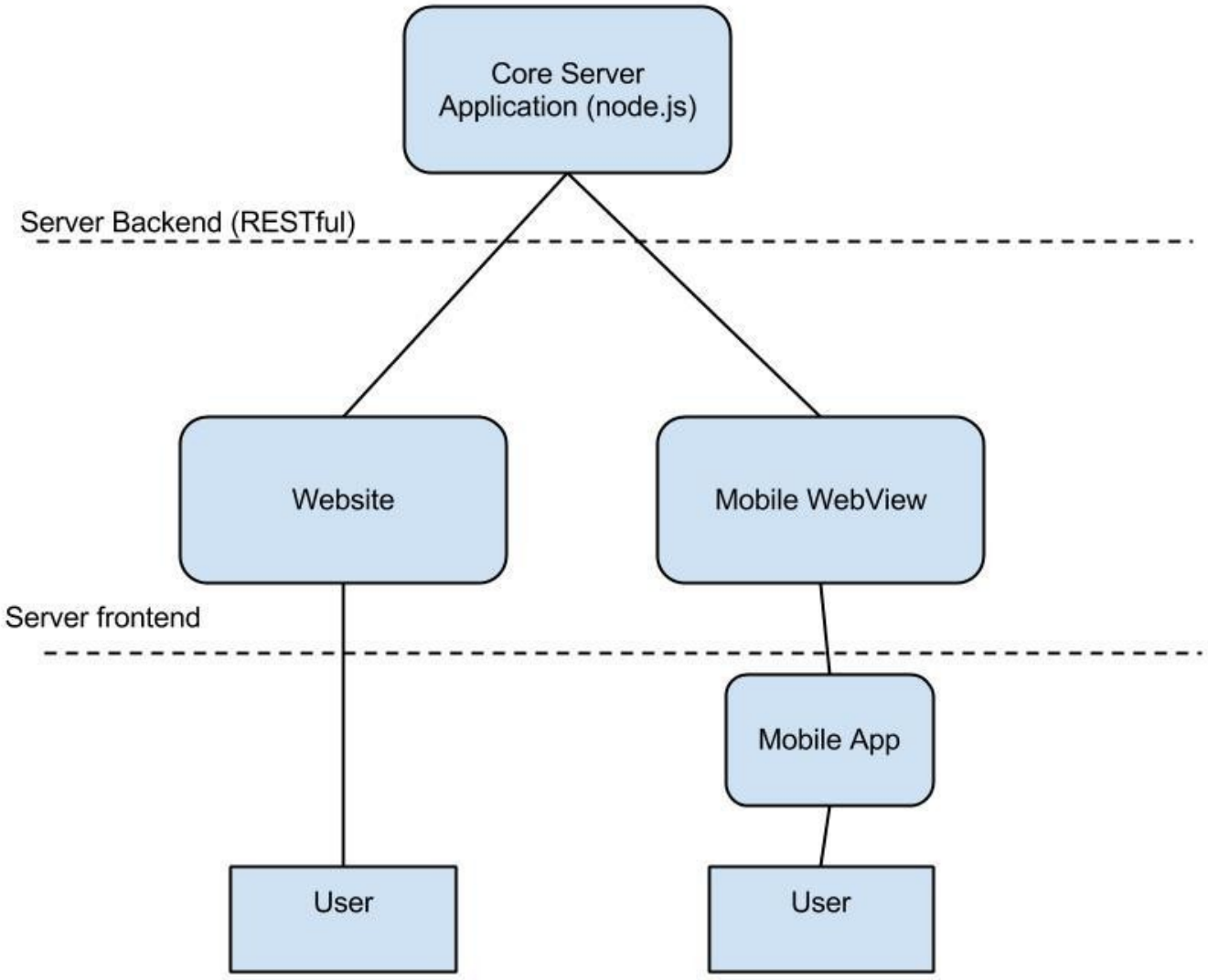
Based on requirements for the alpha prototype, several use cases were defined, split into two groups, the “Parking Admin” and “User”:

Parking Admin
- Create/define new parking spaces with address, stall numbers, pay rates, maximum durations
- View existing parking spaces, including their details and associated QR codes
- View transactions that occurred involving spaces owned by this Parking Admin

User
- Add funds to account
- Pay for a parking space via mobile application
- View all transactions that this User has been involved in

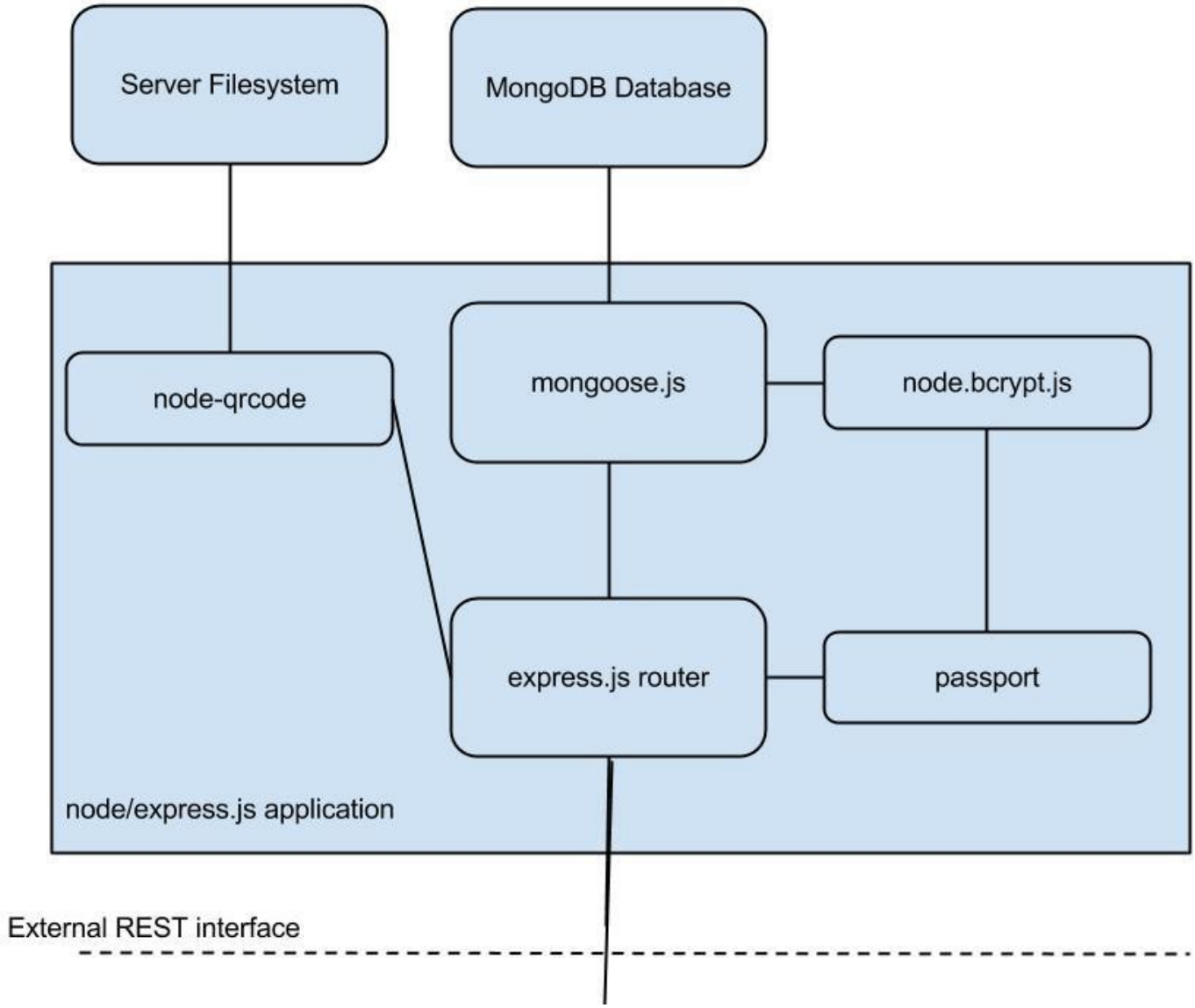
Design Analysis and Justification

High Level System Overview



The system can be viewed in two major components: The backend of the server, labeled here as “Core Server Application”, and the frontend of the server, which hosts both the website and the WebView utilized by the mobile application. The backend contains the core logic of the application, including all database interaction, user management, etc. The frontend represents User Interface logic, and the “glue” to connect to to the backend.

Detailed Backend Overview - Diagram



The above diagram depicts a more detailed view of the connections between the software stack used for the backend of this application. The entire backend is exposed through a RESTful interface, which connects directly to the express.js application router, which manages requests as they come in, sending them where they need to go. The backend uses MongoDB to store raw object data for Users, Transactions, and Parking Spaces. The QR codes are generated by the node-qrcode module of the express application, and saved directly to disk as PNG files.

Detailed Backend Overview - Individual Module Analysis

The following is a brief discussion of each of the components chosen in the backend application stack for this application. Each component's functionality is briefly described.

MongoDB - Document storage, "NoSQL" solution. My data fit nicely in a clustered document paradigm to the point that I felt a relational database(MySql, Oracle SQL, etc) would have just added unnecessary overhead. Some other NoSQL alternatives that I could have chosen are CouchDB or one of Amazon's proprietary services. I ended up going with MongoDB because of familiarity, and because the performance differences between the databases wouldn't be felt until I was up to massive scale. I utilize the 12 byte unique ObjectID functionality of Mongo to assign a unique, 12 byte code to each parking space in the system.

node.js - node is the platform that the application is built on. Node allows for asynchronous, event driven JavaScript to run on the server and serve HTTP - it is lightweight, and easily extended by using widely available modules that offer all kinds of functionality.

express.js - Express is a web application framework built for node.js that adds a multitude of features to the relatively barebone node.js platform. Features prominently used in my application include session support(allowing me to maintain a user's logged in state with cookies) and routing(making custom defined URLs call specific functions on the server). This also includes parameter parsing, elegant console output while the server is running, and more. Express is a core part of any node development stack.

Mongoose.js - This is a "glue" module that allows me to interface with my MongoDB instance from the node application. Mongoose also provides a strong modeling abstraction that lets me get away from the details of the database organization - Mongoose lets me create Schema and instantiate objects based on said Schema(similar to creating Classes in an object oriented programming language and creating instances of these objects). I simply create these objects, and then call a simple save() function which saves them to the database for me. This allows me to write clean code that utilizes design paradigms that I understand from my experience with object oriented programming. In short, it adds a structured, object-oriented way to interface with my database.

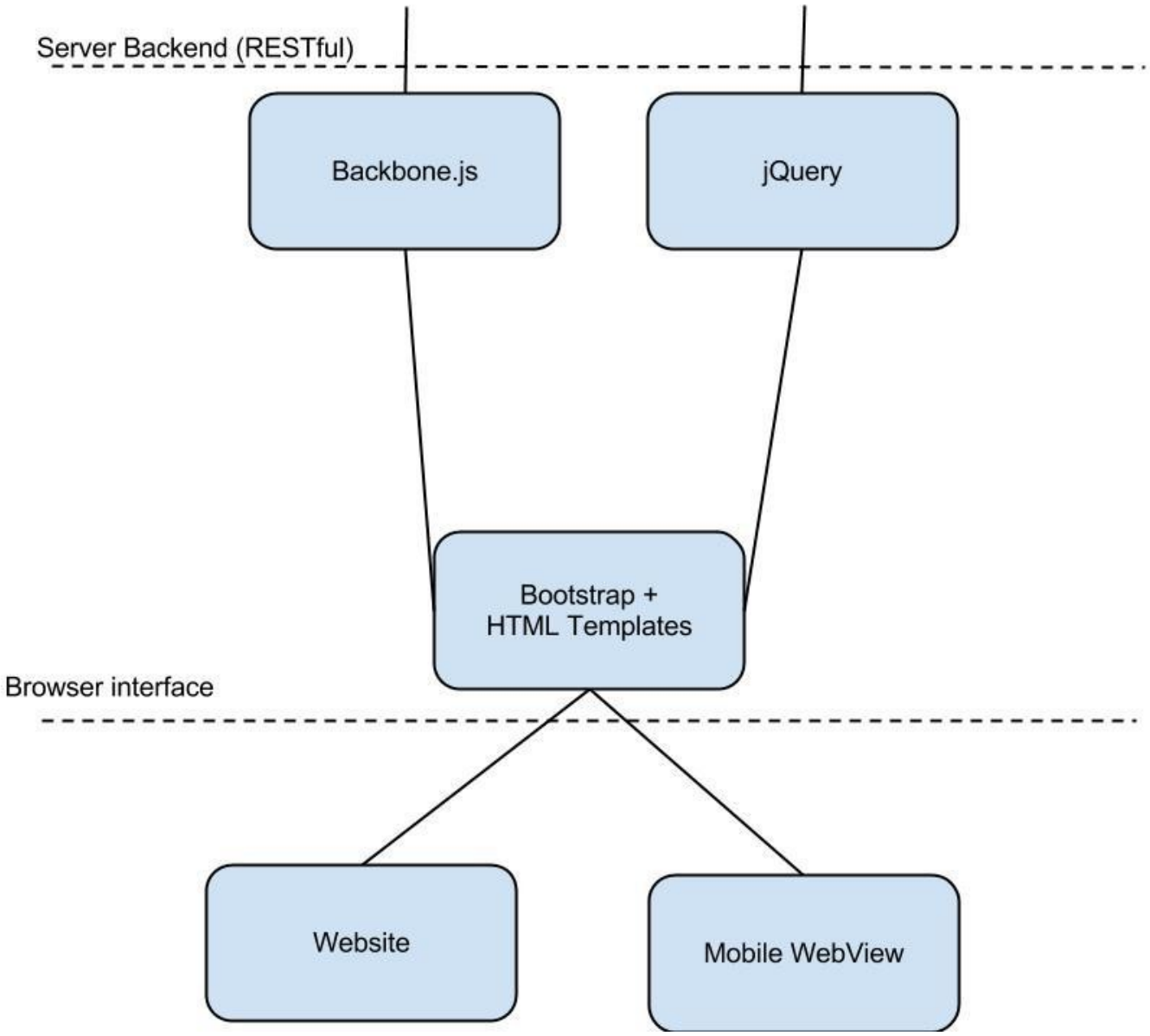
passport - This module is a special type called "middleware" for express - basically, it fits into the middle of my application stack and takes care of all authentication issues, ensuring that the user is logged in if they try to access any route that I have explicitly defined as requiring authentication. It redirects users to the login page if they try to access a route that they aren't permitted to use. It also takes care of session handling by working in conjunction with the session support from express.

bcrypt - This is simply an encryption module that I use to hash passwords based on the bcrypt algorithm, for secure password storage in the database. At the time of writing, bcrypt is the most secure way to hash data. The purpose of this hashing is to secure confidential data stored by my users. Hashed output from the bcrypt module is saved to the database via Mongoose. This module is used both when a user creates an account, and to verify that a username/password

combination was correct

node-qrcode - This is a library that allows for the creation of QR codes from arbitrary data, with multiple error encodings and other options available. Node-qrcode is built from a module called node-canvas, which basically allows node applications to draw arbitrary images programmatically. It can be used explicitly to draw and stream real time data to clients. For my application, there was a tradeoff of CPU time generating QR codes dynamically every time vs disk space in storing the QR codes permanently. I ultimately chose to save the QR codes as soon as I generated them to the disk, as they are extremely small(128x128 px). So, node-qrcode is only called in my app when parking spaces are created.

Detailed Frontend Overview - Diagram



The above diagram depicts the front end of the web application, defined as all of the code that is downloaded to the client machine that visits the website. A major point of interest here is that, because of my choice of front end framework, there is negligible time spent re-coding the site to function correctly with a WebView compared with a traditional desktop browser.

Detailed Frontend Overview - Individual Module Analysis

The following is a brief discussion of each of the components chosen in the frontend application stack for this application. Each component's functionality is briefly described.

HTML5 - All of the webpages are built with HTML5 + CSS + JavaScript for styling.

Bootstrap - Bootstrap is a powerful front end framework by Twitter that allows for streamlined development of web applications that can only be described as beautiful. It comes with a ton of custom CSS/JavaScript/HTML that replaces and extends a ton of the functionality that you get in basic HTML. It replaces all UI elements with aesthetically pleasing, slick, and functional upgrades, and adds many more UI elements of its own that fit well in many commonly used design patterns. It can also be used in "fluid" mode, which makes the website responsive(able to adapt in real time to screen size adjustments - suitable for a wide range of display sizes). This gives developers a fantastic starting point for making a great user interface for their websites.

jQuery - Industry standard JavaScript library used everywhere. jQuery allows interaction with the DOM, which basically means that elements on the page can be dynamically added and removed without reloading the entire page. I have custom jQuery code written on the registration pages to show error codes, and to update the navigation bar at the top of the website to display the user's login email.

Backbone.js - Backbone is a JavaScript library used to give the Model-View-Controller structure to your application. It offers a nice alternative to the ad-hoc, spaghetti code that front end development traditionally ends up being. It allows for structured, simplified, and reusable code that syncs your data models to the backend and takes care of the dynamic View adjustments based on changing Models(data). I chose this module specifically because it is made to be tied to a backend that exposes a RESTful interface, which is what I created, with little additional glue code. Backbone is used in my application on the Parking Space Management page to dynamically create and retrieve parking space data.

Mobile Application Architecture (Alpha Only)

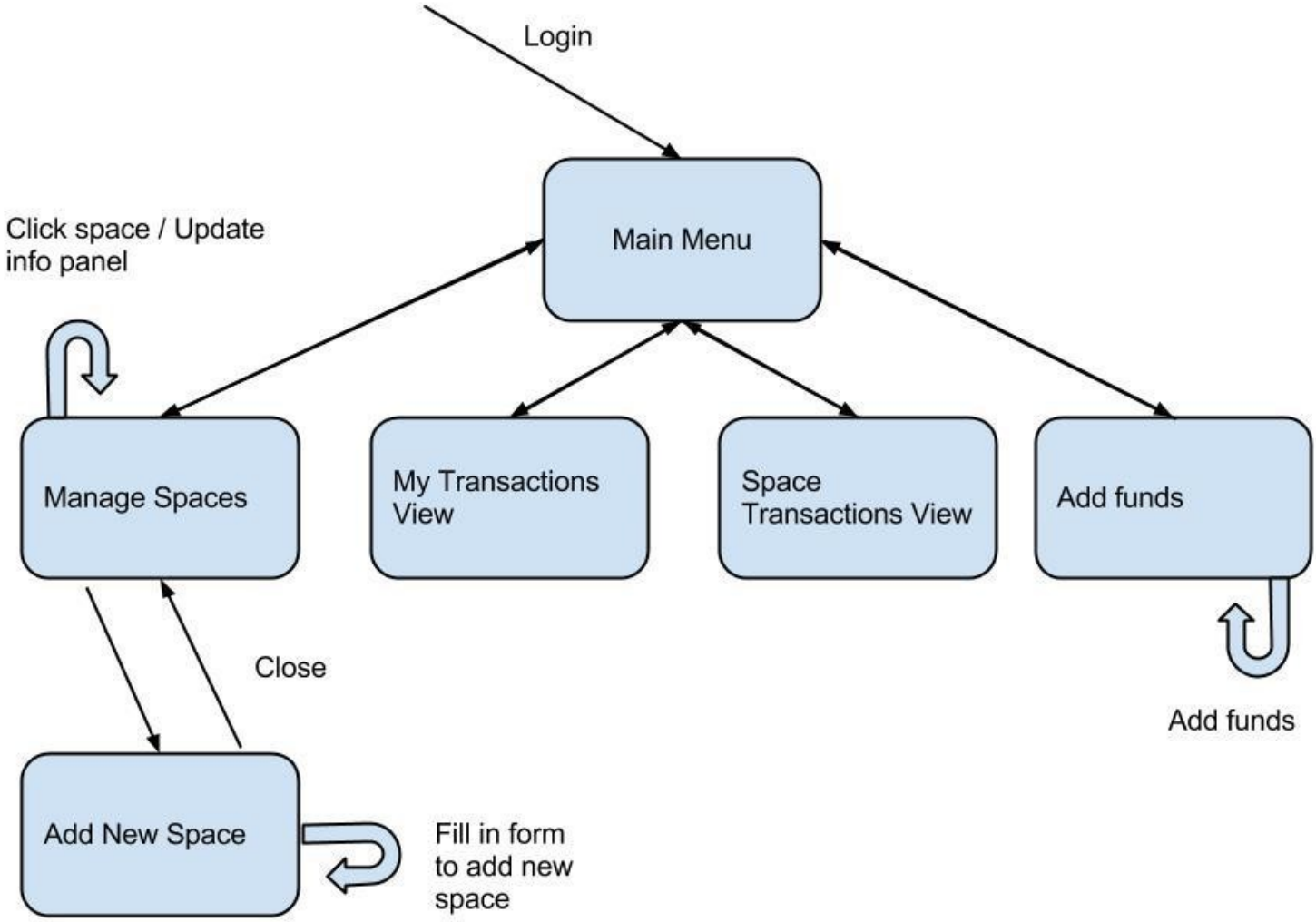
For the Alpha, the mobile application is simply a WebView that contains a command to boot Barcode Scanner, a native android app. Barcode Scanner scans a QR code, parses the contents, and returns it to the calling application, in this case my WebView. In my application, the QR code contains the unique ID of the space to be parked at. This information is used to generate the confirmation page for the space in question.

In the future, I would prefer to have a barcode scanning module integrated into my mobile application. The current Alpha solution requires downloading a 2nd app, Barcode Scanner, in addition to the SmartPark app in order to make everything work. I also have no control over the UI presented while running Barcode Scanner - before sending the code back to the WebView, it displays the unique ID on screen. This information is useless to the user and, at worst, confusing.

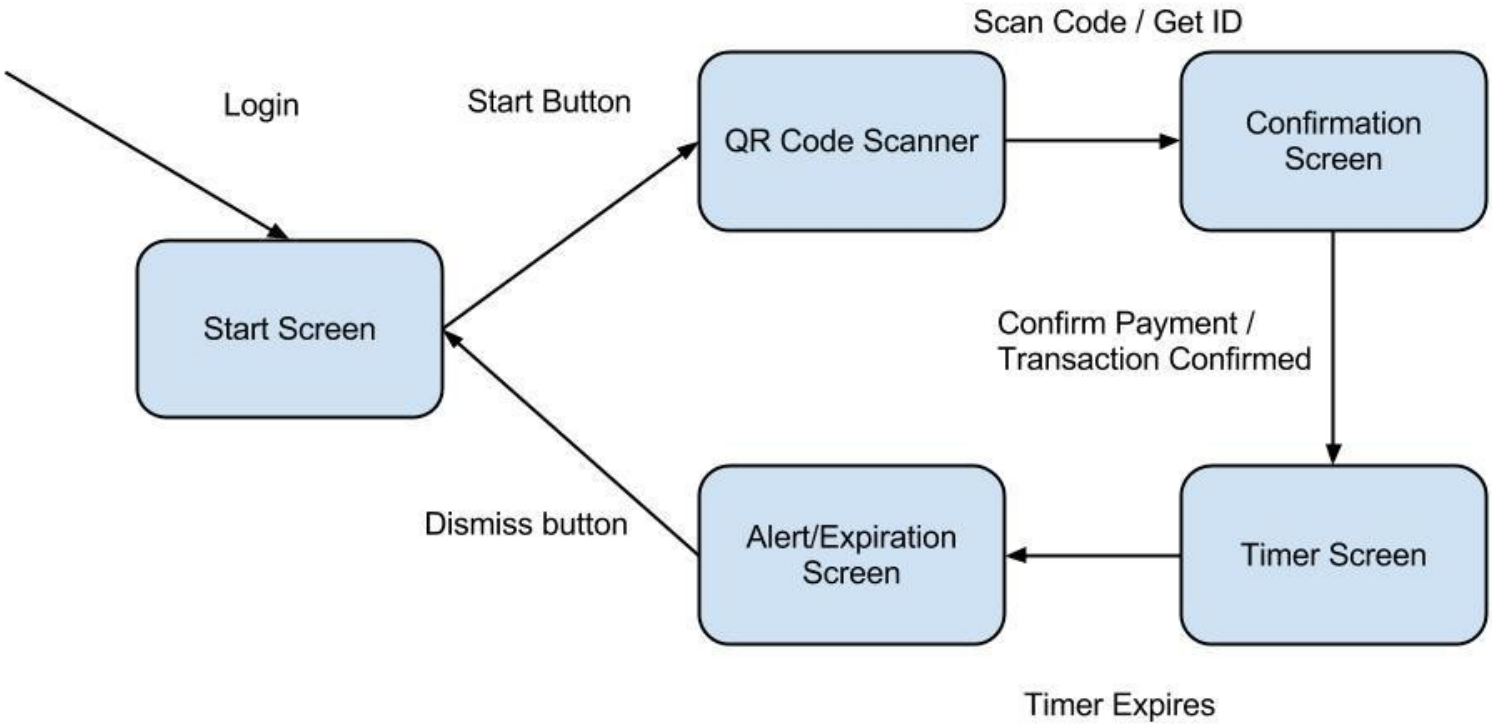
User Level Applications

The two user level applications are the website, accessed through a regular web browser, and the Mobile WebView, which can be accessed through a custom mobile application or through a mobile web browser(the latter technique was used in the alpha prototype). Their functionality is described through state diagrams. Both sites operate from an assumed logged-in state- the setup is such that, if any step is tried by a user that isn't logged in, they will be redirected to the login screen.

Website State Diagram - Transitions without text are hyperlinks to the destination



Mobile Application State Diagram



Testing

The system was tested from a high level functional perspective for the use cases described above. All components of the software are quite modular, and retrofitting with automated testing would be trivial.

The project is still in its infancy, with a finalized usage model that would be fit for a real-world implementation not close enough to being established. For the purposes of demonstrating the technology as a proof of concept, the testing of the established use cases was sufficient.

Future Improvements

- For the purposes of the demos provided, the application is assumed to have well formed inputs. Many forms do have error checking and proper sanitization, but there are some places where parameter enforcement is lacking - ie, emails are not checked for correct syntax upon registration. For a one man project, given the timeline, I felt that some of these features were unnecessary and do not change the quality of the alpha.

- Additionally, there is only one type of user abstraction in the system - the same type of account that creates spaces can be a client of another parking space owner. My next steps would be to split the user model more distinctly between “Parking Administrators” and “Users”.

- The next steps in this project, in order to push it to a commercial production environment, would be deployment automation and version control in order to log bugs. As this was a one man project, I relied mostly on localized version control software.

- As described in the details of the mobile application, the two-application approach using Barcode Scanner leaves much to be desired. More development time and focus would be needed in order to make the SmartPark app utilize the ZXing library as a direct client.

- All of the information necessary is properly tracked in the database, but I would like to add more views of this data - graphically, by space, by address, etc. This just requires more development time.

- Google maps API or similar GPS integration would be extremely helpful in visualizing parking spaces.

Conclusion

Overall System Analysis

The SmartPark Alpha is a strong proof-of-concept and foundation for the core mechanism behind a larger, more fully featured system fit for real world implementation. All use cases work properly, and the mobile phone functionality is streamlined and efficient, as I had originally hoped.

I believe that this project potentially has commercial viability.

Requirements Met and Unmet

As of the Alpha version of SmartPark, all marketing and engineering requirements have been met.

Appendix

Source code for the project is packaged with this document. In addition, it will be made available at:https://github.com/dcosgrove