Parking Application
CS 5551
Final Report

Final Package
Submission date: 12/12/18

Client: Dr. Lee

Fall 2018, CS5551: Advanced Software Engineering, Department of Computer Science & Electrical Engineering, University of Missouri-Kansas City

GitHub: https://github.com/toadSTL/CS5551_Team2_Project
Video: https://www.youtube.com/watch?v=qSaQYkjUIco
ZenHub: https://github.com/toadSTL/CS5551_Team2_Project#zenhub

Group Members

Ali, Muhammad muhammad.ali.ch@hotmail.com
Brown, Gregory gregory.brown@mail.umkc.edu
Kampally, Pravalhika pkqz7@mail.umkc.edu

Table of Contents

User Manual:

Introduction:

Finding a parking space is always a time-consuming process on the UMKC campus. There are many lots to check for parking, and no guarantee that any given lot will necessarily have parking available. Hence, we opted to create an app which is user friendly based on the information collected from users and from sensors fixed at the entry and exit of the parking. Installing this app on ones phone or accessing it on one's computer will give the user access to information about the availability of spaces per parking lot on UMKC campus as well as allow the user to submit information about the availability of parking spaces.

System Use:

As the client-side application is not yet hosted, or accessible via an app store, in order to begin using our application a user must first clone the repository hosted on GitHub at the following link:

https://github.com/toadSTL/CS5551_Team2_Project
After downloading the repository, the user must first run the server. This can be done either by opening folder "../Increment3/source/UMKCParkingAPP/Server/" within the user's preferred IDE and running Server.js, or by navigating to the same folder (directory) in the user's preferred shell and running the server with the following command:

```
>node Server.js
```

Once the server is running, in order to run the client side application the user must navigate (via their preferred shell) to the directory "../Increment3/source/UMKCParkingAPP". The client-side application can be run in browser via the following command:

```
>ionic serve
```

Or in android via the following command:

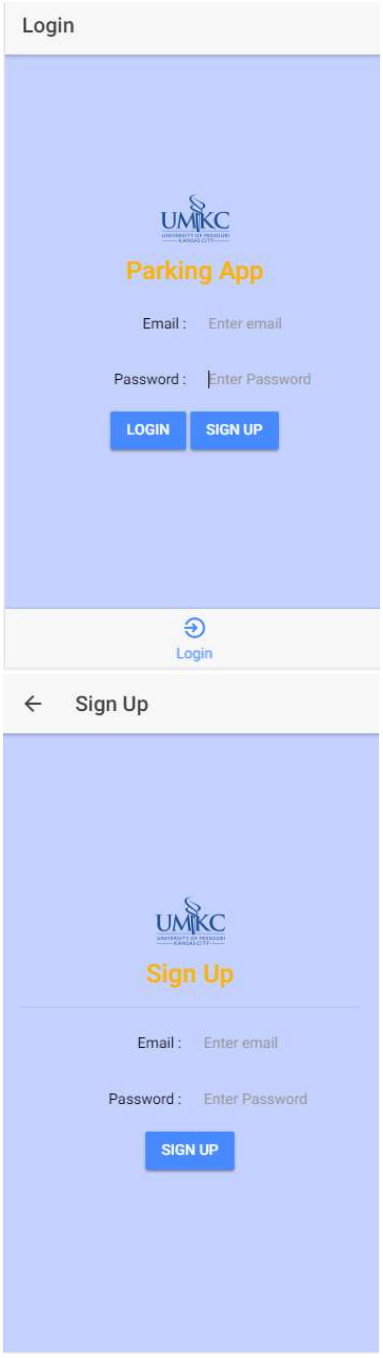
```
>ionic cordova run android
```

Once the client-side application is running, we hope that the UI is sufficiently easy to navigate but will nonetheless provide an explanation of its use. Our application has 4 pages: Login Page, Register Page, Home Page, and Submit Availability Page. On the Login Page, which serves as the landing page for the app, a user can input their email and the registered password associated with that email to login and be navigated to the home page. If the user has not registered, they should navigate to the Register Page and submit their email address and desired password. Once the user has successfully logged in and has navigated to the home page, they will see a map populated with markers, each of which represents a parking lot on the UMKC campus. Clicking or tapping one of those markers will bring up an overlay which shows the name of the lot and the parking information as well as two buttons. The 'Dismiss' button simply dismisses the overlay, while the 'Submit Availability' button will navigate the user to the Submit Availability page. On the Submit Availability Page the user can select the lot for which they would like to submit availability and input the number of spots which they believe are available in that lot, and this information will be incorporated into the space-availability information for the selected lot once the user presses the 'Submit' button.

Error handling:

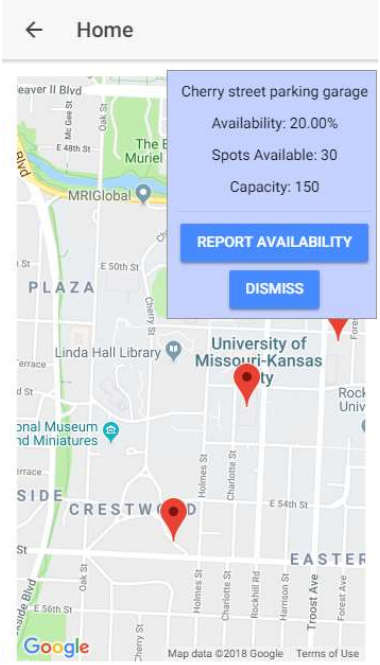
We are still experiencing errors both when running the application in browser and when running the app in Android. In browser there are ionic compiler errors which can be fixed, by reloading the ionic app. Running the app in android sometimes results in an Ionic error in which ionic claims that '@Ionic/app-scripts' appears not to be installed. We do not currently know a work around for this error.

Example Use:



- The first page a user will encounter when running this application is the Login page. This page is used to log into the service by providing their registered email and password.
- If a user has not registered, they should proceed to the registration page, by clicking the button labeled 'Sign Up'.
- For convenience the email `test@test.com` and password `testing` are already registered.
- Once the user has successfully logged in the user will be redirected to the Home page.

- On the Sign Up page a user can sign up for the service by submitting an email and a desired password.
- Clicking the 'Sign Up' button on this page will both submit the user's credentials to firebase (enabling the user to login with those credentials) and redirect the user to the Login page.



- Once a user has been directed to the Home page the user will see a map with markers placed on it. Each marker represents a parking lot on the UMKC campus.
- Clicking on a marker will show a pop-up containing the parking lot name and availability for that parking lot as well as two more buttons.
- Clicking the 'Dismiss' button will dismiss (hide) the pop-up.
- Clicking the 'Report Availability' button will redirect the user to the Submit Availability page where a user can report the availability of the parking lot if it differs from what is shown.

- On the Submit Availability page a user can report the availability of parking for a given lot.
- First the user can select the lot for which they would like to report the availability.
- Next the user can input the number of spaces they believe are available in the parking lot.
- Finally clicking the 'Submit' button on this page will both send the users report to the server to be incorporated into the parking availability information and redirect the user to the home page. (NB: the reported availability will not be incorporated immediately, and may take a little while to update, however it can be made immediate by refreshing the application and logging back in)

Issues and Deficiencies:

Both of the issues noted in the Error Handling section of this User Manual are substantial. Additionally, while our initial inclination was to use sensors to report the parking availability data to the server, our lack of IoT programming experience precluded us from completing this process. We do have an simulated Passive Infrared sensor which can provide dummy parking availability data to the server. This simulated PIR sensor can be run by opening the "../Increment3/source/SimulatedPIRSensor/" in a user's preferred IDE and running app.js or navigating to that directory in a user's preferred shell and running with the following command:

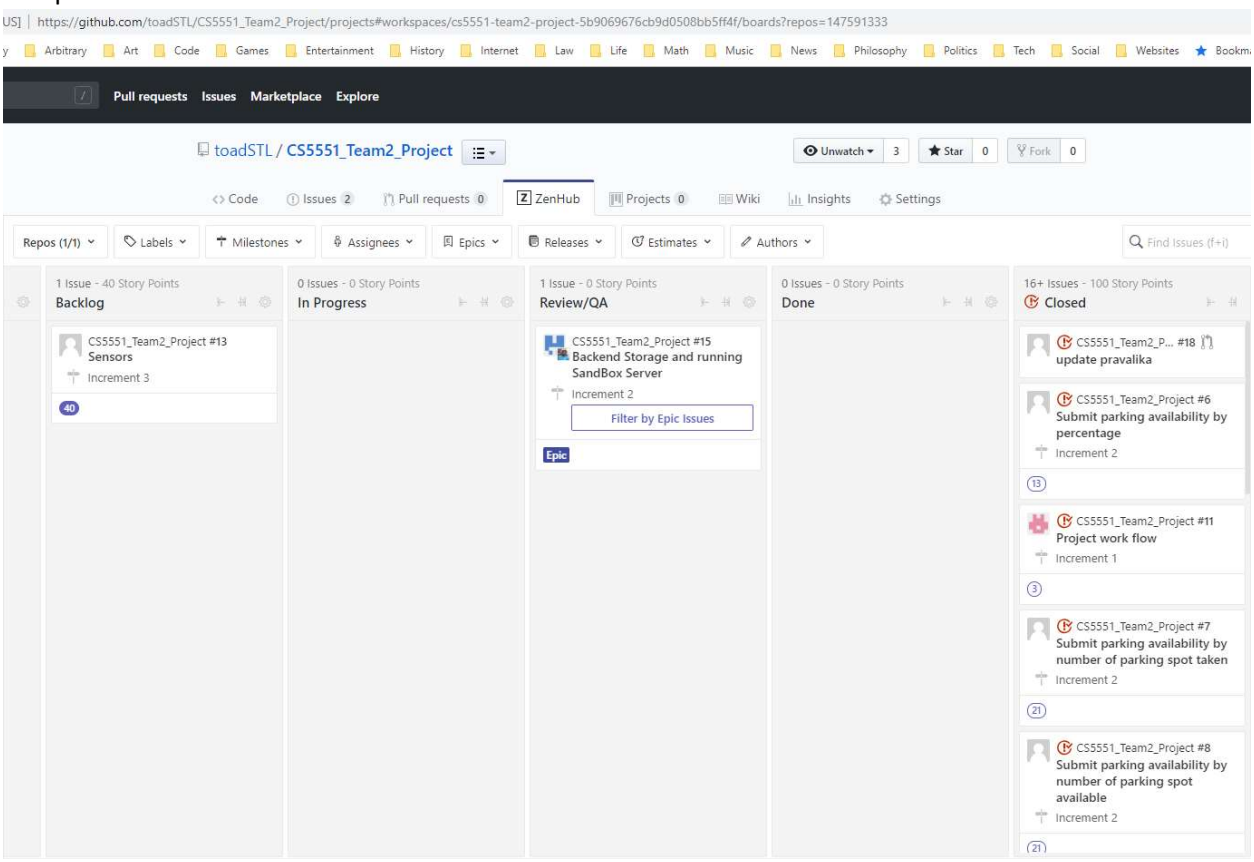
```
>node app.js
```

Once this server is running it will provide dummy data about changes in the availability of parking space in lot 1 (the Rockhill Parking Structure) to our MongoDB. This information, however, is not currently incorporated into the data about the availability of parking in that lot which is used by our client-side application. This is a very substantial deficiency.

Other deficiencies of the client-side application we created, compared to that which we aimed to create, are that our application does not offer a user the ability to Reserve a parking spot, does not offer recommendations, and does not include a search feature (to search for availability parking spots).

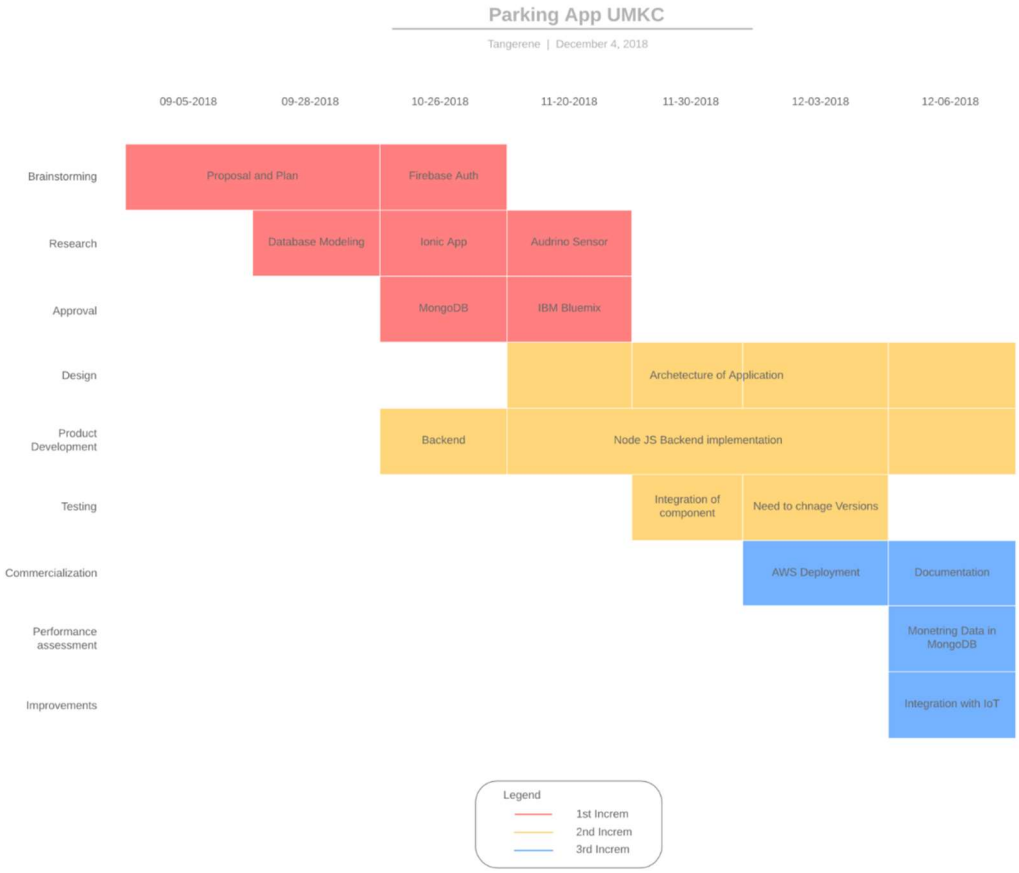
Project Management Report:

For this project we attempted to us the Agile methodology for software development and in some senses were successful. Specifically, our work was incremental and iterative and involved review meetings in which we discussed what had been done and what needed to be done. Additionally, we divided the work as equally as possible among ourselves. We did this by creating issues using the ZenHub tools. Below is a screenshot of our final ZenHub issues tab showing issues which have been completed in the 'Closed' tab:



For the most part, using this methodology enabled us to divide the work, make progress, and re-assess our goals for the project as we worked on it, making our goals more achievable to us. Nonetheless, as the ZenHub screenshot shows, we are still reviewing the work Muhammad did in terms of deploying our backend, and we have not completed the integration of sensors which we acquired and worked with for this project.

Additionally, we have included a Gantt chart of the progress made on this project below:



As this chart shows, we had a large backlog from increment 2 into increment 3, partially to do with our team made (David) dropping the class and ceasing to work with us.

We used Slack as well as email and phone to communicate about project work and met fairly often, but not regularly in order to coordinate and work together towards completion of issues for each increment. Our communication over the course of this project was decent. Meetings with our TA, Ruthvic, for this project were very helpful, and during our final meeting he actually helped us with debugging a particular issue we were having. Additionally, both Ruthvic and Dr. Lee provided helpful suggestions on technologies to use and how to use them in the implementation of our project.

While we each worked on various parts of this project and sometimes worked together below is an outline of generally-who did what:

- Simulated PIR Sensor (Node.js, Bluemix, and Node Red) - Greg
- Submit Parking Availability Page – Pravalhika
- Login and Sign Up Pages – Greg
- Firebase Authentication – Pravalhika
- UI (HTML and CSS) – Greg and Pravalhika
- Home Page with Google Maps API usage – Greg and Muhammad
- Server (connection between client app and MongoDB) – Greg and Muhammad
- MongoDB – David & Greg
- AWS deployment - Muhammad
- Actual PIR Sensor system * - Greg

- Reservation page * - Pravalhika

Portions followed by and asterisk * are incomplete.

Provisional Point Distribution (Including David Oh - David dropped the class immediately before the end of Increment 2):
Greg Brown (Class ID – 5): 33
Pravalhika Kampally (Class ID – 22): 31
David Oh (Class ID – 36): 5
Muhammad Ali (Class ID – 61): 31

Official Point Distribution (not including David):
Greg Brown (Class ID – 5): 36
Pravalhika Kampally (Class ID – 22): 32
Muhammad Ali (Class ID – 61): 32

Implementation and Project Management with time:

The group meets on Friday's at 3:30 and the meeting ends after 3-5 hours, depending on the tasks group have been assigned by themselves or by the group manager that week and the amount of collaborative work needed for those tasks. There may be more frequent meetings as needed, and subgroups will likely meet outside of these group meetings. Any communication between the team members will be done through an application called Slack, which can be accessed on both web and mobile platforms. The project discussions and meetings will be performed either virtually or physically depending on the feasibility of team members' schedule, the difficulty of the current tasks, and the availability of the members.

We will use a variety of third-party management software/tools to keep each other apprised of progress and problems. We will use Google Drive to prepare and share written materials and presentations. The team will follow the schedule decided on in the previous week's meeting and work on the assigned tasks throughout the week, consulting as needed with the group as a whole, or with individual members working on related components. The team will also have a project manager, whose responsibility it is to arrange the team meetings, schedule the task deadlines, and assess the progress of the project within each milestone.

Team Role

The project team is consisting of four people and everyone will be working on all the tasks from front end to back end, the implementation will be based on the group collaboration. Each week the project manager will be changes and delegate all the pending task of backlogs to new project manager. The scrum meeting would be consisting of 30 minutes after every week or as discussed above every Friday 3-5.

Final Project Evaluation:

David, although he dropped the class, came up with the initial idea for this project, and the application we created does fulfill the requirements he initially envisioned, for the most part. However, we attempted to incorporate an IoT aspect into our project in the form of data collected about parking availability from sensor, which was not completed. Additionally, we did not complete certain features we were aiming for, including a reservation page, the ability to receive recommendations, and the ability to search for availability among the listed parking lots.

There were some issues with our design process, but ultimately, we did produce a project which, at least resembles, our initial design. We believe the process would have been better with more frequent or regular meetings, thought this may have been hard to achieve. We have somewhat mixed views on the agile process but as our process for this project did not exactly follow the agile methodology (i.e. we did not have daily meetings and to some extent we felt a lack of leadership). However, these issues would not prevent any of us from using the agile methodology.

We had to update our project schedule to accommodate the loss of a group member and because we did not foresee how long certain aspects of our design would take (in particular the incorporation of sensors). Within our group Greg was designated as a leader, and for the most part we stuck to that structure. Often Pravalhika and Muhammad would provide reminders deadlines or recommendations for the flow of implementation which was helpful and appreciated. In the end both Pravalhika and Muhammad ended up having more leadership responsibilities than initially anticipated, for example Pravalhika took the lead with submission of the increment 3 report, and Muhammad played a leading role in deployment.

One recommendation for future projects would be to not anticipate extra features and only attempt them after completing an initial implementation of the initial design. Additionally, as a recommendation for the class, having more focus on the agile methodology and sticking closer to that methodology for development might at least give students a better sense of what working under the agile methodology is truly like.

INTRODUCTION

- Real world parking issue in congested space of parking.
- UMKC Parking lots limited availability consume a lot time for a student, faculty member and guest to find a good parking spot.
- A web application that can show each parking location with number of spaces available can save a lot of time.

Functional Requirements

A functional requirement are the expected types of outputs when the system is implemented in a certain kind of environment.

- The Parking App will be using a web and mobile interface.
- The Parking App will keep usernames and passwords for each user.
- The Parking App will allow users to signup or login.
- The Parking App will allow users to update availability.
- The Parking App will save the review and availability of all parking lots.
- The Parking App will support simultaneous access to everyone.
- The Parking App will feature display map of lots.
- The Parking App will have an admin page on the website to view server data

Non Functional Requirements

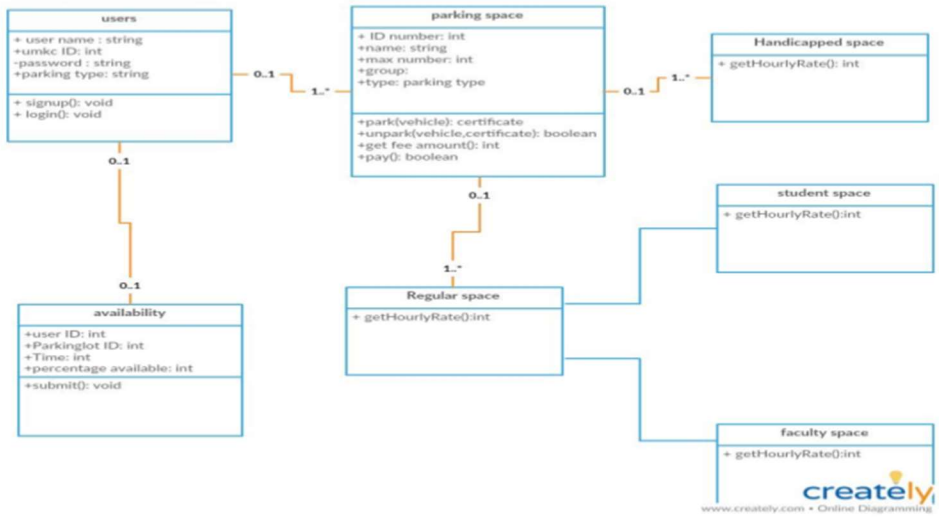
Nonfunctional Requirements define system attributes such as security, reliability, performance, maintainability, scalability, and usability.

- The Parking App will use a database to store user information and parking lots.
- The Parking App will be implemented using Ionic Framework
- The Parking App will use firebase for sign-in and signup.
- The Parking App will use google API do display map
- The Parking App will collect availability from user and store in Mongo DB.
- The Parking App will add user review in Mongo DB.
- The Parking App will be optimized using the user review.
- The Parking App will support multiple simultaneous users without noticeable slowdown.

Paro

UMKC

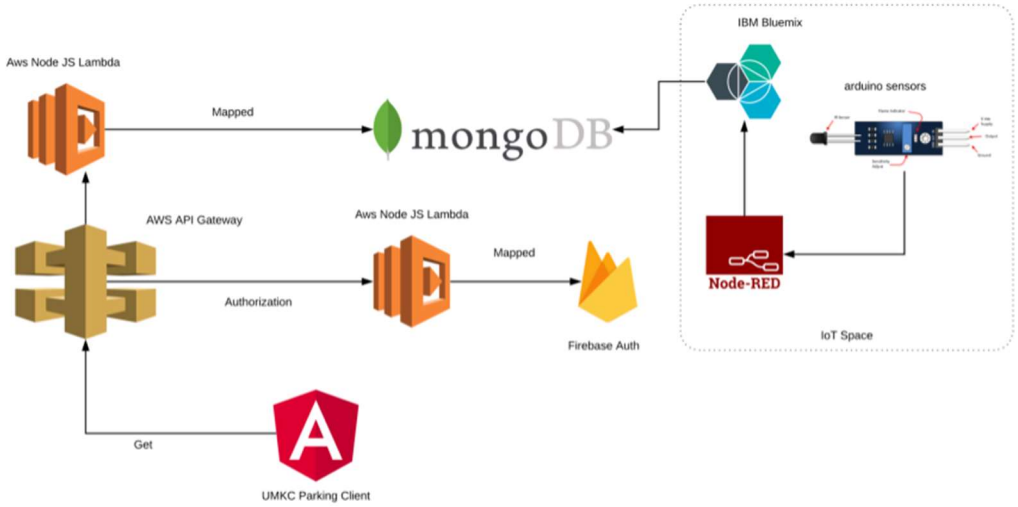
CLASS DIAGRAM



Ali

UMKC

ARCHITECTURE



Ali

UMKC

Implementation

- Ionic 3 Framework
- Angular JS
- Node js
- MongoDB (Mean Stack)
- Firebase
- Google API
- Node Red flow
- AWS

Greg

UMKC

Technologies in depth

MongoDB : Document database – used by your back-end application to store its data as JSON (JavaScript Object Notation) documents
Angular: (formerly Angular.js): Front-end web app framework; runs your JavaScript code in the user's browser, allowing your application UI to be dynamic
Node.js : JavaScript runtime environment – lets you implement your application back-end in JavaScript

Greg

UMKC

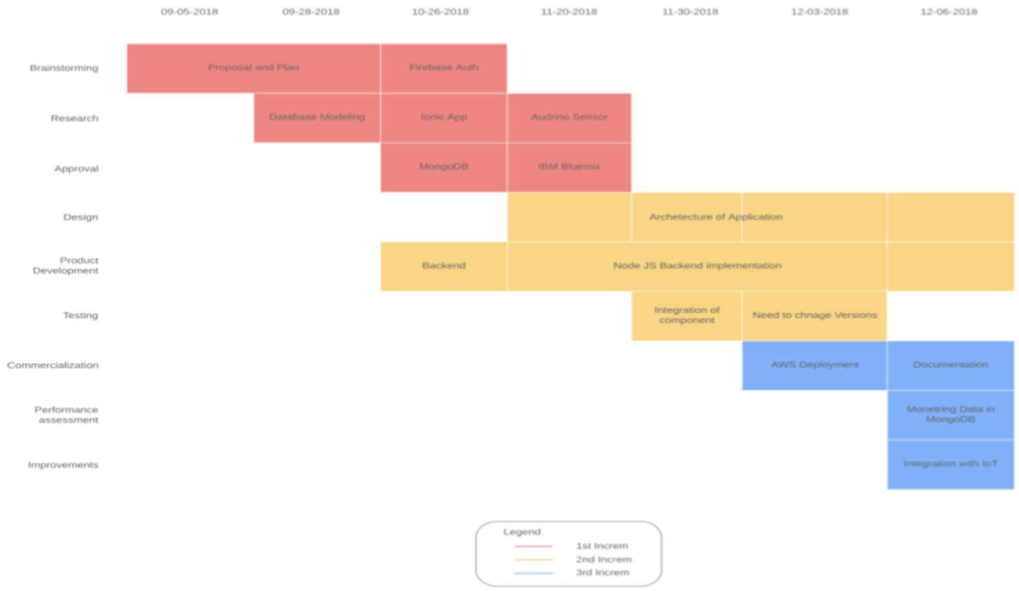
Technologies in depth

Node Red: Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things. Node-RED provides a browser-based flow editor, which can be used to create JavaScript functions.
IBM Bluemix: IBM Bluemix is a cloud platform as a service developed by IBM. It supports several programming languages and services as well as integrated DevOps to build, run, deploy and manage applications on the cloud. Bluemix is based on Cloud Foundry open technology and runs on SoftLayer infrastructure.

Greg

UMKC

Gantt Chart



Greg, Ali

UMKC

INTRODUCTION

- Real world parking issue in congested space of parking.
- UMKC Parking lots limited availability consume a lot time for a student, faculty member and guest to find a good parking spot.
- A web application that can show each parking location with number of spaces available can save a lot of time.

Paroo

UMKC

Project Proposal:

Project Goals and Objectives:

Motivation: Finding the Parking Lot is always a time-consuming process. Installing the app in the smartphone and getting information on the availability of the parking spaces by following the app is the most convenient way. Hence, we opted to create an app which is user friendly based on the information collected from sensors fixed at the entry and exit of the parking.

Significance/Uniqueness: Parking Application will include Internet of Things component, namely data will be collected by sensors at entrances and exits of parking lots to maintain information about the availability of parking spots.

Objectives: Create a service which maintains information about parking availability in UMKC Parking Lots and provides that information to a mobile application. Information number of cars in a given lot will be collected by sensors at entrances and exits of the lot. Application should enable users to check parking availability and provide information about parking availability to the service to update the service.

System Features:

- Android Application showing users up-to-date information about availability of parking spaces in UMKC parking lots.
- Server component which receives information from sensors and serves information to android application.
- Sensor component: sensors and entrances and exits of parking lots will feed information about current occupancy of parking lots to server component.

Related Work:

Smart Parking App: It relieves the stress of driving around and looking for the parking lot availability. It helps in better traffic flow, avoids congestion and maximum use of parking spaces.

PocketParker: It detects the arrival and departure of the vehicles using Recognition algorithms. By estimating the number of drivers not using the PocketParker, a small fraction of drivers can generate accurate predictions.

Bibliography:

Wireless Sensor Network Protocol for Smart Parking Application Experimental Study on the Arduino Platform

Smart City Application: Android Based Smart Parking System

Smart Parking: Parking Occupancy Monitoring and Visualization System for Smart Cities

Parking Management System Using Mobile Application

Plan:

Project Task

The goal of the proposed project is to create a hybrid application which run on web browser as well on android mobile phone. The interface for users to check availability of parking in the UMKC parking lot. The application has never been prepared in a way where student can login and find the available spot in Parking lot. This application will not only help the student to find the free parking spot but also the visitors who have difficulty to find the spot where they can park their vehicles when they are totally new to the place and do not have any guidance to locate what are best places to park near UMKC. The parking is currently divided into 3 parts to find the parking spot some of the parking spot are reserved for disable people. The meter parking lot for guest are few and using the application you would be able to view either those parking spots are available or filled up and which location will be closest from your current location. The UMKC website only provide the campus map with designated parking lot identified on the on the campus map. The application will utilize those location and show number of available spots.

Requirements

A. Functional Requirements
A.1: The Parking App will be using a web and mobile interface.
A.2: The Parking App will keep usernames and passwords for each user.
A.3: The Parking App will allow users to signup or login.
A.4: The Parking App will allow users to update availability**.**
A.5: The Parking App will save the review and availability all lot.
A.6: The Parking App will support simultaneous access to everyone.
A.7: The Parking App will feature display map of lots.

A.8: The Parking App will feature multiple levels parking.
A.9: The Parking App will have an admin page on the website to view server data

A.1: The Parking project will be using a web and mobile interface

For this Parking project, an ionic web application framework will be used and we would be displaying all parking lots on application interface. This interface will be a hybrid application containing the menus, login page, and the parking lots and their availability.

A.2: The Parking project will keep usernames and passwords for each user.

Users of the *Parking* Application that is the goal of this project will be identified with a username and password they create. Each username must be unique, but there are no other restrictions on the username and password.

A.3: The Parking project will allow users to sign up or login

Before using the parking application, a user must either login or sign up. The login page, accessed by a button on the main page and screen in android application, allows users to enter their username and password. If these credentials match a record in the database, the user is allowed to continue to the check availability. To visit the signup page, the user can press the signup buttons on the main page or the login page. To sign up, they must enter a first and last name, a username, and a password. After they sign up, they can return to the login page to enter the parking application.

A.4: The Parking project will allow users to update the parking availability

Once the users log in, they will be directed to a menu where they have the option to begin a choose the parking lot. This *Parking application* will be based on the available parking zone of UMKC parking lot.

A.5: The Parking project will save the review and availability all lot.

As users open the Parking Application, they will earn points by providing the availability in parking lots. The user would also be able to update the availability of parking in the application and getting review on those updates from other users. They will also lose points if get any negative feedbacks on parking availability. At the end of the application they will ask to subscribe the channel of parking update and will get notification of availability lots.

A.6: The Parking App will support simultaneous access to everyone

The application will get the real time parking lot availability data from different users. The data would be simultaneously available to other users using the parking application. There would be no limit of how many users can access the application.

A.7: The Parking App will feature display map of lots

User would be able to see the UMKC map and choose a parking lot where he would be able to find the parking based on the available parking spots. The UMKC campus map is available on UMKC website. The application will utilize that map and identify the parking spaces on each campus building have.

A.8: The Parking App will feature multiple levels of parking

On Parking availability screen the user would be able to see all the level of parking lots. If a parking lot will have different levels the user would be able to see which level of parking space have availability and which level of parking level hold how many disabled parking spaces.

A.9: The Parking App will have an admin page on the website to view server data

In order to increase the usability for the administrators of the Parking App, we will include an admin page on the application which will display information about the application, the database, and the environment. The admin page will be accessible using the admin username and password when logging in on the main page.

B. Non-Functional
B.1: The Parking App will be hosted on AWS Cloud.
B.2: The Parking App will use a database to store user information and scores.
B.3: The Parking App will be implemented using Ionic Framework
B.4: The Parking App will use fire base google API for sign-in and signup.
B.5: The Parking App will collect availability from user and store in Mongo DB.

B.6: The Parking App will add user review in Mongo DB.
B.7: The Parking App will be optimized using the user review.
B.8: The Parking App will support multiple simultaneous users without noticeable slowdown.

B.1: The Parking App will be hosted on AWS Cloud.

The Parking application would be hosted on a cloud server where the architecture would make it really a scalable, highly available application. The application would be using different cloud services to handle usability, API calling and later if someone want to use our application, they can

B.2: The Parking App will use a database to store user information and scores.

The parking application will be storing all the data into a database created in Mongo DB a NoSQL database. The application would be collecting all the information the Arduino sensor which would be placed in the parking area. The current collection method would be vehicle entrance and exist data would be saved in the mongo. The reservation would be allowed in few scenarios where the teacher would be able to add a reservation for their parking. The data for availability of parking and data for availability update would be in mongo db.

B.3: The Parking App will be implemented using Ionic Framework.

The application is developed in ionic framework and would require few system requirements for development. The ionic framework would make angular.js on client side of the application and node.js on server side of the application. Express.js framework would be used with firebase API to control application authentication and user identification.

B.4: The Parking App will use fire base google API for sign-in and signup.

The application is using firebase google API to authenticate the users, it will also make the single sign on mechanism in the application, so user do not need to authorize them again after once they have entered the application. Application also allow user to sign up with a sign-up page if they need to sign-up.

B.5: The Parking App will collect availability from user and store in Mongo DB.

The parking application contain and keep record of availability in the mongo database, the user can enter the number of available parking lots after checking the parking lot, how many parking spots are empty and how many more vehicles can be parked in that parking lot. Some parking spots are available on percentile base also can be populated or the parking lot is full. Once the user enters the application, he can easily navigate to availability page where he can enter the availability of parking spot.

B.6: The Parking App will add user review in Mongo DB.

The parking application will also take the review of availability of parking spot as to rate and trust the user who enter data for parking availability. This method will make sure that we trust on the parking availability from only those users who have provided valid parking spot locations.

B.7: The Parking App will be optimized using the user review.

In future once, the prototype has been seen by number of users we will be able to review the feedback and will optimize the application according to the user feedbacks. In this phase we will just be making an overall porotype which can be later updated. With help of prototype design pattern, we will be able to optimize the application easily.

B.8: The Parking App will support multiple simultaneous users without noticeable slowdown.

The parking application will be hosted in a cloud environment would make it to nimble and easily provide edge content delivery network CDN for more users. The current state of development only considers the application use for UMKC parking structure and we do not think this application would be having more traffic to make AWS server slow.

Optional Features

If time allows at the end of the project (and once we have completed all functional and non-functional requirements to the satisfaction of the client), The user review on the base of their report for parking availability is an optional feature and we also have a state of mind to create the reservation system in parking and that would be also an optional feature.

Scope

The scope of this application is to be a hybrid application which will allow users regardless of their association with UMKC as student, teacher, administration or a visiting guest would be able to login using there google account. After successful login they will be able to see the available parking lots of UMKC and the number of parking spot available in those parking spot. The user will also be able to update the parking space available on those parking spot.

Hardware

The client does need a mobile or any other computation device which support browser with engine v8. For development we of this application we used 3 different machines.

Apple Macbook Pro:

- Processor 2.3 Ghz Quad Core
- Ram 8 GB

Dell Celeron

- Processor Core i7 Alien Ware
- Ram 32GB

Lenovo Y700

- Processor Intel Core i7 2.6 Ghz
- Ram 16GB

Software

Front End UI/UX

This project will be using HTML5 and CSS in order to develop the web accessible page for the application. This will help us create login and top score page. HTML is the standard markup language for creating Web pages. HTML stands for Hyper Text Markup Language. HTML describes the structure of Web pages using markup. HTML elements are the building blocks of HTML pages. HTML elements are represented by tags. HTML tags label pieces of content such as "heading", "paragraph", "table", and so on. Browsers do not display the HTML tags, but use them to render the content of the page. CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once. External stylesheets are stored in CSS files.

Operating System

The operating system used for this project will be AWS EC2, AWS EC2 main operating system. The Debian Linux based EC2, gets updated every two years or so. The previous release, Jessie, came out in 2015, and now its replacement has arrived. Say hello to Stretch.

The Debian Project was first announced in 1993 by Ian Murdock, Debian 0.01 was released on September 15, 1993, and the first *stable* release was made in 1996. The Debian *stable* release branch is the most popular Debian edition for personal computers and network servers and has been used as a base for many other distributions.

Debian has access to online repositories that contain over 50,000 software packages, making it the largest software compilation. Debian officially contains only free software, but non-free software can be downloaded and installed from the Debian repositories. Debian includes popular free programs such as LibreOffice, Firefox web browser, Evolution mail, K3b disc burner, VLC media player, GIMP image editor, and Evince document viewer. Debian is a popular choice for web servers.

AWS EC2

The Amazon web service computational resource EC2 with t2.micro-processor and 8GB ram would spin the angular.js for handling the front end of system. The amazon infrastructure as service was the earliest to launch there EC2 computational web service in 2002. Amazon EC2 instance let you run any web

service with a friction of limitation. The EC2 allow you to configure a service as quickly to use a local hosting service.

Elastic Web Scale Computing

Amazon EC2 make is easy to scale the web server within minutes, you can make as many instances of server as you need. Scaling to maintain you EC2 fleet automatically inside an availability zone is easy and this improve the performance at maximum with minimization of cost.

Completely controlled

You can completely control your EC2 instance any time you need, if you do not have a lot of workload in the night time you can stop your EC2 instance. The EC2 instance can be controlled remotely via API to start, stop and restart.

Flexible cloud hosting services

Inside your EC2 instance it is your choice to use as many or different web servers. Amazon also allows you to use the CPU, instance storage and boot partition size according to your requirements.

Integrated

Amazon EC2 can easily be integrated with number of AWS services such as Simple Storage Service (Amazon S3), Amazon Relational Database Storage Service (Amazon RDS) and you can collectively make your virtual private cloud using these services in a wide range of applications.

Reliable

The Amazon EC2 is reliable as every instance get upgraded with new infrastructure and Amazon guarantee the reliability of 99.99%. The replacement happen rapidly and predictably.

Security

The EC2 instance and other AWS are highly secure as making cloud security is the top priority of the amazon.

AWS Lambda

Amazon Lambda is a compute service which enable you to not worry about underneath hardware and run your server code or a service on AWS Lambda. This way you can get a robust connection between your EC2 server and all other services you need to interact with as we will use AWS Lambda in this application to run Node JS server which will be interacting with our firebase API and mongo database.

AWS API Gateway

The Amazon API gateway help you to easily create the HTTP end point for you AWS Lambda function and API management to handle API request for one resource in your AWS. API Gateway create SDK of your client application in any desire language and test your API for mobile platform iOS and Android. We are using API gateway to connect with our lambda function from EC2 instance Angular JS client.

Application Development Software

Node JS

Node JS is a JavaScript server-side scripting language use in the project to develop server-side scripting to accept request from client and process it based on the function. In AWS Lambda function we will implement the Node JS server-side scripting to make it robust on client request.

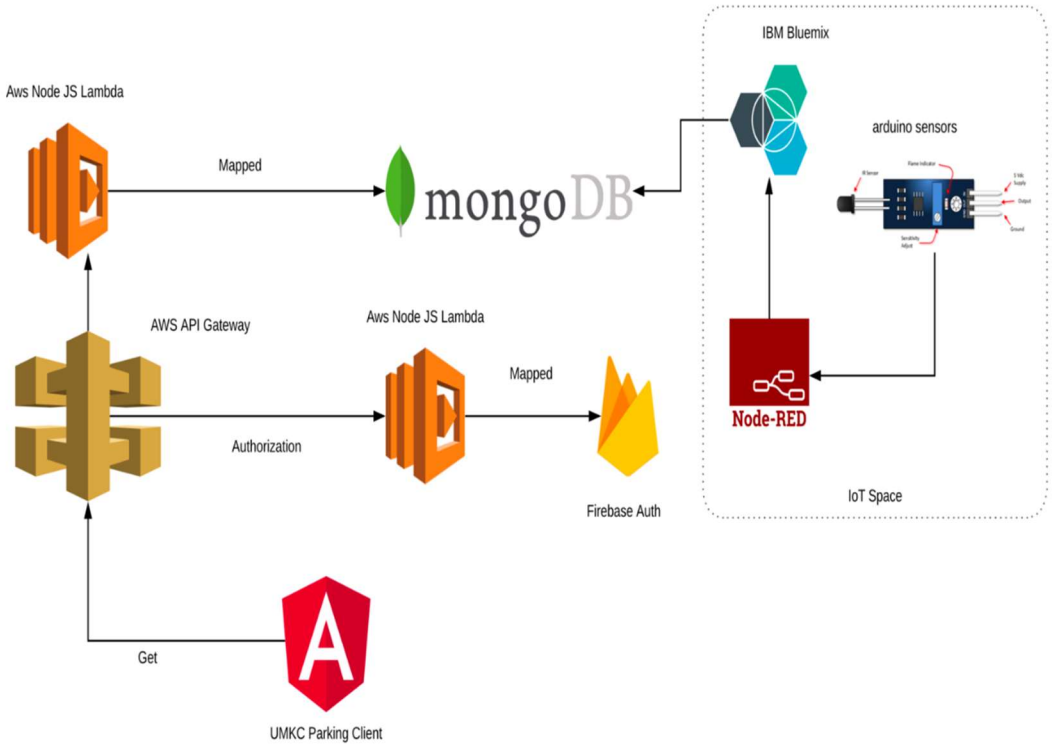
Angular JS

Angular JS is a front-end scripting language which is based on model view controller-based framework. The angular JS library and other tools are maintained by google and other community members. In UMKC parking application the angular application would be the client side of design.

Ionic Framework

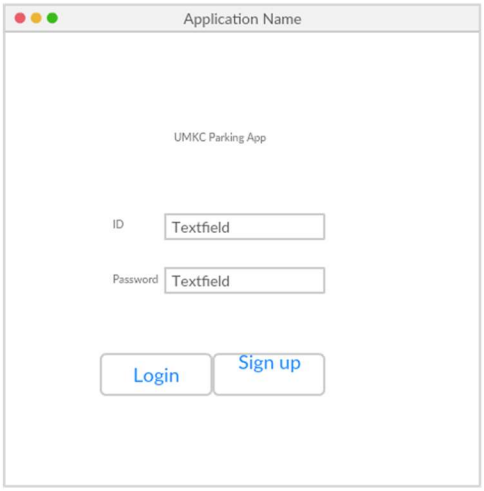
The Ionic framework is used in the application, so we can create the hybrid application based on function requriments.

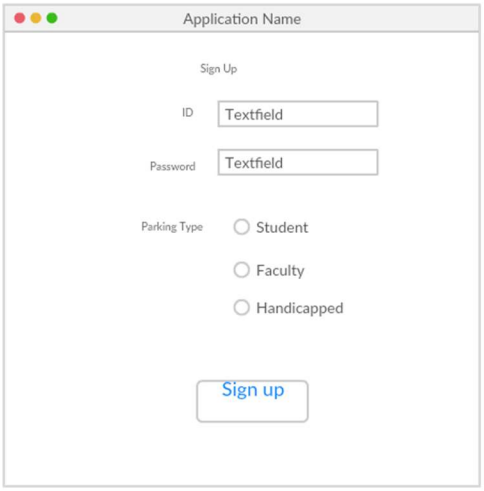
Architecture of Application

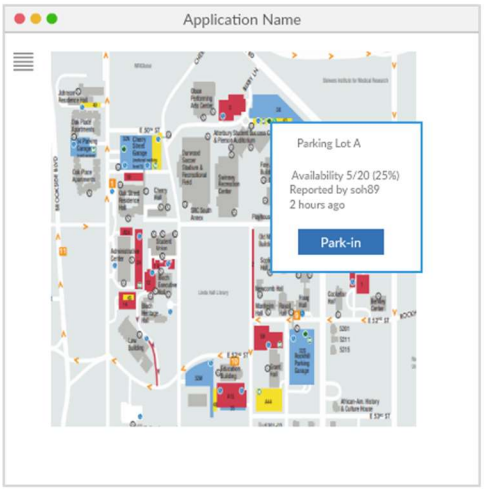


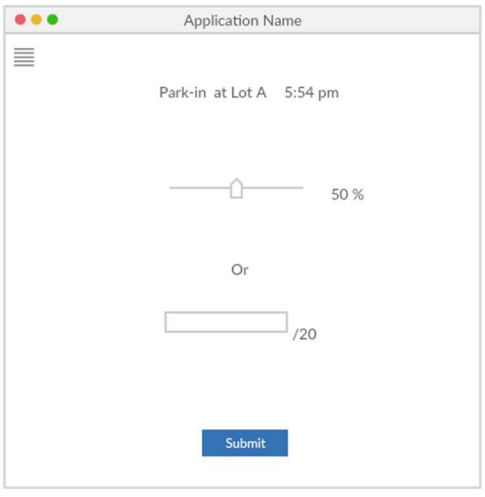
Increment 1:

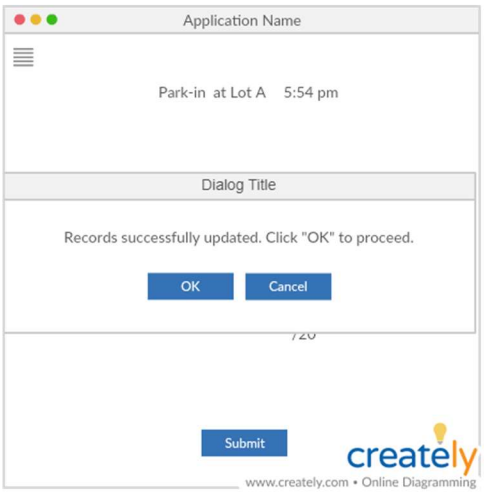
We created Wireframes (see below) to base the design of our UI on.



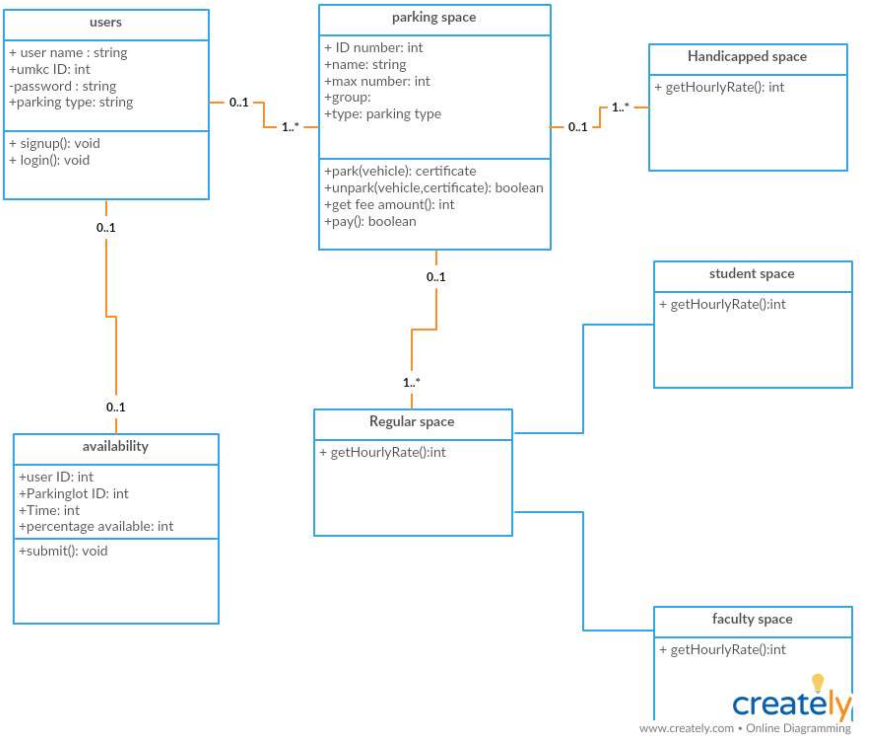


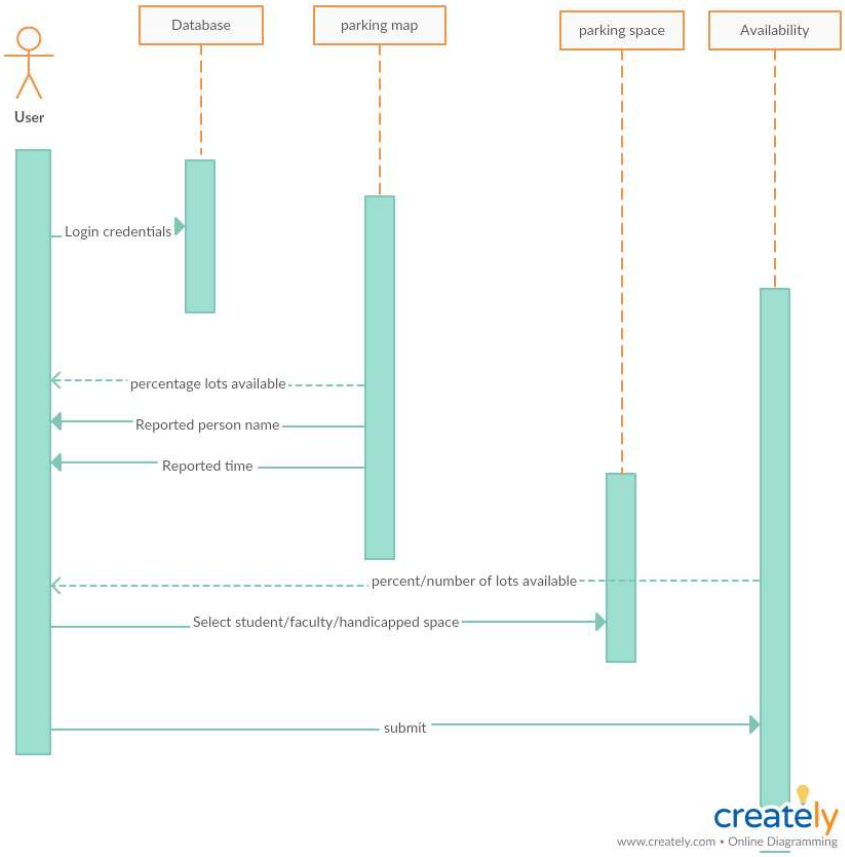






Additionally, created UML Class and Sequence Diagrams (below) to guide the implementation of features.





Implementation

We created a testUMKCParkingApp using Ionic 3. In the deployment section, you can see it being tested in chrome and android; happily it is a functioning hybrid app. However not all of the functionality we aimed to finish for increment 1 is implemented. Below is the functionality of the login page. The Sign In button is fully functional, as all it is intended to do is redirect the user to the Sign Up page. However, the Login button simply lets any user through to the content regardless of whether they even input an 'ID' and 'Password'.

```
public login(){
  if(true){
      this.goToHome();
  }else{

  }

}

public goToSignUp(){
  this.navCtrl.push(SignUpPage);
}

public goToHome(){
  this.navCtrl.push(HomePage);
}
```

Ultimately our goal is that users will be able to login via OAuth 2.0 or by calling for validation from our server, which will check their credentials against a list of signed up 'Users'. These Users make up one of the three major parts of our DB scheme. Below is an out line of the DB schema:

Users

id
name
umkcId

ParkingLots

id

name
maxNumber
group
type

Availability

id
parkingLotId
userId
timeReported
availability
In our current app all three of these are implemented under 'assets/data/' as .json objects.
And on the "Home" page we perform a http.get() call to the parking list. To do this we implement interfaces which for Lot and LotList.

```
/**
 * Interface for a single lot entry
 */
export interface Lot {
    id:         number,
    name:       string,
    maxNumber:  number,
    group:      number,
    type:       string
}

/**
 * Interface for a list of Lot entries
 */
export interface LotList {
    lots: Lot[];
}
```

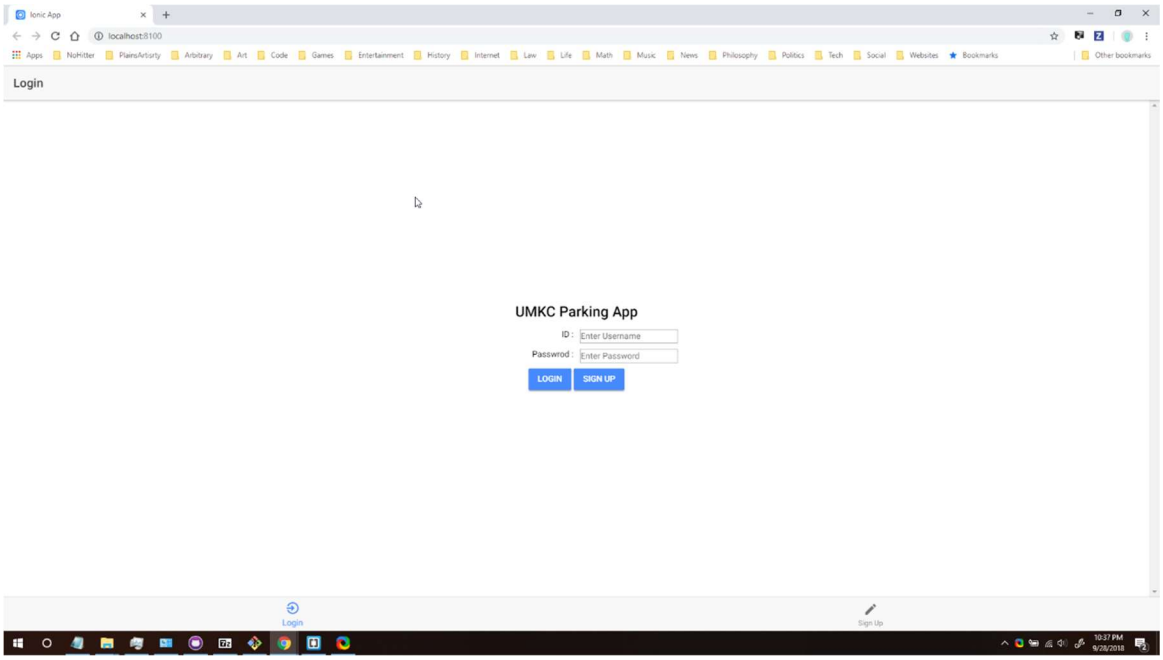
Then in the constructor for the home page we populate the lot list

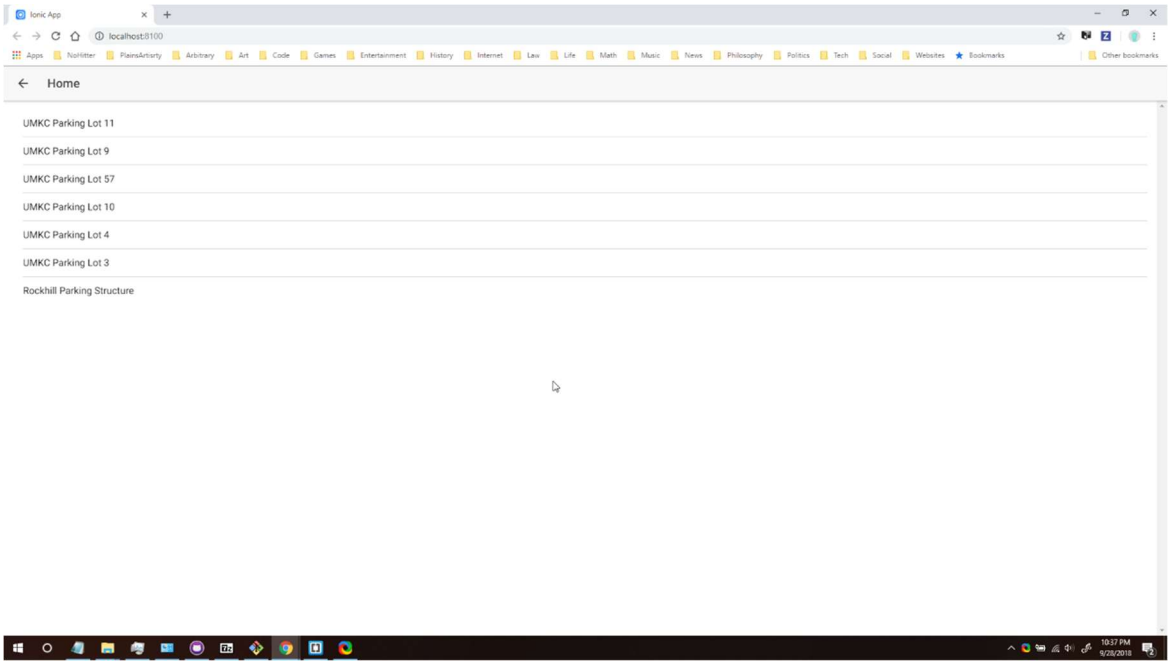
```
    lotList: LotList;
    aList: AvailabilityList;

  constructor(public navCtrl: NavController, public http: HttpClient,
                private alertCtrl: AlertController) {
    this.loadLots('assets/data/lots.json');
    this.loadAvailability('assets/data/availability.json');
  }
...
    loadLots(filePath: string) {
        return this.http.get<LotList>(filePath).subscribe(
            data => {
                this.lotList = data;
                console.log(data); // works
            }
        );
    };

    loadAvailability(filePath: string){
        return this.http.get<AvailabilityList>(filePath).subscribe(
            data => {
                this.aList = data;
                console.log(data); // works
            }
        );
    };

    displayAvailability(){
        let alert = this.alertCtrl.create({
            title: 'Test',
            subTitle: '10% Spaces Available',
            buttons: ['Dismiss']
        });
        alert.present();

    }
```

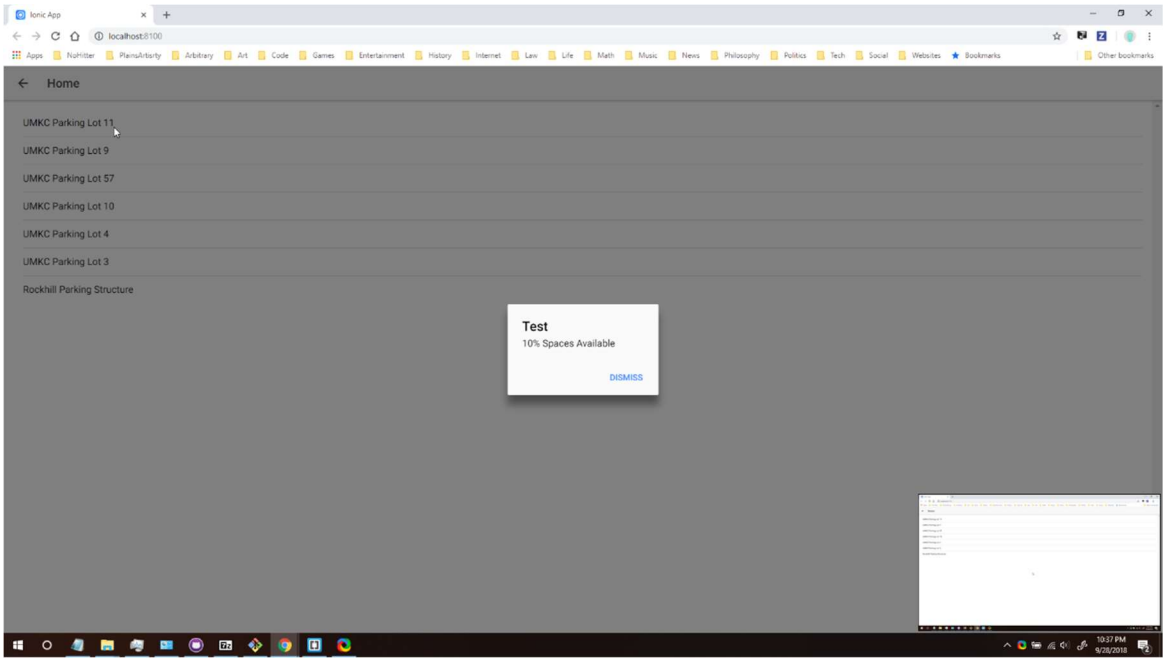
As you can see we also collect the availability 'dummy' information we provided, however this information is not yet what is displayed via the displayAvailability() function. We are still working on the logic for that function.

Deployment

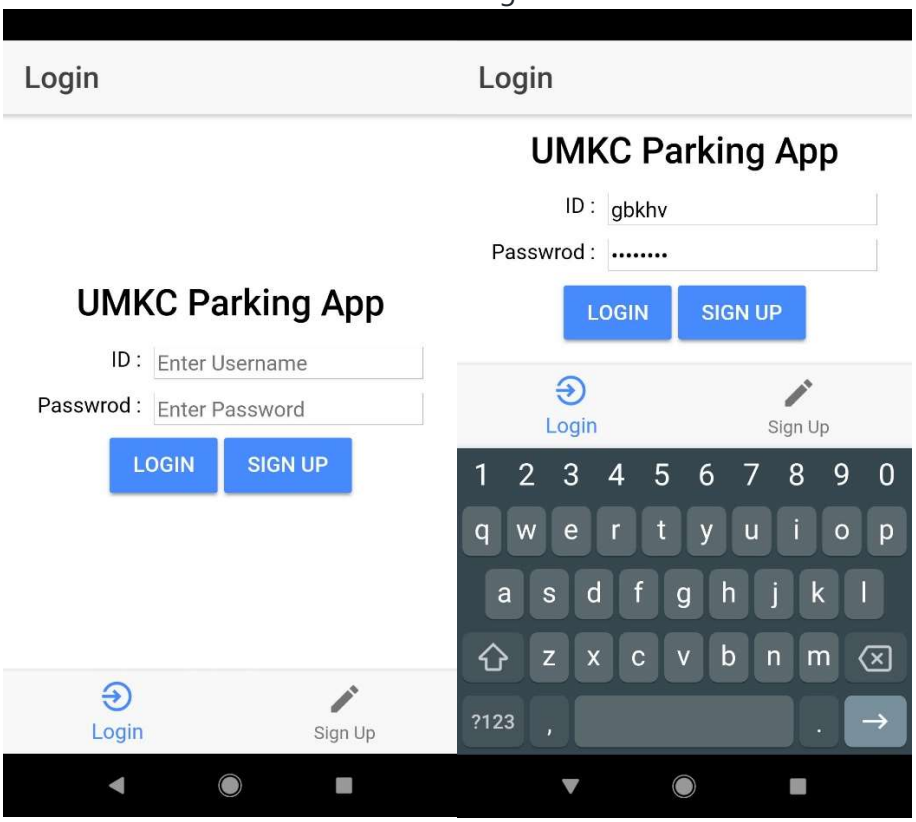
We tested our testUMKCParkingApp in Chrome and Android. Below are the screenshots for Chrome

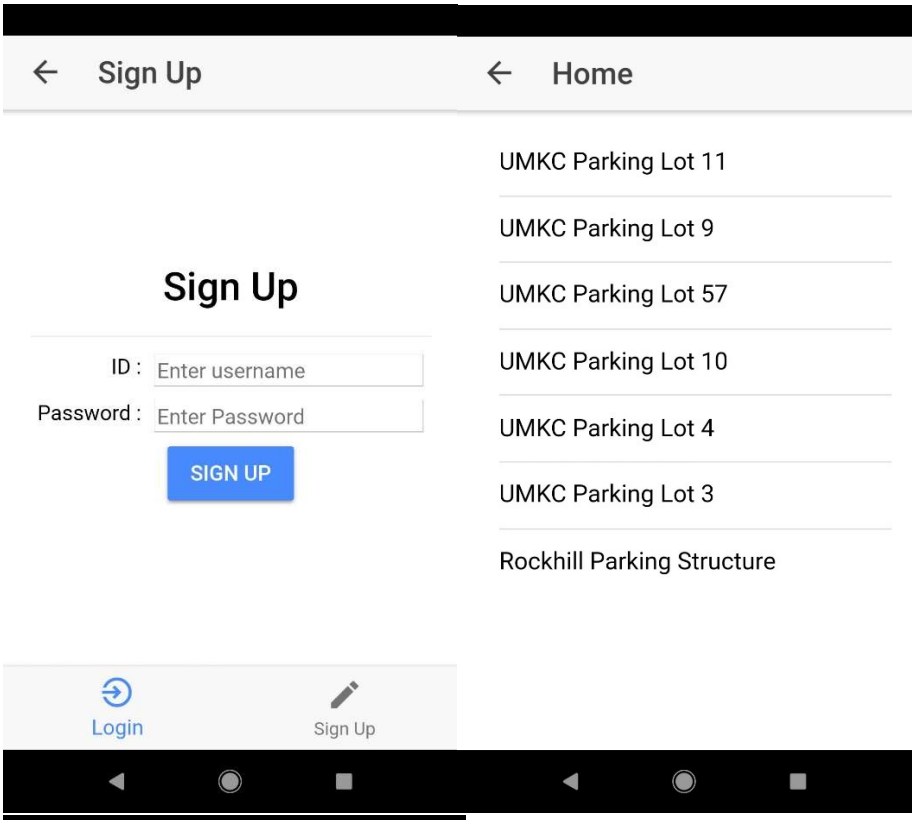


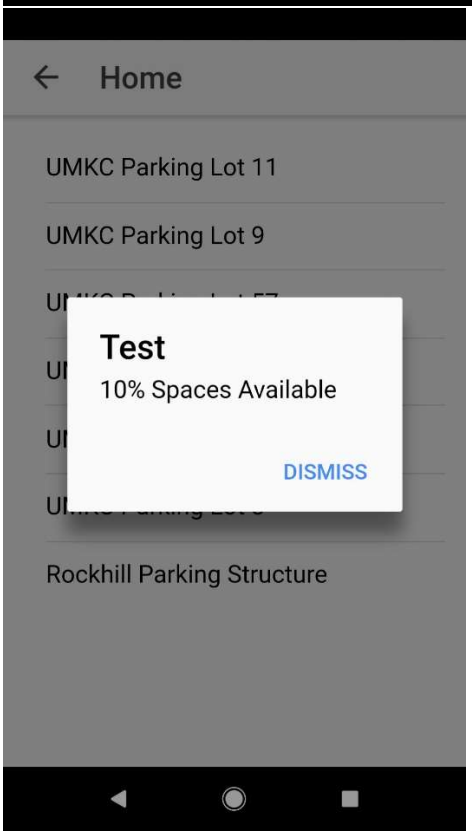




And below are screenshots from testing in chrome.

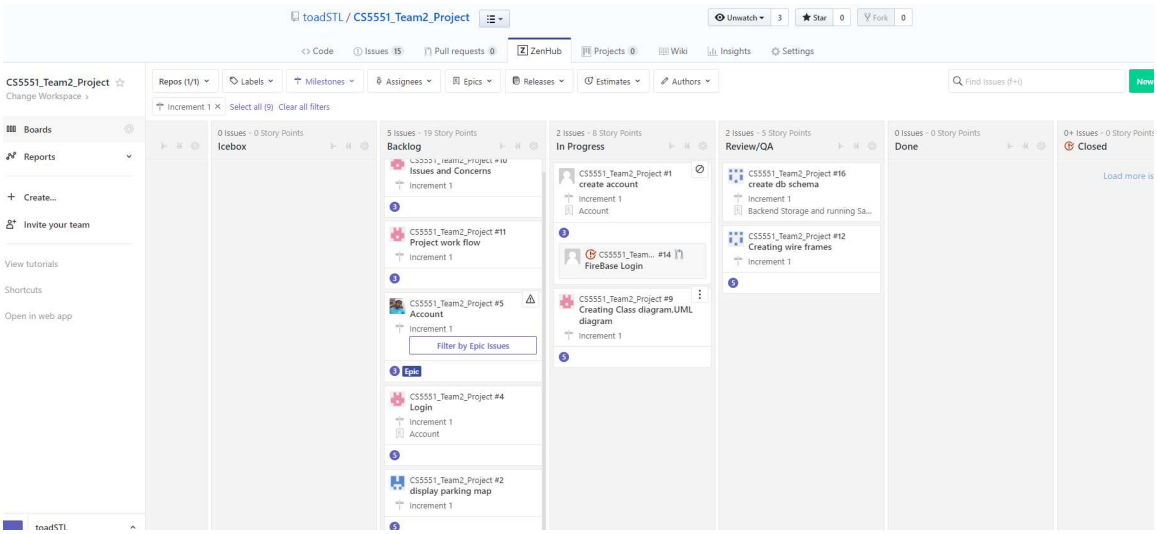



Sign Up
Sign Up
ID :
Enter username
Password :
Enter Password
SIGN UP
Login
Sign Up
Home
UMKC Parking Lot 11
UMKC Parking Lot 9
UMKC Parking Lot 57
UMKC Parking Lot 10
UMKC Parking Lot 4
UMKC Parking Lot 3
Rockhill Parking Structure

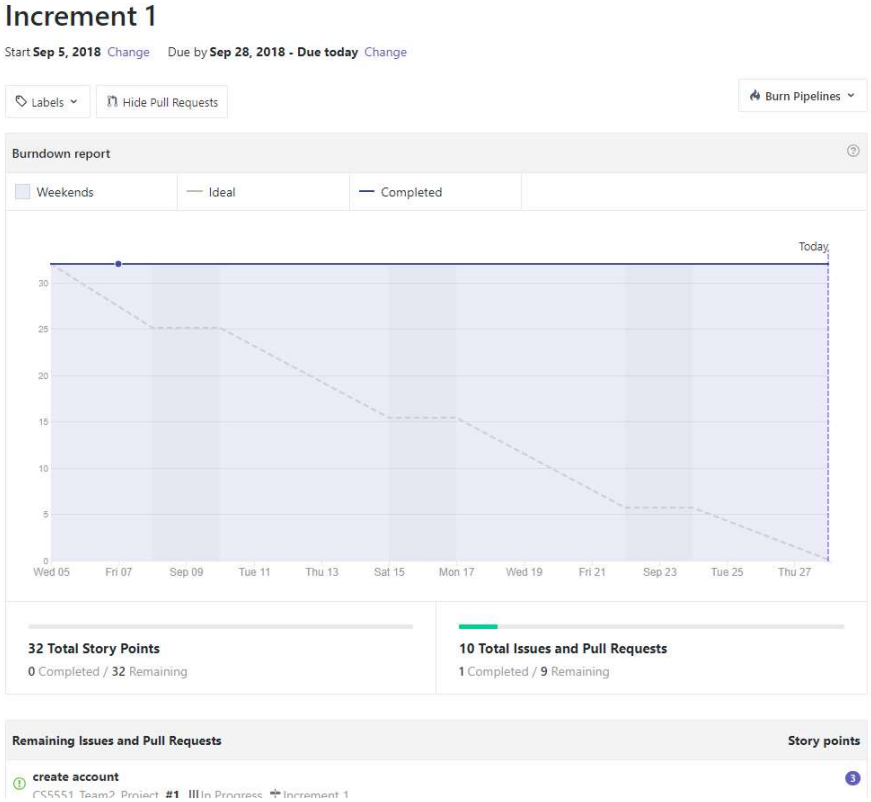

Home
UMKC Parking Lot 11
UMKC Parking Lot 9
Test
10% Spaces Available
DISMISS
Rockhill Parking Structure

Project Management

In zen hub we created several issues for increment 1, however it us not totally clear at time of writing, which issues have been completed and which have not.



For example, there is a login page, which almost looks like it is functioning, however the login, button does no validation of user credentials. Nonetheless we have included an image of the burndown chart from zenhub (below).



More relevantly, below is a description of the accomplishments of this increment:

- created wire frames (David) and UML Class and Sequence Diagrams (Pravalhika) to guide implementation
- created Database Schema for data to be used by this app (David)
- created .json data and populated with 'dummy' data for testing (David & Greg)
- Implemented UI using Ionic3 based on wireframes (Greg) (see testUMKCParkingApp, in code)
- Created 'dummy' functions in testUMKCParking app (Greg)
- Coordinated collection of project data (Pravalhika)
- created AWS for this project, on which we hope to implement a mongodb which will store data about parking lots and availability per the Database Schema (Muhammad)
- developed Oaugth 2.0 login using firebase (Muhammad)

Furthermore below is a list of issues with the existing code, which will serve as a basis for the backlog for increment 2:

- Login function does no validation (need to incorporate OAuth 2.0 login and signup=>mongoDB for sign-in without OAuth 2.0)
- Need to consider how to obfuscate password information in our database (suggest hash/salt)
- Review and Finalize DB Schema and implement DB on server (AWS)
- Actualize displayAvailability() function and create function to enable users to provide availability information
- Coordinate Communication between app functions and server
- Implement Home Page as Map of parking lots rather than list

Generally our team communicated well and made effort towards implementing the project.
It seems like implementing the sensors will be delayed until Increment 3, but we'll see what we can do.

Conclusion

This app doesn't do too much yet, but it is a start toward the UMKC Parking App we are going to make for this project.

References

[Ionic 3 Documentation]
[stackoverflow hide ionic tabs]
[stackoverflow display json in ion-item]

Increment 2:

Implementation

In this increment we incorporated firebase login (not previously incorporated) We updated the UI to include images and use google map (rather than a simple list for differentiating between parking lots) This general UI update also included adding more functionality to the application. Integrated into mean stack (minus for now the MongoDB aspect) producing functions which will call the database to get parking availability and set parking availability via a node js controller. We also created a virtual sensor using IBM Watson IoT platform and node-red, however the data from this virtual sensor is not being served to the

Deployment

We deploy in android using ionic launching and hosting api calls from our node server.

Increment 3:

Implementation

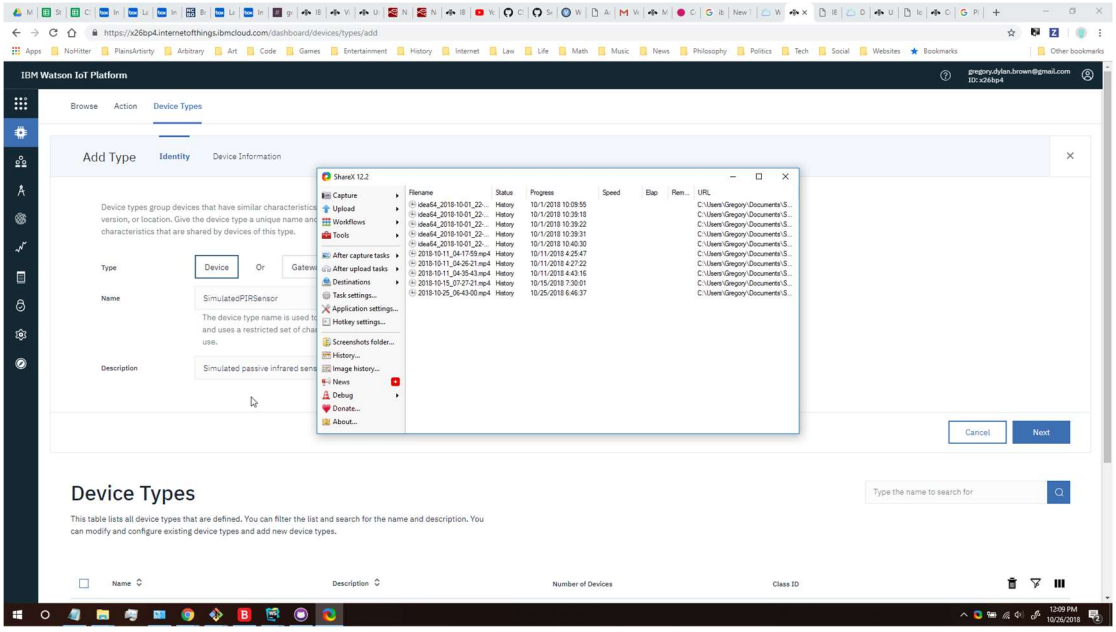
Integrated into mean stack producing functions which will call the database to get parking availability and set parking availability via a node js controller

Updated Mongo DB part

Added additional features

Deployment

We deploy in android using ionic launching and hosting api calls from our node server



References:

Wireless Sensor Network Protocol for Smart Parking Application Experimental Study on the Arduino Platform

Smart City Application: Android Based Smart Parking System

Smart Parking: Parking Occupancy Monitoring and Visualization System for Smart Cities

Parking Management System Using Mobile Application

Ionic 3 Documentation

Node Red Documentation

Simulated Sensor Device

Google Maps API documentation

stackoverflow hide ionic tabs

stackoverflow display json in ion-item

Acknowledgement Statement: "The work has been completed under the guidance of Dr. Yugi Lee, Rajaram Anantharaman, and TAs (Ruthvic Punyamurtula, Bhargavi Nadendla, Sravanthi Gogadi) in CS5551 Advanced Software Engineering, University of Missouri -Kansas City, Fall 2018.

The original parts of this report (i.e. the User Manual, Project Management Report, Final Project Evaluation, and the Plan sections) were written as follows:

- User Manual – Greg
- Project Management Report – Pravalhika
- Final Project Evaluation – Pravalhika
- Plan – Muhammad

The Plan section was expanded to include Functional and Non-functional Requirement, Software, Hardware, and Application Architecture and the remaining documentation was collected and included by Greg.