Living Requirements Document: Sniffit

RFID locator system

Andrew Pang

Braulio Fonseca

Enrique Gutierrez

Nader Khalil

Sohan Shah

Victor Porter

Product Requirements Document Revision History

Date	Version	Revision Class	Comment
10/12/15	1.0	Major	Started PRD by adding Introduction and Glossary
10/26/15	1.1	Major	Added 5 use cases/user stories as well as the Appendices
11/03/15	1.3	Medium	Added 5 more use cases (total of 10)
11/10/15	1.5	Major	Added a description for the System Architecture along with a link to the Test Code (GitHub)
11/20/15	1.8	Medium	Added a high-level view of the System Architecture
11/23/15	2.0	Major	Updated Test Code, updated System Architecture, added Revision History of PRD
11/24/15	2.1	Medium	Added and updated use cases

Introduction

Sniffit is a handy tracking application that helps its user locate lost belongings within a specified location. Users put an inexpensive RFID sticker on their item and register the sticker's ID in the application, along with the layout of the room the item is located in. From from there on out, anytime the user misplaces that item in that room, the app can use RFID sniffers to ping that item's sticker along with nearby stickers and use the various signal strengths to accurately

determine where the user's item is located. This app will be extremely useful for locating small items like a set of keys or a wallet, without using technology that is unaffordable.

Glossary

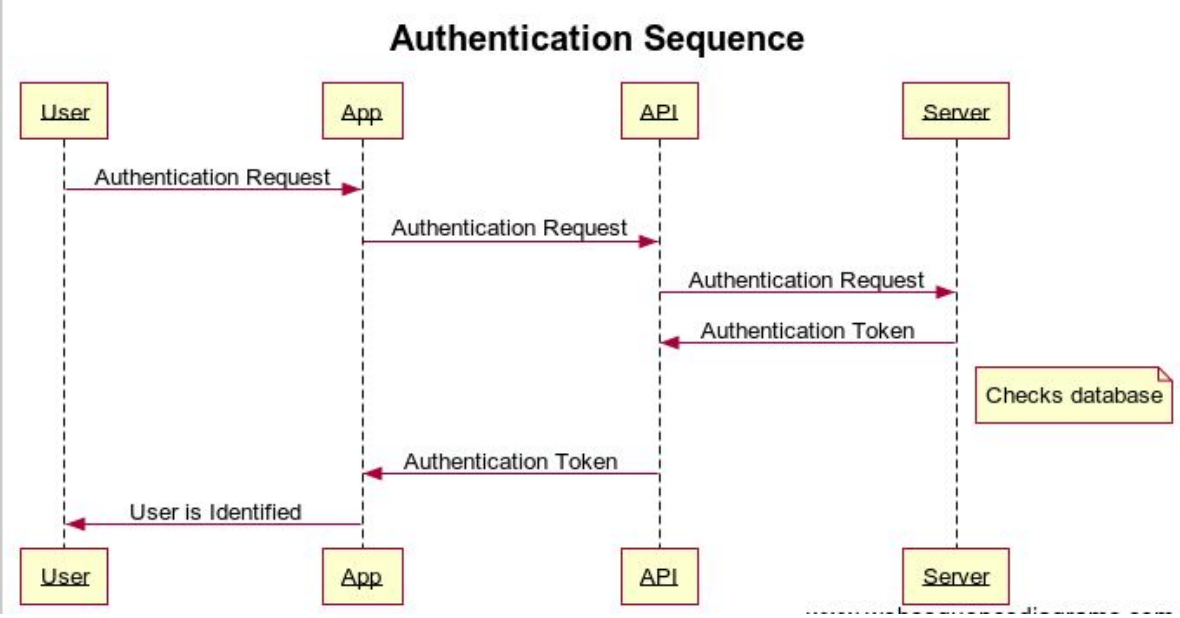
- RFID (Radio-frequency identification) - wireless use of electromagnetic fields to transfer data, for the purposes of automatically identifying and tracking tags attached to objects
- Web server - an information technology that processes requests via HTTP
- Representational State Transfer (REST) - software architectural style of the World Wide Web. RESTful systems typically communicate with simple HTTP calls (GET, POST, PUT, DELETE)
- Application Program Interface (API) - set of routines, protocols, and tools for building software applications

Requirements (Use Cases and User Stories)

Use Cases	Test
User	
As a user I can create an account via username and password.	Ensure user authentication is satisfied, disallow invalid users to view other people's data. Also check for duplicates in database and return error if the username entered is already taken.
As a user, I have the option to create data entries for my belongings.	Make sure that each item creation is attached to a valid ID and avoid duplications.
As a user, I can view a summary of data about all my items that I have created in table format.	Make sure that data displays correctly, whether the database entry being accessed is valid or not.
As a user, I have the option of modifying the data entries for my belongings	Make sure that the modified data is updated in the database and duplication is avoided.
As a user, I have the option to create data entries about my rooms that I want to search.	Make sure that a room entry has all the necessary information required to be a valid room.

As a user, I have the option to modify data entries about the rooms.	Make sure that the modified room information has been updated on the database. There is also detection of duplicate information and duplication is to be avoided.
As a user, I have the option to create data entries about reference tags, which are used to determine the position of tags on belongings.	Make sure that the reference tag information is complete and that duplication is avoided.
As a user, I have the option to modify the room ID of a specific reference tag to indicate that the tag is used in another room.	Make sure that the only information about the reference tag that is modified is the room ID.
As a user, I have the option to delete any of my lost item entries.	Make sure when an item is deleted, it cannot be accessed from the application or database from that point on.
As a user, I can request a live estimation of the location of one of my lost items by the click of a button, and the selection of a room to search.	Assure that the GUI is attached to a sequence of calls and guarantees an estimate; or a proper error message indicating the failure that has occurred.
As a user, I have the option to create data entries about the computers connected to the RFID sensors.	Make sure that the data entry has all the necessary information necessary to allow interaction between the server and the sensors.
Mobile Application	
As an mobile application, I need to be able to communicate with a cloud server and database to request the data that the user has selected from me.	Make sure GUI events trigger sequence of HTTP calls to the server.
As a mobile application, I need to be able to perform asynchronous tasks that allow me to provide an active UI while performing requests and computations behind the scenes.	Make sure that UI events can be triggered while sensors are fired by the Dragonboard 410c, computations are performed on the server, and relevant information is sent to the mobile application.
Web Server	

As a web server, I need to be able to handle incoming requests from the mobile app	Make sure that the controllers in the web server are performing the appropriate functions as indicated by the mobile application
As a web server, I need to forward requests out to the Dragonboards needed to fulfill a mobile application initiated request.	Make sure that the controllers that are activated can communicate with the Dragonboards registered with the user
As a web server, I need to perform the necessary computations and algorithms to locate the item once the information of the Dragonboards has been received.	Make sure that the Python scripts that perform the computations can be called on the server and output the correct information.
As a web server, I need to forward information to the mobile application in JSON format so that the mobile application can appropriately process the data.	Make sure that all the controllers that are called by the mobile application return the data in JSON format.
Dragonboard 410c	
As a Dragonboard, I need to be able to take the raw RFID reader data and parse it into JSON format	Post the data onto the web server and see it update on the web page.
As a Dragonboard, I need to be able to always be running a Python script that is listening for any requests from the server.	Post the response onto the web server and see it update on the web page.
As a Dragonboard, I need to run a driver that allows the SOC to communicate with the RFID sensor and issue commands through UART interface.	Make sure there is a response from the RFID reader, antenna, and RFID tags. This response will be visible from the terminal window.



System Architecture:

The entire system can be described as interactions amongst six different components. Specifically, these parts consists of an Android application on a mobile device, a Node.js web server, a predetermined number of Qualcomm Dragonboard 410c SOCs and RFID readers, an antenna, and specific number of RFID tags inside a room.

In the presumed setup, the RFID tags are separated into two distinct classes, reference tags and user tags. The reference tags are RFID tags that are used to determine relative position of the user tags. The information of each RFID tag is received by a RFID reader which is connected to a single Dragonboard 410c. All the information received from the RFID reader will be preprocessed on the Dragonboard 401c into a manageable format that would then be sent to a Node.js server.

This server will receive information from multiple Dragonboards each connected to their own RFID reader and will further process the data, running a RFID locating algorithm, to determine the relative location of the user RFID tag requested. The result of this information will be displayed on the Android app on a mobile device. The initial request for this information is done through the Android app.

The purpose of the antenna in the overall scheme is to provide an interface between the RFID tags and readers. The connection between the RFID readers and the Dragonboards is through a USB wire.

Onboard the SOCs, preprocessing is done through the use of Python scripts that turn the RFID reader's raw data into JSON format. This JSON formatted information is then sent through a port on the SOC to the web server. Depending on the request from the Dragonboard to the server, such as adding tags to the database, editing information, or deletion of tags, the Node.js server-side code will perform the appropriate action. This is done by an implementation of a RESTful API adapted for Node.js.

The same web server will also handle the requests from the Android app. The Android app send HTTP requests to retrieve the appropriate information. A high-level view of the system architecture can be seen in the following diagram (Figure 1).

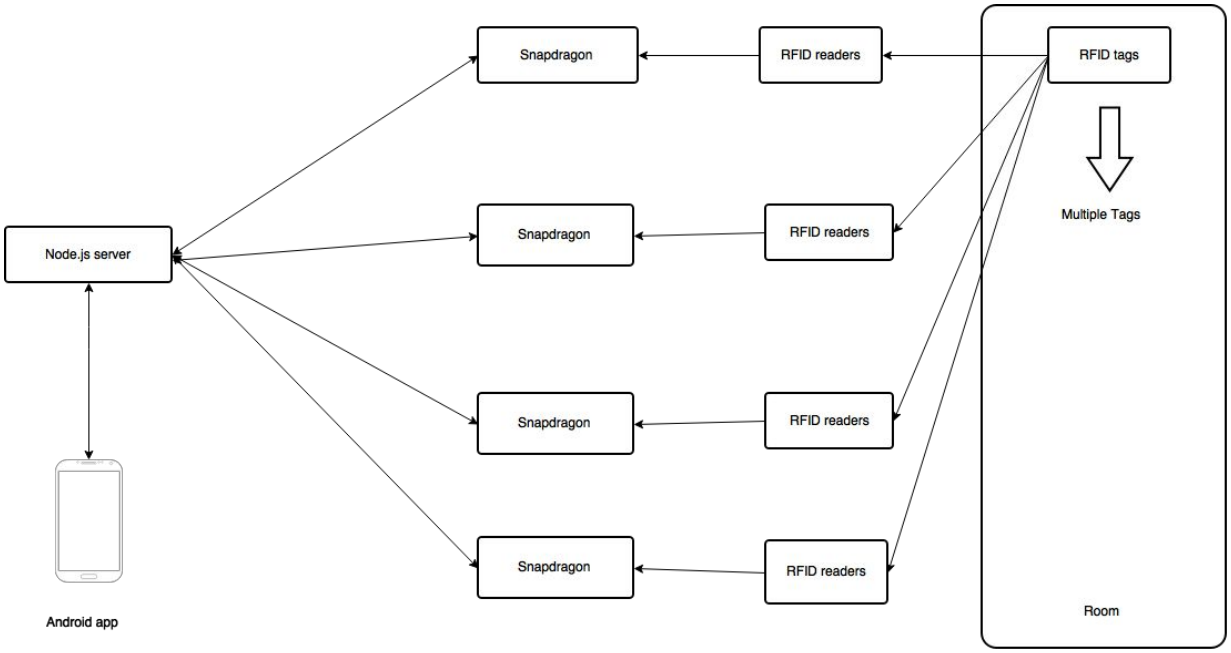


Figure 1 - High-level view of system architecture

Prototyping and Test Code

https://github.com/UCSBCapstoneQualcomm2015

Appendices

- Qualcomm Dragonboard 410c - system-on-chip semiconductor used for mobile devices. It may include multiple CPU cores, a graphics processing unit (GPU), a wireless modem, and other software and hardware
- Node.js - runtime environment for developing server-side web applications written in JavaScript
- Python - high-level programming language
- Amazon Web Services - collection of remote computing services, also called web services, that make up a cloud-computing platform
- RFID reader - network connected device with an antenna that sends power as well as data and commands to the tags.