Technical SW & System Integration Requirements

Version	Release Date	Description	Originator
Version 1.0	June 4, 2013	Initial Release	PNagy
Version 1.1	July 18, 2013	Minor	PNagy

Referenced Documents

The following documents should be used in conjunction with this document to fully understand product specifications.

1. *IA Functional SW Requirements VX.X.pdf*.
2. *IA_BLE Engr Test_RX.X.docx.*

Hardware/OS

1. Bluetooth LE chip will come from BlueGiga Corporation as either BLE112 or BLE113 (lower power consumption version). http://www.bluegiga.com/BLE112_Bluetooth_Smart_module
2. Tablet choice could be an Android based device (Nexus or Galaxy) or iOS Apple device. The factors influencing this decision are the following:
    a. Acceptance from customers. Tablet use is limited in health care settings. When they are used, there seems to be a preference for Apple iOS devices/OS because they are perceived as more secure and stable.
    b. Apple iOS and hardware readily supports Bluetooth LE. Nexus Android does not support LE as of June 2013. Current projections are for a supporting Android release in fall of 2013. Samsung Galaxy products apparently do support LE.
    c. Execution is critical for Invisalert's first product release so we will lean towards solutions that minimize risk.
    d. Cost. Longer term, cost may be a significant factor in selecting hardware.

Backend Database

It has not been determined whether the central database storage for the Invisalert system should be Cloud or client based. Either solution will have to be HIPAA compliant. The factors influencing this decision are the following:

1. Customer acceptance. Some IT departments are averse to Cloud solutions for security reasons.
2. HIPAA compliance.
3. Costs associated with supporting either approach.

Database tables and architecture should be developed with the idea in mind that our product will ultimately have to be integrated in a push-pull fashion with electronic medical record systems (EMR). Data structures should be compatible with EMR exchange standards, HL7 and CCR.

System schematics can be found in the Functional Specification document referenced above.

Prototype Development – *Brian Dolhansky created this document from this point forward.*

Brian Dolhansky, a graduate student at the University of Pennsylvania, developed the prototype. As of August 2013, Brian will be relocating to Washington State but will be available on a limited basis to support the application he developed.

Brian Dolhansky, June 2013.
bdolmail@gmail.com
609 472-1243

This document specifies the overall architecture envisioned for the software system as well as the implementation details needed to realize that application on a new platform. Both a record of the work already completed for the Android platform and a short roadmap for future work are given in this document.

General Overview

The typical front-end usage of the Invisalert (SW) application by users can be broken down into three separate categories:

1. Patient CHECKIN by the nursing staff.
2. Patient OBSERVER updates by the nursing or med-tech observer.
3. Technical changes to the application by a qualified system administrator (SA), and viewing checkup history.

Each case is covered in more detail in the section titled “Use Cases and Interface Tutorial.”

In addition to the aforementioned interactions with the application by users, SW will interface with a backend database to sync records between devices and a centralized storage server.

The following are high-level diagrams of each use case:

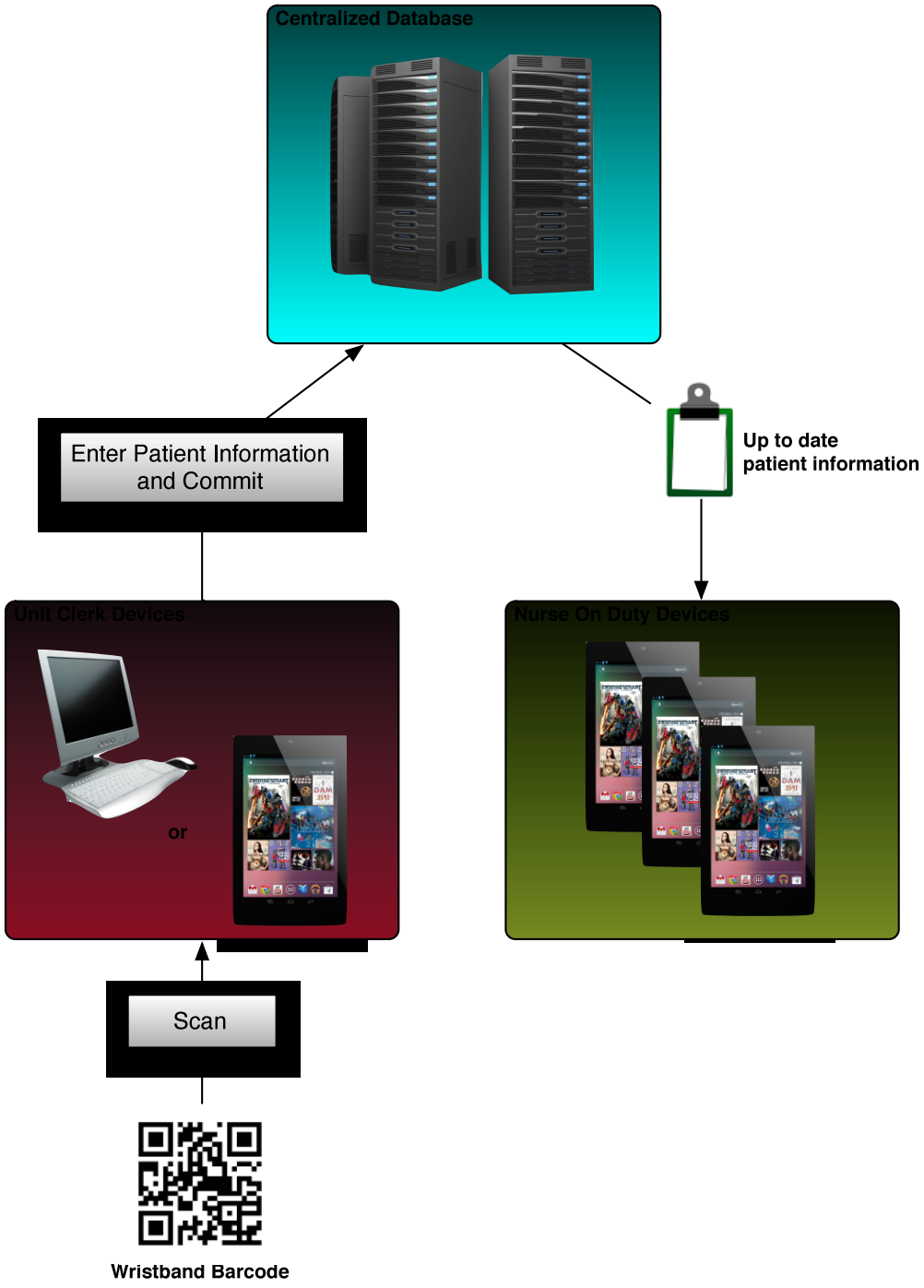


Figure 1 – The dataflow for patient CHECKIN. The wristband is scanned by staff via either a desktop workstation or a Nexus 7 running SW. The staff also enters the patient information and commits it, which ties a patient to a specific wristband. This information is stored in a centralized database and pushed to all applicable nurse on duty devices. Note that actions specifically carried out by a user are in gray rectangles.

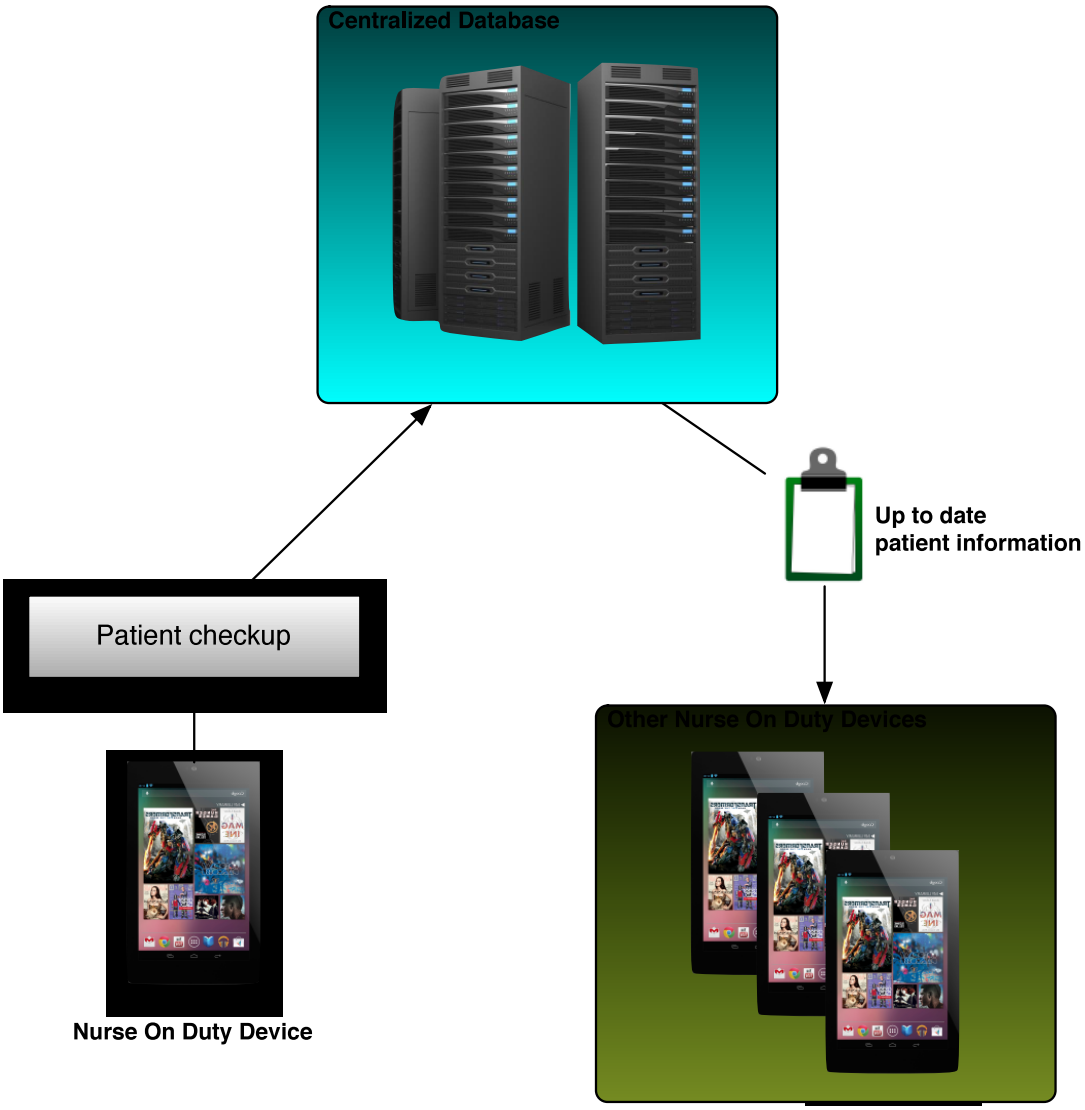


Figure 2 – The dataflow of a patient checkup operation. The observer staff responsible for a particular patient physically checks up on that patient and records this visit. This visit is then pushed to the other tablet devices to ensure that all patient information is up to date and synchronized. Note that actions specifically carried out by a user are in gray rectangles.

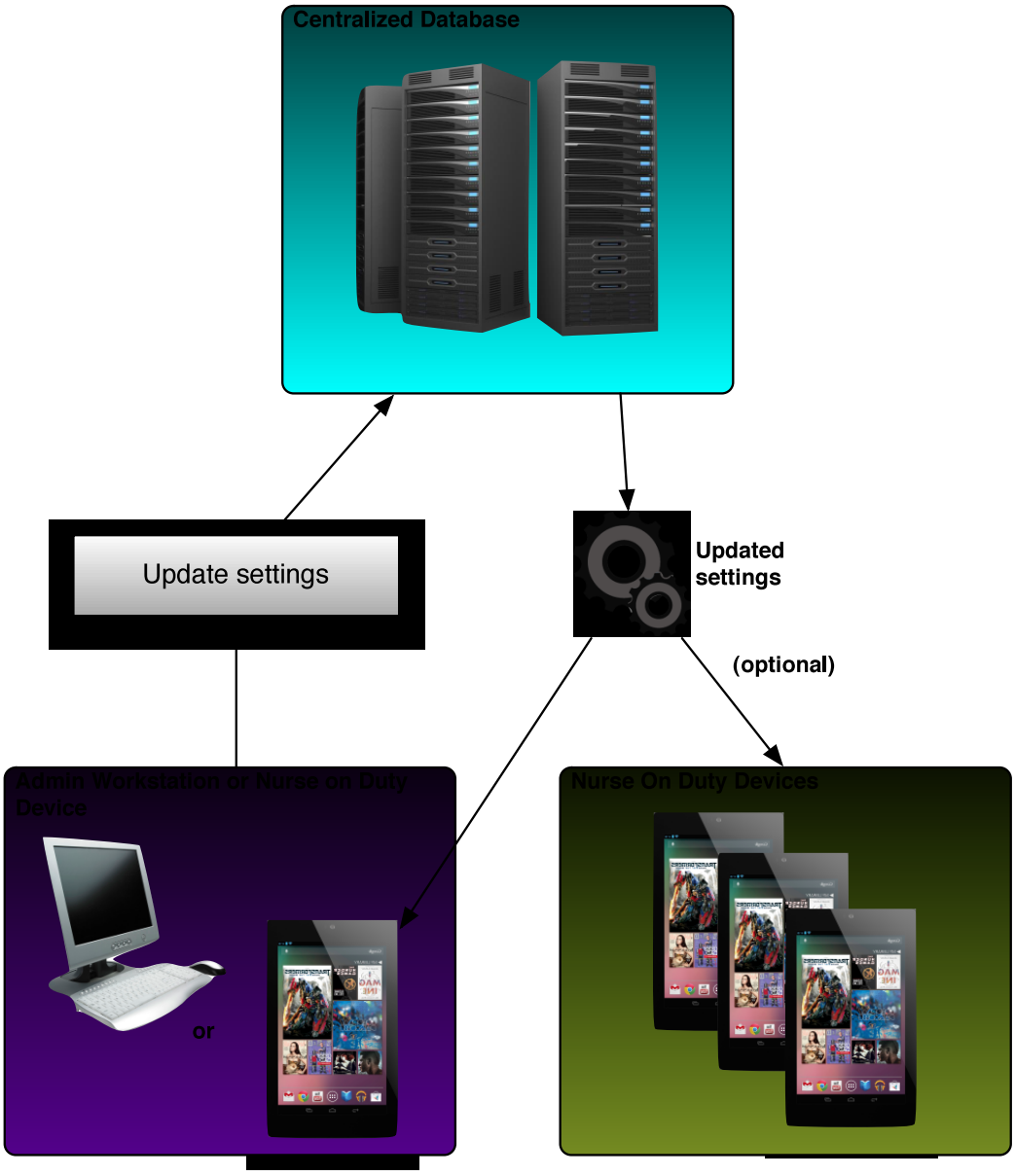


Figure 3 – The dataflow of an admin operation. An admin can change the settings for one or all devices from either a tablet or a workstation. These settings are then stored in the central database and pushed to all applicable devices (e.g. an Admin can update the settings directly on a device that are only applicable to that device). Note that actions specifically carried out by a user are in gray rectangles.

2. Platform Description

SW was designed to run on the Nexus 7, a tablet manufactured by Asus and sold by Google. The tablet features a 7 inch screen and is available for $200. At the time of the completion of this document, it runs the latest version of Android, v. 4.2.2.

The tablet was selected due to its low price, form factor, and inclusion of Bluetooth Low Energy (BLE) or Bluetooth 4.0 capabilities. However, at this time there is no way to access BLE functionality from the standard developer application programming interface (API). In the future, BLE support will apparently be added to the API. Other devices, such as the iPad or Samsung Galaxy and Note, apparently support BLE out of the box.

3. Getting Started

Downloading and Running the Code

SW was developed using the standard integrated development environment (IDE) for Android programming, Eclipse. The version of Eclipse used was Indigo, but newer versions should work as well. In addition to Eclipse, version 21 of the Android SDK was used.

Although details of setting up an environment for Android programming is outside the scope of this document, a developer wishing to run this code should first install Eclipse and the Android SDK by following the instructions here: http://developer.android.com/sdk/index.html. Then to import the Invisalert code, in Eclipse click on File>New>Project, and then "Android Project from Existing Code." Select the root directory of the Invisalert source, and click "Finish." They will then be able to run the application on a device by making sure that USB Debugging has been enabled in its Developer Settings.

Project Layout

The code and resources for this project have been laid out in the typical structure of an Android application. Code files are located in the src/ directory. Interface layout XML files are in res/layout. In res/drawable-mdpi/, there are stock photos used for patients. The project does not rely on any external libraries, but uses a third party source package for scanning QR codes

4. Use Cases and Interface Tutorial for the Prototype

The following uses technical language specific to Android applications (e.g. interfaces, intents, etc.).

System login

To begin, a user is presented with a login activity. They enter their credentials and the appropriate interface is presented. This is accomplished by launching one of three activities – the CHECKIN, OBSERVER, or ADMIN interfaces. Details can be viewed in the file LoginActivity.java.

Patient Checkin

Currently, the system supports checking in patients only from the tablet. To get to the checkin activity (in code, CheckinActivity.java), use the login name "checkin" with no password. Patient details can be entered, a picture of the patient can be taken, and a QR code (which would be attached to the external Bluetooth device) can be scanned.

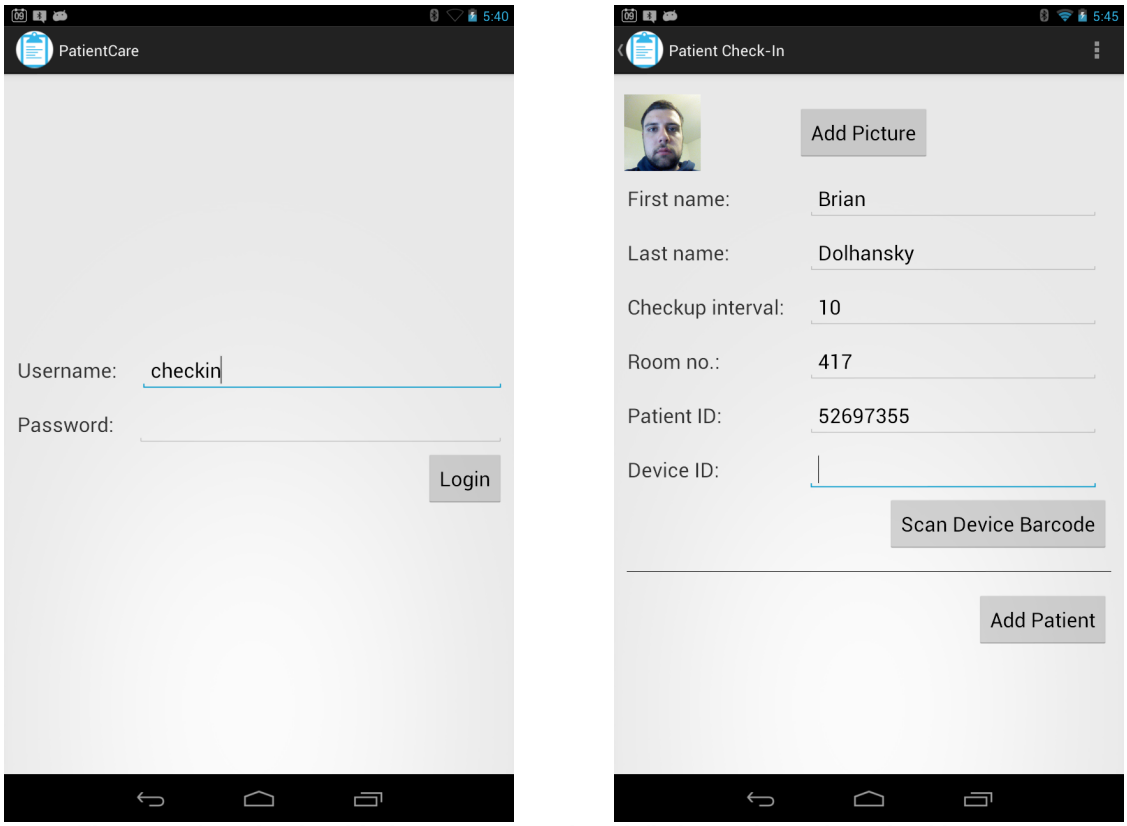


Figure 4 – Logging in with the username "checkin" shows the patient checkin screen.

Entering the QR code creates an IntentIntegrator object, which launches a barcode scanning Intent and returns the decoded result back to the activity.

When the patient information has been saved, it is stored in the global PatientManager, which is accessible from all activities.

Patient Status Updates

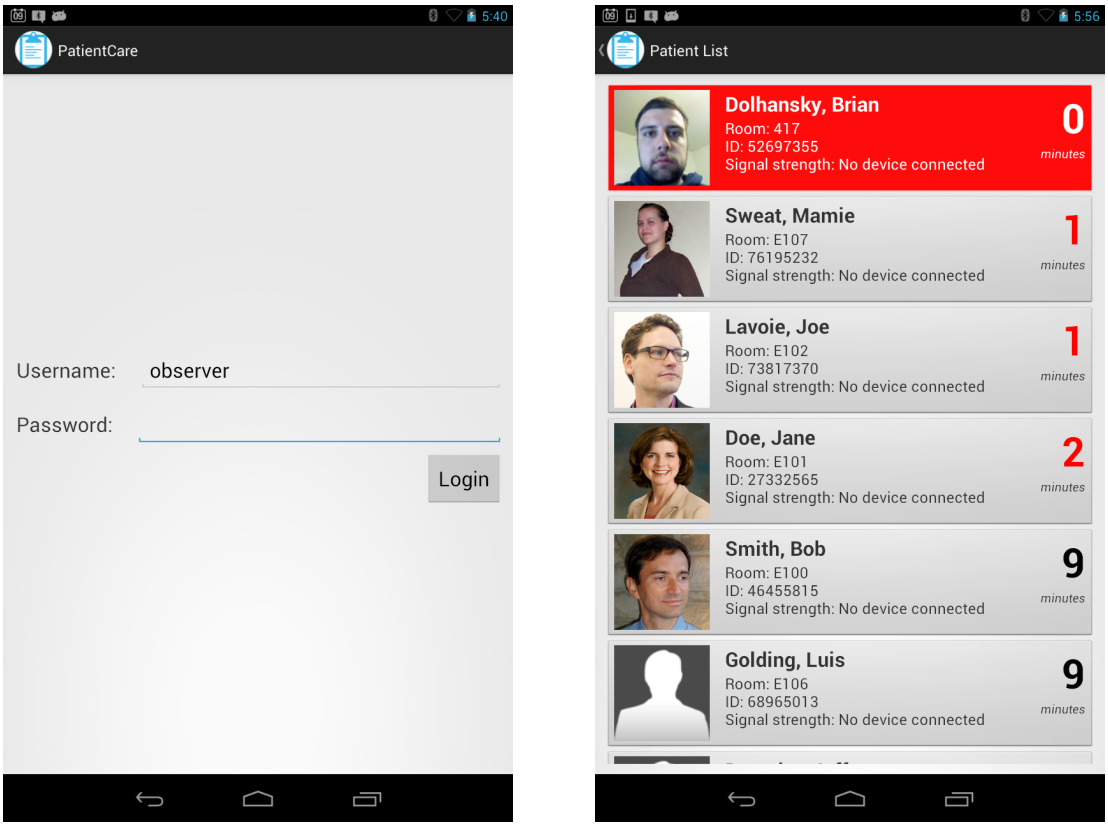


Figure 5 –Logging in with the username “observer” brings up the patient status screen. Each entry is clickable. Numbers to the right signify how many minutes that patient has until their next checkup. Red entries are patients that are past due for their checkup.

The OBSERVER will use this screen (in code, PatientListActivity.java). It can be accessed by logging in with the username “observer.” The patients are ordered by the amount of time they have left until their next checkup. A greyed-out button represents patients that have devices that are out of range. Patients that are past their checkup time are represented by a red button.

Pressing one of the patient buttons will record a checkup. Long pressing a button will allow the user to remove a patient from the observer screen, although they are still kept in the global PatientManager.

Technical Changes by an Admin.

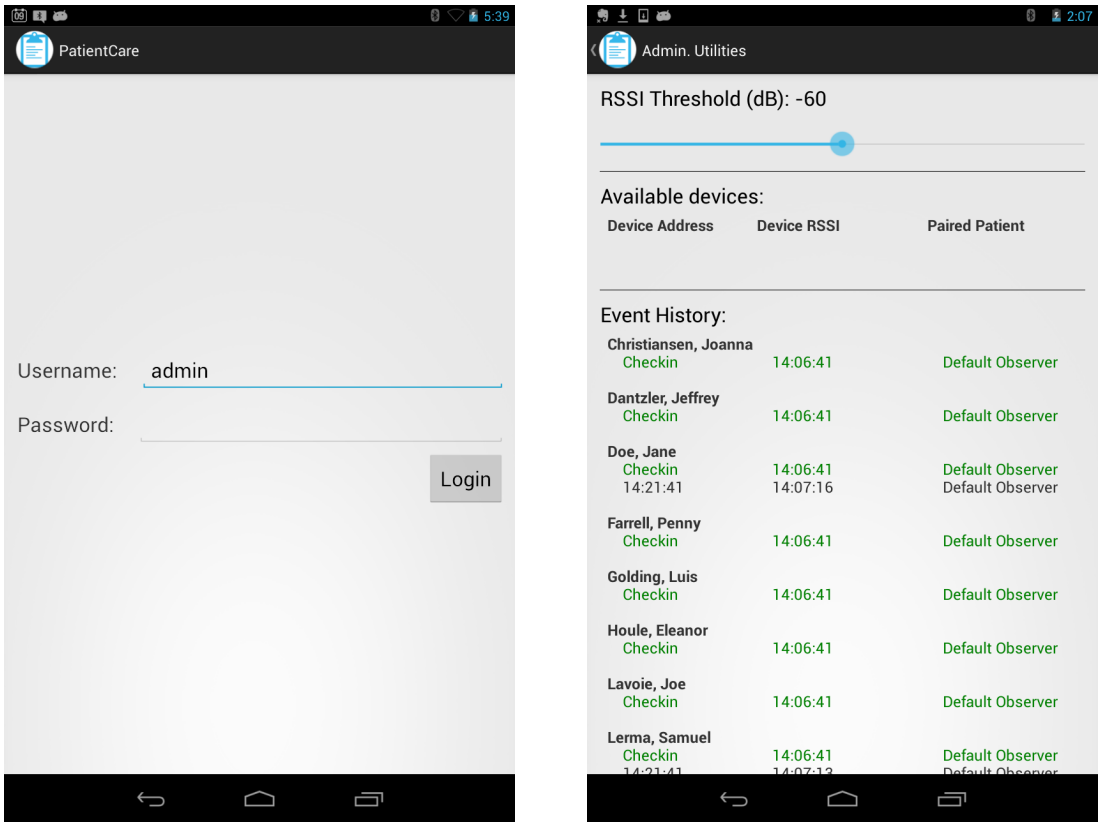


Figure 6 – Logging in with the username “admin” brings up the admin screen, where it is possible to set the RSSI threshold for the tablet, as well as see available BT devices and the checkup history.

This activity can be accessed by logging in with the username “admin” (in code, AdminActivity.java). From here, the global received signal strength indication (RSSI) threshold can be set. The tablet receives periodic updates from a BT device, and each of these updates includes the signal strength that the update was received at. Devices that are far away will have a lower RSSI. Patients whose devices are far enough away such that their update RSSI falls below the threshold will not be able to be checked in.

A list of all available BT devices are shown in the next section of the table. If their MAC address is paired with a specific patient, it will appear here.

Finally, the event history for each patient is shown. The event time is shown in the middle column and the observer (in a real situation, the OBSERVER) is shown in the last column.

The event type is shown in the first column. Patients can undergo one of three events: checkin, checkup, or removal. Instead of showing “CHECKUP” checkup events have the required checkup time in the first column. If a checkup event is late, it will be shown in red.

5. Backend Design

The underlying system architecture is fairly simple. There is a centralized container PatientManager for all patients stored in Patient objects (the container is located in

GlobalContainer.java). When patients are checked in, they are stored in this manager. All activities have access to the manager in order to populate their appropriate lists. In addition, this container maintains the internal checkup countdown. If a counter reaches 0, the PatientList activity will sound a notification (as this is the only place where notifications should be given. The unit clerk or the admin do not need access to patient checkup reminders).

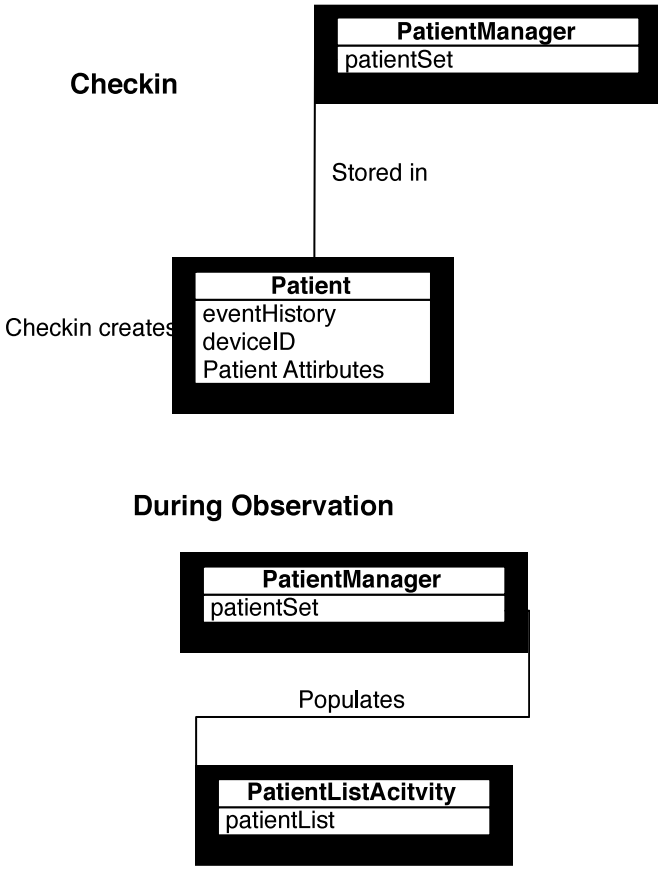


Figure 7 –When a patient is checked in, they are added to the global PatientManager. The patient list populates itself from the PatientManager at the observer screen.

There is also a custom event system. Events are stored in an individual EventHistory object for each Patient. Whenever an event occurs to a patient, it is stored in their EventHistory. This list is traversed for display in the Admin activity.

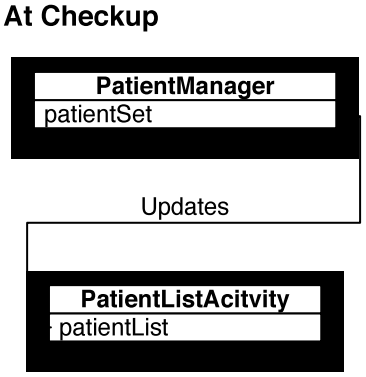
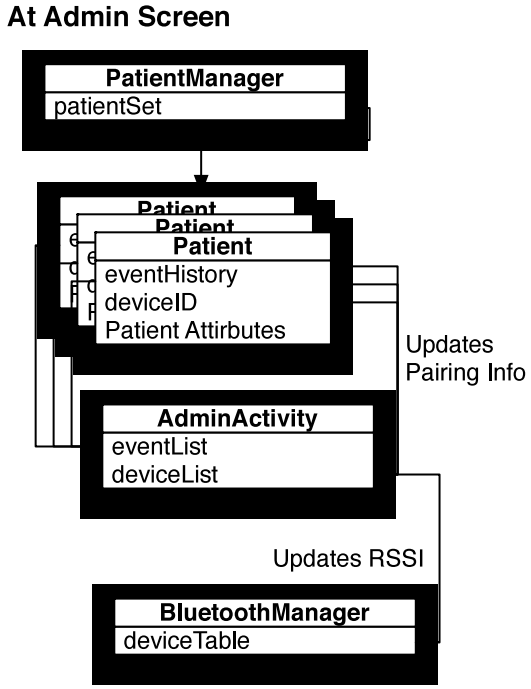


Figure 8 – When an event is recorded (such as checkup or removal), the appropriate activity updates the global patient list.

Finally for the hardware interface to Bluetooth Devices, there is a BluetoothManager object. This object constantly scans for Bluetooth devices and stores them in its device table. The admin activity then accesses the BluetoothManager for a list of devices, and the PatientManager for a list of patient pairings to those devices.



In the future, functionality should be added to PatientManager to allow syncing to an external database. This is not currently present.

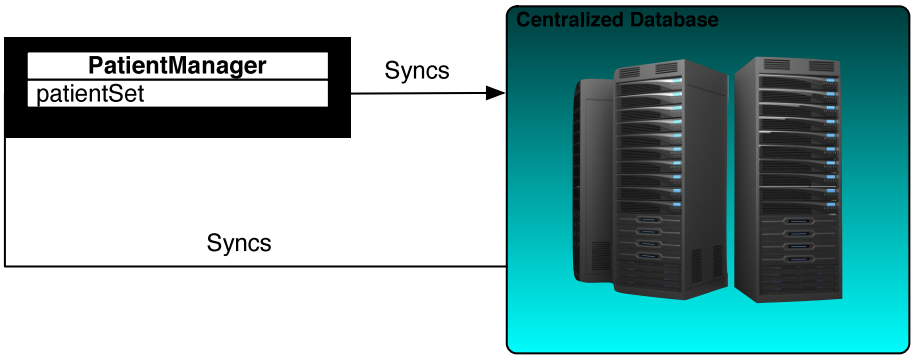


Figure 9 – Future functionality should include a syncing service to make sure the PatientManager is current across all devices.

6. Hardware Notes

I was not responsible for developing the BLE chips, so this is just a short overview of the hardware theory used. Each Bluetooth device is programmed to broadcast at the same power level. Therefore, the received signal strength indication (RSSI) at the tablet should vary only with distance to the device, given that it is operating in an open environment. In a closed environment (such as a hospital) reflections and interference complicate the matter. This is a topic for future investigation, but I assume that we should set a very high threshold so that patients can be checked up only when there is a direct line of sight from the tablet to the wristband.

I *was* responsible for developing the barcode standard. Each wristband should have an affixed QR code. The QR codes should encode the MAC address of the BT device in the following form:

PatientCare,00:00:00:00:00:00

where 00:00:00:00:00:00 has been replaced with the appropriate MAC address. Then, when the barcode is scanned at the checkin screen, the MAC address is assigned to that particular Patient object, so that it can be determined whether or not that patient is in range for a checkup.

No pairing with the Bluetooth devices is actually needed. SW only uses the MAC address of each device. In the future, information could be stored in the wristband microcontroller memory, but this would complicate the current architecture.