Symptom Management App & Server Design Description

Coursera Capstone Project

26 November 2014

Table of Contents

Basic Project Requirements

1. The Symptom Management App supports multiple users via individual user accounts.

Three (3) user roles are supported by the SMA and its server-side implementation. These include Patient, Physician and Administrator. Each patient and physician/doctor has their own account and they can login using these accounts. There are no limitations on the number of patients or physicians that can be created to access the app.

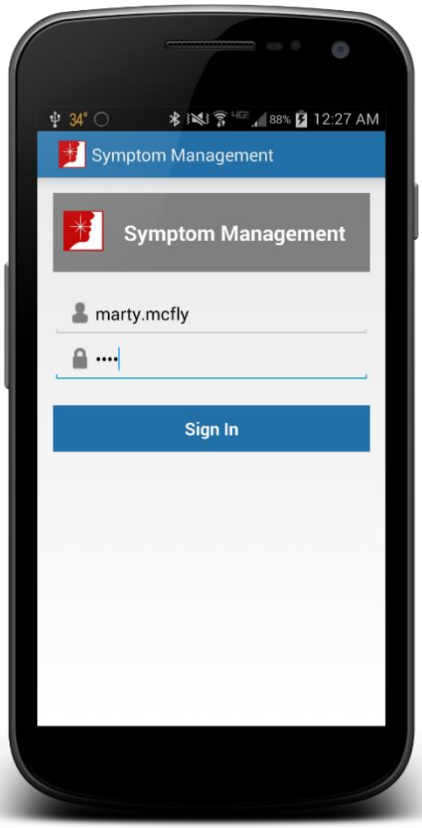


Figure 1. Symptom Management Login Screen.

Patient – A head or neck cancer patient receiving chemotherapy and radiation therapy for a 6-7 week period. The worst symptom for these patients is radiation-induced oral pain and sore throat. After 2-4 weeks of treatment, pain may increase from nothing to severe in a matter of days. Patients will use this app because they want their doctor to be able to give them appropriate care. Patients will want a simple app that allows them to track their pain and medication usage and send it to their doctor. They do not want to rely on their memory or use anything complicated. The patient requirements used to design this app:

- Track Pain
- Track Medication Usage
- Keep Notes for Later
- Reminder to Enter Data
- Quickly Enter Data
- Simple & Easy to Use
- Use Anywhere – Online or Offline
- Verify Data Entered

The Patient version of this app is documented in the section titled Functional Requirements – Patient Workflow & Check-In Process.

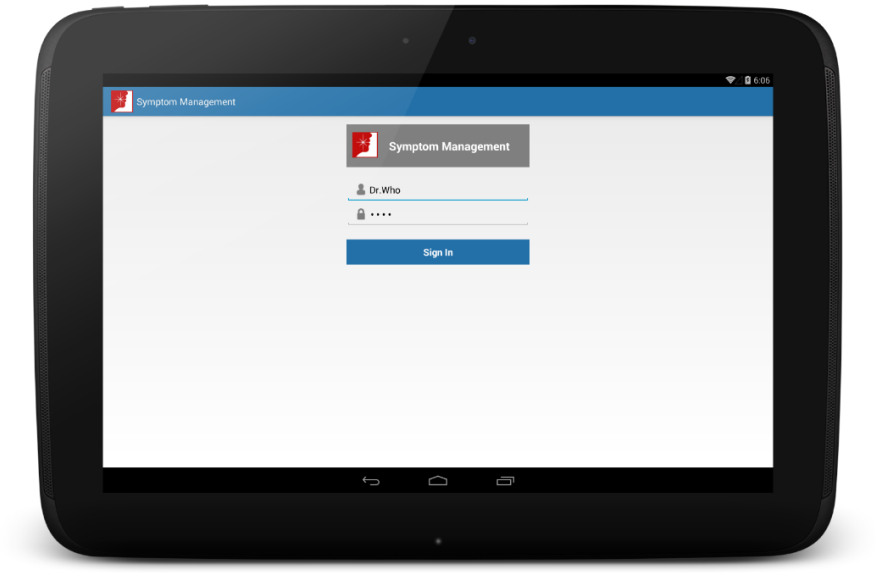


Figure 2. Physician/Doctor Login on a Nexus 10 Tablet.

Physician – A doctor who is caring for a head or neck cancer patient. The doctor wants to use the patient's pain and medication data to help the patient. The more accurate the information from the patient the better care that the doctor can give them. The doctor requirements used to create this app include:

- View List of Assigned Patients
- View Patient Pain Levels
- View Patient Medication Usage
- Notification of Severe Patients
- Visualize Patient Data for Quick Analysis
- Search and View Any Patient's Data
- View Patient's Prescriptions
- Add or Delete Patient Prescriptions
- Add New Medications to Prescribe

The Physician/Doctor version of this app is documented in the section titled [The Physician Workflow](#).

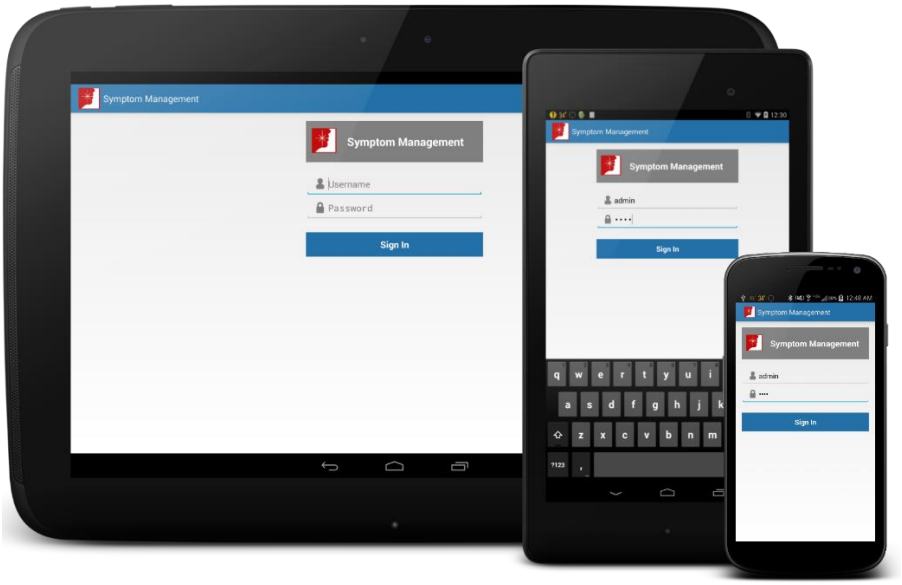


Figure 3. Administrator Login on variety of screen sizes.

Administrator – An administrator manages the patients and physicians accounts for the SMA. There are no official administrator requirements assigned for this project. The administrator role was created to fulfil the following needs:

- Add & Edit Patient Information
- Add & Edit Physician Information
- Assign Physician(s) to Patient
- Create User Accounts on the Server
- Assign Roles and Permissions to the User Accounts

Username :	admin
Password :	pass

There is only one administrative user supported. The administrator is currently the only user who is hard-coded in this project.

The Administrator version of this app is documented in the section titled The Administrator Workflow.

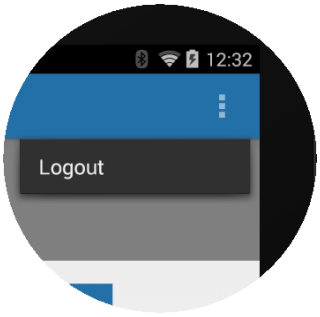


Figure 4. A menu option for logout is available for all users.

2. The Symptom Management App design contains at least one user facing function available only to authenticated users.

All user roles requires secure authentication to access any functionality in the SMA. This occurs via the Login Process (described in detail at this link). A logout option in the menu is available for all users if they wish to close the app and re-authenticate later. This app does not currently have an interface for changing passwords. This could be a future enhancement.

3. The Symptom Management App design App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components: Activity, BroadcastReceiver, Service, ContentProvider

A list of all the classes used in this project are found in the Symptom Management App Class Hierarchy Section. This information was generated using the Javadoc tool in Android Studio. It show the usage of all of the required four components in this project.

Activity – This project uses numerous activities for the three different version of the app. Most activities implement fragments to allow the UI to be flexible and reusable.

BroadcastReceiver – The ReminderReceiver class is used by the Patient version of the app as part of the Reminder Notification processing.

Service Component – There are three services used in this app. A Sync Service, an Authenticator Service and a Reminder Service. A SyncAdapter is used to periodically sync both Physician and Patient information. See the SyncAdapter Processing section for details.

ContentProvider – The PatientContentProvider is only used as part of the Patient version of the app. It locally stores the Patient's tracking data using a SQLite database and manages access with a ContentProvider. The design decision to do this was to allow the Patient to track information offline or online. It is most critical that the patient tracking data is never lost. The Admin and Physician apps must have internet access to work and do not store anything to the device.

4. This Symptom Management App design includes a remotely-hosted Java Spring-based service.

For this project the server runs on the development computer the using Eclipse environment and hosted on the local network but not on the internet. The server uses Spring Boot and implements RESTful services for a client API that works with Retrofit. The database that this design implements is Mongo DB (see Symptom Management Class Diagram for database configuration). The app uses the client API to access the server's services.

5. This Symptom Management App design implements the server using RESTful services via HTTPS.

This is the list of APIs that are implement. Objects use JSON representation of the for Patient, Medication, Physician, Alert and UserCredential classes. Please see Class Diagram for description of these object. Note that because the database used is Mongo DB, the ids are String representations of Object IDs. Also all endpoints require authorization and therefore HTTPS.

Path	Available Commands	Authorized Roles
/patient	GET POST	Admin, Physician Admin
/patient/{id}	GET PUT DELETE	Patient, Admin, Physician Patient, Admin, Physician Admin
/patient/find?name="first last"	GET	Admin, Physician
/physician	GET POST	Admin, Physician Admin
/physician/{id}	GET PUT DELETE	Admin, Physician Admin, Physician Admin
/physician/{id}/alert	GET	Admin, Physician
/medication	GET POST	Patient, Admin, Physician Admin, Physician
/medication/{id}	GET PUT DELETE	Patient, Admin, Physician Admin, Physician Admin
/medication/find?name="name"	GET	Patient, Admin, Physician
/alert	GET POST	Admin Admin, Physician
/alert/{id}	DELETE	Admin, Physician
/credential/find?username="username"	GET	No Roles (Pre-Login usage)

6. This Symptom Management App design allows users to navigate between 3 or more user interface screens at runtime.

The design and figures in the following Functional Description section describes the applications screen flow.

Figure 5. This icon found on the Patient Notes screen allows the patient to capture an image and store it on the device.

7. This Symptom Management App design uses the advanced capability of both multimedia capture and touch gestures.

The patient version of the app has an option to "Add Notes." The notes screen includes an image button that when clicks allows them to take and store a photo with their device's camera. They can later show the doctor the images stored on their device if they wish. See the section on tracking additional information. The Picasso library is used to capture and store the images.

The physician has the capability to use touch gestures when viewing the patient graph data. This is described in the Physician Workflow section. The AndroidPlot.com library is used to create and process these graphs.

8. This Symptom Management App design supports at least one operation that is performed off the UI Thread in one or more background Threads of Thread pool.

This app implements AsyncTasks to request and obtain data from the cloud server via the service. The design implements a SyncAdapter to sync the patient's data with the cloud. Please see the source or classes used to show this. You can also view the SyncAdapter processing later in this document.

Functional Description and App Requirements

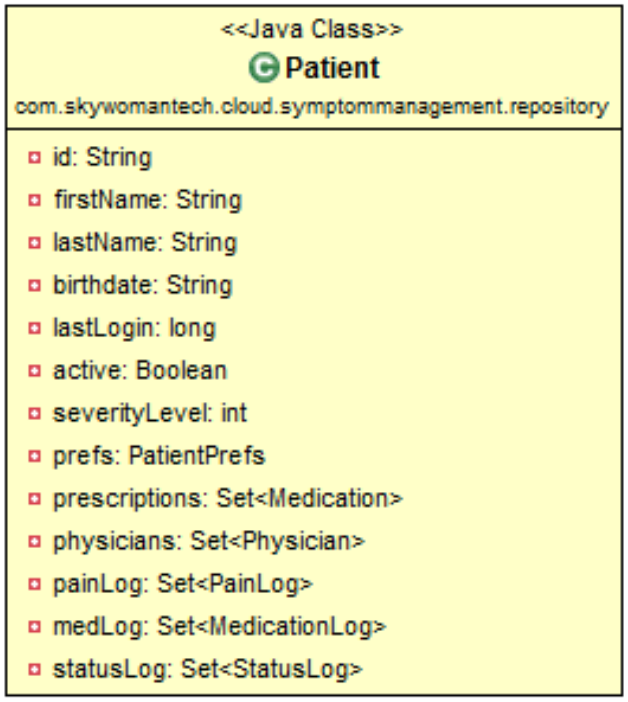


Figure 6. Patient Class.

1. Requirement: The Symptom Management app identifies a Patient as a user with first name, last name, date of birth, a (unique) medical record number, and possibly other identifying information). A patient can login to their account.

This figure shows the Patient class which is designed to include these fields along with additional fields needed to implement the app. The object id is the same thing as the unique medical record id. This id is displayed on the physician screens (see Physician Workflow).

The user login process section later in this document discusses how the patient logs into the app. This is also discussed in the Basic Requirements section.

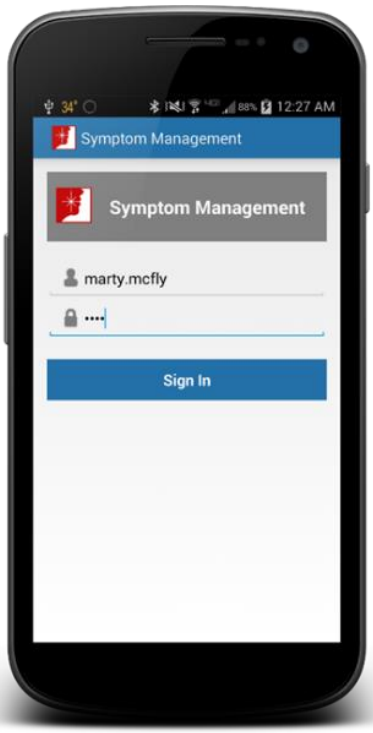


Figure 7. Patient Login Screen.

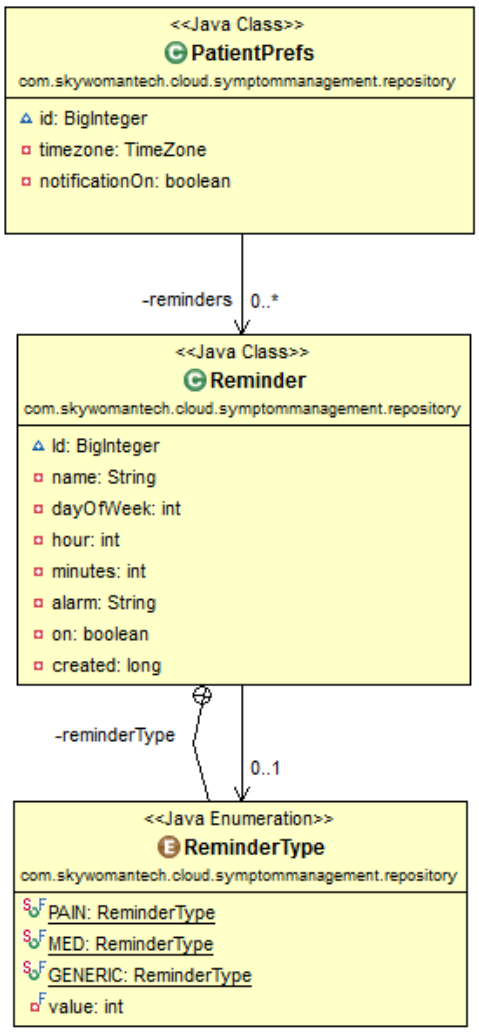


Figure 8. Reminder class

2. Requirement: App defines a Reminder as an alarm or notification which can be set to patient-adjustable times (at least four times per day).

The Reminder class (Figure 3) is designed to allow the app to set alarms at a specified time of the day and to allow the user to give a name to each alarm for identification. It has additional fields for future enhancements. The initial implementation is simply to enter a name and the time of day.

There is no enforced limit to the number of reminders the patient can create. New reminders can be added by the patient. Each reminder can be turned on or off or deleted. The name and times can also be edited. An alarm manager notifies the Patient when an alarm is received and sets the screen flow to a check-in flow. Please see screen capture below.

Additional screen captures can be seen in the Patient Workflow & Check-in section on Reminders.

A set of Reminders is a member of the PatientPrefs class. You can view all of the class relationships in the Class Diagram.

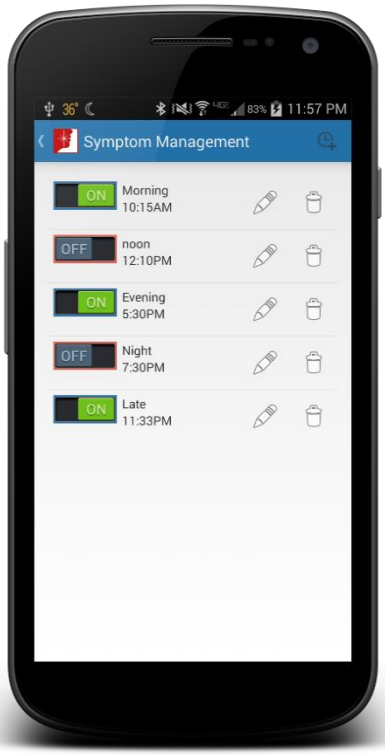


Figure 9. Reminder settings screen.

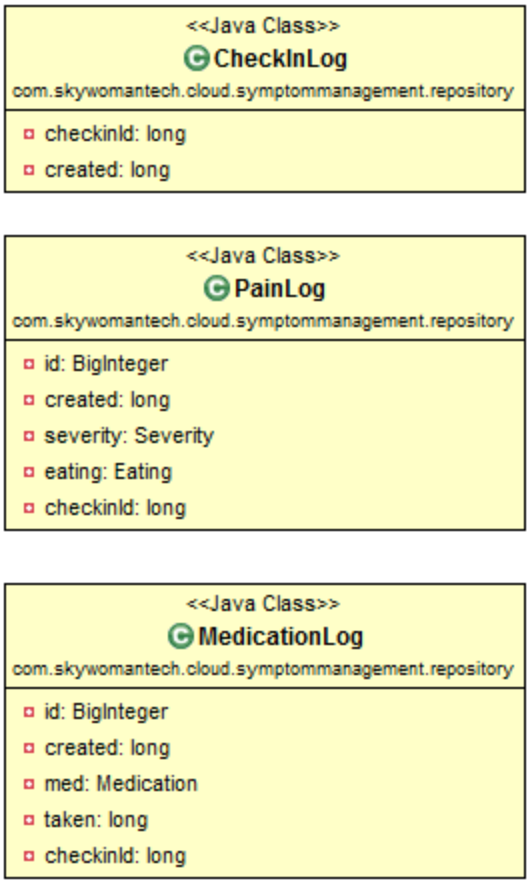


Figure 10. Check-In information as stored in the PainLog, MedicationLog & CheckInLog classes. They each us a checkinId to connect the information to a single check-in.

3. Requirement: A Reminder triggers a Check-In, which is defined by the app as a unit of data associated with a Patient, a date, a time, and that patient's responses to various questions at that date and time.

The design approach I took for this does not use a single check-in object but instead it stores the required Check-In data (patient, date, time, and responses) in multiple objects called "logs." The Patient object associated with the logs holds a list of each type of log.

This means that the Check-In is associated with a Patient because it is stored in a Patient object. The Check-In log holds the created date and time of the check-in. The responses are stored in a pain log and medication logs. All logs are associated with a check-in Id.

The app separates the answers from the check-in questions into "Pain" related data and "Medication" related data. It associates them together with a "checkinId" field. Therefore logs with the same checkinId can be said to be a single check-in. There is also a CheckInLog class that store the check-in id along with a timestamp. This log can be used with the other logs for reporting purposes and tracking the number of logins.

I believe that splitting the check-in information into multiple objects gives this app more flexibility for the user. This implementation allows the patient to enter either medication tracking or pain symptoms any time they feel necessary without going through the "official" reminder-prompted check-in process. I think that without this flexibility the app may become frustrating for the user or too complicated.

When a patient is using the app, the implementation stores these logs on the device in a SQLite database that is accessed with a Content Provider. This keeps tracking data from being lost when there is no internet available. As compared to when a doctor or administrator is using the app, the functionality requires an internet access and no patient data is stored locally on the device.

The SyncAdapter periodically tries to sync these pain and medication logs with the server so that the doctor can access them. But a method to immediately sync the data is called after the pain and med logs are entered so the check-in data will be immediately pushed to the server where the physician can access it. Since the physician must be connected to the internet for their version of the app to work, they should get the information quickly.

See the SyncAdapter Processing section for more information.

The Patient Workflow & Check-In Process

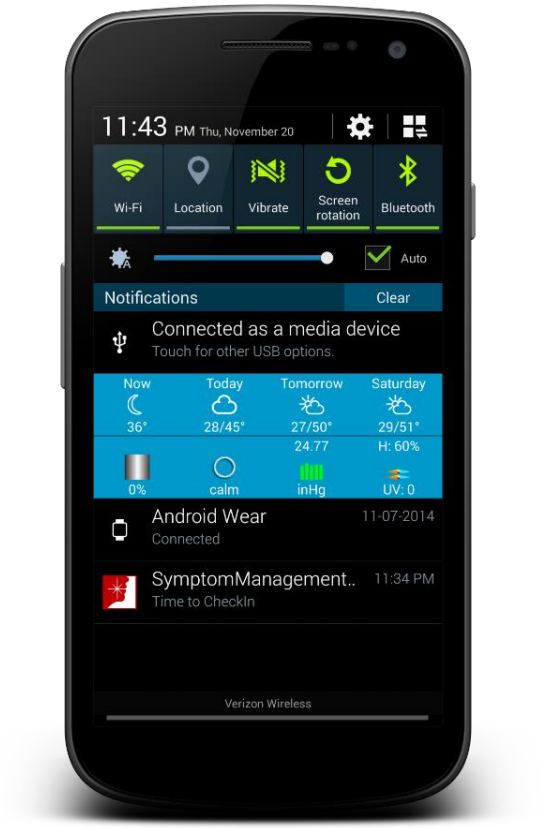


Figure 11 A Notification will be sent to Patient when it is time for a check-in.

This section addresses Patient workflow and the check-in process.

A check-in is triggered by the alarm manager and a notification is displayed. A screen capture of the notification can be seen in this picture. In tells the patient that it is time to check-in.

The user can then click on the notification and it will take them directly to the check-in process. Or they can open the application and the application will automatically start the check-in process.

Each check-in has a unique check-in identifier for the check-in. All data associated with the check-in has the same identifier attached to it.

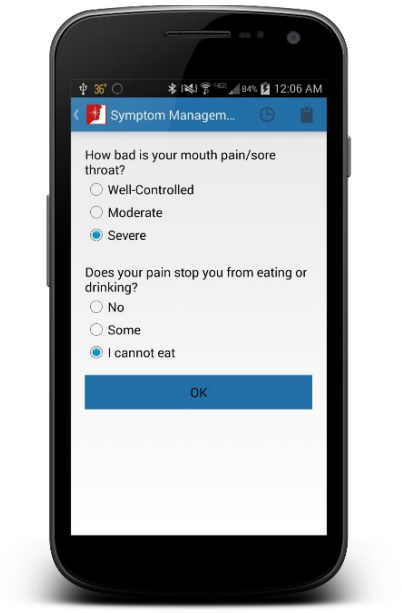


Figure 12. Pain related information for requirements 4 and 8.

4. Requirement: Check-In includes the question, "How bad is your mouth pain/sore throat?" to which a patient can respond, "well-controlled," "moderate," or "severe."

This figure shows the first screen that will be displayed during the Check In process. This screen is displayed to the patient directly after the patient acknowledges the reminder. When the OK button is pressed then the next screen with medication questions is presented.

This data is stored as a pain log in the patient's record. Its check-in id will associate it with the check-in log and the medication logs that follow and all together these logs make a check-in complete.

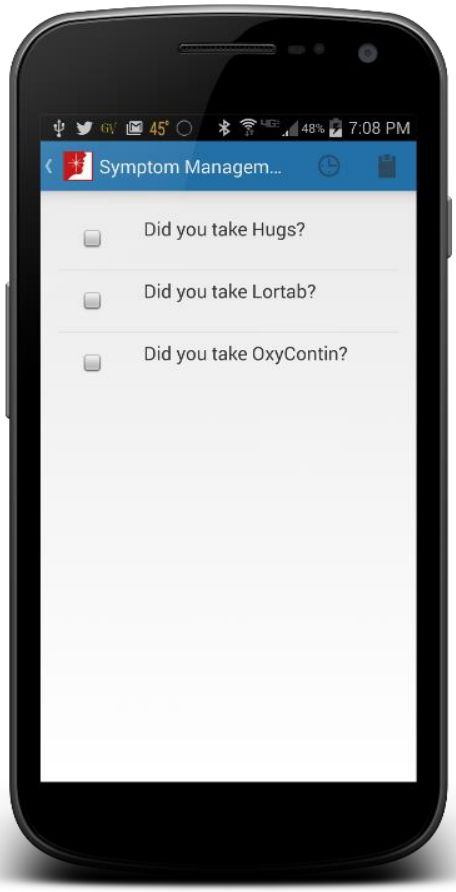


Figure 13. Medication related user input screen for Check in process. This is a custom list.

5. Requirement: Check-In includes the question, "Did you take your pain medication?" to which a Patient can respond "yes" or "no".

After entering the pain severity information shown in the above screen, then this screen is displayed. The list of questions is created from the list of prescriptions (medications) that the doctor has assigned to the patient.

So instead of saying ***"Did you take your pain medication?"*** the list will ask ***"Did you take [medication name]?"*** for each of the assigned medications. This figure shows a screen capture of a patient with 3 prescriptions. So there is a question for each of the prescriptions.

Instead of displaying the words Yes or No this interface uses a checkbox for the patient to click if they have taken the medication and wish to enter the time taken (see requirement 7 below). If they leave the box unchecked then the answer is no.

6. Requirement: A Check-In for a patient taking more than one type of pain medication includes a separate question for each medication (e.g., "Did you take your Lortab?" followed by "Did you take your OxyContin?"). The patient can respond to these questions with "yes" or "no."

As mentioned in the response for Requirement 5, this screen is a scrolling list of questions and is created from the list of prescriptions (medications) that the doctor has assigned to the patient. Each time a user checks a list item the response is stored as a single medication log in the patient's record. The same check-in id is assigned to all of the medication logs during one check-in because they are all in the same check-in.

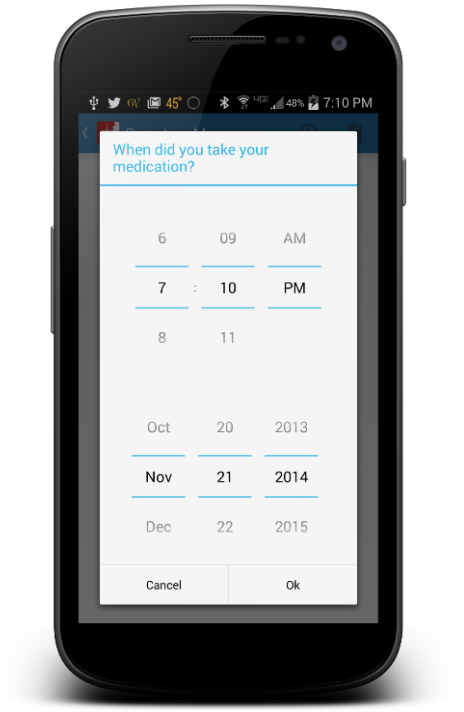


Figure 14. User input screen for date and time medication was taken.

7. Requirement: During a Check-In, if a patient indicates he or she has taken a pain medication, the patient will be prompted to enter the time and date that he or she took the specified medicine.

In the previous screen, when the patient checks the box indicating that they have taken the medication then this screen shown in this figure will be displayed. The date and time will default to the current time. This allows the user to just click the OK button if they have just taken the medication. Otherwise they can dial the correct time and/or date and then save it. When the OK button is clicked this will return to the previous screen and display the entered time/date on the second line of the list view.

This data is stored in a single medication log along with the medication name and the log is assigned a check-in id that associates the data with the current check-in.

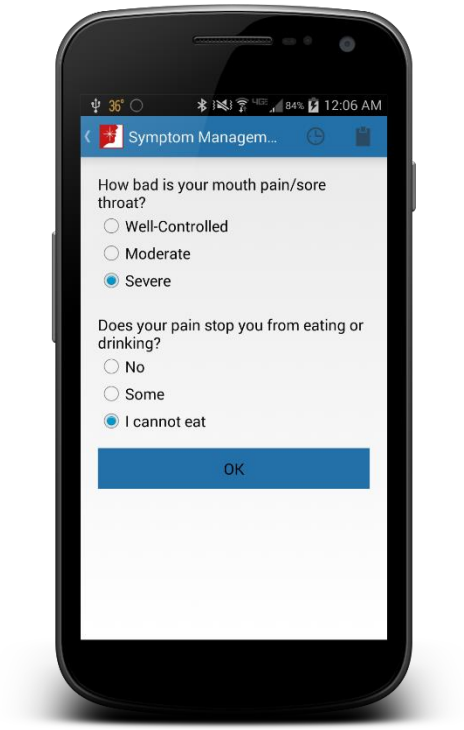


Figure 15. Entry screen for pain information.

8. Requirement: During a Check-In, the patient is asked "Does your pain stop you from eating/drinking?" To this, the patient can respond, "no," "some," or "I can't eat.

This screen which is also discussed in Requirement 4. It is the first screen that is display during a check-in. This information is stored with the information from Requirement 4 in a single pain log. This pain log has a check-in id that associates with the current check-in.

Additional Patient Tracking Options

The following sets of screen design images show how the patient can use the Symptom Management app to track additional pain and medication data. The app uses many of the same screens from the check in process.

For these screen flows, the patient is logged in and there is no currently activated alarm and therefore no current check-in id is available. They are offered three main options. These options allow them to enter pain symptoms, track medication taken, or add a note and/or image to share/discuss with their doctor.

Patient Tracks Pain Symptoms

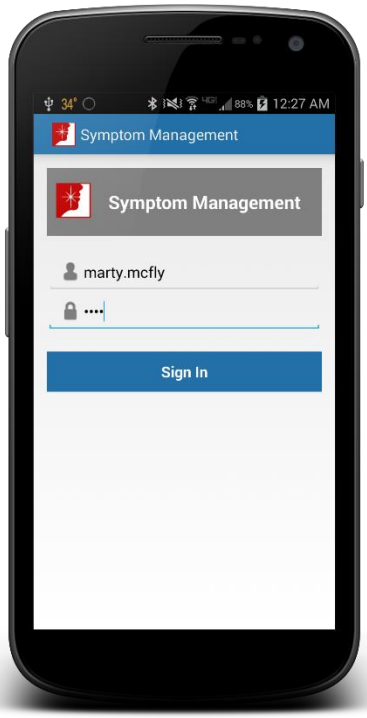


Figure 16 Patient Logs In and there is no current check-in.

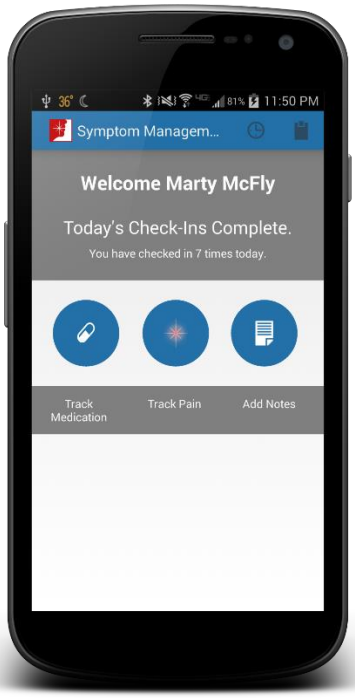


Figure 17 Patient Chooses Track Pain

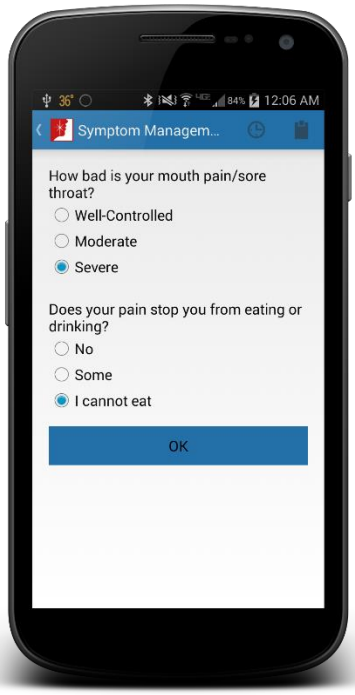


Figure 18 Patient Enters Pain Symptoms and presses OK to store the information.

Patient Tracks Medication Usage

Figure 19. Patient chooses Track Medication.

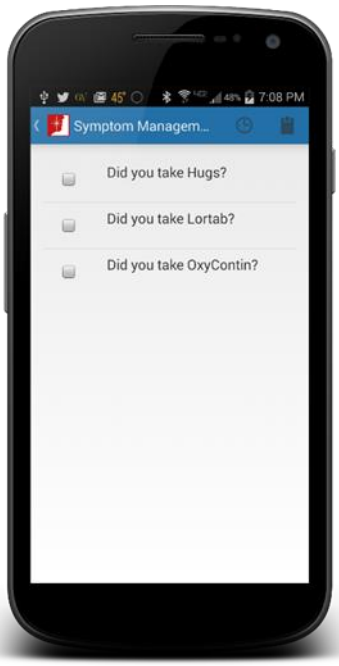


Figure 20. Patient chooses medication(s) that they are tracking.

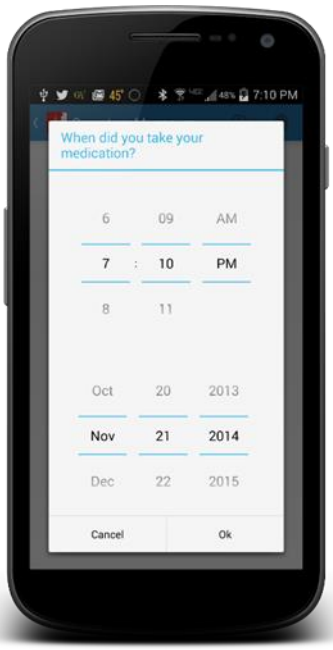


Figure 21. Patient enters time the medication was taken. The time defaults to the current time so they can just hit enter if they are taking it now. OK returns to previous screen.

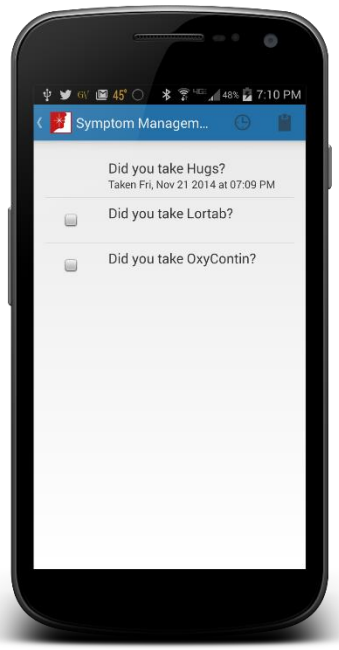


Figure 22. The summary line show the time that the medication was logged as being taken.

Patient Tracks Other Symptoms or Captures Images in Status Notes

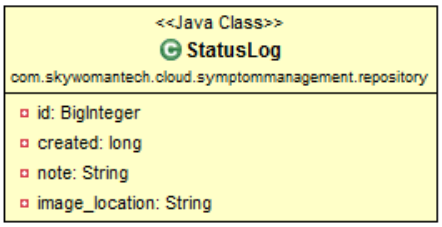


Figure 23. Status Log Class

Similar to the Pain and Medication Logs as used above, the Status logs can be entered anytime the patient feels the need to track additional information. The images are stored locally on the device only in an external storage directory call "Symptom Management." The patient can then view them or use other applications to share them with the doctor. The status will show that an image was taken and the doctor is able to request the image from patient by other means. The status log notes are stored locally and on the server. There is no check-in id associated with the status log notes because they are not part of the check-in process.

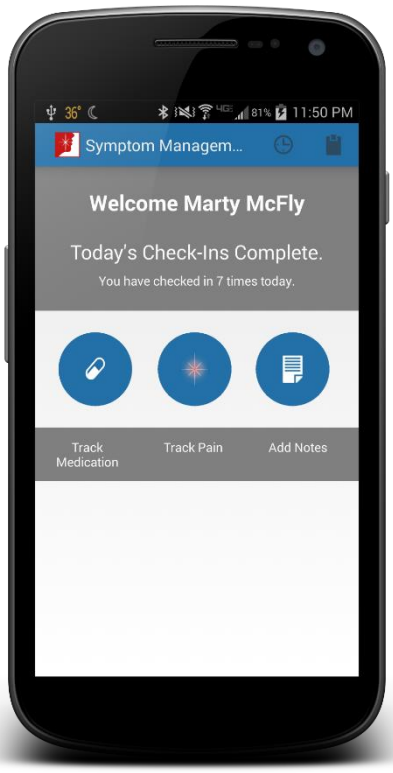


Figure 26. Patient chooses Add Notes.

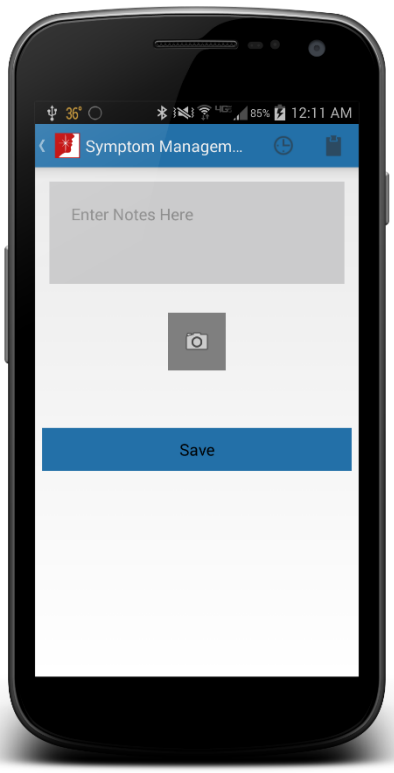


Figure 24. Patient enters Multi-Line Text. Can select camera to take an image that is stored locally on the device.

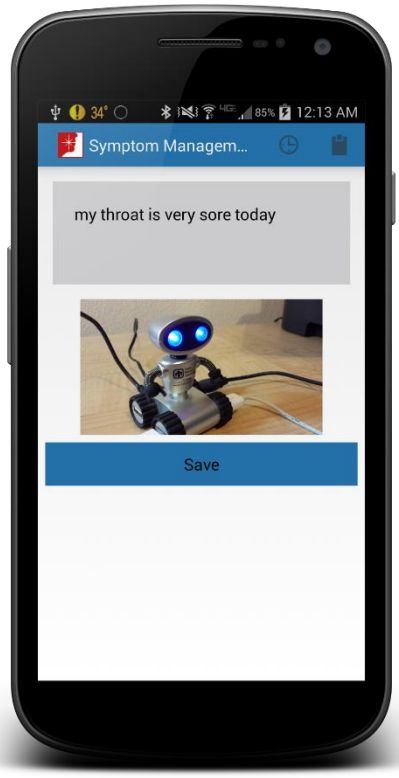
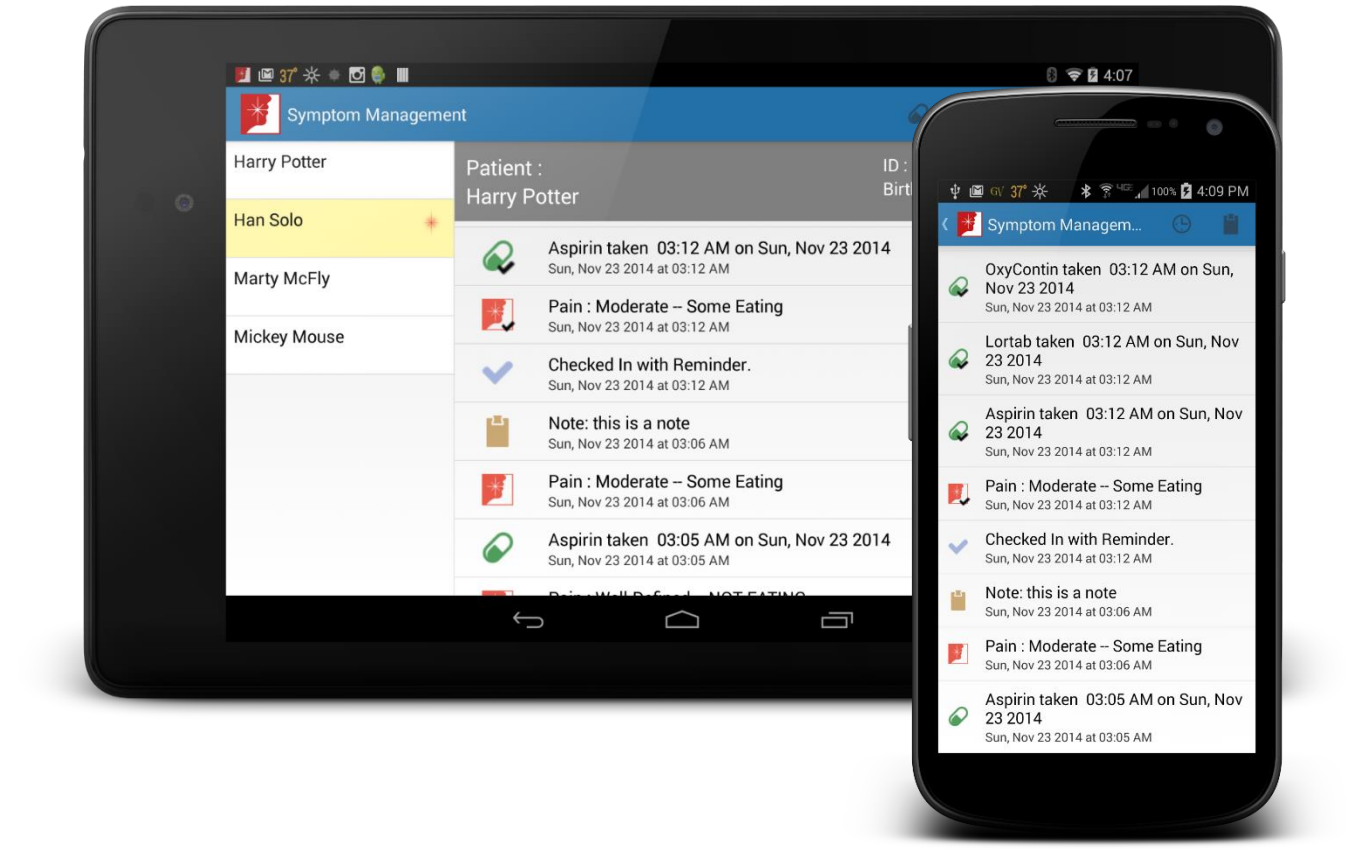


Figure 25. Patient selects the save button to save the log to the patient's record.

Patient Views Tracking Logs & History

Both the Patient and Doctor versions of the app include the capability to see a History Log. These are a list of the combined Pain, Medication, Notes and CheckIn Logs. Each item in the list represents a log type. The icon indicates the type of log. If there is a checkmark next to the icon then it means the log is associated with an official check-in. These logs are a good way for the Patient to verify that his information was entered and is an additional way for the doctor to see the patient data. Logs are sorted by timestamps.



History Log Icon Meanings

Medication Log for Check In.

Pain Log for Check In.

Check In Log.

Patient Notes / Status Log. Not related to Check In.

Pain Log for Tracking Purposes. No associated Check In.

Medication Log for Tracking Purposes. No associated Check In.

Patient Reminder Settings

The following sets of screen captures show how the patient can use the Symptom Management app to set Reminders.

The Reminder is an option menu choice when the patient is logged in and not currently performing a Check-In process.

Figure 29. Reminder icon on the Patient's screen.

After selecting the reminders option, the patient is shown a custom list view. The list displays each Reminder that has been configured or none if empty. A switch indicates if the Reminder is active or not running. The patient can easily turn on or off the reminder. Each Reminder is able to be edited or deleted.

Another option menu item allows the patient to add new Reminders. When the options menu item is selected a new screen is displayed to allow the patient to give an identifying name to the reminder and select a time of day for the reminder to trigger. This same screen is used when editing the Reminder.

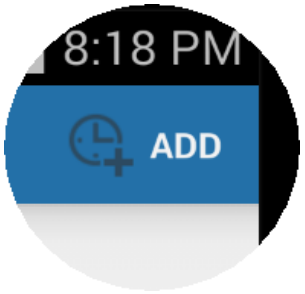


Figure 28. Add a new reminder option menu item.

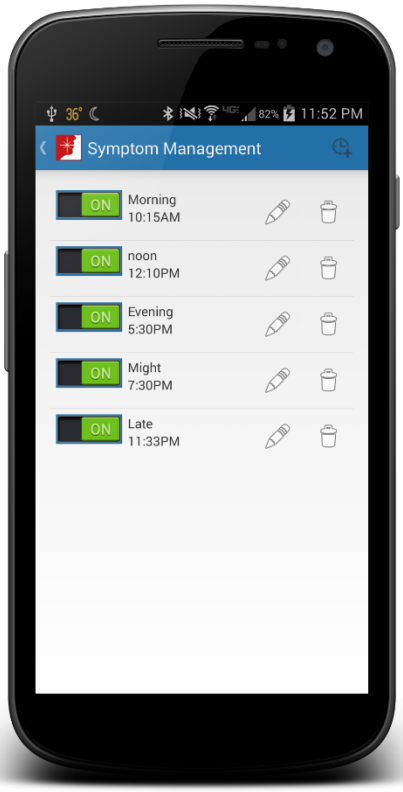


Figure 31. Show configured reminders.

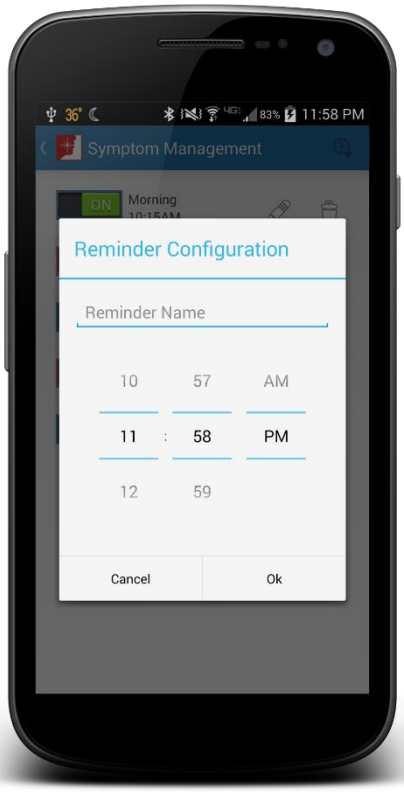


Figure 30. Adding a new reminder. Reminders can also be edited.

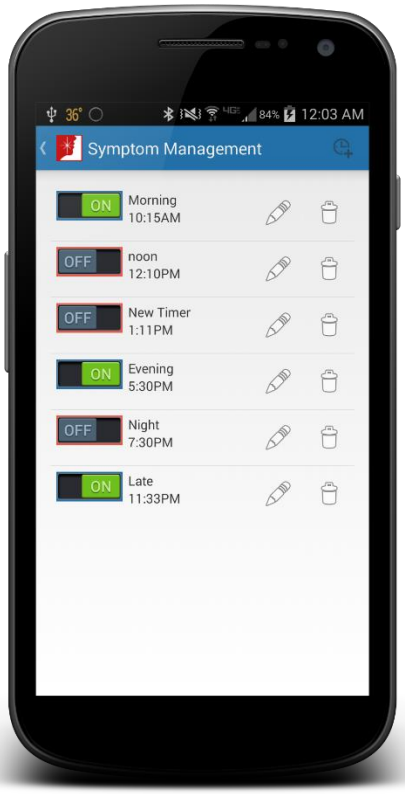


Figure 27. Turning reminders on and off by moving the switch on or off.

The Physician Workflow

Doctor Sign-in Process

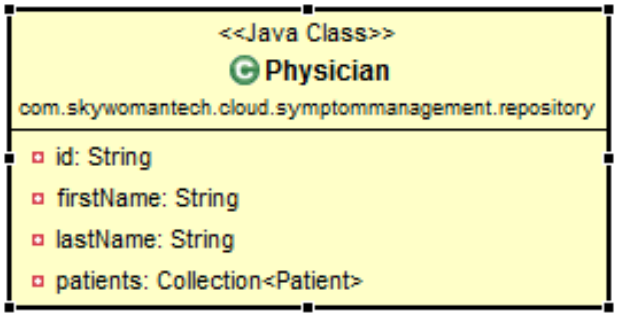


Figure 32. Physician Class

9. App defines a Doctor as a different type of user with a unit of data including identifying information (at least first name, last name, and a unique doctor ID) and an associated list of Patients that the doctor can view a list of. A doctor can login.

This figure shows the class diagram for the Physician object as designed. The physician object has a unique ID, first and last name and a Collection of Patients associated with it.

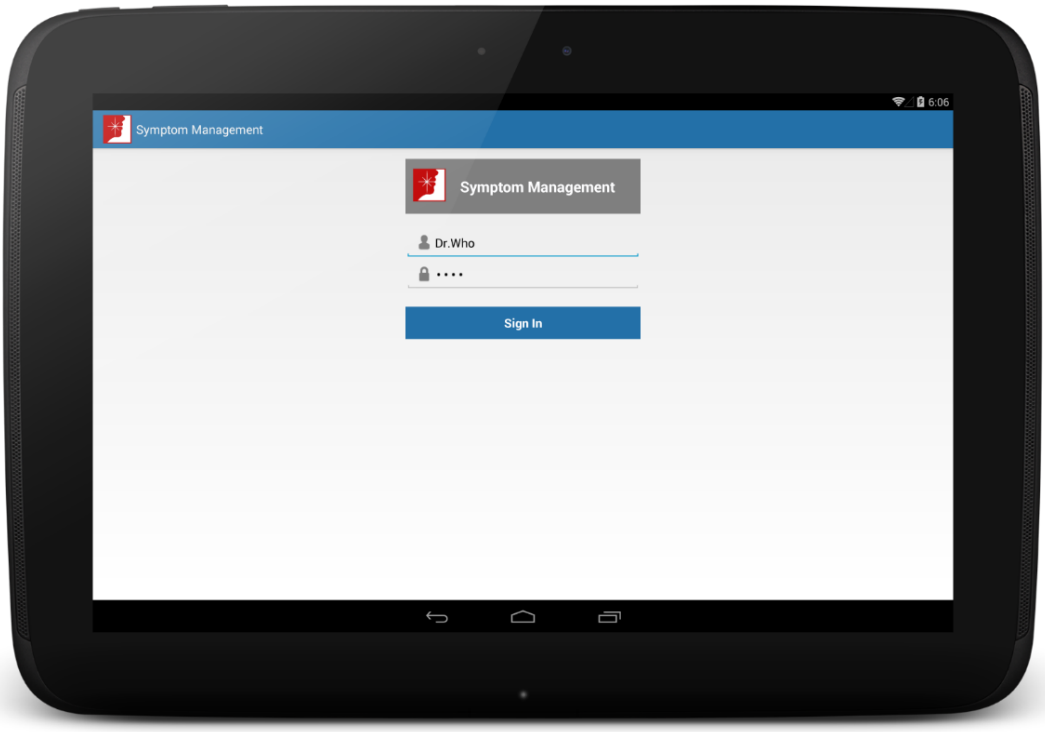


Figure 33. Doctor Login screen. It is recommended that doctors use tablets for this app. But smaller screens can also be used.

Doctors use the same login screen as the patient but they follow different screens and workflow than the patient users. Administrators add physicians or doctors to the database to allow them access and assign them the physician role. The app determines the user type with the UserCredential information. After logging in the next figure shows an example of what a physician would see on a tablet. This information includes a patient list.

Doctor Monitors Check-In Data / Doctor Dashboard

10. App allows a patient's Doctor to monitor Check-Ins, with data displayed graphically. The data is updated at some appropriate interval (perhaps when a Check-In is completed).

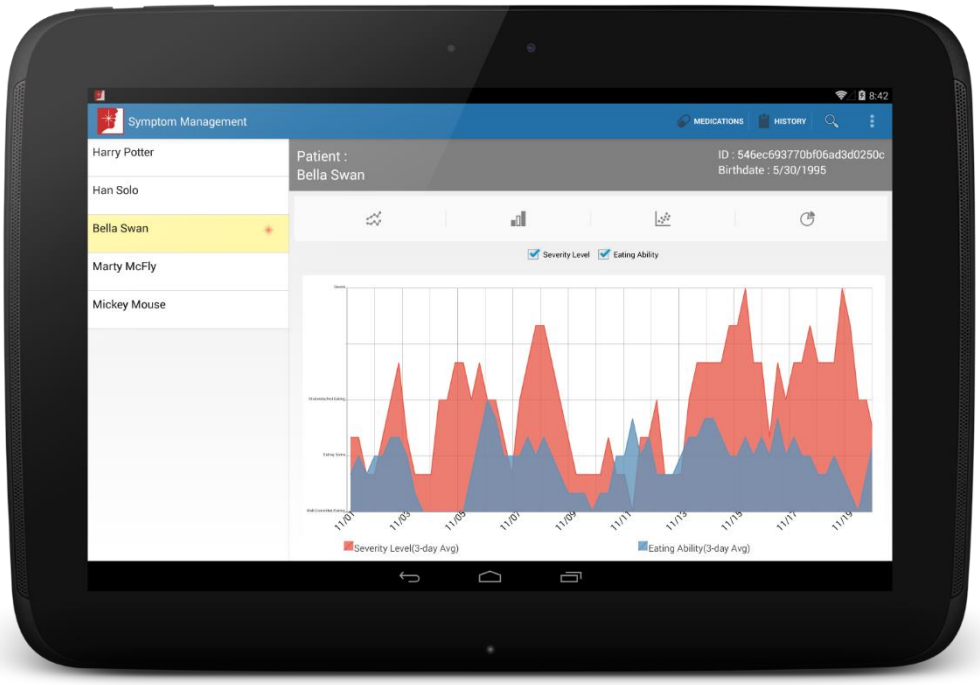


Figure 34. After login, the doctor can see a patient list and related patient details including graphed check-in data.

It is important to note that all **doctor** functionality requires internet access as it does not store any patient data on the local device but only in memory. Almost all activities will require fresh data from the server. This is how the doctor will have very quick updates. Also the patient app also tries to sync immediately after logging data.

After a doctor logs into the app, a scrolling list of patients associated with the doctor will be displayed. The association between physicians and patients was created previously by the administrator using the administrator version of the app.

If an alert exists indicating that a patient has experienced 12 hours of "severe pain," 16 hours of "moderate" to "severe pain," or 12 hours of "I can't eat" then the patient list highlights those patients so the physician can identify them immediately. A small icon is displayed along with changing the background color of the patient in the list. You can see this in the figure above.

Doctor Dashboard Graphs

In this implementation there are four types of graph data that can be viewed by the physician/doctor including a Line Plot, Bar Chart, Scatter Plot, and Pie Chart.

Figure 35. Icon for the line graph.

LINE PLOT - The first graph displays the eating and severity information transformed into a line graph and showing 3-day moving averages. This is calculated by using fuzzy logic and assigning values to each of the selections and then doing mathematical computations on these values. The pain severity values and the eating ability values for this graph are given different scales (eating ability is 50% of pain severity level). For each day the value of the current, previous and next day are averaged to get the plotted point. A color is used to fill in the area beneath the graph. This graph can be zoomed and scrolled using touch. The severity level and eating ability data points can be turned on and off using the checkboxes.

Figure 36. The same view on a Nexus 10 tablet where the graphs can be viewed larger.

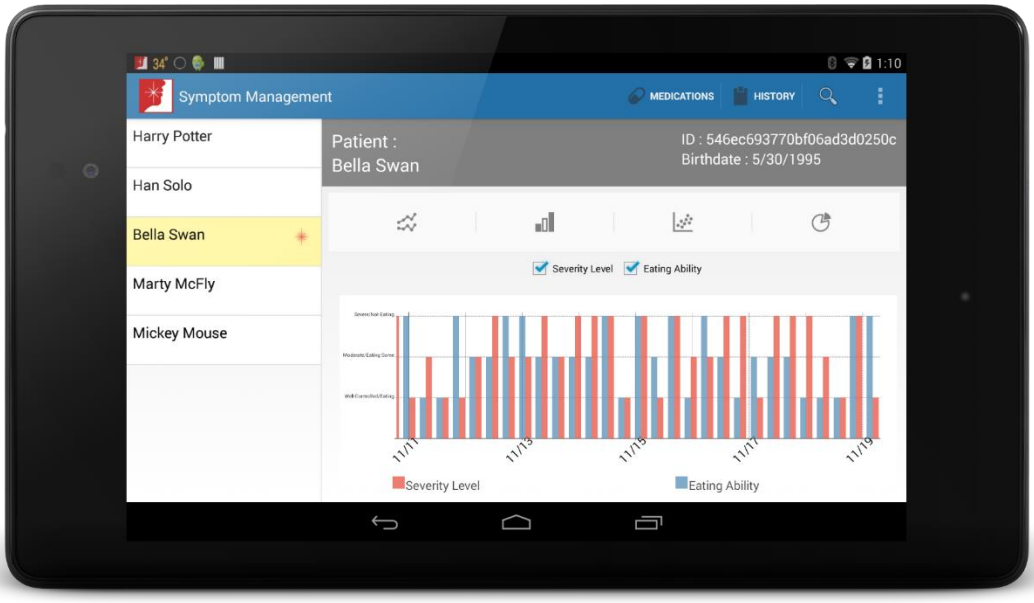


Figure 37. This is a bar chart graph viewed on a Nexus 7.

Figure 38. Icon for the bar chart

BAR CHART - The second graph displays the eating and severity information transformed into a bar chart and comparing the values side by side. This is calculated by using fuzzy logic and assigning values to each of the patient answers and then doing mathematical computations on these values. The pain severity values and the eating ability values for this graph are given on the same scale. A different color is used to differentiate between the two series in the graph. This graph can be zoomed and scrolled using touch. The severity level and eating ability data points can be turned on and off using the checkboxes.

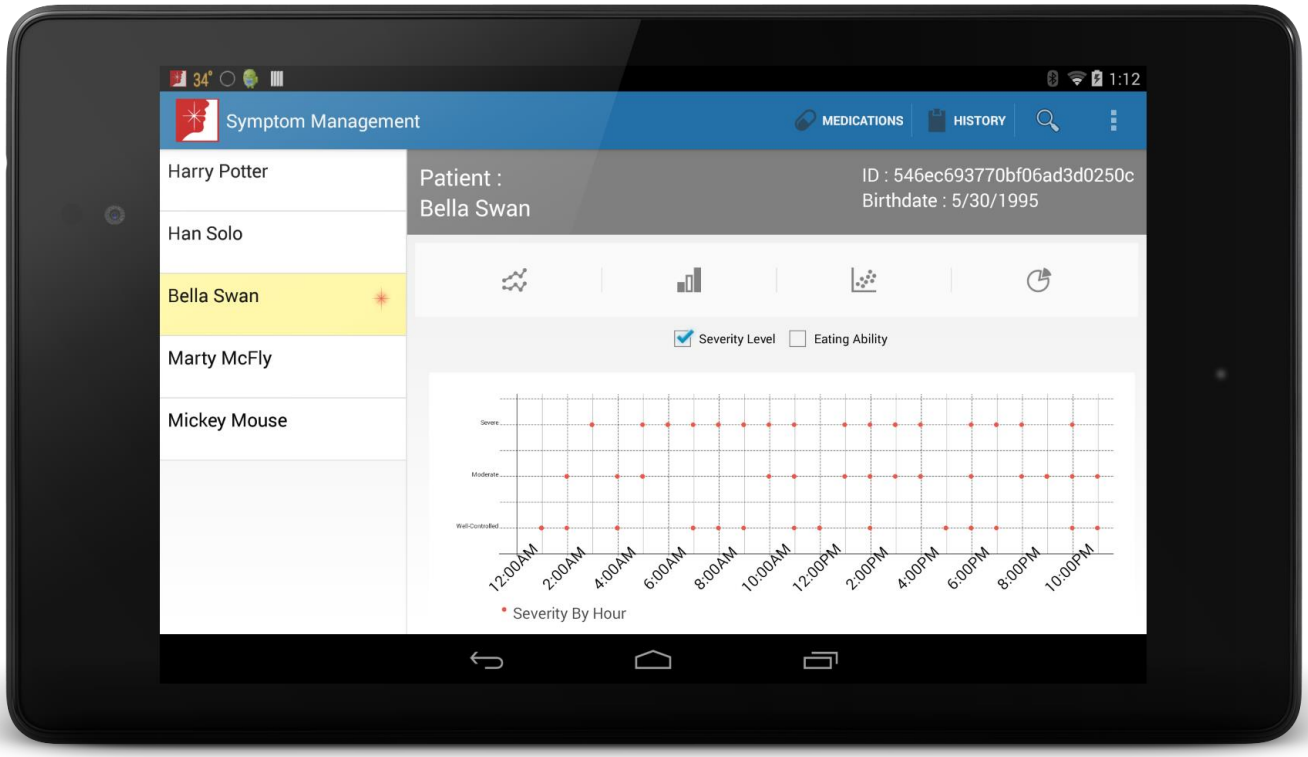


Figure 39. Icon for the scatter plot.

Scatter Plot - The third graph displays the eating and severity information transformed into a scatter plot showing the times of day that each answer was reported by the patient. The patient answers are given fuzzy values for both series and they use the same scale. The goal of this graph was to be more like a heat map but that was not easily done with the androidplot.com library in time for this project to be complete. This could be changed to a heat map for a future enhancement. This list can be zoomed and scrolled using touch. The severity level and eating ability can be turned on and off using the checkboxes. Future enhancements could show other relationships between the answers. e.g. How many answers given on a specific the day of the week or day of the month? Also medication times could be plotted here too. Just ran out of time for this project.

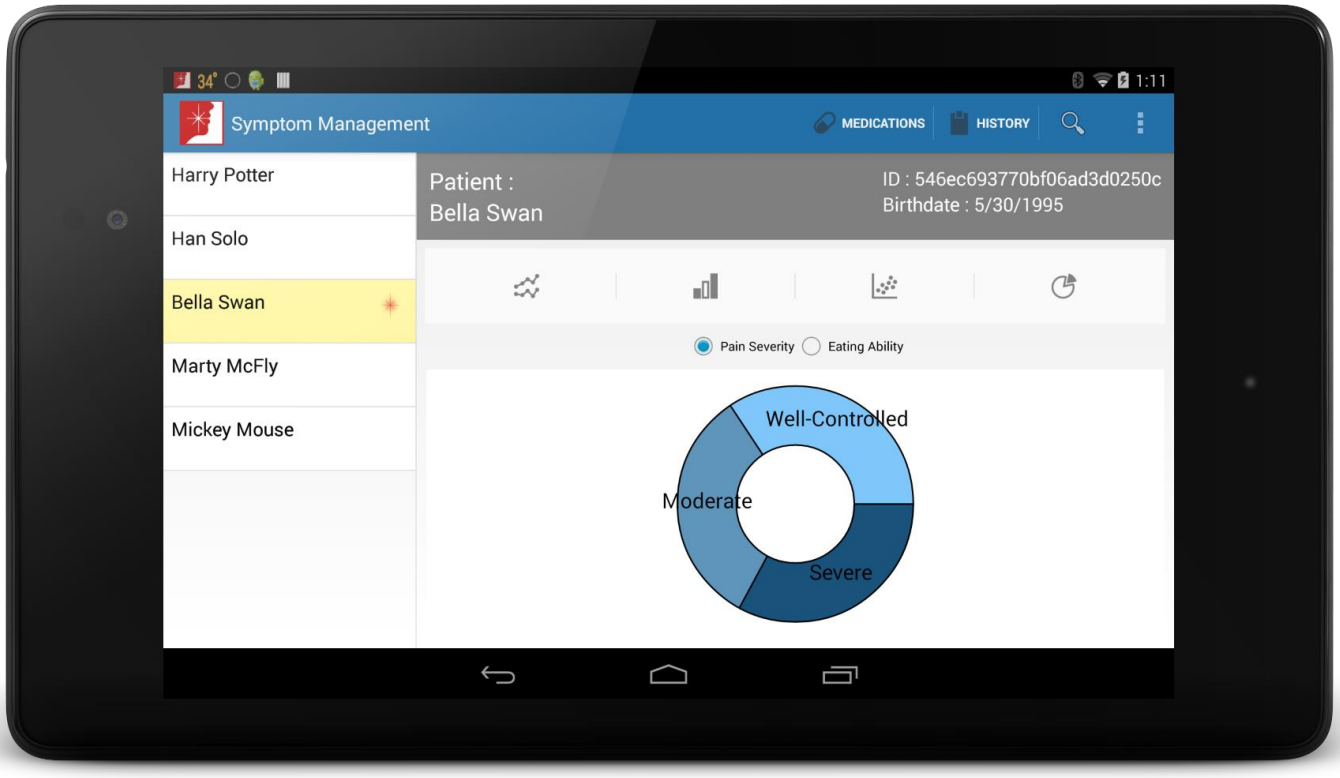


Figure 40. This graph shows the percentage of each response from a check-in for pain severity or eating ability. This view is a Nexus 7.

Figure 41. Icon for the pie chart.

Pie Chart - The last chart displays the percentage of patient responses given for eating ability or pain severity information and transformed into a pie chart. This chart uses radio buttons to choose which data series to display. The data is from all of the log records entered. Future enhancement could filter the percentages in other ways such as within the past week or month.

11. A doctor can search for a given Patient's Check-In data by the patient's name (an exact text search hosted server-side).

Figure 43. Search Patients Icon

On the doctor's app, there is option menu selection shown by the magnifying glass icon. This menu option allows the doctor to search for a patient by name. The search uses a server API to find the patient and sends back the patient information to be displayed to the physician. The name entered must exactly match the server name.

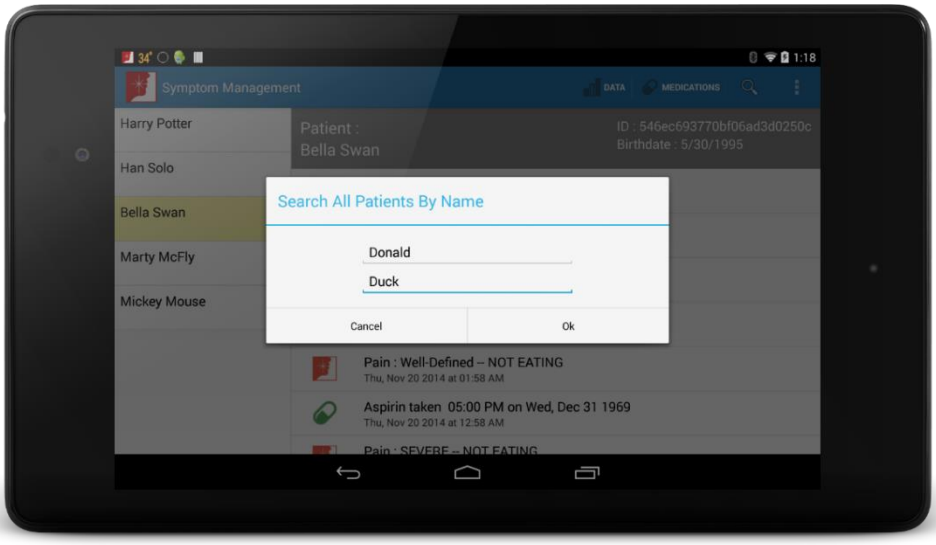


Figure 44. Search dialog used by the doctor to find a patient on the server.

Figure 42. Results of a patient search. Note that the patient is TEMPORARILY added to the doctor's patient list. The patient is NOT reassigned to the doctor.

Doctor Manages Medications

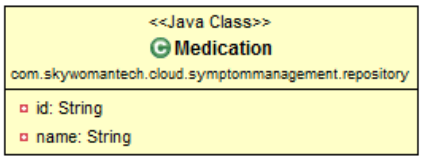


Figure 45. Medication class.

12. A doctor can update a list of pain medications associated with a Patient. This data updates the tailored questions regarding pain medications listed above in (6).

The medication class seen in this figure was created to handle this requirement. The server has a repository to store all of the possible medications and provides endpoints for managing these medication.

Figure 46. Icon on Doctor Dashboard to Add Prescription/Medication

Additionally the Patient class has a set of Medication objects that are assigned by the physician. These also called prescriptions. The doctor can add or delete medications/prescriptions from the patient.

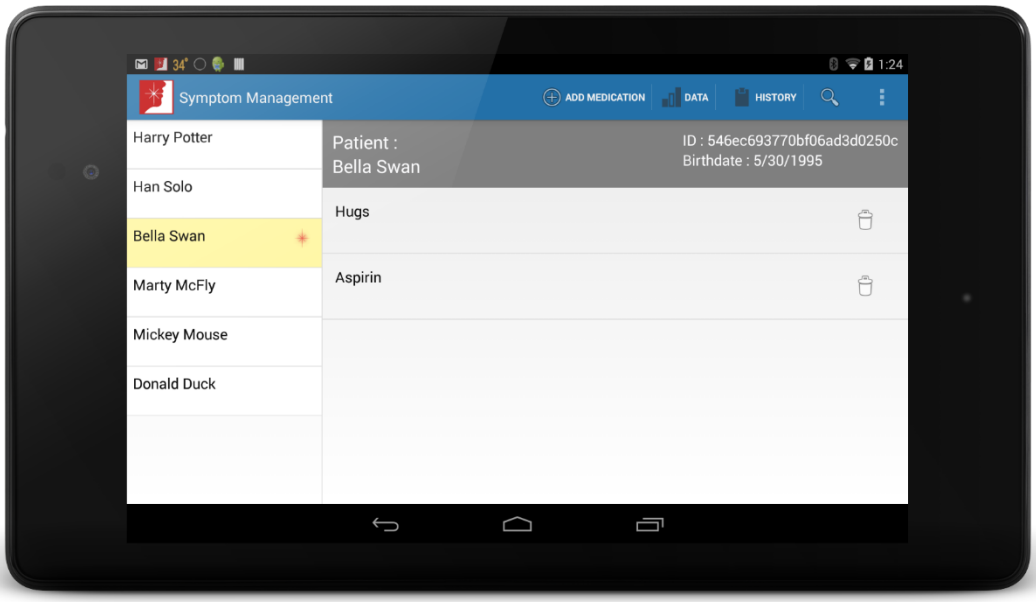


Figure 47. Patient Prescription List on the Doctor Dashboard.

Figure 48. Icon to add new medication to the database.

If the doctor does not find a medication to add to the patient's prescriptions then they can add a new medication using an options menu selection. After choosing the add medication option menu item, a list of medication assigned to the patient will be displayed. This custom list view will have a button that allows the doctor to remove a medication from the list. An Add option menu item will allow the doctor to add a new medication to the patient's list. If the Add option is selected then a list of all available medications is displayed. The doctor can select one and it will return to the patient's medication list with that item added to it. If the medication is not in the list the doctor can choose the Add option item from the list of all medications and an Add entry screen will be displayed. This will allow the doctor to add a new medication to the database of all medications. The patient's app will periodically check for updates to their medication list using the SyncAdapter processing.

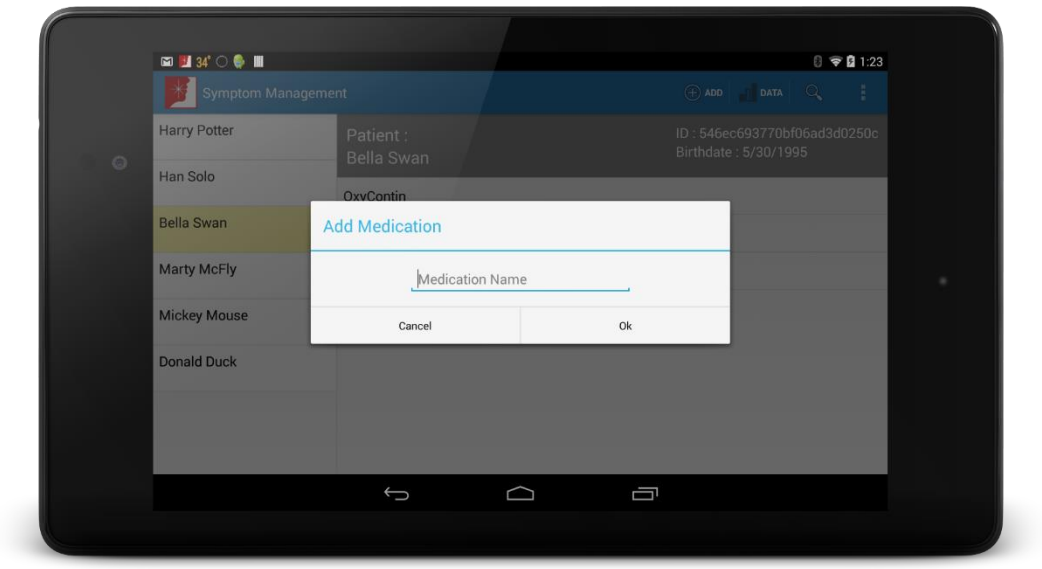


Figure 49. Doctors can new medications to the database if one does not exist. Then this new medication can be added as a prescription for the patient.

Doctor Receives Notifications Concerning Patients

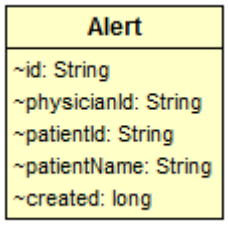


Figure 51. Alert Class

13. A doctor is alerted if a patient experiences 12 hours of "severe pain," 16 or more hours of "moderate" or "severe pain" or 12 hours of "I can't eat."

When a patient performs a pain log check-in and saves it to the server, the server then checks the patient's history to determine if an alert needs to be created for the patient. An alert is created for every doctor assigned to the patient. The doctor's app uses the SyncAdapter to periodically check the server for Alerts that match the physician's id. If any are found an Android Notification is created on the doctor's device. If there is more than one Alert all of them will be added to the Android's Notification manager. If the doctor selects the notification message it go immediately to the doctor dashboard or login screen. The doctor's app will check whenever it updates patient data or updates it patient list for alerts.

Figure 50. Doctor alerted that patient has severe symptoms.

Figure 53. Doctor is notified of the severe patient via Notifications.

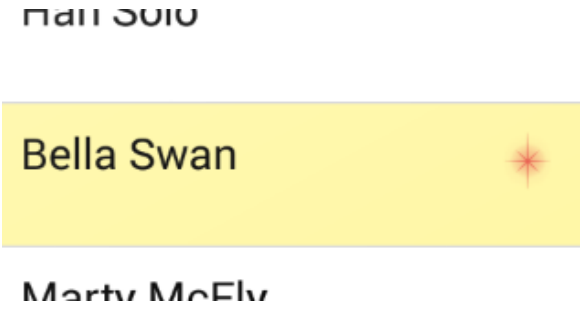


Figure 52. Doctor's notified of severe patients in their assigned patient list.

Doctors are also notified of a severe patient when it is highlighted an a special "severe" icon is displayed next to the patient's name on their list.

14. A patient's data should only be accessed by his/her doctor over HTTPS.

This app implements OAuth2 on both the server and all app versions. Below is a list of available endpoints and users that can access them. This is handled with authorization and the user type. This list includes a user type of "admin" which is not currently described in this design but will be a user of this system if fully implemented.

Path	Available Commands	Authorized Roles
/patient	GET POST	Admin, Physician Admin
/patient/{id}	GET PUT DELETE	Patient, Admin, Physician Patient, Admin, Physician Admin
/patient/find?name="first last"	GET	Admin, Physician
/physician	GET POST	Admin, Physician Admin
/physician/{id}	GET PUT DELETE	Admin, Physician Admin, Physician Admin
/physician/{id}/alert	GET	Admin, Physician
/medication	GET POST	Patient, Admin, Physician Admin, Physician
/medication/{id}	GET PUT DELETE	Patient, Admin, Physician Admin, Physician Admin
/medication/find?name="name"	GET	Patient, Admin, Physician
/alert	GET POST	Admin Admin, Physician
/alert/{id}	DELETE	Admin, Physician
/credential/find?username="username"	GET	No Roles (Pre-Login usage)

The Administrator Workflow

Administrator users can login in to the app and they are taken to a special screen flow. This screen flow allows them to add new patients or edit existing ones. They are also able to add new doctors or edit the existing ones. And finally they are able to associate a doctor with a patient.

Administrator Sign In

The administrator login is the same as the patient and the doctor. The administrator has the only hard-coded username and password. There is only one administrative user.

Username :	admin
Password :	pass

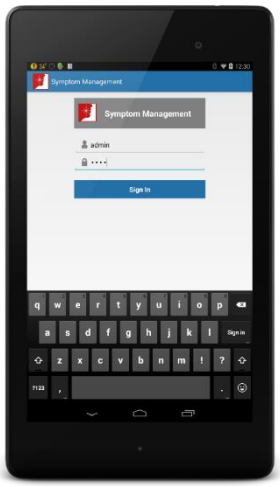


Figure 54. Same login screen for the administrator. The administrator app works well on smaller devices.

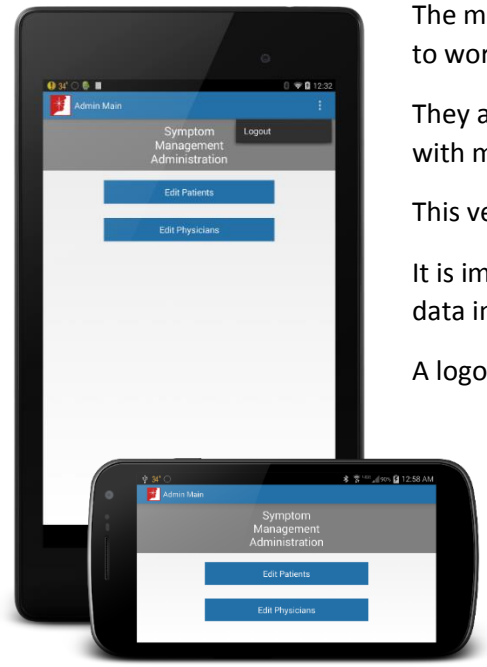


Figure 56. Main administrator screen.

The main screen for the administrator allows them to work with either patients or physicians.

They are also able to logout when they are done with managing the users of this app.

This version of the app works well on any size of device.

It is important to note that the administrator stores no data locally on the device. All data interactions are done over the internet.

A logout menu option is available to ensure security.

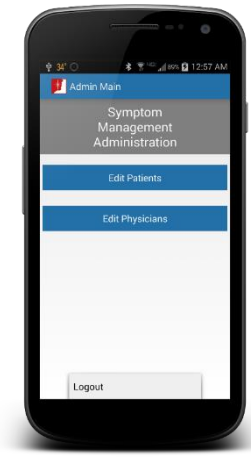


Figure 55. Admin users can logout for security.

Administrator Manages Patients

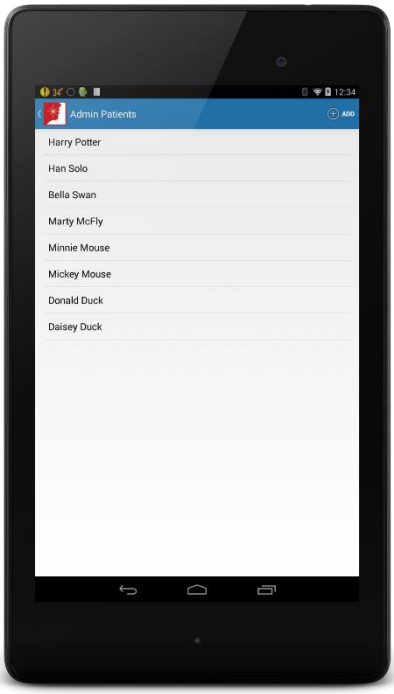


Figure 57. After choosing to edit patients the administrator is able to see a list of all the patients in the server database.

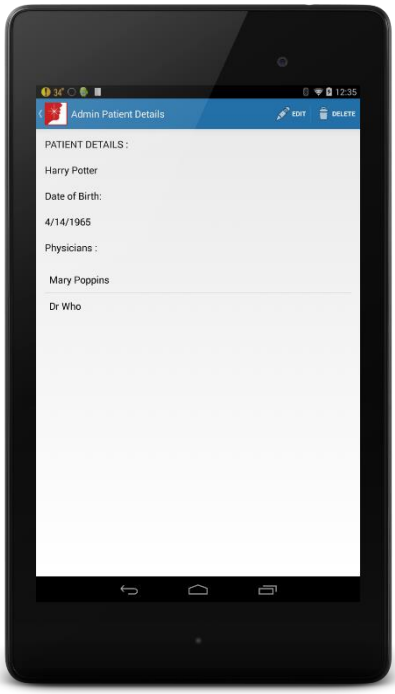


Figure 59. If the administrator clicks on a patient they are show the patient details screen above. Patients can have more than one doctor assigned to them. They can be edited or deleted from menu items on this screen.

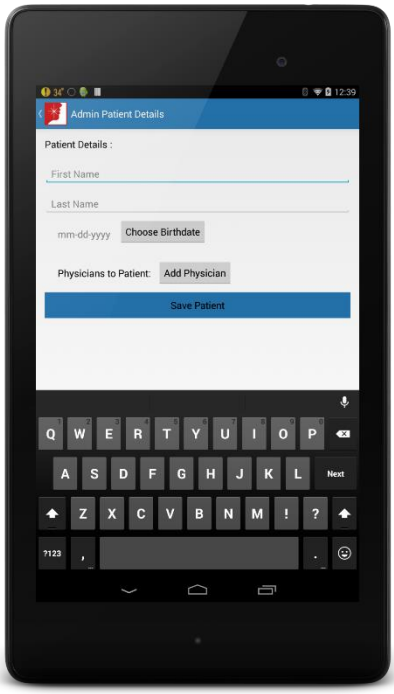


Figure 58. If the administrator chooses to add a new patient from the patient list, then they are show a new patient entry screen.

Administrator Manages Physicians

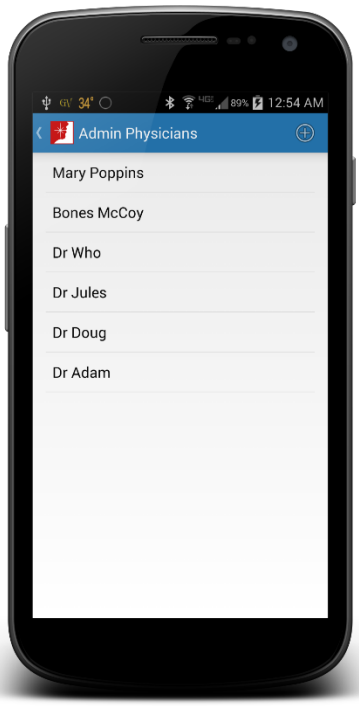


Figure 60. If the administrator chooses to work with Physicians/Doctors then they are shown a list of doctors in the database.

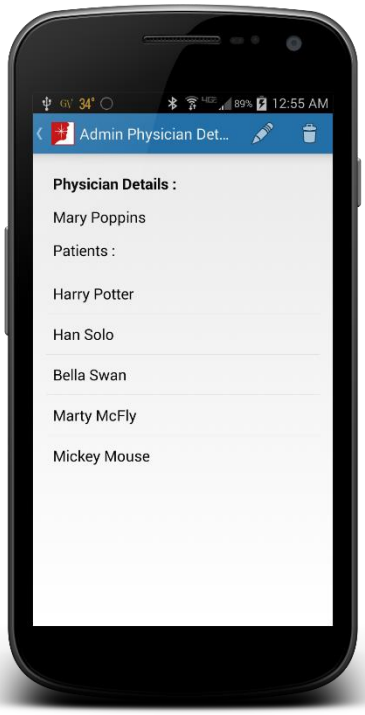


Figure 62. If the admin selects a doctor then they are shown the doctor's details. They can edit or delete from this screen.

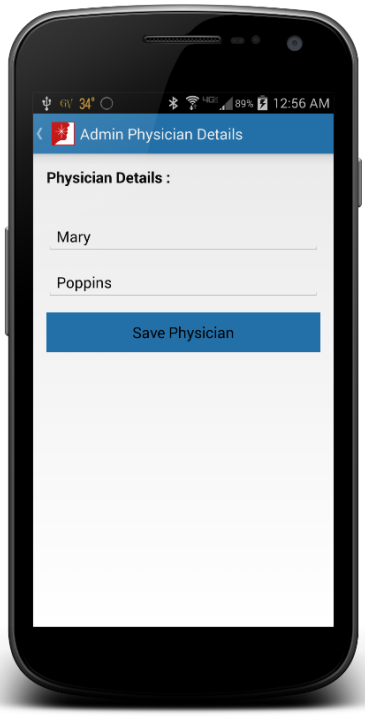


Figure 61. If the admin chooses the add option menu item then they can add a new doctor to the database.

Administrator Assigns Physicians to Patients

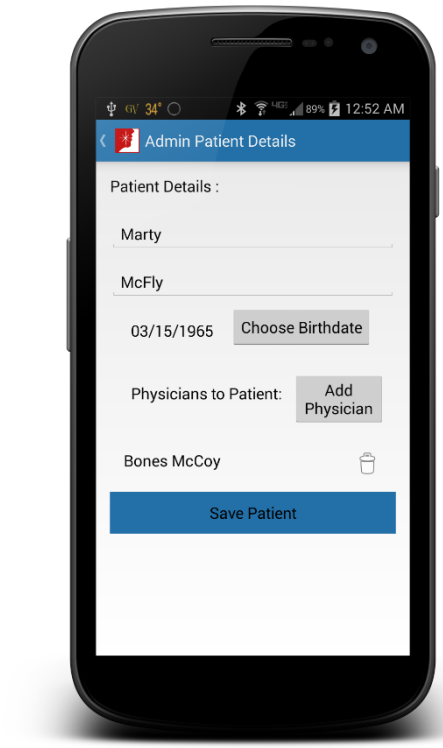
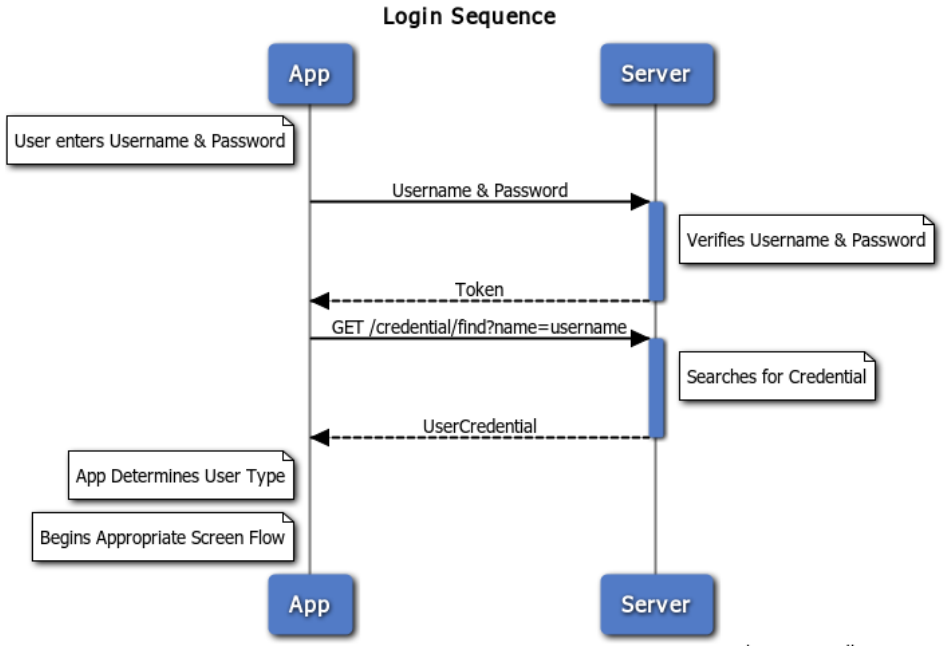


Figure 63. Administrators can add physicians to the patient. An unlimited number of doctors can be assigned to each patient. The patient must be saved to the database before adding any physicians! The server will automatically add the patient to the doctor's patient list so there is no app screens to do this.

The Login Process

This section describes the login authentication process between the app and the server. This design is adequate for the Capstone project only and would not be recommended for a production environment. OAuth2 security is configured on the Spring Server and app. The app is using EasyHTTPClient and ignores certificates. The unsecure keystore used in course examples was utilized in this implementation.



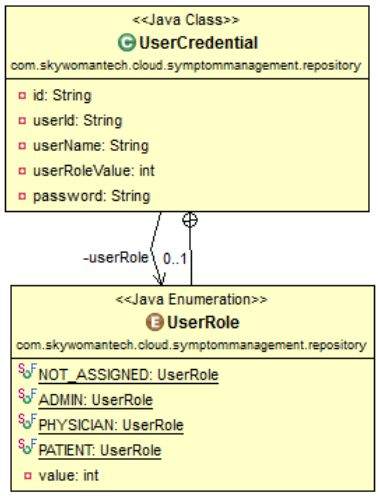


Figure 64. User Credential Object as given to the app from the server.

The UserCredential is generated when the administrator adds a new patient to the repository. The administrator is the only user with the permission to add new patients. The usernames are created with a combination of the first and last names. The password generated is hard-coded to "pass." There is

Username: firstname.lastname
Password: pass

The only accepted API call to the UserCredentialRepository is the findByUserName method.

The Symptom Management Check-In Process

"A *Reminder* triggers a *Check-In*, which is defined by the app as a unit of data associated with a *Patient*, a date, a time, and that patient's responses to various questions (items 4-8) at that date and time." – Functional

In order to meet the above requirements and to give the app the flexibility of a pain tracking and medication tracking app, the check-in process design required the check-in data as defined above to be divided into smaller pieces and associated together with a check-in id.

The design views the check-in data as really four categories of data including: "check-in data", "pain severity level", "eating ability", and "medication tracking."

Check In Data / Check In Identifier	This is considered an event that associates specific group of pain severity, eating ability and medication tracking data using a check in identifier.
Pain Severity Level	Is the patient response of "well controlled," "moderate," or "severe" when answering the question: "How bad is your mouth pain/sore throat?"
Eating Ability	Is the patient response of "no," "some," or "I can't eat" when answering the question: "Does your pain stop you from eating/drinking?"
Medication Tracking	Is the patient response including the medication name and the time and date that he or she took the specified medicine when answering the question "Did you take your [medication name]?"

To manage the above information, this design uses what it calls "log" objects. You can view the related classes in the Class Diagram. In the class diagram you can also see that the Patient Class holds Collections of the check-in, pain and medication logs and this is how this design associates the Patient to the check-in data. All data with the same check in id are associated with the same check-in event.

Check In Log	Check In ID Timestamp
Pain Log	Pain Severity Level Eating Ability Check In ID Timestamp
Medication Log	Medication Taken Time Taken Check In ID Timestamp
History Log	Generic list of all pain, medication and check in logs combined.

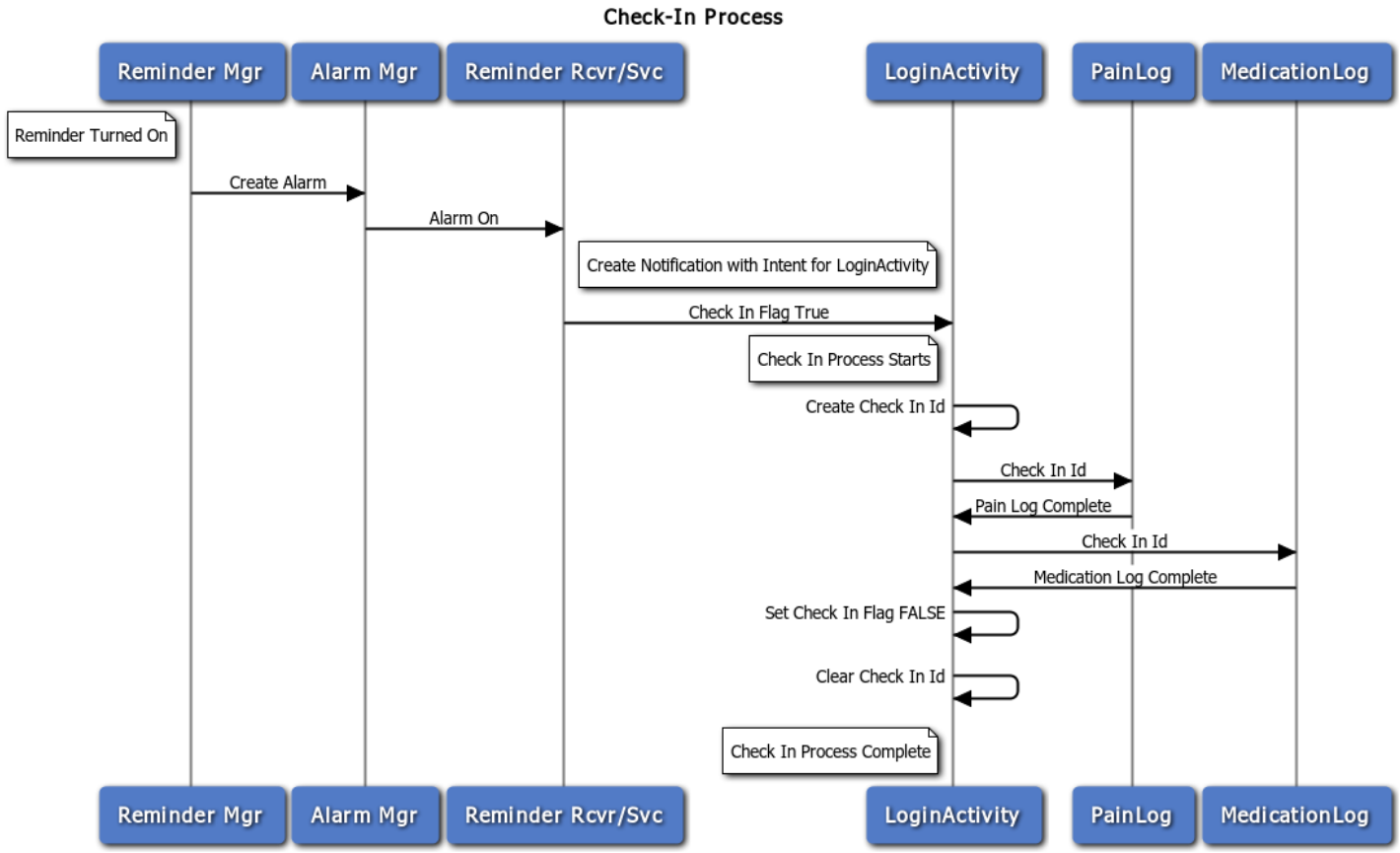


Figure 65. This figure describes how this app processes the check-in event from Reminder to clearing the check in id after the medication logs are entered.

1. The Reminder is switched on by the user.
2. The Reminder Manager creates an alarm to trigger at the specified time of day & sends it to the Alarm Manager.
3. The Alarm Manager triggers the alarm and the Reminder Receiver is notified.
4. The Reminder Receiver triggers the Reminder Service to tell Login Activity to set Check In to TRUE. It also creates a notification with a Pending Intent to call the Login Activity.
5. The Login Activity sets the check in flag to TRUE. Then it creates a unique check in id. It then begins the check in screen flow by making the Pain Log Fragment active.
6. The Pain Log Fragment requests the answers to pain severity and eating ability and saves a pain log with the check in id attached.
7. The Pain Log Fragment tells the Login Activity that is completed.
8. The Login Activity then Creates a Check In Log and activates the Medication Log Fragment with check in flag set to TRUE and current check in id.
9. The Medication Log request time taken for prescriptions. During onPause the Medication Log tells Login Activity to set check in flag to FALSE.
10. Login Activity sets flag to FALSE and clears the check in id. This ensures that no logs will be associated with any check in until the next reminder is triggered.

SyncAdapter Processing

When there is internet access the SyncAdapter will sync the Patient's local SQLite data with REST server database (which uses Mongo DB). The SyncAdapter also updates patient data from the physician such as the prescription lists. The SyncAdapter runs about every 20 minutes but is also called to immediately push data to the cloud whenever the patient enters check-in or status logs. The SyncAdapter will keep trying on a periodic basis to sync data with the cloud server even when internet is not available.

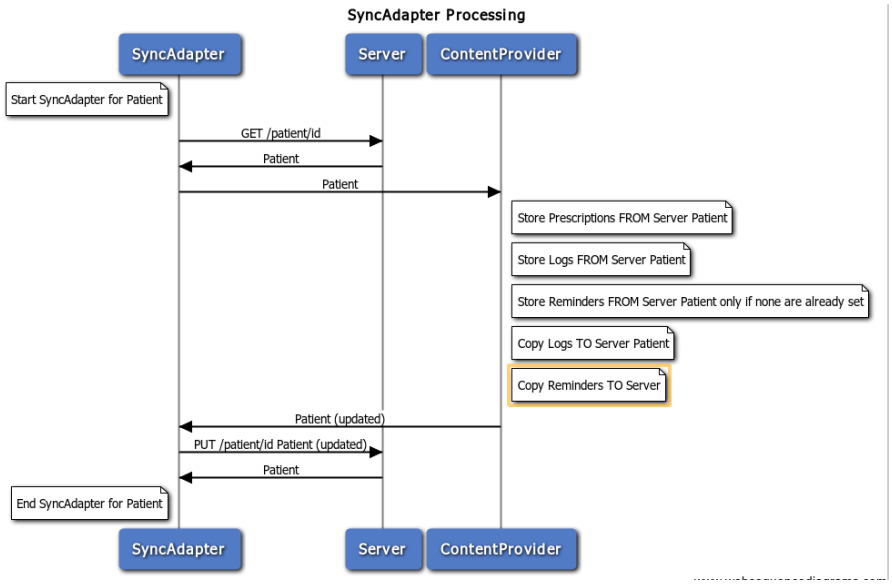


Figure 66. SyncAdapter Patient Processing

If a physician is logged in, then the SyncAdapter works differently. Instead it will periodically check for alerts concerning severe patients. Since no data is stored on the device for the physician app, the doctor will use data that is recently updated.

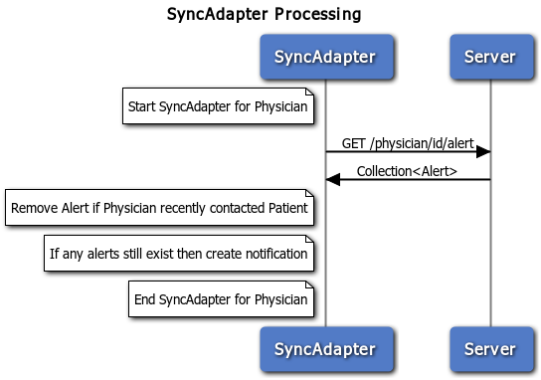
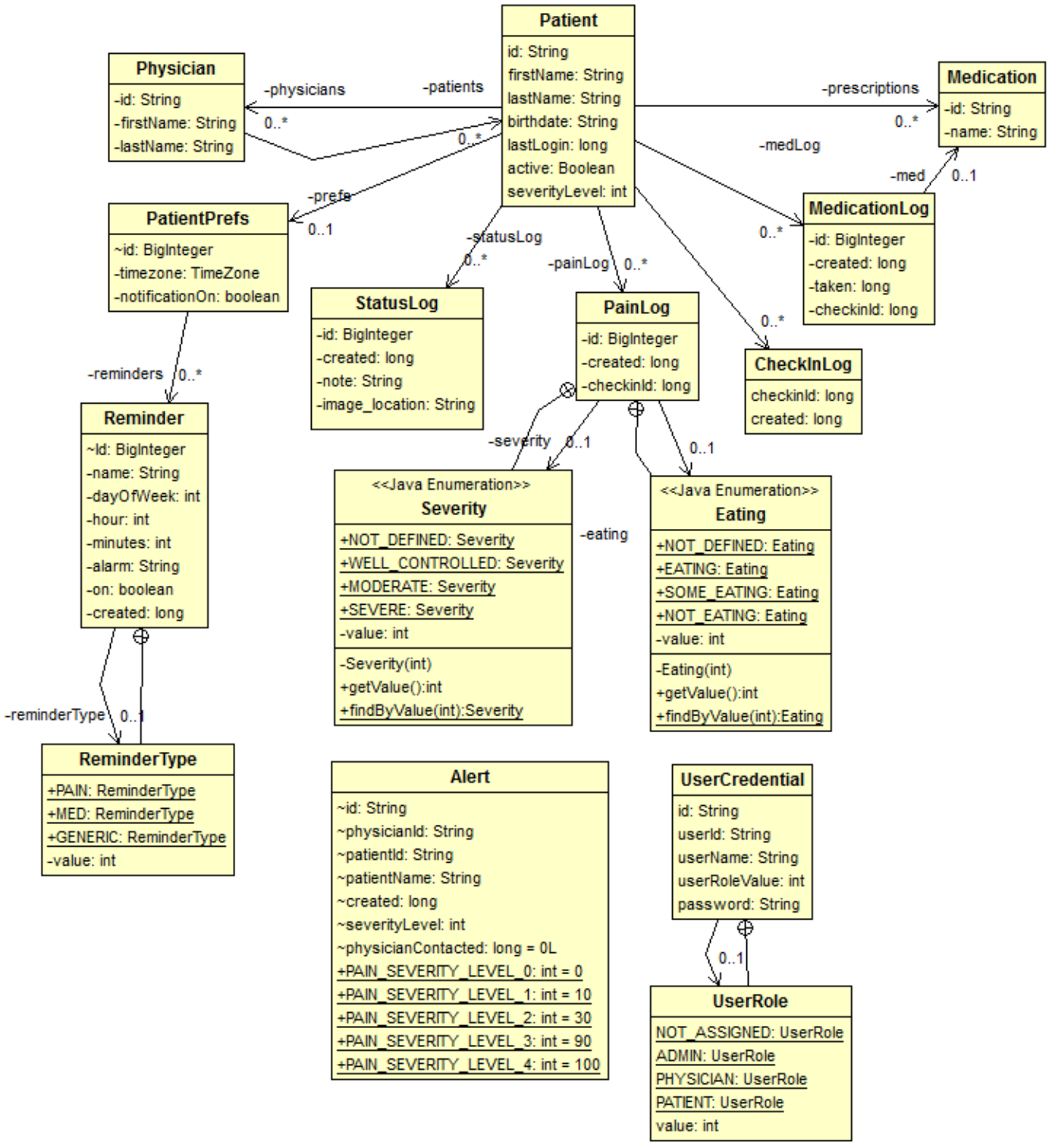


Figure 67. SyncAdapter Physician Processing.

Symptom Management Class Diagram



Symptom Management Class Hierachy

[Generated Using Java Doc from Android Studio]

Highlights show Activity, BroadcastReceiver, ContentProvider and Service Classes for Requirements

Class Hierarchy

java.lang.Object
<any>
com.skywomantech.app.symptommanagement.client.CallableTask<T>
com.skywomantech.app.symptommanagement.physician.HistoryLogAdapter
com.skywomantech.app.symptommanagement.LoginActivity.UserLoginTask
com.skywomantech.app.symptommanagement.patient.MedicationLogListAdapter
com.skywomantech.app.symptommanagement.physician.PatientListAdapter
com.skywomantech.app.symptommanagement.admin.Patient.PhysicianEditListAdapter
com.skywomantech.app.symptommanagement.physician.PrescriptionAdapter
com.skywomantech.app.symptommanagement.patient.ReminderListAdapter
AbstractAccountAuthenticator
com.skywomantech.app.symptommanagement.sync.SymptomManagementAuthenticator
AbstractThreadedSyncAdapter
com.skywomantech.app.symptommanagement.sync.SymptomManagementSyncAdapter
Activity
com.skywomantech.app.symptommanagement.admin.Patient.PatientPhysicianListActivity
(implements com.skywomantech.app.symptommanagement.admin.Physician.PhysicianListFragment.Callbacks)
Activity
com.skywomantech.app.symptommanagement.admin.Patient.AdminPatientListActivity (implements
com.skywomantech.app.symptommanagement.admin.Patient.PatientListFragment.Callbacks)
Activity
com.skywomantech.app.symptommanagement.admin.Patient.AdminPatientDetailActivity
(implements com.skywomantech.app.symptommanagement.admin.Patient.BirthdateDialog.Callbacks,
com.skywomantech.app.symptommanagement.admin.Patient.PatientDetailFragment.Callbacks)
Activity
com.skywomantech.app.symptommanagement.admin.Physician.AdminPhysicianListActivity
(implements com.skywomantech.app.symptommanagement.admin.Physician.AdminPhysicianDetailFragment.Callbacks,
com.skywomantech.app.symptommanagement.admin.Physician.PhysicianListFragment.Callbacks)
Activity
com.skywomantech.app.symptommanagement.admin.Physician.AdminPhysicianDetailActivity
(implements com.skywomantech.app.symptommanagement.admin.Physician.AdminPhysicianDetailFragment.Callbacks)
Activity
com.skywomantech.app.symptommanagement.admin.Medication.AdminMedicationListActivity
(implements com.skywomantech.app.symptommanagement.physician.MedicationAddEditDialog.Callbacks,
com.skywomantech.app.symptommanagement.physician.MedicationListFragment.Callbacks)
Activity
com.skywomantech.app.symptommanagement.admin.AdminMain
Activity
com.skywomantech.app.symptommanagement.physician.PhysicianActivity (implements
com.skywomantech.app.symptommanagement.physician.HistoryLogFragment.Callbacks,
com.skywomantech.app.symptommanagement.physician.MedicationAddEditDialog.Callbacks,
com.skywomantech.app.symptommanagement.physician.MedicationListFragment.Callbacks,
com.skywomantech.app.symptommanagement.physician.MedicationManager.Callbacks,
com.skywomantech.app.symptommanagement.physician.PatientGraphicsFragment.Callbacks,
com.skywomantech.app.symptommanagement.physician.PatientManager.Callbacks,
com.skywomantech.app.symptommanagement.physician.PatientMedicationFragment.Callbacks,
com.skywomantech.app.symptommanagement.physician.PatientSearchDialog.Callbacks,
com.skywomantech.app.symptommanagement.physician.PhysicianListPatientsFragment.Callbacks,

com.skywomantech.app.symptommanagement.physician.PhysicianManager.Callbacks,
com.skywomantech.app.symptommanagement.physician.PhysicianPatientDetailFragment.Callbacks,
com.skywomantech.app.symptommanagement.physician.PrescriptionAdapter.Callbacks)
com.skywomantech.app.symptommanagement.physician.PhysicianListPatientsActivity
com.skywomantech.app.symptommanagement.physician.PhysicianPatientDetailActivity
Activity
com.skywomantech.app.symptommanagement.LoginActivity
Activity
com.skywomantech.app.symptommanagement.patient.PatientMainActivity (implements
com.skywomantech.app.symptommanagement.physician.HistoryLogFragment.Callbacks,
com.skywomantech.app.symptommanagement.patient.MedicationLogListAdapter.Callbacks,
com.skywomantech.app.symptommanagement.patient.MedicationTimeDialog.Callbacks,
com.skywomantech.app.symptommanagement.patient.PatientMainFragment.Callbacks,
com.skywomantech.app.symptommanagement.patient.PatientMedicationLogFragment.Callbacks,
com.skywomantech.app.symptommanagement.patient.PatientPainLogFragment.Callbacks,
com.skywomantech.app.symptommanagement.patient.PatientStatusLogFragment.Callbacks,
com.skywomantech.app.symptommanagement.patient.ReminderAddEditDialog.Callbacks,
com.skywomantech.app.symptommanagement.patient.ReminderFragment.Callbacks,
com.skywomantech.app.symptommanagement.patient.ReminderListAdapter.Callbacks)
com.skywomantech.app.symptommanagement.data.Alert
BroadcastReceiver
com.skywomantech.app.symptommanagement.patient.Reminder.ReminderReceiver
com.skywomantech.app.symptommanagement.BuildConfig
ContentProvider
com.skywomantech.app.symptommanagement.data.PatientContentProvider
DefaultHttpClient
com.skywomantech.app.symptommanagement.client.oauth.unsafe.EasyHttpClient
DialogFragment
com.skywomantech.app.symptommanagement.admin.Patient.BirthdateDialog
DialogFragment
com.skywomantech.app.symptommanagement.physician.PatientSearchDialog
DialogFragment
com.skywomantech.app.symptommanagement.physician.MedicationAddEditDialog
DialogFragment
com.skywomantech.app.symptommanagement.patient.ReminderAddEditDialog
DialogFragment
com.skywomantech.app.symptommanagement.patient.MedicationTimeDialog
Fragment
com.skywomantech.app.symptommanagement.admin.Patient.PatientDetailFragment
Fragment
com.skywomantech.app.symptommanagement.admin.Patient.PatientAddEditFragment
Fragment
com.skywomantech.app.symptommanagement.admin.Physician.AdminPhysicianDetailFragment
Fragment
com.skywomantech.app.symptommanagement.admin.Physician.AdminPhysicianAddEditFragment
Fragment
com.skywomantech.app.symptommanagement.admin.AdminMainFragment
Fragment
com.skywomantech.app.symptommanagement.physician.PhysicianPatientDetailFragment
Fragment
com.skywomantech.app.symptommanagement.physician.PatientGraphicsFragment
Fragment
com.skywomantech.app.symptommanagement.patient.ReminderFragment
Fragment
com.skywomantech.app.symptommanagement.patient.PatientStatusLogFragment
Fragment
com.skywomantech.app.symptommanagement.patient.PatientPainLogFragment

Fragment
com.skywomantech.app.symptommanagement.patient.PatientMedicationLogFragment
Fragment
com.skywomantech.app.symptommanagement.patient.PatientMainFragment
com.skywomantech.app.symptommanagement.data.HistoryLog
com.skywomantech.app.symptommanagement.physician.HistoryLogAdapter.ViewHolder
ListFragment
com.skywomantech.app.symptommanagement.admin.Patient.PatientListFragment
ListFragment
com.skywomantech.app.symptommanagement.admin.Physician.PhysicianListFragment
ListFragment
com.skywomantech.app.symptommanagement.physician.PhysicianListPatientsFragment
ListFragment
com.skywomantech.app.symptommanagement.physician.PatientMedicationFragment
ListFragment
com.skywomantech.app.symptommanagement.physician.MedicationListFragment
ListFragment
com.skywomantech.app.symptommanagement.physician.HistoryLogFragment
com.skywomantech.app.symptommanagement.LoginUtility
com.skywomantech.app.symptommanagement.data.Medication
com.skywomantech.app.symptommanagement.data.MedicationLog
com.skywomantech.app.symptommanagement.patient.MedicationLogListAdapter.ViewHolder
com.skywomantech.app.symptommanagement.physician.MedicationManager
com.skywomantech.app.symptommanagement.data.PainLog
com.skywomantech.app.symptommanagement.data.Patient
com.skywomantech.app.symptommanagement.data.PatientCPContract
com.skywomantech.app.symptommanagement.data.PatientCPContract.CredentialEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.MedLogEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.PainLogEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.PatientEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.PhysicianEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.PrefsEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.PrescriptionEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.ReminderEntry
com.skywomantech.app.symptommanagement.data.PatientCPContract.StatusLogEntry
com.skywomantech.app.symptommanagement.data.PatientCPcvHelper
com.skywomantech.app.symptommanagement.data.PatientDataManager
com.skywomantech.app.symptommanagement.physician.PatientListAdapter.ViewHolder
com.skywomantech.app.symptommanagement.physician.PatientManager
com.skywomantech.app.symptommanagement.physician.PatientManager.HistoryLogSorter (implements
java.util.Comparator<T>)
com.skywomantech.app.symptommanagement.data.PatientPrefs
com.skywomantech.app.symptommanagement.data.Physician
com.skywomantech.app.symptommanagement.admin.Patient.PhysicianEditListAdapter.ViewHolder
com.skywomantech.app.symptommanagement.physician.PhysicianManager
com.skywomantech.app.symptommanagement.physician.PrescriptionAdapter.ViewHolder
com.skywomantech.app.symptommanagement.R
com.skywomantech.app.symptommanagement.R.attr
com.skywomantech.app.symptommanagement.R.color
com.skywomantech.app.symptommanagement.R.dimen
com.skywomantech.app.symptommanagement.R.drawable
com.skywomantech.app.symptommanagement.R.id
com.skywomantech.app.symptommanagement.R.layout
com.skywomantech.app.symptommanagement.R.menu
com.skywomantech.app.symptommanagement.R.string
com.skywomantech.app.symptommanagement.R.style
com.skywomantech.app.symptommanagement.R.xml

com.skywomantech.app.symptommanagement.data.Reminder
com.skywomantech.app.symptommanagement.patient.ReminderListAdapter.ReminderHolder
com.skywomantech.app.symptommanagement.patient.ReminderListAdapter.ViewHolder
com.skywomantech.app.symptommanagement.patient.Reminder.ReminderManager
com.skywomantech.app.symptommanagement.patient.Reminder.ReminderManager.IntentSaver
com.skywomantech.app.symptommanagement.patient.Reminder.ReminderManager.ReminderSorter
(implements java.util.Comparator<T>)
retrofit.RestAdapter.Builder
com.skywomantech.app.symptommanagement.client.oauth.SecuredRestBuilder
Service
com.skywomantech.app.symptommanagement.sync.SymptomManagementSyncService
Service
com.skywomantech.app.symptommanagement.sync.SymptomManagementAuthenticatorService
Service
com.skywomantech.app.symptommanagement.patient.Reminder.ReminderService
SQLiteOpenHelper
com.skywomantech.app.symptommanagement.data.PatientDBHelper
com.skywomantech.app.symptommanagement.data.StatusLog
com.skywomantech.app.symptommanagement.client.SymptomManagementService
java.lang.Throwable (implements java.io.Serializable)
java.lang.Exception
java.lang.RuntimeException
com.skywomantech.app.symptommanagement.client.oauth.SecuredRestException
com.skywomantech.app.symptommanagement.data.graphics.TimePoint
com.skywomantech.app.symptommanagement.data.graphics.EatingPlotPoint
com.skywomantech.app.symptommanagement.data.graphics.MedicationPlotPoint
com.skywomantech.app.symptommanagement.data.graphics.SeverityPlotPoint
com.skywomantech.app.symptommanagement.data.UserCredential