*Florida International University*

*School of Computing and Information Sciences*

CIS 4911 - Senior Capstone Project

Software Engineering Focus

Final Deliverable

Medzmate 1.0

**Team Members**

Juan C. Correa

Pedro Montero

**Product Owner**: Prof. Peter Dickson

**Instructor**: Prof. Masoud Sadjadi

**Trademark**

The name Medzmate has been granted a Trademark by the U.S. Trademark Office to Peter Dickson.

**Use of existing patents**

The proposed Medzmate software does not use any existing software modules or code patented or not. It is all original intellectual property and code developed by Juan Correa and Pedro Montero.

The Medzmate hardware does not use any existing patented technology and none of the prior art that surfaced in searching the patent office database using the term “pill dispensing” uses dispensing straws and a dispensing mechanism that releases pills from the straw.

A patent for the hardware’s dispensing mechanism is in the process of being applied for. According to the terms and conditions for granting a patent there must not be public disclosure of the details of the technology and this restricts the copying and distribution of the documentation on this project including its operating software and IP.

***Abstract***

*Medzmate is a revolutionary product intended to help elderly patients to receive their daily medications in an easy and timely way. It is a solution that requires a software suite to support all the features planned for its 1.0 version. Using standard software engineering techniques and following the SCRUM framework we were able to deliver a comprehensive project documentation that contains system information, user stories, architectural patterns, design patterns and UML diagrams. This document is also a reflection of the entire development process followed by our team to be in compliance with software engineering principles-- such as separation of concerns, modularity, and abstraction, anticipation of change, generality and incremental development.*

**Table of Contents**

**Introduction**

Current System

Purpose of New System

**User Stories**

Implemented User Stories

Pending User Stories

**Project Plan**

Hardware and Software Resources

Sprints Plan

*Sprint 1*

*Sprint 2*

*Sprint 3*

*Sprint 4*

*Sprint 5*

*Sprint 6*

*Sprint 7*

**System Design**

Architectural Patterns

System and Subsystem Decomposition

Deployment Diagram

Design Patterns

**System Validation**

**Glossary**

**Appendix**

Appendix A - UML Diagrams

*Static UML Diagrams*

*Dynamic UML Diagrams*

Appendix B - User Interface Design

Appendix C - Sprint Review Reports

Appendix D - Sprint Retrospective Reports

**References**

# Introduction

The medical and pharmaceutical industries face multiple issues at the time of dispensing medicines to elderly. These issues arise on several stages of the process--during the dispensing in the pharmacy, making the home delivery and finally ensuring the patient takes the right medication at the right time. Health care providers spend millionaire sums to guarantee accuracy on every stage of this process and it is still highly susceptible to errors.

Medzmate is a revolutionary product intended to help elderly patients to receive their daily medications in an easy and timely way. Medzmate is not like any other product in the market; it provides live alert to the patient and relatives when medications are dispensed, holds a supply of medications that can last for months without refill, and eases the refill process enormously. Similar products in the market fall short in providing the necessary accuracy and constant medication supply without continuous refills. The typical medications dispenser needs to be refilled weekly meanwhile Medzmate can last up to several months.

## Current System

The current pill dispensing systems are presented on the following images. The target customers of the Medzmate largely use a pill box.



Figure 1: Pills box currently used by prospective customers.

From which daily doses (morning and night) are sorted using compartments like the ones on Figure 2.



Figure 2: Common medication compartments

Some more sophisticated devices have been developed but have not sold well to date. They fail to provide a general solution for the dispensing of medications or are too expensive.

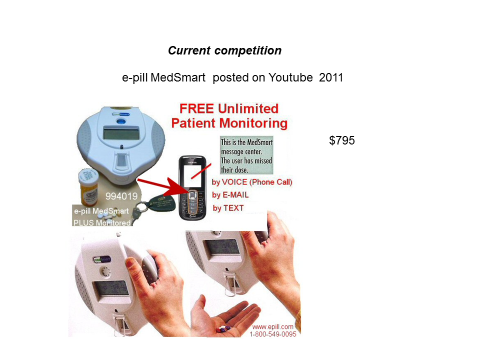


Figure 3: e-pill MedSmart

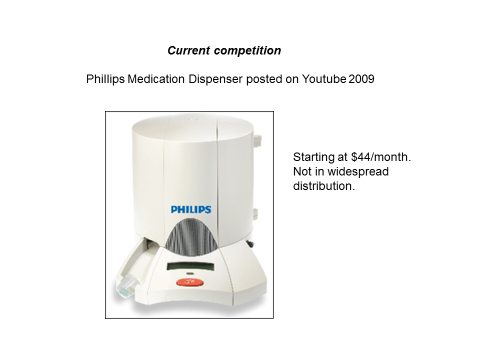


Figure 4: Philips Medication Dispenser.

The target customer segment (audience) are the 20 million Americans who take 4 or more medicines a day particularly the 10 million over the age of 65 who take 4 or more medicines a day (most commonly for controlling cholesterol build up, blood pressure and diabetes).

*Annals of Internal Medicine*. 2012; 157, 785-795.

The estimated cost of non-adherence to prescribed medication schedule is between $100 billion and $250 billion a year and current interventions, technologies and programs are not very effective at all. *Annals of Internal Medicine*. 2012; 157, 785-795.

The current systems are not under the control and supervision of a pharmacist who loads the dispensing device according to doctor prescriptions *and* who can check on actual dispensing vs scheduled dispensing and then communicate with the patient via the device/technology.

## Purpose of New System

The main purpose of Medzmate is to increase the health and wellness of those who have to take many drugs a day, and need help in taking them safely, according to the prescription provided by the doctor. At the same time provide reassurance and encouragement to the patient, and reduce the burden on children and love ones.

This product will significantly reduce healthcare cost associated with medication adherence failure. Medzmate will alert the patient when is time to take the medicine. It will be able to carry enough pills during a month for more than 10 different medications. Medzmate will also eliminate the need of a home visit to millions of elderly to check whether they have taken the medication or not by sending notifications to the care provider as well as a designated contact.

The software supporting Medzmate consists of two main components--a mobile application and software module running within the Medzmate station. The mobile application will be use by a pharmacist to configure the schedule, medicine information, patient information and notifications. The Medzmate station software will be monitoring when the patient needs to be notified to take the medication and therefore trigger the dispensing of the pills. The station will also send notifications for different types of events such as, when the medication was taken or not.

# User Stories

This section contains a list of all the user stories gathered from the product owner. These stories represent the scenarios, features, and functionality of the Medzmate system. Due to time constraints some of these user stories may not be implemented this semester. In order to differentiate them we created two subsections “Implemented User Stories” and “Pending User Stories”.

## Implemented User Stories

**001 Add new medicine information (Required)**

**As a**

Pharmacist

**I want to**

click one of buttons which correspond to the prescription container within the

Medzmate loading dock.

**So that**

a prescription entry form opens for this location.

**Description**

After login, the pharmacist should be able to see a screen that mimics the loading dock of Medzmate

**002 Add new medicine detailed information (Required)**

**As a**

Pharmacist

**I want to**

add the following information:

1. Medicine name

2. Doctor's name

3. Pills quantity

4. Expiration Date (MM/DD/YY)

5. Mass

6. Dosage frequency (per day)

7. Dosage times

8. Link with order container (type A,B,C)

9. Symptoms

10. Side Effects

**So that**

I can save it to the Medzmate app

**Description**

The pharmacist can select a specific slot in the loading dock and a form will pop up allowing him/her to enter detailed information about the medicine for that particular slot.

**003 Station: Create Medzmate Emulator (Required)**

**As a**

Developer

**I want to**

be able to work with a Medzmate Station that provides basic functionality and printed info in a console.

**So that**

communication with the Medzmate mobile application can be tested.

**Acceptance Criteria:**

* Emulator should print comprehensive messages based on the actions triggered by the mobile application.
* Emulator needs to be able load the configuration stored in the Medzmate that was previously sent by the mobile application.
* Emulator should be able to throw alerts based on the medication’s schedule.

**004 App-Station Pharmacist Login (Required)**

**As a**

Pharmacist

**I want to**

Login in to the Medzmate mobile app

**So that**

I can get access to the advance configuration screen

**Description**

Only the pharmacist or dedicated health care employee should be able to add/remove medicine information, and/or modify the schedule.

**005 Connect to Medzmate Station Using its IP Address (Required)**

**As a**

Pharmacist, Patient or Relative

**I want to**

be able to type a unique identifier (IP address for 1.0 version) and click connect.

**So that**

I can connect to the correct Medzmate.

**Description**

In order to keep the app generic so that one mobile device can connect to any Medzmate, the initial screen will allow the user to enter a unique id so that it can connect to the Medzmate and then login.

**Acceptance Criteria:**

* Page accepts a Medzmate IP address.
* When user clicks connect a request is sent to the station.
* Upon confirmation the login page is loaded.

**006 User should confirm the entered medicine detailed information (Required)**

**As a**

Pharmacist

**I want to**

See a confirmation page with the data entered in the detailed medicine form.

**So that**

I can double check the information before I SAVE/SUBMIT it to the Medzmate station.

**Description**

The user should confirm the detailed information before submission, this ensures correctness of the data, as well as it relies the responsibility on the user to provide the correct data.

**Acceptance Criteria:**

* Form should be validated for required fields.
* A popup message should appear on submit, displaying the data and a message “By clicking *Send* you acknowledge that the information above is correct and you are responsible for its accuracy”.
* On submit data will be saved.
* On cancel the user will stay on the form to continue filling the data or correcting it.

**007 App-Station Data Transmission (Required)**

**As a**

Pharmacist

**I want to**

be able to transmit the configuration data from the app to the Medzmate station via wireless connection (E.g. Wi-Fi).

**So that**

I can login and configure the schedule in the station.

**Description**

As a pharmacist I want to be able to transmit the configuration data from the mobile application to the Medzmate station via wireless connection (E.g. Wi-Fi).

**Acceptance Criteria:**

* The configuration must be loaded by the Medzmate station.
* All the transmitted data should be stored in the Medzmate station.

**008 Main Screen (Optional)**

**As a**

Pharmacist

**I want to**

have a main screen

**So that**

I can access the settings, the patient information, and the loading deck of the station.

**Acceptance Criteria:**

* The main page should display links to:

1. Loading deck screen.
2. Settings screen.
3. Patient information screen.

**009 Patient information (Optional)**

**As a**

Pharmacist

**I want to**

save the patient information

**So that**

can send it to the station.

**Description**

Patient information should contain:

1. Name\*
2. Address\*
3. Phone\*
4. Email
5. Medicare #
6. Medicaid #
7. Health insurance #
8. Primary care provider: name\*, phone\*, and Email.
9. *Note: fields denoted with \* are required*

**Acceptance Criteria:**

* The patient information screen should match the medicine details form style
* It should contain all fields in the description
* It should submit the data as a JSON file to the station
* And it should create a log.

**010 Station: Communication Provider (Required)**

**As a**

Pharmacist

**I want to**

be able to transmit data using an internet connection (e.g. WiFi or Ethernet)

**So that**

data transmitted from the mobile application can be received by the Medzmate Station.

**Acceptance Criteria:**

* Http POST request should be processed by the communication provider in the Medzmate.
* Incoming files must be stored in the proper file location (e.g. *RunningDirectory/Documents*)
* Upon receive of new configurations the Medzmate process must be restarted.

**011 Station: Create Tests Module to Assert Software Quality (Required)**

**As a**

Developer

**I want to**

be able to test the functionality in the Medzmate station.

**So that**

future code changes do not break functionality.

**Acceptance Criteria:**

* All the public functions in a class need to be tested.

**012 Station: Reset Medzmate (Required)**

**As a**

Pharmacist

**I want to**

be able to reset Medzmate

**So that**

it restarts with the new configuration provided and begins a new scheduling.

**Acceptance Criteria:**

* Once a new configuration or straw information is sent to the Medzmate station it has to be able to restart and resume execution loading the new configuration provided.

**013 Create automated tests (Required)**

**As a**

Developer

**I want to**

Have automated test.

**So that**

when code changes are made, I can run automation to verify existing functionality was not affected.

**Acceptance Criteria:**

* All the public functions in a class need to be tested.

**014 Settings screen (Optional)**

**As a**

Pharmacist

**I want to**

be able to configure the settings of the station.

**So that**

I can set the type of alarm/s, set who to contact in case medicine is not taken, set the time threshold before and after the schedule time for the alarm to start/stop.

**Description**

Settings should contain:

1. Alarm Type ( Light, sound)
2. Sound type (Default, Customized)
3. Time interval to start alarm before schedule time (ex : 10 mins)
4. Time interval to stop alarm after schedule time (ex:30 mins)
5. Primary contact name (required)
6. phone
7. email
8. Secondary contact name
9. phone
10. email

**Acceptance Criteria:**

* The settings screen should match the medicine details form style
* It should contain all fields in the description
* It should submit the data as a JSON file to the station
* And it should create a log.

**015 Station: Dispensing Alerts (Required)**

**As a**

Patient

**I want to**

receive “Dispensing Alerts” based on the configured schedule in the Medzmate station by the pharmacist using the mobile application

**So that**

there is a sound notification, or visual notification when it is time to dispense a medication.

**Acceptance Criteria:**

* Based on the predefined schedule produced with the mobile application by the pharmacist, the Medzmate Station must signal that medications can be dispensed.
* Signaling must be presented as a print out in the console.

**016 Station: Load Medzmate Configuration into Medzmate Process (Required)**

**As a**

User

**I want to**

See the effect of a loaded configuration in JSON format into the Medzmate running process.

**So that**

scheduling and general configuration can be loaded into the process logic so it reacts for dispensing alerts, throws the configured alerts and saves the patient’s information.

**Acceptance Criteria:**

* Stored configuration in the Medzmate in a JSON file needs to be loaded by the Medzmate Process.
* The process needs to be able to understand the data stored in the JSON file and react based on the configuration reflected there.

**017 App: Log Off (Should)**

**As a**

Pharmacist

**I want to**

have a way of log off

**So that**

I can connect to a different Medzmate station.

**Acceptance Criteria:**

* Pharmacist can log off without closing the application.

**018 Medzmate Station Integration Testing (Must)**

**As a**

Developer

**I want to**

test and refactor the Medzmate Station components and their integration.

**So that**

all the components are properly connected and functioning as expected.

**Acceptance Criteria:**

* All messages should display correctly and in the right time.
  + Patient’s information.
  + Global configuration.
  + Straws information
* Application should restart when the mobile app sends a new straw and/or loads new global configuration
* Code must be organized and commented.
* The application is ready for demo.

**019 Test and Refactor (Must)**

**As a**

Developer

**I want to**

test and refactor the Medzmate system as needed.

**So that**

all the overall functionality is covered, bugs are fixed and code is cleaner and more organized.

**Acceptance Criteria:**

* All manual tests must be executed and pass
* All Automated tests must pass after refactoring.
* Code must be organized and commented.
* The application is ready for demo.

## Pending User Stories

**020 App-Station Personalize notification (Optional)**

**As a**

Patient, Relative, or Pharmacist

**I want to**

enable or disable different types of notifications:

1. Glowing light.
2. Alarms sound.

**So that**

the patient can have a more personalized notification suited to his/her needs.

**Description**

The user should be able to select among multiple forms of notifications.

**021 App: Retrieve Pills information (Optional)**

**As a**

Patient, Relative, or Pharmacist

**I want to**

click an information button in the app

**So that**

I can see the information of the medicine loaded in the machine.

**Description**

The user must be able to see the information about the medicine loaded into Medzmate

**022 Station: Retrieve Pills (Required)**

**As a**

Patient

**I want to**

press a button on the Medzmate when the machine notifies me it's time for my pills

**So that**

I can retrieve the scheduled medicine

**Description**

After the user is notified, the user should press a button to get the correct medicine dispensed from Medzmate.

**023 Record a customized message (Optional)**

**As a**

Patient, Relative, or Pharmacist

**I want to**

Record a personalized message

**So that**

this message is played when it is time to take the medication

**Description**

The user should be able to record a personalized message, that will be used later to notify the patient that it is time to take the medication.

# Project Plan

The Medzmate project plan describes the work done to date, the choices we had regarding hardware/software, and a brief explanation of how or why we arrived to the decision to use those specific resources.

## Hardware and Software Resources

**Hardware:**

1. Nexus 7
2. Raspberry PI
3. PC I7, 16GB Ram , 500GB SSD

**Software:**

1. Operating Systems:
   1. Ubuntu 12.0: Free operating system with better capabilities than Raspbian to use as development environment for the Medzmate Station software.
   2. Raspbian: Free operating system based on Debian optimized for the Raspberry Pi hardware.
   3. Android: Although the developed mobile application can be compiled to several mobile operating systems, we relayed on Android to deploy the completed versions of our application and test it on a Nexus 7 device.
   4. Windows 10: The development environments provided by Microsoft run on Windows platforms. To be able to develop the mobile application we needed Windows and IDEs such as Visual Studio 2015. The multi-platform capabilities provided in compilation time by Visual Studio 2015 with Apache Cordova.
2. IDEs:
   1. Visual Studio 2015: As explained earlier Visual Studio is a great IDE with excellent development capabilities and the ability to integrate with SDKs such as Apache Cordova. Provides excellent code analyzers and friendly user interface.
   2. NetBeans: Great IDE to use in Ubuntu with native Linux C++ compilers and standard Linux libraries. Used as the development environment for the code running within the Medzmate Station. Provides excellent code analyzers and friendly user interface.
3. Frameworks:
   1. Apache Cordova: Platform that wraps your HTML/JavaScript app into a native container which can access the device functions of several platforms. These functions are exposed via a unified JavaScript API, allowing one set of code to target nearly every phone or tablet on the market today and publish to their app stores.
   2. JQuery Mobile: A framework that allows you to design a single highly-branded responsive web site or application that will work on all popular smartphone, tablet, and form inputs are enhanced to be touch-friendly, including the addition of a set of touch-friendly UI widgets.
   3. Node js: Platform whose libraries are focused on building server-side applications in JavaScript. (Needed for the service on the Medzmate station).
   4. Selenium: It is the core technology for Appium.
   5. Appium: Open source Mobile Automation testing tool, supports both Android and iOS.
4. Tools:
   1. FileZilla client: ftp client to transfer files to/from Raspberry Pi.
   2. Putty: Tool to connect via SSH console to the Raspberry Pi, useful for remote development.

## Sprints Plan

### Sprint 1

(08/28/2015 - 09/11/2015)

Planning and environment setup

### Sprint 2

(09/11/2015 - 09/25/2015)

**User Story #001 Add new medicine information**

***Tasks***

* Create app shell
* Create Loading Deck Screen

***Acceptance Criteria***

* After login, the pharmacist should be able to see a screen that mimics the loading deck of Medzmate
* When the Pharmacist clicks one of the buttons a form to fill the medicine prescription should pop up

***Modeling***

Refer to UML diagrams in Appendix A that were created or modified to model the functionality that will be implemented in this sprint.

**User Story # 002 Add new medicine detailed information**

***Tasks***

* Create form fields
* Format fields

***Acceptance Criteria***

* A form should display requiring the information mentioned above
* Information should save

***Modeling***

Refer to UML diagrams in Appendix A that were created or modified to model the functionality that will be implemented in this sprint.

### Sprint 3

(09/25/2015 - 10/09/2015)

**User Story # 002 Add new medicine detailed information (*continuation …*)**

***Tasks***

* Create JSON formatted string from values on submit
* Save medicine information into a file or database
* Each form should contain a straw id

***Acceptance Criteria***

* A form should display requiring the information mentioned above
* Information should save for each form

**User Story # 003 Create Medzmate Emulator**

***Tasks***

* Design JSON structure.
* Load schedule configuration from JSON.
* Throw alerts based on the pills schedule.
* Process schedule configuration.
* Design scheduling algorithm.

***Acceptance Criteria***

* **Emulator should print comprehensive messages based on the actions triggered by the mobile application.**
* **Emulator needs to be able load the configuration stored in the Medzmate that was previously sent by the mobile application.**
* **Emulator should be able to throw alerts based on the medication’s schedule.**

***Modeling***

Refer to UML diagrams in Appendix A that were created or modified to model the functionality that will be implemented in this sprint.

### Sprint 4

(10/09/2015 - 10/22/2015)

**User Story # 004 App-Station Pharmacist Login**

***Tasks***

* Add Style to login page
* Save authenticated user to log file
* Install MongoDB on raspberry pi
* Research TCP/IP connection using java script
* Research TCP/IP connection using C++
* Implement web service (TCP/IP) using NODE js
* Connect to station login service

***Acceptance Criteria***

* The login page should be touch friendly
* System should validate username
* System should validate password
* On valid login the system should display the welcome screen

**User Story # 005 Add a page to connect to a Medzmate given a unique id**

***Tasks***

* Add connect to Medzmate form

***Acceptance Criteria***

* Page accepts a Medzmate id
* When user clicks connect a request is sent to the station
* Upon confirmation the login page is loaded

**User Story # 006 User should confirm the entered medicine detailed information**

***Tasks***

* Create confirmation display form
* Save log

***Acceptance Criteria***

* Form should be validated for required fields
* A popup message should appear on submit, displaying the data and a message “By clicking Send you acknowledge that the information above is correct and you are responsible for its accuracy”
* On submit data will be saved
* On cancel the user will stay on the form to continue filling the data or correcting it

**User Story # 007 App-Station: Data Transmission**

***Tasks***

* Connect to service on station and send data to be saved
* Save data to a JSON file in the station

***Acceptance Criteria***

* The configuration must be loaded by the Medzmate station.
* All the transmitted data should be stored in the Medzmate station.

***Modeling***

Refer to UML diagrams in Appendix A that were created or modified to model the functionality that will be implemented in this sprint.

### Sprint 5

(10/23/2015 - 11/06/2015)

**User Story # 007 Data Transmission *(continuation…)***

***Tasks***

* Compile code in Linux
* Compile and install code in raspberry pi

***Acceptance Criteria***

* The configuration must be loaded by the Medzmate station.
* All the transmitted data should be stored in the Medzmate station.

**User Story #008 Main Screen**

***Tasks***

* Create form

***Acceptance Criteria***

* Main screen contain access to :
  + Settings
  + Patient information
  + Loading deck

**User Story #009 Patient Information**

***Tasks***

* Create patient info form
* Set style to match medicine info form
* Create JSON file with patient information
* Send patient information to Medzmate station
* Create a log file on form submission

***Acceptance Criteria:***

* The patient information screen should match the medicine details form style
* It should contain all fields in the description
* It should submit the data as a JSON file to the station
* And it should create a log.

**User Story #010 Station-Communication Provider**

***Tasks***

* Document the setup process of the Raspberry PI
* Validate configuration file.
* Implement trigger functionality to reload the configuration.
* Research and design.
* Compile and install code in Raspberry Pi

***Acceptance Criteria***

* **Http POST request should be processed by the communication provider in the Medzmate.**
* **Incoming files must be stored in the proper file location (e.g. RunningDirectory/Documents)**
* **Upon received a new configuration the Medzmate process must be restarted.**

**User Story #011 Create Tests Module to Assert Software Quality**

***Tasks***

* Refactor code to remove compiler warnings.
* Create test for scheduler.
* Implement trigger functionality to reload the configuration.
* Create Tests for JSON parser.
* Create Tests for JSON lexer.

***Acceptance Criteria***

* All the public functions in a class need to be tested.

**User Story #012 Station: Reset Medzmate**

***Tasks***

* Implement trigger functionality to reload the configuration and restart the process.

***Acceptance Criteria***

* Once a new configuration or straw information is sent to the Medzmate station it has to be able to restart and resume execution loading the new configuration provided.

***Modeling***

Refer to UML diagrams in Appendix A that were created or modified to model the functionality that will be implemented in this sprint.

### Sprint 6

(11/06/2015 - 11/20/2015)

**User Story #013 Create automated tests**

***Tasks***

* Create test cases for User Story #002 Add new medicine detailed information.
* Create test cases for User Story #001 Add new medicine information.
* Refactor code to match Visual Studio 2013 structure.
* Write test cases for login.
* Install Appium and Selenium dependencies.
* Implement smoke test.
* Research Appium automation framework.

***Acceptance Criteria***

* All the public functions in a class need to be tested.Alarm Type ( Light, sound).

**User Story #014 Settings Screen**

***Tasks***

* Create form
* Create confirmation popup
* Set form style
* Create JSON file and send it to station

***Acceptance Criteria***

* Application should have a form with the following fields
  + Alarm Type ( Light, sound)
  + Sound type (Default, Customized)
  + Time interval to start alarm before schedule time (ex : 10 mins)
  + Time interval to stop alarm after schedule time (ex:30 mins)
  + Primary contact name (required)
  + phone
  + email
  + Secondary contact name
  + phone
  + email
* Upon submit info will be save to the app database and sent to Medzmate station

**User Story #015 Station: Dispensing Alerts**

***Tasks***

* Load Multiple Straw configuration files
* Throw Alerts Types based on general configuration
* Change compiler directives in Make file
* Refactor Scheduler Logic

***Acceptance Criteria***

* The Medzmate Station should be able to load multiple dispenser / straws configurations from JSON files into the application
* When the application starts the station should print the information of each straw in the console
* The main configuration loaded on US # 013 together with the straws information must be processed to throw the dispensing alerts.
* Two main types of alerts must be displayed in the console: Blinking light and Sound notifications. If one of them is removed from the main configuration file, then it cannot appear as an alert.

**User Story #016 Load Medzmate Configuration into Station**

***Tasks***

* Create Object Model to Support the General Configuration
* Load General Configuration from JSON.

***Acceptance Criteria***

* Medzmate Station should be able to load a JSON file containing the following fields:
  + Alarm Type ( Light, sound)
  + Sound type (Default, Customized)
  + Time interval to start alarm before schedule time (ex : 10 mins)
  + Time interval to stop alarm after schedule time (ex:30 mins)
  + Primary contact name (required)
  + phone
  + email
  + Secondary contact name
  + phone
  + email
* Medzmate Station should be able to print this information in the console once the program starts.

***Modeling***

Refer to UML diagrams in Appendix A that were created or modified to model the functionality that will be implemented in this sprint.

### Sprint 7

(11/20/2015 - 12/4/2015)

**User Story #017 App: Log Off (Should)**

***Tasks***

* Implement log off
* Add log off confirmation
* Add log off button
* Refactor Navigation

**Acceptance Criteria:**

* Pharmacist can log off without closing the application.

**User Story #018 Medzmate Station Integration Testing (Must)**

***Tasks***

* Integration Testing and Refactoring.
* Test Wi-Fi Communication

**Acceptance Criteria:**

* All messages should display correctly and in the right time.
  + Patient’s information.
  + Global configuration.
  + Straws information
* Application should restart when the mobile app sends a new straw and/or loads new global configuration
* Code must be organized and commented.
* The application is ready for demo.

**User Story #019 Test and Refactor (Must)**

***Tasks***

* Refactor code.
* Run tests and fix bugs.

**Acceptance Criteria:**

* All manual tests must be executed and pass
* All Automated tests must pass after refactoring.
* Code must be organized and commented.
* The application is ready for demo.

***Modeling***

Refer to UML diagrams in Appendix A that were created or modified to model the functionality that will be implemented in this sprint.

# System Design

The purpose of this chapter is to show and explain the software architecture chosen for Medzmate. Firstly, there is an overview of the entire software architecture by providing a package diagram featuring the subsystem decomposition of the system as well as an explanation of the purpose of each subsystem. The system’s architectural patterns to be used are also explained. Subsequently, each subsystem is further explained in detail, along with the use cases associated with each one. The next subsection deals with hardware and software mapping for the Medzmate system; a deployment diagram is used to enhance clarity when describing the details of this section. The final subsection is dedicated to identify the design patterns used and briefly explained why they were selected.

## Architectural Patterns

In this project we used the Three-Tier pattern as the primary architecture and the Event-Driven model as the secondary architectural pattern. Three-Tier pattern consists of the following layers: Interface, Application Logic and Storage.

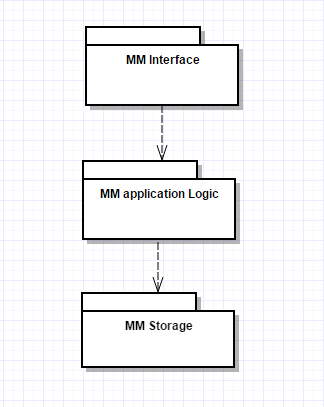


Figure 4.1 - Medzmate 3-tier Architecture. MM Interface, MM Application Logic and MM Storage

In our system, the package called MM Interface deals with the UI and it is concerned with enhancing user experience by providing user-friendly options that can be easily understood by the pharmacist. Once a certain option is selected, the necessary input is passed to the Application Logic. The Application Logic in our system consists of three subsystems: MM *Manage, MM Functions and MM Security*. The purpose of MM Manage is to provide all the managerial functions –e.g. Create new user, System logs-. On the other hand, MM Functions provides the logic behind the features that can be used by any user in the system –e.g. See medicine information, Record customized message-. The final subsystem in the Application Logic is MM Security, which handles the authentication and/or authorization by both the pharmacist and regular users. Finally, the Storage layer is represented by a single subsystem called MM Storage. It consists of flat files that contain relevant information such as Configurations for different straws and the user's Password. The main reason for selecting the Three-Tier Architectural Pattern is that the main features of our system: Interface, Logic and Storage are well-defined, independent layers; therefore, a change in one will not affect the others, which makes the process of updating/changing some requirements of the system much easier to manage. Also, three-tier architecture allows us to enable the development or modification of multiple user interfaces for the same application logic.

The event-driven pattern works as the source of all processes within our system. Each package is associated with a function that comes from one of the subsidiary functions. These are events that when processed are acknowledged and handled by the subsystem; however this does not necessarily mean that a change in one will affect another, as described in our three-tier model.

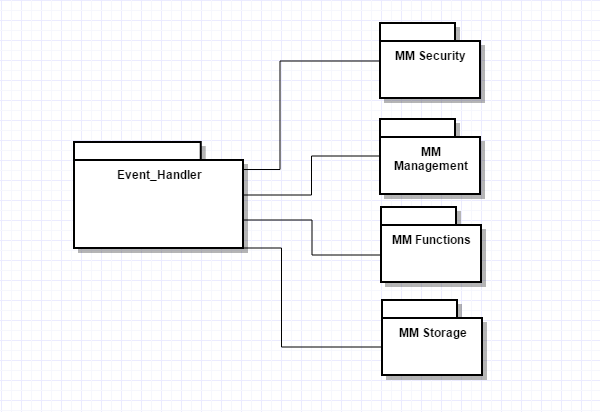


Figure 4.2 - Medzmate Event-Driven Architecture. Including major subsystems.

**Medzmate Station**

The Medzmate Station software is implemented based on a component-based architecture. The packages or components within the system are separated considering their responsibilities and a well-defined set of functions. The separation of concerns and the establishment of modules or components brings several benefits in terms of organization, implementation, testing and understanding of system.

The Medzmate Station integrates six main packages of software. The **Medzmate Process** is the main one, which is the entry point for any execution. This process is in charge of running and monitoring all the actions performed over the Medzmate station. It is basically the living soul of the equipment. The **Communication Provider** is in charge of all the network communication related functionality, and this is the one that isolates the Medzmate process from the underlying communication infrastructure—Wi-Fi, Bluetooth, cable, etc. Data obtained by means of the **Communication Provider** is serialized and de-serialized by the **Medzmate Process** using the **Serializer** package.

The **Serializer** will be in charge of transforming the incoming message from the network into an instance of the model that the Medzmate process works with. Using this approach the entire logic built using the object model is isolated from the underlying message structure, format or communication protocol. The **Serializer** uses a custom made Parser to load the simplified JSON structure received. Using Lexicographic analysis the entire JSON is tokenized and then Parser based on our specific Grammar.

The **Scheduler** and **Alert Manager** packages work together in the process of notifying the user when medications must be taken. The Alert Manager complexity is based on the fact that it will take care of interacting with hardware controllers and kernel level calls to throw different types of alerts such as light blinking, voice messages or sound notifications. The Scheduler will take care of processing the time allocations of each medication based on the computer time.

Finally, but not less important the **Authentication Service** will manage all the complexity related to security and user accounts within the **Medzmate Station**.

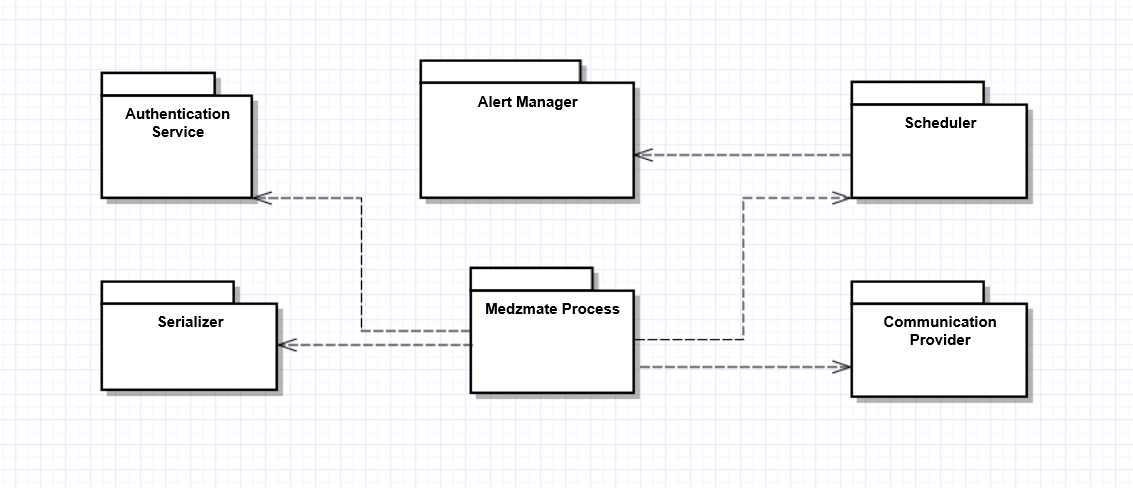


Figure 4.3 - Medzmate Station Component based architecture.

### System and Subsystem Decomposition

**Interface Subsystem:** the interface subsystem contains functionality such as selecting a Medzmate to configure it, entering a username and password, seeing logs, selecting the correct straw, entering detailed medicine information, creating customized messages, and canceling the schedule submission. Patients will access only the message and the medicine information retrieval sections. In the case of the pharmacist, he/she will be allowed to enter username/password and perform the rest of the aforementioned features.

**Medzmate Functions Subsystem:** the MM Functions subsystem provides the logic behind the features that can be used by any user in the system. Regular users will access features such as customize message; whereas the pharmacist can add detailed information, retrieve logs, cancel schedule, and submit schedule.

**Manage Subsystem:** The purpose of MM Manage is to provide the logic behind the features associated with the pharmacist. This subsystem will authenticate the user pharmacist, and then grant authorization to the functionality specific to the user level of security.

**Security Subsystem:** The MM Security subsystem handles the authentication and/or authorization by both the pharmacist and patients. It also deals with the encryption of the data in order to prevent misuser from accessing accounts.

**Storage Subsystem:** The MM Storage subsystem consists of flat files that contain relevant information such as valid username and password, system logs, and medicine schedules for each Medzmate machine.

### Deployment Diagram

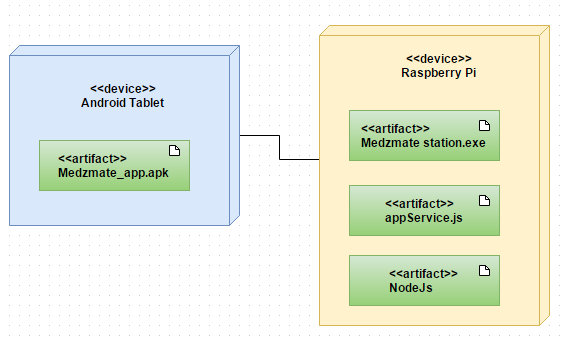


Figure 6.1 -Medzmate Deployment Diagram

### Design Patterns

The Medzmate software utilizes mainly Structural and Behavioral Patterns--according to the classification established by Gamma *et al.* in “Design Patterns Elements of Reusable Object-Oriented Software. Below you will find a description of how each pattern was useful in our design and implementation.

*Behavioral Patterns*

**Observer or Publish-Subscribe:**

This pattern is used in the mobile application to be able to implement the UI. Events triggered by user actions in UI elements are subscribed to actions in the application logic—such as Save, Submit, Cancel buttons. It also monitors the physical buttons on the device such as the “back” and “home” buttons (Gamma *et al.*).

e.g.:

document.addEventListener("backbutton", onBackKeyDown, false);

function onBackKeyDown(e) {

console.log("backbutton pressed")

e.preventDefault();

navigator.app.exitApp();

}

Publisher-Subscriber pattern was particularly useful in the Medzmate Station to trigger the alert events when the scheduler (Publisher) raises the event that it is time to take the medication. The Alert Manager module (Subscriber) handles the different types of alerts configured--light alarm or sound notification.

**Interpreter:**

The Medzmate Station receives the configuration data in a JSON format. We implemented the Interpreter Pattern to represent the structure of our specific JSON files in a grammar. This grammar representation was useful to design a Parser and Lexer that translates this JSON into instances of our object model (Gamma *et al.*). The grammar used was the following:

JSON ::= ‘[‘ PropertyArray

PropertyArray::= Property Properties

Properties ::= ‘,’ Property Properties | ‘]’

Property ::= ‘{‘ NameValuePair

NameValuePair ::= ‘ “name” :’ *alphanumeric* ‘,’ ‘ “value” :’ *alphanumeric* ‘}’

**Iterator:**

The result of the lexicographic analysis to parse the JSON formatted files was a stream of key-value pair objects. By means of the iterator pattern we isolated the underlying implementation by which these elements are obtained. The Serializer receives a plain collection of key-value pairs which can be iterated to initialize each property of the object being constructed.

*Structural Patterns*

**Facade:**

This pattern is used in both the app and the station. To provide a single entry point that allows the communication between them. The *Facade* entry point is the web service that receives the HTTP requests and handles the complex subsystems of the Medzmate Station and their interactions (Gamma *et al.*).

The Medzmate Station also hides the functionality of the Parser and Lexer into a *Facade* class--the Serializer. This class provides a simplified interface to deal with the complex set of calls invoked on the Lexer and Parser and how they translate into an instance of the application’s Object Model.

# System Validation

The main objective of our test plan is to find and report as many bugs as possible to improve the integrity of our program. Although exhaustive testing is not possible, we will exercise a broad range of tests to achieve our goal. We will be testing the Medzmate app, the Medzmate station, as well as the integration between them.

**User Story # 004 - Pharmacist Login**

Note: all test cases for this story assume user has the correct Medzmate id

* **Test 004A** - Test if user is able to login successfully.

**Preconditions:**

1. User must be registered already

**Input Data:**

1. correct username,
2. correct password

**Steps to be executed:**

1. Enter input(correct )username and password on the respective fields
2. Click submit/login

**Expected result:**

1. User must successfully login to the web page

* **Test 004B** -Test if unregistered users is not able to login to the site

**Preconditions:**

1. N/A

**Input Data:**

1. incorrect username,
2. incorrect password

**Steps to be executed:**

1. Enter input(incorrect )username and password on the respective fields
2. Click submit/login

**Expected result:**

1. Proper error must be displayed and prompt to enter login again

* **Test 004C** - Test with valid username and empty password such that login must get failed.

**Preconditions:**

1. User must be registered already

**Input Data:**

1. Valid username
2. Empty password

**Steps to be executed:**

1. Enter the valid username in the user id and enter no password in the password field
2. Click submit/login

**Expected result:**

1. Proper error must be displayed and prompt to enter login again

* **Test 004D** -Test with empty username and valid password such that login must get failed.

**Preconditions:**

1. Registered user's password

**Input Data:**

1. Empty username
2. Valid password

**Steps to be executed:**

1. Leave the username empty in the user id
2. Enter a valid user's password in the password field
3. Click submit/login

**Expected result:**

1. Proper error must be displayed and prompt to enter login again

* **Test 004E** - Test with empty username and empty password and check if login fails

**Preconditions:**

1. N/A

**Input Data:**

1. N/A

**Steps to be executed:**

1. Leave the username empty in the user id
2. Leave the password empty in the password field
3. Click submit/login

**Expected result:**

1. Proper error must be displayed and prompt to enter login again

* **Test 004F** -Check of the password is masked on the screen i.e., password must be in bullets.

**Preconditions:**

1. N/A

**Input Data:**

1. Some password(can be a registered/unregistered)

**Steps to be executed:**

1. Enter the password field with some characters

**Expected result:**

1. The password field should display the characters in bullets such that the password is not visible on the screen

* **Test 004G** - Check if the login function handles case sensitivity.

**Preconditions:**

1. Registered user's password which is originally in lower case changed to uppercase or vice versa

**Input Data:**

1. Case changed username /password

**Steps to be executed:**

1. Enter the case changed username /password in the respective field and
2. Click login button

**Expected result:**

1. Proper error must be displayed and prompt to enter login again.

**User Story # 001 - Add new medicine information**

* **Test 001A – Verify each straw button opens the correct form.**

**Preconditions:**

1. User must be on loading deck screen

**Input Data:**

1. N/A

**Steps to be executed:**

1. Click button ‘A’

**Expected result:**

1. Medicine details from for straw ‘A’ should open

**User Story # 002 Add new medicine detailed information**

* **Test 002A – Verify all fields are editable.**

**Preconditions:**

1. User must be on medicine detailed information form

**Input Data:**

1. Medicine name = Aspirin
2. Doctor's name = Pedro Montero
3. Pills quantity = 1
4. Expiration Date (MM/DD/YY) = today + 1 month
5. 5. Mass = 1
6. 6. Dosage frequency (per day) = 4
7. 7. Frequency (days of the week) = daily
8. 8. Link with order container (type A, B, C) = none
9. 9. Symptoms = some Symptoms
10. 10. Side Effects = thinned blood

**Steps to be executed:**

1. Enter data
2. Click button ‘Submit’

**Expected result:**

1. Confirmation Popup appears with the same data as input
2. Confirmation Popup contains “Cancel” and “Submit” buttons.

* **Test 002B – Verify user can cancel submission.**

**Preconditions:**

1. Successfully run Test 003A

**Input Data:**

1. N/A

**Steps to be executed:**

1. Click button ‘Cancel’

**Expected result:**

1. User is back at Medicine detailed form
2. Form field are populated with recently entered data.
   1. Medicine name = Aspirin
   2. Doctor's name = Pedro Montero
   3. Pills quantity = 1
   4. Expiration Date (MM/DD/YY) = today + 1 month
   5. Mass = 1
   6. Dosage frequency (per day) = 4
   7. Frequency (days of the week) = daily
   8. Link with order container (type A, B, C) = none
   9. Symptoms = some Symptoms
   10. Side Effects = thinned blood
3. No data was saved to database

* **Test 002C – Verify user can Submit data.**

**Preconditions:**

1. Successfully run Test 003A

**Input Data:**

1. N/A

**Steps to be executed:**

1. Click button ‘Submit’

**Expected result:**

1. User is back at loading deck screen
2. Data was saved to database.
   1. Medicine name = Aspirin
   2. Doctor's name = Pedro Montero
   3. Pills quantity = 1
   4. Expiration Date (MM/DD/YY) = today + 1 month
   5. Mass = 1
   6. Dosage frequency (per day) = 4
   7. Frequency (days of the week) = daily
   8. Link with order container (type A, B, C) = none
   9. Symptoms = some Symptoms
   10. Side Effects = thinned blood
3. JSON file sent to station

* **Test 002D – Verify user can not submit without the Medicine Name.**

**Preconditions:**

1. User must be on medicine detailed information form

**Input Data:**

1. Medicine name = null
2. Doctor’s name = Pedro Montero
3. Pills quantity = 1
4. Expiration Date (MM/DD/YY) = today + 1 month
5. Mass = 1
6. Dosage frequency (per day) = 4
7. Frequency (days of the week) = daily
8. Link with order container (type A, B, C) = none
9. Symptoms = some Symptoms
10. Side Effects = thinned blood

**Steps to be executed:**

1. Enter data
2. Click button ‘Submit’

**Expected result:**

1. Form scrolls to Medicine name field and display a “Required field message”

* **Test 002E – Verify user can not submit without the pills quantity.**

**Preconditions:**

1. User must be on medicine detailed information form

**Input Data:**

1. Medicine name = Aspirin
2. Doctor’s name = Pedro Montero
3. Pills quantity = null
4. Expiration Date (MM/DD/YY) = today + 1 month
5. Mass = 1
6. Dosage frequency (per day) = 4
7. Frequency (days of the week) = daily
8. Link with order container (type A, B, C) = none
9. Symptoms = some Symptoms
10. Side Effects = thinned blood

**Steps to be executed:**

1. Enter data
2. Click button ‘Submit’

**Expected result:**

1. Form scrolls to Pills quantity field and display a “Required field message”

* **Test 002F – Verify user can not submit without the Frequency.**

**Preconditions:**

1. User must be on medicine detailed information form

**Input Data:**

1. Medicine name = Aspirin
2. Doctor’s name = Pedro Montero
3. Pills quantity = 1
4. Expiration Date (MM/DD/YY) = today + 1 month
5. Mass = 1
6. Dosage frequency (per day) = 4
7. Frequency (days of the week) = null
8. Link with order container (type A, B, C) = none
9. Symptoms = some Symptoms
10. Side Effects = thinned blood

**Steps to be executed:**

1. Enter data
2. Click button ‘Submit’

**Expected result:**

1. Form scrolls to Frequency y field and display a “Required field message”

* **Test 002G – Verify user can submit without non required fields**

**Preconditions:**

1. User must be on medicine detailed information form

**Input Data:**

1. Medicine name = Aspirin
2. Doctor's name = null
3. Pills quantity = 1
4. Expiration Date (MM/DD/YY) = null
5. 5. Mass = null
6. 6. Dosage frequency (per day) = 4
7. 7. Frequency (days of the week) = daily
8. 8. Link with order container (type A, B, C) = none
9. 9. Symptoms = null
10. 10. Side Effects = null

**Steps to be executed:**

1. Enter data
2. Click button ‘Submit’

**Expected result:**

1. Confirmation Popup appears with the same data as input
2. Confirmation Popup contains “Cancel” and “Submit” buttons.

**User Story # 011 - Station - Create Test Module to Assert Software Quality**

* **Test 011A.1 – Lexer Tests - Next() method.**

**Preconditions:**

1. JSON test file in hard drive.

**Input Data:**

1. [ { “name” : “Medicine Name”, “value”: “Ibuprofen 2345”}]

**Steps to be executed:**

1. Initialize Lexer with test file“TestResources/LexerTest.JSON”.
2. Call Next() function.
3. Call Next() function.
4. Call Next() function.
5. Call Next() function.
6. Call Next() function.
7. Close Lexer.

**Expected result:**

1. Lexer is initialized.
2. Returns token “[“.
3. Returns token “{“.
4. Returns token “”“.
5. Returns token “n“.
6. Returns token “a“.
7. Lexer Closed.

* **Test 011A.2 – Lexer Tests - Next() method.**

**Preconditions:**

1. JSON test file in hard drive.

**Input Data:**

1. & { “name” : “Medicine Name”, “value”: “Ibuprofen 2345”}]

**Steps to be executed:**

1. Initialize Lexer with test file“TestResources/LexerTest.JSON”.
2. Call Next() function.

**Expected result:**

1. Invalid character ‘&’
2. Lexer Closed.

* **Test 011B – Lexer Tests - Peek() method.**

**Preconditions:**

1. JSON test file in hard drive.

**Input Data:**

1. [ { “name” : “Medicine Name”, “value”: “Ibuprofen 2345”}]

**Steps to be executed:**

1. Initialize Lexer with test file“TestResources/LexerTest.JSON”.
2. Call Peek() function.
3. Call Peek() function.
4. Call Next() function.
5. Call Peek() function.
6. Call Peek() function.
7. Close Lexer.

**Expected result:**

1. Lexer is initialized.
2. Returns token “[“.
3. Returns token “[“. Successive calls to peek should not move to the next character.
4. Skips “[“
5. Returns token “{“.
6. Returns token “{“. Successive calls to peek should not move to the next character.
7. Lexer Closed.

* **Test 011C -- Lexer Tests - Get() method.**

**Preconditions:**

1. JSON test file in hard drive.

**Input Data:**

1. [ { “name” : “Medicine Name”, “value”: “Ibuprofen 2345”}]

**Steps to be executed:**

1. Initialize Lexer with test file“TestResources/LexerTest.JSON”.
2. Call Next() function.
3. Call Next() function.
4. Call Next() function.
5. Call Match() function.
6. Call Next() function.
7. Call Next() function.
8. Call Next() function.
9. Call Match(“Medicin”) function.
10. Call Get() function.
11. Call Get() function.
12. Call Get() function.
13. Close Lexer.

**Expected Result:**

1. Returns token “[“.
2. Returns token “{“.
3. Returns token “”“.
4. Skips “name“.
5. Returns token “”“.
6. Returns token “:“.
7. Returns token “”“.
8. Skips “Medicin“.
9. Returns token “e”.
10. Returns token “ “ (white space).
11. Lexer Closed.

* **Test 011D – Lexer Tests - Next() method.**

**Preconditions:**

1. JSON test file in hard drive.

**Input Data:**

1. [ { “name” : “Medicine Name”, “value”: “Ibuprofen 2345”}]

**Steps to be executed:**

1. Initialize Lexer with test file“TestResources/LexerTest.JSON”.
2. Call Next() function.
3. Call Next() function.
4. Call Next() function.
5. Call Match(“name”) function.
6. Call Next() function.
7. Close Lexer.

**Expected result:**

1. Lexer is initialized.
2. Returns token “[“.
3. Returns token “{“.
4. Returns token “”“.
5. Skips “name”
6. Returns token “”“.
7. Lexer Closed.

* **Test 011E -- Parser Tests - ParseSinglePropertyNode method.**

**Preconditions**

1. JSON test file in hard drive.

**Input Data:**

1. [ { “name” : “Medicine Name”, “value”: “Ibuprofen 2345”}]

**Steps to be executed:**

1. Initialize Parser with test file “TestResources/ParserTestSingleProperty.JSON”.
2. Call Parser.Start() method to obtain the collection of pairs <name, value>
3. Assert values are correct

**Expected results:**

1. Only one <name,value> tuple should be read with the corresponding values:
   1. name = Medicine Name
   2. value = Ibuprofen 2345

* **Test 011F -- Parser Tets - ParseMultiplePropertyNodes method.**

**Preconditions**

1. JSON test file in hard drive.

**Input Data:**

1. [ { “name” : “Medicine Name”, “value”: “Ibuprofen 2345”}]
2. [ { “name” : “Doctors Name”, “value”: “Test Name”}]

**Steps to be executed:**

1. Initialize Parser with test file “TestResources/ParserTestMultipleProperty.JSON”.
2. Call Parser.Start() method to obtain the collection of pairs <name, value>.
3. Assert two pairs and only two are read, with the correct values.

**Expected Results:**

1. Two <name,value> tuples should be read with the corresponding values:
   1. name = Medicine Name
   2. value = Ibuprofen 2345
   3. name = Doctors Name
   4. value = Test Name

# Glossary

*Medzmate (Station):* Actual Medzmate hardware where the pills are going to be dispensed.

*Medzmate App:* Mobile app that controls the Station.

# Appendix

## Appendix A - UML Diagrams

### Static UML Diagrams

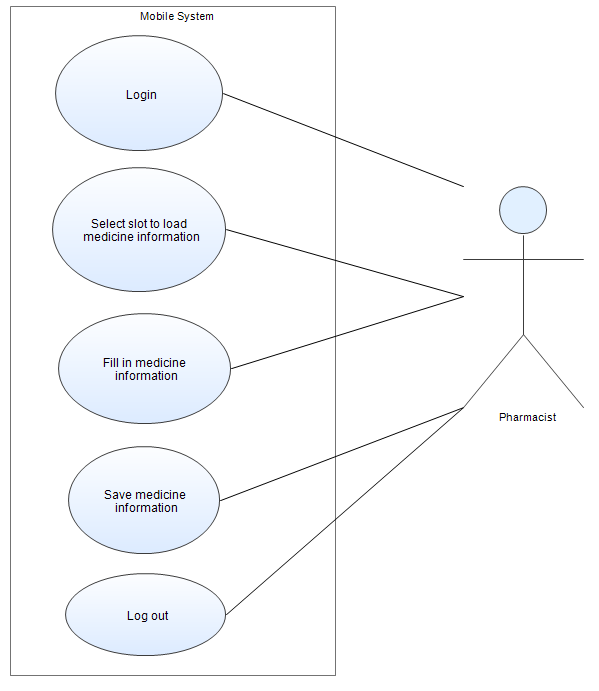


Figure a.1 - Use case Diagram: Pharmacist adds medicine

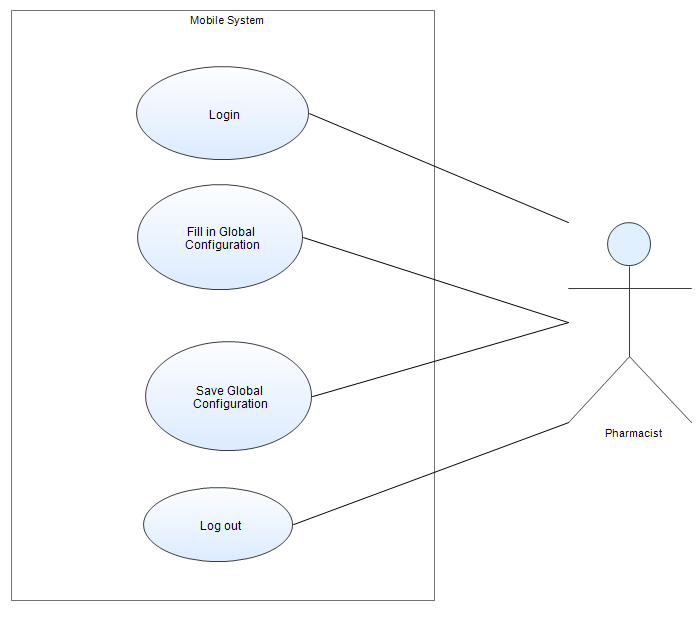


Figure a.2 - Use case Diagram: Pharmacist fills in global configuration

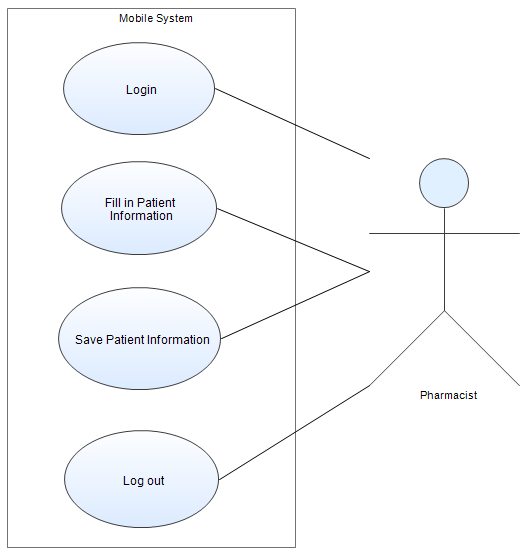


Figure a.3 - Use case Diagram: Pharmacist fills in Patient’s Information

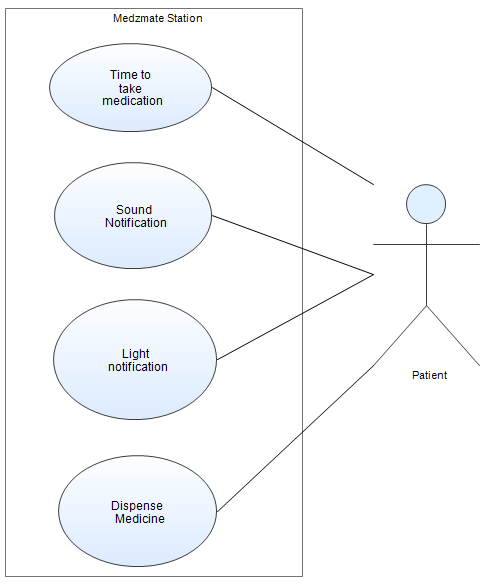


Figure a.4 - Use case Diagram: Patient receives alerts to take medication

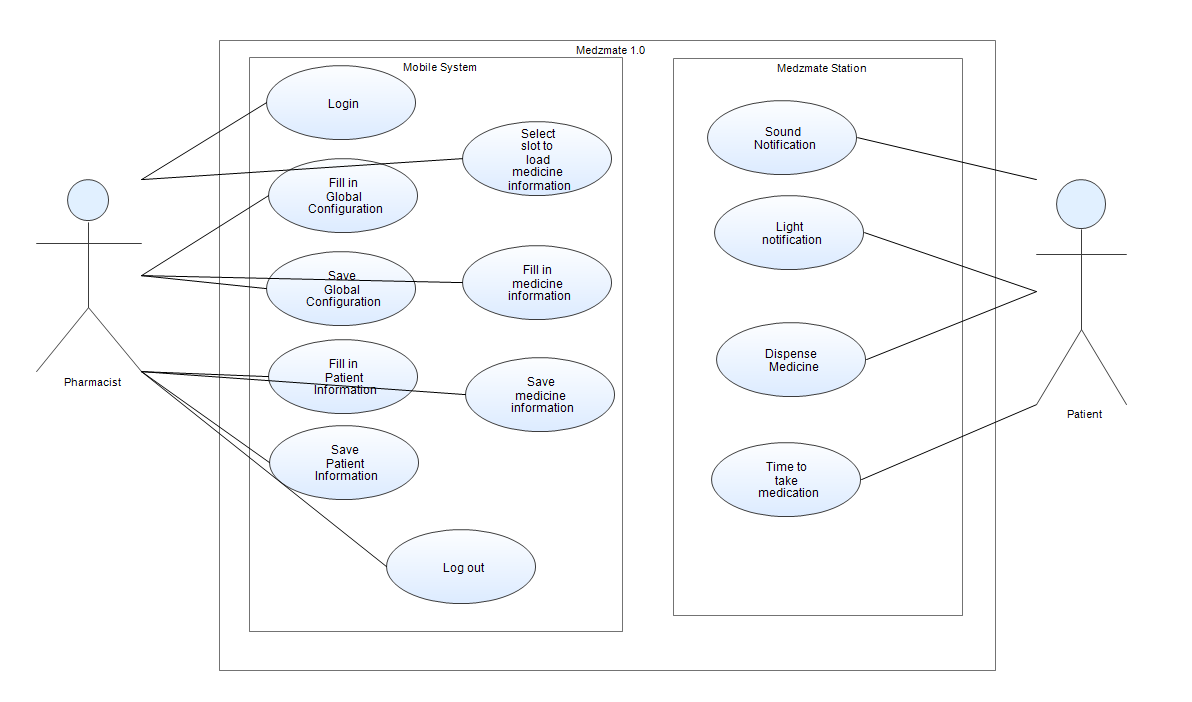


Figure a.5 - Use case Diagram: General use case scenario

### Dynamic UML Diagrams

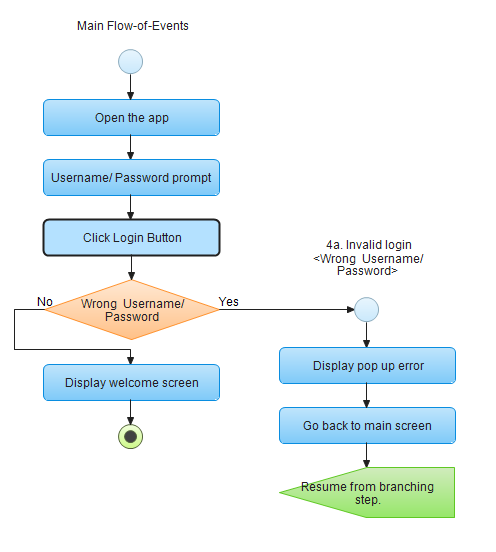


Figure a.1.2 UC-001 Login flow diagram

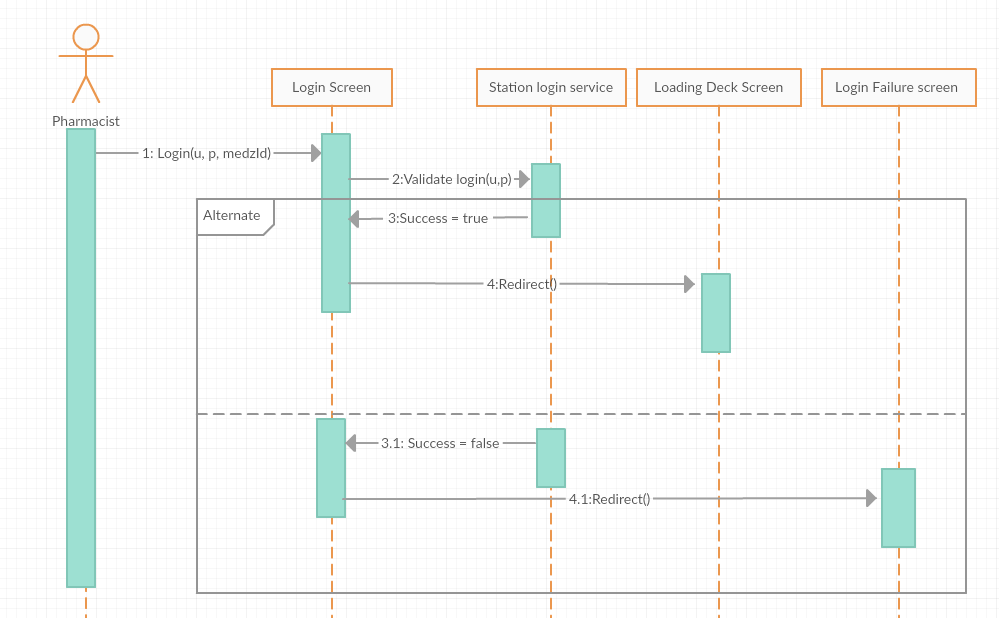


Figure a.1.2.1 UC-001 Login sequence diagram (u = username, p = password)

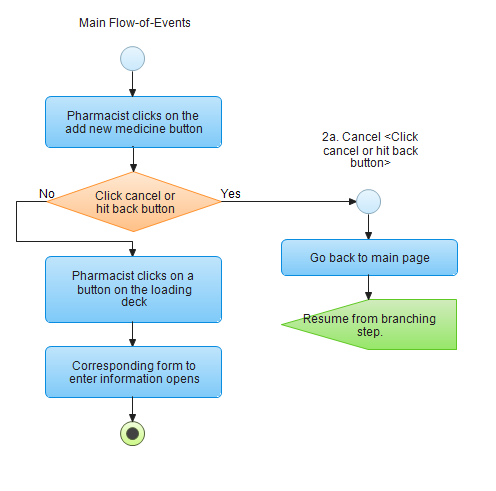


Figure a.1.3 UC - 002 Select slot to load medicine information flow diagram

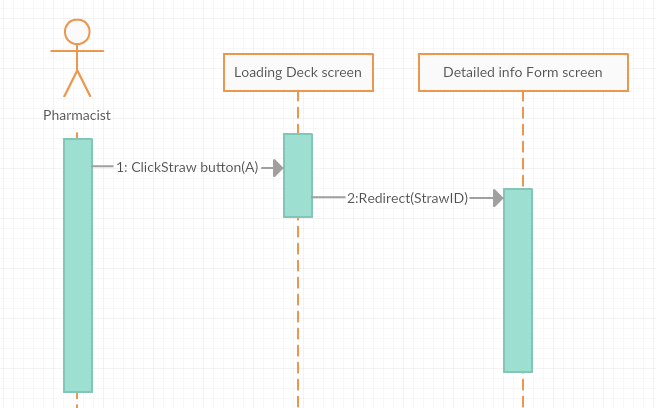


Figure a.1.3.1 UC - 002 Select slot to load medicine information sequence diagram

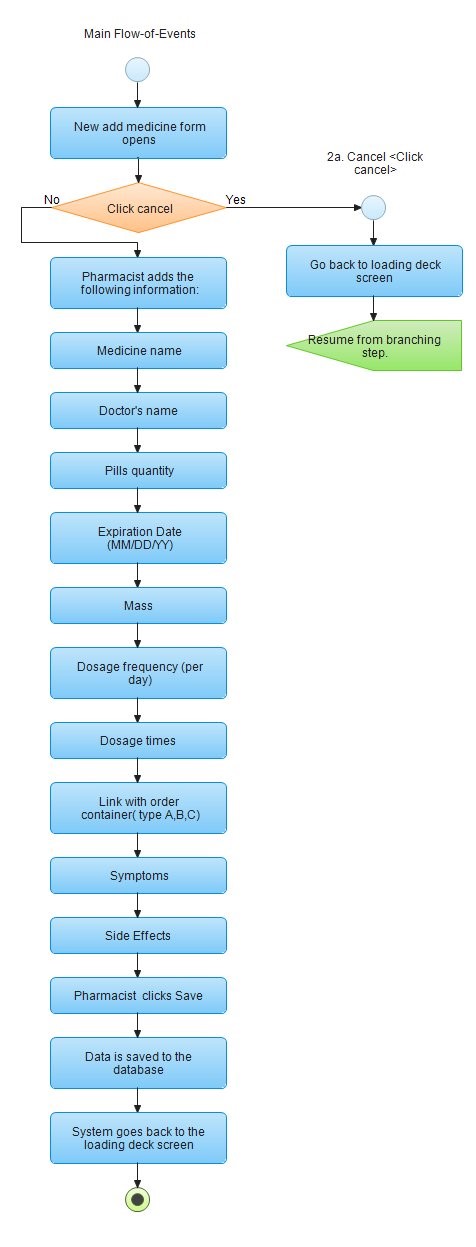


Figure a.1.4 UC - 003 Save medicine information flow diagram

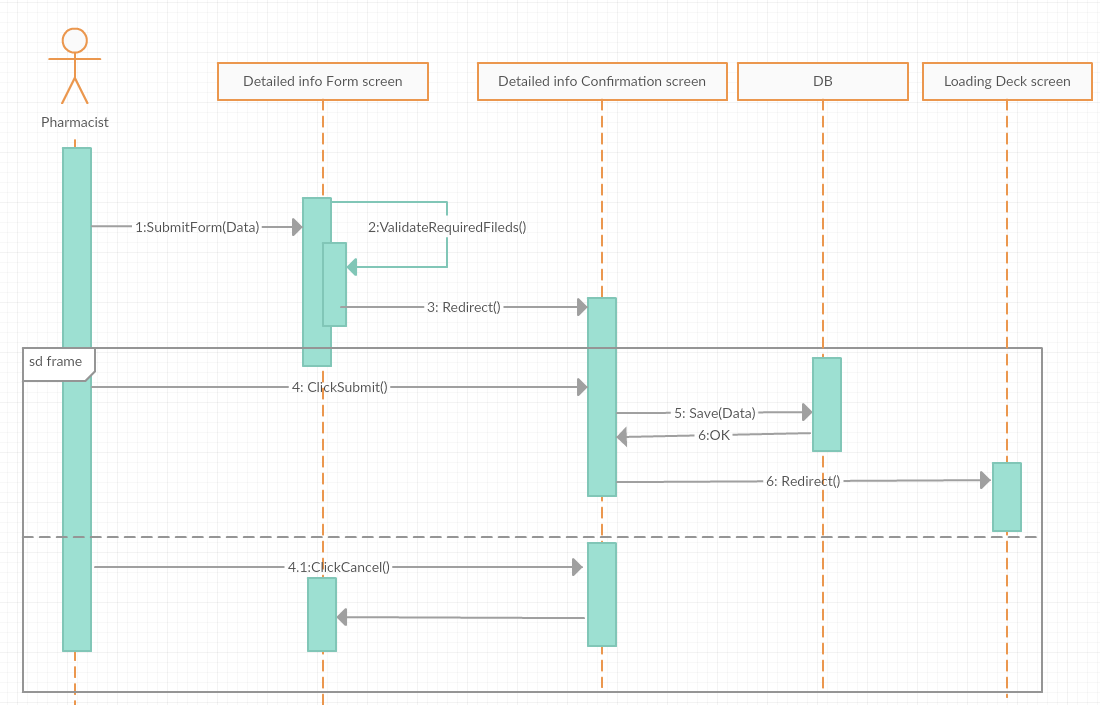


Figure a.1.4.1 UC - 003 Save medicine information sequence diagram

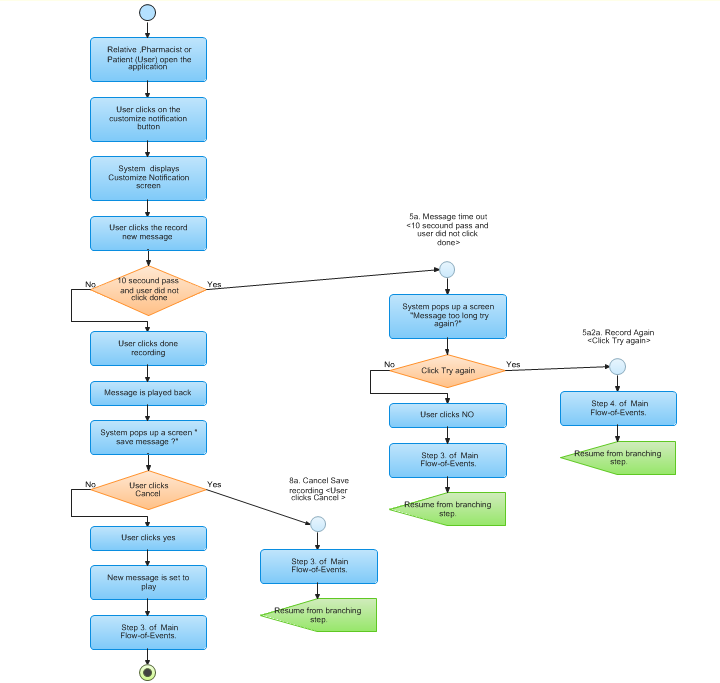


Figure a.1.4 UC - 004 Record message flow diagram

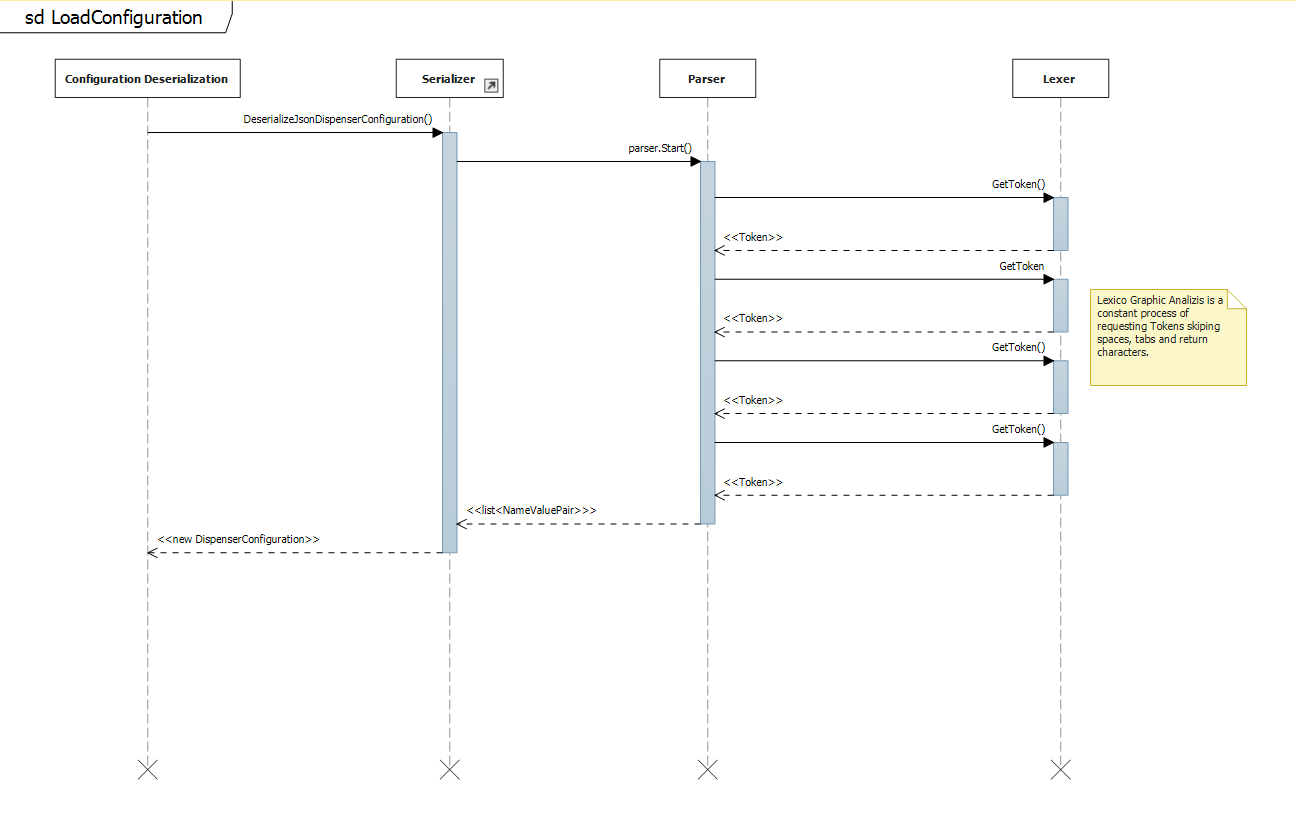
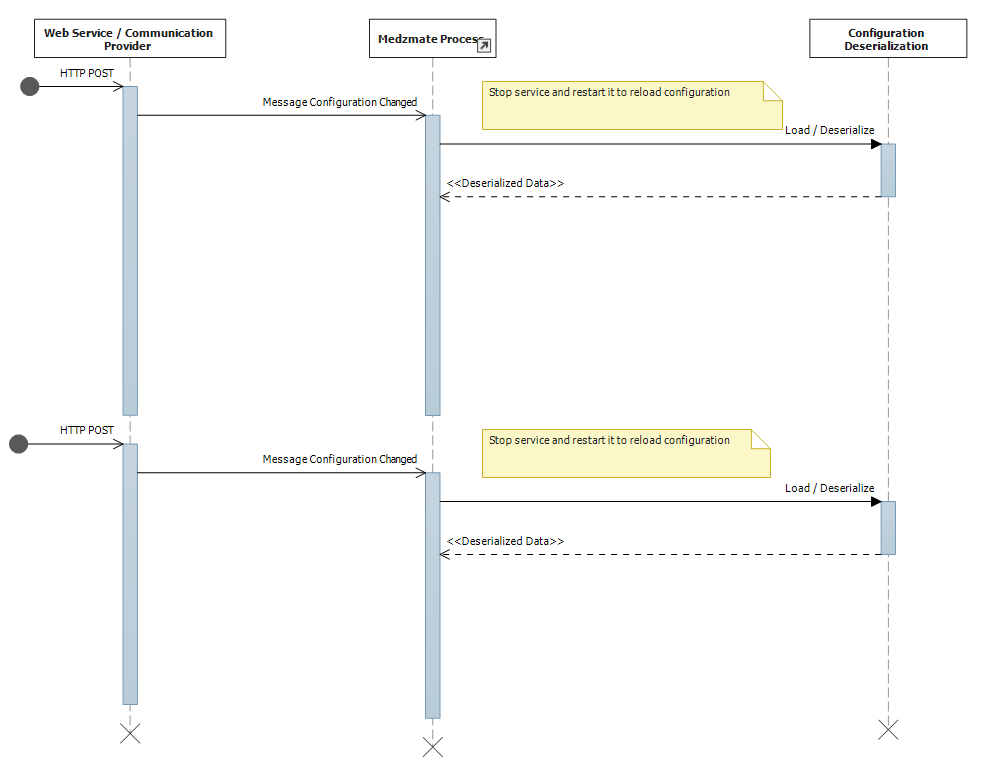


Figure a.1.5 Configuration Deserialization

Figure a.1.6 UC - Triggering a Configuration Loading

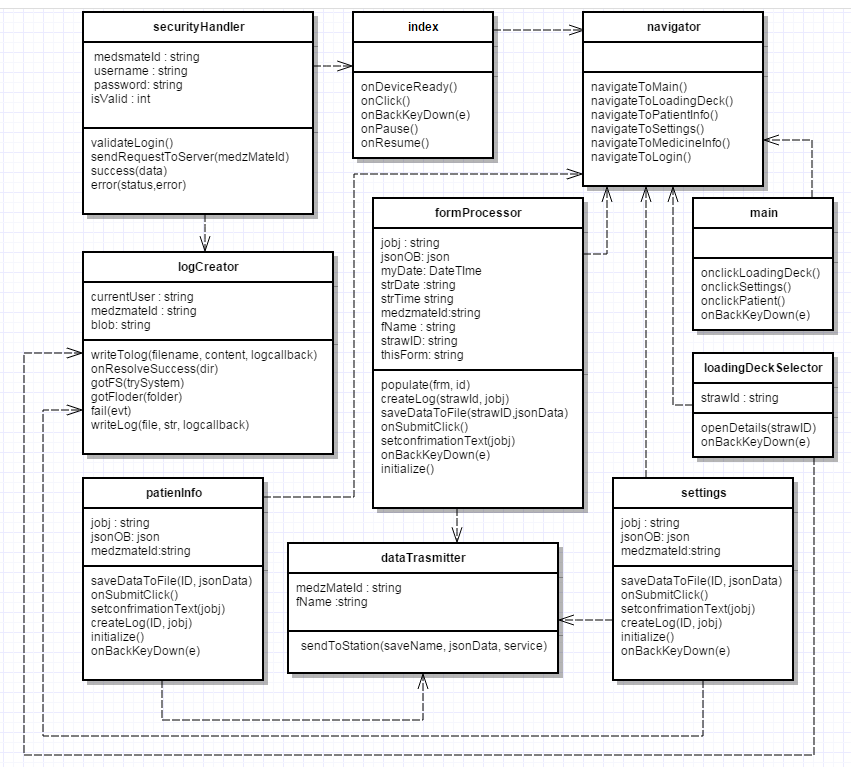


Figure a.2 - Class Diagram Medzmate App

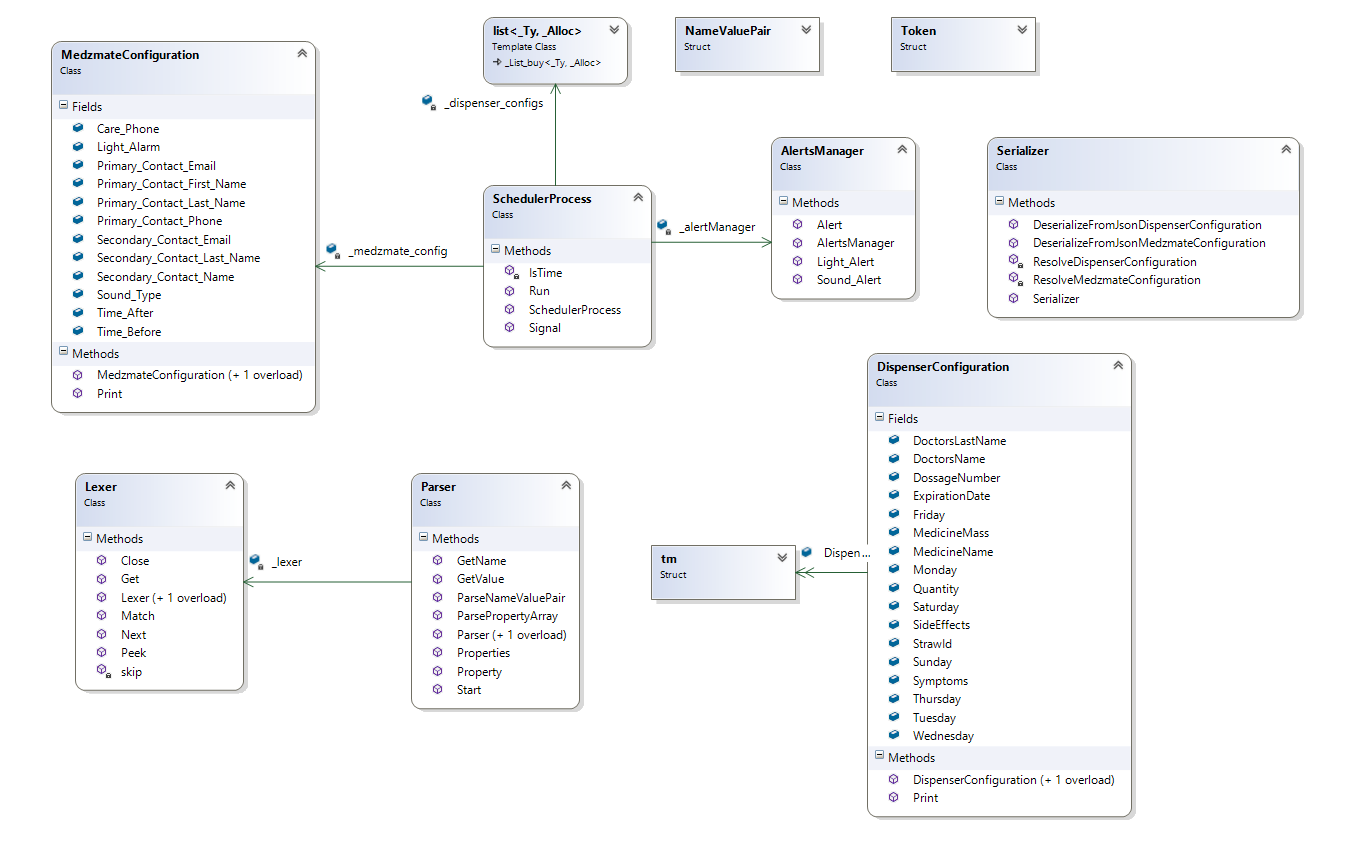


Figure a.2.1 - Medzmate Station Class Diagram.

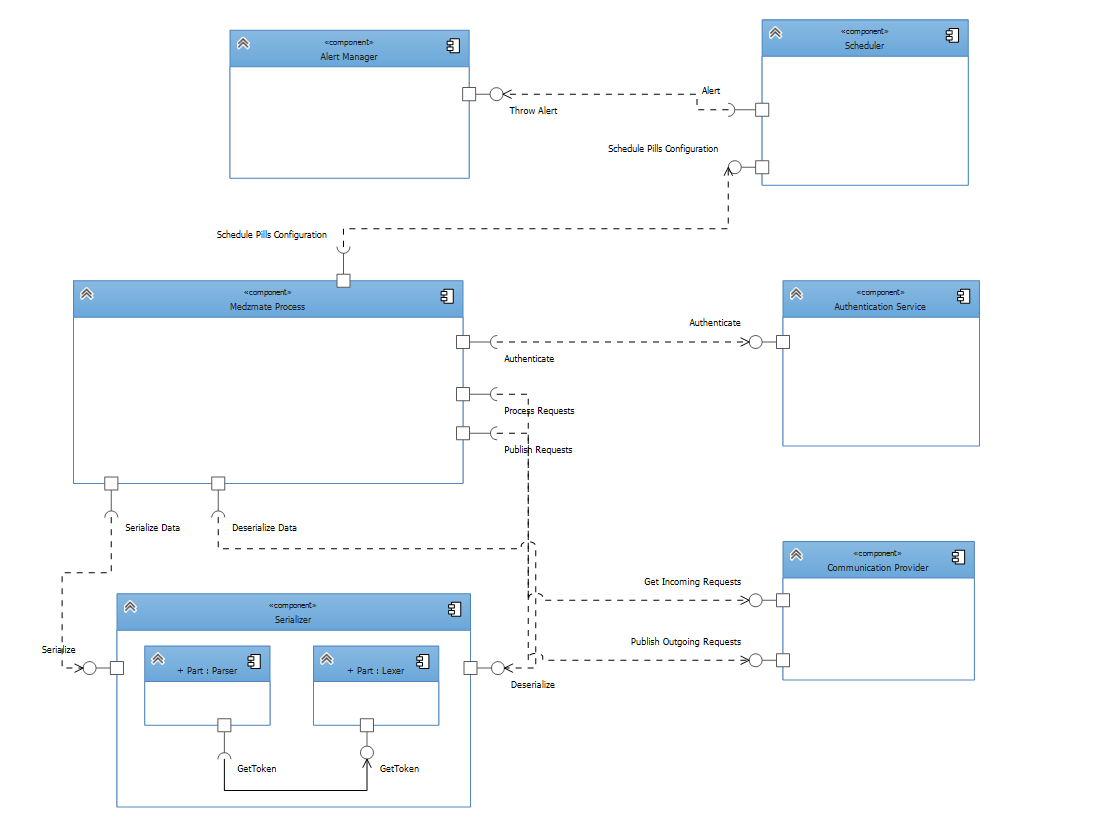


Figure a2.2 - Medzmate Station Component Diagram and Modules Interactions.

## Appendix B - User Interface Design

## 

Figure **b1** - Login page

## 

Figure **b2** - Loading deck page

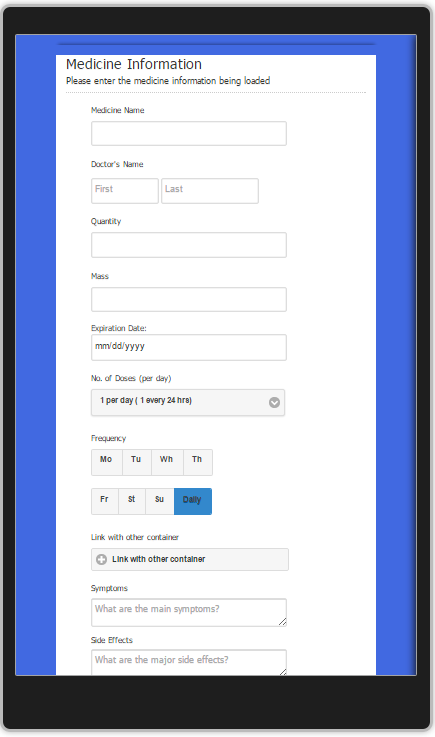


Figure **b3** - Detailed medicine form

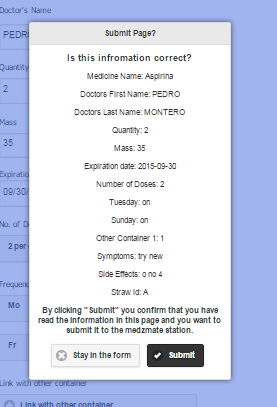


Figure **b4** - Detailed medicine submit confirmation

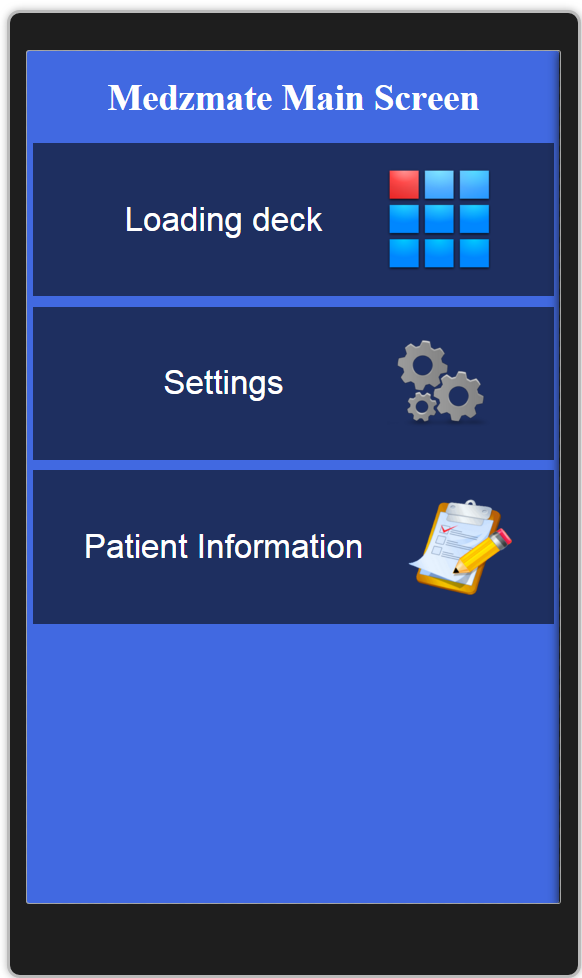


Figure **b5** - Mezdsmate main screen

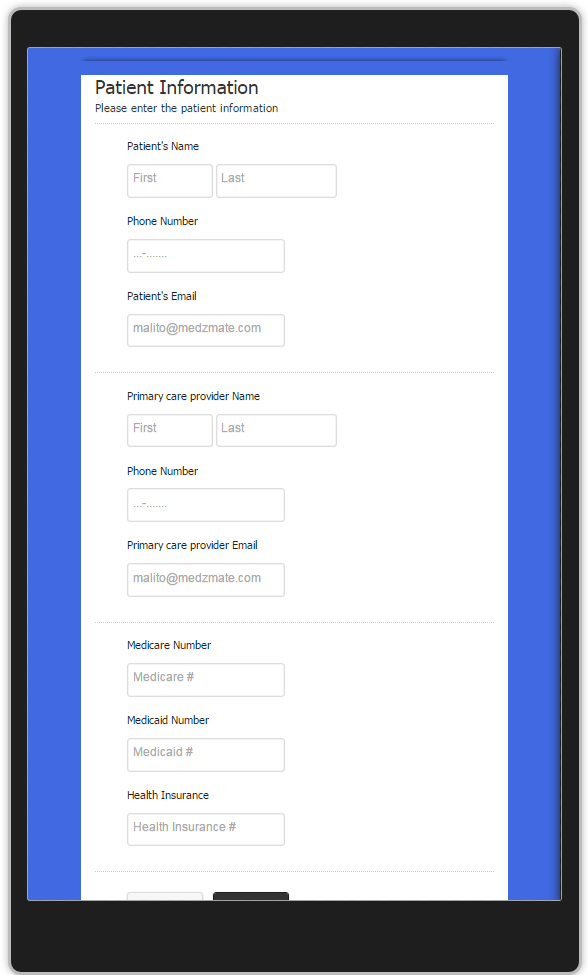


Figure **b6** - Patient Information form

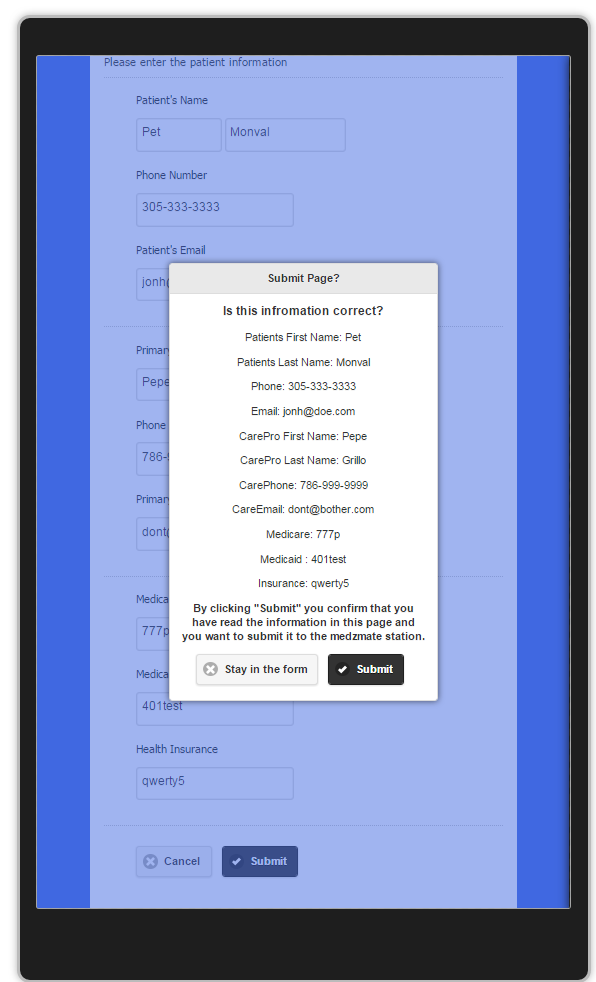


Figure **b7** - Patient information submit confirmation

## Appendix C - Sprint Review Reports

**Sprint 1 Report**

**Date:** September 11, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

Sprint 1 was technically what is known in SCRUM as Sprint 0. We worked on getting the environments ready, getting access to mingle and GitHub repository. Together with our product owner we gathered the user stories from which we created use cases and requirements to work on on the upcoming sprints.

**Sprint 2 Report**

**Date:** September 25, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

We were able to create the mobile application shell and two of the main screens of the product--the “add new medicine information” form and “loading deck”. We met with the computer engineering team and decided to go with the Raspberry PI as the Medzmate main hardware inside the dispensing station. We also did an extensive research on the technologies associated with Raspberry PI, development environments and tools.

**Sprint 3 Report**

**Date:** October 9, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

We did a lot of research on how to access the android file system to be able to save the data in the mobile device. We decided to use JSON files as format for data transmission and storage. We were able to save the JSON file locally in the mobile device containing the data from each straw using a unique straw identifier. Furthermore we were designing the scheduler algorithm.

**Sprint 4 Report**

**Date:** October 22, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

For the login page we added the CSS styles. We were able to keep a log of all the actions in the mobile application and store it. The installation of MongoDB in the Raspberry PI was successfully completed after a lot of research and drawbacks due to the PI cpu architecture. More research was done to support TCP/IP connection in both JavaScript and C++.

The research yielded that Node.js was the best option for the communication via TCP/IP. Based on these results the user login web service was implemented.

On the medicine detailed information form, a pop-up was created to confirm the data to be submitted. When the user confirms submission the data is saved otherwise the user will stay on the form to continue filling the data or correcting it.

A field was added to the login screen to enter the unique identifier of the Medzmate allowing the pharmacist to login into different stations.

JSON files containing the medicine information data is now being transmitted to the station using the previously implemented web service, and it is also saved within the station file system.

**Sprint 5 Report**

**Date:** November 6, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

The second Raspberry PI just arrived and we had to set up this second environment. We discovered that the scheduler code developed in Ubuntu was not compiling in Raspbian due to minor differences in system calls between the two systems. This roadblocks were overcome and we were able to compile and install the code in the Raspberry PI.

On the other hand the main screen to give the user access to settings, patient information and loading deck was created. The patient information form was created and all the associated data captured in the form was saved, and sent to the station in a JSON format. Also every action is being recorded in the log file.

**Sprint 6 Report**

**Date:** November 20, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

The settings screen form was completed. It contains the global configuration for the Medzmate Scheduler including the alarm types, sounds, and time intervals before and after the alarm is triggered, contact information etc. Data obtained from the form was transmitted and saved in the system. Upon data received the Medzmate process restarts and loads all the saved configuration from the hardware’s file system. All this information is displayed in the console for debugging purposes and the scheduler starts. Once the scheduler starts it throws various alerts depending on the configuration sent by the mobile application.

**Sprint 7 Report**

**Date:** December 4, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

This sprint was essentially to wrap up all the missing functionalities, do integration testing, fix bugs and refactor code. Log off functionality was completed using physical buttons of the mobile device. Also navigation was refactored to solve a bug found during the integration testing. All bugs found were fixed and all the displayed information was reviewed. Plenty of work was done on documentation too.

## Appendix D - Sprint Retrospective Reports

**Sprint 1 Retrospective**

**Date:** September 11, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

What went well?

* We gathered all the user stories and created the use cases and requirements.
* Environments were set up.

… and not so well?

* Our product owner was not familiar with SCRUM and the process to follow.
* We faced issues with mingle and GitHub accounts.
* The computer engineers of the team are still not defined.

Action items.

* Familiarized Prof. Dickson with the SCRUM methodology.

**Sprint 2 Retrospective**

**Date:** September 11, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

What went well?

* We decided on the hardware to be used on the station.
* The application shell was completed.

… and not so well?

* The hardware for the station was not available yet. The main cause was shipping delay.

Action items.

* None.

**Sprint 3 Retrospective**

**Date:** October 9, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

What went well?

* We were able to persist data on the mobile application.
* Completed all the planned tasks.

… and not so well?

* The hardware for the station was not available yet. The main cause was shipping delay.

Action items.

* Pedro will donate a personal Raspberry PI and set it up to work for the project.

**Sprint 4 Retrospective**

**Date:** September 11, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

What went well?

* Communication between mobile application and Station (Pedro’s Raspberry PI) was established.
* Completed all the planned tasks.

… and not so well?

* There hardware for the station was not available yet. The main cause was shipping delay.

Action items.

* After sprint code complete Pedro will test the station code in the Raspberry PI.

**Sprint 5 Retrospective**

**Date:** November 6, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

What went well?

* … and the Raspberry PI arrived!!! We had two stations to test in parallel all the implemented functionality.
* Completed all the planned tasks.

… and not so well?

* The scheduler code did not compile in Raspbian.

Action items.

* Compilation should be done instantly in Raspbian every time the scheduler code is modified.

**Sprint 6 Retrospective**

**Date:** November 20, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

What went well?

* We added “desired” features that were not committed in the initial planning.

… and not so well?

* We have not hear from the computer engineers.
* We don’t have a hardware prototype and there is functionality that works on physical buttons in the hardware.

Action items.

* We decided to emulate the hardware for this 1.0 version.

**Sprint 7 Retrospective**

**Date:** December 4, 2015

**Attendees:** Pedro Montero, Juan C. Correa

**Discussed Topics:**

What went well?

* Detected and fixed bugs.
* Prototype Medzmate 1.0 was a success.
* Product owner was very happy with the outcome of the project.

… and not so well?

* We regret not having a real hardware for this first release.

Action items.

* N/A

# References

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns Elements of Reusable Object-Oriented Software* (First ed.). Pearson Education.