DRUMRS
Design Document

Authors:
Kah Hin Lai
Amiel Hassan
Ross Faber
Chun F Chak

Group: 12

Document Revision History

Date	Version	Description	Author
11/13/2017	0.0	Initial draft	Kah Hin Lai
11/13/2017	0.0	Initial draft	Amiel Hassan
11/13/2017	0.0	Initial draft	Ross Faber
11/13/2017	0.0	Initial draft	Chun F Chak
11/20/2017	1.0	First Draft (Group)	Group 12

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide an insight on the designed development of the Dynamic Response to Urgent Maintenance Request System (DRUMRS).

1.2 System Overview

Dynamic Response to Urgent Maintenance Request System (DRUMRS) is a system that will aid various maintenance service in making requests, managing work orders, and managing request.

1.3 Design Objectives

We will be using object oriented design the design stage of our system. Object oriented design will provide security, usability as well as maintainability to our system. Object oriented design will provide abstraction of the implementations, making the system usable and safer from breaches to the system.

We will also group different components and functions of the system into different objects and implement separate methods and attributes for each one of them, making them easy to maintain and very intuitive.

1.4 References

- SRS Document
- Twilio API

1.5 Definitions, Acronyms, and Abbreviations

DRUMRS - Dynamic Response to Urgent Maintenance Request System
CCM(s) - Campus Community member(s)
SMS - Simple Message Service
RP - Request Point

2 Design Overview

2.1 Introduction

We will be using object-orientated design to implement the Dynamic Response to Urgent Maintenance Request System (DRUMRS). We will be using the client-server and data-centric architecture for our system design. The tools we use to produce this design document is Google Doc, Lucidchart, and Dia.

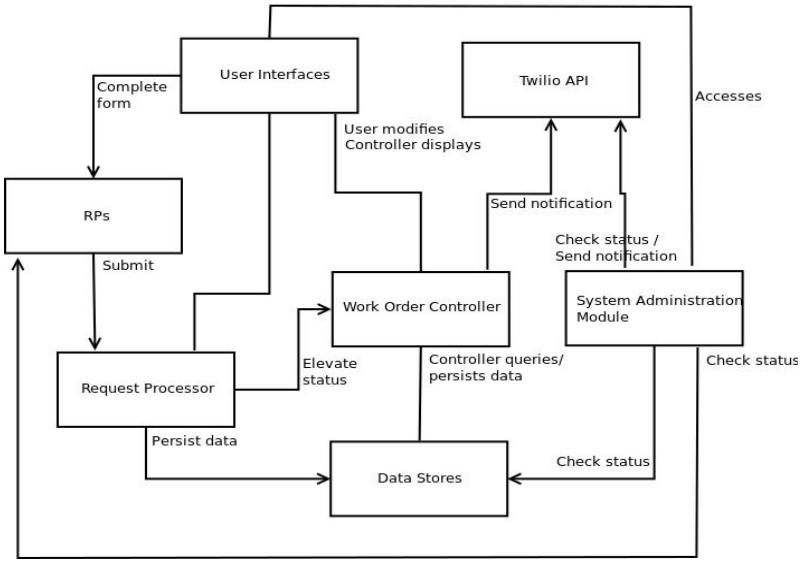
2.2 Environment Overview

DRUMRS will run on wherever current facilities management software is run. The system will use current databases, namely those that store work order and employee information. Physical RPs will be installed in all serviced buildings and communicate with the single instance of DRUMRS. DRUMRS will be executed on the maintenance company's dedicated server, and when each RP is switched on, they will establish a connection with DRUMRS. DRUMRS relies on the Twilio API for its notification system, and consequently, Twilio must be up and running for DRUMRS to work properly.

2.3 System Architecture

DRUMRS consists of seven major components: User Interfaces, RPs, Request Processor, Work Order Controller, Data Stores, System Administration Module, and the Twilio API. Most relationships between components are unidirectional. Most interactions between User Interfaces and other components are bidirectional because of the back-and-forth communication between the components; the only exception being the interaction between User Interfaces and RPs. This is because functionality associated with the RPs is unidirectional. Users won't receive feedback from an RP, but rather the Twilio API.

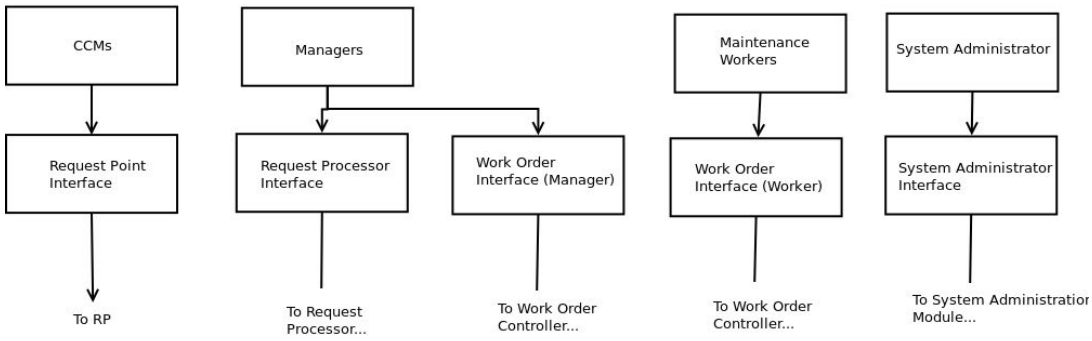
Top-Level System Structure of DRUMRS



2.3.1 User Interfaces Sub-system

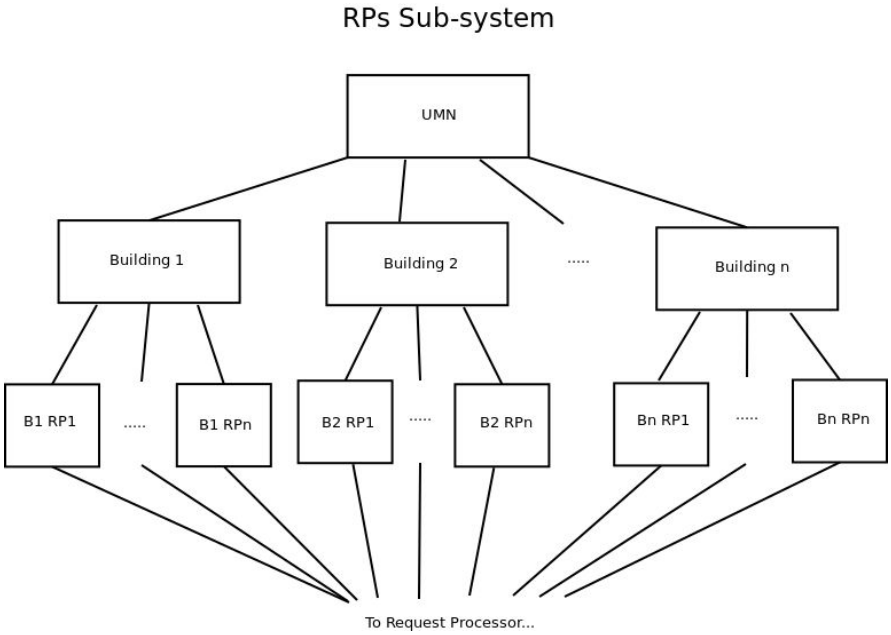
User Interfaces are separated into their own sub-system in order to more clearly show what users will use what components. The Work Order Interface is split into two different interfaces (Manager and Worker) because the interfaces work with almost the exact same data, but Managers have extended functionality. This will be covered more in depth later in the document.

User Interfaces Sub-system



2.3.2 Request Points Sub-system

Request pointss are separated into their own sub-system to illustrate more how the request point hardware will be implemented in serviced buildings as well as the potential multiplicity of request points.



2.3.3 Request Processor Sub-system

The Request Processor Sub-system is not given its own diagram because its purposes are clear through the top-level system diagram as well as the diagrams of the components it interacts with: Data Stores, Work Order Controller, RPs, and User Interfaces. The Request Processor receives requests from RPs and persists them in one of the Data Stores; managers use one of the User Interfaces to view requests and choose which to elevate to a work order; elevated work orders are sent to the Work Order Controller.

2.3.4 Work Order Controller Sub-system

The Work Order Controller is the heart of DRUMRS. Most of DRUMRS functionality involves this controller. To help illustrate this and how the controller is used, entities from other components are included in this diagram. The Work Order Displayer/Manipulator is the most computationally intensive part of DRUMRS since this is where the sorting, displaying, and modification of work orders takes place.

2.3.5 System Administration Module Sub-system

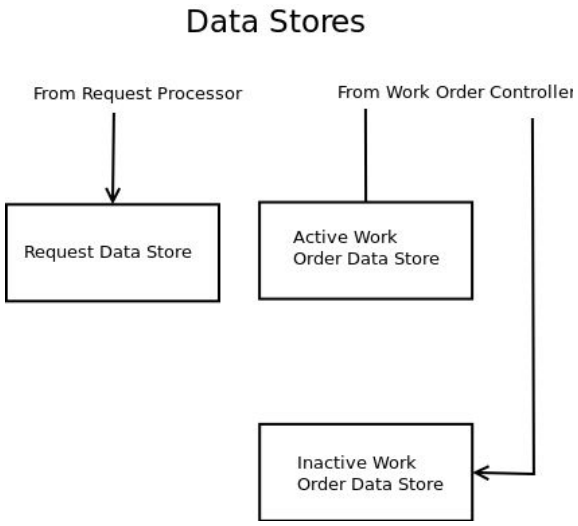
The System Administration Module's primary purpose is for maintaining DRUMRS components as well as carrying out functionality specific to system administrators. It is not given its own diagram because its purposes are clear through the top-level system diagram and its annotations.

2.3.6 Twilio API

The Twilio API is not given its own diagram because its purposes are clear through the top-level system diagram. The Twilio API is used as DRUMRS method of notification. Essentially, whenever a notification is required to be sent, the Twilio API is used.

2.3.7 Data Stores

DRUMRS uses a mixture of old and new data stores. New data stores are covered in slightly more depth later in the document. The inactive work order data store is generalized to a single entity as those details are out of our scope.



2.4 Constraints and Assumptions

1. DRUMRS relies on Short Message Service (SMS) for all notifications as outlined in the specifications document. SMS was chosen over other means of contact like email because recipients of DRUMRS notifications are more likely to have a ringer/buzzer on their phones for text messages over emails, complimenting the urgent nature of most DRUMRS notifications.
2. Requests will be allowed to file at every 30 minutes. This prevent there are multiple requests have been requested with the same issues. However, CCMs are still able to make a request if it is within the 30 minutes interval.
3. When a system check fails, a message will be sent to IT support team. This helps to maintain the system if anything is suspicious. The SMS text will contain a general issues topic.
4. A system Administrators will be able to grant superior access over maintainers, staff. This will allow them to be able to make change on top of maintainers and staff.

3 Interfaces and Data Stores

3.1 System Interfaces

3.1.1 Work Order Interface

This interface is used to access the work order database. It allows for a staff member to see all the available work orders. Staff will also be able to filter the work orders.

3.1.2 Request Interface

This interface is used to access the request database. It allows for a staff member to see all the available requests. Staff will also be able to filter the requests. Managers will be able to assign the requests to work orders.

3.1.3 Access Point Interface

This interface is used to have the physical access point system for CCM's communicate to the database about when a request is created.

3.1.4 System Administrator Notification Interface

This interface is used by the system administrators to send notifications about anything related to maintenance.

3.1.5 System Health Interface

This interface is used by the system administrators to monitor the health of the system

3.1.6 Twilio SMS message interface

This interface will communicate the requests to the staff members via SMS. It will communicate between the phone (environment) and the database (system).

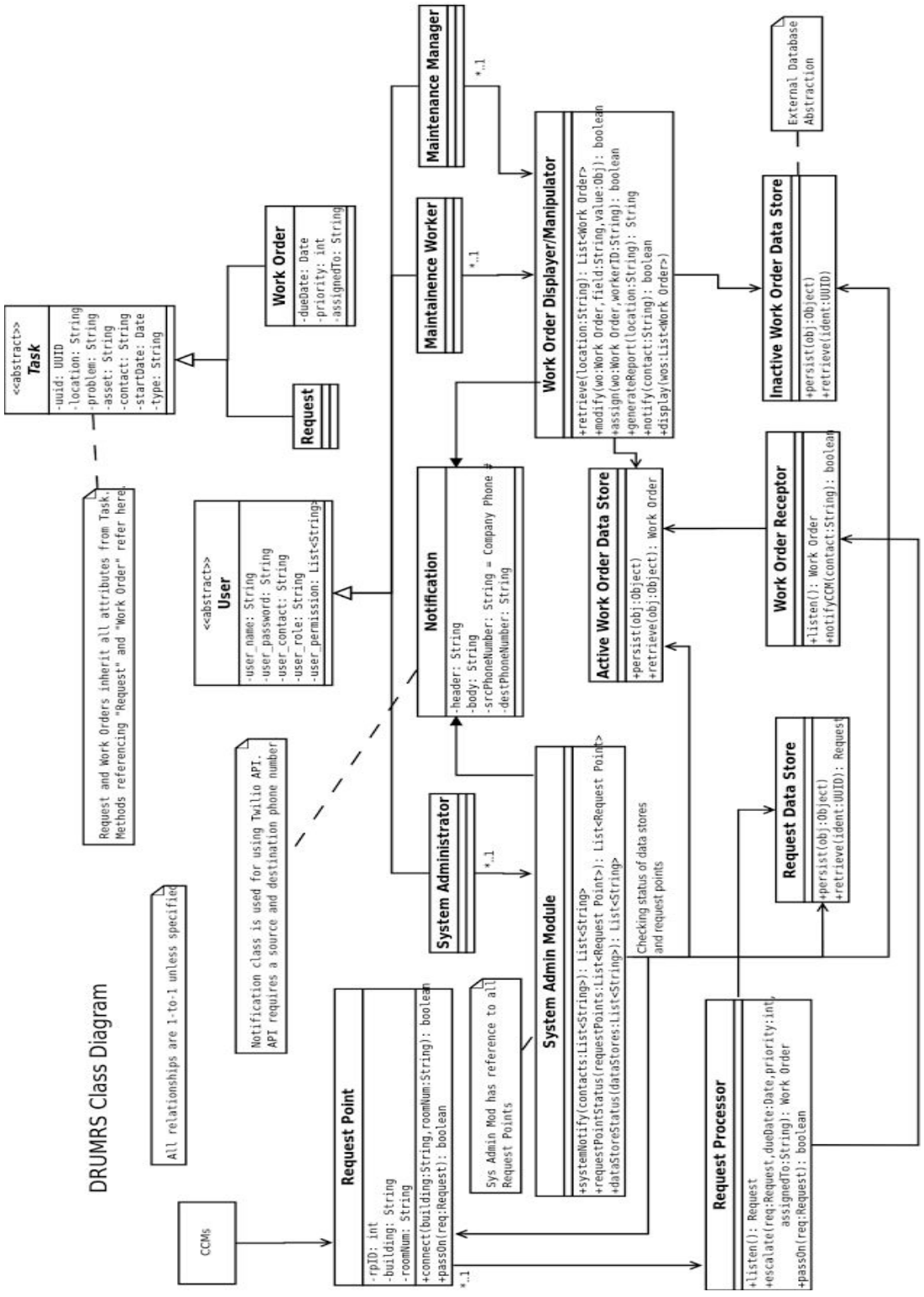
3.2 Data Stores

An active work orders data store will be created as part of DRUMRS. The purpose of this data store is to provide a local repository of all work orders that are created through DRUMRS. Inactive work orders will be stored wherever work orders are currently warehoused at UMN.

A request data store will be created as part of DRUMRS. This data store will maintain a log of all requests received by DRUMRS, regardless if the request was elevated to a work order. The purpose of this data store is for analyzing requests for trends and other statistical purposes.

4 Structural Design

4.1 Class Diagram



4.2 Class Descriptions

4.2.1 Class: User

- Purpose: To model what is a user for the system.
- Constraint: None
- Persistent: No

4.2.1.1 Attribute Description

- user_name
  - Description: The name of user
  - Type: string
  - Constraints: Not Null

- user_password
  - Description: The password required to log in
  - Type: string
  - Constraints: must be at least 8 characters, with Symbols,numbers,uncapitalized and capitalize characters

- user_contact
  - Description: Contact information of the user
  - Type: string
  - Constraints: Not Null

- user_role
  - Description: Role of user
  - Type: string
  - Constraints: Not Null

- user_permission
  - Description: Permission level of user
  - Type:String
  - Constraints: Not Null

4.2.2 Class: Request Point

- Purpose: Present the physical request points used by CCMs to create requests
- Constraints: None
- Persistent: No. Request Point make themselves known to the DRUMRS system on initialization

4.2.2.1 Attribute Description

- rpID
  - Description: Request point identifier
  - Type: int
  - Constraints: None

- building
    - Description: the name of the building
    - Type: String
    - Constraints: Must be serviced building

- roomNum
    - Description: the room number of the building
    - Type: String
    - Constraints: Must be valid room number within the building

4.2.2.2 Method Description

- connect(string building, string roomNum)
    - ReturnType: Boolean
    - Parameters: building- the name of the building, roomNum - the room number
    - Return Value: success or failure
    - Precondition: DRUMRS is initialized and running
    - Post-condition: DRUMRS is aware of request point
    - Attributes read/used: rpID, buidling, roomNum
    - Methods called: None
    - Processing logic: Request point attempts to establish a connection with DRUMRS. Once established, request point sends rpID, building, and roomNum so DRUMRS knows location and identifier of request point.
    - TestCase: Called connect(Keller Hall, 3-310). Expected Output: success

- passOn(Request Req)
    - ReturnType: Boolean
    - Parameters: req- the request will be passed to DRUMRS
    - Return Value: success or failure
    - Precondition: DRUMRS is intialzied and running and connection has been established
    - Post-condition: DRUMRS Request Processor received Request
    - Attributes read/used: None
    - Methods called: None, but connect() must have succeeded first
    - Processing logic: Method receives a Request object and sends it to the DRUMRS Request Processor
    - TestCase: Called passOn(req001). Expected Output: success

4.2.3 Class: Notification

- Purpose: Encapsulating system notifications and interfacing with Twilio API
- Constraints: See Twilio API
- Persistent: No

4.2.3.1 Attribute Description

- header
    - Description: Subject line of notification
    - Type: String
    - Constraints: None

- body
    - Description: Notification body

- Type: String
- Constraints: None

- srcPhoneNumber
  - Description: Company's phone number/designated phone number for DRUMRS notifications. Twilio requirement
  - Type: String
  - Constraints: Must be a valid 10 digit phone number

- srcPhoneNumber
  - Description: Phone number of notification recipients
  - Type: String
  - Constraints: Must be a valid 10 digit phone number

4.2.4 Class: Task

- Purpose: Abstractly represent a maintenance task.
- Constraints: Identifier is not modifiable
- Persistent: Yes (storage is dependent on implementing class)

4.2.4.1 Attribute Description

- uuid
  - Description: UUID
  - Type: Universally unique identifier
  - Constraints: Unique

- location
  - Description: Name of building
  - Type: String
  - Constraints: Limited to building maintenance currently services

- problem
  - Description: short description of task
  - Type: String
  - Constraints: None

- asset
  - Description: Asset involved in task
  - Type: String
  - Constraints: None

- contact
  - Description: Phone number of person who initiated the task
  - Type: String
  - Constraints: 10 digit number

- StartDate
  - Description: the date the task was created
  - Type: Date
  - Constraints: None

- type
  - Description: the category oftask
  - Type: String

- Constraints: None

4.2.5 Class: Request

- Purpose: Concrete implementation of Task abstract class. For representing service requests
- Constraints: None
- Persistent: Yes. Stored in Request Data Store
-

4.2.5.1 Attribute Description

Request implements all attributes described in the Task class.

4.2.6 Class: Work Order

- Purpose: Concrete implementation of Task abstract class. For representing work orders.
- Constraints: None
- Persistent: Yes. Initially stored in Active Work Order Data Store, eventually moved to Inactive Work Order Data Store.

4.2.6.1 Attribute Description

Work Order implements all attributes described in the Task clas and additionally has the following attributes.

- DueDate
  - Description: Deadline of work order
  - Type: Date
  - Constraints: Must be after startDate

- priority
  - Description: Priority of work order as defined by maintenance criteria (higher number = higher priority)
  - Type: Int
  - Constraints: None

- AssignedTo
  - Description: ID number of worker in charge of fulfilling work order
  - Type: String
  - Constraints: Currently employed worker

4.2.7 Class: Request Processor

- Purpose: Receive requests and allow managers to escalate their status to work orders
- Constraints: Only one instance can exist
- Persistent: No. Initialized with DRUMRS

4.2.7.1 Method Description

- listen()
  - Return Type: Request
  - Parameters: None
  - Return Value: Request received
  - Precondition: DRUMRS is initialized
  - Post-condition: Successfully received Request
  - Attributes read/used: None
  - Methods called: None

- Processing logic: Request Processor waits and hangs until it receives a Request from a Request Point. How connection is established depends on implementation (ex. TCP vs. UDP)
- TestCase: Called listen(). Expected Output is “Request received”.

- escalate(Request req, Date dueDate, int priority, String assignedTo)
  - Return Type: Work Order
  - Parameters: req-Request to be escalated, dueDate- Deadline of new work order, priority- priority of new work order, assignedTo - Assigned worker(may be null)
  - Return Value: Newly created work order
  - Precondition: Request has been received and manager knows additional work order fields
  - Post-condition: Work order created
  - Attributes read/used: None
  - Methods called: None, but work typically be called after a successful listen()
  - Processing logic: Create a work order given specified parameters.

- passOn(Request req)
  - Return Type: boolean
  - Parameters: req- request will be passed to Work order receptor
  - Return Value: Success or failure
  - Precondition: DRUMRS is initialized(namely Work Order Receptor)
  - Post-condition: Work Order sent
  - Attributes read/used: None
  - Methods called: None, but would typically be called after escalate()
  - Processing logic: Depending on network implementation, Request Processor may either wait to establish a connection with Work Order Receptor or just push out Work Order into network.
  - TestCase: Called passon(req001). Expected Output is success.

4.2.8 Class: Request Data Store

- Purpose: Store all request received by DRUMRS
- Constraints: None
- Persistents: Yes

4.2.8.1 Method Description

- persist(Object obj)
  - Return Type: None
  - Parameters: obj- request to be stored
  - Return Value:None
  - Precondition: Data store is accessible
  - Post-condition: Request is stored in data store
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Store request in data store

- retrieve(UUID ident)
  - Return Type: Request
  - Parameters: ident - UUID for desired Request
  - Return Value: Request or error
  - Precondition: Data Store is accessible

- Post-condition: Request or error returned
- Attributes read/used: None
- Methods called: None
- Processing logic: Find request, return if applicable, otherwise error.

4.2.9 Class: Active Work Order Data Store

- Purpose: Data store for currently active, incomplete work orders
- Constraints: None
- Persistent: Yes

4.2.9.1 Method Description

- persist(Object obj)
  - Return Type: None
  - Parameters: obj- Work order to be stored
  - Return Value:None
  - Precondition: Data store is accessible
  - Post-condition: Work Order is stored in data store
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Store Work Order in data store

- retrieve(UUID ident)
  - Return Type: Work Order
  - Parameters: ident - UUID for desired Work order
  - Return Value: Work Order or error
  - Precondition: Data Store is accessible
  - Post-condition: Work order or error returned
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Find Work Order, return if applicable, otherwise error.

4.2.10 Class: Inactive Work Order Data Store

- Purpose: Legacy Data Store(prior to DRUMRS). Store Work Order handler DRUMRS
- Constraints: None
- Persistents: Yes

4.2.10.1 Method Description

- persist(Object obj)
  - Return Type: None
  - Parameters: obj- Work order to be stored
  - Return Value:None
  - Precondition: Data store is accessible
  - Post-condition: Work Order is stored in data store
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Store Work Order in data store

- retrieve(UUID ident)
  - Return Type: Work Order
  - Parameters: ident - UUID for desired Work order

- Return Value: Work Order or error
- Precondition: Data Store is accessible
- Post-condition: Work order or error returned
- Attributes read/used: None
- Methods called: None
- Processing logic: Find Work Order, return if applicable, otherwise error.

4.2.11 Class: System Admin Module

- Purpose: Prove system administrators with required functionality.
- Constraints: Only one instance can exist
- Persistent: No, initialized with DRUMRS

4.2.11.1 Method Description

- systemNotify(List<String> contacts)
  - Return Type: List<String>
  - Parameters: contacts- List of phone numbers to contact
  - Return Value: List of phone numbers of notified people
  - Precondition: contacts is not empty, Twilio API is accessible
  - Post-condition:All, some or no contacts are notified via SMS (thru Twilio)
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Loop through all contacts to create Notification objects, and use those objects to use the Twilio to notify necessary personnel.
  - TestCase: Called SystemNotify(contacts). Expected Output: a list of phone numbers.
- dataStoreStatus(List<String> dataStores)
  - Return Type: List<String>
  - Parameters: dataStores - List of data stores to query
  - Return Value: List of successfully quieried data stores
  - Precondition: System Admin Module has a list of all necessary data stores
  - Post-condition: All, some or no data stores are running
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Loop through all data stores and attempt to ask them a trivial query. Return a list of all data stores that successfully respond.
  - TestCase: Called dataStoreStatus(dataList). Expect Ouput is a list of data stores.

- requestPointsStatus(List<Request Point> requestPoints)
  - Return Type: List<Request Point>
  - Parameters: requestPoints - List of request points to ping
  - Return Value: List of successfully pinged Request Points
  - Precondition: System Admin Module has been keeping track of established request points
  - Post-condition: All, some or no request points are successfully connect.
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Loop through all request points and attempt to ping them. Return a list of all request points that ping back.
  - TestCase: Called requestPointStatus(rp). Expect Ouput is a list of successful rp.

4.2.12 Class: Work Order Receptor

- Purpose: Receive work orders, persist them , and notify CCMs
- Constraints: Only one instance can exist
- Persistents: No. Initialized with DRUMRS

4.2.12.1 Method Description

- listen()
  - Return Type: Work Order
  - Parameters: None
  - Return Value: Received work order
  - Precondition:DRUMRS is initialized
  - Post-condition: SUccessfully received Work Order
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: WOrk Order Receptor waits and hangs until it recevies a work from the Request Processor
  - TestCase: Called listen(0. Expected Output is “Received work order”

- notifyCCM(string contact)
  - Return Type: boolean
  - Parameters:contact- phone number of CCM to notify. Retrieved from work order
  - Return Value: success or failure
  - Precondition: Work order receptor has received a work order
  - Post-condition: CCM notified via SMS
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Extract phone number contact from received Work Order and construct a Notification to send using the Twilio API.
  - TestCase: Called notifyCCM(Name). Expected Output is success.

4.2.13 Class: Work Order DIsplayer/Manipulator

- Purpose: Display and manipulate work orders. Construct aggregate work order reports
- Constraints: Only one instance can exist
- Persistent: No. Initialized with DRUMRS

4.2.13.1 Method Description

- modify(WorkOrder wo, string field, Object value)
  - Return Type: boolean
  - Parameters: wo- Work Order to be modified, field = Work Order field to be modified, value - Value to store in field
  - Return Value: success or failure
  - Precondition: Work Order exists in Active Work Order Data Store
  - Post-condition: Work Order is modified in Active Work Order Data Sotre
  - Attributes read/used: None
  - Methods called: None
  - Processing logic: Modify given Work Order using given field and value, update Active Work Order Data Store

- retrieve(String location)
    - Return Type: List<Work Order>
    - Parameters: location
    - Return Value: List of all active work orders associated with location
    - Precondition: Location is valid, Active Work Order Data Store
    - Post-condition: Work Order is modified in Active Work Order Data Stores
    - Attributes read/used: None
    - Methods called: None
    - Processing logic: Modify given Work Order using given field and value, update Active Work Order Data Store

- assign(WorkOrder wo, String workerID)
    - Return Type: boolean
    - Parameters: wo- Work Order to be modified, workerID- ID of worker to assign WO to
    - Return Value: success or failure
    - Precondition: Work Order exists in Active Work Order Data Store, workerID is a real employed worker
    - Post-condition: Work Order is modified in Active Work Order Data Store
    - Attributes read/used: None
    - Methods called: None
    - Processing logic: Modify given Work Order using given workerID, update Active Work Order Data Store.
    - TestCase: Called assign(wo001, smith123). Expected Output: success

- generateReport(String location)
    - Return Type: String
    - Parameters: location - building to generate work order report for
    - Return Value: Aggregate work order report
    - Precondition: location is a serviced UMN building
    - Post-condition: Work Orders unchanged
    - Attributes read/used: None
    - Methods called: retrieve
    - Processing logic: retrieve() method used to get work orders associated with location but instead of return a list of work orders, returns a formatted string version of all the work orders.

- notify(string contact)
    - Return Type: boolean
    - Parameters: contact
    - Return Value: success or failure
    - Precondition: Twilio API is accessible
    - Post-condition: contact is notified
    - Attributes read/used: None
    - Methods called: None
    - Processing logic: Construct a Notification using the provide contact number and send it using Twilio API.
    - TestCase: Called notify(Smith). Expect Output is success.

- display(List<WorkOrder> wos)
    - Return Type: None. DIsplayed at User Interface
    - Parameters: wos - List of work orders to display
    - Return Value: None. Displayed at User Interface
    - Precondition: wos is a list of valid Work Orders

- Post-condition: Work Orders displayed on user interface
- Attributes read/used: None
- Methods called:NOne
- Processing logic: Similar to generateReport(). Creates a formatted version of provided Work Orders to dispaly on USer Interface.
- TestCase:

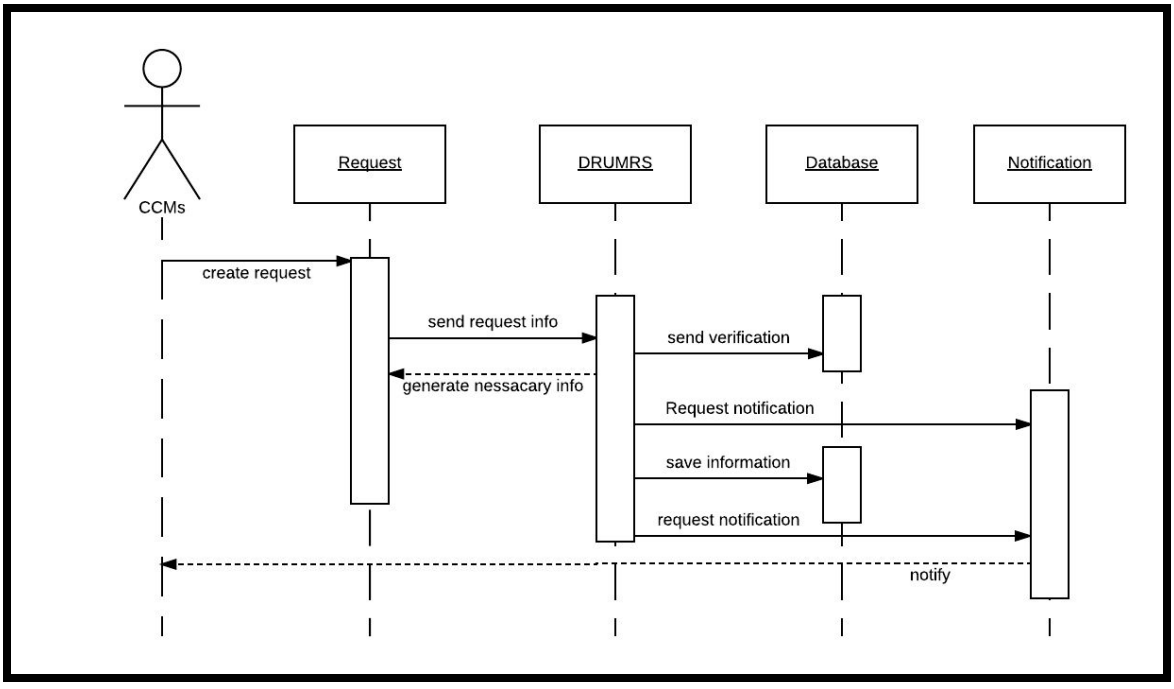
5 Dynamic Model

5.1 Scenarios

5.1.1 Make a request

Description: CCM makes a request in a location.
Steps:

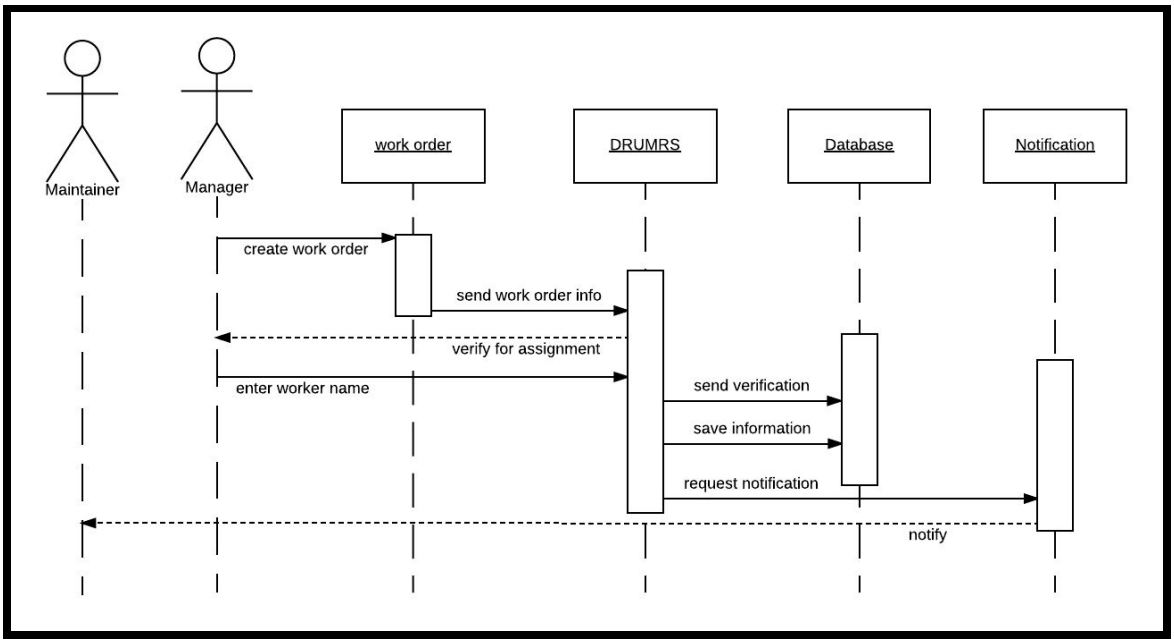
1. CCM enters all necessary details requested for making a request.
2. CCM submits the request form
3. The request will automatically be sent to DRUMRS to verify if there are no conflict request that was previously placed.
4. The system automatically generate necessary details for the request (for example, time or location).
5. System send to notification to the CCM
6. The system save request to database.



5.1.2 Create and assign a work order

Description: Maintenance staff manager create and assign work order.
Steps:

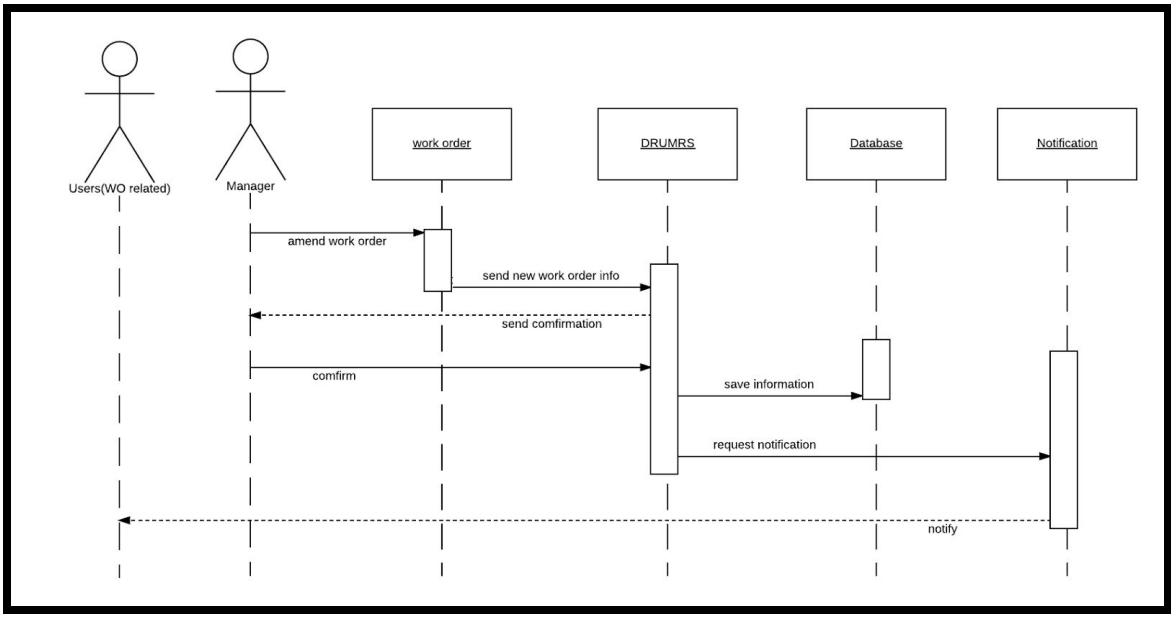
1. Maintenance staff manager selects request that don't have corresponding work order.
2. Work order is created by system.
3. System verify if manager want to assign the work order
4. Manager enter the worker name to assign the work order to the worker.
5. System verify if work order is assignable to the worker.
6. System assign work order to the worker.
7. System notifies worker.



5.1.3 Amending a work order

Description: Manager amends a work order
Steps:

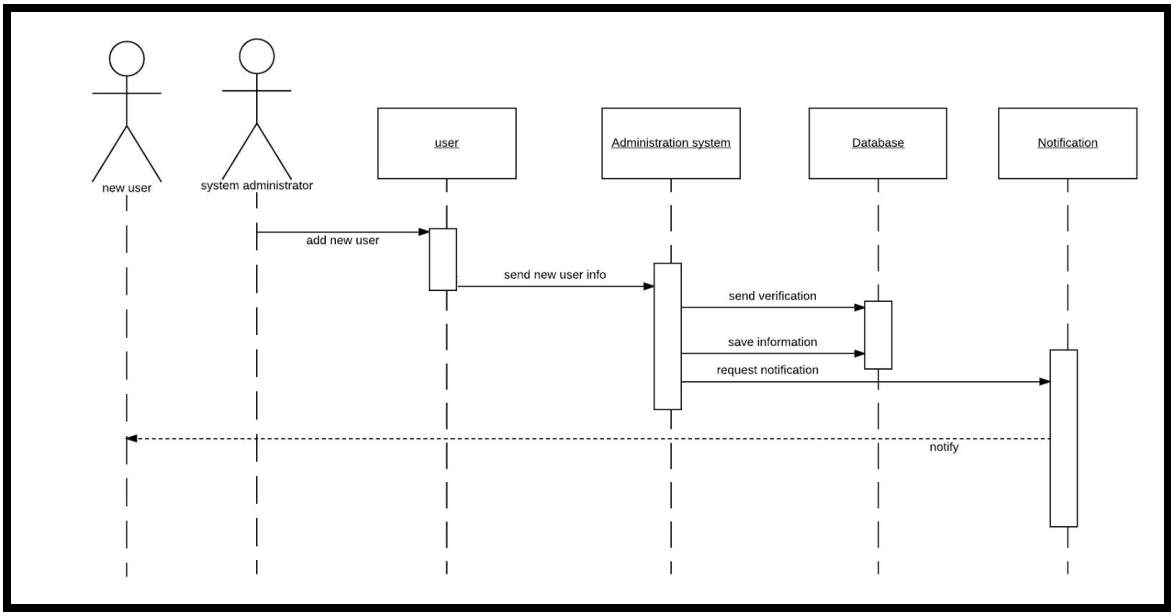
1. Maintenance staff Manager selects a work order
2. Manager make changes to the work order
3. Manager confirms the changes make
4. The changes to the work order is saved
5. The system notifies other user who is involved in this work order about the changes.
6. The system records the name of the user who amend the work order and also the time when the order is amended.



5.1.4 User Administration (Add user, Remove user, update user permission and etc)

Description: System Administrator adds new user to the system.
Steps:

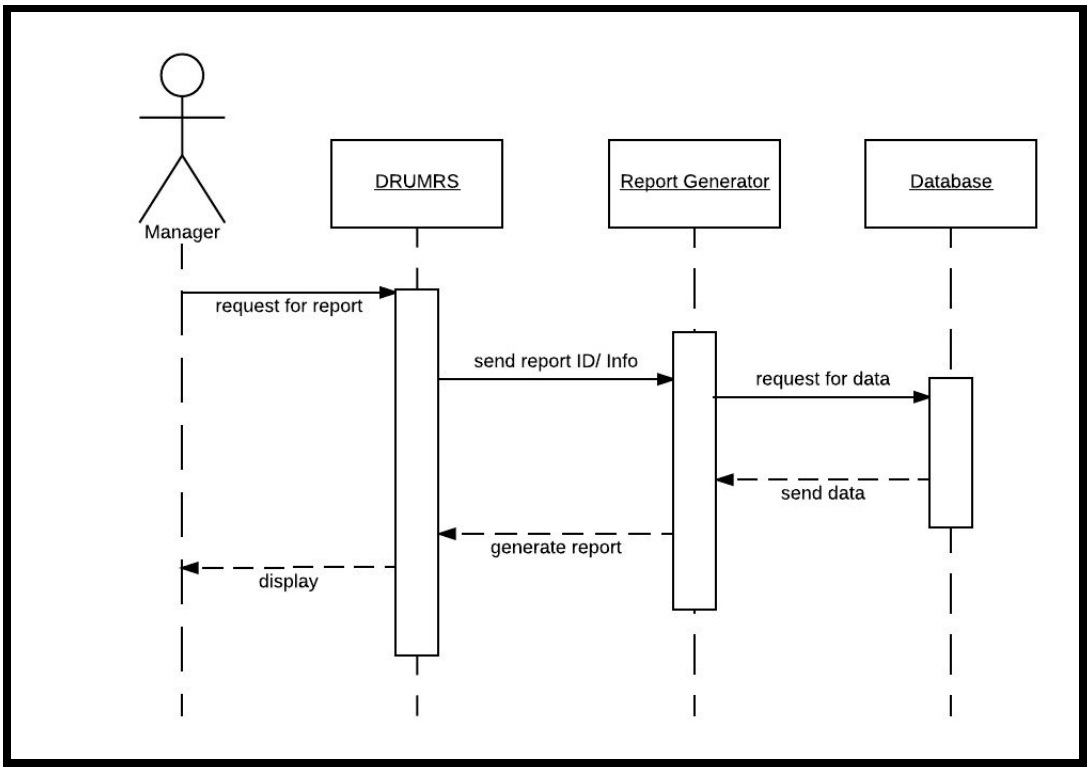
1. System Administrator enter necessary information of the new user.
2. System verify if there is any conflicts in the database.
3. New user is added to the system.
4. System notifies new user



5.1.5 Generate Report

Description: Manager generate report
Steps:

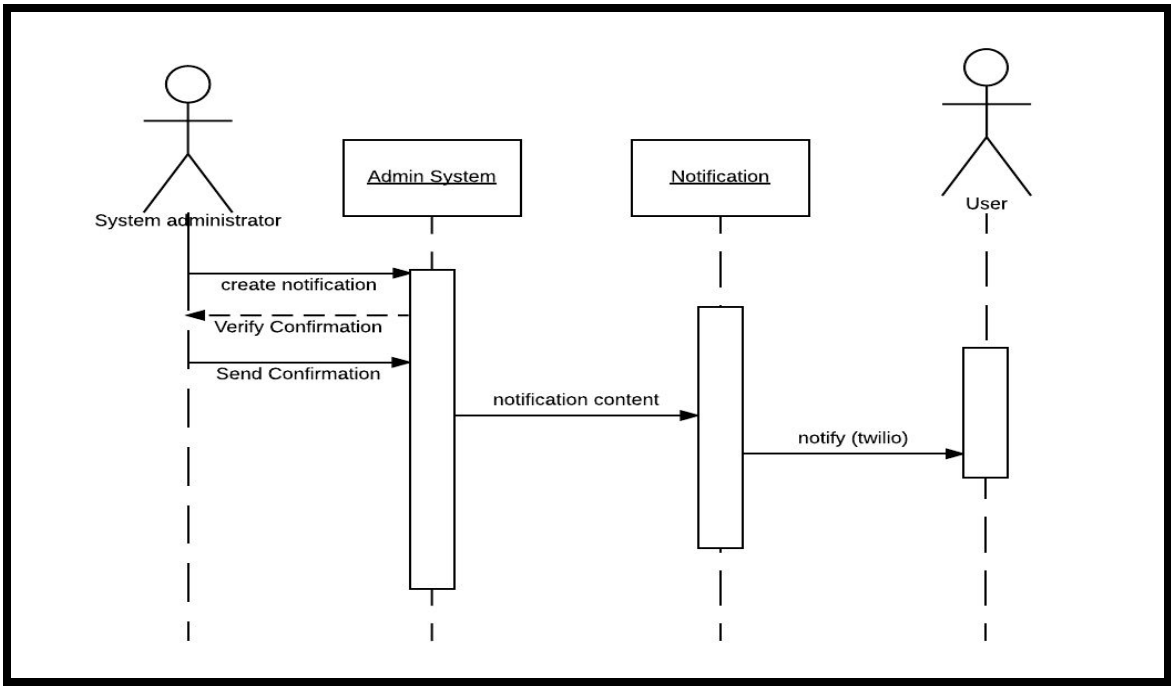
1. Manager enter necessary information to request for report.
2. Manager request for report
3. System use report generator to generate report with input
4. System display report to manager.



5.1.6 System Administrator Notification

Description: System Administrator send Notification to selected user
Steps:

1. System Administrator enter necessary information
2. System Administrator create notification
3. System verify confirmation from System Administrator
4. System Administrator sends confirmation
5. Notification is sent to User via twilio



6 Non-functional requirements

6.1 Non-functional Overview

- Security Requirement
  - We will be using object oriented design for the software and this will provide essential security for the software. With object oriented design, an object has restrictions from accessing another object by not giving the first object a reference to the second.
- Usability Requirement
  - We use object oriented design to have a distinct separation of different classes and their usage, giving a clear meaning to each part and their functionalities to users of this system. Furthermore, abstraction is a nice feature provided by object oriented design for a better usage experience for the users.
- Maintainability Requirement
  - We expect to meet the maintainability requirement by keeping certain implementation and design standards for all the classes in the system, making it easy to maintain and to make changes.

6.2 Requirements from SRS

3.3.1: Work order attributes listed in requirement 3.3.1 are maintained by ensuring fields are entered at creation of request task, and maintained when escalated to a work order and persisted in DRUMRS data store.

3.4: History of work order modifications are maintained by DRUMRS work order data store.

3.5: Permanence of work orders is enforced by integrity constraints placed on DRUMRS work order data store.

5.1: DRUMRS user permissions are maintained by a mandatory access control system similar to those used in a DBMS where each class of user is given a set of functionality.

7 Supplementary Documentation

- SRS Document
- Twilio API - https://www.twilio.com/