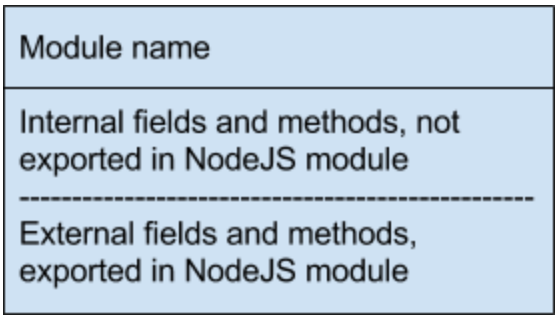
System Design Document

Mobile Student Intranet

Table of Contents

Notes

- DAO = Database Access Object
- FCM = Firebase Cloud Messaging
- Channel refers to a top level document in the icube collection in the server database
- Channel ID refers to a channel._id field
- Area refers to a channel.areas[i] object field where i is a valid index
- Area ID refers to channel.areas[i].id field where is i is a valid index
- Subject refers to a channel.areas[i].subjects[j] object field where i and j are valid indexes
- Subject ID refers to a channel.areas[i].subjects[j].id field where i and j are valid indexes
- Topic key refers to a channel.areas[i].subjects[j].topic_key field where i and j are valid indexes
- Module legend:
    - Attributes in bold text are other modules
    - Methods in gray text are not yet implemented
    - Dashed line as shown below indicates what is and is not exported

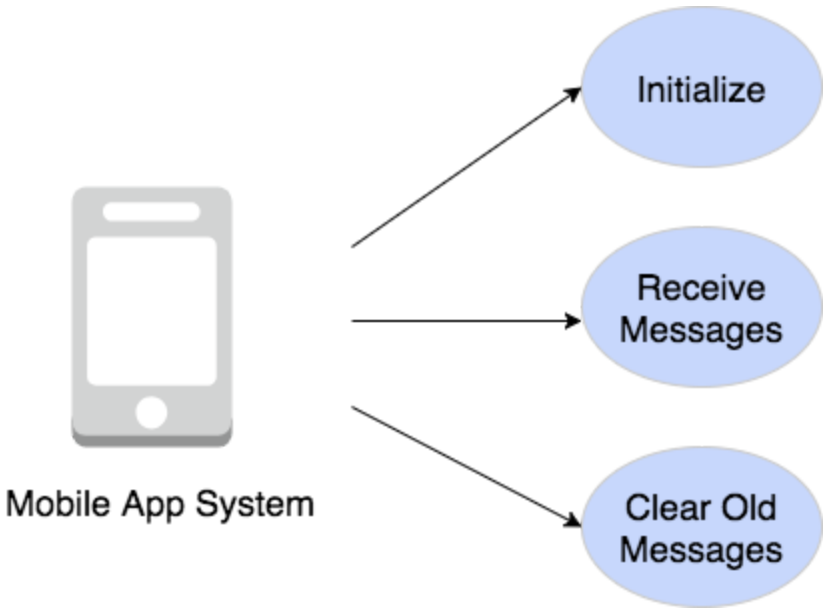


Software Design

Use Case Flow

Mobile App Use Cases

Mobile App System



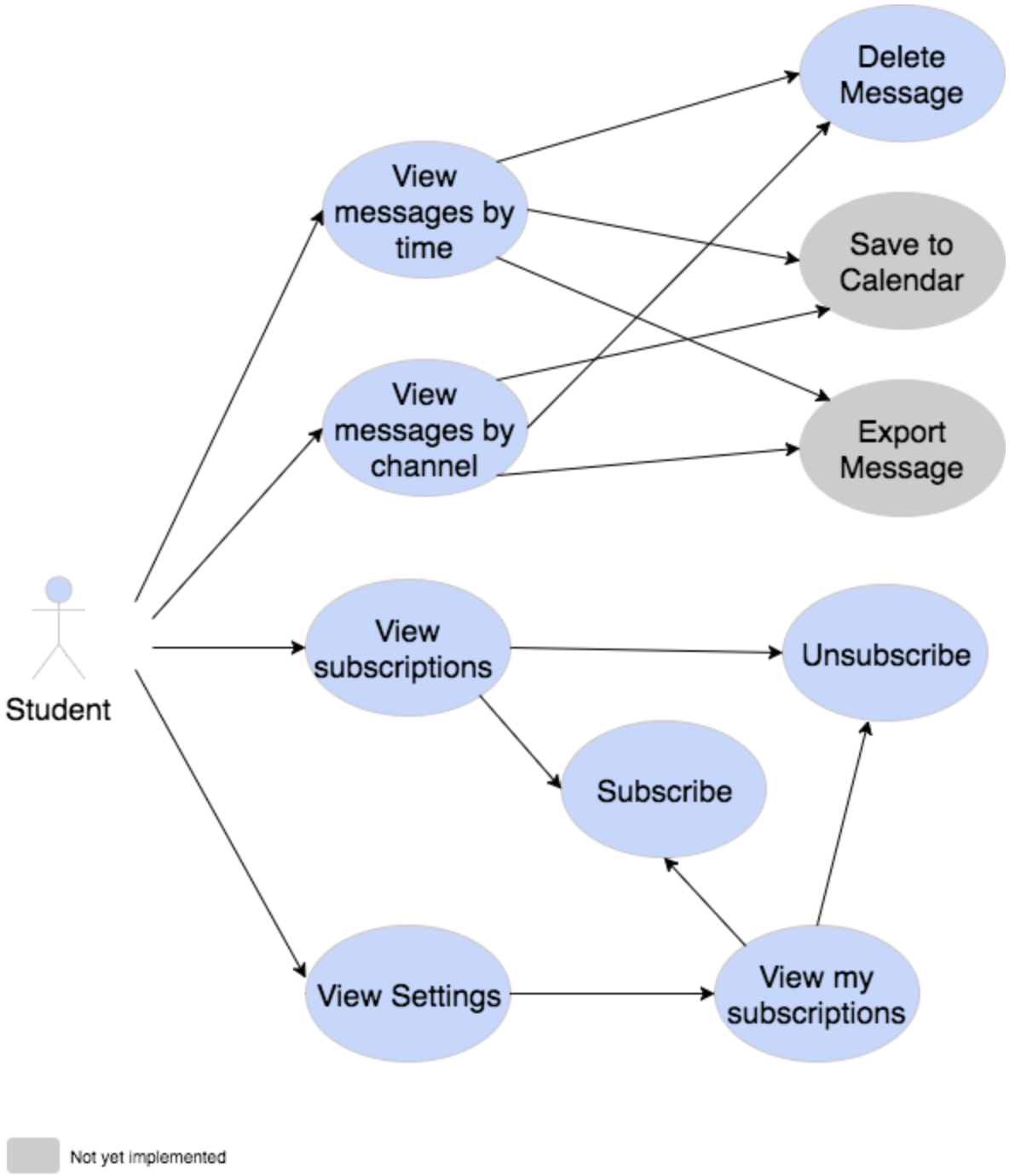
Initialize	
Summary	Sets up the app to be able to receive messages and subscribes to forced and recommend topics
Pre-Conditions	• First time running app • Network connection
Flow	1. Request permissions if on iOS device 2. Subscribe to all topics (channel/area/subject combinations) that are currently in the information cube with level “Forced” or “Recommended”
Post-Conditions	If successful, the app will be able to start receiving messages and will

	be subscribed to the current set of Forced and Recommended topics. If not, based on the current implementation the app should be uninstalled and reinstalled to reattempt this process (enhancement pending for reattempt to be done without reinstallation).

Receive Messages	
Summary	Processes and displays information and alert messages from subscribed topics
Pre-Conditions	● Permissions have been granted (if running on iOS device) ● App has been initialized ● App is subscribed to topic of message ● App is running in the background (if running on iOS device)
Flow	● Push notification displayed if alert message ● Message saved to app's local database ● Message added to app's message views
Post-Conditions	If pre-conditions met, the message will be stored and viewable in the app. Otherwise the message will be dropped in the current implementation (enhancement pending).

Clear Old Messages	
Summary	Clears messages that are older than a week from the local database
Pre-Conditions	
Flow	1. When the app starts it will clear messages older than a week from the local database
Post-Conditions	The local database will only keep messages from the last seven days of the week

Student



View messages by time

Summary	View and interact with messages sorted by most recent
Pre-Conditions	• Some messages must be in the local database to be able to see certain items in the “Flow” section below

Flow	1. The system loads the messages from the local database sorted by timestamp
	2. The system retrieves information cube for each message
	3. The user can
	  a. Toggle view between list and detailed
	  b. Expand/collapse individual messages
	  c. See visual indication for alert messages
	  d. See visual indication for date within messages
	  e. See which channel, area, and subject messages were sent from
	  f. See dynamic header for date of messages currently viewing
	  g. See dynamic header for weekday of messages currently viewing

Post-Conditions	

View messages by channel	
Summary	View and interact with messages filtered by channel
Pre-Conditions	• Some messages must be in the local database to be able to see certain items in the “Flow” section below

Flow	1. The system loads the messages from the local database filtered by channel
	2. The system retrieves information cube for each message
	3. The user can
	  a. Toggle view between list and detailed
	  b. Expand/collapse individual messages
	  c. See visual indication for alert messages
	  d. See visual indication for date within messages
	  e. See which channel, area, and subject messages were sent from
	  f. See dynamic header for date of messages currently viewing
	  g. Change channel to filter messages by

Post-Conditions	

Delete message	
Summary	Delete messages from app
Pre-Conditions	• There must be at least one message in the local database to be able to delete a message
Flow	1. User swipes left on message 2. User views deletion prompt a. Press yes to confirm and delete b. Press no to cancel 3. If user confirms, the system removes the message from the local database and the display
Post-Conditions	Message is removed from both the local database and display

Save message to calendar (not implemented, later enhancement)	
Summary	This will allow users to save messages with dates/events to their calendar apps
Pre-Conditions	Not defined yet
Flow	Not defined yet
Post-Conditions	Not defined yet

Export message (not implemented, later enhancement)	
Summary	This will allow users to save messages (i.e. provides a way to keep a message before it becomes old enough to be purged from the local database)
Pre-Conditions	Not defined yet
Flow	Not defined yet
Post-Conditions	Not defined yet

View subscriptions	
Summary	View available subscriptions by information cube hierarchy
Pre-Conditions	• Network connection when app is opened
Flow	1. The system loads the information cube from the local database if already loaded or from the web-based REST endpoint 2. The user can a. Change channel to filter subscriptions by b. Expand/collapse areas c. See if subject has forced subscription level and whether it is currently subscribed to d. Subscribe to subjects (see corresponding use case) e. Unsubscribe from subjects (see corresponding use case)
Post-Conditions	

View Settings	
Summary	View available settings options
Pre-Conditions	
Flow	1. The user can a. Navigate to specific setting options
Post-Conditions	If user selects a setting option, the user will be navigated to that view

View My Subscriptions	
Summary	View all subscriptions currently subscribed to
Pre-Conditions	• Network connection
Flow	1. The system uses a FCM API to retrieve the topics the user is currently subscribed to 2. The user can a. See all subjects currently subscribed to b. Subscribe to subjects (see corresponding use case) c. Unsubscribe from subjects (see corresponding use case)

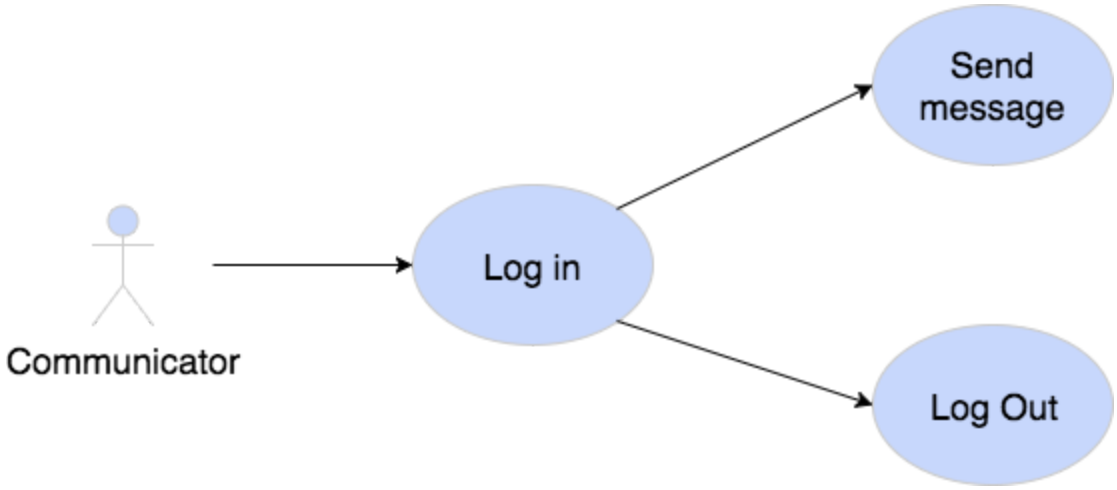
Post-Conditions	

Subscribe	
Summary	Subscribe to a subscription
Pre-Conditions	● Network connection
Flow	1. The system will use FCM API to add the user to the topic
Post-Conditions	The user will subscribed to the selected topic

Unsubscribe	
Summary	Unsubscribe from a subscription
Pre-Conditions	● Network connection
Flow	1. The system will use FCM API to remove the user from the topic
Post-Conditions	The user will be unsubscribed from the selected topic

Message Hub Use Cases

Communicator



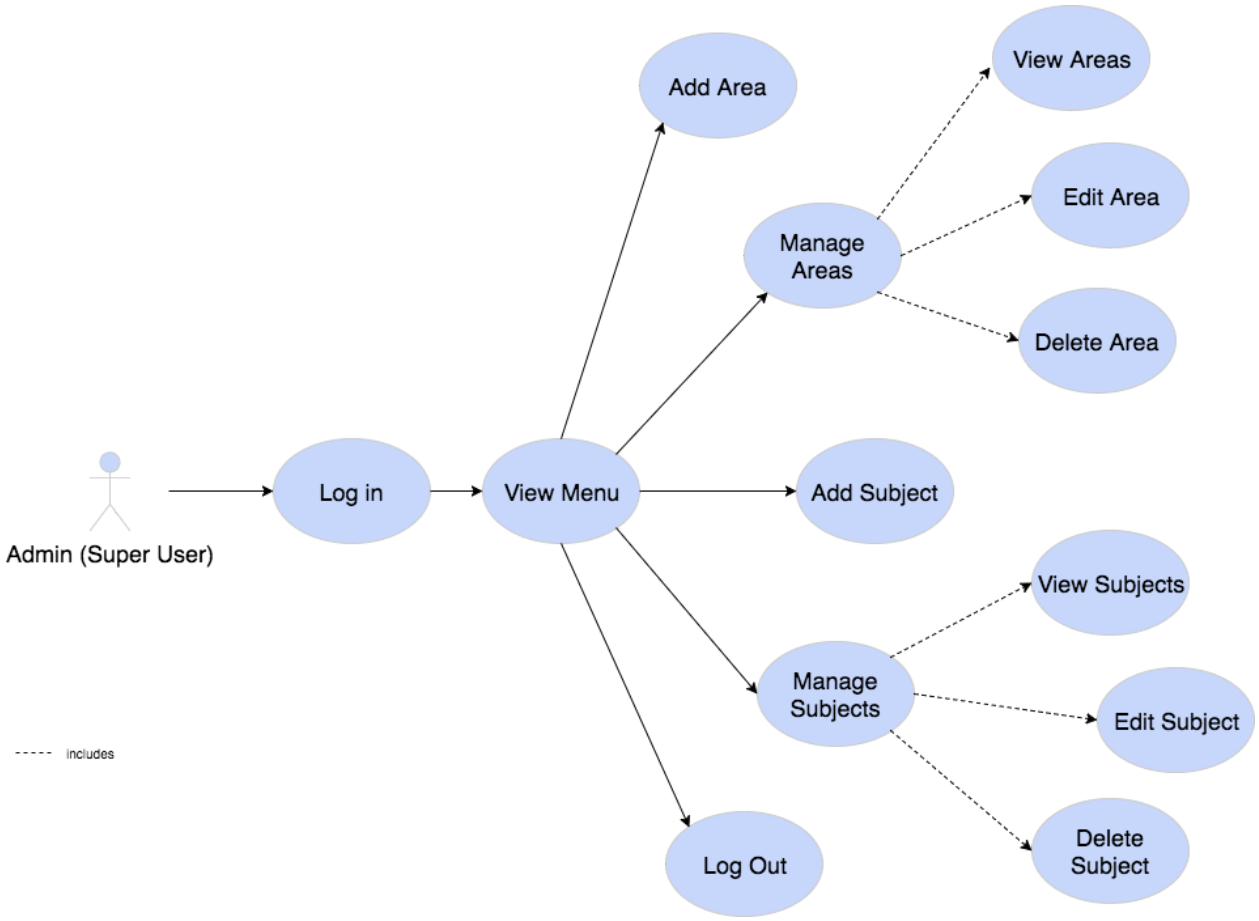
Log In	
Summary	A communicator can log in to access the portal
Pre-Conditions	
Flow	1. A cookie for the user of “Communicator” will be set 2. The communicator will have access to the communicator portion of the portal
Post-Conditions	The communicator will be logged in and the user cookie will be set

Send Message	
Summary	A communicator can send alert or information messages to a topic pertaining to a part of the information cube
Pre-Conditions	• The communicator should be logged in
Flow	1. The system loads the information cube 2. The communicator selects a channel 3. The communicator selects an underlying area 4. The communicator selects an underlying subject

	5. The communicator fills out the message form fields a. The description field for the push notification body will only be shown for “Alert” distribution subjects 6. The communicator submits the form 7. The system logs the message in the database 8. The system relays the message via FCM API 9. The communicator will be shown a confirmation message and option to navigate back to the message form
Post-Conditions	A message will be relayed via FCM to mobile devices that are subscribed to the specified topic.

Log Out	
Summary	The communicator logs out of the portal
Pre-Conditions	● The communicator should be logged in
Flow	1. The user cookie is unset 2. The communicator is returned to the main page
Post-Conditions	The communicator will be logged out and the user cookie will be unset

Admin (Super User)



Log in	
Summary	An admin can log in to access the portal
Pre-Conditions	
Flow	1. A cookie for the user “Super User” will be set 2. The admin will have access to the admin portion of the portal
Post-Conditions	An admin will be logged in and the user cookie will be set

View Menu	
Summary	An admin can view the menu options
Pre-Conditions	• The admin should be logged in
Flow	• An admin can navigate to different menu options
Post-Conditions	An admin will be navigated to the page for a selected menu option

Log Out	
Summary	The admin logs out of the portal
Pre-Conditions	• The admin should be logged in
Flow	1. The user cookie is unset 2. The admin is returned to the main page
Post-Conditions	The admin will be logged out and the user cookie will be unset

Add Area	
Summary	Adds an area to the information cube
Pre-Conditions	• There must be at least one pre-existing channel
Flow	1. The admin selects a channel to add a new area to 2. The admin enters the information for the new area 3. The admin submits the form 4. The system will add the area to the channel document in the database 5. The admin will be returned to the menu
Post-Conditions	A new area will be added to the information cube

View Areas	
Summary	View areas under a channel
Pre-Conditions	• There must be at least one pre-existing channel
Flow	1. The admin selects a channel to see areas from

	2. The system shows all the areas underneath the selected channel a. The admin can expand/collapse the areas to see more/less details b. The admin can access the edit and delete options when an area is expanded
Post-Conditions	

Edit Area	
Summary	Updates an area
Pre-Conditions	● There must be at least one pre-existing channel ● There must be at least one pre-existing area
Flow	1. The admin updates the area information in the form 2. The admin clicks the “Edit” button 3. The area object in the channel document in the database will be updated 4. The admin is navigated back to the menu
Post-Conditions	An area will be updated

Delete Area	
Summary	Deletes an area
Pre-Conditions	● There must be at least one pre-existing channel ● There must be at least one pre-existing area
Flow	1. The admin clicks the “Delete” button 2. The area element will be removed from the channel document in the database 3. The admin is navigated back to the menu
Post-Conditions	An area will be deleted

Add Subject	
Summary	Adds a subject

Pre-Conditions	● There must be at least one pre-existing channel ● There must be at least one pre-existing area
Flow	1. The admin selects a channel to add a new subject to 2. The admin selects an area to add a new subject to 3. The admin enters the information for the new subject 4. The admin submits the form 5. The system will add the subject to the channel document in the database 6. The admin will be returned to the menu
Post-Conditions	A subject will be added to the information cube

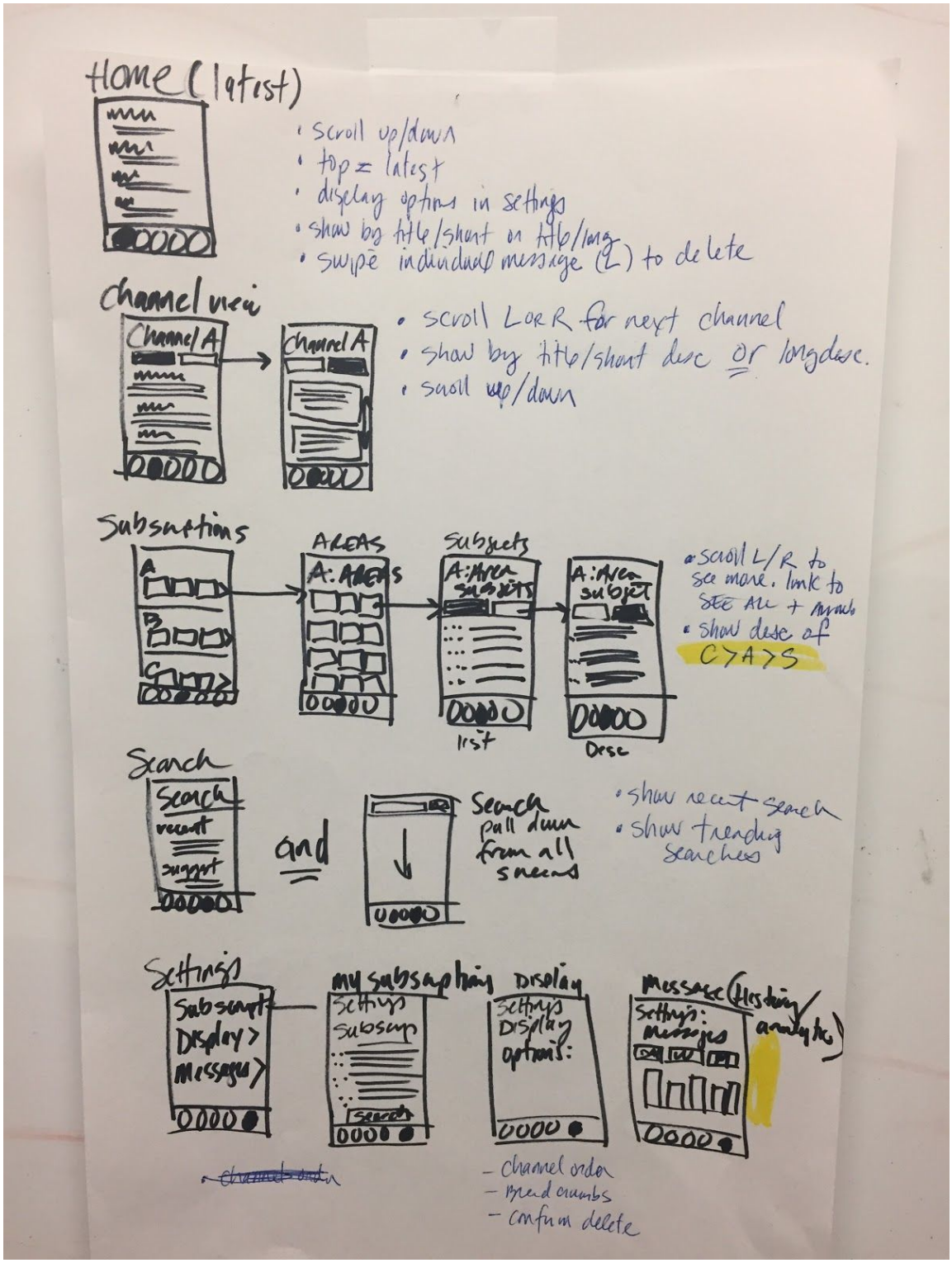
View Subjects	
Summary	View subjects under a channel and area
Pre-Conditions	● There must be at least one pre-existing channel ● There must be at least one pre-existing area
Flow	1. The admin selects a channel to see subjects from 2. The admin selects an area to see subjects from 3. The system shows all the subjects underneath the selected channel and area a. The admin can expand/collapse the areas to see more/less details b. The admin can access the edit and delete options when a subject is expanded
Post-Conditions	

Edit Subject	
Summary	Updates a subject
Pre-Conditions	● There must be at least one pre-existing channel ● There must be at least one pre-existing area ● There must be at least one pre-existing subject
Flow	1. The admin updates the subject information in the form 2. The admin clicks the “Edit” button 3. The subject object in the channel document in the database will be updated 4. The admin is navigated back to the menu

Post-Conditions	A subject will be updated in the information cube

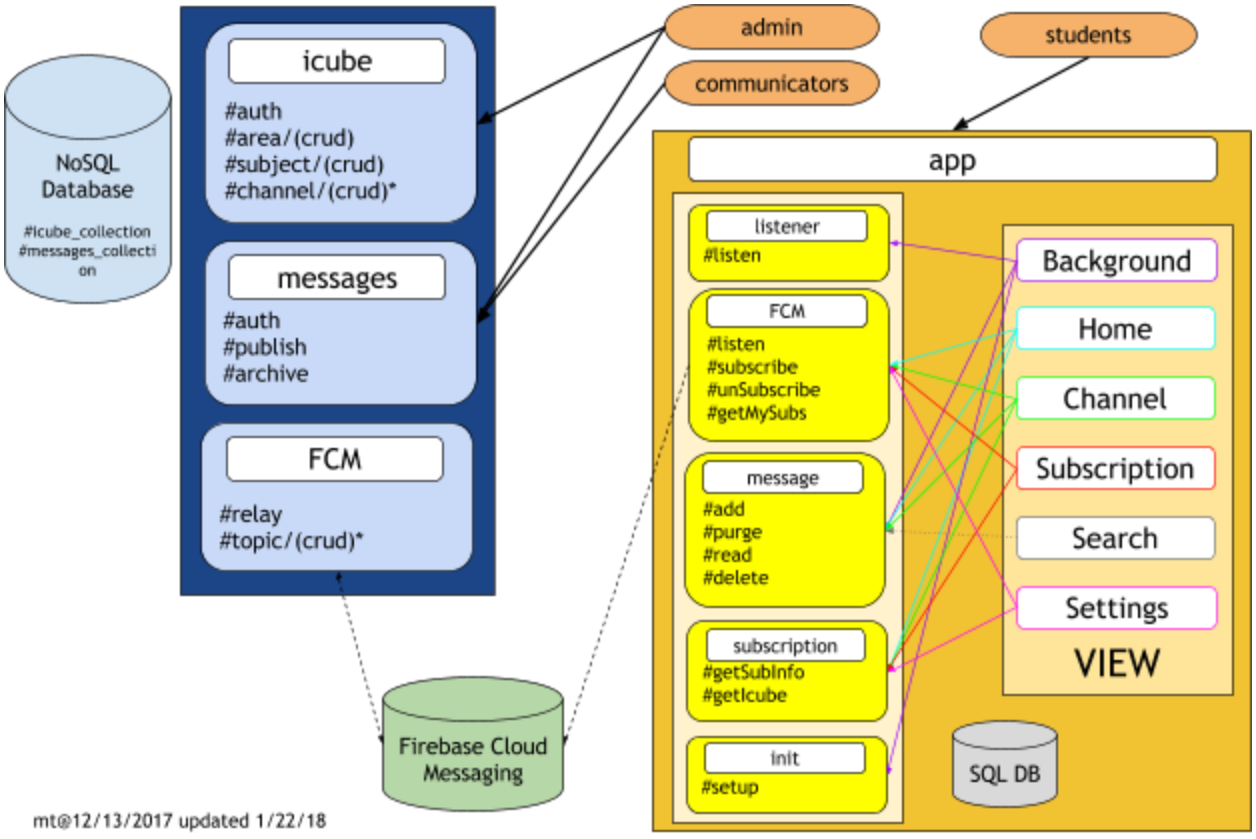
Delete Subject	
Summary	Deletes a subject
Pre-Conditions	● There must be at least one pre-existing channel ● There must be at least one pre-existing area ● There must be at least one pre-existing subject
Flow	1. The admin clicks the “Delete” button 2. The subject element will be removed from the channel document in the database 3. The admin is navigated back to the menu
Post-Conditions	A subject will be removed from the information cube

Mobile App UI Components



System Design

Architecture



API Documents

REST API

Channel

Endpoint	/channel
Method	GET
URL Parameters	None
Body	None
Outputs/Effects	Returns all entries from icube collection as array of objects in JSON format.

Area

Endpoint	/area/:channelID
Method	POST
URL Parameters	The URL parameter :channelID is a string and is required. It must correspond with a channel ID (see Notes section).

Body	The body must contain a JSON object for a new Area to be added to the areas array field in the specified channel. The body must be the following format:
	{
	"name": string,
	"desc": string
	}

Outputs/Effects	The given new area object will be initialized with necessary fields (a unique ID, an empty access array, and an empty subjects array) and will be added to the specified channel.

Endpoint	/area/:channelID/:areaID
Method	PUT
URL Parameters	The URL parameters :channelID and :areaID are strings and are both required. The :channelID parameter must correspond with a channel ID and the :areaID must correspond with an area ID (see Notes section).

Body	The body must contain a JSON object for fields to update in an area. These fields will replace the current values, so if any field is to remain unchanged the current value must be supplied. The body must be the following format:
	{
	"name": string,
	"desc": string
	}

Outputs/Effects	The specified area's attributes will replaced with the given ones in the body.

Endpoint	/area/:channelID/:areaID
Method	DELETE
URL Parameters	The URL parameters :channelID and :areaID are strings and are both required. The :channelID parameter must correspond with a channel ID and the :areaID must correspond with an area ID (see Notes section).
Body	None
Outputs/Effects	The specified area will deleted from the specified channel.

Subject

Endpoint	/subject/:channelID/:areaID
Method	POST

URL Parameters	The URL parameters :channelID and :areaID are strings and are both required. The :channelID parameter must correspond with a channel ID and the :areaID must correspond with an area ID (see Notes section).

Body	The body must contain a JSON object for a new subject to be added to subjects array field in the specified area element of the specified channel's areas. The body must be the following format:
	{
	"name": string,
	"desc": string,
	"opt": {
	"level": enumerated("Forced | "Recommended" | ""),
	"distribution": enumerated("Alert" | "Information")
	}
	}

Outputs/Effects	The given new subject object will be initialized with necessary fields (a unique ID, an empty access array, an empty tags array, and a topic key) and will be added to the specified area of the specified channel.

Endpoint	/subject/:channelID/:areaID/:subjectID
Method	PUT
URL Parameters	The URL parameters :channelID, :areaID, and :subjectID are strings and all required. The :channelID parameter must correspond with a channel ID, the :areaID must correspond with an area ID, and the :subjectID must correspond with a subject ID (see Notes section).

Body	The body must contain a JSON object for fields to update in a subject. These fields will replace the current values, so if any field is to remain unchanged the current value must be supplied. The body must be the following format:
	{
	"name": string,
	"desc": string,
	"opt": {
	"level": enumerated("Forced | "Recommended" | ""),
	"distribution": enumerated("Alert" | "Information")

	} }
Outputs/Effects	The specified subject's attributes will replaced with the given ones in the body.

Endpoint	/subject/:channelID/:areaID/:subjectID
Method	DELETE
URL Parameters	The URL parameters :channelID, :areaID, and :subjectID are strings and all required. The :channelID parameter must correspond with a channel ID, the :areaID must correspond with an area ID, and the :subjectID must correspond with a subject ID (see Notes section).
Body	None
Outputs/Effects	The specified subject will deleted from the specified area and channel.

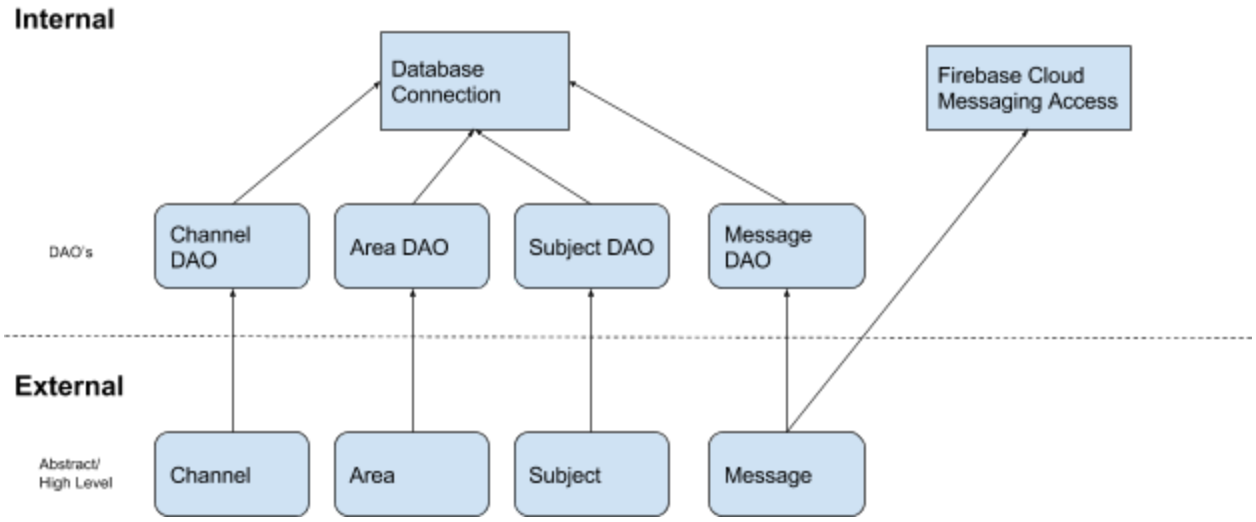
Message

Endpoint	/message
Method	POST
URL Parameters	None
Body	The body must contain a JSON object of message fields to be logged and used for relaying the message via FCM. The topic_key field must correspond to a topic_key field of a subject (see Notes section). The desc field can be equal to "" for Information type messages (opt.distribution == "Information"). Below is the required JSON format for the body: { "title": string, "desc": string, "message": string, "topic_key" : $topic_key,

	"distribution": enumerated("Alert" | "Information") }
Outputs/Effects	The message object will be given a unique identifier (msi_key) and timestamp, logged in the database, and relayed via FCM.

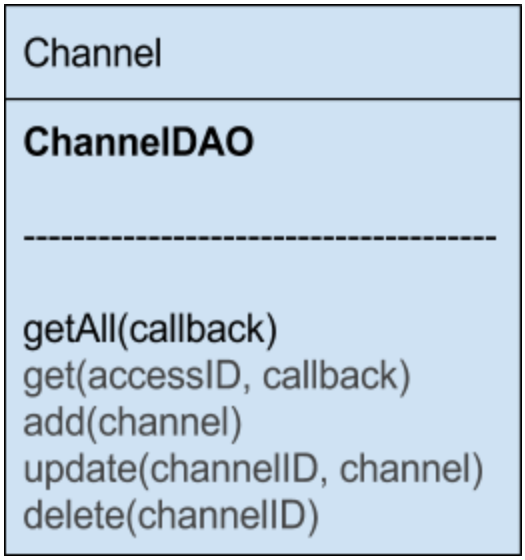
Server Modules

Overview



External Modules

Channel



Description

This module provides an abstract interface for performing CRUD operations on a channel object. It uses a database access object module so its methods are only dependent on the schema of the data and thus not dependent on the database connection and driver implementation.

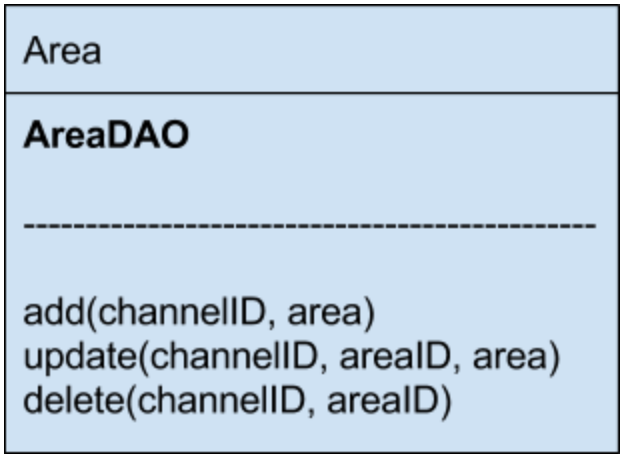
Attributes

- The ChannelDAO module is used for performing CRUD operations.

Methods

getAll				
Summary	This method retrieves all of the channels and their nested areas and subjects using the database access object module.			
Inputs	name	datatype	description	required?
	callback	callback function	Function to perform with retrieved information cube	yes
Outputs/Effects	Subscription data in the form of an array of all icube documents is passed to the given callback function.			

Area



Description

This module provides an abstract interface for performing CRUD operations on an area object. It uses a database access object module so its methods are only dependent on the schema of the data and thus not dependent on the database connection and driver implementation.

Attributes

- The AreaDAO module is used for performing CRUD operations.

Methods

add				
Summary	This method uses the databases access object module to add an area under a specified channel.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for under which channel to add a new area	yes
	area	object	New area to add. Must be in the below format.	yes

Outputs/Effects	An area will be added to an icube document in the database.

add - area input format:

```
{
  "id": MongoDB ObjectID,
  "access": [],
  "subjects": [],
  "name": string,
  "desc": string,
}
```

update				
Summary	This method uses the database access object module to replace an area's fields with the provided values.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for which channel's area to update	yes
	areaID	string	Identifier for which area to update	yes
	area	object	Values to replace in the area. Must be in the below format.	yes
Outputs/Effects	The fields of a specified area in an icube document in the database will be replaced with the provided values.			

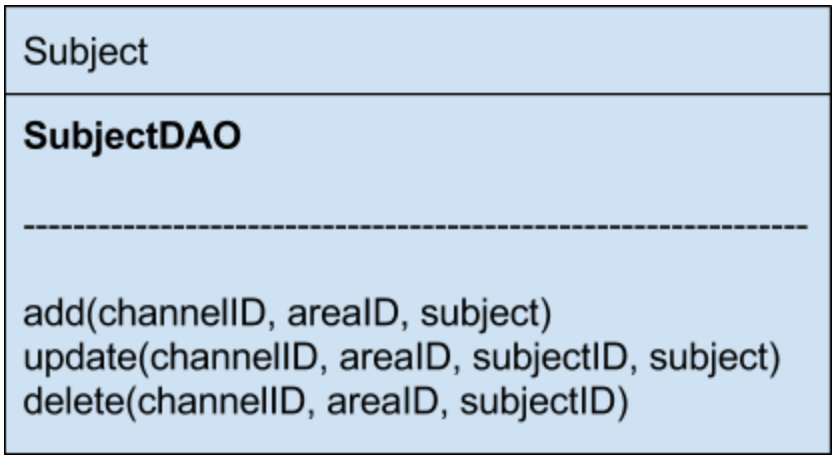
update - area input format:

```
{
  "name": string,
  "desc": string,
}
```

delete	
Summary	This method uses the database access object module to remove an area.

Inputs	name	datatype	description	required?
	channelID	string	Identifier for which channel to remove an area from	yes
	areaID	string	Identifier for which area to remove	yes
Outputs/Effects	The specified area will be deleted from an icube document in the database.			

Subject



Description

This module provides an abstract interface for performing CRUD operations on a subject object. It uses a database access object module so its methods are only dependent on the schema of the data and thus not dependent on the database connection and driver implementation.

Attributes

- The SubjectDAO module is used for performing CRUD operations.

Methods

add				
Summary	This method uses the databases access object module to add a subject under a specified channel's area.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for under which channel to add a new subject	yes
	areaID	string	Identifier for under which area to add a new subject	yes
	subject	object	New subject to add. Must be in the below format.	yes
Outputs/Effects	A subject will be added to an icube document in the database.			

add - subject input format:

```
{
  "id": MongoDB ObjectID,
  "topic_key":  string,
  "access": [],
  "tags": [],
  "name": string,
  "desc": string,
  "opt": {
    "level": enumerated("Forced | "Recommended" | ""),
    "distribution": enumerated("Alert" | "Information")
  }
}
```

update				
Summary	This method uses the database access object module to replace a subject's fields with the provided values.			
Inputs	name	datatype	description	required?

	channelID	string	Identifier for under which channel to update a subject	yes
	areaID	string	Identifier for which area's subject to update	yes
	subjectID	string	Identifier for which subject to update	yes
	subject	object	Values to replace in the subject. Must be in the below format.	yes
Outputs/Effects	The fields of a specified subject in an icube document in the database will be replaced with the provided values.			

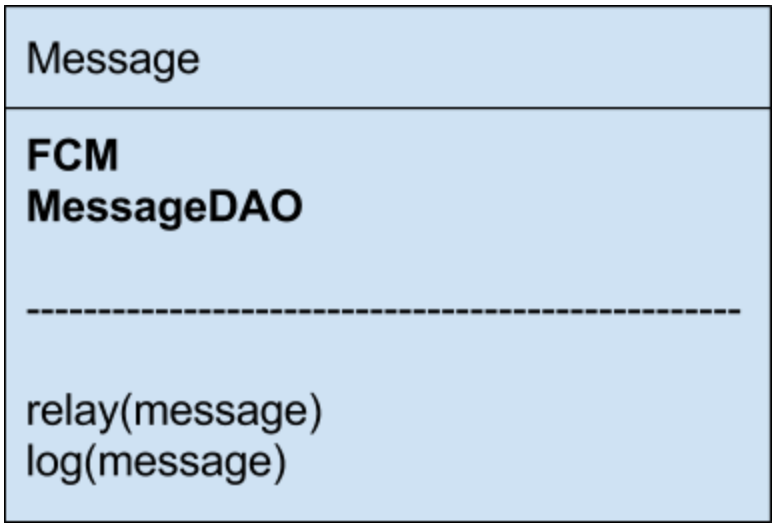
update - subject input format:

```
{
  "name": string,
  "desc": string,
  "opt": {
    "level": enumerated("Forced | "Recommended" | ""),
    "distribution": enumerated("Alert" | "Information")
  }
}
```

delete				
Summary	This method uses the database access object module to remove a subject.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for which channel to remove a subject from	yes

	areaID	string	Identifier for which area to remove a subject from	yes
	subjectID	string	Identifier for which subject to remove	yes
Outputs/Effects	The specified subject will be deleted from an icube document in the database.			

Message



Description

This module provides an abstract interface for logging and relaying messages. It uses a database access object module so its log method is only dependent on the schema of the data and thus not dependent of the database connection and driver implementation. Also, it uses a FCM access module so its relay message is just dependent on the data schema and thus not dependent on FCM authorization or the specifics of the API formatting.

Attributes

- The MessageDAO module is used for performing CRUD operations
- The FCM module is used for cloud messaging

Methods

relay				
Summary	This method determines the distribution type of the message and uses the FCM access module to relay the message accordingly.			
Inputs	name	datatype	description	required?
	message	object	The message object must be the below format.	yes
Outputs/Effects	An information or alert type message will be relayed via FCM			

relay - message input format:

```
{
  "msi_key": string,
  "timestamp": unix timestamp integer,
  "title": string,
  "desc": string,
  "message": string,
  "topic_key" : $topic_key,
  "distribution": enumerated("Alert" | "Information")
 }
```

log				
Summary	This method uses the database access object to archive a message.			
Inputs	name	datatype	description	required?
	message	object		yes
Outputs/Effects	A message document will be added to the message collection in the database.			

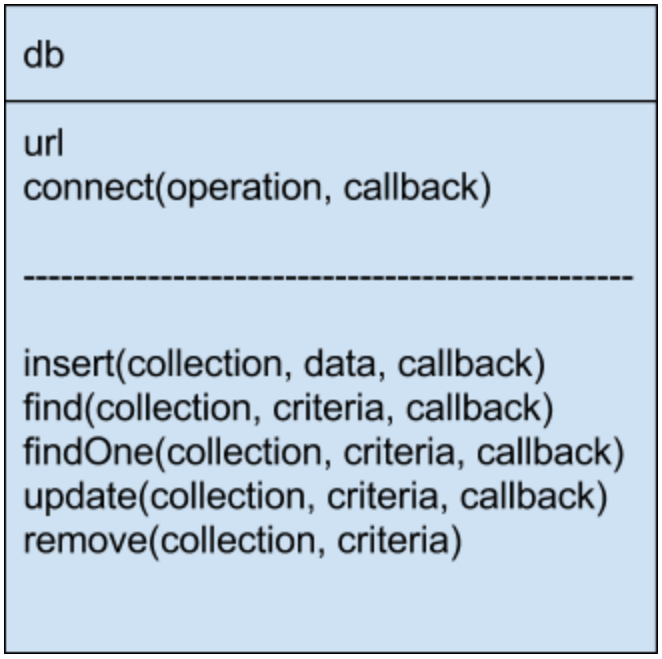
log - message input format:

```
{
  "msi_key": string,
  "timestamp": unix timestamp integer,
  "title": string,
  "desc": string,
```

```
  "message": string,
  "topic_key" : $topic_key,
  "distribution": enumerated("Alert" | "Information")
}
```

Internal Modules

Database



Description

This module provides a connection and interface to the MongoDB database.

Attributes

- The url is for the location of the database to have the module connect.

Methods

connect				
Summary	Internal method for connecting to the database and performing the given operation callback function with the specified collection. Passes a collection object to the operation callback to be able to perform the operation.			
Inputs	name	datatype	description	required?

	operation	callback function	Database operation to perform	yes
	collection	string	MongoDB collection to perform operation on	yes
Outputs/Effects	Logs error to console on failed attempt to connect to the database			

insert				
Summary	Method for inserting new document into a specified collection. It passes an insertion operation and the collection string to the internal connect method.			
Inputs	name	datatype	description	required?
	collection	string	MongoDB collection to insert new document into	yes
	data	object	object of new document to add to collection	yes
	callback	callback function	Callback function to see write results	no
Outputs/Effects	Logs error to console if any. If given a callback, will pass output information as a parameter to it.			

find				
Summary	Method for finding documents in the specified collection with the matching criteria. It passes a find operation and the collection string to the internal connect method.			
Inputs	name	datatype	description	required?

	collection	string	MongoDB collection to retrieve documents from	yes
	criteria	MongoDB query object	Criteria for finding documents, can be {} to find all	yes
	callback	callback function	Callback function	yes
Outputs/Effects	Logs error to console if any. Passes array of found documents to given callback.			

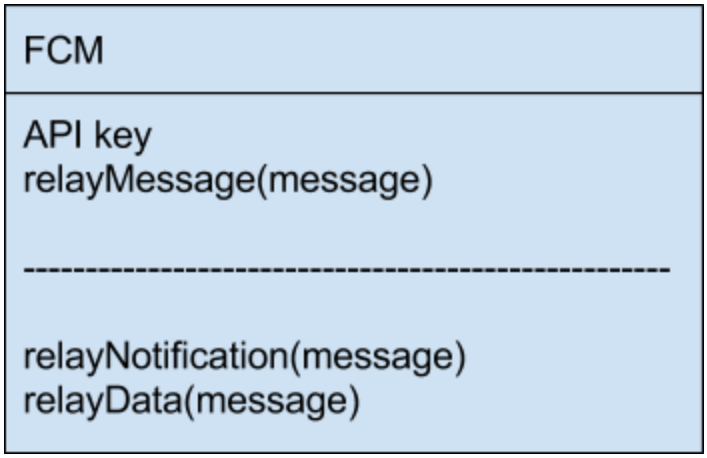
findOne				
Summary	Method for finding one matching documents in the specified collection with the matching criteria. It passes a findOne operation and the collection string to the internal connect method.			
Inputs	name	datatype	description	required?
	collection	string	MongoDB collection to retrieve a document from	yes
	criteria	MongoDB driver query object	Criteria for finding a document	yes
	callback	callback function	Callback function	yes
Outputs/Effects	Logs error to console if any. Passes found document to given callback.			

update	
Summary	Method for updating documents in the specified collection with the matching criteria. It passes an update operation and the collection string to the internal connect method.

Inputs	name	datatype	description	required?
	collection	string	MongoDB collection in which to update documents	yes
	criteria	MongoDB selector object	Criteria for which document(s) to update	yes
	data	MongoDB document object	Fields and values to update	yes
Outputs/Effects	Logs error to console if any.			

remove				
Summary	Method for deleting documents in the specified collection with the matching criteria. It passes a remove operation and the collection string to the internal connect method.			
Inputs	name	datatype	description	required?
	collection	string	MongoDB collection to remove documents from	yes
	criteria	MongoDB driver query object	Criteria for finding documents	yes
Outputs/Effects	Logs error to console if any.			

FCM



Description

This module provides an interface for using Firebase Cloud Messaging, handling authorization with the API key and necessary formatting for the FCM API.

Attributes

- The API key corresponds with a Firebase project that is used in conjunction with the server this module resides on. It is needed for authorization for calling the FCM REST endpoint.

Methods

relayMessage				
Summary	Internal method that makes a HTTP POST request to the FCM API with the pre-formatted message for the request body and using the api_key for authorization.			
Inputs	name	datatype	description	required?
	message	JSON object	Contains message body, already formatted for FCM API. (See FCM Mapping section).	yes

Outputs/Effects	A message with data and/or a notification will be relayed via FCM.

relayNotification				
Summary	This method takes in fields for an alert message, formats these fields into the necessary format for a notification message as per FCM's API, and passes the formatted message to the internal relayMessage method to send the message.			
Inputs	name	datatype	description	required?
	message	object	The message object must be the below format.	yes
Outputs/Effects	The notification component (see FCM Mapping section) of a message will be relayed.			

relayNotification - message input format:

```
{
  "msi_key": string,
  "timestamp": unix timestamp integer,
  "title": string,
  "desc": string,
  "message": string,
  "topic_key" : $topic_key,
  "distribution": enumerated("Alert" | "Information")
 }
```

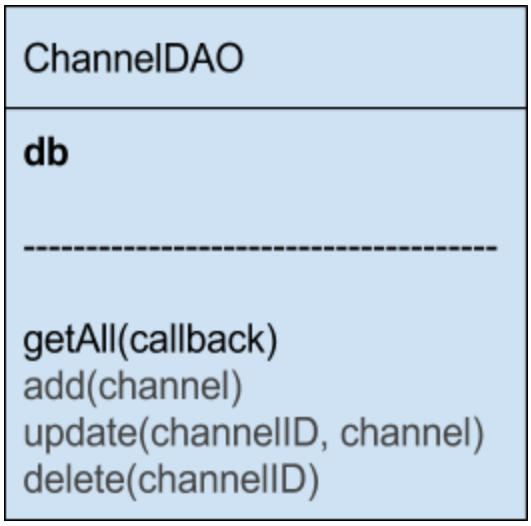
relayData				
Summary	This method takes in fields for an information message, formats these fields into the necessary format for a data message as per FCM's api, and passes the formatted message to the internal relayMessage method to send the message.			
Inputs	name	datatype	description	required?
	message	object	The message object must be	yes

			the below format.	
Outputs/Effects	The data component (see FCM Mapping section) of a message will be relayed.			

relayData - message input format:

```
{
  "msi_key": string,
  "timestamp": unix timestamp integer,
  "title": string,
  "desc": string,
  "message": string,
  "topic_key" : $topic_key,
  "distribution": enumerated("Alert" | "Information")
 }
```

Channel DAO



Description

This module serves as a database access object to perform CRUD operations on a channel object. It uses the database connection module and methods implemented based on that module and the MongoDB driver.

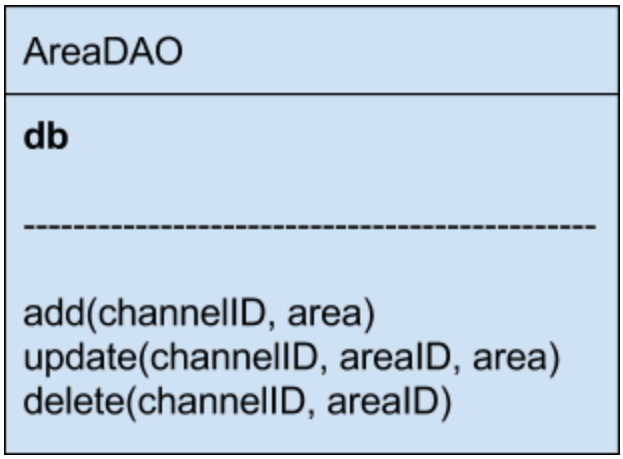
Attributes

- The database module is used for performing CRUD operations.

Methods

getAll				
Summary	This method retrieves all of the channels and their nested areas and subjects from the database using the database connection module.			
Inputs	name	datatype	description	required?
	callback	callback function	Function to perform with retrieved information cube	yes
Outputs/Effects	Subscription data in the form of an array of all icube documents are passed to the given callback function.			

Area DAO



Description

This module serves as a database access object to perform CRUD operations on an area object. It uses the database connection module and methods implemented based on that module and the MongoDB driver.

Attributes

- The database module is used for performing CRUD operations.

Methods

add				
Summary	This method uses the database connection module to add an area under a specified channel.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for under which channel to add a new area	yes
	area	object	New area to add. Must be in the below format.	yes
Outputs/Effects	An area will be added to an icube document in the database.			

add - area input format:

```
{
  "id": MongoDB ObjectID,
  "access": [],
  "subjects": [],
  "name": string,
  "desc": string,
}
```

update				
Summary	This method uses the database connection module to replace an area's fields with the provided values.			
Inputs	name	datatype	description	required?

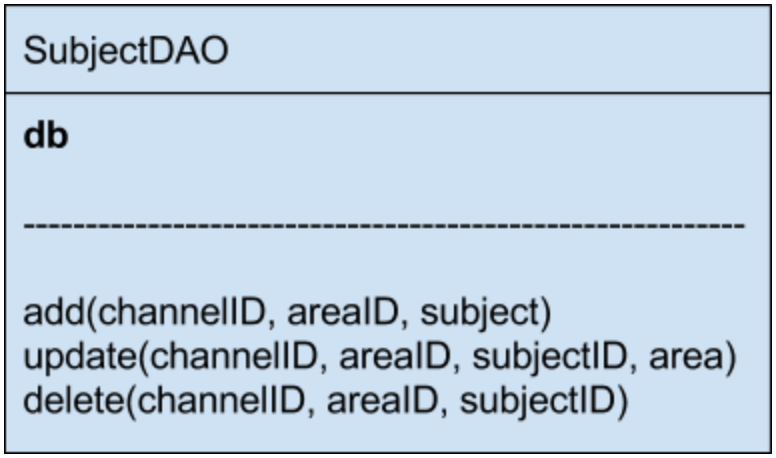
	channelID	string	Identifier for which channel's area to update	yes
	areaID	string	Identifier for which area to update	yes
	area	object	Values to replace in the area. Must be in the below format.	yes
Outputs/Effects	The fields of a specified area in an icube document in the database will be replaced with the provided values.			

update - area input format:

```
{
  "name": string,
  "desc": string,
}
```

delete				
Summary	This method uses the database connection module to remove an area.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for which channel to remove an area from	yes
	areaID	string	Identifier for which area to remove	yes
Outputs/Effects	The specified area will be deleted from an icube document in the database.			

Subject DAO



Description

This module serves as a database access object to perform CRUD operations on a subject object. It uses the database connection module and methods implemented based on that module and the MongoDB driver.

Attributes

- The database module is used for performing CRUD operations.

Methods

add				
Summary	This method uses the databases connection module to add a subject under a specified channel's area.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for under which channel to add a new subject	yes
	areaID	string	Identifier for under which area to add a new subject	yes

	subject	object	New subject to add. Must be in the below format.	yes
Outputs/Effects	A subject will be added to an icube document in the database.			

add - subject input format:

```
{
  "id": MongoDB ObjectID,
  "topic_key":  string,
  "access": [],
  "tags": [],
  "name": string,
  "desc": string,
  "opt": {
    "level": enumerated("Forced | "Recommended" | ""),
    "distribution": enumerated("Alert" | "Information")
  }
}
```

update				
Summary	This method uses the database connection module to replace a subject's fields with the provided values.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for under which channel to update a subject	yes
	areaID	string	Identifier for which area's subject to update	yes
	subjectID	string	Identifier for which subject to update	yes
	subject	object	Values to replace in the	yes

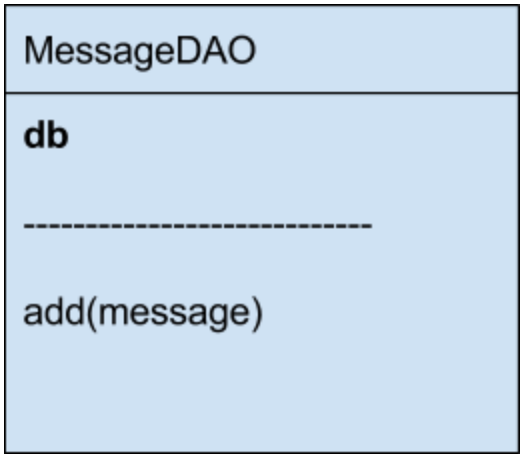
			subject. Must be in the below format.	
Outputs/Effects	The fields of a specified subject in an icube document in the database will be replaced with the provided values.			

update - subject input format:

```
{
  "name": string,
  "desc": string,
  "opt": {
    "level": enumerated("Forced | "Recommended" | ""),
    "distribution": enumerated("Alert" | "Information")
  }
}
```

delete				
Summary	This method uses the database connection module to remove a subject.			
Inputs	name	datatype	description	required?
	channelID	string	Identifier for which channel to remove a subject from	yes
	areaID	string	Identifier for which area to remove a subject from	yes
	subjectID	string	Identifier for which subject to remove	yes
Outputs/Effects	The specified subject will be deleted from an icube document in the database.			

Message DAO



Description

This module serves as a database access object to perform CRUD operations on a message document. It uses the database connection module and methods implemented based on that module and the MongoDB driver.

Attributes

- The database module is used for performing CRUD operations.

Methods

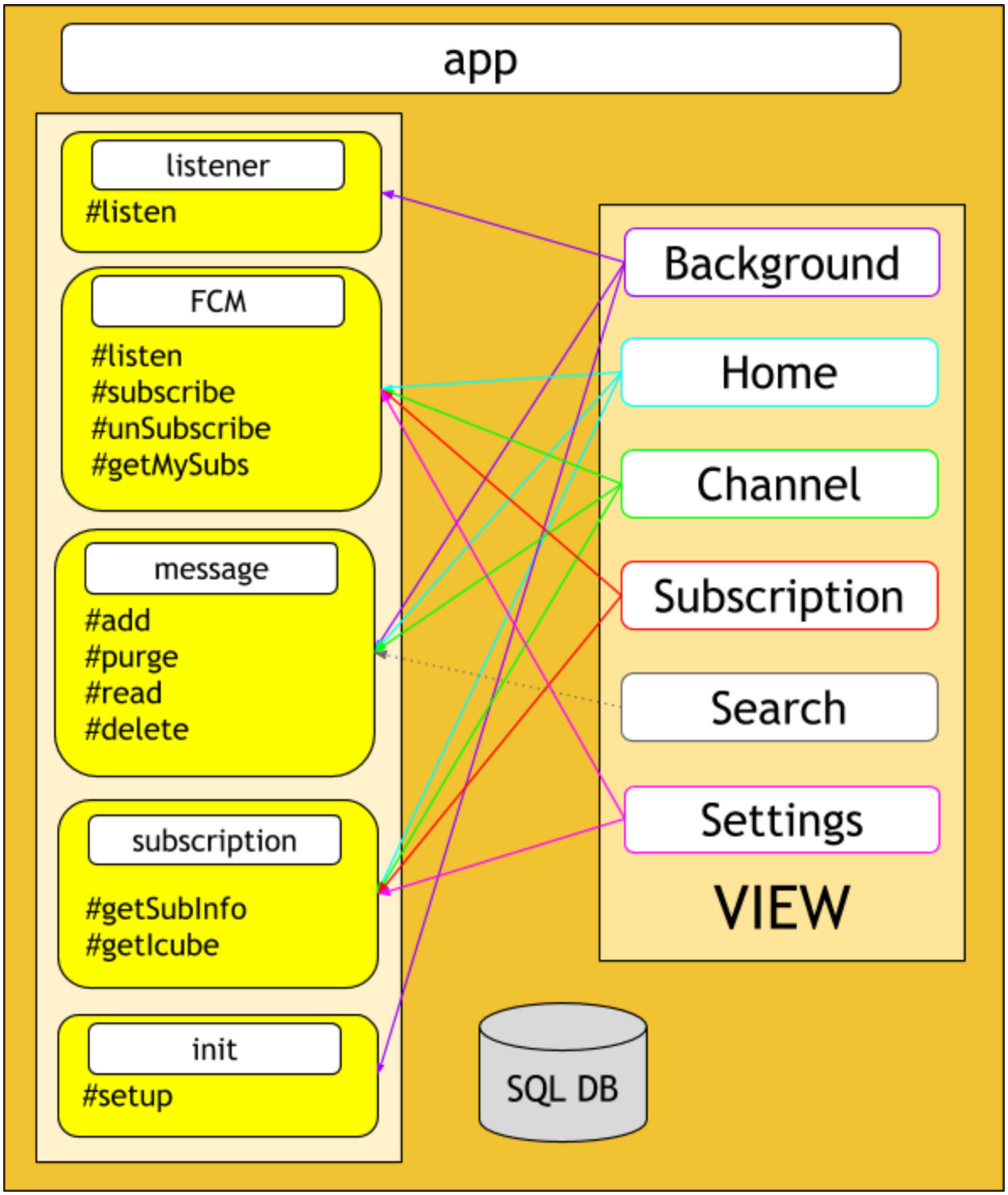
add				
Summary	This method adds a new message.			
Inputs	name	datatype	description	required?
	message	object	The message object must be the below format.	yes
Outputs/Effects	A message document will be added to the message collection in the database.			

add - message input format:

```
{
  "msi_key": string,
  "timestamp": unix timestamp integer,
  "title": string,
  "desc": string,
  "message": string,
  "topic_key" : $topic_key,
  "distribution": enumerated("Alert" | "Information")
 }
```

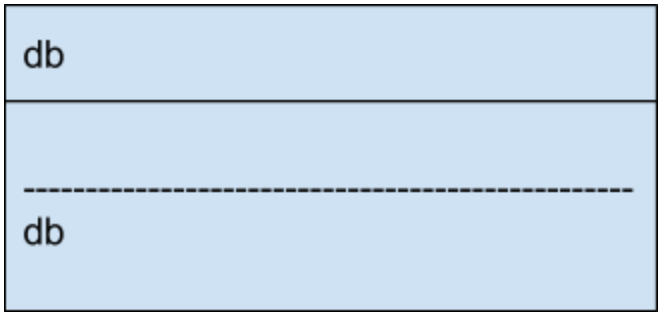
Mobile App Modules

Overview



Modules

Database



Description

This module connects to the SQLite database, creates the Message table if it does not already exists, and exports the connection database object (db).

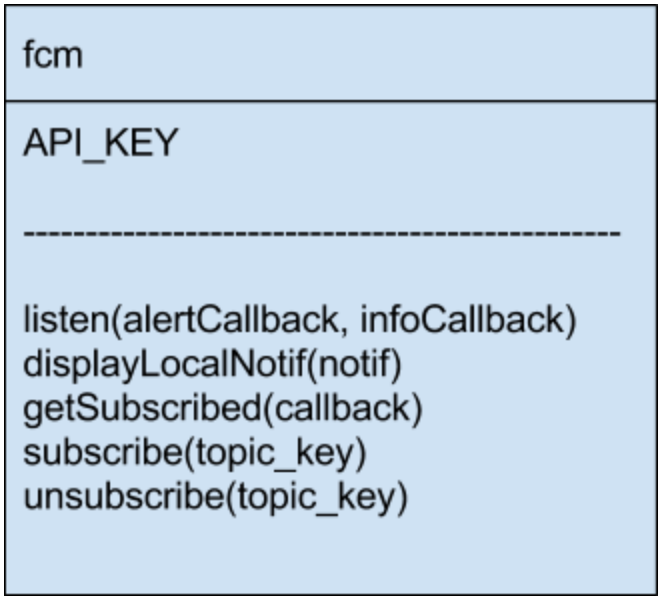
Attributes

- db is a SQLite database connection object

Methods

None

FCM



Description

This module provides an access object for message processing and device subscribe/unsubscribe functionality from the react-native-fcm npm module. In addition, it also provides access to the FCM web REST API for retrieving all the topics to which a device is subscribed.

Attributes

- The API key is used for authorization to use the FCM REST API

Methods

listen				
Summary	This method listens for FCM messages and is able to perform actions in given callbacks when a message is received.			
Inputs	name	datatype	description	required?
	alertCallback	callback	Callback function to call if a notification message is	no

			received	
	infoCallback	callback	Callback function to call if a data message is received	no
Outputs/Effects	If a notification message is received, the alertCallback will be called. If a data message is received, the infoCallback will be called.			

displayLocalNotif				
Summary	Displays local notification on Android. This is a workaround method for the behavior with Android and the react-native-fcm library. In other words, push notifications are not displayed if the app is open in the foreground on Android so this generates the display manually.			
Inputs	name	datatype	description	required?
	notif	react-native-fcm notification object	Notification object received by the above listen function	yes
Outputs/Effects	On Android, this adds a push notification to the status bar and generates a pop up. This method cannot be used on iOS.			

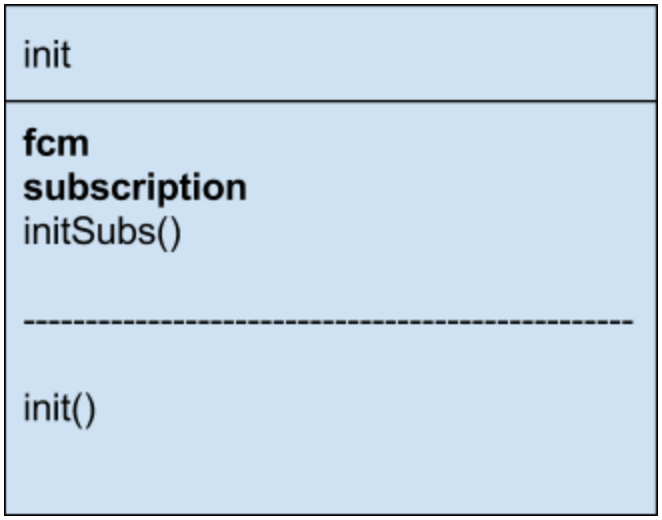
getSubscribed				
Summary	Retrieves all the topic_keys the device is subscribed to			
Inputs	name	datatype	description	required?
	callback	callback function	Callback function to handle list of subscribed topics	yes
Outputs/Effects	Passes an array of topic_key strings to the given callback function			

subscribe	
Summary	Subscribes the device to a topic

Inputs	name	datatype	description	required?
	topic_key	string	Identifier for a FCM topic, should correspond to a channel, area, subject combination in the server database	yes
Outputs/Effects	The device will be subscribed to the topic corresponding with the given topic_key. Note that if no other devices are currently subscribed to the topic_key, this will create the topic in FCM.			

unsubscribe				
Summary	Unsubscribes the device to a topic			
Inputs	name	datatype	description	required?
	topic_key	string	Identifier for a FCM topic, should correspond to a channel, area, subject combination in the server database	yes
Outputs/Effects	The device will be unsubscribed from the topic corresponding with the given topic_key.			

Initialization



Description

This module provides functionality for initializing the mobile app for being able to receive FCM messages and to automatically subscribe the device to forced and recommend subscriptions. Note this module exports just the init function, not an exports object.

Attributes

- The fcm module is used for being able to subscribe the device
- The subscription module is used for retrieving the information cube

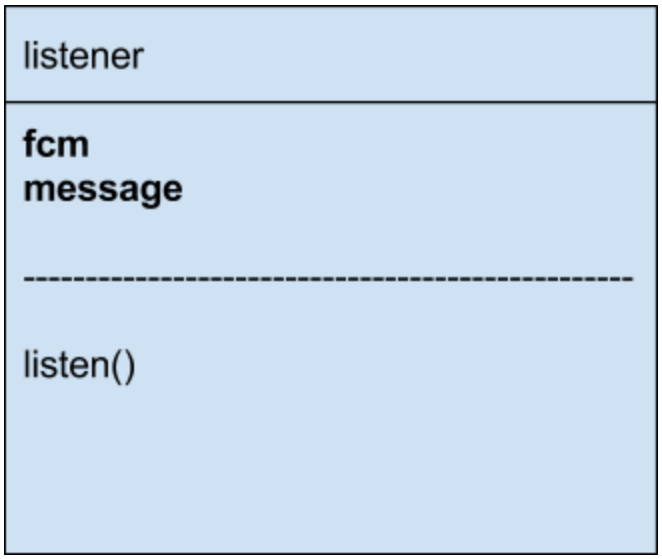
Methods

initSubs	
Summary	This internal function retrieves the information cube and subscribes the device to any forced or recommended topic.
Inputs	None
Outputs/Effects	If successful, the device will be subscribed to all the topics that are currently in the information cube as either forced or recommended

init

Summary	This function requests permissions if the app is running on an iOS device, calls initSubs to automatically subscribe the device to forced and recommended topics, and marks the app as initialized.
Inputs	None
Outputs/Effects	If successful, the app will be able to receive messages, be subscribed to forced and recommend subscriptions, and be marked as initialized.

Listener



Description

This modules provides a method for the main listening behavior of the mobile app to run in the background and process incoming messages.

Attributes

- The fcm module is used for being able to listen to messages and display local notifications on Android
- The message module is used for saving messages to the mobile app's database

Methods

listen

Summary	This method is a wrapper for the fcm module's listen method with callbacks for displaying a local notification on Android (see the FCM module displayLocalNotif function for more on this) and saving received data messages in the app's database (if the message has not already been saved).
Inputs	None
Outputs/Effects	If the app is running in the foreground on an Android device, a local notification will be generated. Received data messages will be saved in the local database.

Message

message
db --- addMessage(msi_key, topic_key, dist, title, description, message, timestamp) getMessages(callback) exists(msi_key, callback) removeMessage(msi_key) clearOldMessages()

Description

This module provides an interface for CRUD related operations on the Message table in the mobile app's local SQLite database.

Attributes

- The db connection object is used to be able to perform operations on the database

Methods

addMessage

Summary	Adds a new message			
Inputs	name	datatype	description	required?
	msi_key	string	Message identifier	yes
	topic_key	string	Identifier for a FCM topic, should correspond to a channel, area, subject combination in the server database	yes
	dist	string	How the message was distributed (either “Alert” or “Information”)	yes
	title	string	Title of message	yes
	description	string	Push notification body text or “” if information type message	yes
	message	string	The main message text	yes
	timestamp	Unix timestamp as integer	The timestamp the message was created (server-side)	yes
Outputs/Effects	A new message record will be inserted into the Message table in the database			

getMessages	
Summary	Retrieves all the stored messages sorted by order of most recent

Inputs	name	datatype	description	required?
	callback	Callback function	Callback function that is passed the retrieved messages	yes
Outputs/Effects	The given callback function will be passed an array of message objects			

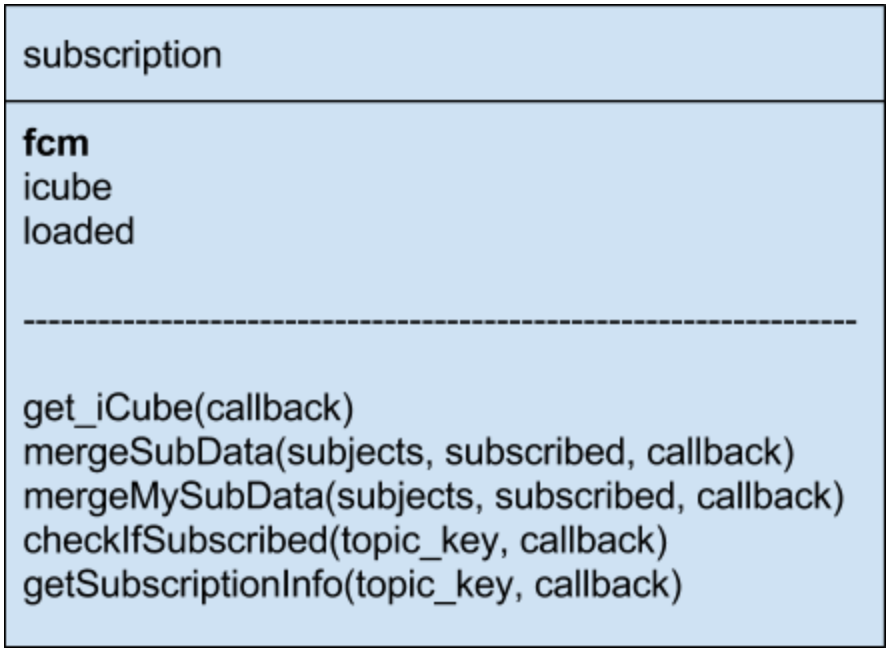
exists				
Summary	Determines whether a message with the specified msi_key currently exists in the Message table in the database			
Inputs	name	datatype	description	required?
	msi_key	string	Message identifier	yes
	callback	Callback function	Callback function to pass found result	yes
Outputs/Effects	The given callback function will be passed a boolean for whether the message currently exists in the database			

removeMessage				
Summary	Deletes a message with the specified msi_key			
Inputs	name	datatype	description	required?
	msi_key	string	Message identifier	yes
Outputs/Effects	Any message with a msi_key matching the one given will be removed from the Message table in the database			

clearOldMessages	
Summary	Removes week old messages; specifically removes those that are older than a week from the end of the current day
Inputs	None

Outputs/Effects	Message table records with a timestamp from a day that is a week or older from the current day will be deleted

Subscription



Description

This module is used as an interface for retrieving data from the server-side information cube and the FCM module as well as merging and formatting that data.

Attributes

- The fcm module is used for retrieving which topics a device is subscribed to
- The icube variable is used for caching the information cube
- The loaded variable is a boolean for whether the information cube has been stored in the icube variable yet

Methods

get_iCube	
Summary	If loaded is true, the information cube will retrieved from the icube

	variable. If loaded is false, it will be retrieved from the REST API endpoint, stored in the icube variable, and then the loaded flag will be set to true.			
Inputs	name	datatype	description	required?
	callback	Callback function	Callback to pass the retrieved information cube to	yes
Outputs/Effects	The information cube will be passed to the given callback as an array of objects. If it had not been cached yet, it will be saved and the loaded flag will be set in the module.			

mergeSubData				
Summary	Takes in subject data and the list of currently subscribed topics, updates given subject data to have a boolean field for whether currently subscribed to			
Inputs	name	datatype	description	required?
	subjects	Array of objects	Array of subject objects	yes
	subscribed	Array of strings	Array of topic keys that correspond with topics the device is currently subscribed to	yes
	callback	Callback function	Callback function that will be passed the merged data	yes
Outputs/Effects	The callback will be passed an array of subject objects that now have a subscribed field			

mergeMySubData	
Summary	Takes in subject data and the list of currently subscribed topics, updates the data to only include subscribed subjects and to have a boolean field

	for whether currently subscribed to			
Inputs	name	datatype	description	required?
	subjects	Array of objects	Array of subject objects	yes
	subscribed	Array of strings	Array of topic keys that correspond with topics the device is currently subscribed to	yes
	callback	Callback function	Callback function that will be passed the merged data	yes
Outputs/Effects	The callback will be passed an array of subject objects that now have a subscribed field and have been filtered to only include those currently subscribed to			

checkIfSubscribed				
Summary	Checks whether a subject with the specified topic key is currently subscribed to			
Inputs	name	datatype	description	required?
	topic_key	string	Identifier for a FCM topic, should correspond to a channel, area, subject combination in the server database	yes
	callback	callback function	Callback function to pass found result	yes
Outputs/Effects	A boolean for whether the subject is subscribed to is passed to the given callback			

getSubscriptionInfo				
Summary	Retrieves the channel, area, and subject names and subscription level for the given topic key as well as whether the topic is currently subscribed to.			
Inputs	name	datatype	description	required?
	topic_key	string	Identifier for a FCM topic, should correspond to a channel, area, subject combination in the server database	yes
	callback	callback function	Callback function to pass retrieved subscription information	yes
Outputs/Effects	Passes subscription information as an object in the below format to the given callback function. If the topic_key is not found, placeholder values as listed below will be returned.			

getSubscriptionInfo - Data Output Format:

```
{
  "channel": string,
  "area": string,
  "subject": string,
  "level": enumerated("Forced" || "Recommended" || ""),
  "subscribed": boolean,
}
```

getSubscriptionInfo - Placeholder Output Values:

```
{
  "channel": "Channel",
  "area": "Area",
  "subject": "Subject",
  "subscribed": false,
}
```

Databases & Data Structures

Server

Database

The server uses a MongoDB database, which is noSQL based. It is used to store the information cube (channels, areas, and subjects pertaining to subscriptions) and archive messages.

Schema

Collections

There are two collections in the database - icube and message. The first holds pieces of the information cube and the latter holds messages submitted through the communicator portal.

JSON

icube

```
{
   "_id": ObjectID
   "name": "channel name",
   "desc": "channel desc",
   "access": [],
   "areas": [
    {
      "id": ObjectID,
      "name": "area name",
      "desc": "area desc",
      "access": [],
      "subjects": [
       {
         "id": ObjectID,
         "name": "subject name",
         "desc": "subject desc",
         "access": [],
```

```
          "topic_key": channel._id + "-" + area.id + "-" + subject.id,
          "opt": {
            "level": enum("Forced" || "Recommended" || ""),
            "distribution": enum("Alert" | "Information")
          },
          "tags": []
        }
      ]
    }
  ]
}
```

Message

$subject refers to an element from channel.areas[i].subjects from the icube collection in the MongoDB database

$sender - see data dictionary description

```
{
        "_id" : ObjectId,
        "msi_key" : ObjectId,
        "title" : "title for alert and in app message display",
        "desc" : "body for alert and short desc for in app message display",
        "message" : "the actual message, only shown in app message display",
        "topic_key" : $subject.topic_key,
        "distribution" : $subject.opt.distribution,
        "sender" : $sender,
        "timestamp" : unix_timestamp
}
```

Data Dictionary

- i, j refer to valid indices in the areas and subjects arrays, respectively

icube

Attribute Name	Data Type/ Constraints	Description	Uses/Views
_id	MongoDB ObjectID	Unique identifier for channel	

name	String	Name of channel	Viewable in portal forms and mobile app
desc	String	Description of channel	Viewable in portal forms
access	Array	Identifiers of those who have access to channel and underlying areas and subjects	Can be used later for enhancement
areas	Array	The channel's underlying area objects	
areas[i].id	MongoDB ObjectID	Unique identifier for area	
areas[i].name	String	Name of area	Viewable in portal forms and mobile app
areas[i].desc	String	Description of area	Viewable in portal forms
areas[i].access	Array	Identifiers of those who have access to area and underlying subjects	Can be used later for enhancement
areas[i].subjects	Array	The area's underlying subject objects	
areas[i].subjects[j].id	MongoDB ObjectID	Unique identifier for subject	
areas[i].subjects[j].name	String	Name of subject	Viewable in portal forms and mobile app
areas[i].subjects[j].desc	String	Description of subject	Viewable in portal forms
areas[i].subjects[j].access	Array	Identifiers of those who have access to subject	Can be used later for enhancement
areas[i].subjects[j].topic_key	_id + "-" + areas[i].id + "-" + areas[i].subjects[j].id	Subscription (FCM topic) identifier	Used with FCM to relay messages

areas[i].subjects[j].opt	Object	Data regarding topic options	
areas[i].subjects[j].opt.level	Enumerated (“Forced” || “Recommended” || “”)	Indicates the optionality of subscribing to the topic. “Forced”: mobile app users are automatically subscribed and can not unsubscribe. “Recommended”: users are automatically subscribed but can unsubscribe. “”: default value, mobile app users are not automatically subscribed but can subscribe.	Used in mobile app in setup process and subscriptions display
areas[i].subjects[j].opt.distribution	Enumerated (“Alert” || “Information”)	Indicates the way messages from that topic are distributed. “Alert”: mobile app users receive both push notification and message in app “Information”: mobile app users silently receive message in app	Used to determine how to relay messages via FCM
areas[i].subjects[j].tags	Array	For associating topics with keywords	Can be used later for enhancement

message

Attribute Name	Data Type/ Constraints	Description	Uses/Views

_id	MongoDB ObjectID	Unique identifier for message document	
msi_key	MongoDB ObjectID	Identifier for relayed message	
title	String	Title of message	Displayed as title of mobile push notification if applicable and title of a message in mobile app
desc	String	Push notification body text, is “” for information messages	Displayed as body text of mobile push notification if applicable
message	String	Actual message text	Displayed as message text in mobile app
topic_key	String, Reference to areas[i].subjects[j].topic_key attribute in an icube document	Which FCM topic (subscription) the message was relayed to	Used with FCM for message relay and parsed in the mobile app to display channel, area, and subject information for a message
distribution	String, Value of areas[i].subjects[j].opt.distribution attribute in an icube document at time message was sent	Whether message was sent as type alert or information	
sender	String	User identifier of who sent the message retrieved from a session cookie (currently just "Communicator" or "Super User")	
timestamp	Unix timestamp integer	Time message created	Displayed as date/time of message in mobile app

Mobile App

Database

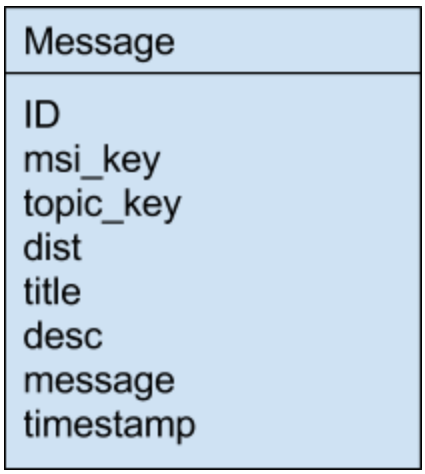
The mobile app uses a SQLite database, which is both SQL and file based. It is used to cache data on the client side of the Mobile Student Intranet system.

Schema

Table Creation SQL

```
CREATE TABLE IF NOT EXISTS Message (
	ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	msi_key VARCHAR(100),
	topic_key VARCHAR (100),
	dist VARCHAR (50),
	title VARCHAR(500),
	desc VARCHAR(500),
	message VARCHAR(1500),
	timestamp INTEGER
);
```

ER Diagram



Data Dictionary

Attribute Name	Data Type/ Constraints	Description	Uses/Views
ID	INTEGER, Primary Key, Auto Increment, Not Null	Auto-incremented primary key generated upon record insertion	Provide primary key for database storage
msi_key	VARCHAR(100)	Unique message identifier string made of hexadecimal characters	Used as unique identifier for messages lists in the app
topic_key	VARCHAR(100)	Unique identifier of the subscription (topic) a message was sent out with	Used to get Channel, Area, and Subject information to display and filter in message views
dist	VARCHAR(50)	The way a message was distributed, either “Alert” or “Information”	Used in app for visual indicator for alert messages in message views
title	VARCHAR(500)	Title of a message received, also the title from the push notification if the message was alert type	Displayed in message views
desc	VARCHAR(500)	The description text for alert type messages that shows in push notifications, is equal to an empty string (“”) for information messages	Currently not displayed within the app
message	VARCHAR(1500)	The actual message text part of a message	Displayed in message views
timestamp	INTEGER	Unix milliseconds timestamp of what time message was distributed from the communication hub	Used to sort messages by most recent and to display date time of when message received in message views

FCM Mapping

Overview of FCM

FCM works with both notification, data, and combined (hybrid notification) payloads in the HTTPS request body. Data only payloads deliver information silently while the others generate a pop-up (push notification). Combined payloads lose data if push notifications are dismissed by the user (i.e. the data portion will not be delivered if the notification is not opened). More on messages types can be found in the Firebase documentation here: https://firebase.google.com/docs/cloud-messaging/concept-options. FCM can relay messages to a subscription group, called a topic. A topic is not created until a device subscribes to it. More on topics can be found in the Firebase documentation here: https://firebase.google.com/docs/cloud-messaging/ios/topic-messaging.

Mapping

FCM topics map to a combination of a channel, area, and subject in the information cube (from the icube collection of the server database). As for messages, two kinds of messages can be sent out from the portal - alert and information messages. (The type of message depends on the distribution attribute of the subject under which a message is to be sent out). Alert messages both send push notifications/pop-ups and information to the mobile app. These messages consist of two separate HTTPS requests to FCM, one with a hybrid notification payload and one with just a data payload. Information messages send information silently to the mobile app and only involve one HTTPS request to FCM with a data only payload.

Message Format

- A component refers to one HTTPS request body
- $message refers to a document from the message collection from the server-side MongoDB

Alert Message

Notification Component

```
{
   "to" : "/topics/"+$message.topic_key,
   "priority" : "high",
   "notification" : {
      "title" : $message.title,
```

```
      "body" : $message.desc,
      "sound" : "default"
   },
   "data" : {
     "msi_key" : $message.msi_key
   }
}
```

Data Component

```
{
   "to" : "/topics/"+$message.topic_key,
   "priority" : "high",
   "content_available" : true,
   "data" : {
      "msi_key" : $message.msi_key,
      "topic_key": $message.topic_key,
      "dist" : $message.distribution,
      "title" : $message.title,
      "desc" : $message.desc,
      "message" : $message.message,
      "timestamp" : $message.timestamp
   }
}
```

Information Message

Data Component

```
{
   "to" : "/topics/"+$subject.topic_key,
   "priority" : "high",
   "content_available" : true,
   "data" : {
      "msi_key" : $message.msi_key,
      "topic_key": $message.topic_key,
          "dist" : $message.distribution,
      "title" : $message.title,
      "desc" : $message.desc,
      "message" : $message.message,
```

```
        "timestamp" : $message.timestamp
    }
}
```