Two possibilities for IoT Topic storage/retrieval.

1. We can build the topics by querying for the section, client, and device id then joining them into one topic - this requires a lot of joins.
2. We can make a topic name field in the Device table and store topic strings immediately when a device is registered

## 

## The Process(es)

1. Create a client
2. Create a device registration request
   1. Hit the endpoint/Lambda function to make the request
3. Approve the request by assigning it to a section
   1. Hit the endpoint/lambda function to approve the request
      1. Once the device is created we can take one of two paths
         1. The device is created with a device name, and section\_id.
            1. Thereafter we would only find the device topic by doing a series of joins(queries) to find the client\_id from the content or the section.
         2. The device is created with the topic name, device name and section id.
4. When subscribing to a topic a device can find it’s content by either:
   1. hit an endpoint/lambda for its own info, then subscribing to the relevant topic name.
      1. Taking the section id from the device, querying for all device content
         1. Use the section id from the device query then query for the section info to get the client id
            1. Then create the topic name from that combination
         2. Find all content ids that match the device\_id in the device\_content table
         3. Load all related active content onto the screen
      2. Query the device\_table using the device name and Take the section id from and the topic name from the device info
         1. Find the device\_content and find the related content from those content ids
         2. load them onto the screen

## The Solution

1. Active, inactive, saved fields in device\_content table
2. Topic name field in device table
3. Queue in smart tv app to keep track of active and inactive content

## Device Registration -

1. From the smart TV App we will want to know if a device has been registered
   1. Check for two things
      1. If a device exists
         1. If yes, then load the content card display screen
      2. If not then check if the device
         1. If yes then check its approval status
            1. If it’s pending then load the pending screen
            2. If it’s denied then load the request denied screen
   2. On the pending screen, subscribe to an IoT topic (i.e 1/#/3) and listen for an approval flag
   3. On the device content card display screen then make a request to the API to load all of the active content for that device.
      1. Also, subscribe to the relevant topic for that device (i.e 1/2/3)
   4. On the denied screen, still listen to the IoT topic from part b. Just in case the approval status changes

## Helpful research -

<https://stackoverflow.com/questions/37810289/how-can-i-publish-to-a-mqtt-topic-in-a-amazon-aws-lambda-function>

Maybe have the tv pull on an iot publish

Iot publish, makes the tv do a pull from the lambda

Have a tv display a max of 10 and just scale

The lambda retrieves all cards by the tv that are in the db device content

New Solution for saving content pushing etc.

New saves, set status, and calls publish

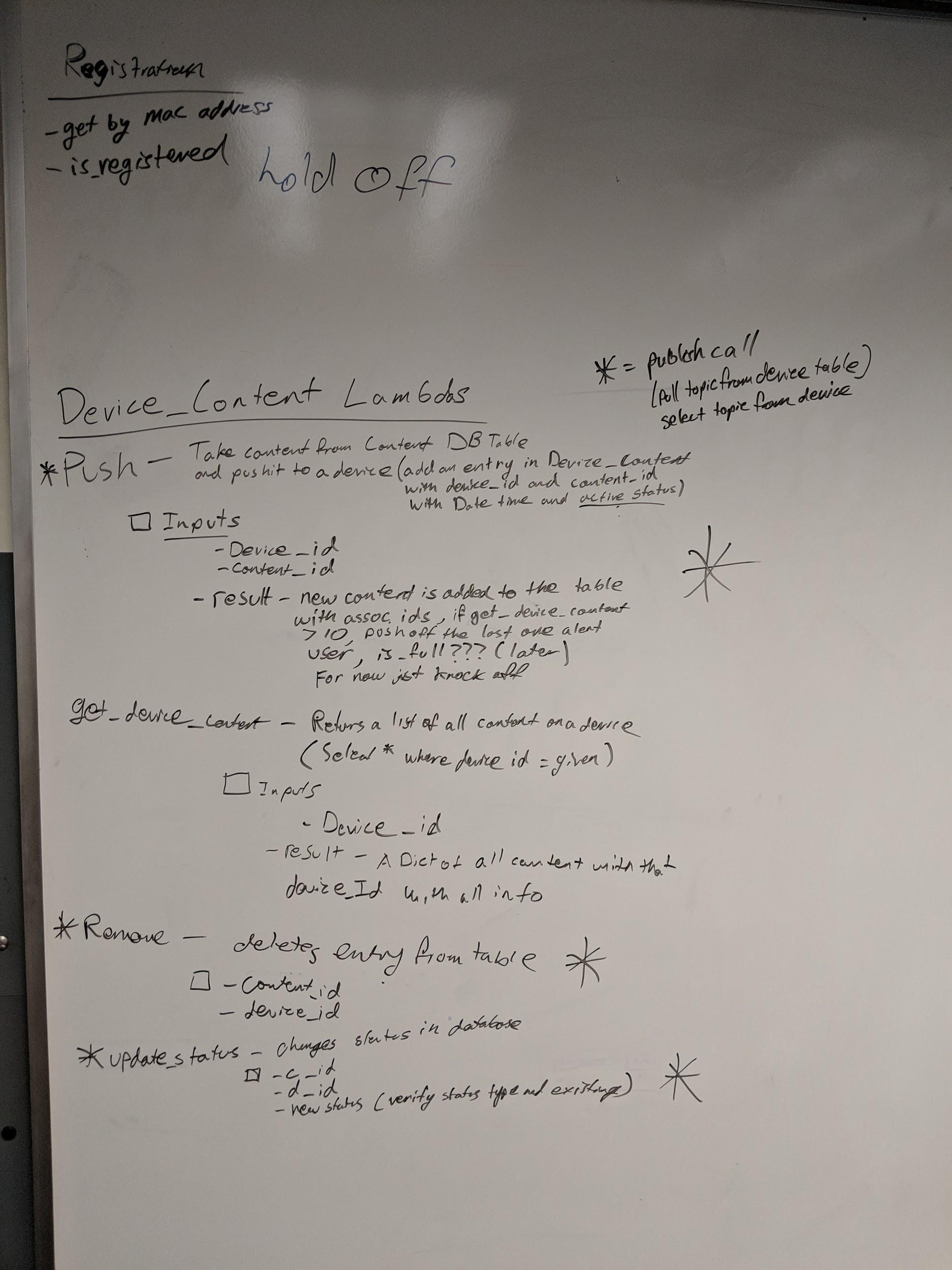
Update status uses the the id and calls publish to the specific topic from the device id

Store unique andorid id in the db instead of mac address to register the device in the system

May be best to create a guid and store that? Idk

<https://stackoverflow.com/questions/33842404/dynamically-add-more-cards-in-a-list-row-android-tv-leanback>

^for dynamic cards?



Get device content should optionally take in mac address

## For announcements/broadcasts

The topic structure for broadcasting messages to all devices in a section could be ‘client\_id/section\_id/announcements’. We can publish and subscribe to this topic and all devices would receive message broadcasts.

Likewise for broadcasting throughout an entire client we would want ‘client\_id/announcements’. And have the web app publish to this topic and all devices subscribe to it.