

# Final Project - Out of Order Book Finder

Fall 2021 - CS437 Internet of Things

Adam Goodell (goodell5) / Derek Chapman (derek4) / Team Skynet

Team Video: [https://mediaspace.illinois.edu/media/t/1\\_8f0quvgb](https://mediaspace.illinois.edu/media/t/1_8f0quvgb)

Team Repo: <https://gitlab.com/goodell5/cs437-iot-labs/-/tree/main/FinalProject>

## Objective

We attempt to create a solution to find and report on books that are out of order in a library stack. Our solution consists of a small RC like car as a moveable camera platform, a small pi-camera, an onboard Raspberry Pi for image processing and logic, and a frontend being hosted on Amazon Web Services.

## Motivation

Each year most libraries across the country perform 'book count' where they manually verify that each book is on the shelf and in the correct order. Even for a small town library this relies on dozens of volunteers contributing hundreds if not thousands of hours. Now imagine a major library such as the Notre Dame library that is 14 stories tall comprising miles of shelves and hundreds of thousands of books. It would be a nearly insurmountable task to check each book manually every year.

## Technical Approach

In general our system is split into two parts. The first part is the car, picture taking, image processing, and logic of determining out of order books. The second part is the cloud storage of the information and photos, parsing of information, and presentation to the user.

### Basic Steps in the Pipeline

1. RC car maintains a distance of 15cm from the shelf and moving forward about 3-6 inches after each photo to ensure overlap between pictures.
2. The camera takes a picture of the bookshelf. The image is converted into the BGR color space for easier processing with OpenCV which defaults to that color space, and then is also converted into a low fidelity grayscale image.
3. Each image is then sent through a processing phase consisting of a number of methods to prep the image for use with OCR.
4. Once we have all of the labels these are then sent through the OCR library Tesseract using the

```
label2string(labels[2])  
✓ 0.4s  
'R\n612\nNAG\n\x0c'
```

Pytesseract bindings. Tesseract extracts text from the image (book label) and sends it back as a string.

5. Once the text is extracted from each label it is then passed onto a method to clean up the string, split it into chunks, and find the actual call number. We decided to focus on only nonfiction call numbers for this project to make sure we had a consistent format. All call numbers are in the format of (“###.#”, “AAA”).
6. The call numbers are prepped by removing duplicate neighbor entries. They are then processed to find call numbers that are out of order.
7. A list of dictionaries is created to hold information about each book such as the call number, shelf number, and out of order flag.

```
{'shelf': 1, 'book_number': 1, 'call_number': ('612', 'NAG'), 'out_of_order': False}  
{'shelf': 1, 'book_number': 2, 'call_number': ('801.1', 'TED'), 'out_of_order': True}  
{'shelf': 1, 'book_number': 3, 'call_number': ('612', 'BIL'), 'out_of_order': False}
```

8. Once this information is created and packaged it is passed onto our AWS instance.
9. The payload is sent via MQTT to IoT Core under the topic ‘scanner/books’ with a unique ID for the shelf object.
10. The payload is inserted into an AWS database to be queried by our React.js app. Images of the bookshelf are also uploaded to AWS storage.
11. To access the app we setup user authentication so that users can create an account with their email address then login to view their bookshelf data.
12. Our React.js app queries the database for the bookshelf records, parses the data then displays each shelf with its corresponding book data in the front-end.

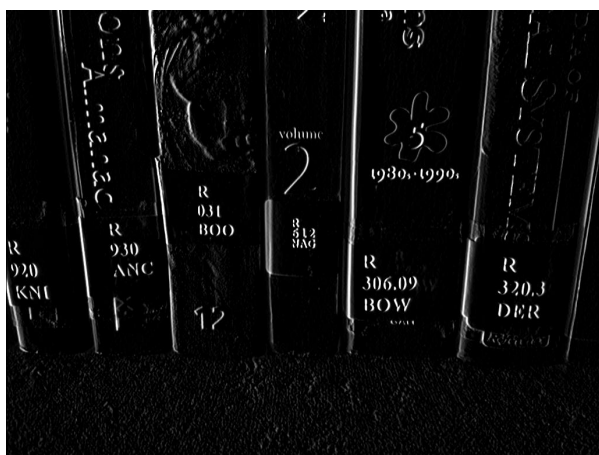
## Implementation Details

### Step 3 - Image Processing

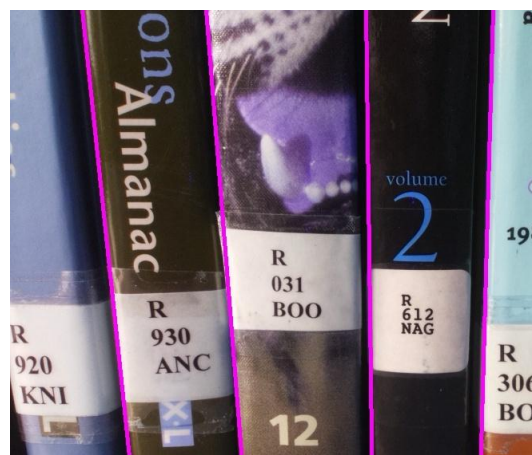
This step consists of:

- a. Sobel edge detection using OpenCVs Sobel method. We keep only the vertical edges since we are looking to split the picture into ‘slices’ later in the process corresponding to a single book. *Figure 3*
- b. Line detection is then performed on the image using OpenCVs Probabilistic Hough Lines function. The parameter search for this step was extensive to make sure we are only getting lines that correspond to the space between books. Unfortunately this method is extremely finicky and needed a lot of iterations to get to a decent state. We decided to have Hough Lines find more lines than needed and then to remove extra/unnecessary lines via our own method rather than trying to tweak the Hough Lines too much and lose out on some book slices. However in the beginning Hough Lines would find far too many lines including some nonsensical lines running at angles between disconnected points.
- c. The lines are then extended from the top of the image to the bottom to ensure a full cut between books. *Figure 4*
- d. Duplicate lines are then removed as the Hough Lines method would often find lines directly next to each other in the same section of the edge detected image. *Figure 6*
- e. The image is then cut up into slices that contain an individual book. *Figure 7*

- f. From each slice we use a probabilistic approach to extract the label from the spine of the book. The image is first converted into the HSL image space (Hue, Saturation, Light). When finding the labels in each HSL slice we look at the overall whiteness of the image and then run a small kernel down the picture to find the most likely area that the label starts and finishes. We look at the 95th percentile of 'whiteness' in the picture as an indication of the location of the label. *Figure 8*



*Figure 3*



*Figure 4*



*Figure 5*

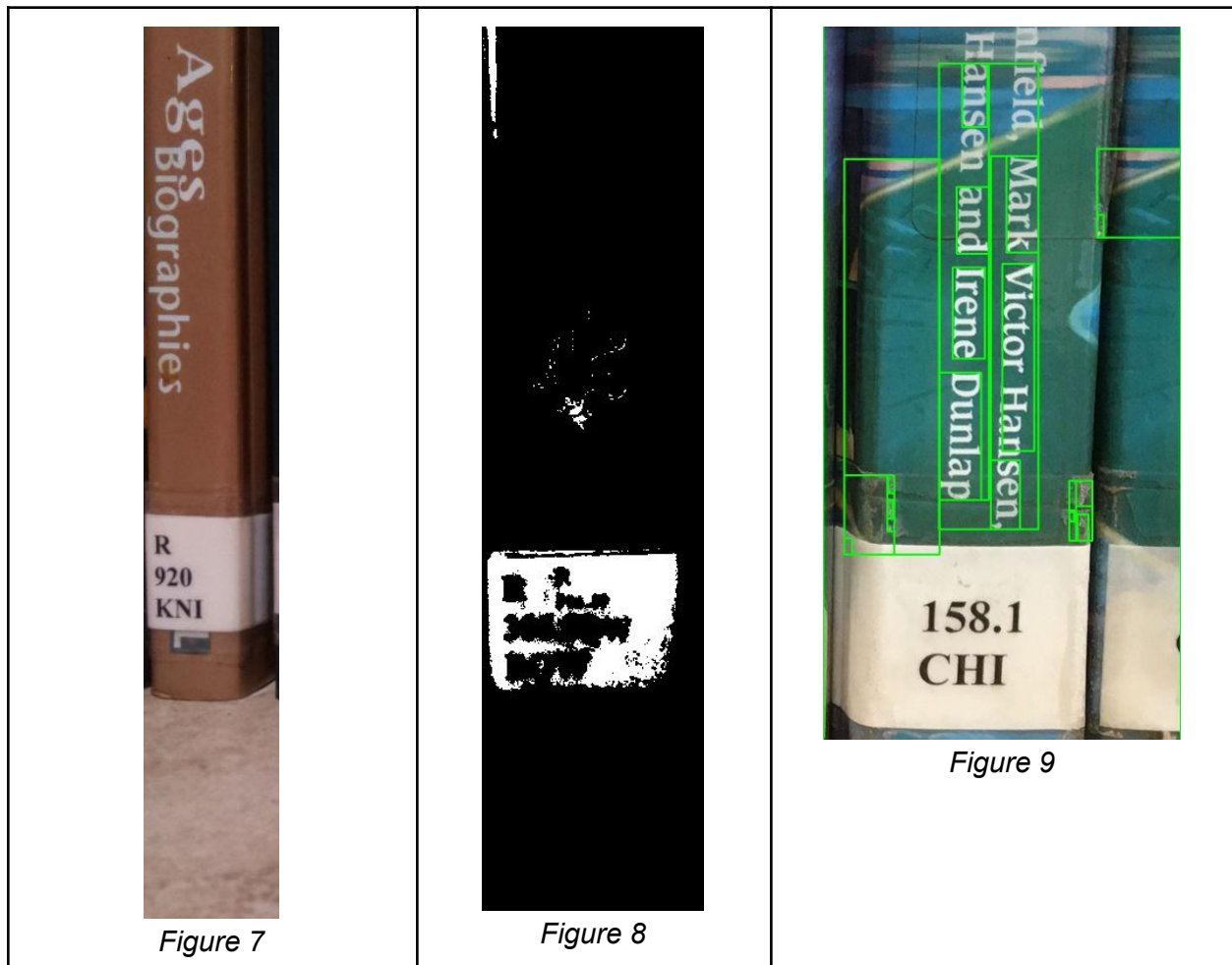


*Figure 6*

#### **Step 4 - Tesseract**

A working setup for this step took a lot of time as Tesseract had an extremely difficult time reading the book labels. Although it is black text on a white background the initial accuracy was less than 10%. A number of image manipulations were tried including, white balancing, sharpening/blurring, various color spaces, lightening/darkening. We also performed extensive

testing on the available parameters of Tesseract to eventually find a setup that provided higher accuracy. Unfortunately even with all of our tweaks Tesseract still frequently fails to find any text in a well cropped and well-lit image of the label. For example in *figure 9* we can see that the label text is well lit, extremely legible and yet tesseract still fails to even recognize it as text. In the screenshot you can see the green bounding boxes for other text including vertical text and also false positives but nothing on the label itself.



### Step 6 - Call Numbers

We remove duplicate neighbor call numbers as we can't be sure, without further processing, whether they are the same book from overlapping pictures, or it's a popular book and there are multiple copies of the same book with the same call number. Because of this we decided to remove duplicate neighbors and treat each call number as being only one book. This still allows us to find the same book if it is out of order and in a different location than its twin.

## Step 8 - AWS Infrastructure

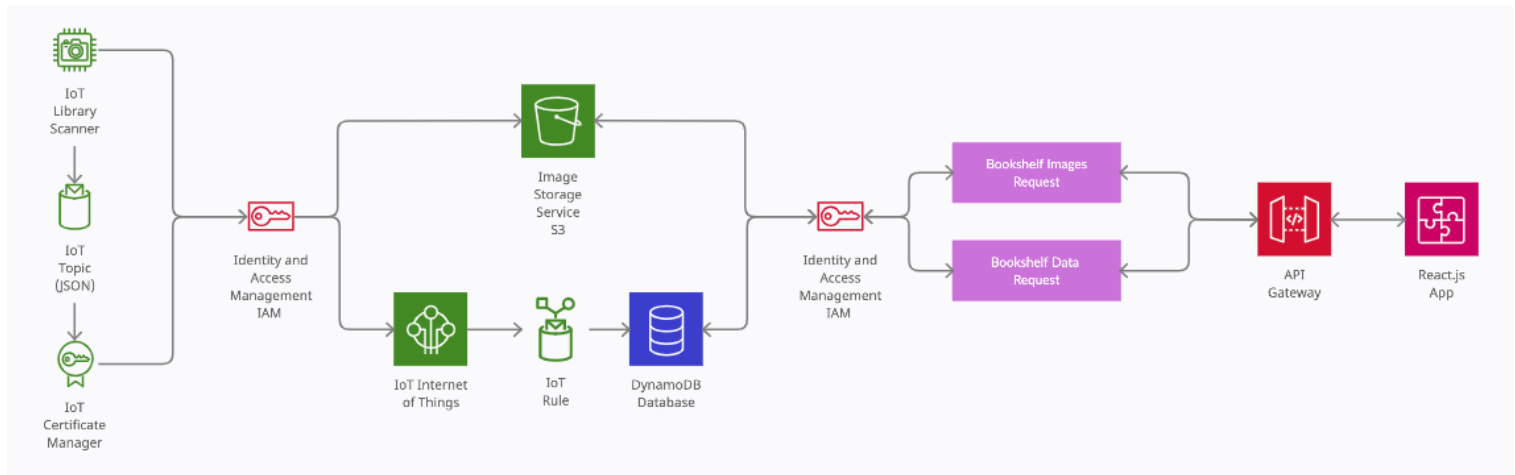


Figure 10

## Step 9 - AWS IoT MQTT

We created a Thing in IoT Core, then created and downloaded certificates to our device (Raspberry Pi). The Thing has a policy attached to publish MQTT messages to IoT Core under the topic 'scanner/books'. Images taken by the Pi's camera and used in OCR are also uploaded to an AWS S3 bucket for each bookshelf (see *Figure 4* above).

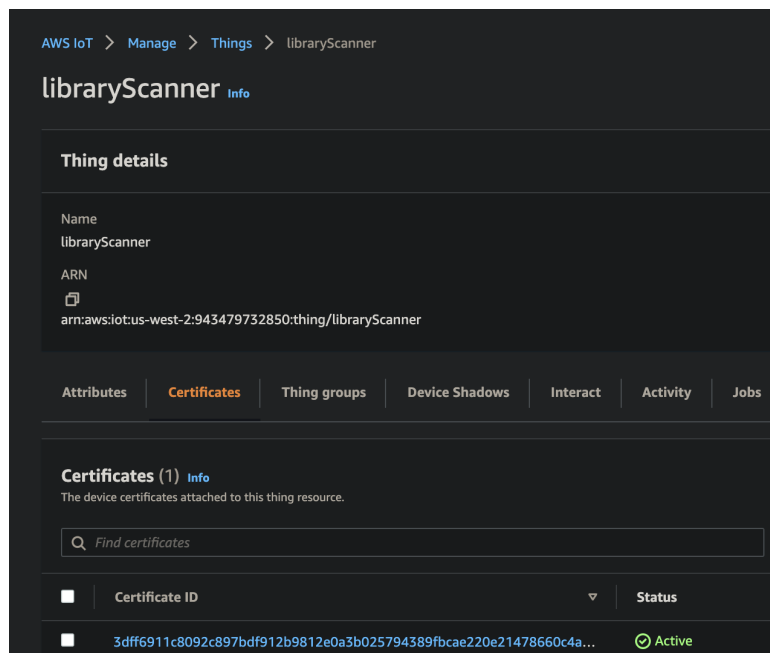


Figure 11

AWS IoT > MQTT test client

MQTT test client

Info

You can use the MQTT test client to monitor the MQTT messages being passed in your AWS account. Devices publish MQTT message topics and publish MQTT messages to topics by using the MQTT test client.

Subscribe to a topic

Publish to a topic

Topic filter

Info

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

scanner/books

► Additional configuration

Subscribe

Subscriptions

scanner/books

scanner/books

♡ ×

▼ scanner/books

{  
 "id": 1639196214,  
 "shelf": [  
 {  
 "id": 1639196214,  
 "shelf": 1,  
 "book\_number": 1,  
 "call\_number": [  
 "160.9",  
 "ADR"  
 ],  
 "out\_of\_order": false,  
 "updatedAt": "2021-12-10T20:16:54.091227Z"  
 },  
 {  
 "id": 1639196214,  
 "shelf": 1,  
 "book\_number": 2,  
 "call\_number": [  
 "204",  
 "DER"  
 ],  
 "out\_of\_order": false,  
 "updatedAt": "2021-12-10T20:16:54.091236Z"  
 }  
 ]  
}

Figure 12

Amazon S3 > skynetlibrarybucket

skynetlibrarybucket Info

Objects

Properties

Permissions

Metrics

Management

Access

Objects (5)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get

Copy S3 URI

Copy URL

Download

Open

Find objects by prefix

	Name	Type
<input checked="" type="checkbox"/>	bookshelf000/	Folder
<input checked="" type="checkbox"/>	bookshelf001/	Folder
<input type="checkbox"/>	s3_sample_upload_1.png	png
<input type="checkbox"/>	s3_sample_upload_2.png	png
<input type="checkbox"/>	s3_sample_upload_3.png	png

Figure 13

## Step 10 - AWS IoT Payload to Database

We use an IoT Rule to send the message from IoT Core to a DynamoDB database. Message contents are parsed to update the database table correctly.

The screenshot shows the AWS IoT console interface for configuring a rule named 'libraryScannerToDynamo'. The breadcrumb navigation at the top reads 'AWS IoT > Rules > libraryScannerToDynamo'. The rule is currently 'ENABLED'. On the left, there is a sidebar with 'Overview' and 'Tags' tabs. The main content area is divided into sections: 'Description' (with an 'Edit' link and the text 'No description'), 'Rule query statement' (with an 'Edit' link and a SQL query: `SELECT * FROM 'scanner/books'`), and 'Actions'. The 'Rule query statement' section includes a note 'The source of the messages you want to process with this rule.' and specifies 'Using SQL version 2016-03-23'. The 'Actions' section explains that actions occur when a rule is triggered and lists two existing actions: 'Split message into multiple columns of a Dyna...' and 'Insert a message into a DynamoDB table', both associated with the resource 'Bookshelf-dczkwezjvvebbpu5hlekgtcawe-staging'. Each action has 'Remove' and 'Edit' links. An 'Add action' button is located at the bottom of the actions list.

AWS IoT > Rules > libraryScannerToDynamo

RULE

### libraryScannerToDynamo

ENABLED Actions ▾

**Overview**

**Description** Edit

No description

**Rule query statement** Edit

The source of the messages you want to process with this rule.

```
SELECT * FROM 'scanner/books'
```

Using SQL version 2016-03-23

**Actions**

Actions are what happens when a rule is triggered. [Learn more](#)

- Split message into multiple columns of a Dyna... Remove Edit ▸  
Bookshelf-dczkwezjvvebbpu5hlekgtcawe-staging
- Insert a message into a DynamoDB table Remove Edit ▸  
Bookshelf-dczkwezjvvebbpu5hlekgtcawe-staging

[Add action](#)

Figure 14

In terms of schema, each bookshelf has a unique key/ID along with a list of books. Each book also contains a unique key/ID. All records include timestamp information for sorting records chronologically. We keep the payload string in the bookshelf as it is used for parsing. We encountered issues with data types, parsing, and storage so we retain data that is required at various stages within our data processing pipeline.



```
type Bookshelf @model {
  id: ID!
  name: String
  payload: String
  books: [Book]
}

type Book @model {
  book_number: Int!
  call_number: String!
  out_of_order: Boolean!
  createdAt: AWSDateTime
}
```

Figure 15

### Steps 11 & 12 - AWS Amplify (React.js App)

We also developed a web application for users to view the results of the book scanning device. The app supports mobile and desktop devices. Our app was deployed with:

- React.js and AWS Amplify libraries for the front-end components
- A custom GraphQL API, DynamoDB and S3 for the backend components

AWS Amplify helped us implement user authentication in which users can create an account with their email address then login to our app to view their library's bookshelf data.

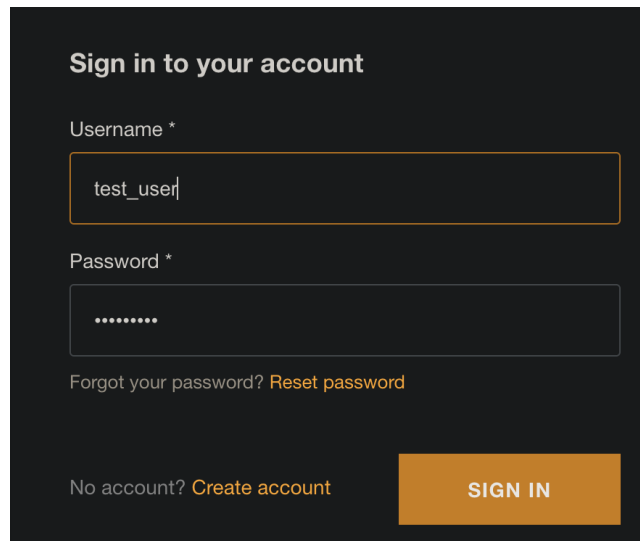
A screenshot of a web application's sign-in page. The page has a dark background. At the top, the text "Sign in to your account" is displayed in white. Below this, there are two input fields. The first is labeled "Username \*" and contains the text "test\_user". The second is labeled "Password \*" and contains a series of dots. Below the password field, there is a link that says "Forgot your password? Reset password". At the bottom left, there is a link that says "No account? Create account". At the bottom right, there is a large orange button with the text "SIGN IN" in white capital letters.

Figure 16

Our app queries the database for bookshelf records, parses the data then displays each shelf with its corresponding book data. Books are numbered according to their counted position within the shelf. Call numbers are also displayed, which are used to determine whether the book is in order or not. If a book is found to be out of order, a bold red message is displayed to the user.

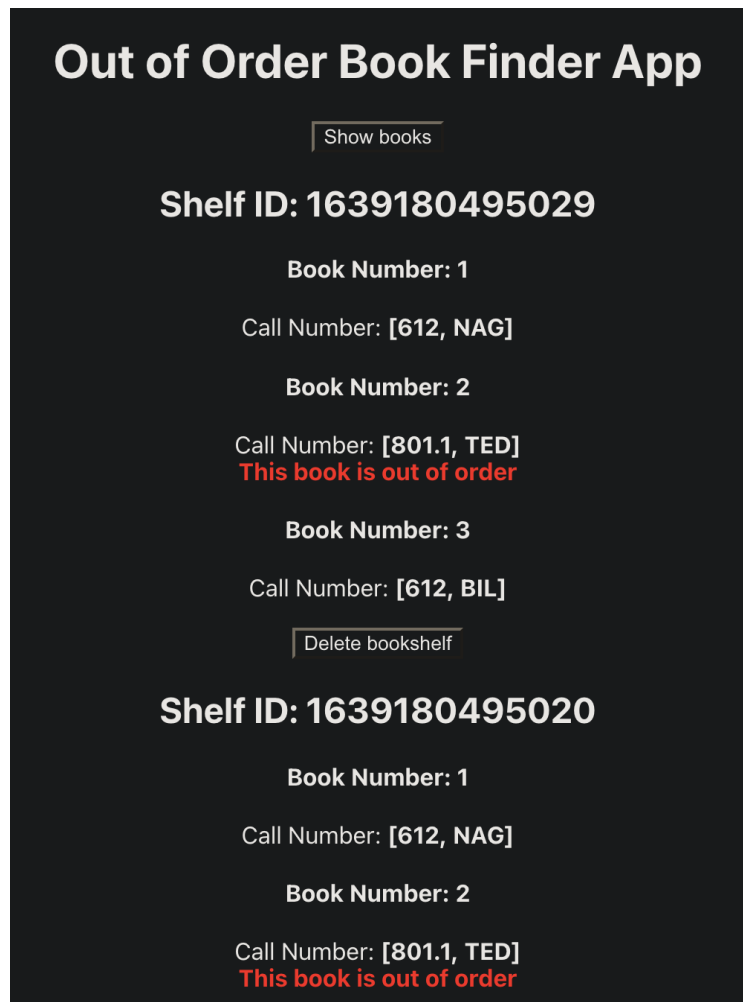


Figure 17

If no books are found out of order, the bookshelf data is still presented to the user to provide info surrounding books that are out of order. If a librarian processed the shelf and resolved the book that was out of order, they can return to the bookshelf in the app and click the *Delete Bookshelf* button to remove the bookshelf from their list of bookshelves to be sorted.

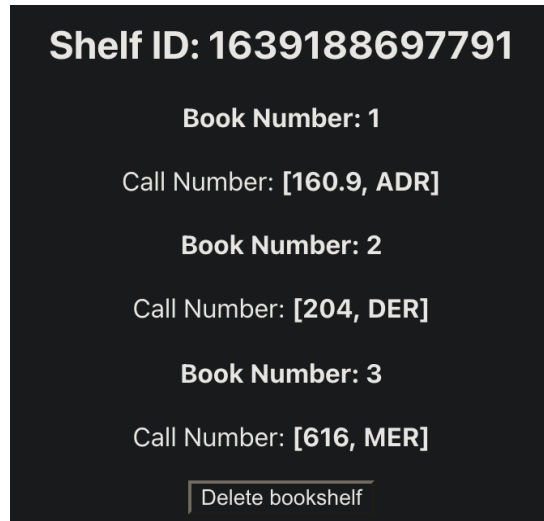


Figure 18

## Results

In an ideal situation our framework functions well. However it is not robust enough to handle the many different edge cases and issues present in a real environment. In our testbed our methods can correctly identify about half of the presented books.

## Learning

One of the biggest takeaways was in project management involving IoT. This project was different than many of the projects we have undertaken in other classes in this program in that it was a number of different devices put together. Each part of the entire pipeline had its own issues and friction points. Putting many small parts together was more difficult than trying to troubleshoot a single large method/program. In the initial planning phases we had many great ideas and mapped out how each part would connect. But as we began to run into issue after issue it became apparent that we would need to cut features. This was an eye opening experience and shed some light on why some software releases may get delayed for months or years.

## Further Development and Edge Cases

There are a number of places where this project could be improved. The biggest impact area would be in finding the label. Given more time training a CNN to find the labels may be a better route rather than trying to programmatically cut up the image by stringing methods together.

This could alleviate some of the edge cases that we ran into and were unable to solve such as:

- White label on a white book spine
- Labels that are partially wrapped around the edges of a skinny book
- Unable to find a vertical cut line between two books with dark covers
- Books that have a shiny book cover creating false positives in the thresholded white image

Our method of determining the direction of the book order (ascending or descending call numbers) is too simplistic for some possible edge cases and could be made more robust. For example:

- If there are two of the same book at the beginning of the shelf
- If the first two books on the shelf are out of order

Additionally further investigation into dealing with multiples of the same book would be helpful. Our method was to ignore them, however this may not be the best option or desired by an end user. Discussion with a library staff would be beneficial in determining how to handle this frequent case.

Our database schema, API, and AWS infrastructure could be improved further to minimize parsing and optimize query results for large scale libraries. We would also like to display the image data in the app to assist the user with validating results. Another thing to consider is filtering topic data if there are multiple library scanning devices being used within one library. Our app's user settings could also be expanded to support organizations, including group roles and permissions.

## Sources

- PiCamera Recipes - <https://picamera.readthedocs.io/en/release-1.10/recipes1.html>
- Use an Existing S3 Bucket or DynamoDB Table for your Amplify Project - <https://aws.amazon.com/blogs/mobile/use-an-existing-s3-bucket-for-your-amplify-project>
- AWS Diagram - <https://creately.com>