CHAPTER 9: IMPLEMENTATION

Once again stressing that the mobile application is only used to display and show the results created by the tracking system.

9.1 Screenshots

9.1.1 Screenshots for login page

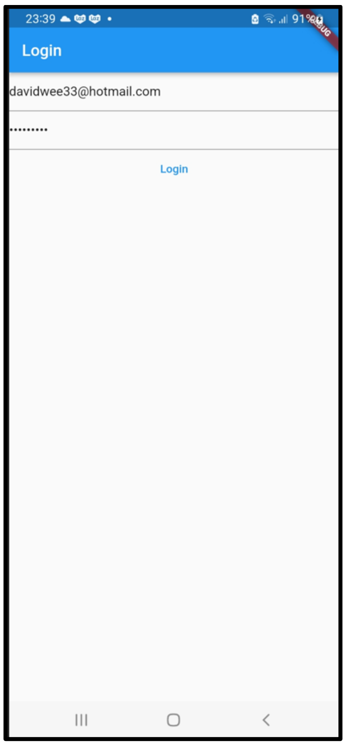


Figure 1: Implemented Login Page

In the login page, users are required to enter the correct credentials (email & password) to access the features of the application.

9.1.2 Screenshots for location page

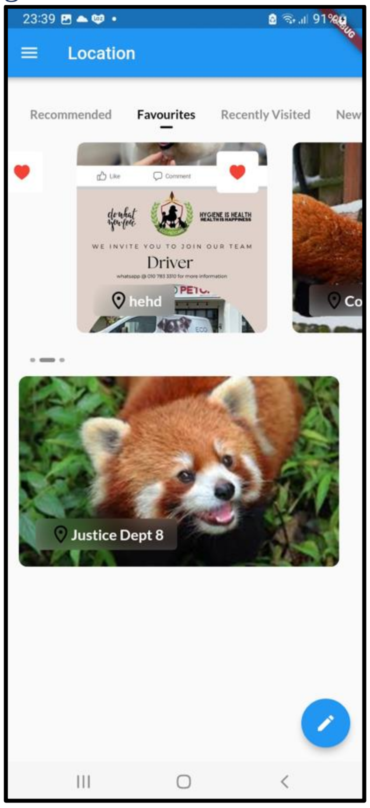


Figure 2: Implemented Location Page

In this page, there are two sections displaying the locations, the top and bottom. The top section compromises of locations that adheres to the labels above (“recommended”, ”favourites”, …) Whilst those that do not met the requirements are left at the bottom section. For instance, by tapping the heart icon, the location will become one of the user’s favourite or not and be relocated depending on its favouritism status when favourite label is selected.

Starting from this page, users are able to tap on the hamburger menu, or also known as the toggle menu to access user registration page. The pen icon below allows users to access the location registration page. Last but not least, by tapping any of the locations, the user is able to access the page exclusive to the selected location.

9.1.3 Screenshots for Hamburger Menu

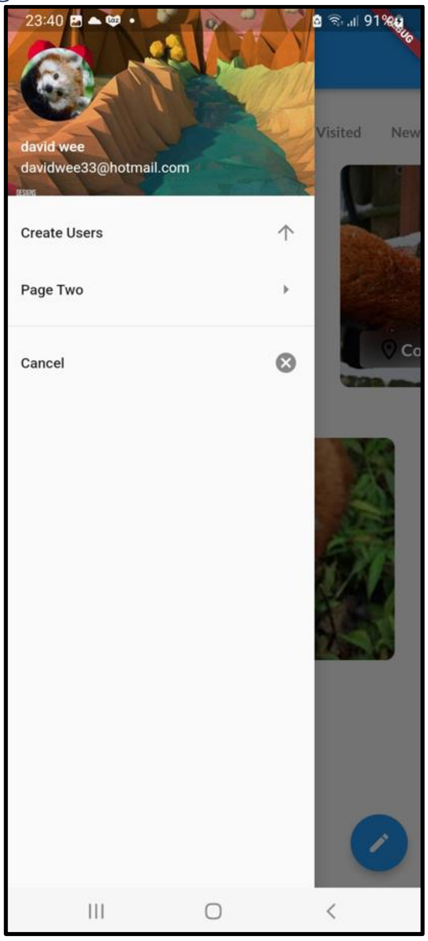


Figure 3: Implemented Hamburger Menu

As previously mentioned, user may access the user registration page through here. At the same moment, user may view its basic details such as their name, account picture and email.

9.1.4 Screenshots for user registration page

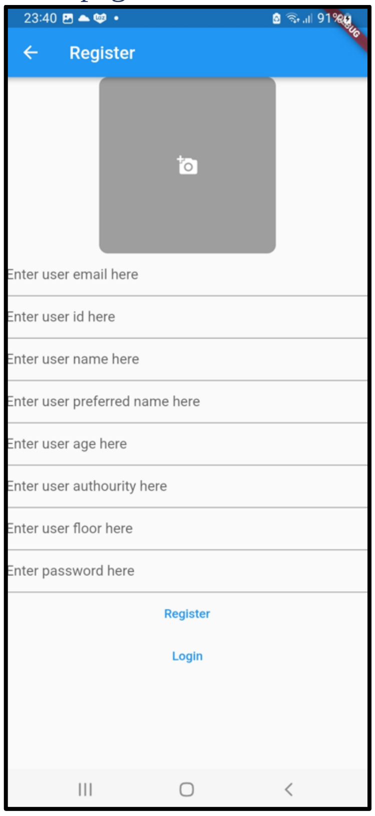


Figure 4: Implemented User Registration Page

In this page, users may register other users, so that they may log into the application. Credentials include image of user, email, id, name, preferred name, age, authority, floor, and password.

9.1.5 Screenshots for location registration page

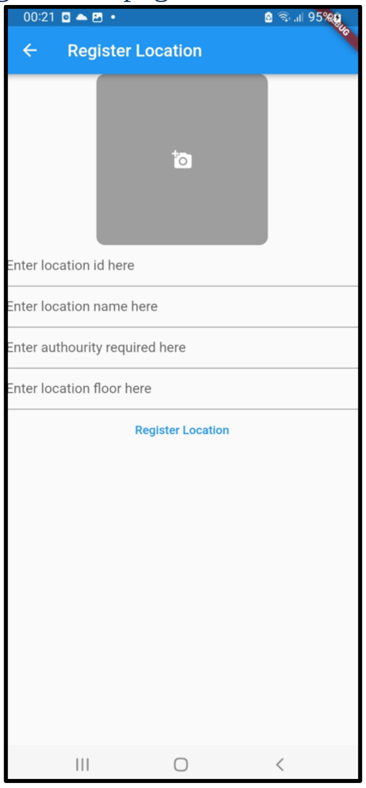
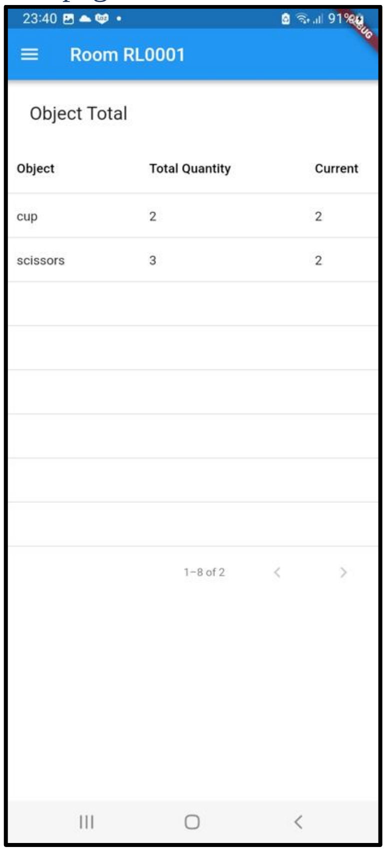


Figure 5: Implemented Location Registration Page

Almost identical to user registration page, except for its attributes.

9.1.6 Screenshots for direct location page 1


Room RL0001

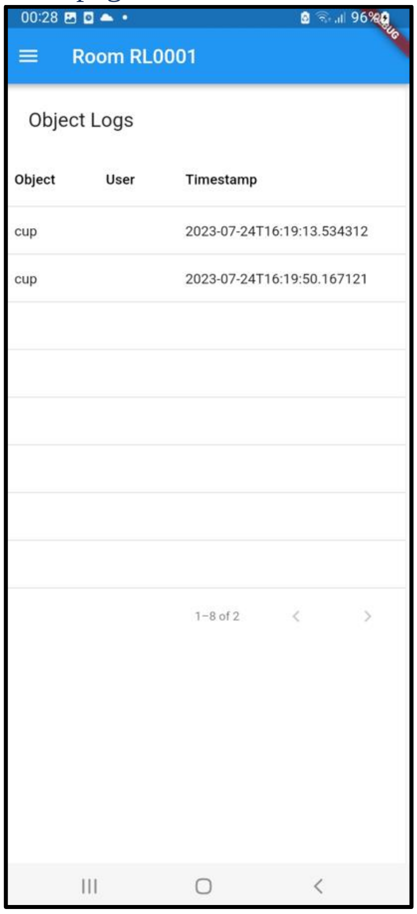
Object Total

Object	Total Quantity	Current
cup	2	2
scissors	3	2

Figure 6: Implemented Direct Location Page 1

In this page, users may view the object quantity details of the selected location. Users may access the other pages within this page by swiping left or right.

9.1.7 Screenshots for direct location page 2



Object Logs

Object	User	Timestamp
cup		2023-07-24T16:19:13.534312
cup		2023-07-24T16:19:50.167121

1–8 of 2

Figure 7: Implemented Direct Location Page 2

In this page, users may view the object logs of the selected location. The table is swipe-able which reveals an additional column(“Status”).

9.1.8 Screenshots for direct location page 3

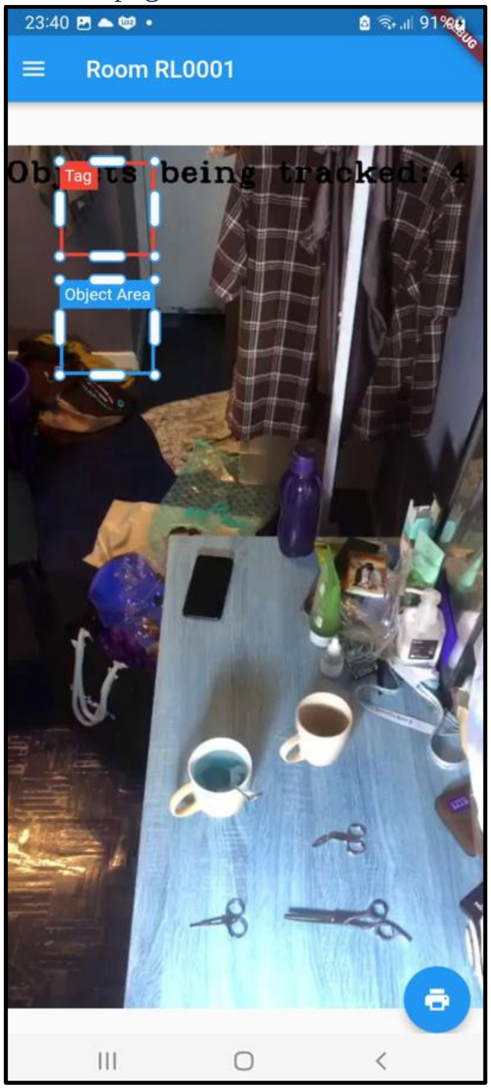


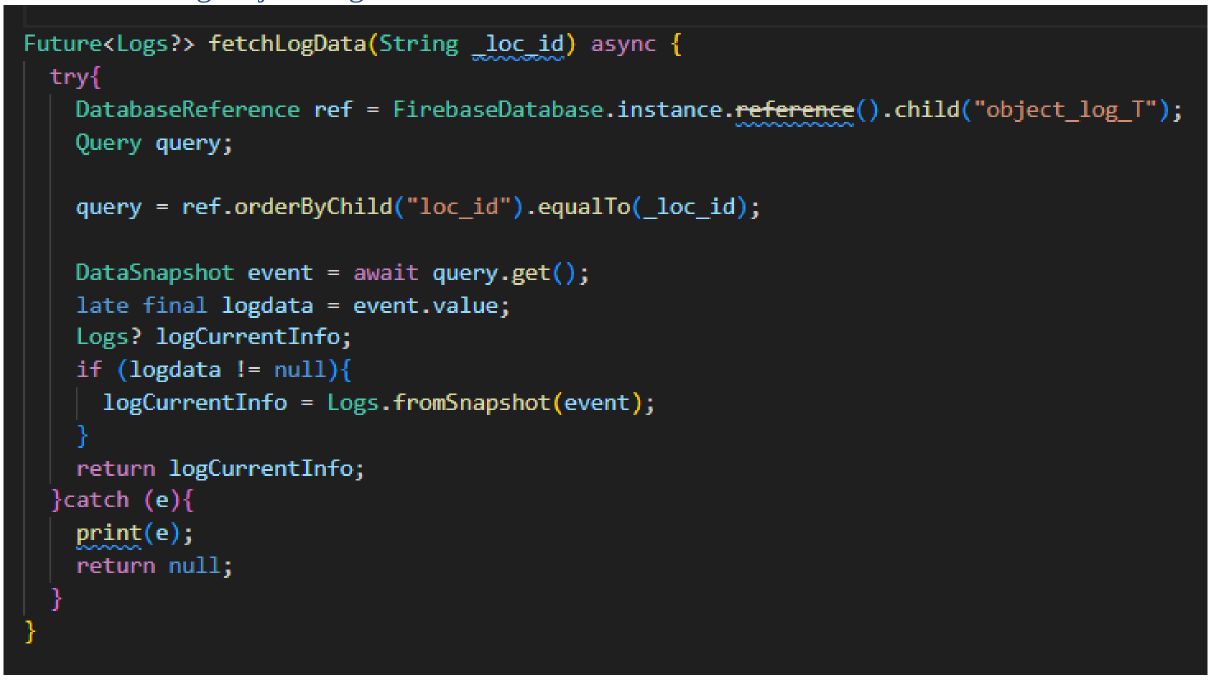
Figure 8: Implemented Direct Location Page 3

For the tracking system to be successful, this page is of utmost importance. This is due to the fact that objects outside the intended area may be processed through the system, which may result in accuracies and errors within the results produced. Additionally, with the absence of a NFC sensor, an area is used to mimic the area of which the NFC sensor will be placed. By properly placing the boxes and pressing the printer icon, the coordinate of the boxes will be transferred to firebase.

9.2 Sample Codes

9.2.1 Mobile Application Sample Codes

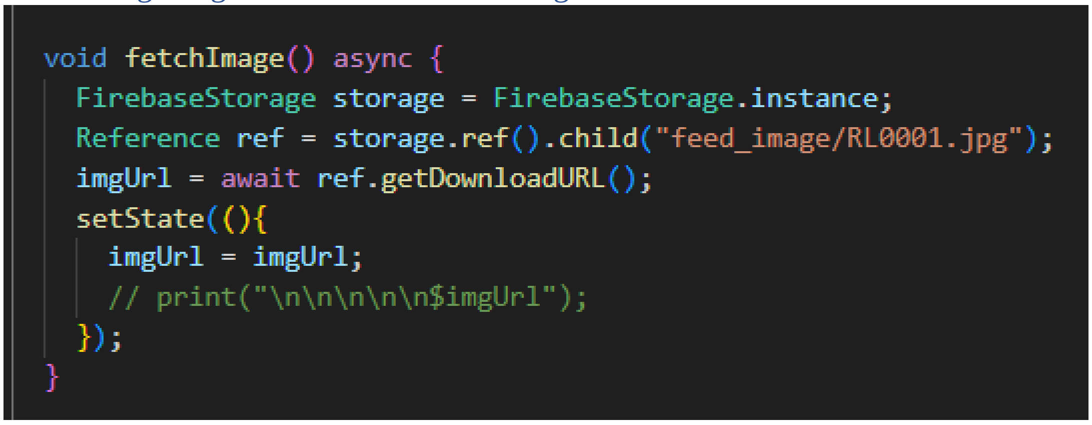
9.2.1.1 Fetching Object Log Data from Firebase Database



```
Future<Logs?> fetchLogData(String _loc_id) async {
  try{
    DatabaseReference ref = FirebaseDatabase.instance.reference().child("object_log_T");
    Query query;

    query = ref.orderByChild("loc_id").equalTo(_loc_id);

    DataSnapshot event = await query.get();
    late final logdata = event.value;
    Logs? logCurrentInfo;
    if (logdata != null){
      logCurrentInfo = Logs.fromSnapshot(event);
    }
    return logCurrentInfo;
  }catch (e){
    print(e);
    return null;
  }
}
```

Figure 9: Sample Code, Fetch Log Data from Firebase Database

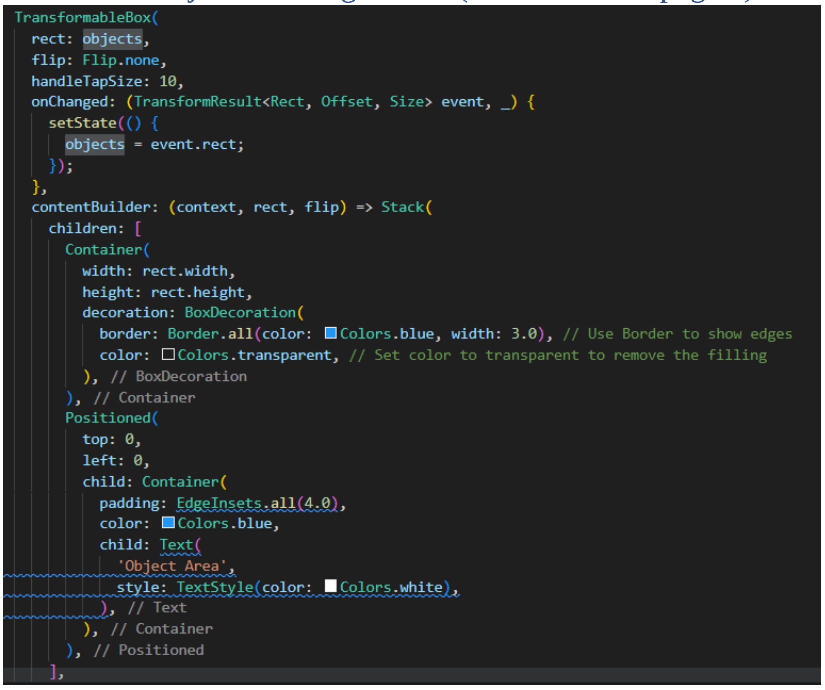
9.2.1.2 Fetching Image Url from Firebase Storage



```
void fetchImage() async {
  FirebaseStorage storage = FirebaseStorage.instance;
  Reference ref = storage.ref().child("feed_image/RL0001.jpg");
  imgUrl = await ref.getDownloadURL();
  setState((){
    imgUrl = imgUrl;
    // print("\n\n\n\n\n$imgUrl");
  });
}
```

Figure 10: Sample Code, Fetch Image Url from Firebase Storage

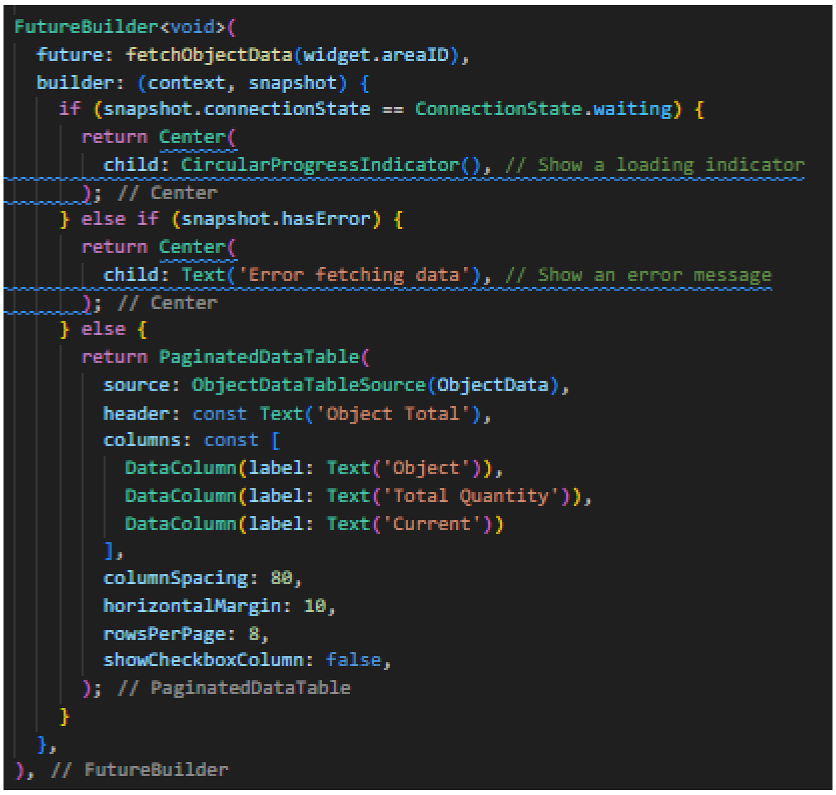
9.2.1.3 Movable Boxes in Adjustable Target Area (direct location page 3)



```
TransformableBox(
  rect: objects,
  flip: Flip.none,
  handleTapSize: 10,
  onChanged: (TransformResult<Rect, Offset, Size> event, _) {
    setState(() {
      objects = event.rect;
    });
  },
  contentBuilder: (context, rect, flip) => Stack(
    children: [
      Container(
        width: rect.width,
        height: rect.height,
        decoration: BoxDecoration(
          border: Border.all(color: Colors.blue, width: 3.0), // Use Border to show edges
          color: Colors.transparent, // Set color to transparent to remove the filling
        ), // BoxDecoration
      ), // Container
      Positioned(
        top: 0,
        left: 0,
        child: Container(
          padding: EdgeInsets.all(4.0),
          color: Colors.blue,
          child: Text(
            'Object Area',
            style: TextStyle(color: Colors.white),
          ), // Text
        ), // Container
      ), // Positioned
    ],
```

Figure 11: Sample Code, Adjustable Boxes (TransformableBox)

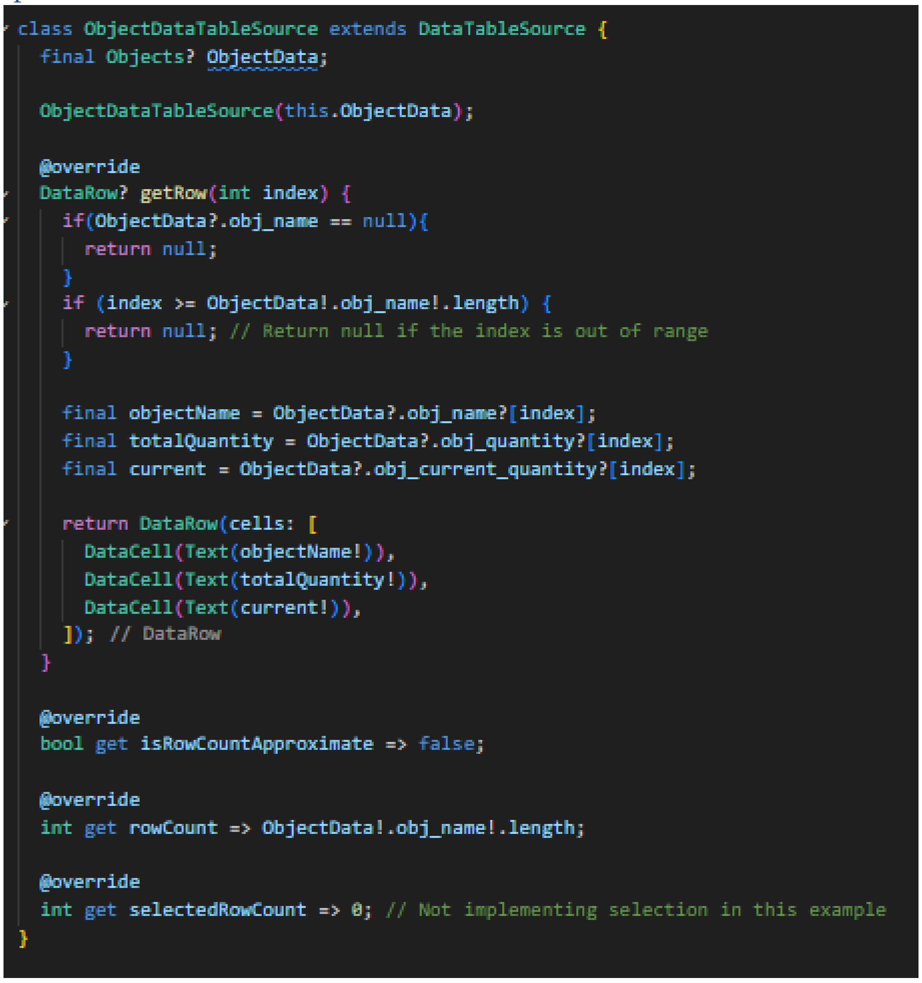
9.2.1.4 Sample Table



```
FutureBuilder<void>(
  future: fetchObjectData(widget.areaID),
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return Center(
        child: CircularProgressIndicator(), // Show a loading indicator
      ); // Center
    } else if (snapshot.hasError) {
      return Center(
        child: Text('Error fetching data'), // Show an error message
      ); // Center
    } else {
      return PaginatedDataTable(
        source: ObjectDataTableSource(ObjectData),
        header: const Text('Object Total'),
        columns: const [
          DataColumn(label: Text('Object')),
          DataColumn(label: Text('Total Quantity')),
          DataColumn(label: Text('Current'))
        ],
        columnSpacing: 80,
        horizontalMargin: 10,
        rowsPerPage: 8,
        showCheckboxColumn: false,
      ); // PaginatedDataTable
    }
  },
), // FutureBuilder
```

Figure 12: Sample Code, Object Table Display

9.2.1.5 Sample Table Source



```
class ObjectDataTableSource extends DataTableSource {
  final Objects? ObjectData;

  ObjectDataTableSource(this.ObjectData);

  @override
  DataRow? getRow(int index) {
    if(ObjectData?.obj_name == null){
      return null;
    }
    if (index >= ObjectData!.obj_name!.length) {
      return null; // Return null if the index is out of range
    }

    final objectName = ObjectData?.obj_name?[index];
    final totalQuantity = ObjectData?.obj_quantity?[index];
    final current = ObjectData?.obj_current_quantity?[index];

    return DataRow(cells: [
      DataCell(Text(objectName!)),
      DataCell(Text(totalQuantity!)),
      DataCell(Text(current!)),
    ]); // DataRow
  }

  @override
  bool get isRowCountApproximate => false;

  @override
  int get rowCount => ObjectData!.obj_name!.length;

  @override
  int get selectedRowCount => 0; // Not implementing selection in this example
}
```

Figure 13: Sample Code, Object Table's Data

9.2.2 Tracking System

In this segment, the working flow of the tracking system would be illustrated and elaborated as best as possible. For an in-depth explanation on the algorithm, please refer to the sample codes.

9.2.2.1 Before Yolov7 Commences

Before the Yolov7 performs its detection algorithm on the video, several declaration and processes must be done. Declaration includes connection to firebase database and storage, as well as the execution of threads. As for threads, there are 3 threads that are executed. One to monitors if personnel took and placed back an object of the same class, or if personnel took or placed an object but did not identify itself. The second thread listens for updates in the identification table, which is manually updated when a personnel taps onto a imitated NFC sensor. This thread specifically takes information of the personnel closest to the NFC sensor and updates his actions to firebase. The last thread is also a listener, but it awaits the coordinates for the target areas that is supposed to be updated by the mobile application.

9.2.2.2 Simple Pre-processing

The results procured by the Yolov7 is frame by frame, and the frame per second in this situation should be around 30. As a result, the developer has to properly organize the data so that it can be processed many at once. Thus, the data per frame is written into a txt file, which is then read and compiled within another txt files which will contain 20+ frames of data. Additionally, detection involving person is segregated from other detected classes. Throughout, the data is properly processed to ensure data is properly written and read. For instance, “[943534 4359 34953 345345]” is supposed to be a int list, but is now a string, with impurities that needs to be removed.

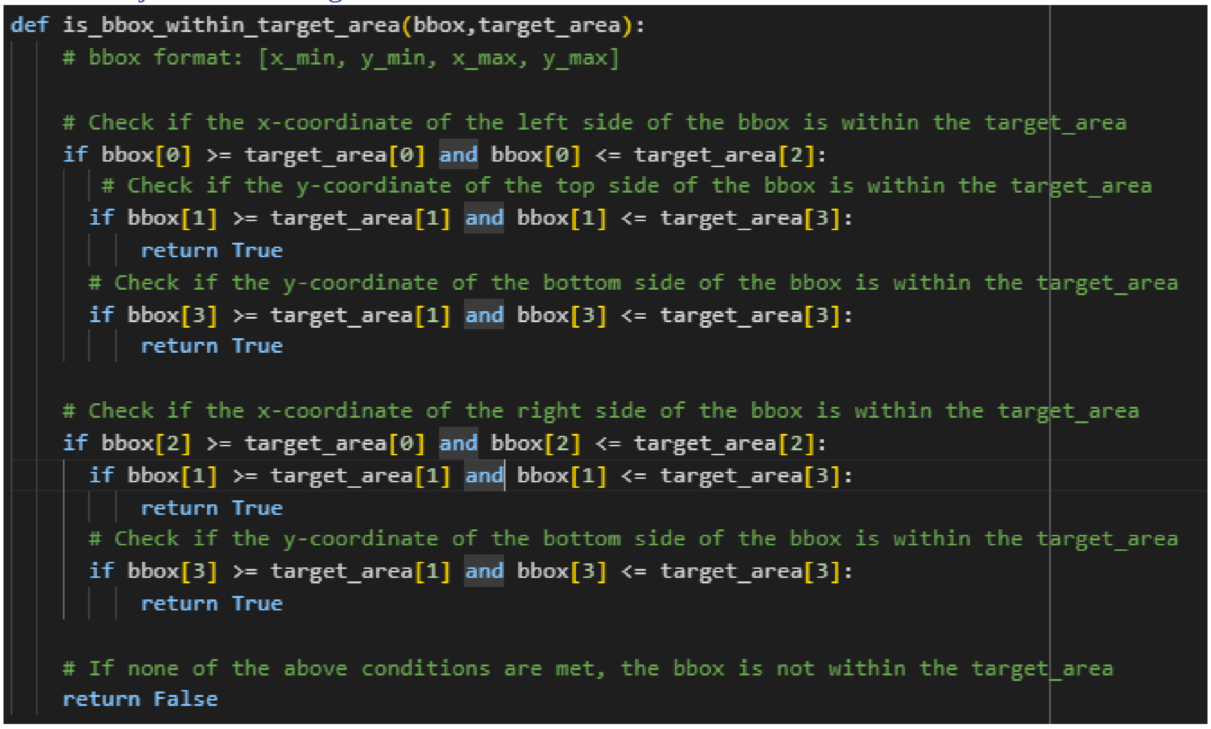
9.2.2.3 Number of Objects

During certain circumstances, false positive may occur, detecting an object that is non-existent or false negative, where an object that should be detected is not detected in certain frames. Thus, the reasoning for the processing of combined frames. Primarily, the xyxy coordinates of a singular detected class is grouped in quantity with an algorithm based on their locations. Simply put, it determines if an object in fact exists and was not detected due to the false positives or false negatives of Yolov7. Moreover, an object that has constantly been detected in previous frames suddenly stopped being detected can be easily noticed and pinpointed.

9.2.2.4 Object - Person Linking

By determining the current number object and the previous number of objects based on the frames, the system may determine if an object is taken or placed. As the system has been using the xyxy coordinates to pinpoint the location of the objects, the system refers to previous frames to link the object to the closest person based on the timing of which the object is detected to be taken or placed. At which point, the track id of the person is recorded for future use. As previously stated, there is a thread that listens to the NFC updates, and if an identity is updated, it takes the track id of the closest person to the sensor area and updates using the identity and the actions associated with the track id.

9.2.2.1 Object within Target Area

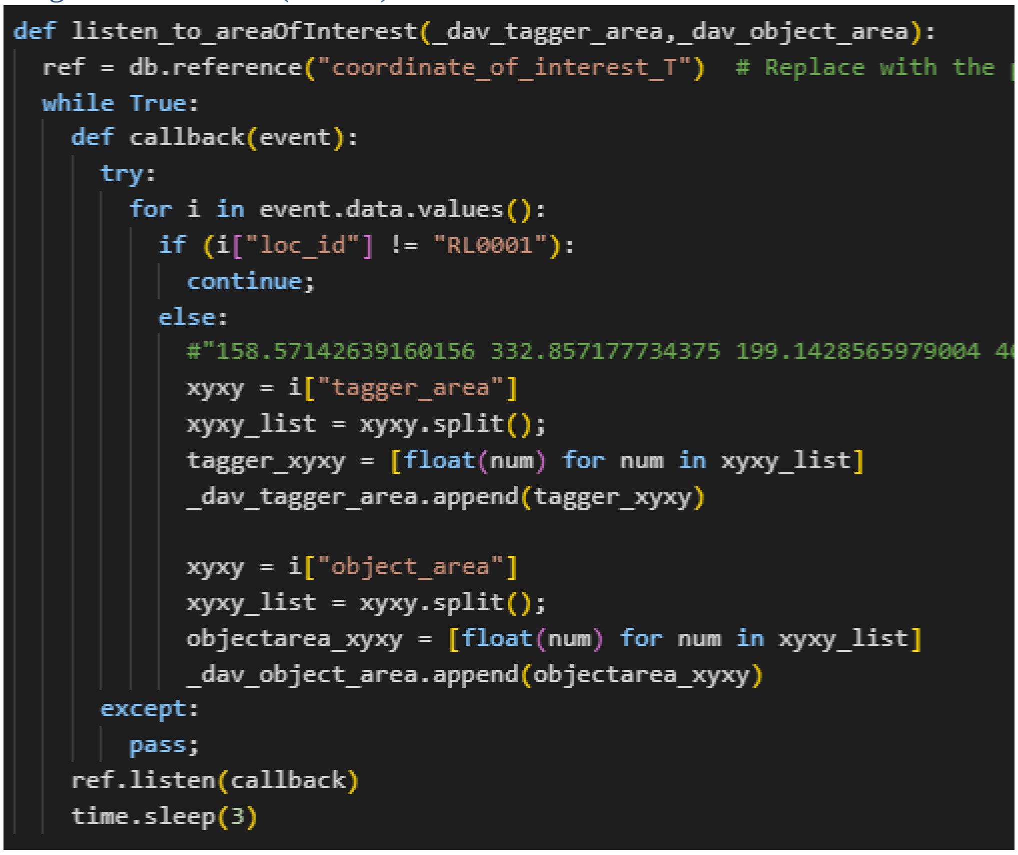


```python
def is_bbox_within_target_area(bbox,target_area):
    # bbox format: [x_min, y_min, x_max, y_max]

    # Check if the x-coordinate of the left side of the bbox is within the target_area
    if bbox[0] >= target_area[0] and bbox[0] <= target_area[2]:
        # Check if the y-coordinate of the top side of the bbox is within the target_area
        if bbox[1] >= target_area[1] and bbox[1] <= target_area[3]:
            return True
        # Check if the y-coordinate of the bottom side of the bbox is within the target_area
        if bbox[3] >= target_area[1] and bbox[3] <= target_area[3]:
            return True

    # Check if the x-coordinate of the right side of the bbox is within the target_area
    if bbox[2] >= target_area[0] and bbox[2] <= target_area[2]:
        if bbox[1] >= target_area[1] and bbox[1] <= target_area[3]:
            return True
        # Check if the y-coordinate of the bottom side of the bbox is within the target_area
        if bbox[3] >= target_area[1] and bbox[3] <= target_area[3]:
            return True

    # If none of the above conditions are met, the bbox is not within the target_area
    return False
```

Figure 14: Sample Code, object within target area

As previously stated, target area is procured from the user via the mobile application. This function filters out any objects excluding person within bounding boxes that doesn't overlap with the target area. In other words, any objects' bounding boxes that doesn't in anyway touches the target area is eliminated. This code allows the system to only track objects that are within the target area.

9.2.2.2 Target Area Listener (Thread)

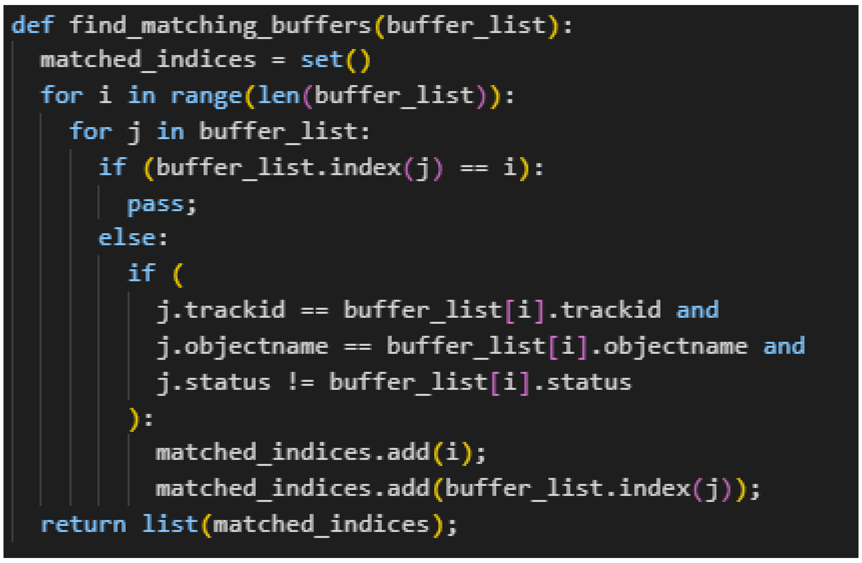


```python
def listen_to_areaOfInterest(_dav_tagger_area,_dav_object_area):
    ref = db.reference("coordinate_of_interest_T")  # Replace with the
    while True:
        def callback(event):
            try:
                for i in event.data.values():
                    if (i["loc_id"] != "RL0001"):
                        continue;
                    else:
                        #"158.57142639160156 332.857177734375 199.1428565979004 4
                        xyxy = i["tagger_area"]
                        xyxy_list = xyxy.split();
                        tagger_xyxy = [float(num) for num in xyxy_list]
                        _dav_tagger_area.append(tagger_xyxy)

                        xyxy = i["object_area"]
                        xyxy_list = xyxy.split();
                        objectarea_xyxy = [float(num) for num in xyxy_list]
                        _dav_object_area.append(objectarea_xyxy)
            except:
                pass;
        ref.listen(callback)
        time.sleep(3)
```

Figure 15: Target Area Listener (Thread)

To ensure that system may produce accurate results despite the difference in venue of the footage, a listener is created to await the coordinates of the target areas. If the coordinates are not pre-defined, the system will wait until the coordinates are updated to firebase. Simply put, this listener listens to both the previous and recent data.

9.2.2.3 Find Matched Buffers

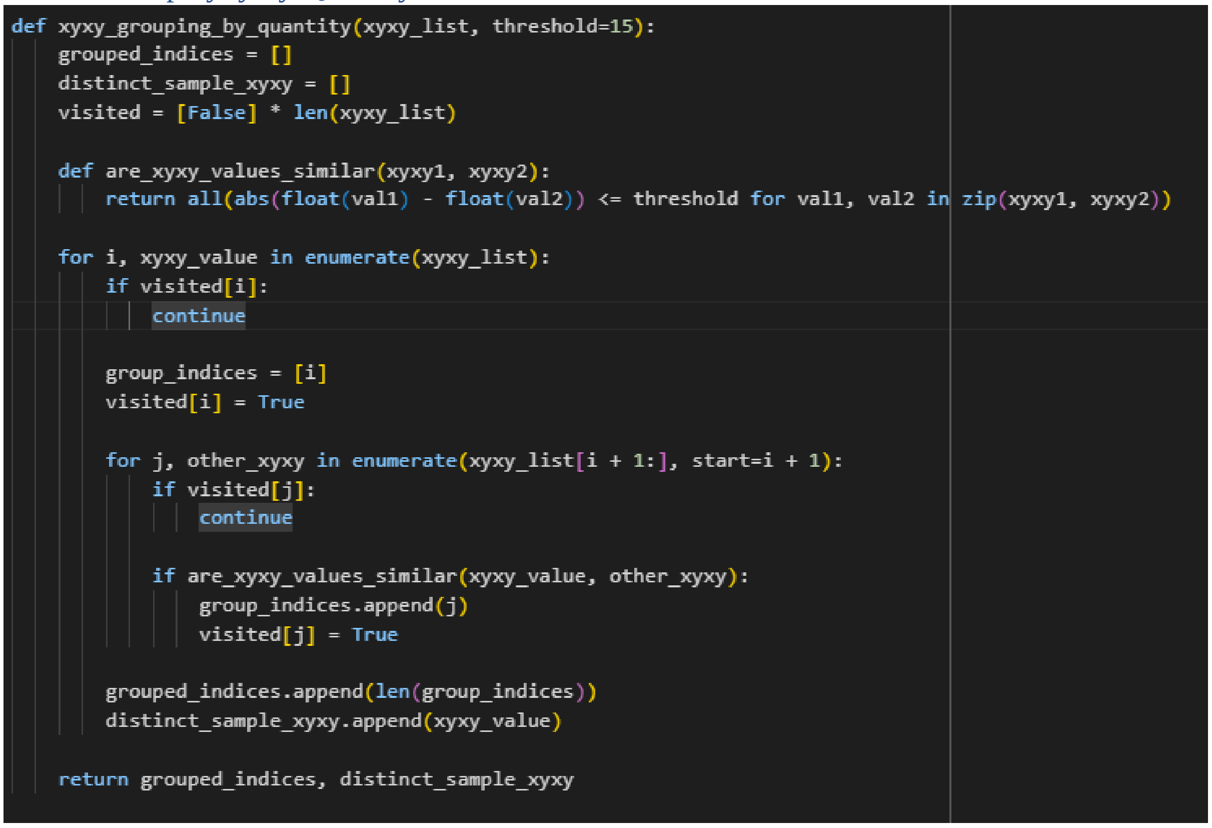


```python
def find_matching_buffers(buffer_list):
    matched_indices = set()
    for i in range(len(buffer_list)):
        for j in buffer_list:
            if (buffer_list.index(j) == i):
                pass;
            else:
                if (
                    j.trackid == buffer_list[i].trackid and
                    j.objectname == buffer_list[i].objectname and
                    j.status != buffer_list[i].status
                ):
                    matched_indices.add(i);
                    matched_indices.add(buffer_list.index(j));
    return list(matched_indices);
```

Figure 16: Sample Code, Find Matched Buffers

Buffers refers to objects taken or put back but the identity of the personnel involved has not been identified via NFC technology. Interestingly, each object/person has a distinct track id for easy verification. Therefore, if the track id is identical, the object influenced is identical and the action involved (taken/put) is different, then it is selected. The selected data will then be purged from the buffer txt.

9.2.2.4 Group xyxy by Quantity



```python
def xyxy_grouping_by_quantity(xyxy_list, threshold=15):
    grouped_indices = []
    distinct_sample_xyxy = []
    visited = [False] * len(xyxy_list)

    def are_xyxy_values_similar(xyxy1, xyxy2):
        return all(abs(float(val1) - float(val2)) <= threshold for val1, val2 in zip(xyxy1, xyxy2))

    for i, xyxy_value in enumerate(xyxy_list):
        if visited[i]:
            continue

        group_indices = [i]
        visited[i] = True

        for j, other_xyxy in enumerate(xyxy_list[i + 1:], start=i + 1):
            if visited[j]:
                continue

            if are_xyxy_values_similar(xyxy_value, other_xyxy):
                group_indices.append(j)
                visited[j] = True

        grouped_indices.append(len(group_indices))
        distinct_sample_xyxy.append(xyxy_value)

    return grouped_indices, distinct_sample_xyxy
```

Figure 17: Sample Code, Group xyxy by Quantity

YOLO has a very interesting phenomenon where the track id of an object would change when it is block for a certain duration. Therefore, the coordinate of the objects is regarded as a more accurate attribute. However, the coordinate of objects may still subject to little changes due to detection capabilities of YOLO or footage quality. Thus, this function helps to takes the coordinates of numerous frames and group up coordinates that are extremely close with each other, with only a minimal difference of 15 pixels in distance. Distinct objects may then be easily identified. For illustration purposes, in 30 frames, the quantity produced by the function may be 24, which means the particular object is detected 24 times of the 30 frames in the relatively same position, indicating that there is indeed an object there and is not a detection error. This function then returns the number of times distinct objects has been detected and their xyxy coordinates.

9.2.2.5 Find Closest Element

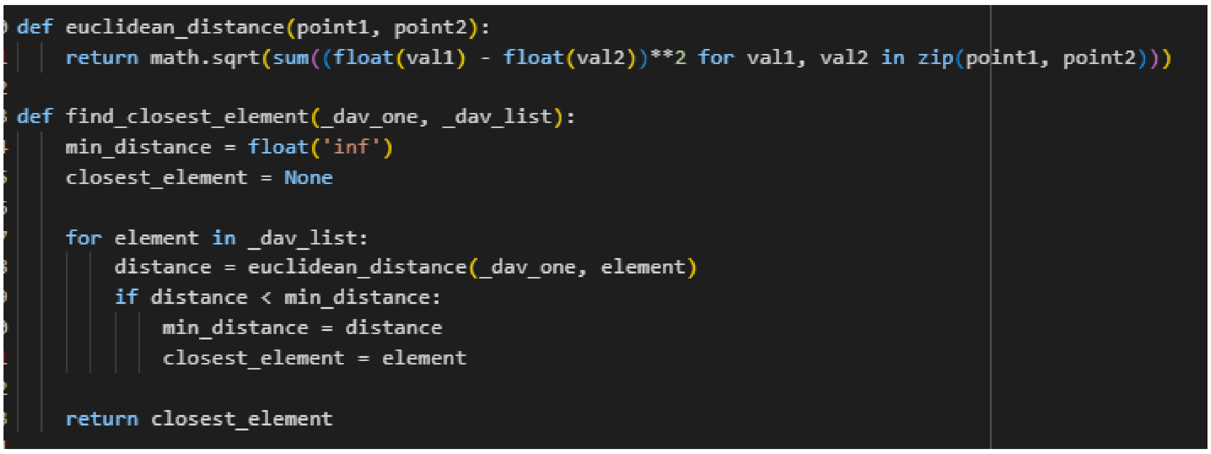

```python
def euclidean_distance(point1, point2):
    return math.sqrt(sum((float(val1) - float(val2))**2 for val1, val2 in zip(point1, point2)))

def find_closest_element(_dav_one, _dav_list):
    min_distance = float('inf')
    closest_element = None

    for element in _dav_list:
        distance = euclidean_distance(_dav_one, element)
        if distance < min_distance:
            min_distance = distance
            closest_element = element

    return closest_element
```

Figure 18: Sample Code, Find_Closest_Element

To link one detected object to another detected object, finding the closest object is crucial. For instance, a cup is taken from the table, and there is more than one person in the room. Therefore, by finding the closest detected person to the removed object, the removed object can be linked to the closest person.

9.2.2.6 Preprocessing Sample

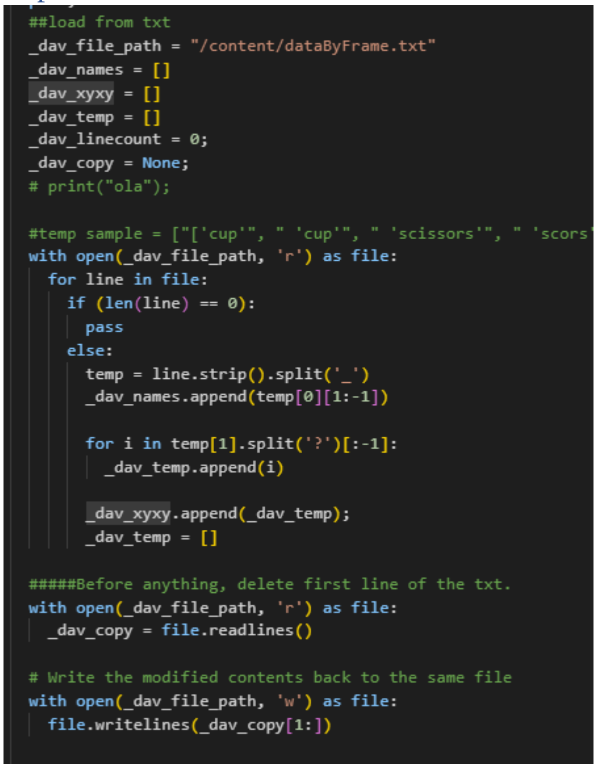
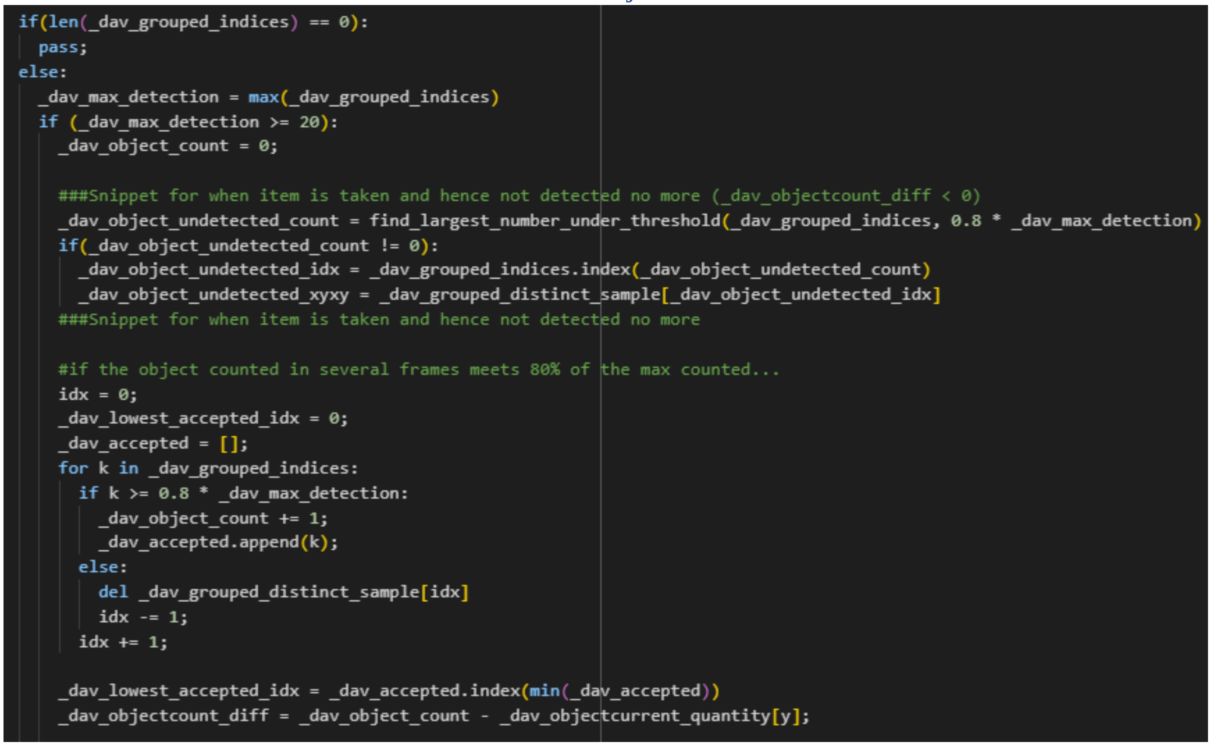

```python
##load from txt
_dav_file_path = "/content/dataByFrame.txt"
_dav_names = []
_dav_xyxy = []
_dav_temp = []
_dav_linecount = 0;
_dav_copy = None;
# print("ola");

#temp sample = ["['cup'", " 'cup'", " 'scissors'", " 'scors'
with open(_dav_file_path, 'r') as file:
  for line in file:
    if (len(line) == 0):
      pass
    else:
      temp = line.strip().split('_')
      _dav_names.append(temp[0][1:-1])

      for i in temp[1].split('?')[:-1]:
        _dav_temp.append(i)

      _dav_xyxy.append(_dav_temp);
      _dav_temp = []

#####Before anything, delete first line of the txt.
with open(_dav_file_path, 'r') as file:
  _dav_copy = file.readlines()

# Write the modified contents back to the same file
with open(_dav_file_path, 'w') as file:
  file.writelines(_dav_copy[1:])
```

Figure 19: Sample Code, Preprocessing Sample

From each detected objects, their class name and xyxy coordinates will be written to and read from txt files. It becomes extensively complex when multiple objects are detected each frame and more than 30 frames are compiled together to be processed.

9.2.2.7 Determine Total Number of Detected Object

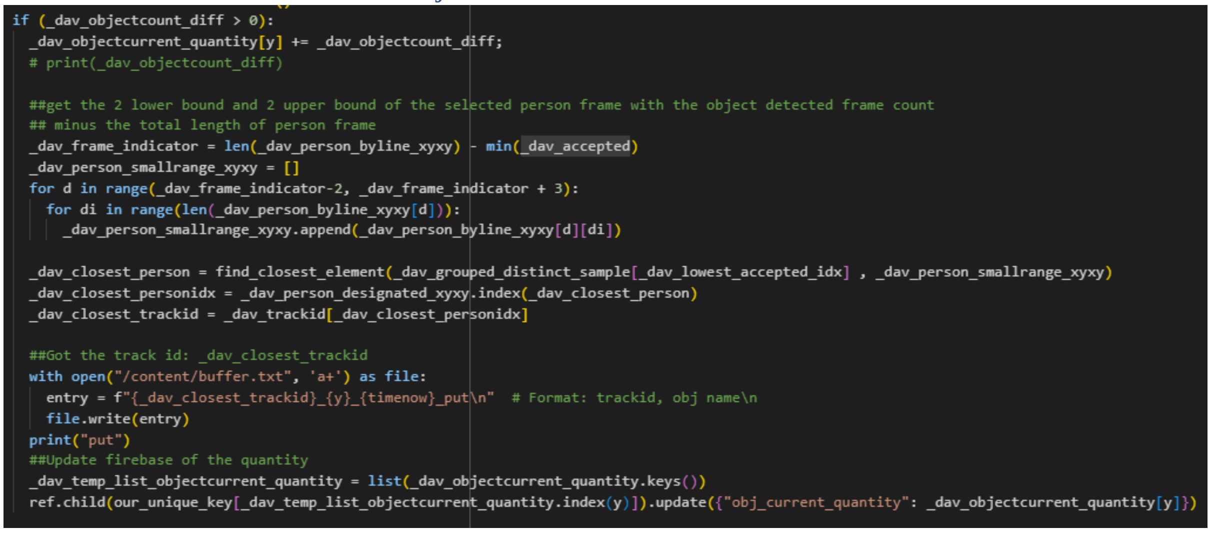


```
if(len(_dav_grouped_indices) == 0):
    pass;
else:
    _dav_max_detection = max(_dav_grouped_indices)
    if (_dav_max_detection >= 20):
        _dav_object_count = 0;

        ###Snippet for when item is taken and hence not detected no more (_dav_objectcount_diff < 0)
        _dav_object_undetected_count = find_largest_number_under_threshold(_dav_grouped_indices, 0.8 * _dav_max_detection)
        if(_dav_object_undetected_count != 0):
            _dav_object_undetected_idx = _dav_grouped_indices.index(_dav_object_undetected_count)
            _dav_object_undetected_xyxy = _dav_grouped_distinct_sample[_dav_object_undetected_idx]
        ###Snippet for when item is taken and hence not detected no more

        #if the object counted in several frames meets 80% of the max counted...
        idx = 0;
        _dav_lowest_accepted_idx = 0;
        _dav_accepted = [];
        for k in _dav_grouped_indices:
            if k >= 0.8 * _dav_max_detection:
                _dav_object_count += 1;
                _dav_accepted.append(k);
            else:
                del _dav_grouped_distinct_sample[idx]
                idx -= 1;
            idx += 1;

        _dav_lowest_accepted_idx = _dav_accepted.index(min(_dav_accepted))
        _dav_objectcount_diff = _dav_object_count - _dav_objectcurrent_quantity[y];
```

Figure 20: Sample Code, Determine Total Number of Detected Object

After executing 9.2.2.4 Object - Person Linkingthat counts the number of detections in 30+ frames. So, for assurance purposes, the code would only proceed if one of the detect object achieves detection in more than 20 frames. Ignoring the snippet surrounded by comments and look below, objects with quantity that is below 80% of the max quantity is considered as detection being halted some point in the frames or the object has not remain motionless for a period of time. These objects that have been eliminated is then utilized to pinpoint the location of it, if an object has been taken or the current number of detect objects is different from the previous detections. On the contrary, objects with quantity that complies with the above 80% condition, is officially treated as currently existed objects.

9.2.2.8 Find Closest Person to Object



```
if (_dav_objectcount_diff > 0):
  _dav_objectcurrent_quantity[y] += _dav_objectcount_diff;
  # print(_dav_objectcount_diff)

  ##get the 2 lower bound and 2 upper bound of the selected person frame with the object detected frame count
  ## minus the total length of person frame
  _dav_frame_indicator = len(_dav_person_byline_xyxy) - min(_dav_accepted)
  _dav_person_smallrange_xyxy = []
  for d in range(_dav_frame_indicator-2, _dav_frame_indicator + 3):
    for di in range(len(_dav_person_byline_xyxy[d])):
      _dav_person_smallrange_xyxy.append(_dav_person_byline_xyxy[d][di])

  _dav_closest_person = find_closest_element(_dav_grouped_distinct_sample[_dav_lowest_accepted_idx] , _dav_person_smallrange_xyxy)
  _dav_closest_personidx = _dav_person_designated_xyxy.index(_dav_closest_person)
  _dav_closest_trackid = _dav_trackid[_dav_closest_personidx]

  ##Got the track id: _dav_closest_trackid
  with open("/content/buffer.txt", 'a+') as file:
    entry = f"{_dav_closest_trackid}_{y}_{timenow}_put\n"  # Format: trackid, obj name\n
    file.write(entry)
  print("put")
  ##Update firebase of the quantity
  _dav_temp_list_objectcurrent_quantity = list(_dav_objectcurrent_quantity.keys())
  ref.child(our_unique_key[_dav_temp_list_objectcurrent_quantity.index(y)]).update({"obj_current_quantity": _dav_objectcurrent_quantity[y]})
```

Figure 21: Sample Codes, Find Closest Person to Object

The figure above demonstrates the processes that executes when an object is added. The system scour over nearby ranges of frame of which the object was first detected. About 5 frames is taken to compare which person was the closest towards the added object. The linking occurs and the data is updated.

CHAPTER 10: SYSTEM VALIDATION

10.1 Mobile Application

10.1.1 Login page

Test ID	Test Case	Expected	Actual	Pass/Fail
1.1	1. Fill in with correct credentials. 2. Press login button.	Login Success	As Expected	Pass
1.2	1. Leave Password field empty. 2. Press login button.	Login Fail	As Expected	Pass
1.3	1. Leave Email field empty. 2. Press login button.	Login Fail	As Expected	Pass
1.4	1. Fill in with wrong credentials. 2. Press login button.	Login Fail	As Expected	Pass

10.1.2 Location Page

Test ID	Test Case	Expected	Actual	Pass/Fail
1.1	1. Tap Favourites	Only favourited location is displayed.	As Expected	Pass
1.2	1. Tap Recently Visited	Only 2 recently visited location is displayed.	As Expected	Pass
1.3	1. Tap Heart Icon of 1 to black 2. Tap Favourites	The particular location is removed from favourites.	As Expected	Pass

1.4	1. Tap Heart Icon of 1 to red 2. Tap Favourites	The particular location is displayed in favourites.	As Expected	Pass
1.5	1. Tap Recently Visited. 2. Tap location that is not displayed. 3. Tap Recently Visited.	The particular location is displayed in recently visited.	As Expected	Pass
1.6	1. Tap pen icon button. 2. Tap plus icon button.	Redirected to location register page.	As Expected	Pass
1.7	1. Tap hamburger. 2. Tap create users.	Redirected to user registration page.	As Expected	Pass

10.1.3 User Register

Test ID	Test Case	Expected	Actual	Pass/Fail
1.1	1. No field filled. 2. Press register button.	Registration fails.	As Expected	Pass
1.2	1. Only email and password filled. 2. Press register button.	Registration success.	As Expected	Pass
1.3	1. All field but email and password filled. 2. Press register button.	Registration fails.	As Expected	Pass
1.4	1. Press login button	Redirected to login page	As Expected	Pass
1.5	1. All field filled but email don't have '@'. 2. Press register button.	Registration fails.	As Expected	Pass

10.1.4 View Information of Selected Location

Test ID	Test Case	Expected	Actual	Pass/Fail
1.1	View Object Total.	Object total of the location can be viewed.	As Expected	Pass
1.2	View Object Logs.	Object logs of the location can be viewed.	As Expected	Pass
1.3	View Footage.	Footage of the location can be viewed.	As Expected	Pass
1.4	1. Adjust box areas. 2. Tap printer icon button.	Firebase is updated.	As Expected	Pass

10.2 Tracking System Evaluation

Test ID	Situation	Expected	Actual	Pass/Fail

1.1	1. Table has 1 Cup, 2 Scissors. 2. Add 1 cup to the table. 3. Tap on sensor (manually update credentials)	1. current_quantity of cup in object_T will +1. 2. object_log_T is appropriately updated with status = "put".	As Expected	Pass
1.2	1. Table has 2 Cup, 2 Scissors. 2. Take 1 cup from the table. 3. Tap on sensor (manually update credentials)	1. current_quantity of cup in object_T will -1. 2. object_log_T is appropriately updated with status = "taken".	As Expected	Pass
1.3	1. Table has 2 Cup, 2 Scissors. 2. Take 1 Scissors from the table. 3. Tap on sensor (manually update credentials)	1. current_quantity of scissors in object_T will -1. 2. object_log_T is appropriately updated with status = "taken".	As Expected	Pass
1.4	1. Table has 2 Cup, 2 Scissors. 2. Take 1 Scissors from the table. 3. Put back Scissors to the table. 4. Take 1 Scissors from the table. 5. Tap on sensor (manually update credentials)	1. current_quantity of scissors in object_T will -1. 2. object_log_T is appropriately updated with status = "taken". 3. Only 1 log is added, instead of 3.	As Expected	Pass
1.5	1. Table has 2 Cup, 2 Scissors. 2. Take 1 Scissors from the table. 3. Take 1 Cup from the table. 4. Tap on sensor (manually update credentials)	1. current_quantity of scissors and cup in object_T will -1. 2. object_log_T is appropriately updated with status = "taken".	Half as expected, second log did not have user id.	fail
1.6	1. Table has 2 Cup, 2 Scissors. 2. Take 1 Scissors from the table. 3. Take 1 Cup from the table.	1. current_quantity of scissors and cup in object_T will -1. 2. object_log_T is appropriately updated with status = "taken". 3. Log will not have a user_id.	As Expected. But an extra set of take and put was recorded.	Pass
1.7	1. Table has 2 Cup, 2 Scissors. 2. Take 1 Cup from the table. 3. Place the cup away from the target area.	1. current_quantity of cup in object_T will -1. 2. object_log_T is appropriately updated with status = "taken". 3. Log will not have a user_id.	As Expected.	Pass
1.8	1. Table has 2 Cup, 2 Scissors. 2. Obstruct one cup for 5 seconds.	No Action	As Expected	Pass

1.9	1. Table has 2 Cup, 2 Scissors. 2. Move them close to each other.	No Action	3 Logs were created.	Fail

10.2.1 Fails Reasoning

Test ID	Actual/Possible Reason for Failure
1.5	A part of the table was briefly detected as a person.
1.9	Objects were too close and sometimes they're detected as one sometimes two.

Based on the output of the detections, test ID 1.5 failed because for some reason, in the small part of the area where the scissors were placed, it was detected as a person twice, in a very, very, very short period. The developer stresses it appeared less than 5 frames each appearance. It severely puzzled the developer as Yolov7 is considerably accurate and it did not happen previously tested videos. The failure of test ID 1.9 was somewhat predicted by the developer. The tracking system is not capable of tracking objects that are too close with each other.

CHAPTER 11: CONCLUSIONS AND REFLECTIONS

11.1 Critical Evaluation

In critical evaluation, it will provide an insight on the circumstances experienced by the developer whilst undergoing this project. Additionally, critiques on the expected result and actual results will be discussed.

11.1.1 YOLO Problems

In the beginning of the project, the developer intended to utilize You Only Look Once (YOLO) algorithm to develop the underlying tracking system. However, the developer did not have a particular version in mind and misunderstood that all YOLO versions are the same and the newest version is the best in all categories in terms of performance. Initially, the developer started off with YOLOv7, when realising that the frame per second during live-stream was too low, the developer switched to YOLOv3-tiny that is known for its high real-time compatibility. Unfortunately, whilst training the weight for YOLOv3-tiny, it took the developer a significant amount of time before realising that the accuracy for YOLOv3-tiny is simply too low. Each training session took an average of 2.5 hours, data gathering took around 45 minutes, data processing required an average of 30 minutes, and each mean average precision (mAP) test took 10 minutes. At that time, the developer was totally convinced that the training would work if minor modifications were done on the YOLOv3-tiny configurations as the implementation fully adheres to the procedures of YouTube videos, GitHubs and websites. During the developer's last attempt, despite having a nice learning curve, the mAP for YOLOv3-tiny was around 6% whilst the mAP for YOLOv3 was around 33%. As a result, the developer decides to ignore hardware limitation and opted for YOLOv7 to continue progressing the project. In other words, due to time wasted on YOLOv3-tiny, YOLOv7 is used, including pre-trained YOLOv7 weights.

11.1.2 YOLOv7 Result Manipulation Difficulties

The YOLOv7 code acquired has both detecting and tracking functionalities. As such, each object will be accompanied by a distinct track id. It was also initially assumed that if the object and the footage does not move, the coordinates of the bounding boxes of the detected objects will remain unchanged. Nevertheless, conundrums such as the changing of track id when its view is obstructed, different coordinates for motionless objects, and inaccuracy when detecting objects started to surface. As a result, the workload of the developer spiked, in spite of the delay caused by YOLOv3-tiny. The developer's intention was to prioritize the mobile application instead of the YOLO algorithm, but was instead forced to prioritize the tracking system instead, shifting the core of the project to the tracking system.

11.1.3 Mobile Application Functionalities

Previously declared functionalities such as alert features, live-streaming, and beautiful user interfaces are withheld. As a result of the increase in the difficulty of manipulating the results of YOLOv7, the mobile application is now regarded as a simple prototype to reflect the end-results of the tracking system. In other words, the mobile application will still retain core functionalities that strongly associates with the end-results produced by the tracking system.

11.1.4 NFC Procurement Failure

In light of the delays and low frame rate of YOLOv7, an NFC technology was not procured. The initial thought process was to purchase an NFC technology or more specifically, an NFC tag and an NFC sensor. The purpose of the NFC technology was for the personnel to identify themselves through them, and the firebase may be updated accordingly. To mitigate this problem, data is manually added to the firebase to mimic a working NFC technology. As the live stream has been substituted to a recorded video, the developer may add the data at the precise time to test the tracking system.

11.1.5 Firebase Weird Phenomenon

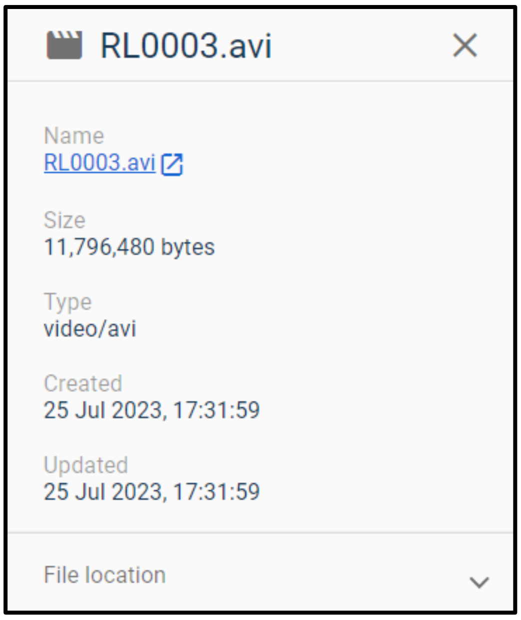
When the tracking system finishes creating bounding boxes, and labels on the detected objects, the result is in the form of a video. The video is then transferred onto firebase using python codes and is then accessed by the mobile application. Although the Url linked to the video is working perfectly, it was unable to be loaded in the mobile application. Several assumptions related to the problem were tested but to no avail. Assumptions include…

1. The Url procured by the mobile application is wrong.
   - ⌘ During the upload of the video, the Url of the video is taken and put into the mobile codes as a dummy data. Therefore, the Url is correct. Additionally, the Url is working when entered through a browser.

2. The video was not properly uploaded.
   - ⌘ As previously mentioned, the Url is working when entered through a browser. Thus, signifying that the video was completely uploaded to the firebase.

3. The code to load the video is wrong.
   ⌘ The code was used to download manually uploaded files of the same directory, but it only failed when loading videos uploaded by python code.

It is believed that the firebase has a bug that is not known to the community. Proof can be seen in the pictures below. Additionally, it is imperative to note that the video uploaded using python codes remained black and un-tappable despite hours have passed. Luckily, in the last minute, the developer used a code to fix the video and it worked. The name turned blue as well.



RL0003.avi

Name
RL0003.avi

Size
11,796,480 bytes

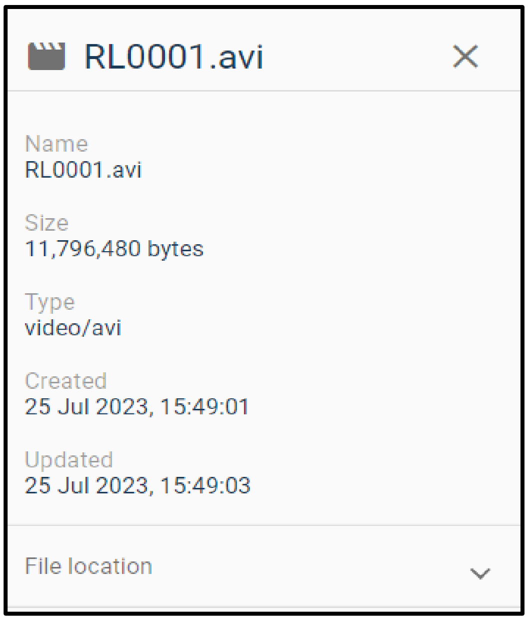
Type
video/avi

Created
25 Jul 2023, 17:31:59

Updated
25 Jul 2023, 17:31:59

File location

Figure 22: Manually Uploaded Video



RL0001.avi

Name
RL0001.avi

Size
11,796,480 bytes

Type
video/avi

Created
25 Jul 2023, 15:49:01

Updated
25 Jul 2023, 15:49:03

File location

Figure 23: Python Uploaded Video

11.1.6 Tracking System's Performance

To process a 17 second video, the tracking system took about 1 minute to finish. Although the low speed is concerning, it can be resolved with better hardware. Regarding its accuracy, the tracking system works as intended, except during occasions where the objects are placed in extremely close proximity. During those moments, the system would misinterpret the objects as one. Nevertheless, the accuracy of the system has exceeded expectations along with its dynamic capabilities. Dynamic capabilities refer to the system's capability to adapt to different locations or background or venues. Several processes have been put in place to ensure objects out of target areas are not considered and the personnel that tags his NFC tag may be easily identified and conclude the items he took or placed.

11.2 Conclusion

Through hard work and dedication, the developer has successfully designed and implemented a sophisticated system that harnesses the power of YOLOv7's detection and tracking feature. This system effectively manipulates the detected objects and delivers accurate as well as reliable results. Firebase has also seamlessly integrated with the system, allowing smooth uploading or access of data. Although the mobile application is but a means to demonstrate the results delivered by the system, it is still a fully working mobile application that utilizes Firebase to display the necessary data.

11.2.1 System Limitation

The tracking system has some problem tracking and detecting the scissors. For certain scenarios, it was regarded as a person, which was extremely unexpected from Yolov7 that is revered to be highly accurate. However, it is understandable as scissors is small and comes any different forms. Nevertheless, it is highly accurate when detecting the cups. Additionally, if objects are too close to each other, it would mistakenly group them as one item.

11.2.2 Research Limitation

Although the responses from the questionnaires are immensely helpful, it does not provide insights on the technical aspects of the tracking system. The questionnaire revolved around the usability and relatability of the proposed system. Thus hindering the developer's progress in manipulating YOLO results and creating the mobile application.

11.2.2.1 Interview

An interview could have been conducted to gain a deeper understanding of the features within YOLO and the techniques to manipulate its results. The teachings in regard to YOLO should be divided into two components: theory base and section definition. Theory base refers to how the results should be manipulated, what the developer should expect while doing so and what the outcome should be. On the other hand, section definition refers to the teachings of the variables, the functions, any objects that may be useful while applying the theory. This is because YOLO has multiple python files, and the developer only made configurations on one python file. Therefore, although the developer might have successfully manipulated the results but many of his codes might already have pre-implemented solutions that were overlooked. In plain terms, the codes implemented could've been written faster or easier if the developer's knowledge on YOLO was deeper with the help of an interview.

11.2.2.2 Literature Review _ Hands On

Extensive focus of the literature review done was placed on similar systems and general information related to an object detection system. As the literature review was done before deciding the software that will be used in this project, the attention of the developer was split, and an in-depth literature review was not done on YOLO. Combined with the fact that online tutorials are few and scarce, the developer was forced to troubleshoot at unresolvable problem. The developer believes that a trial period for the selected software is necessary before conducting a thorough literature review. This is due to the fact that the developer will be the only developer on the project. On that account, any software that does not suit the developer's taste can immediately be removed from candidate position. During this period, the developer may switch and swap wherever necessary and not be forced to commit to a single software.

11.2.3 Future Enhancement

11.2.3.1 Mobile Application

The mobile application was ultimately deemed as a sort of accessory for the tracking system. Therefore, the mobile application's user interface was evidently not well designed and mainly consists of functions that displays the core features of the tracking system. Therefore, for future enhancement, the application could be better designed, adhering to usability goals and design principles. Usability goals are concerned with how useful and easy it is to use a product or system. (Educative, n.d.)

11.2.3.2 Tracking System

To start with, the YOLOv7 weight utilized is pre-trained with a total of 80 detectable classes. As a result, major processing resources are wasted to detect classes irrelevant to the footage at hand. To combat that, a custom weight will be trained to detect only necessary classes. Additionally, a customized weight opens the door to model tuning and neural network modification that boosts detection accuracy.

11.2.4 Personal Reflection

11.2.4.1 Request for Help

As previously stated, the developer put a lot of sweat and tears on resolving the issues with YOLOv3-tiny without avail. As a result, the motivation of the developer suffered a huge blow. The developer reflects that he should've seek for second or third opinion from someone outside the box rather than blindly believing that the issue would resolve one day or another. If the developer had done so, the developer trusts that he would've move on to YOLOv7 in the blink of an eye.

11.2.4.2 Set Maximum Duration for Tasks

First and foremost, there is more than one YOLO versions. The developer could've used other YOLO versions but was too committed with YOLOv3-tiny. However, if the developer had set a maximum duration for the YOLOv3-tiny to work, the developer would as easily realise that YOLOv3 doesn't have a high accuracy either or decides to utilize the pre-trained weight instead as a last-ditch effort.

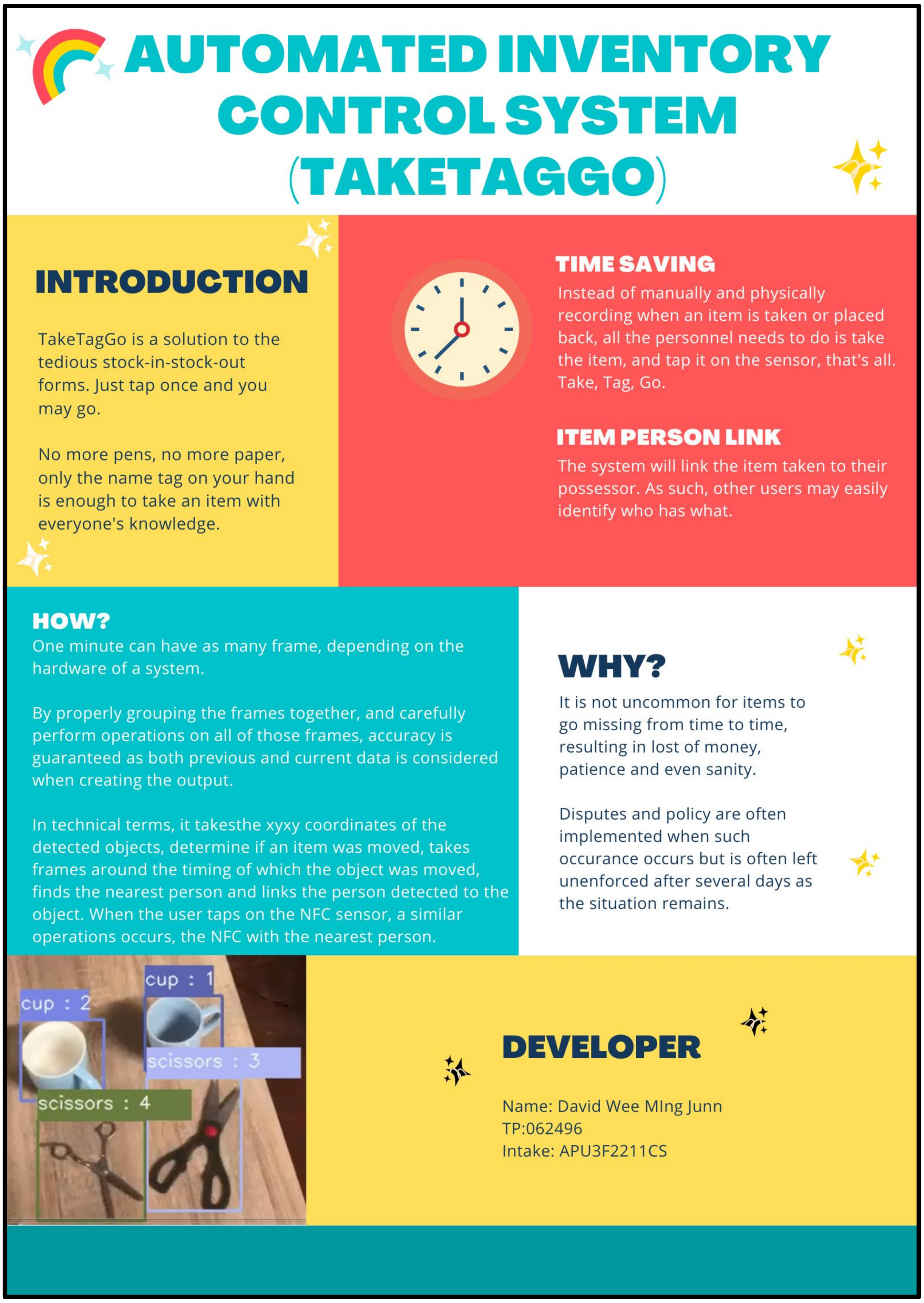
11.2.4.3 Time Planning

As sufficiently elaborated, the tracking system took a huge chunk of time, causing the developer to rush up the remaining workloads such as the mobile application and documentation. The developer also did not realise that the documentation would be so extensive, racking up unnecessary anxiety and completing work that could be better represented. The developer deeply regrets on this and vows to plan ahead with 7 days to spare for whatever unforeseen hold-up due to incompatibility, non-resolvable errors or even procrastination.

11.2.4.4 Preference trumps Theory

Initially React Native was chosen instead of Flutter, as both were theorized to have a low learning curve and React Native is known to have a larger community. However, during the first few days of dabbling with React Native, the developer decided to go with Flutter as Flutter felt easier and more organized, due to the preference of the developer. Therefore, the developer reflects that personal preference should be highly prioritized.

Posters



AUTOMATED INVENTORY CONTROL SYSTEM (TAKETAGGO)

INTRODUCTION

TakeTagGo is a solution to the tedious stock-in-stock-out forms. Just tap once and you may go.

No more pens, no more paper, only the name tag on your hand is enough to take an item with everyone's knowledge.

TIME SAVING

Instead of manually and physically recording when an item is taken or placed back, all the personnel needs to do is take the item, and tap it on the sensor, that's all. Take, Tag, Go.

ITEM PERSON LINK

The system will link the item taken to their possessor. As such, other users may easily identify who has what.

HOW?

One minute can have as many frame, depending on the hardware of a system.

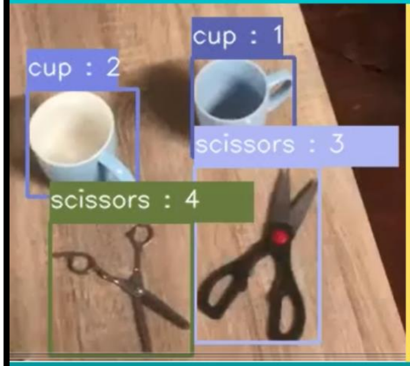
By properly grouping the frames together, and carefully perform operations on all of those frames, accuracy is guaranteed as both previous and current data is considered when creating the output.

In technical terms, it takesthe xyxy coordinates of the detected objects, determine if an item was moved, takes frames around the timing of which the object was moved, finds the nearest person and links the person detected to the object. When the user taps on the NFC sensor, a similar operations occurs, the NFC with the nearest person.

WHY?

It is not uncommon for items to go missing from time to time, resulting in lost of money, patience and even sanity.

Disputes and policy are often implemented when such occurance occurs but is often left unenforced after several days as the situation remains.



DEVELOPER

Name: David Wee MIng Junn
TP:062496
Intake: APU3F2211CS

Figure 24: Poster