Checklist App

Team LoraK
Lora Kalthoff

Project also available on GitHub: https://github.com/LoraKalt/CSCI3800_ChecklistApp

1 Project Description

Checklist app is a to-do-list for everyday needs. Users can create new tasks, modify existing ones, and delete. Creating a task is easy, with each task, user can specify the urgency of the task and optionally what date does the task need to be done. Once created, the user can view this task and are given the option to edit the task or to delete it instead.

Once the user has a list of tasks, they then can sort them by oldest-to-newest based on creation order; by date, with most upcoming dates on top; and by priority, with most urgent on top of the list. Clicking on these sorting options, every time the user creates a new task, it will automatically be sorted in the given order.

Clicking the checkbox indicates that the task is completed. If there are several completed tasks, the user has the option to delete all checked tasks to clear up their list.

Every change the user makes to the checklist will be saved on the database so that when the return to the app, they'll be able to see their created list from the last session.

It was created using API: 23, Android 6.0 (Marshmallow) with the Pixel 4 API 26

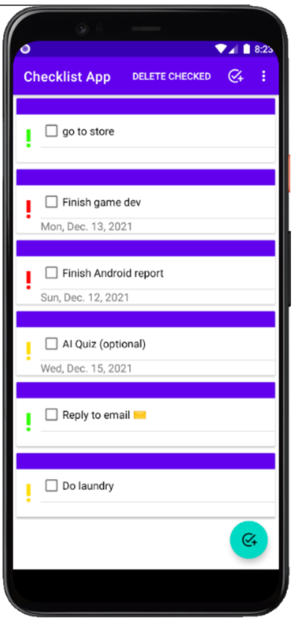
2 Requirement Specifications

As I was creating this app, the proposed requirements changed a bit. The new requirement specifications:

- Three views: The main activity that displays the list of tasks that need to be done, the add/edit view where tasks can be created and edited, and the read/view activity that gives an overview one task.
- Tasks. They have three fields: the string the user inputs, the priority which is one of three options (low priority, moderate, and urgent), and a due date. The due date is optional.
  - App implements CRUD actions in that: Users can create a task, they can read an overview of any task, they can update a pre-existing task, and they can delete it.
- Display the list of tasks. Here the user can check/uncheck each task.
- Delete all checked: Deletes all checked tasks on the list.
- Sort list: Can sort the list either by oldest to newest, by due date and by priority.

3 Technical Description

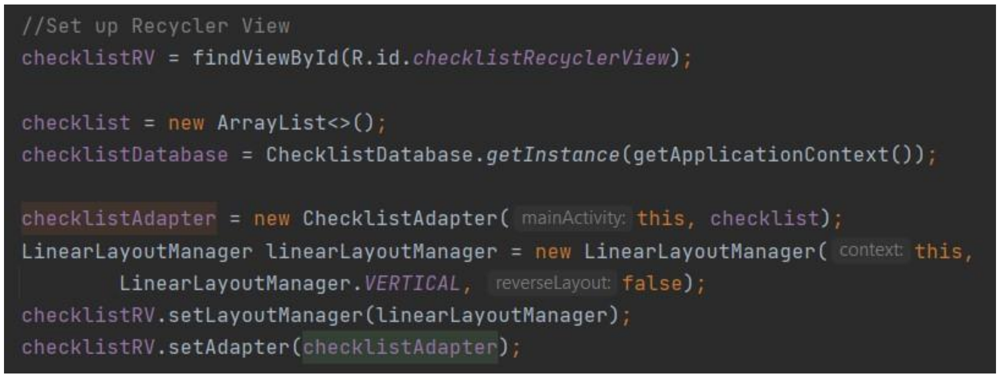
3.1 Displaying Tasks



Each task has the three primary fields (the task string, priority, and the optional due date) and is displayed in a card. I created one Card View that is then displayed with specific information using Recycler View.

This was achieved by making an adapter for the Recycler View called ChecklistAdapter that modifies each item on the list individually and keeps track of its indices.

Inside the Main Activity, the Recycler View is set up using a linear layout manager:

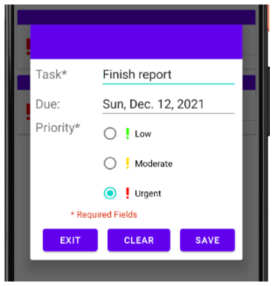


```
//Set up Recycler View
checklistRV = findViewById(R.id.checklistRecyclerView);

checklist = new ArrayList<>();
checklistDatabase = ChecklistDatabase.getInstance(getApplicationContext());

checklistAdapter = new ChecklistAdapter( mainActivity: this, checklist);
LinearLayoutManager linearLayoutManager = new LinearLayoutManager( context: this,
        LinearLayoutManager.VERTICAL,  reverseLayout: false);
checklistRV.setLayoutManager(linearLayoutManager);
checklistRV.setAdapter(checklistAdapter);
```

As seen in the following code, the data is stored in an ArrayList of Checklist Object. The Checklist Class stores all important data:

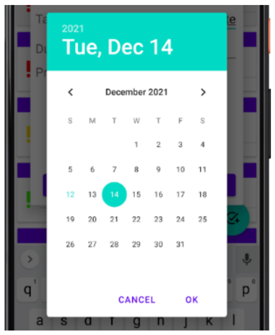
Task of string type, priority of integer type, due date of Calendar type, status of Boolean type, and creation date of Calendar type. Since moving the app to a database, the creation date is no longer used for sorting as it was before.

Each card has a click listener that when the user clicks on one, the View Card dialog will be prompt.



3.2 Adding and Editing a Task (Create and Update)

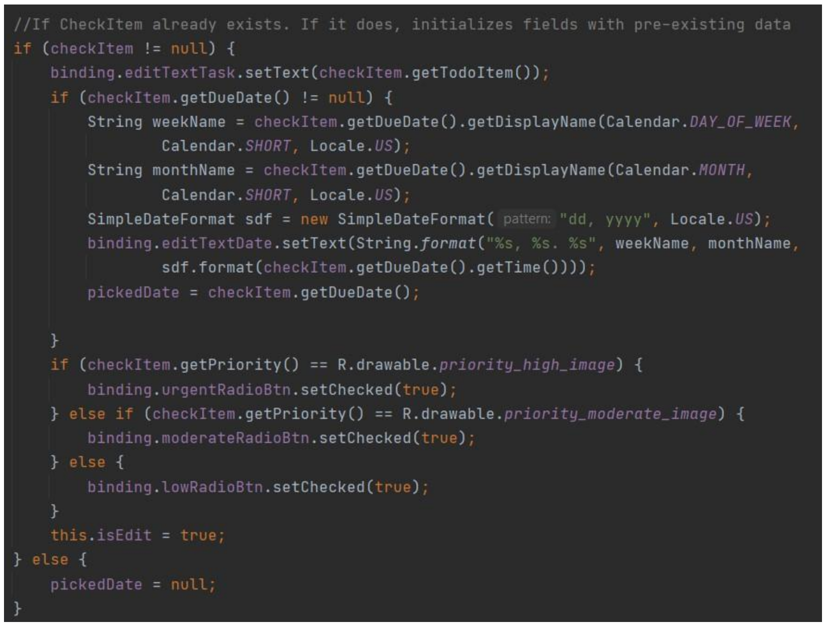
There are two ways to add a task. In the main screen, you can either click on the floating action button on the bottom right screen or you can click on the icon in the toolbar. Both adding and editing a task uses the same dialog activity and class called AddChecklistDialog that handles the entered data fields and pushing the new item on the list (or modifying an existing item).



Task takes a string that the user inputs. While not intentional, the user can use emojis if they so wish to. (It was a nice surprise to find out that the database can handle emojis by default).

Entering the Due date prompts of the DatePicker dialog that makes it easier to pick a date. After picking a date, it is then converted into a string for display. This is done by using SimpleDateFormat and String format.

Lastly, the user can choose the type of priority they want by clicking on the radio buttons. It is saved in the database as the ID of the image that Android automatically assigns when creating images.



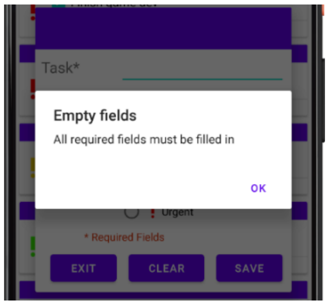
```
//If CheckItem already exists. If it does, initializes fields with pre-existing data
if (checkItem != null) {
    binding.editTextTask.setText(checkItem.getTodoItem());
    if (checkItem.getDueDate() != null) {
        String weekName = checkItem.getDueDate().getDisplayName(Calendar.DAY_OF_WEEK,
                Calendar.SHORT, Locale.US);
        String monthName = checkItem.getDueDate().getDisplayName(Calendar.MONTH,
                Calendar.SHORT, Locale.US);
        SimpleDateFormat sdf = new SimpleDateFormat( pattern: "dd, yyyy", Locale.US);
        binding.editTextDate.setText(String.format("%s, %s. %s", weekName, monthName,
                sdf.format(checkItem.getDueDate().getTime())));
        pickedDate = checkItem.getDueDate();

    }
    if (checkItem.getPriority() == R.drawable.priority_high_image) {
        binding.urgentRadioBtn.setChecked(true);
    } else if (checkItem.getPriority() == R.drawable.priority_moderate_image) {
        binding.moderateRadioBtn.setChecked(true);
    } else {
        binding.lowRadioBtn.setChecked(true);
    }
    this.isEdit = true;
} else {
    pickedDate = null;
}
```

When the dialog is first created, there are checks to see if the user is adding a new task or updating a pre-existing one. If it's a pre-existing one, then it will fill in the fields with the data at hand. In the code to the side, it shows the fields being modified.

There are also three buttons. Exit brings the user back to the main activity; the clear button clears all fields and save pushes the filled data into the database. Saving a task has a lot going on in the background besides adding to the ArrayList or modifying an item. There are a lot of checking to see if the data entered is valid.

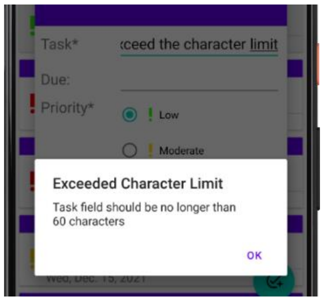
3.2.1 Catching Invalid Fields

There are three alerts that pop up if entered fields are not valid. First, task string cannot be empty, nor can it exceed the 60-character limit and the due date must be set to today's date or after. Checking the string was simple since it's just checking to see if TextView is empty and if the size of the string exceeded 60 characters.



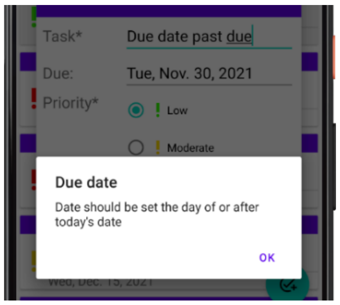
Empty fields

All required fields must be filled in



Exceeded Character Limit

Task field should be no longer than 60 characters

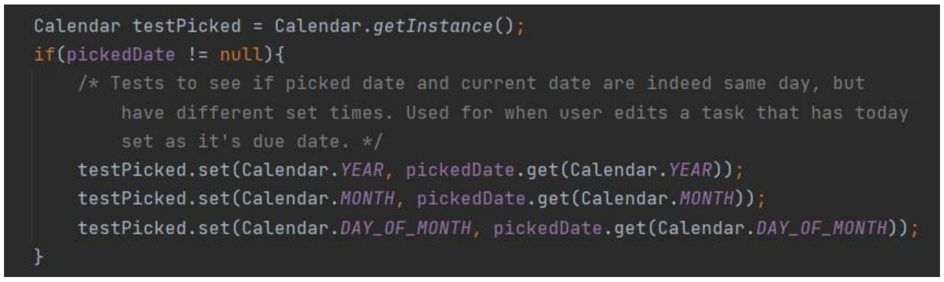


Due date

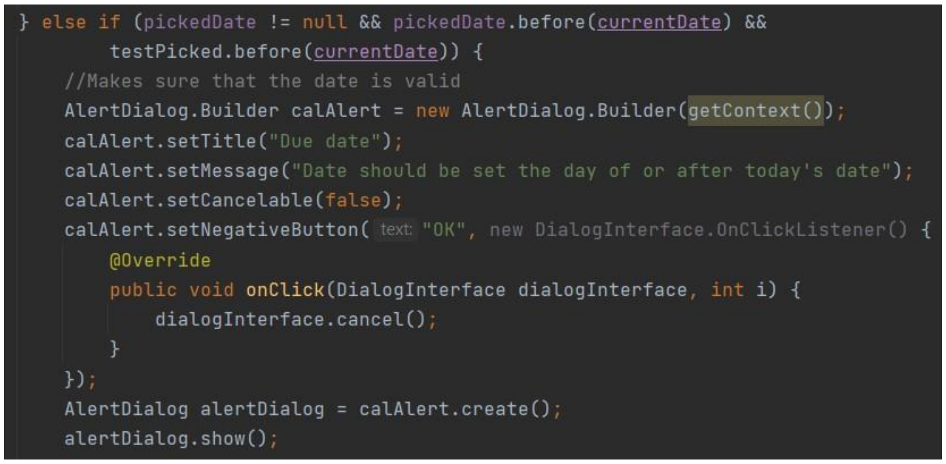
Date should be set the day of or after today's date

Checking the due date was a bit more involved. I used a Calendar timestamp using the getInstance() method, that is then compared to the picked date. If the date picked is before the timestamp, it is invalid and the alert will pop up.

One of the more difficult things was to set up when the user is editing a task that had today's date set as the due date. The alert message would still pop up if the field was left alone after trying to save. To overcome this, I created another Calendar instance and changed the year, month and day to match the Calendar date in the Checklist object. This allowed it so that alert message to not pop up, making it so if the user just wanted to edit the priority or the task string, they could do so.

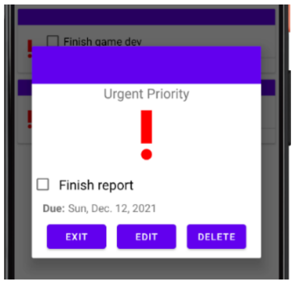


```
Calendar testPicked = Calendar.getInstance();
if(pickedDate != null){
    /* Tests to see if picked date and current date are indeed same day, but
        have different set times. Used for when user edits a task that has today
        set as it's due date. */
    testPicked.set(Calendar.YEAR, pickedDate.get(Calendar.YEAR));
    testPicked.set(Calendar.MONTH, pickedDate.get(Calendar.MONTH));
    testPicked.set(Calendar.DAY_OF_MONTH, pickedDate.get(Calendar.DAY_OF_MONTH));
}
```



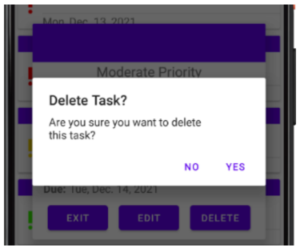
```
} else if (pickedDate != null && pickedDate.before(currentDate) &&
        testPicked.before(currentDate)) {
    //Makes sure that the date is valid
    AlertDialog.Builder calAlert = new AlertDialog.Builder(getContext());
    calAlert.setTitle("Due date");
    calAlert.setMessage("Date should be set the day of or after today's date");
    calAlert.setCancelable(false);
    calAlert.setNegativeButton( text: "OK", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            dialogInterface.cancel();
        }
    });
    AlertDialog alertDialog = calAlert.create();
    alertDialog.show();
```

3.3 Viewing a Task (Read)



As mentioned earlier, clicking on a card will pop up the view dialog activity in which the ViewChecklistDialog class handles. It gives a basic overview of the task as well as options to edit and delete the task. Clicking the edit button will bring the user to the Add Dialog activity but with data already filled in, the exit button will bring the user back to the Main Activity and deleting will delete the task. Here, the ability to click on the checkbox to update the status is disabled.

3.4 Deleting a Task



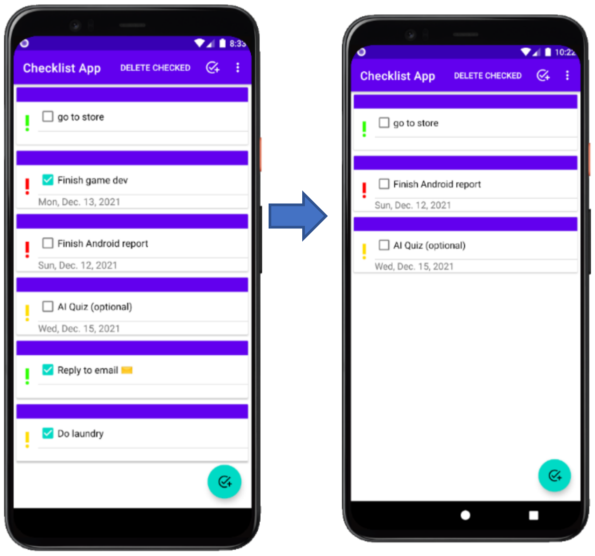
Delete Task?

Are you sure you want to delete this task?

When clicking on the delete button in the View Dialog, an alert message will prompt up asking if the user is sure they want to delete. Clicking on okay will promptly delete the task from the database. This achieved by making an alert dialog that pops up when button is clicked.

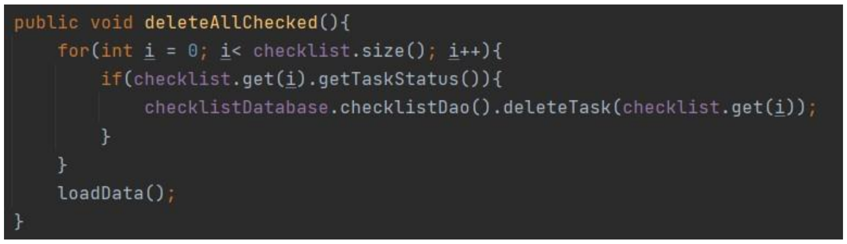
3.5 Deleting Multiple Completed Tasks

In the Main activity in the toolbar, there is an option called "Delete Checked." This will delete all completed/checked tasks. Before deleting, an alert message will pop up just like when deleting a single task asking for user confirmation that this is what they want to do. Clicking Okay will clear the list with only uncompleted/unchecked tasks remaining.



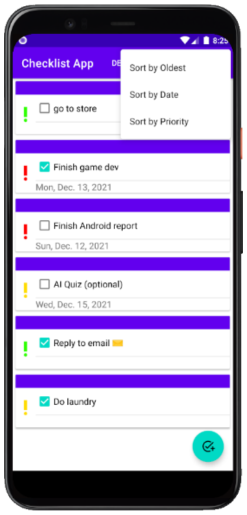
Trying to delete from an empty list will pop up a toast message letting the user know it can't do so.

Originally before I implemented the database, I used ArrayLists's removeIf() method that uses a filter to find elements that match specified conditions. Instead of that, it goes through a simple for loop to find all finished/checked tasks and deletes them one at a time:

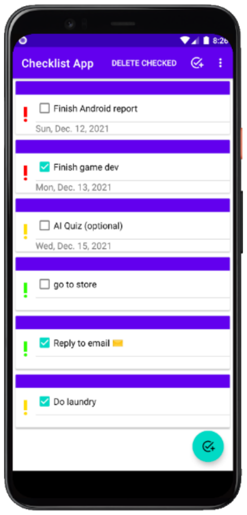

```java
public void deleteAllChecked(){
    for(int i = 0; i< checklist.size(); i++){
        if(checklist.get(i).getTaskStatus()){
            checklistDatabase.checklistDao().deleteTask(checklist.get(i));
        }
    }
    loadData();
}
```

3.6 SORTING THE LIST

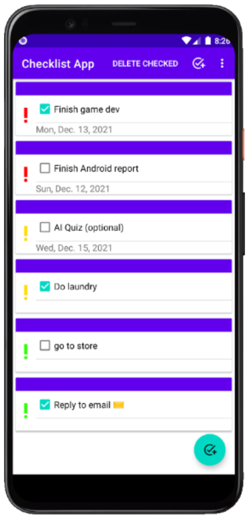
In the top right corner is a drop-down list that allows the user to sort their tasks by three options: by oldest-to-newest, by due dates (with unspecified dates on the bottom), and by highest-to-lowest priority. Once the user selects an option, the list will automatically be sorted even when adding new tasks.



Sort by Oldest (default)



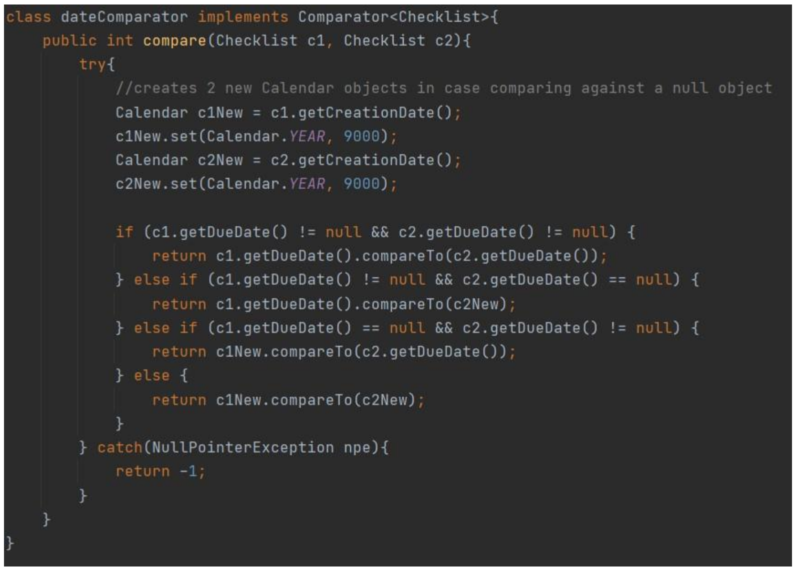
Sort by Date



Sort by Priority

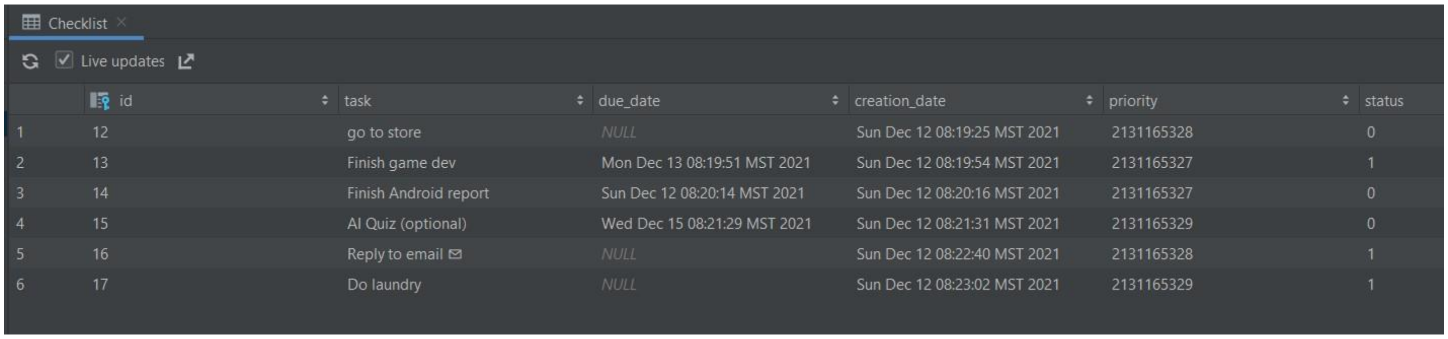
This is achieved by using comparators which are insides the Checklist.java file. The Sort by Oldest and Sort by Priority are straightforward and takes advantage of Integer's compare method. Sort by Oldest uses the database's id since it is always incrementing in number every time a new item is created and priority is sorted by an assigned number the priority is given (1 is urgent, 2 is moderate, 3 is low).

Sorting by Date however is far more involved than that. Although Calendar has a compare method, it does not handle null values. So, I had to create a Calendar instance set to the year 9000 for tasks without due dates to be sorted at the bottom of the list.


```java
class dateComparator implements Comparator<Checklist>{
    public int compare(Checklist c1, Checklist c2){
        try{
            //creates 2 new Calendar objects in case comparing against a null object
            Calendar c1New = c1.getCreationDate();
            c1New.set(Calendar.YEAR, 9000);
            Calendar c2New = c2.getCreationDate();
            c2New.set(Calendar.YEAR, 9000);

            if (c1.getDueDate() != null && c2.getDueDate() != null) {
                return c1.getDueDate().compareTo(c2.getDueDate());
            } else if (c1.getDueDate() != null && c2.getDueDate() == null) {
                return c1.getDueDate().compareTo(c2New);
            } else if (c1.getDueDate() == null && c2.getDueDate() != null) {
                return c1New.compareTo(c2.getDueDate());
            } else {
                return c1New.compareTo(c2New);
            }
        } catch(NullPointerException npe){
            return -1;
        }
    }
}
```

A toast message will pop up if the user tries to sort an empty list specifying that there is nothing to sort.

3.7 Saving and Loading (Database)

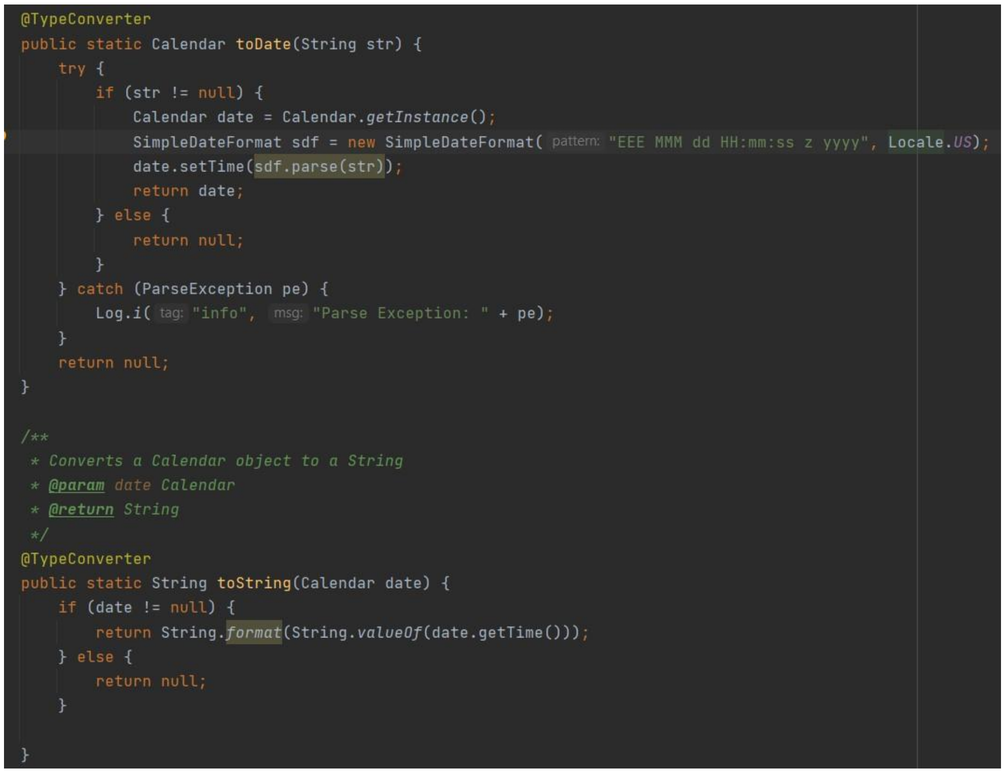


Checklist

Live updates

	id	task	due_date	creation_date	priority	status
1	12	go to store	NULL	Sun Dec 12 08:19:25 MST 2021	2131165328	0
2	13	Finish game dev	Mon Dec 13 08:19:51 MST 2021	Sun Dec 12 08:19:54 MST 2021	2131165327	1
3	14	Finish Android report	Sun Dec 12 08:20:14 MST 2021	Sun Dec 12 08:20:16 MST 2021	2131165327	0
4	15	AI Quiz (optional)	Wed Dec 15 08:21:29 MST 2021	Sun Dec 12 08:21:31 MST 2021	2131165329	0
5	16	Reply to email 🖂	NULL	Sun Dec 12 08:22:40 MST 2021	2131165328	1
6	17	Do laundry	NULL	Sun Dec 12 08:23:02 MST 2021	2131165329	1

To save and load the data, I used the Room Persistence Library to create a simple database. This database only has one primary key, which is the id of the task data. The Main Activity communicates with the database and loads data from it after each change (whether it be adding, updating, or deleting). The only tricky part in implementing the database was converting Calendar to a parse string and vise versa. For this, I created a class called Converter that handles all conversions.



```java
@TypeConverter
public static Calendar toDate(String str) {
    try {
        if (str != null) {
            Calendar date = Calendar.getInstance();
            SimpleDateFormat sdf = new SimpleDateFormat( pattern: "EEE MMM dd HH:mm:ss z yyyy", Locale.US);
            date.setTime(sdf.parse(str));
            return date;
        } else {
            return null;
        }
    } catch (ParseException pe) {
        Log.i( tag: "info",  msg: "Parse Exception: " + pe);
    }
    return null;
}

/**
 * Converts a Calendar object to a String
 * @param date Calendar
 * @return String
 */
@TypeConverter
public static String toString(Calendar date) {
    if (date != null) {
        return String.format(String.valueOf(date.getTime()));
    } else {
        return null;
    }

}
```

The converter uses SimpleDateFormat to convert the string into the date using a parser. The parser needs a try/catch or else it cannot work.