Listify

CSC 221 - Project Report

Instructor: Albi Arapi

Nasif Rahman, Adeeb Ahmed, Sufian Ali

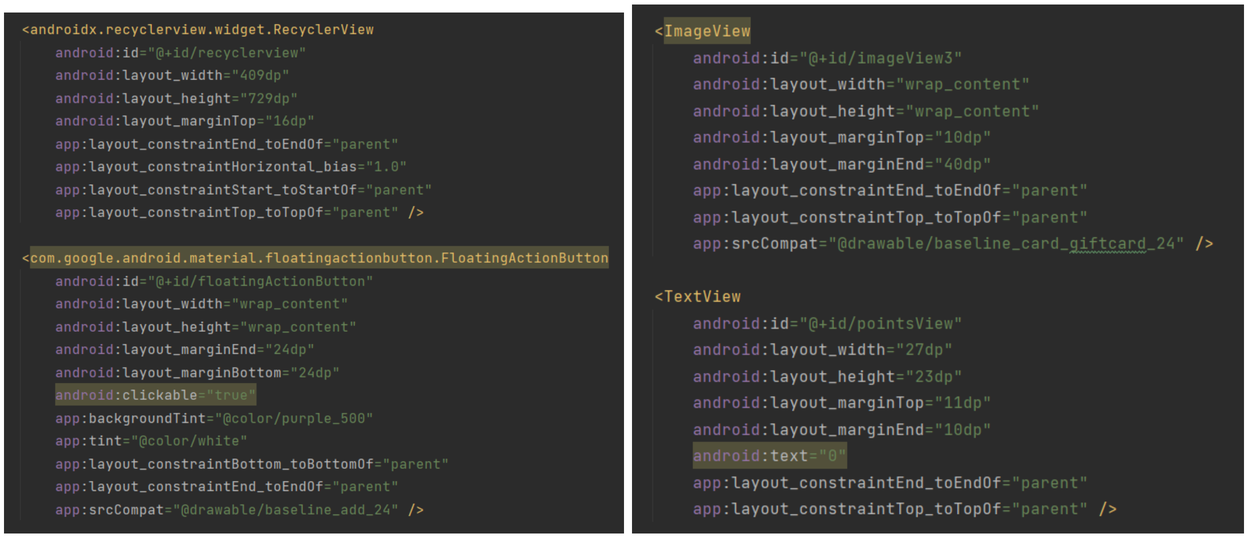
Spring 2023

<u>Project Overview:</u>

For our CSC 221 final project, we built a to-do list android application called "Listify". The app allows users to add tasks to the to-do list, set a due date for each task, edit/delete the task, save tasks in the cloud, and earn points and level up by completing tasks. For our project, we used several different software/services. For the frontend part of our project, we used Android Studio. Android Studio allowed us to design the layout, and user interface of the app. For layout and overall design, we used XML. Listify was programmed in Java language. We used Intellij as our IDE. For the backend component of our app, we used Firebase. More specifically, we used Firebase's Firestore Database to store task information.

<u>App Layout:</u>

For the app's layout, we have four different XML files for the main activity, splash activity, add new task, and each task. In our activity_main.xml, we have a Recycler View to display the task list. At the bottom right corner of the screen, we have a floating action button (FAB), and at the top right corner, we have a giftcard icon that refers to the amount of points earned. Right next to it, there's a textview displaying the number of points. The add_new_task.xml contains the layout for the add new task activity. The EditText component allows user input. There's a text button Set due date that opens up a pop-up calendar window. To the right, there's an Add button for adding the task and a mic icon for Google's Speech-to-Text function. The each_task.xml contains the layout of how each task is displayed on the list. The task name appears in black, and right underneath the due date is shown in a gray font. There’s a checkbox that allows users to mark a task as done.



```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="409dp"
    android:layout_height="729dp"
    android:layout_marginTop="16dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="1.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/floatingActionButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="24dp"
    android:layout_marginBottom="24dp"
    android:clickable="true"
    app:backgroundTint="@color/purple_500"
    app:tint="@color/white"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:srcCompat="@drawable/baseline_add_24" />
```

```
<ImageView
    android:id="@+id/imageView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:layout_marginEnd="40dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/baseline_card_giftcard_24" />

<TextView
    android:id="@+id/pointsView"
    android:layout_width="27dp"
    android:layout_height="23dp"
    android:layout_marginTop="11dp"
    android:layout_marginEnd="10dp"
    android:text="0"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Figure 1.1: Main Activity components (recyclerview, floating action button, imageview, and textview.

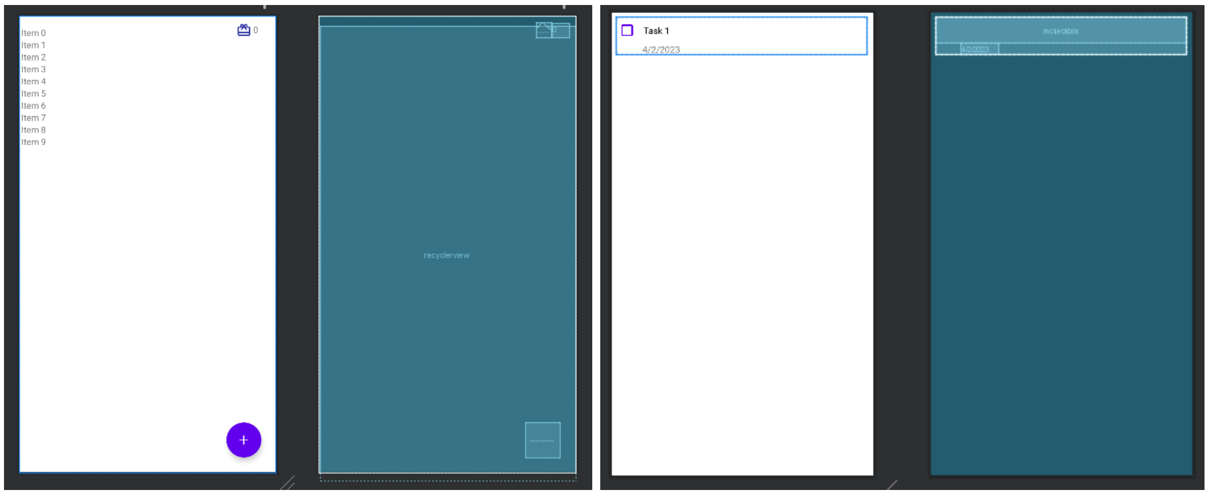


Figure 1.2: Design of MainActivity and each task

<u>MainActivity Class:</u>

The MainActivity class serves as the main entry point for the Listify application. It extends the AppCompatActivity class and implements the OnDialogCloseListener interface, allowing it to handle dialog close events.

Initialization and Layout Setup

The code initializes various UI components and variables required for the application. These include the RecyclerView, FloatingActionButton, FirebaseFirestore (Firebase Firestore instance), and SharedPreferences (for storing the number of points earned locally on the device).

RecyclerView and Adapter Setup

The RecyclerView is an essential component that displays a list of tasks. It is configured with a LinearLayoutManager to organize the tasks vertically. The ToDoAdapter is a custom adapter class responsible for binding the data from the List of ToDoModel objects to the RecyclerView.

Floating Action Button (FAB) Listener

The FloatingActionButton (FAB) is responsible for triggering the addition of new tasks. The onClick listener is set on the FAB, which opens a dialog (AddNewTask) allowing users to input details for a new task.

Data Retrieval and Display

The showData() method retrieves task data from the Firestore database. A Firestore query is created to fetch tasks from the "task" collection, ordered by time in descending order. The addSnapshotListener listens for changes in the database and triggers the onEvent method when changes occur. For each added document, the documentChange loop processes the data and adds the tasks to the mList list. The adapter.notifyDataSetChanged() method is called to reflect the changes in the RecyclerView.

TouchHelper for Item Interaction

ItemTouchHelper is an Android class that enables user interactions with the RecyclerView items, such as dragging and swiping to perform actions. The TouchHelper class extends

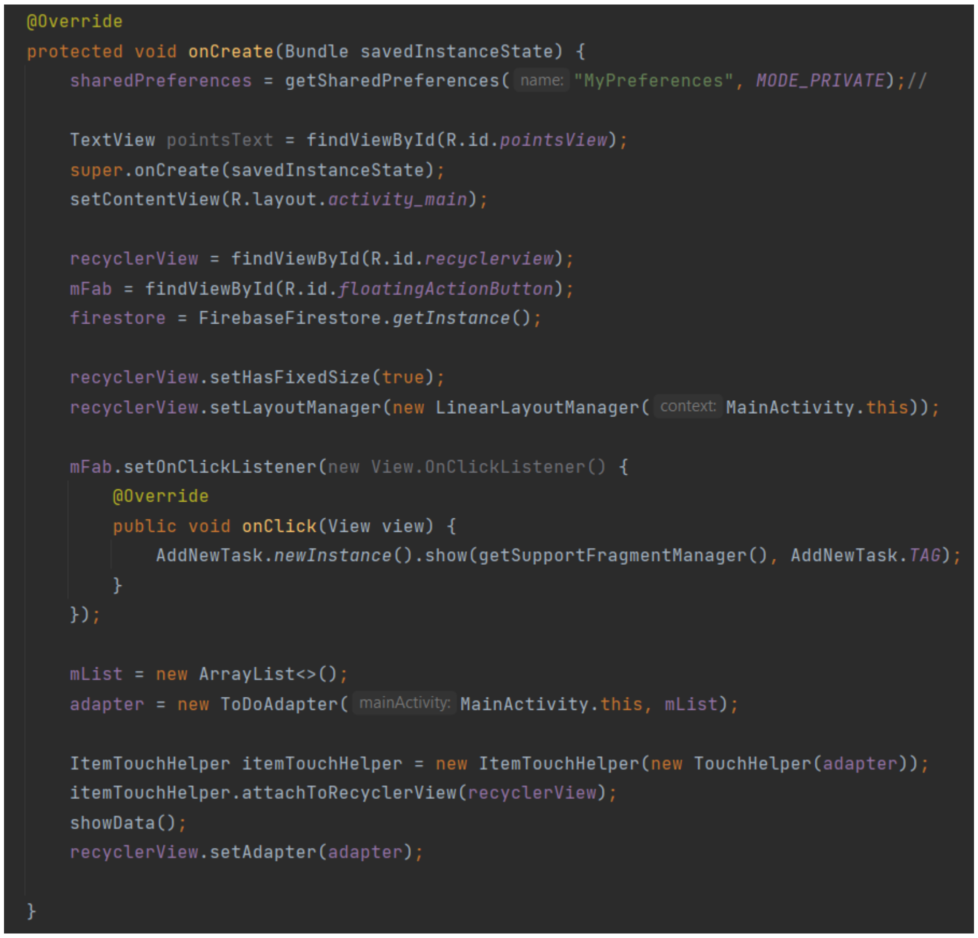
ItemTouchHelper.Callback and is responsible for handling these item interactions, such as deleting or updating a task.

Dialog Close Event Handling

The onDialogClose() method implements the OnDialogCloseListener interface. It is called when the dialog for adding a new task is closed. This method clears the mList list, retrieves the updated task data, and notifies the adapter to update the RecyclerView.

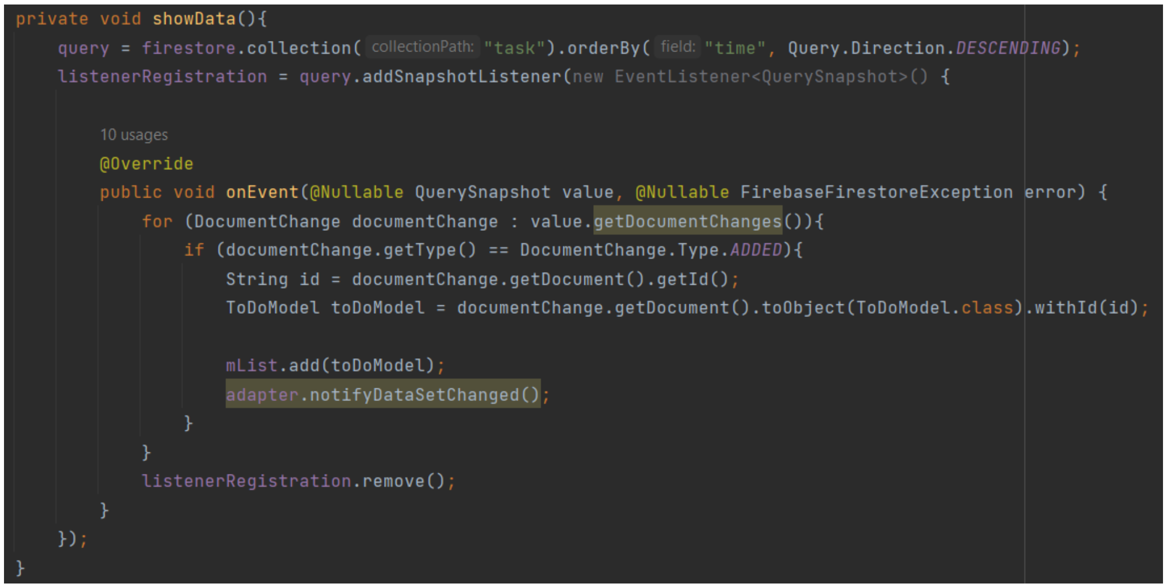
Shared Preferences for Persistence

The onPause() and onResume() methods utilize SharedPreferences to persistently store and retrieve data. In this case, it saves and retrieves the text value of a TextView named "pointsView" that displays points associated with the tasks.



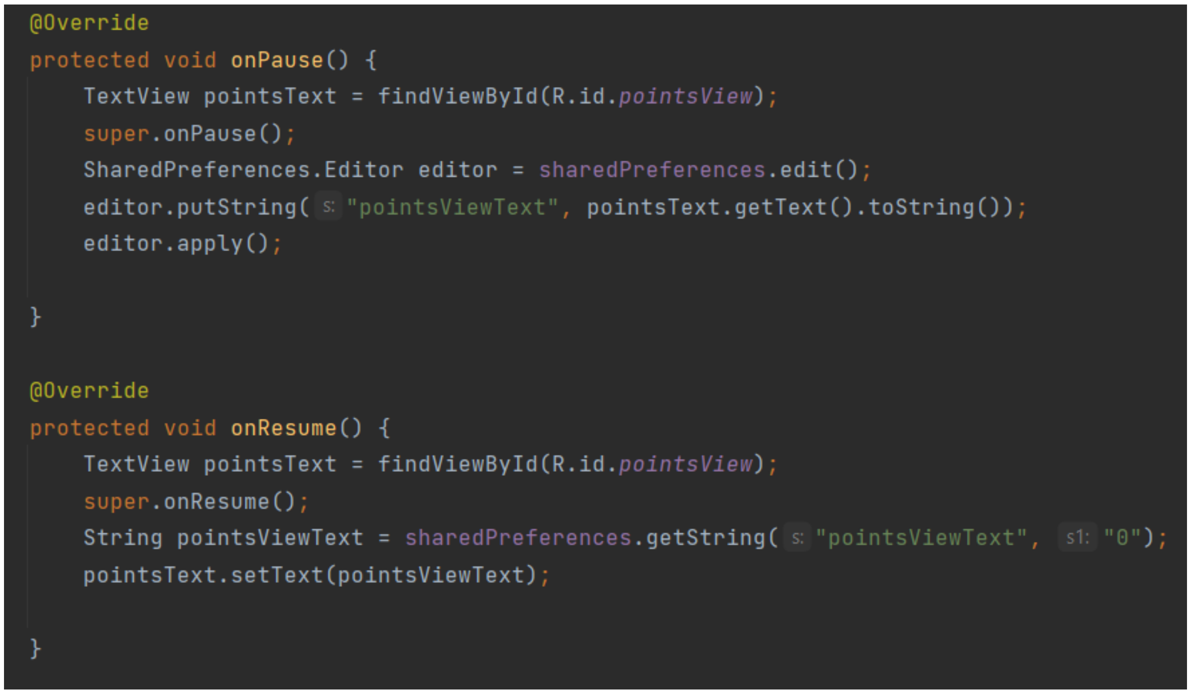
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    sharedPreferences = getSharedPreferences( name: "MyPreferences", MODE_PRIVATE);//

    TextView pointsText = findViewById(R.id.pointsView);
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    recyclerView = findViewById(R.id.recyclerview);
    mFab = findViewById(R.id.floatingActionButton);
    firestore = FirebaseFirestore.getInstance();

    recyclerView.setHasFixedSize(true);
    recyclerView.setLayoutManager(new LinearLayoutManager( context: MainActivity.this));

    mFab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            AddNewTask.newInstance().show(getSupportFragmentManager(), AddNewTask.TAG);
        }
    });

    mList = new ArrayList<>();
    adapter = new ToDoAdapter( mainActivity: MainActivity.this, mList);

    ItemTouchHelper itemTouchHelper = new ItemTouchHelper(new TouchHelper(adapter));
    itemTouchHelper.attachToRecyclerView(recyclerView);
    showData();
    recyclerView.setAdapter(adapter);

}
```

Figure 2.1: The onCreate function that initialized variables and sets onClickListener for the FAB



```
private void showData(){
    query = firestore.collection( collectionPath: "task").orderBy( field: "time", Query.Direction.DESCENDING);
    listenerRegistration = query.addSnapshotListener(new EventListener<QuerySnapshot>() {

        10 usages
        @Override
        public void onEvent(@Nullable QuerySnapshot value, @Nullable FirebaseFirestoreException error) {
            for (DocumentChange documentChange : value.getDocumentChanges()){
                if (documentChange.getType() == DocumentChange.Type.ADDED){
                    String id = documentChange.getDocument().getId();
                    ToDoModel toDoModel = documentChange.getDocument().toObject(ToDoModel.class).withId(id);

                    mList.add(toDoModel);
                    adapter.notifyDataSetChanged();
                }
            }
            listenerRegistration.remove();
        }
    });
}
```

Figure 2.2: Function to retrieve task information from Firebase database



```
@Override
protected void onPause() {
    TextView pointsText = findViewById(R.id.pointsView);
    super.onPause();
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString( s: "pointsViewText", pointsText.getText().toString());
    editor.apply();

}

@Override
protected void onResume() {
    TextView pointsText = findViewById(R.id.pointsView);
    super.onResume();
    String pointsViewText = sharedPreferences.getString( s: "pointsViewText",  s1: "0");
    pointsText.setText(pointsViewText);

}
```

Figure 2.3: Functions to store the amount of points earned locally when the app is closed and retrieve the value when the app is reopened

AddNewTask Class:

The AddNewTask class in our project is responsible for displaying a bottom sheet dialog where users can add or update tasks.

Class Overview

The AddNewTask class extends the BottomSheetDialogFragment class and serves as the entry point for adding or updating tasks in the Listify application. It provides a user interface for entering task details, setting due dates, and saving or updating tasks in the Firestore database.

View Initialization and Layout Setup

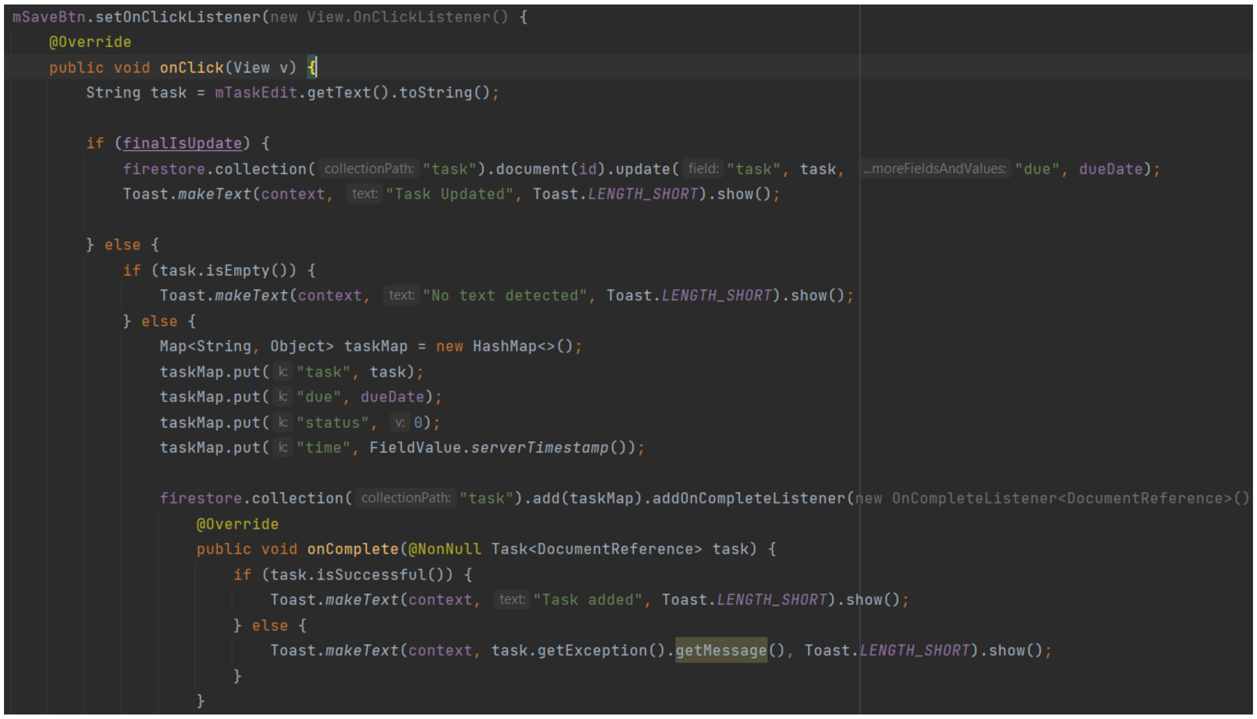
The onCreateView() method inflates the add_new_task layout and returns the corresponding view. This layout contains various UI components such as TextViews, EditTexts, Buttons, and an ImageView for speech recognition.

Speech Recognition

The micBtn ImageView is responsible for triggering the speech recognition feature. When clicked, it creates an intent to start the speech recognition activity. The result is obtained in the onActivityResult() method, where the recognized speech text is retrieved and set as the task text in the EditText.

Task Editing and Saving

The mTaskEdit EditText allows users to enter the task description. The mSaveBtn Button is responsible for saving or updating the task. If the class is in the update mode (isUpdate = true), the Firestore document is updated with the new task and due date. Otherwise, a new task document is created in the Firestore collection with the provided details.

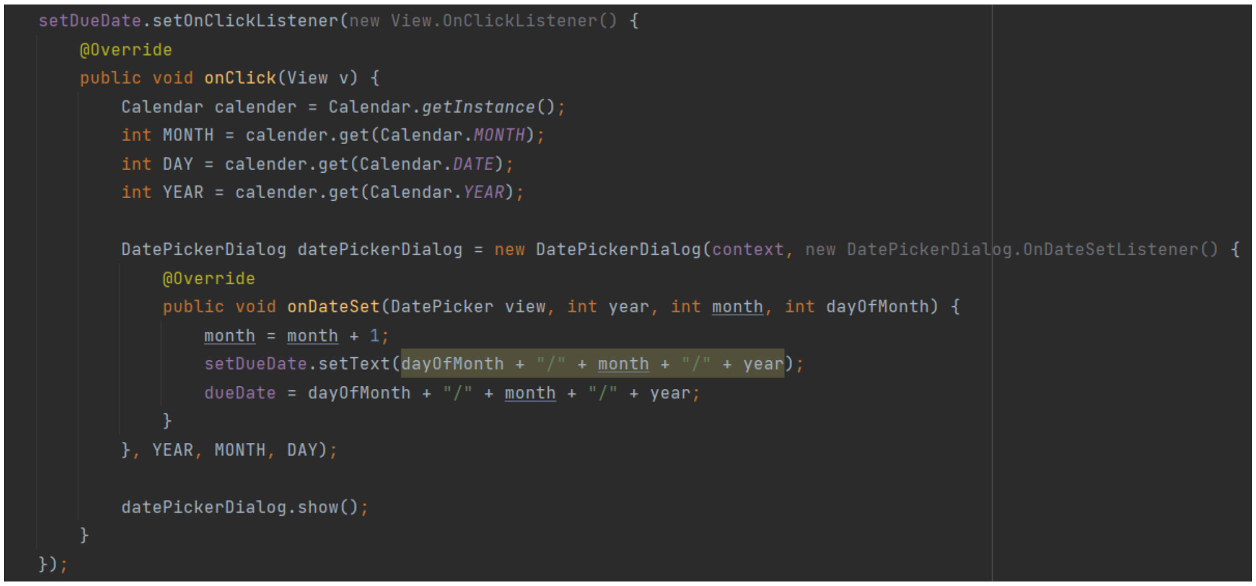


```
mSaveBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String task = mTaskEdit.getText().toString();

        if (finalIsUpdate) {
            firestore.collection( collectionPath: "task").document(id).update( field: "task", task,  ...moreFieldsAndValues: "due", dueDate);
            Toast.makeText(context,  text: "Task Updated", Toast.LENGTH_SHORT).show();

        } else {
            if (task.isEmpty()) {
                Toast.makeText(context,  text: "No text detected", Toast.LENGTH_SHORT).show();
            } else {
                Map<String, Object> taskMap = new HashMap<>();
                taskMap.put( k: "task", task);
                taskMap.put( k: "due", dueDate);
                taskMap.put( k: "status",  v: 0);
                taskMap.put( k: "time", FieldValue.serverTimestamp());

                firestore.collection( collectionPath: "task").add(taskMap).addOnCompleteListener(new OnCompleteListener<DocumentReference>()
                    @Override
                    public void onComplete(@NonNull Task<DocumentReference> task) {
                        if (task.isSuccessful()) {
                            Toast.makeText(context,  text: "Task added", Toast.LENGTH_SHORT).show();
                        } else {
                            Toast.makeText(context, task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                        }
                    }
```

Figure 3.1: Function to save task information in Firestore database

Due Date Selection

The setDueDate TextView allows users to set a due date for the task. When clicked, a DatePickerDialog is displayed, allowing users to select the desired date. The chosen date is then displayed in the setDueDate TextView and stored in the dueDate variable.



```
setDueDate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Calendar calender = Calendar.getInstance();
        int MONTH = calender.get(Calendar.MONTH);
        int DAY = calender.get(Calendar.DATE);
        int YEAR = calender.get(Calendar.YEAR);

        DatePickerDialog datePickerDialog = new DatePickerDialog(context, new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
                month = month + 1;
                setDueDate.setText(dayOfMonth + "/" + month + "/" + year);
                dueDate = dayOfMonth + "/" + month + "/" + year;
            }
        }, YEAR, MONTH, DAY);

        datePickerDialog.show();
    }
});
```

Figure 3.2: Function for setting the dueDate

Text Validation and UI Updates

The mTaskEdit EditText is monitored using a TextWatcher. If the text is empty, the mSaveBtn Button is disabled and its background color is set to gray. When the user enters text, the button is enabled and the background color is changed to a custom green-blue color.

Communication with MainActivity

The onAttach() method is overridden to establish a connection between the AddNewTask dialog and the MainActivity. This allows the dialog to communicate with the activity and trigger the onDialogClose() method when it is dismissed.

ToDoAdapter Class:

The ToDoAdapter class in the Listify project is responsible for binding the task data to the RecyclerView in the MainActivity.

Class Overview

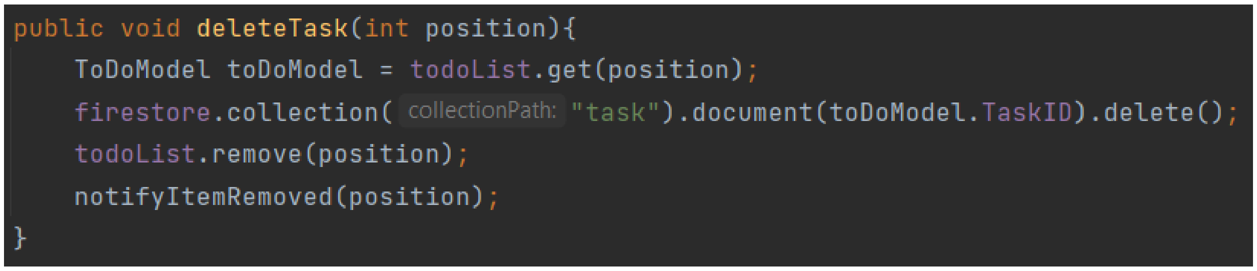
The ToDoAdapter class extends RecyclerView.Adapter<MyViewHolder> and acts as the adapter for the RecyclerView in the MainActivity. It handles the creation of view holders, binding data to the views, and implementing various interactions such as task deletion, task editing, and checkbox status changes.

View Holder Creation and Layout Inflation

The onCreateViewHolder() method is responsible for inflating the each_task layout and creating an instance of the MyViewHolder class. The layout contains a CheckBox for task completion, a TextView for the due date, and other UI elements.

Task Deletion

The deleteTask() method allows the user to delete a task from the RecyclerView. It retrieves the corresponding ToDoModel object from the todoList, deletes the associated Firestore document, removes the task from the list, and notifies the adapter of the item removal.

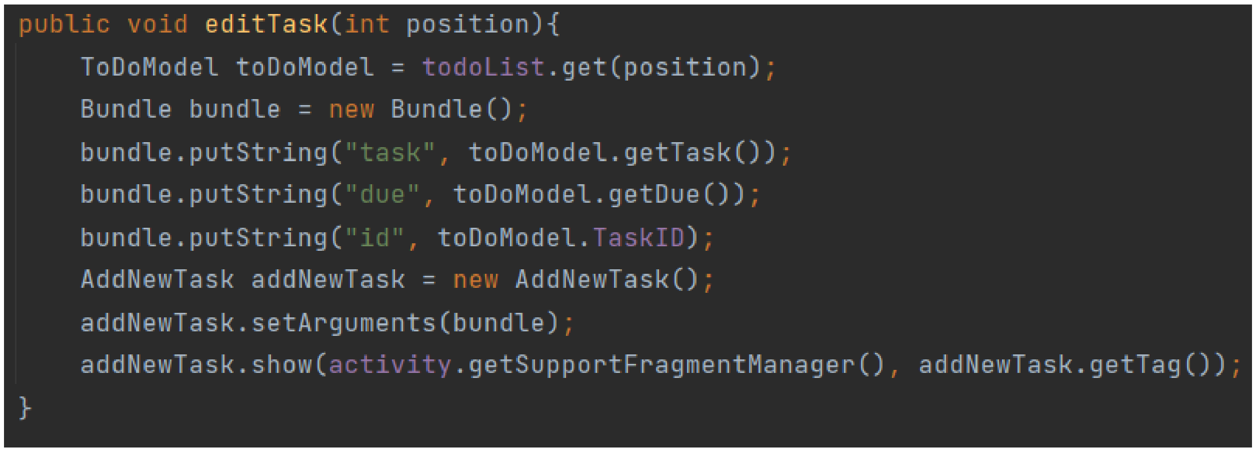


```java
public void deleteTask(int position){
    ToDoModel toDoModel = todoList.get(position);
    firestore.collection( collectionPath: "task").document(toDoModel.TaskID).delete();
    todoList.remove(position);
    notifyItemRemoved(position);
}
```

Figure 4.1: Function to remove a task from the list and from Firestore database

Task Editing

The editTask() method enables the user to edit a task. It retrieves the selected ToDoModel object from the todoList, creates a Bundle containing the task details, and passes it to an instance of the AddNewTask class. The AddNewTask dialog is then shown, allowing the user to edit the task.



```java
public void editTask(int position){
    ToDoModel toDoModel = todoList.get(position);
    Bundle bundle = new Bundle();
    bundle.putString("task", toDoModel.getTask());
    bundle.putString("due", toDoModel.getDue());
    bundle.putString("id", toDoModel.TaskID);
    AddNewTask addNewTask = new AddNewTask();
    addNewTask.setArguments(bundle);
    addNewTask.show(activity.getSupportFragmentManager(), addNewTask.getTag());
}
```

Figure 4.2: Function for editing an existing task and updating it on Firebase

Binding Data to Views

The onBindViewHolder() method binds the data from the ToDoModel object to the views within the MyViewHolder. It sets the task description, due date, and checkbox status based on the values stored in the todoList.

Checkbox Status Changes

The mCheckBox CheckBox in each view holder represents the completion status of the task. When the checkbox state changes, the onCheckedChanged() listener is triggered. If the checkbox is checked, the Firestore document's status field is updated, and a TextView representing the points view is updated accordingly. Additionally, if the points reach certain thresholds (100, 200, 300), an AlertDialog is shown to notify the user of their progress.

Data Conversion

The toBoolean() method converts the integer status value stored in the ToDoModel object to a boolean value, indicating whether the task is completed or not.

Things We Could Improve On:

- Add an authentication feature so that each user would have their own task list that would be retrieved from firebase.
- Reward users with new themes/coupons for leveling up.
- A leaderboard showing user names with most points earned.
- A notification system that would inform the user when the due date of a task has arrived.

Conclusion:

In conclusion, the Listify project served as our very first project with Android Studio and Java, and it provided valuable learning experiences in mobile app development. The project aimed to create a task management app that allows users to add, edit, and complete tasks. Throughout the development process, we encountered various challenges and gained insights into Android app development. Working on the project laid a strong foundation for our future app development endeavors. We gained essential knowledge about working with activities, fragments, RecyclerView, Firestore, and various UI components. The project allowed us to grasp fundamental concepts such as data binding, event handling, and navigation within an Android application. Moving forward, we aspire to enhance our skills and create even better applications.