





# TO DO APPLICATION

STUDENT NUMBER:

19005741

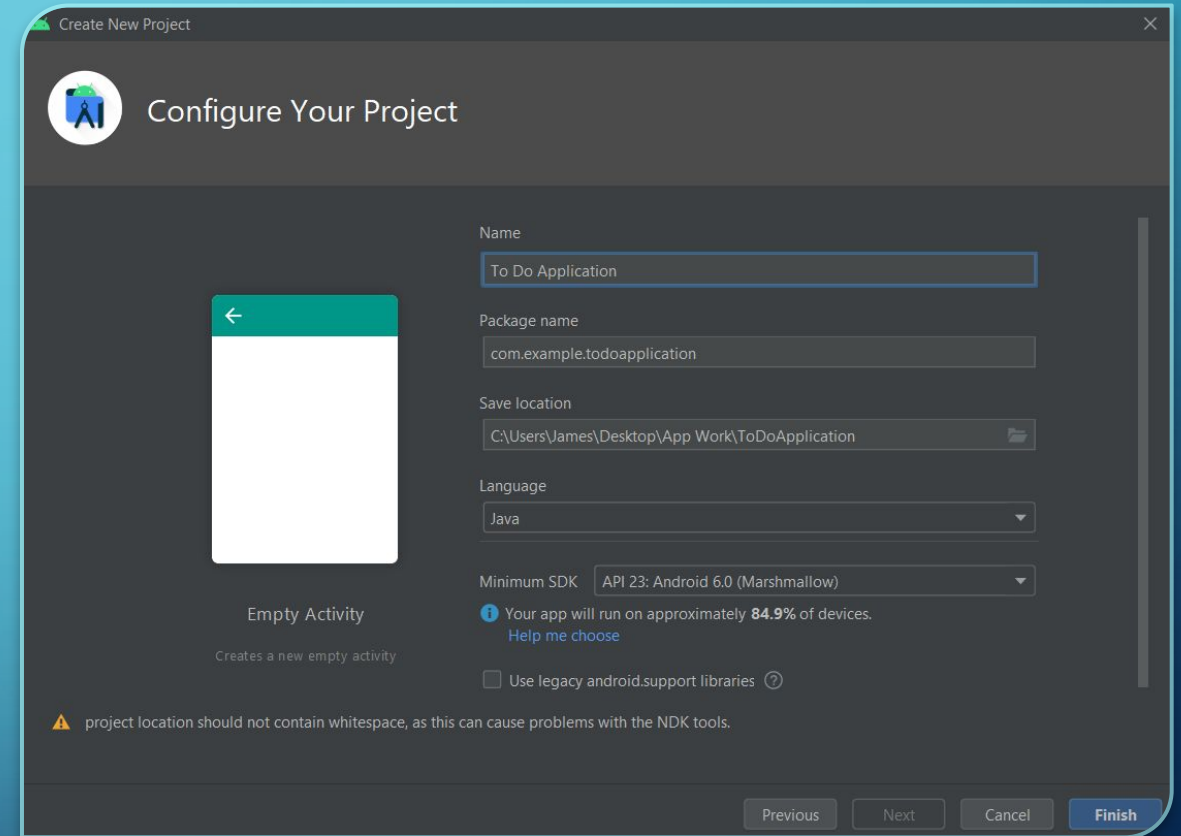


# PART 1: SETTING UP AND CODING THE APP USING ANDROID STUDIO



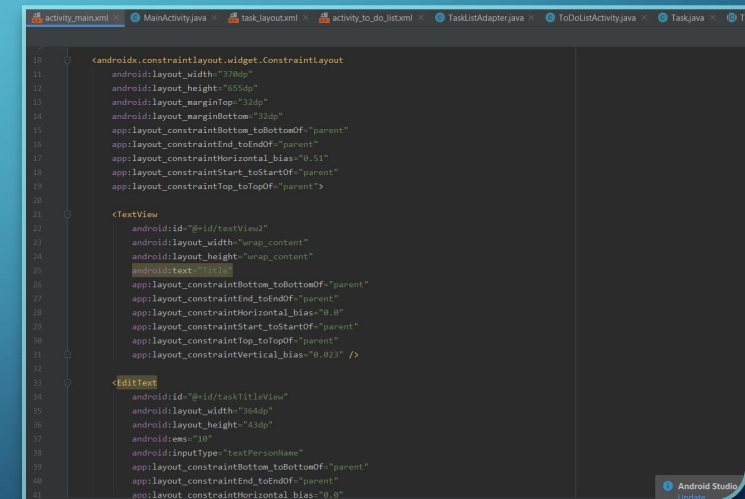
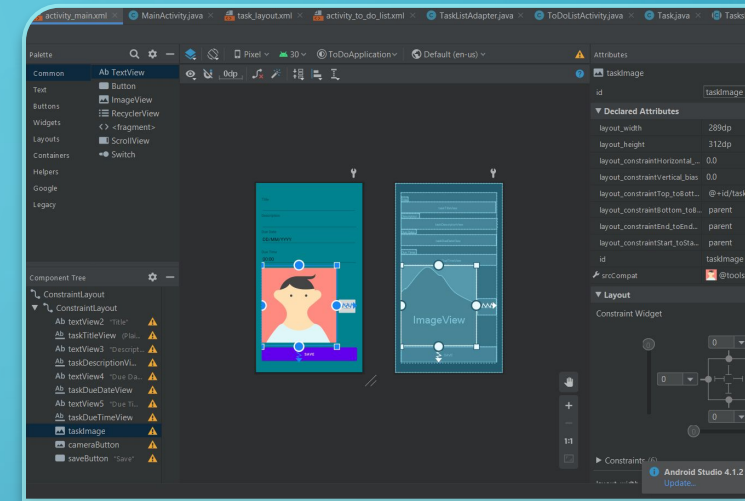
# PART 1: START UP

- On start up for using Android Studio, Empty Activity was selected and the name of the project was declared, in this case it is named ToDoApplication.
- The development for this app is any device from Android 6.0 onwards. So Android 6.0 'Marshmallow' was selected.

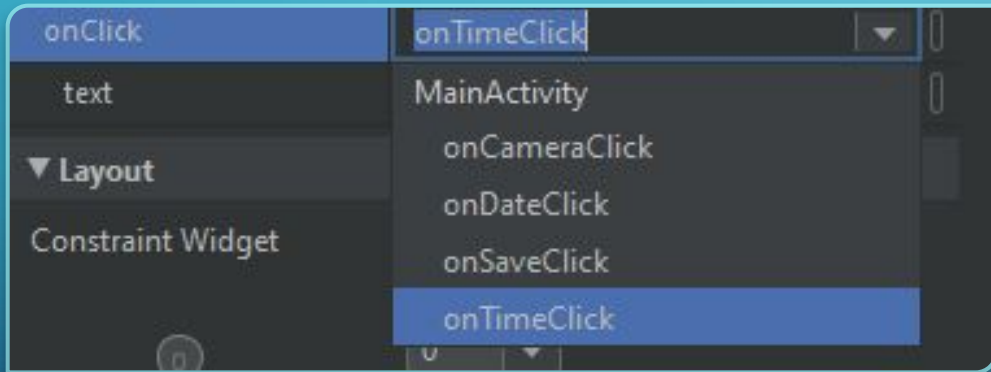


# PART 1: DESIGNING THE UI

- Android Studio allows for you to create the look of the app in two ways. 1). Pure code in the XML file 2). Via graphical drag and drop, so you can see the apps overall look being built in real time.



# PART 1: DESIGNING THE UI



- Each component of the UI allows for you to assign an `onClick` method to it. This allows for that component to be assigned to a method to carry out that specified function.
- For example, the component which acts as the time input, is linked to the function which allows you bring up clock in app and select a time.

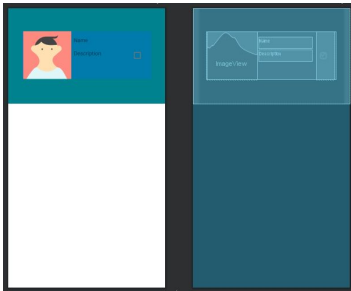
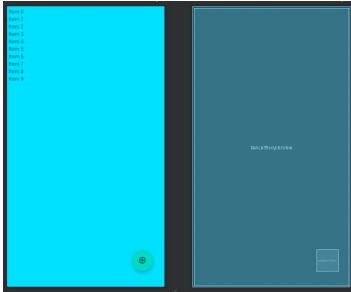
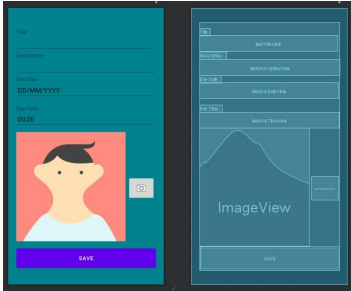
# PART 1: CODING

- On creating the Empty Activity. It comes with a Java file which houses the code of the XML file. Because this activity window houses the task logging, the Java file houses the code for both setting of the date and time of the task.

```
MainActivity.java
1 package com.example.todoapplication;
2
3 import ...
4
39
40 public class MainActivity extends AppCompatActivity {
41
42     TasksDB db;
43
44     static final int REQUEST_IMAGE_CAPTURE = 1;
45     Uri imageUri;
46
47     //Creates Form in the App
48     @Override
49     protected void onCreate(Bundle savedInstanceState) {
50         super.onCreate(savedInstanceState);
51
52         //Removes the title bar
53         requestWindowFeature(Window.FEATURE_NO_TITLE);
54         getSupportActionBar().hide();
55
56         setContentView(R.layout.activity_main);
57
58         db = Room.databaseBuilder(getApplicationContext(),
59             TasksDB.class,
60             "tasks_database_name").build();
61     }
62
63     //Logs data when Save Button is pressed
64     public void onSaveClick(View v) {
65         Log.d("ToDoApp", "onSaveClick");
66
67         EditText titleView = findViewById(R.id.taskTitleView);
68         EditText descView = findViewById(R.id.taskDescriptionView);
69
70         String title = titleView.getText().toString();
71         String description = descView.getText().toString();
72
73         //Saves Fields to the database
74         final Task task = new Task();
75         task.title = title;
76         task.description = description;
77         task.image = imageUri.toString();
78         task.done = false;
79
80         //Passes data to database
81         Executor myExecutor = Executors.newSingleThreadExecutor();
```



# PART 1: USER INTERFACE



- This was put together in the XML editor of Android Studio, using a mixture of the drag and drop features and the XML.
- The code works a lot like web scripting, which can be edited and updated graphically in real time. But the graphical design side of the editor updates the code in real time also.

## PART 1: CODING – TITLE AND TASK

- The MainActivity.java file sets what is inputted into the input boxes. The values held within the variables are passed back to the database so the task can be saved and retrieved on start up.

```
//Logs data when Save Button is pressed
public void onSaveClick(View v) {
    Log.d( tag: "ToDoApp", msg: "onSaveClick");

    EditText titleView = findViewById(R.id.taskTitleView);
    EditText descView = findViewById(R.id.taskDescriptionView);

    String title = titleView.getText().toString();
    String description = descView.getText().toString();
}
```



# PART 1: INPUT

```
//Updates the Date of the Widget
public void onClick(View view){
    Log.d( tag: "ToDoApp", msg: "onClick");

    DatePickerDialog.OnDateSetListener listener = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
            EditText dateView = findViewById(R.id.taskDueDateView);
            dateView.setText(dayOfMonth + "/" + month + "/" + year);
        }
    };

    DatePickerDialog dialog = new DatePickerDialog( context: this, listener, year: 2021, month: 1, dayOfMonth: 1);
    dialog.show();
}
```

```
//Updates the Time of the Widget
public void onClick(View view){
    Log.d( tag: "ToDoApp", msg: "onClick");

    TimePickerDialog.OnTimeSetListener listener = new TimePickerDialog.OnTimeSetListener() {
        @Override
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            EditText timeView = findViewById(R.id.taskDueTimeView);
            timeView.setText(hourOfDay + ":" + minute);
        }
    };

    TimePickerDialog dialog = new TimePickerDialog( context: this, listener, hourOfDay: 0, minute: 0, is24HourView: true);
    dialog.show();
}
```

- Once the components of the app were created, the functions of them were declared. This is located in the MainActivity.java file.
- Functions for onClick and onTimeClick are declared which update the fields in the app live, the components are linked up using the onClick selection of the xml editor.

# PART 1: CAMERA

```
//Updates the image view to display the most recent image taken
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent resultData){
    super.onActivityResult(requestCode,resultCode,resultData);

    //Sets the ImageView to the image taken in by the camera
    if(requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK){
        ImageView taskImage = findViewById(R.id.taskImage);
        taskImage.setImageURI(imageUri);
    }
}
```

```
//Method launches the camera feature of the task creation
public void onCameraClick(View view){
    Log.d( tag: "ToDoApp", msg: "onCameraClick");

    String imageFilename = "JPEG_" + System.currentTimeMillis() + ".jpg";

    File imageFile = new File(getFilesDir(),imageFilename);

    imageUri = FileProvider.getUriForFile( context: this, authority: ".fileprovider",imageFile);

    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
    startActivityForResult(takePictureIntent,REQUEST_IMAGE_CAPTURE);
}
```

- The main task creation of the app contains a button which allows for the user to take a picture of the task.
- It is passed through the linked up function onCameraClick, it passes the image through and saves it into the app.
- The image is set in an onActivityResult method, which sets it to the image box next to the camera button.

# PART 1: DATABASE

```
@Entity
public class Task {

    @PrimaryKey(autoGenerate = true)
    public int uid;

    @ColumnInfo(name = "title")
    public String title;

    @ColumnInfo(name = "description")
    public String description;

    @ColumnInfo(name = "date")
    public Long duedate;

    @ColumnInfo(name = "image")
    public String image;

    @ColumnInfo(name = "done")
    public boolean done;
}
```

```
@Database(entities = {Task.class}, version = 1)
public abstract class TasksDB extends RoomDatabase {

    public abstract TaskDAO taskDAO();

    private static final String DB_NAME = "task_database_name";
    private static TasksDB db;

    public static TasksDB getInstance(Context context){
        if(db == null) db= buildDatabaseInstance(context);
        return db;
    }

    private static TasksDB buildDatabaseInstance(Context context) {
        return Room.databaseBuilder(context, TasksDB.class, DB_NAME).build();
    }
}
```

- A class for the database was created called Task.java. This class declares the fields of the database.
- An abstract class for the database was created called TaskDB.java. This houses the code for the database itself.

# PART 1: DATABINDING

- Databinding was added to allow for the app to save the task to the landing page in a task card.
- The databinding allows for the task card to be deleted by swiping to the right. This removes the task completely from the database.
- This was achieved by adding a class called TaskListAdapter which stored the data of a task into a List, which is passed back through to the landing page. Which holds the task cards.

```
public class TaskListAdapter extends RecyclerView.Adapter<TaskListAdapter.ViewHolder> {  
  
    private List<Task> tasks;  
    private TasksDB db;  
  
    //Assigns components to held variables.  
    public class ViewHolder extends RecyclerView.ViewHolder{  
  
        TextView titleView;  
        TextView descView;  
        TextView dateView;  
        TextView timeView;  
        ImageView imageView;  
        CheckBox doneCheckBox;  
  
        public ViewHolder(@NonNull View itemView){  
            super(itemView);  
  
            titleView = itemView.findViewById(R.id.taskListTitle);  
            descView = itemView.findViewById(R.id.taskListDescription);  
            imageView = itemView.findViewById(R.id.taskListImage);  
            doneCheckBox = itemView.findViewById(R.id.taskListDone);  
        }  
    }  
  
    //Sets the task list  
    public void setTaskList(TasksDB db, List<Task> tasks){  
        this.db = db;  
        this.tasks = tasks;  
        notifyDataSetChanged();  
    }  
  
    @NonNull  
    @Override  
    public TaskListAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)  
    {  
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());  
  
        View view = inflater.inflate(R.layout.task_layout, parent, attachToRoot: false);  
  
        return new ViewHolder(view);  
    }  
}
```



# PART 1: SET UP DEBUG

- Each method of the MainActivity.java, contains a log line, which is called and can be read in the terminal of Android Studio when the relevant method is called e.g. onSaveClick.

```
Log.d( tag: "ToDoApp", msg: "onSaveClick");
```

- Breakpoints can be set in the program to allow for isolating code in order to trap unknown bugs and fix them where applicable.

```
EditText titleView = findViewById(R.id.taskTitleView);
```

# PART 1: SET UP DEBUG

- Stack Tracing is used by pressing ALT-6 and bringing up the Logcat whilst the app is being run in debug mode, and it allows for the programmer to view the methods which throw exceptions.

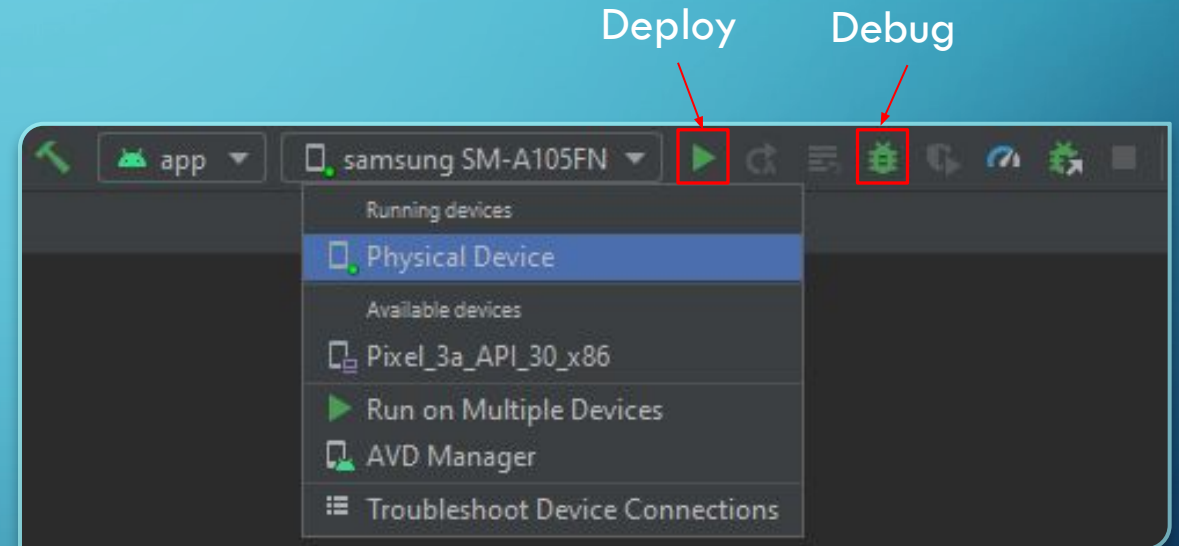
However, in the case of this code for this app, no exception throwing was used in the code. But it is an example of how to use Android Studio to fix and trace where problems arise.


```
java.io.IOException: Invalid device key response.  
    at huk.a(:com.google.android.gms@201817022@20.18.17 (040700-311416286):46)  
    at hui.a(:com.google.android.gms@201817022@20.18.17 (040700-311416286):41)  
    at hud.a(:com.google.android.gms@201817022@20.18.17 (040700-311416286):0)  
    at hug.a(:com.google.android.gms@201817022@20.18.17 (040700-311416286):10)  
    at fzn.call(:com.google.android.gms@201817022@20.18.17 (040700-311416286):5)  
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)  
    at sji.b(:com.google.android.gms@201817022@20.18.17 (040700-311416286):12)  
    at sji.run(:com.google.android.gms@201817022@20.18.17 (040700-311416286):7)  
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1167)  
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:641)  
    at spj.run(:com.google.android.gms@201817022@20.18.17 (040700-311416286):0)  
    at java.lang.Thread.run(Thread.java:923)
```



# PART 1: SET UP DEBUG AND DEPLOY

- After the initial coding of the app has taken place. Android Studio allows for you to deploy the app to either an emulator which is set to the device of a Google Pixel 3, or deploy it to a connected physical device.





# PART 2: MOBILE CONTEXT



# PART 2: MOBILE CONTEXT

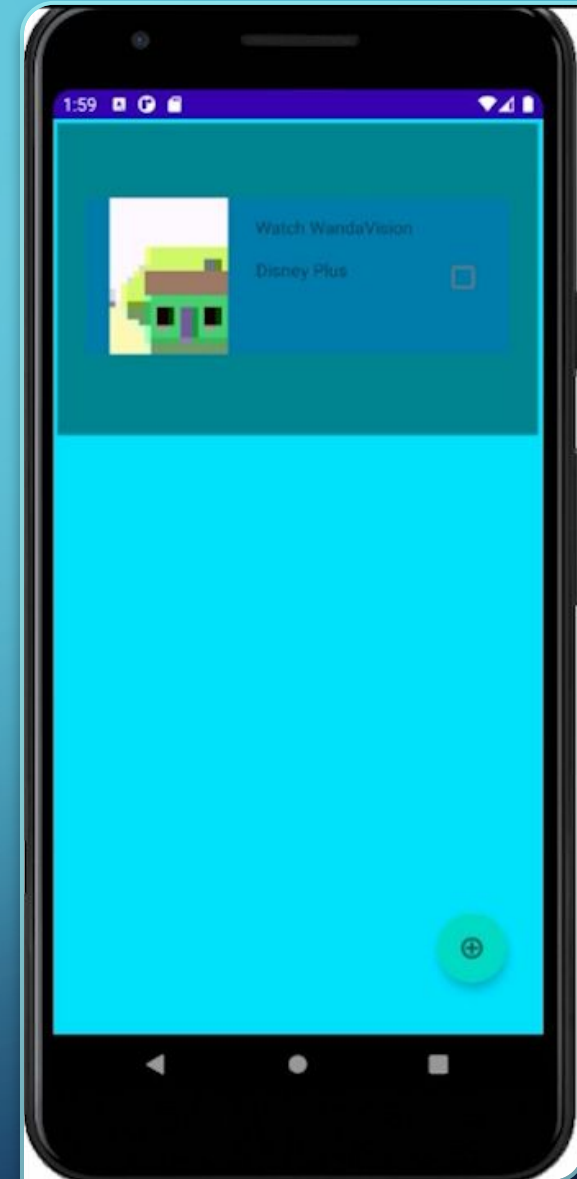
- In regards to Mobile Context, the design of this app has to be one that is fully interactable by the user using the touch screen.
- The app is designed with a horizontal orientation in mind, the features are coded to be in that orientation.
- This app is also designed with the assumption that the user-end device has a camera, which can make use of the feature. Even though more than most modern Android devices have a camera enabled.

# PART 2: MOBILE CONTEXT

- The app has fully interactable components, that allow for the user to input data into it so it can be stored as a relevant task.
- The app has a landing page which houses a list of stored tasks for the user to look through and has a create button, to allow for the user to create and log a new task.

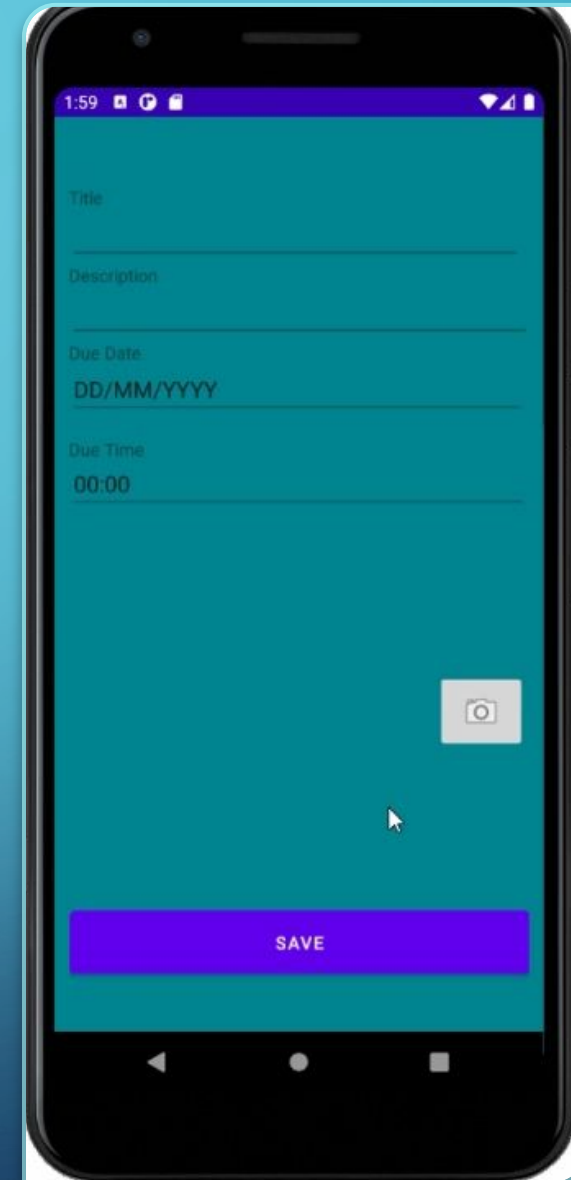
## PART 2: MOBILE CONTEXT – UI

- Whilst being interactable. The app is also minimalistic and not overbearing to a user. For example, it is very common for the button to 'create' to be in the lower right hand corner as it is closest to the thumb. For example, Twitter and email apps such as Gmail have the 'compose' or in this case create in the lower right hand corner.



## PART 2: MOBILE CONTEXT – UI

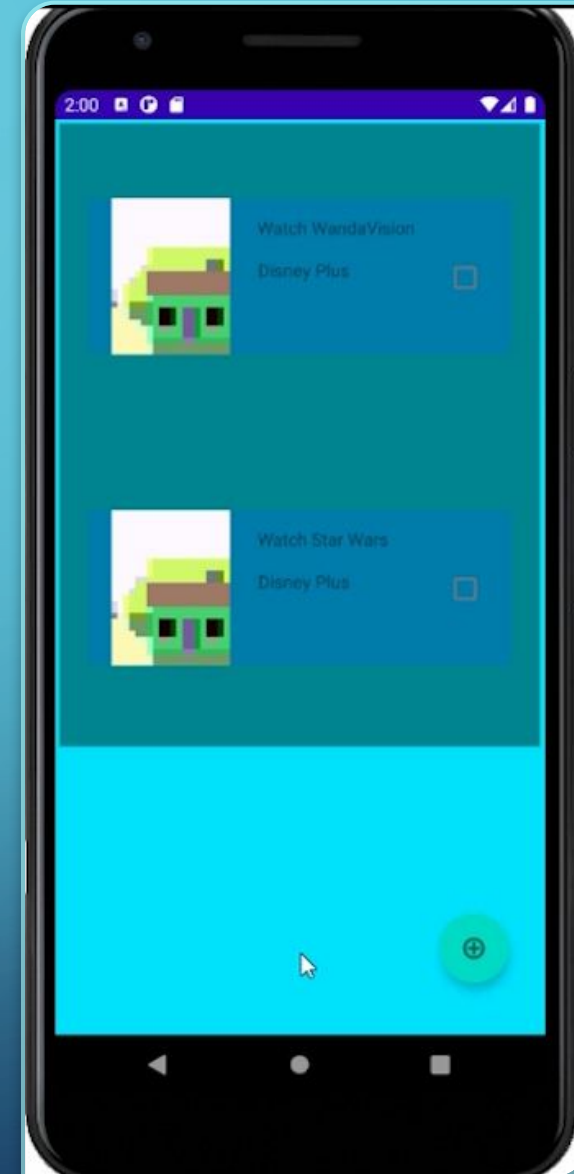
- The task creation form has fully interactable components for the user to interact with, this is so that they can set a task of their choosing with an allotted time and date, with an image to accompany it.





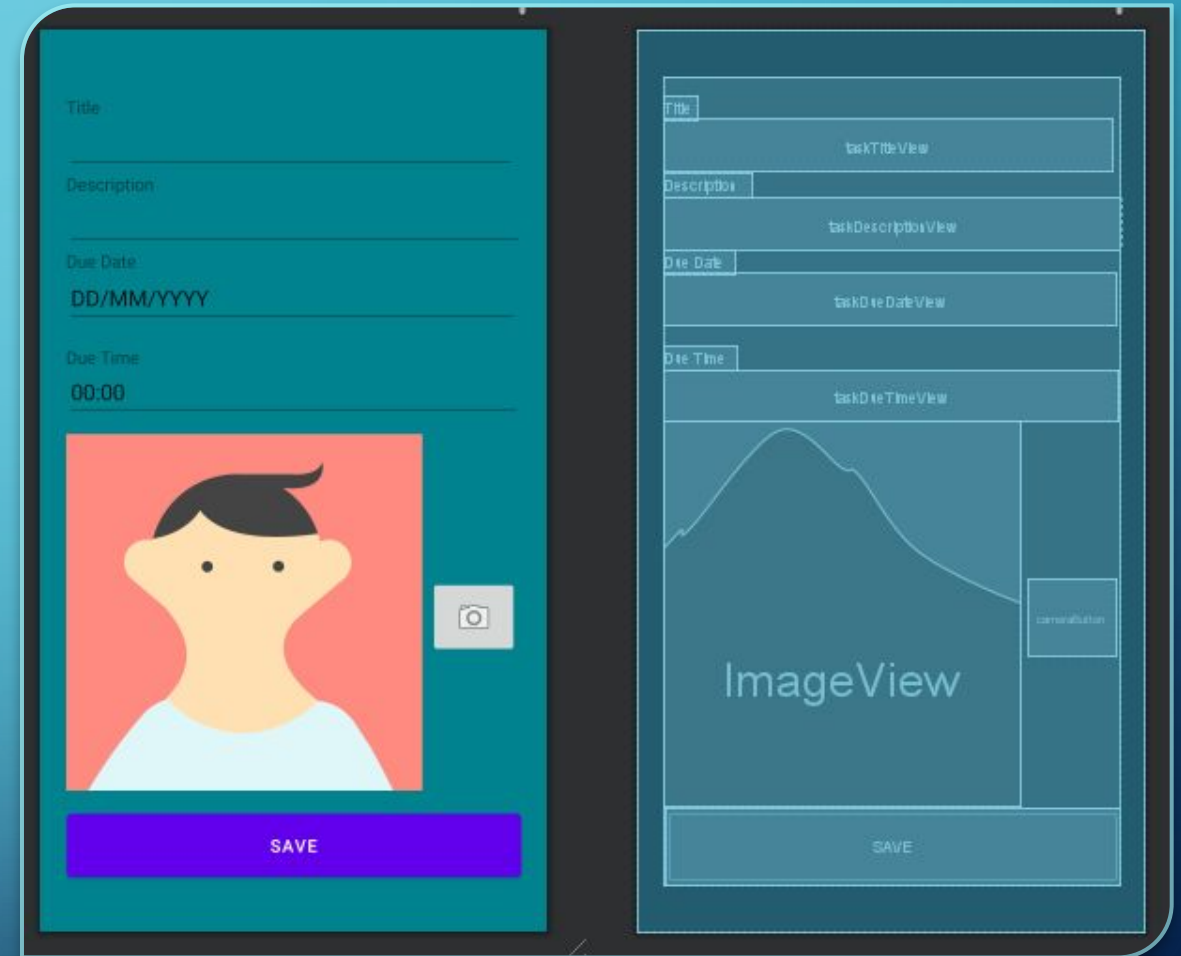
## PART 2: MOBILE CONTEXT – INTERACTION

- The app has a swipe feature, which allow to swipe (or delete) a saved task from the database.



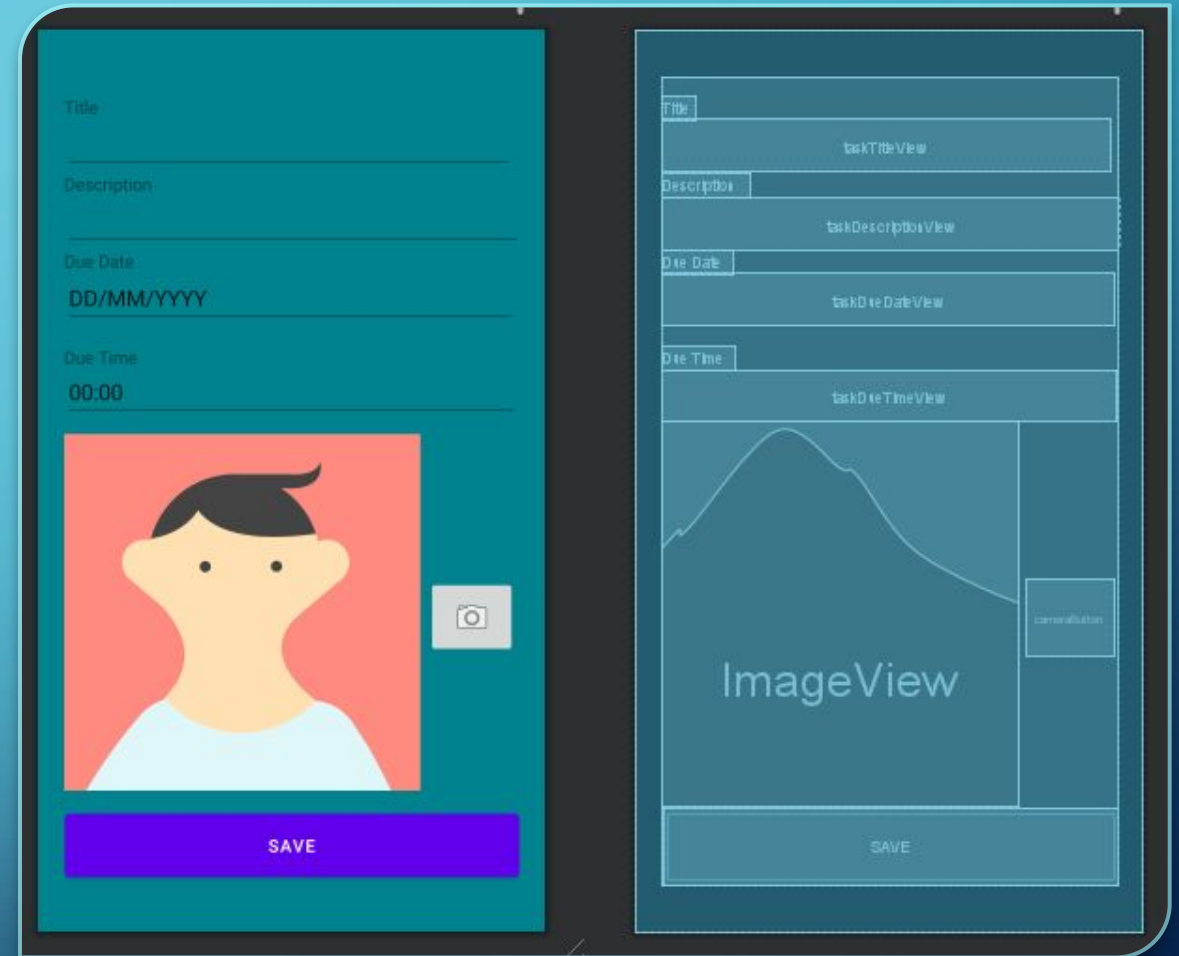
## PART 2: MOBILE CONTEXT – PRINCIPLES AND PROBLEMS

- One of the principles followed was to make sure that everything was relevant to the user. So no feature is displayed in the User Interface that is irrelevant to the user.
- Another principle is making sure each component is easily interactable with as little additional effort as possible, so no component is too small for the user to touch. A problem this raises, is making sure each component is displayed and easily interactable without compromising the usage of the others.



## PART 2: MOBILE CONTEXT – PRINCIPLES AND PROBLEMS

- Another potential problem in designing the UI, is that the components values maybe too small for the user to pickup on due to trying to make sure all the relevant features are displayed in a single form. This potential problem doesn't seem to have prevailed as each feature is well sized and well placed in the design of the UI.



An abstract graphic on the left side of the slide, consisting of white lines and circles on a blue background, resembling a circuit board or a stylized tree structure.

# PART 3: COMPONENTS

A single vertical white line located on the right side of the slide.

## PART 3: COMPONENTS

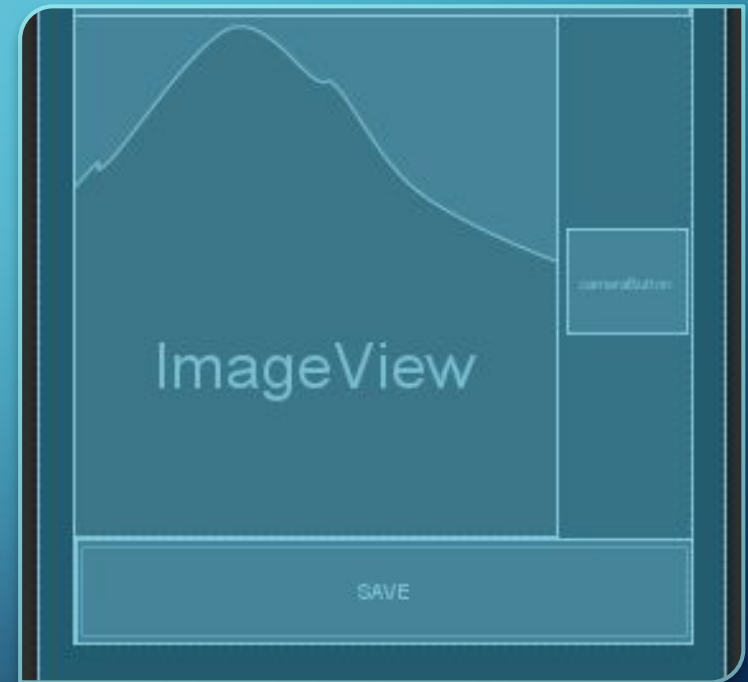
- The Create Task UI consists of:
- Title – which allows the user to enter the title of their task.
- Description – like Title, this stores a string which acts as a short summary of the task at hand.
- Date – this allows for the user to input a due date.
- Time – this allows for the user to enter a due time.

The diagram illustrates the 'Create Task' UI form, which is a vertical stack of four input fields. Each field is labeled on the left and associated with a specific view component on the right:

- Title**: Associated with `taskTitleView`.
- Description**: Associated with `taskDescriptionView`.
- Due Date**: Associated with `taskDueDateView`.
- Due Time**: Associated with `taskDueTimeView`.

## PART 3: COMPONENTS

- The Create Task UI consists of (cont.):
- Camera Button – This button is clicked to bring up the camera function of the app.
- Image Box – this updates when an picture of the task is taken.
- Save Button – this is pressed to save the currently held data inputted into the components of the Create Task form.





## PART 3: XML CONSTRAINTS

- Constraints are used in each of the XML forms. They are used to align each of the component housed in the form, from both horizontal and vertical directions. It aligns it as you wish. If constraints are not implemented each component will be automatically set to the top left hand corner by default, so constraints are important as they align to the screen of the device.

### Component Tree

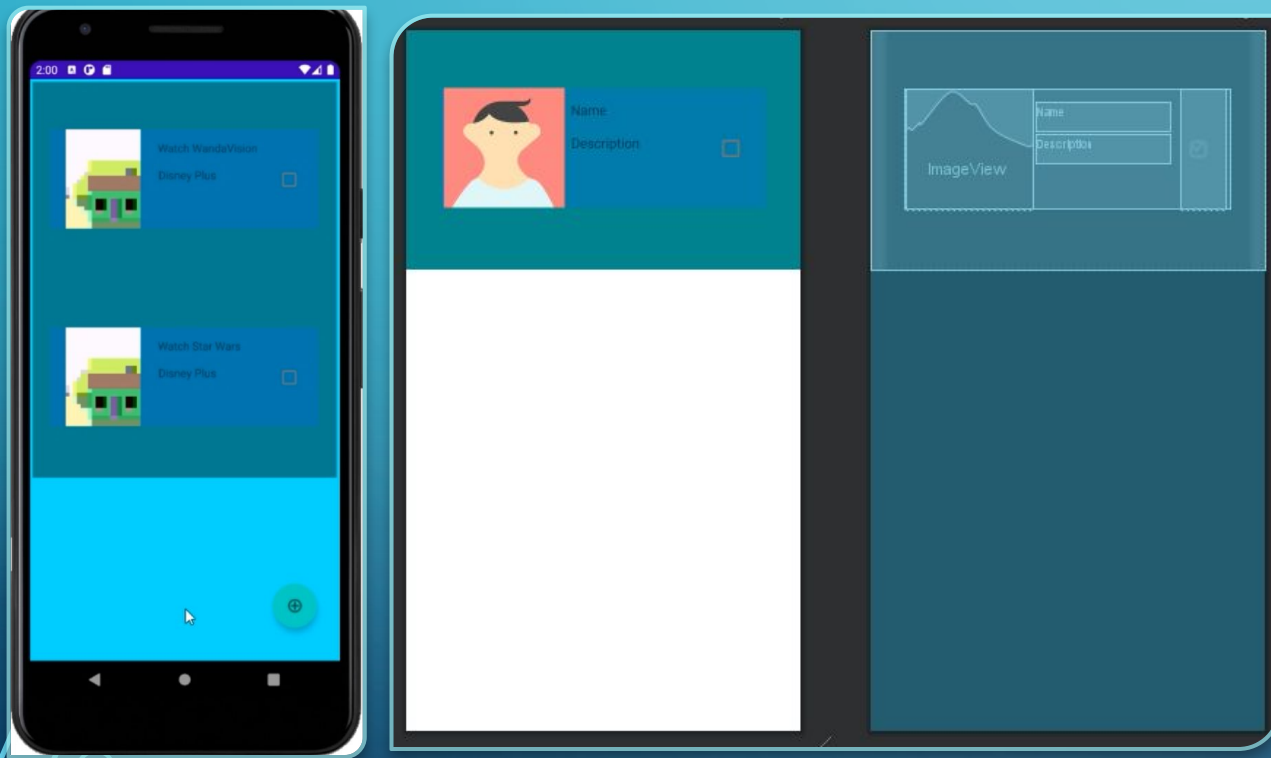
```
ConstraintLayout
└─ ConstraintLayout
    Ab textView2 "Title"
    Ab taskTitleView (Plain Text)
    Ab textView3 "Description"
    Ab taskDescriptionView (Plain Text)
    Ab textView4 "Due Date"
    Ab taskDueDateView "DD/MM/YYYY"
    Ab textView5 "Due Time"
    Ab taskDueTimeView "00:00"
    taskImage
    cameraButton
    saveButton "Save"
```

# PART 3: XML FORM LAYOUT



- There are three XML forms in the app:
- The Main Creation which houses the components which allow the user to enter and save a task.
- The landing page which allows the user to view held tasks and houses the 'create task' button which brings up the Main Creation form.

## PART 3: XML FORM LAYOUT



- The landing page also lists each of the stored activities held in the apps database.
- It contains the main detail of the task, such as the image, title and description.
- It also houses the check function which allows the user to check off the task.
- It also has a swipe feature to delete the task from the database.

## PART 3: XML FORM LAYOUT

```
<!-->Launches the ToDoListActivity first<!-->
<activity android:name=".ToDoListActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".MainActivity">
</activity>
```

- In order to launch the landing page of the app, is to change the Android Manifest XML file to call the `ToDoListActivity` which launches the landing page first and not the task creation form.



# PART 4: DATABASE





# PART 4: DATABASE

```
@Entity
public class Task {

    @PrimaryKey(autoGenerate = true)
    public int uid;

    @ColumnInfo(name = "title")
    public String title;

    @ColumnInfo(name = "description")
    public String description;

    @ColumnInfo(name = "date")
    public Long duedate;

    @ColumnInfo(name = "image")
    public String image;

    @ColumnInfo(name = "done")
    public boolean done;

}
```

```
@Database(entities = {Task.class}, version = 1)
public abstract class TasksDB extends RoomDatabase {

    public abstract TaskDAO taskDAO();

    private static final String DB_NAME = "task_database_name";
    private static TasksDB db;

    public static TasksDB getInstance(Context context){
        if(db == null) db= buildDatabaseInstance(context);
        return db;
    }

    private static TasksDB buildDatabaseInstance(Context context) {
        return Room.databaseBuilder(context, TasksDB.class, DB_NAME).build();
    }
}
```

- A databases was created and is implemented into the code of the app, this is to store the data of each held task.
- This is to allow for each task to be retrieved and deleted if need be by the user.



## PART 4: DATABASE

- The MainActivity.java file houses an onSaveClick method which catches the inputted data and passes it back through the database as an object of Task.

```
//Logs data when Save Button is pressed
public void onSaveClick(View v) {
    Log.d( tag: "ToDoApp", msg: "onSaveClick");

    EditText titleView = findViewById(R.id.taskTitleView);
    EditText descView = findViewById(R.id.taskDescriptionView);

    String title = titleView.getText().toString();
    String description = descView.getText().toString();

    final Task task = new Task();
    task.title = title;
    task.description = description;
    task.image = imageUri.toString();
    task.done = false;

    Executor myExecutor = Executors.newSingleThreadExecutor();
    myExecutor.execute(new Runnable() {
        @Override
        public void run() { db.taskDAO().insert(task); }
    });

    LiveData<List<Task>> tasks = db.taskDAO().getAll();

    tasks.observe( owner: this, new Observer<List<Task>>() {
        @Override
        public void onChanged(List<Task> tasks) {
            for(Task task: tasks){
                Log.d( tag: "ToDoApp", msg: task.title + ":" + task.description);
            }
        }
    });

    this.finish();
}
```

# PART 4: DATABASE

- The landing page of the app, allows the user to see a list of saved tasks which can be either viewed, updated, or deleted from the database entirely.
- This is held together by the databinding of the app. Which allows for data to be fully stored in the app.

```
public class ToDoListActivity extends AppCompatActivity {

    TasksDB db;
    TaskListAdapter taskListAdapter;

    //Creates the task block viewing list
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getSupportActionBar().hide();

        setContentView(R.layout.activity_to_do_list);

        db = Room.databaseBuilder(getApplicationContext(),
            TasksDB.class,
            "tasks_database_name").build();

        RecyclerView recyclerView = findViewById(R.id.taskListRecyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        taskListAdapter = new TaskListAdapter();
        recyclerView.setAdapter(taskListAdapter);

        //Allows user to delete task by swiping the task card to the right
        ItemTouchHelper.SimpleCallback touchHelperCallback = new ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.RIGHT) {
            @Override
            public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder, @NonNull RecyclerView.ViewHolder target) {
                return false;
            }


            @Override
            public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction){
                int swipedPosition = viewHolder.getAdapterPosition();

                taskListAdapter.deleteTask(swipedPosition);
            }
        };

        ItemTouchHelper itemTouchHelper = new ItemTouchHelper(touchHelperCallback);
        itemTouchHelper.attachToRecyclerView(recyclerView);
    }
}
```

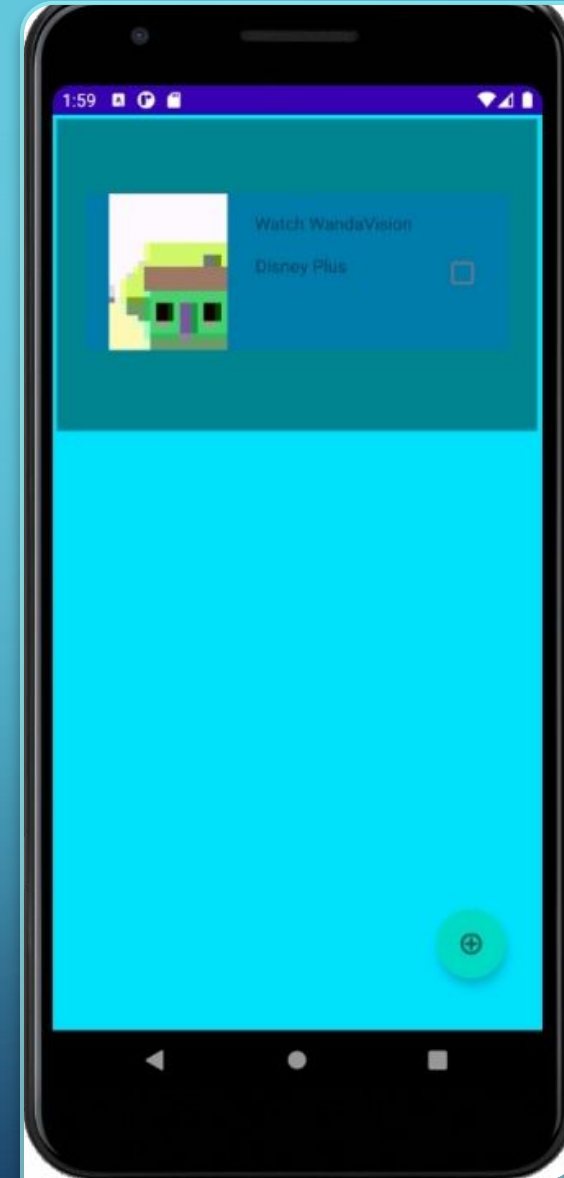


## PART 4: DATABASE

- Because of the two-way databinding, what happens in the main task creation is stored and is added to the landing page to be viewed as a saved task.
  - The databinding allows for the data to be created, deleted, read and updated, it allows for full usability and allows for the app to be closed and reopened with the data held being kept stored and safe.
- 

# FULL VIDEO DEMONSTRATION

- Here is a the demonstration of the app working in its entirety. This demonstration is running on the built in android emulation which is a virtual Google Pixel 3.



# REFERENCES

- Bowers, C. (2021) 'Worksheet 2 - Creating a User Interface', Mobile Application Development. Available at:  
<https://worchesterbb.blackboard.com/> (Accessed: 25th January 2021)