# TODO App Design Specification

Emma George - CST 338 Final Project

The TODO App will allow users to generate a TODO list with editable and sortable tasks. It will have a minSDK of 29 and a targetSDK of 32.

## Phase 1: The Model

### The Task Class

#### Private Member Data

```
private final UUID mId = UUID.randomUUID();
private Date mDeadline;
private boolean mCompleted;
private String mTitle;
private String mDescription;
private int mPriority; // Range 1-5 (inclusive). 1 for high priority.
```

#### Public Methods

- **public Task(String mTitle, Date mDeadline, String mDescription, int mPriority)** - Constructor instantiates mTitle, mDeadline, mDescription, and mPriority with given values. The Constructor also instantiates mCompleted as false and assigns mId a random UUID.
- **Accessors** for private member data.
- **Mutators** for mDeadline, mCompleted, mTitle, mDescription, and mPriority.

### The TaskList Class

#### Private Member Data

```
private static TaskList sTaskList;
private final ArrayList<Task> mTasks;
```

#### Public Methods

- **public static TaskList get(Context context)** - Public accessor for sTaskList. If sTaskList has not been instantiated, this method will instantiate sTaskList using the private constructor.
- **Accessor** for mTasks

- **public Task getTask(UUID id)** - Find a task in mTasks that matches the UUID parameter and return the Task, otherwise return null.
- **public getIncompleteTasks()** - Returns a list of all incomplete tasks from mTasks.
- **public clearCompletedTasks()** - Removes completed tasks from the mTasks list.
- **sortByPriority(),sortByDeadline(),sortByTitle()** - Define 3 inner classes implementing the Comparator interface and use to sort mTasks by Priority, Deadline, and Title.

#### Private Methods

- **private TaskList(Context context)** - Private constructor for TaskList. Assigns a new ArrayList to mTasks.

# Phase 2: The View

**layout/activity_fragment** - Contains a toolbar and a FrameLayout named fragment_container. This layout will be used by SingleFragmentActivity, so should be generic.

## Task View

- **layout/fragment_task** - Displays a task's details and all properties using TextView elements.
- **layout/fragment_edit_task** - Displays a task's details and allows the user to edit a task's properties.
- **menu/fragment_task** - Menu contains an edit button.
- **menu/fragment_edit_task** - Displays toolbar.

## TaskList View

- **layout/fragment_task_list** - Displays a RecyclerView of a TaskList object and a FloatingActionButton for adding a task.
- **layout/list_item_task** - Customize the list item to display a Task's name, deadline, and a checkbox for completion.
- **menu/fragment_task_list** - Menu contains a filter button.

## Onboarding View

- **layout/activity_sign_up** - Displays a Create Account TextView and has EditText elements for Name, Email, Password, and to Confirm Password. A button for Sign Up is at the bottom.
- **layout/activity_login** - Displays the app name in a TextView and provides EditText elements for an Email and Password. Buttons for Login, Forgot Password, and Sign Up.

### Dialogs

- **layout/dialog_date** - Holds a DatePicker element.
- **layout/fragment_date_picker** - Contains a FrameLayout
- **layout/fragment_filter** - Displays a RadioGroup of RadioButton objects to select a sorting method for a list of tasks. Displays a switch to toggle the visibility of completed tasks. Provides a Button to delete completed tasks from the list of tasks.

# Phase 3: The Controller

### SingleFragmentActivity

SingleFragmentActivity creates a single Fragment and extends AppCompatActivity. This will serve as a generic activity to be extended by TaskActivity, TaskListActivity, and EditTaskActivity.

## Task Controller

### TaskActivity

```
public static final String EXTRA_TASK_ID;
```

The TaskActivity class extends SingleFragmentActivity. TaskActivity will hold a TaskFragment and pass to it an extra containing a TaskID.

### TaskFragment

```
  // Model
   private Task mTask;
   // View Elements
   private TextView mTitleTextView;
   private TextView mDateTextView;
   private TextView mDescriptionTextView;
   private TextView mPriorityTextView;
```

The TaskFragment will display all properties of the Task with provided TaskID in the view.

### EditTaskActivity

The EditTaskActivity class extends SingleFragmentActivity and holds an EditTaskFragment. Passes an extra containing TaskID to EditTaskFragment.

### EditTaskFragment

```
  private static final String DIALOG_DATE = "DialogDate";
```

```
    // Request key
    private static final String REQUEST_DATE = "Deadline";
    // Model
    private Task mTask;
    private Task taskCopy;
    private TaskList sTaskList;
    private boolean isNewTask = false;
    // View elements
    private TextView mTitleEditTextView;
    private Button mDateEdit_btn;
    private Button mUpdateTask_btn;
    private Spinner mPriority_spinner;
    private TextView mDescriptionEditTextView;
```

The EditTaskFragment will display all properties of the Task with provided TaskID in the view and allow the user to update these properties. If the Task is new, mUpdateTask_btn will be updated to say Create Task and the EditViews will only display hints. The mUpdateTask_btn will not be enabled until a Title has been entered.

## TaskList Controller

### TaskListActivity

The TaskListActivity extends SingleFragmentActivity and creates a TaskListFragment.

### TaskListFragment

```
   // Request Keys
    private static final String REQUEST_FILTER = "ListFilter";
    private static final String DIALOG_FILTER = "DialogFilter";
    // View Elements
    private RecyclerView mTaskRecyclerView;
    private TaskAdapter mAdapter;
    private FloatingActionButton addTaskBtn;

    private String sortType = "Priority";
    private boolean showCompleted = false;
```

The TaskListFragment displays a RecyclerView of a TaskList.
Clicking the filter button in the options menu will open the FilterFragment dialog. When selections are returned from the FilterFragment dialog, the list display will be updated accordingly. The addTaskBtn will start an EditTaskActivity. Clicking on an item in the list will open a TaskActivity with the given TaskID.

The mTaskRecyclerView will contain TaskHolder objects, a private class extending ViewHolder that must be defined in this activity.

## Onboarding Controller

### LoginActivity

```
private Button loginBtn;
private Button forgotPassBtn;
private Button signUpBtn;
private EditText mEditEmailView;
private EditText mEditPasswordView;
```

LoginActivity extends AppCompatActivity. The loginBtn will be disabled until the email and password EditText fields are not empty. If the loginBtn is enabled and pressed, a TaskListActivity will be started. The signUpBtn will start the SignUpActivity. The forgotPassBtn will show a toast saying password cannot be reset at this time.

### SignUpActivity

```
private Button signUpBtn;
private EditText mEditNameView;
private EditText mEditEmailView;
private EditText mEditPasswordView;
private EditText mEditPasswordConfirmView;
```

SignUpActivity extends AppCompatActivity. The signUpBtn will be disabled until the name, email, and password fields are not empty and the password and confirm password fields match. When the signUpBtn is enabled and pressed, a TaskListActivity will start and a toast will show with a message “Account has been created successfully”.

## Dialog Controller

### DatePickerFragment

```
public static final String ARG_DATE = "date";
private static final String REQUEST_DATE = "Deadline";
private DatePicker mDatePicker;
```

The DatePickerFragment will extend DialogFragment and be used to set a task deadline. When a newInstance is created from an EditTextActivity, the given Date will be bundled and displayed on the DatePicker. The result will be sent back to EditTaskFragment to update the Task’s deadline.

## FilterFragment

```
 // Bundle and Request Keys
  private static final String REQUEST_FILTER = "ListFilter";
  public static final String ARG_SORT = "SortType";
  public static final String ARG_COMPLETED = "ShowCompleted";
```

The FilterFragment will extend DialogFragment and provide options for the user to sort and filter the TaskList. The current sorting method and visibility of completed tasks will be bundled when the newInstance is created. A group of RadioButtons will allow the user to select a sorting method (by priority, title, or deadline), a switch will allow the user to toggle the display of completed tasks, and a button will allow the user to delete completed tasks from the list of tasks. Once selections have been made and the OK option is pressed, the results will be returned to the TaskListActivity.