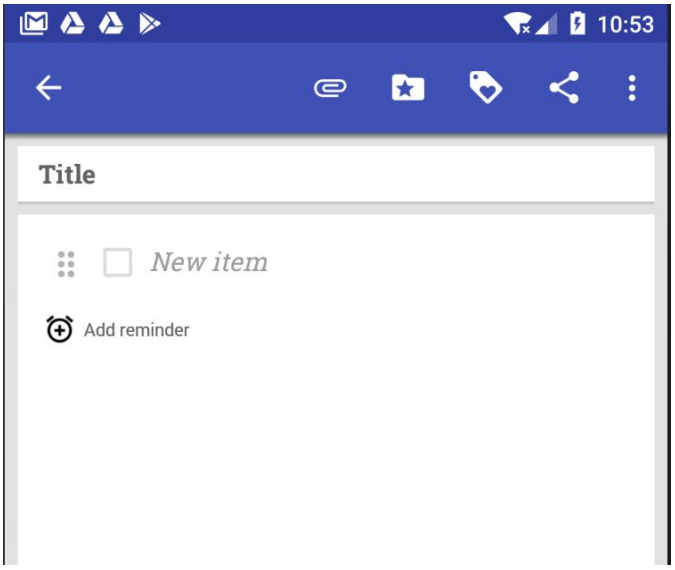
Creating Notes

The first feature we explore is creating notes. It's essential without doubt since the app can't be called Omni-Notes if it can't create notes in the first place! We found the code of creating a note view in *DetailFragment.java*. An empty note view (the one you see when you start a new note) looks like this:



When a new note is about to be created, users need to click "+" on the main user interface to get an empty note view. **initView()** is called during the initialization. In this method, it initializes everything in the note view and **initViewTitle()** and **initViewContent()** are two key methods it calls for editing the new note.

```
@SuppressLint("NewApi")
private void initViews () {

  // Sets onTouchListener to the whole activity to swipe notes
  root.setOnTouchListener(this);

  // Color of tag marker if note is tagged a function is active in preferences
  setTagMarkerColor(noteTmp.getCategory());

  initViewTitle();

  initViewContent();

  initViewLocation();

  initViewAttachments();

  initViewReminder();

  initViewFooter();
}
```

initViewTitle() sets the EditText **title** with the string returned from noteTmp.getTitle(). In the case of creating a note, noteTmp is an empty note. It allows users to simply edit the title of the note.

```
private void initViewTitle () {
  title.setText(noteTmp.getTitle());
  title.gatherLinksForText();
  title.setOnTextLinkClickListener(textLinkClickListener);
  // To avoid dropping here the  dragged checklist items
  title.setOnDragListener((v, event) -> {
//                ((View)event.getLocalState()).setVisibility(View.VISIBLE);
    return true;
  });
  //When editor action is pressed focus is moved to last character in content field
  title.setOnEditorActionListener((v, actionId, event) -> {
    content.requestFocus();
    content.setSelection(content.getText().length());
    return false;
  });
  requestFocus(title);
}
```

initViewContent() sets the EditText **content** with the string returned from noteTmp.getContent(). The same situation applies to it - noteTmp is an empty note. It allows users to edit the content of the note and mark an item as completed by clicking the checkbox.

```
private void initViewContent () {

  content.setText(noteTmp.getContent());
  content.gatherLinksForText();
  content.setOnTextLinkClickListener(textLinkClickListener);
  // Avoids focused line goes under the keyboard
  content.addTextChangedListener( watcher: this);

  // Restore checklist
  toggleChecklistView = content;
  if (noteTmp.isChecklist()) {
    noteTmp.setChecklist(false);
    AlphaManager.setAlpha(toggleChecklistView,  alpha: 0);
    toggleChecklist2();
  }
}
```

At first we didn't know how the app saves a new note. By using the app we knew that a non-empty note would be saved once you press the back button. We searched "save" then found **saveAndExit()** and **saveNote()**. They are also in *DetailFragment.java*.

We found the most relevant usage of **saveAndExit()** is in **onBackPressed()** in *MainActivity.java*. When the DetailFragment is loaded and you press the back button, **saveAndExit()** will be called.

```
// DetailFragment
f = checkFragmentInstance(R.id.fragment_container, DetailFragment.class);
if (f != null) {
  ((DetailFragment) f).goBack = true;
  ((DetailFragment) f).saveAndExit((DetailFragment) f);
  return;
}
```

saveAndExit() saves a non-empty note and shows a message “Note updated” at the top of the screen when users return to the main user interface. It calls the method **saveNote()** for a successful save event. **saveNote()** sets noteTmp with title and content that users edited. To get the title and the content (including the status of every item) from corresponding EditTexts and CheckBoxes, **getNoteTitle()** and **getNoteContent()** get called correspondingly. Then **saveNote()** filter all empty notes. If a note is empty, it shows a message “Can’t save an empty note”. If not, it initializes a **SaveNoteTask** to really save the note.

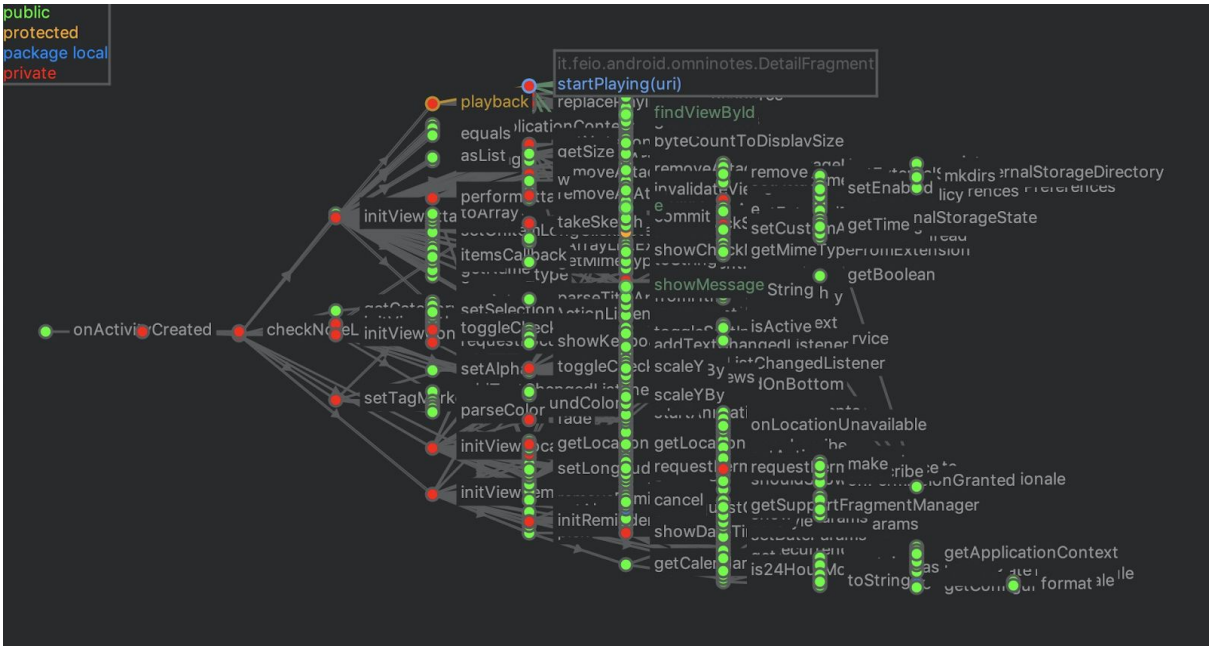
```
public void saveAndExit (OnNoteSaved mOnNoteSaved) {
  if (isAdded()) {
    exitMessage = "Note updated";
    exitCroutonStyle = ONStyle.CONFIRM;
    goBack = true;
    saveNote(mOnNoteSaved);
  }
}

/**
 * Save new notes, modify them or archive
 */
void saveNote (OnNoteSaved mOnNoteSaved) {

  // Changed fields
  noteTmp.setTitle(getNoteTitle());
  noteTmp.setContent(getNoteContent());

  // Check if some text or attachments of any type have been inserted or is an empty note
  if (goBack && TextUtils.isEmpty(noteTmp.getTitle()) && TextUtils.isEmpty(noteTmp.getContent())
      && noteTmp.getAttachmentsList().size() == 0) {
    LogDelegate.d("Empty note not saved");
    exitMessage = "Can't save an empty note";
    exitCroutonStyle = ONStyle.INFO;
    goHome();
    return;
  }
```

SaveNoteTask extends *AsyncTask*. It deals with note-saving tasks in the background. For this feature, only the title part and the content part are highly relevant. It writes the new note to the database in **doInBackGround()** using a *DBHelper*.

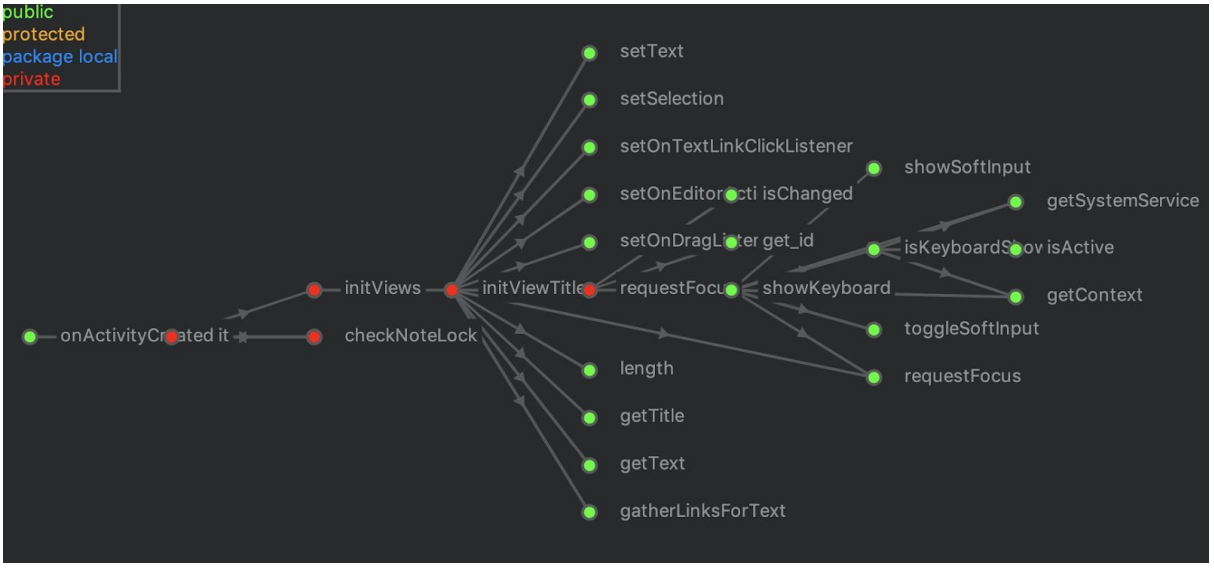
```java
@Override
protected Note doInBackground (Note... params) {
  Note note = params[0];
  purgeRemovedAttachments(note);
  boolean reminderMustBeSet = DateUtils.isFuture(note.getAlarm());
  if (reminderMustBeSet) {
    note.setReminderFired(false);
  }
  note = DbHelper.getInstance().updateNote(note, updateLastModification);
  if (reminderMustBeSet) {
    ReminderHelper.addReminder(context, note);
  }
  return note;
}
```

Call Graphs for This Feature

initView()



initViewTitle()



initViewContent()

saveNote()/saveAndExit()