Meilenstein 4

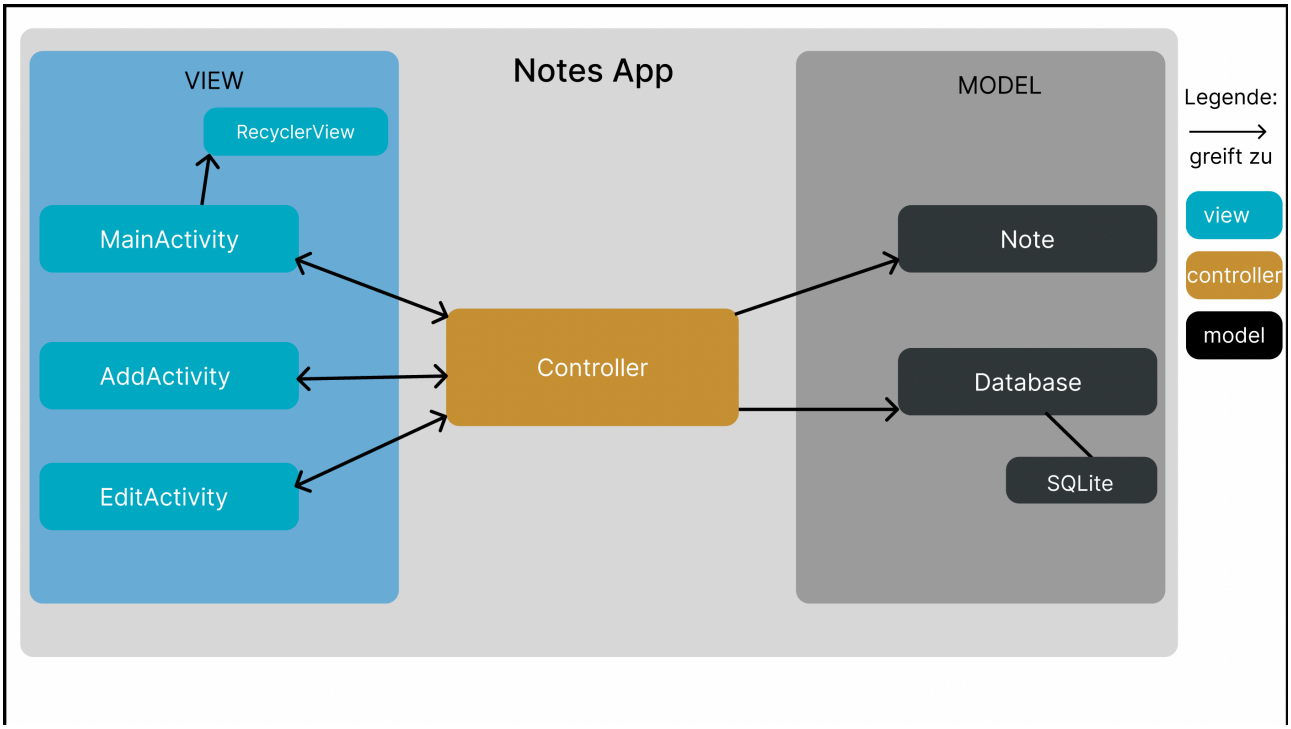
Notes Application

Notes Application ist ein simples, minimalistisches aber dennoch ein sicheres und performantes Notizen App.
Mann kann Notizen schreiben und sie in Datenbank Persistent(Offline) speichern.
Die Notizen werden auf dem Display als Liste angezeigt und man kann die Notizen durchsuchen um ein bestimmte Notiz zu finden und man kann auch sie öffnen, bearbeiten und löschen.

Github Repository:

https://github.com/RomanBehroz/Android-Notes-Application

Struktur und Komponenten Interaktionen



1.Komponente MainActivity (VIEW)

1.1. Zusammenfassung

Ist die Haupt Oberfläche des Apps.
Auf dem Main Activity wird die Notizen in form eine liste(Recycler View) dargestellt.
Oben rechts steht ein Search Button um die Notizen durchzusuchen. Unten rechts

steht Plus(+) Das Button wird eine *neue Activity(AddActivity)* öffnen.
Nutzt RecyclerView Komponente.

1.2. Innere Struktur und Technik

RecyclerView und Adapter wird hier erzeugt und benutzt.
Die addButton(+) verwendet onClickListener um andere Activity zu öffnen
Während Scrolling durch die Liste, geht die Tastatur automatisch wieder zu falls sie offen ist. RecyclerView Elemente haben onClickListener, das Notiz kann geöffnet werden(EditActivity)
Die ActionBar für die Activity wird auch hier individuell geändert.

1.3. Schnittstellen

Verwendet keine Schnittstellen

1.4. Test

das Activity ist Simple aufgebaut und daher auch einfach zu testen

Test Klasse: MainActivityTest

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Add Button soll auf display zu sehen sein	addButtonIsDisplayed()	Espresso	Ja
Add Button öffnet Add Activity View	addButtonOpensAddActivity()	Espresso	Ja
RecyclerView ist zu sehen	recyclerViewIsDisplayed()	Espresso	Ja
Search Funktion soll zu sehen sein	searchIsDisplayed()	Espresso	Ja
App Title ist korrekt	notesTitleIsDisplayedCorrectly()	Espresso	Ja

2.Komponente AddActivity (VIEW)

2.1. Zusammenfassung

Auf die AddActivity kann man neues Notiz schreiben und speichern.

2.2. Innere Struktur und Technik

Verwendet Form: Title(String), Text/Note(String), Save(Button)
Save Button hat ein onClickListener um das Notiz zu speichern
Keine besondere Technologien verwendet

2.3. Schnittstellen

Verwendet keine Schnittstellen

2.4. Test
das Activity ist Simple aufgebaut und daher auch einfach zu testen

Test Klasse: AddActivityTest

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Save Button soll auf display zu sehen sein	saveButtonIsDisplayed()	Espresso	Ja
Title Text Field, muss zu sehen sein auf Bildschirm	titleTextFieldIsDisplayed( )	Espresso	Ja
Note Text Field, muss zu sehen sein auf Bildschirm	noteTextFieldIsDisplayed ()	Espresso	Ja
Notiz schreiben und speichern	writeNoteAndSaveIt()	Espresso	Ja

3.Komponente EditActivity (VIEW)

3.1. Zusammenfassung
Auf die EditActivity kann man ein bereits gespeichertes Notiz bearbeiten und speichern oder das Notiz komplette löschen.
Verwendet individuelle ActionBar mit LöschenIcon oben rechts

3.2. Innere Struktur und Technik
Verwendet Form: Title(String), Text/Note(String), Save(Button)
Save Button hat ein onClickListener um das Notiz zu speichern
Keine besondere Technologien verwendet

3.3. Schnittstellen
Verwendet keine Schnittstellen

3.4. Test
das Activity ist Simple aufgebaut und daher auch einfach zu testen

Test Klasse: AddActivityTest

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Save Button soll auf display zu sehen sein	saveButtonIsDisplayed()	Espresso	Ja
Title Text Field, muss zu sehen sein auf Bildschirm	titleTextFieldIsDisplayed( )	Espresso	Ja

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Note Text Field, muss zu sehen sein auf Bildschirm	noteTextFieldIsDisplayed ()	Espresso	Ja

4.Komponente NotesController (CONTROLLER)

4.1. Zusammenfassung
Ermöglicht die Kommunikation zwischen Model und Views.
Enthält die Logik für das App. Model(Notes) und Datenbank Objekte werden durch Controller erzeugt.

4.2. Innere Struktur und Technik
Ganz normale Java Methoden wie Add, Remove, Update, Delete bei einer Liste.

4.3. Schnittstellen
<<NotesControllerInterface>>

```
public interface NotesControllerInterface {

    /**
     * Adds a Note to the Database
     * @param title title of the Note
     * @param text text of the Note
     */
    void addNote(String title, String text);
    /**
     * Updates a Note in the Database. Id remains the same
     * @param id id of the Note
     * @param title title of the Note
     * @param text text of the Note
     */
    void updateNote(int id, String title, String text);
    /**
     * Deletes a Note from the Database
     * @param id id of the Note that gets deleted
     */
    void deleteNote(int id);
    /**
```

```
     * Deletes all the NOtes from the Database
     */
    void deleteAllNotes();

    /**
     * Gets the Data from the Database and Saves it in the Notes list
     * @return returns the number of objects/rows in  the Database
     */
    int syncNotes() throws ParseException;

    /**
     *Gets the List of the Notes
     * @return Notes list
     */
    List<Note> getNotes();
}
```

4.4. Test

Unit Tests(JUnit)

Test Klasse: NotesControllerTest

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Anzahl von Notizen	getNotesCount()	JUnit	Ja
Notizen von Datenbank holen und in List<Note) speichern	syncNotes()	JUnit	Ja
Notiz bearbeiten/ aktuallisieren	updateNote()	JUnit	Nein
Notiz hinfügen	addNote()	JUnit	Ja
Alle Notizen löschen	deleteAllNotes()	JUnit	Ja

5.Komponente Note (MODEL)

5.1. Zusammenfassung

Das Model für das Notes Application
Eine Java Klasse Note, repräsentiert ein Notiz. Notiz hat Id, Title, Text und Datum

5.2. Innere Struktur und Technik

ganz normaler Java Klasse mit primitive Datentypen und dazu die Setters und Getters

5.3. Schnittstellen

<<NoteInterface>>

```
    /**
     * gets note id
     * @return id
     */
     int getId();

    /**
     * sets note id
     * @param id id
     */
     void setId(int id);

    /**
     * gets note title
     * @return the title
     */
     String getTitle();

    /**
     * sets note title
     * @param title the title
     */
     void setTitle(String title);

    /**
     * gets the note text/content
     * @return
     */
     String getText();

    /**
     * sets note text/content
     * @param text
     */
     void setText(String text);

    /**
     * gets the date on which the note was created
     * @return date of the note
     */
     Date getDate();

    /**
     * sets the date of creation for the note
     * @param date
     */
     void setDate(Date date);
}
```

5.4. Test

Unit Test (JUnit)

Test Klasse: NoteTest

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Notiz Id auf korrektheit	getId()	Juni	Ja
Notiz id set Korrektheit	setId()	Juni	Ja
Title, Text, Datum Alle getters und setters	setTitle(), getTitle(), setText(). getText), setDatum(), getDatum()	Juni	Ja

6.Komponente DatabaseHelper(Database) (MODEL)

6.1. Zusammenfassung
Verwendung von SQLite Datenbank. Die Klasse macht möglich SQLite zu benutzen, die Daten auf Datenbank zu speichern. Das Datenbank + Tabelle Erzeugung.

6.2. Innere Struktur und Technik
SQLite Daten Offline(Persistent) auf dem Handy zu speichern.
Ein Android Integrierte Datenbank

6.3. Schnittstellen
Erbt von SQLiteOpenHelper

6.4. Test
Unit Test (JUnit)

Test Klasse: DatabaseHelperTest

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Daten Speichern	addData()	JUnit	Ja
Alle Daten Löschen	testDeleteAllData()	JUnit	Ja
Nur ein Element löschen	testDeleteOnlyOne	JUnit	Ja
Daten aktualisieren	updateData()	JUnit	Nicht implementiert

9. Integrationstest (RecyclerEspressoTest)
Um die Komponente und verschiedene Schichten zusammen zu testen und zu gucken dass die Interaktion zwischen Schichten funktionieren.
Es wird paar fälle auf einem Virtual Device simuliert.

Was wird getestet	Name des Tests	Test methode	Erfolg ja/nein
Add Button clicken, neues Notiz(title, text) schreiben, auf Save Button clicken, Notiz speichern und MainActivity öffnen	writeNewNoteAndSaveIt( )	Espresso	Ja