TODOPATH Mobile Computing 2016 Projektreport

Elisha Witte
11107636

Vica Katherine
11099454

ABSTRACT

TodoPath ist eine Android Applikation zum Erstellen und Verwalten von Todos. In dieser Dokumentation gehen wir auf die Funktionalität der App, sowie die spezifische Implementierung ein. Dabei werden insbesondere die Architektur, das Datenmodell und die einzelnen Komponenten betrachtet.

Keywords

Android Development, Java, Mobile Computing, App, Todo

EINFÜHRUNG

In der heutigen Zeit hat das Smartphone eine zentrale Rolle im Alltag der Menschen eingenommen. Es bietet alle möglichen Funktionen, für die früher separate Geräte oder Ressourcen notwendig waren, wie fotografieren, filmen, Musik hören, navigieren, im Internet surfen, etc. Eine dieser Funktionen für die früher vielleicht noch Notizzettel oder ähnliches nötig waren ist das Verwalten von Aufgaben. Todo-Listen sind eine einfache Möglichkeit die eigene Produktivität zu steigern und nie den Überblick über anstehende Aufgaben und Ziele zu verlieren.

Ziel des Mobile Computing Kurses war es eine native Android Applikation zu entwickeln, welche es dem Benutzer ermöglicht Todos festzuhalten. Hierbei haben wir uns von Anfang an auf zwei Alleinstellungsmerkmale fokussiert, welche uns von der großen Masse an anderen Todo-Applikationen abgrenzen sollten: die Möglichkeit Todos zu kategorisieren und gruppieren (über Sub-Listen), sowie das Zuweisen von Prioritäten. In dieser Dokumentation werden wir zunächst auf die Funktionalität unserer Applikation eingehen, und anschließend einen Überblick über die konkrete Realisierung und Architektur geben. Dabei werden wir außerdem auf die Schwierigkeiten und Probleme eingehen, auf die wir im Entwicklungsprozess gestoßen sind.

FUNKTIONALITÄT DER APPLIKATION

Benutzer können in *TodoPath* Listen und Todos anlegen.

Listen

Listen sind nichts weiter als Gruppen von Todos welche unter einem allgemeinen Begriff zusammengefasst werden. So wird die Kategorisierung und Abgrenzung von Todos ermöglicht. Beispielsweise könnte der Benutzer so eine Liste von Todos die mit der Organisation eines Events zusammenhängen erstellen, und gleichzeitig separat eine einfache Einkaufsliste verwalten. Durch die Abgrenzung durch Listen wird sichergestellt, dass Todos aus unterschiedlichen Kontexten nicht miteinander vermischt werden, und der Benutzer einen guten und organisierten Überblick über seine Aufgaben behält.

Todos

Todos enthalten in *TodoPath* nicht nur den eigentlichen Aufgabentext, sondern können außerdem auch noch zusätzliche Informationen enthalten. Benutzer können ihre Todos optional mit einem Kontakt aus dem Kontaktbuch ihres Smartphones verknüpfen.

Dies könnte nützlich sein, wenn sie beispielsweise ein Meeting mit einer bestimmten Person haben. Somit könnten sie, sobald sie an das Todo erinnert werden, direkt aus der Applikation heraus den Kontakt aufrufen und zusätzliche Informationen wie Telefonnummer oder Adresse abfragen, ohne selber den Kontakt heraussuchen zu müssen.

Benutzer können ebenfalls eine Deadline hinzufügen, zu der das Todo fällig wird. Sobald ein Datum festgelegt wurde, weiß die Applikation an welchem Tag sie den Benutzer an das Todo mithilfe einer Notifikation erinnern soll.

Ein weiteres wichtiges Feature ist die Möglichkeit den Todos Prioritäten zuzuweisen (niedrig / mittel / hoch). Dies hat zweierlei Zwecke: zum einen ermöglicht es eine bessere Übersicht für den Benutzer, da Todos nach Priorität sortiert angezeigt werden, und der Benutzer nicht erst durch eine eventuell längere Liste scrollen muss um die wichtigsten Einträge zu finden. Zum anderen kann der Benutzer einstellen, nur Notifications für wichtige (i.e. hohe Priorität) Todos zu erhalten.

Todos können selbstverständlich als erledigt markiert werden und erscheinen dann in der Liste dementsprechend durchgestrichen.

VERGLEICH MIT ANDEREN APPLIKATIONEN

Nachdem wir nun etwas genauer auf unsere Features und Funktionalitäten eingegangen sind, möchten wir noch kurz einen Vergleich mit anderen Applikationen aufstellen. Todo-Applikationen gibt es reichlich, und viele haben interessante Features welche wir aufgrund von Zeit- oder Aufwandsgründen nicht implementieren konnten, welche jedoch im Kontext des Entwicklungsprozesses eine Besprechung wert sind.

Applikationen aus dem PlayStore

Bevor wir überhaupt mit der Konzeptualisierung unserer Applikationen begonnen haben, haben wir uns angeschaut welche bestehende und erfolgreiche Applikationen es bereits im PlayStore gibt, und diese etwas eingehender analysiert. Dadurch konnten wir eine gute Basis mit den wichtigsten Features für unser Konzept herleiten, welches dann durch unsere Alleinstellungsmerkmale erweitert wurde.

Konkret haben wir uns die beiden Applikation *Wunderlist* und *Any.do* angeschaut. Im Folgenden gehen wir kurz auf die wichtigsten Features ein, und wie wir diese eventuell in unser Konzept übernommen haben.

WUNDERLIST

Ein wesentliches Feature bei Wunderlist ist die Möglichkeit verschiedene Listen zu erstellen, welches wir ähnlich in unserer Applikation umgesetzt haben. Auch hier können Todos mit einem Fälligkeitsdatum verknüpft werden, sodass der Benutzer rechtzeitigt benachrichtigt werden kann. Es gibt außerdem ein grundlegendes Prioritätssystem, in dem bestimmte Todos ‚*favorisiert*' werden können, und somit an den Anfang der Liste geschoben werden. Dies haben wir als Inspiration für unser etwas erweitertes Konzept mit mehreren Prioritätsstufen genommen.

ANY.DO

Ähnlich wie bei Wunderlist können auch hier verschiedene Listen erstellt werden, und Todos mit einem Datum verknüpft werden. *Any.do* geht allerdings noch einen Schritt weiter: man kann beliebige Dateien als Anhang an ein Todo anheften, sowie zusätzlich noch Unteraufgaben hinzufügen. Während der Nutzen der Anhänge nach unserer Einschränkung nicht allzu hoch ist, fanden wir die Idee mit den Unteraufgaben sehr interessant. Dies wäre vielleicht etwas was man zukünftig noch implementieren könnte.

Eine wichtige Gemeinsamkeit die beide Applikationen haben, ist die Möglichkeit die Todos mit der Cloud zu synchronisieren, und so auf mehreren Endgeräten verfügbar zu machen. Dies ist in der heutigen Zeit, wo alles vernetzt ist und Benutzer ihre Daten auf allen möglichen Geräten verwalten sicherlich ein wichtiges Feature. Nichtsdestotrotz haben wir uns bei unserer Applikation der Einfachheit halber dazu entschlossen einen rein lokalen Speicher zu verwenden.

Würden wir unsere Daten stattdessen auf einen Server verlegen wollen, müssten wir abgesehen von grundlegenden Sicherheitsmerkmalen wie Benutzerauthentifizierung, etc. auch noch verschiedene Aspekte der Mobility berücksichtigen. So müssten beispielsweise Änderungen des Benutzers zwischengespeichert werden bis eine günstige Internetverbindung verfügbar ist (Batching) bzw. bestimmte Daten auf dem Gerät temporär gespeichert werden damit nicht jedes Mal eine neue Anfrage an den Server geschickt werden muss (Caching).

In Rahmen dieser Veranstaltung wäre dies zu aufwändig geworden, auch wenn es sicherlich ein Feature ist welches man für die eventuelle spätere Veröffentlichung der Applikation beachten sollte.

Im Zusammenhang mit der Cloud-Synchronisation fällt ebenfalls auf, dass beide Vergleichsapplikation das Teilen von Listen bzw. Todos mit anderen Benutzern ermöglichen.

Auch dies ist ein interessantes Feature, wofür allerdings wieder eine gewisse serverseitige Implementierung nötig wäre.

Applikationen von Kommilitonen

Abschließend können wir noch kurz auf Funktionalitäten von Applikationen von anderen Gruppen eingehen. Auch wenn die Vorgaben die Gleichen sind, und alle Applikationen in der grundlegenden Struktur sehr ähnlich sind, gibt es doch ein paar Unterschiede.

So hat eine Gruppe beispielsweise ähnlich wie bei *Wunderlist* die Möglichkeit implementiert abgeschlossene Todos komplett auszublenden, während sie bei uns lediglich durchgestrichen werden.

Eine andere Gruppe hat (zumindest im Ansatz) Filter und Label implementiert nach denen die Ansicht sortiert werden kann.

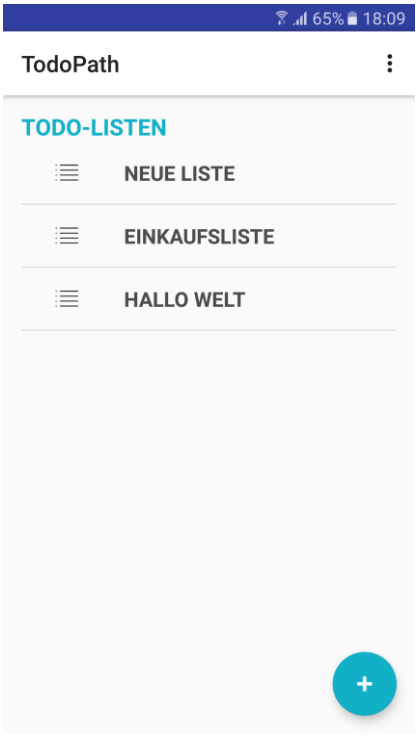


Figure 1. Startseite der Applikation mit Übersicht über alle vom Benutzer erstellten Listen.

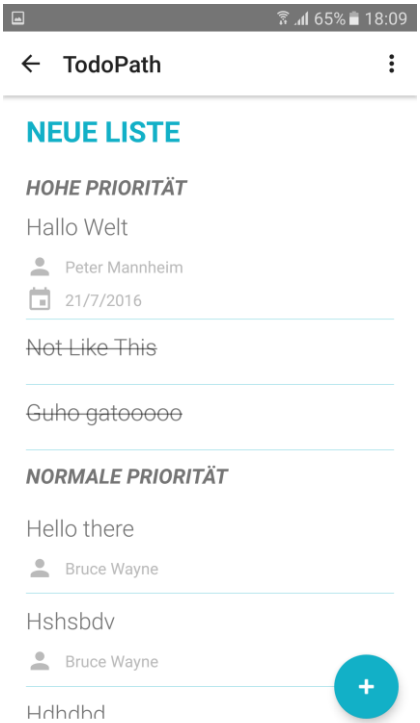


Figure 2. Übersicht mit allen zu einer Liste gehörigen Todos, samt zusätzlichen Informationen wie verknüpfte Kontakte oder Fälligkeitsdatum.

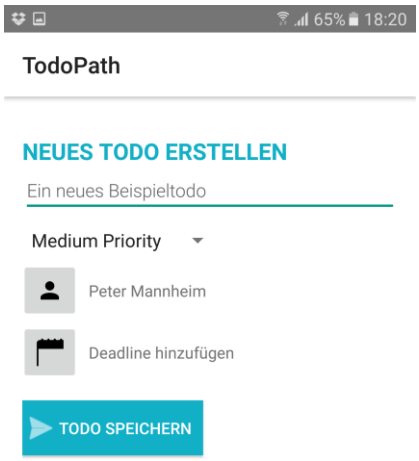


Figure 3. Erstellen eines neues Todos, oder Bearbeiten eines existierendes Todos. Hier können Kontakt und Deadline zu einem Todo hinzugefügt werden.

MODELLSTRUKTUR

Um unser Konzept umzusetzen sind zwei Modelle notwendig: Liste und Todo. Eine Liste wird hierbei durch das ***List*** Modell repräsentiert, und enthält im Wesentlichen nur den Titel sowie das Erstellungsdatum.

Ein ***Todo*** enthält alle relevanten Informationen die zu einer bestimmten Aufgabe gespeichert werden müssen: Name (bzw. Aufgabentext), Status (erledigt oder nicht), Priorität (niedrig/mittel/hoch), Kontaktdaten und Datum.

Ein Todo gehört immer nur zu einer Liste, eine Liste wiederum enthält keine, ein oder mehrere Todos. Es handelt sich also um eine 1:cn Beziehung.

DATENSEMANTIK

Alle Modelldaten werden in einer SQLite Datenbank abgespeichert. Hierfür sind also zwei Tabellen nötig: List und Todo, mit einem *List_ID* Foreign Key für jedes Todo.

Zur Abfrage und Manipulation von Datenbankeinträgen benutzen wir einen `SQLiteOpenHelper`, welcher abgesehen von den Standardmethoden wie `onCreate` (zur Erstellung der Datenbanktabellen), `onUpgrade()` oder `onConfigure()` auch noch diverse Methoden zum Erstellen, Bearbeiten und Abfragen von Listen und Todos implementiert. So kann beispielsweise mit `createList(List list)` ein neues List-Objekt in die Datenbank eingefügt werden, und später wieder über `getList(long list_id)` oder `getAllLists()` abgerufen werden.

Da wir in der Veranstaltung zu Beginn, noch bevor wir mit dem eigentlichen Datenmodell begonnen hatten, mit `ArrayList` gearbeitet haben um Einträge in einem `ListView` darzustellen, haben wir dies anschließend beibehalten, und Methoden wie `getAllLists()` oder `getAllTodos()` liefern als Rückgabewert eine `ArrayList`.

Eine vielleicht etwas besser Alternative wäre es gewesen stattdessen direkt mit `Cursor` zu arbeiten, da Android hier mit dem `CursorAdapter` eine leichte Möglichkeit bietet Einträge aus der Datenbank direkt in eine Liste einzufügen. Stattdessen benutzen wir momentan einen erweiterten `BaseAdapter` welcher mit einer `ArrayList` arbeitet.

Eine andere Möglichkeit wäre es gewesen direkt einen kompletten `ContentProvider` zu schreiben, welcher kontrolliert Zugriff auf bestimmte Daten gewährt, und diese auch anderen Applikationen zur Verfügung stellen kann. Da dies aber für unsere Applikation nicht unbedingt sinnvoll war, haben wir uns dagegen entschieden.

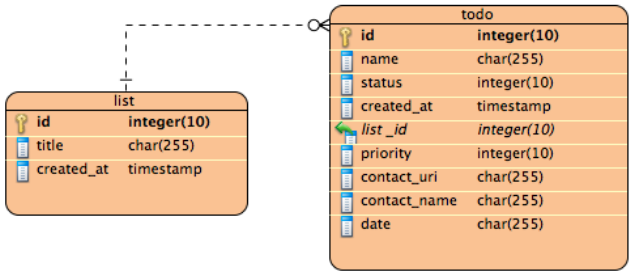
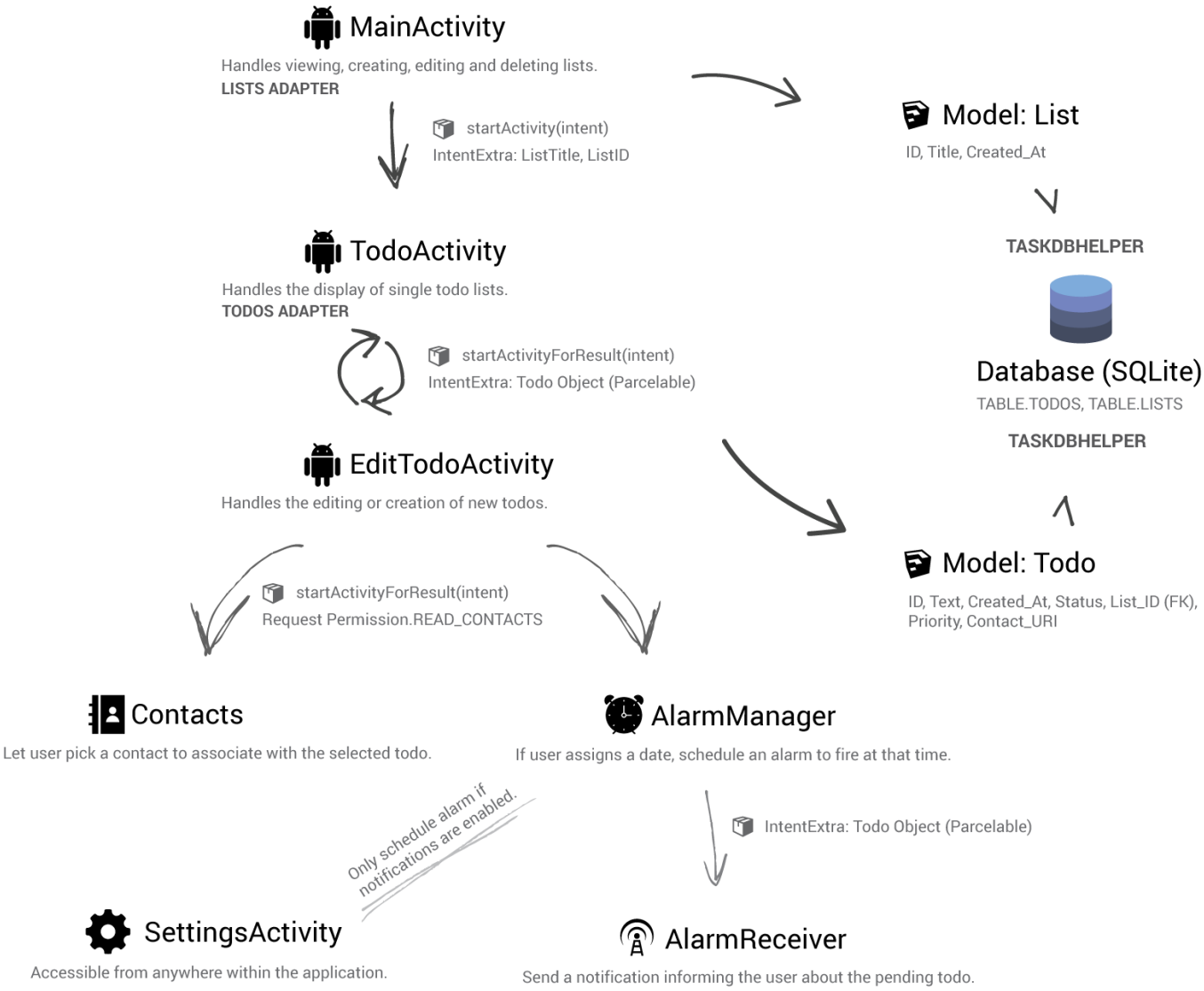


Figure 4. Grafische Darstellung der Datenbanktabellen in unserer SQLite Datenbank. *List_id* ist Foreign Key der eine bestimmte Liste referenziert.

APPLIKATIONSARCHITEKTUR

Bevor wir näher auf die einzelnen Komponenten der Applikation eingehen, folgt zunächst mal ein grober Überblick über die Architektur. Es gibt im Wesentlichen drei `Activities`, also drei Screens, die der Benutzer aufrufen kann. In der `MainActivity` werden die Listen dargestellt, in der `TodoActivity` die Todos die zu einer Liste gehören, und in der `EditTodoActivity` werden neue Todos erstellt oder bestehende bearbeitet. Von hier aus werden noch zwei weitere Komponenten aufgerufen: der `Contacts Provider` um Informationen über Kontakte des Benutzers abzurufen (falls der Benutzer ein Todo mit einem seiner Kontakte verknüpfen möchte) sowie der `AlarmManager` welcher ein neues Event registriert was dann vom `AlarmReceiver` abgefangen wird und eine Notification ausgibt (falls der Benutzer für das Todo ein bestimmtes Fälligkeitsdatum festlegen möchte).

Zusätzlich gibt es noch die `SettingsActivity`, wo der Benutzer Einstellungen der Applikation anpassen kann.



MainActivity

Die `MainActivity` dient als Einstiegspunkt und wird ausgeführt, wenn der Benutzer die Applikation startet. Der View besteht im Wesentlichen aus einer Überschrift, einem `ListView` welches alle vom Benutzer erstellten Listen anzeigt, und ein Platzhaltertext falls der Benutzer noch keine Listen angelegt hat.

Alle Listen liegen in einer `ArrayList` welche über einen einfachen `ListsAdapter` mit dem `ListView` synchronisiert wird. Für das `ListView` wird ein `ContextMenu` angelegt, sodass der Benutzer bei gedrückt halten auf einen Eintrag die zwei Optionen *Bearbeiten* und *Löschen* hat. Möchte der Benutzer eine neue Liste anlegen oder eine bestehende bearbeiten, wird ein `AlertDialog` gestartet, in dem der Name eingegeben werden kann. Mit einem `onItemClickListener` wird die `TodoActivity` gestartet sobald der Benutzer auf einen Listeneintrag klickt. Hierbei wird die ID der Liste als Extra mit dem `Intent` überreicht.

In der `MainActivity` war die einzige größere Schwierigkeit die wir hatten das dynamische Aktualisieren der Ansicht, wenn der Benutzer Listen anlegt, bearbeitet oder löscht. Hierzu haben wir eine kleine `updateUI()` Methode geschrieben, welche den `ListsAdapter` über Änderungen informiert und somit die Ansicht aktualisiert. Diese Methode wird dann an allen Stellen des Programms aufgerufen, an denen Änderungen stattfinden.

In der Posterpräsentation war das Feedback zu dieser Ansicht größtenteils positiv. Von den Gästen haben wir

den Tipp erhalten, dass wir ein anderes Icon für das Erstellen von neuen Listen verwenden sollten damit das Design der Applikation etwas konsistenter ist (die Buttons für das Erstellen von Listen und Todos sollten gleich bzw. sehr ähnlich sein). Dies haben wir anschließend auch umgesetzt.

Von Lyubomir Ganev erhielten wir außerdem den Hinweis, dass das `ContextMenu` zum Bearbeiten bzw. Löschen von Listen/Todos nicht unbedingt intuitiv bzw. ersichtlich für den Benutzer ist. Er schlug vor, dass wir stattdessen zum Löschen eine Swipe-Motion nach rechts implementieren könnten, mit der Einträge aus der Liste „rausgewischt" und somit entfernt werden. Zum Bearbeiten wäre eventuell ein eigener Icon-Button hilfreich. Fabian Bützow wiederum war der Meinung, dass das `ContextMenu` kein Problem darstelle, und in genug Applikationen vorkommt, sodass der Benutzer inzwischen an ein solches Designelement gewöhnt ist.

Aufgrund dieser etwas geteilten Meinung und dem höheren Aufwand der mit der Umsetzung der genannten Vorschläge verbunden ist, haben wir uns dazu entschlossen unsere jetzige Version beizubehalten. Eventuell könnte man in einer späteren Live-Version der App einen Tutorial-Screen implementieren welcher den Benutzer auf dieses Feature hinweist.

TodoActivity

Die `TodoActivity` wird gestartet, sobald der Benutzer einen Listen-Eintrag öffnet. Hier werden alle Todos die zu der jeweiligen Liste gehören angezeigt.

Ähnlich wie bei der `MainActivity` gibt es auch hier eine Überschrift und ein `ListView`. Da eins unserer zentralen Features das Sortieren von Todos nach Priorität ist, war hier allerdings eine etwas komplexere Implementierung des `ListView` nötig. Nach eingehender Recherche und Vergleichen verschiedener Libraries haben wir uns dazu entschlossen `StickyListHeaders` [1] zu verwenden, eine Android Library, mit der Einträge in einem `ListView` einem bestimmten Header (i.e. Kategorie) zugeordnet werden können. Die Header sind hierbei außerdem „sticky", der Benutzer sieht somit also jederzeit in welcher Kategorie er sich momentan befindet.

Beim Start der `TodoActivity` wird zuerst der empfangene `Intent` ausgelesen und mithilfe der List ID die nötigen Datenbankabfragen gestartet. Mit einem `onItemClickListener` wird ein Todo als erledigt markiert und durchgestrichen sobald der Benutzer es anklickt. Auch hier gibt es ähnlich wie in der `MainActivity` ein `ContextMenu` mit den zwei Optionen *Bearbeiten* und *Löschen*.

In der `TodoActivity` hatten wir in einer der ersten Iterationen der Applikation das Erstellen bzw. Bearbeiten von Todos ebenfalls mit einem `AlertDialog` implementiert. Da wir anschließend dem Todo aber immer mehr Informationen und Features (wie z.B. Priorität, Kontaktdaten und Datum) hinzugefügt haben, wurde schnell klar, dass ein Dialogfenster nicht genug Platz bietet, bzw. das Interface unübersichtlich macht. Aus diesem Grund haben wir für das Erstellen und Bearbeiten von Todos eine neue Activity angelegt, welche über `startActivityForResult()` mit den beiden Fällen *EDIT_TODO* oder *CREATE_TODO* ausgeführt wird.

Da wir für diese `EditTodoActivity` eigentlich das gesamte Todo-Objekt benötigen, und den Intents ja normalerweise nur grundlegende Datentypen wie String oder Integer direkt als Extras mitgegeben werden können, mussten wir zu diesem Zeitpunkt unser Todo-Modell noch etwas erweitern und das `Parcelable` Interface implementieren.

Ein `Parcel` in Android ist eine Art Container, welcher serialisierte Daten enthält. Mithilfe des `Parcelable` Interface können ganze Objekte in `Parcels` geschrieben werden, und somit dann einem Bundle hinzugefügt werden welches als Extra dem `Intent` angehängt wird.

Zum Abfangen des Ergebnisses der `EditTodoActivity` mussten wir noch die `onActivityResult` Methode implementieren, welche das geänderte oder neu erstellte Todo Objekt empfängt, in die Datenbank einträgt und die `ArrayList` aktualisiert.

Ähnlich wie bei der `MainActivity` haben wir auch hier eine `updateUI()` Methode welche die Ansicht aktualisiert und die Todos nach Priorität sortiert.

Um die Todos korrekt in der `StickyListHeadersListView` darzustellen, haben wir einen eigenen `TodosAdapter` implementiert, welcher den `BaseAdapter` erweitert. Die wichtigste Methode ist hier `getView`, wo die Eigenschaften des Todos den korrekten View-Elementen zugewiesen werden. Außerdem wird geprüft was der Status des Todos ist, und dementsprechend der Text normal oder durchgestrichen formatiert. Falls das Todo Kontakt oder Datum hat, wird diese Information ebenfalls

dargestellt. Mit `getHeaderView` und `getHeaderID` werden die Todos den Prioriätskategorien zugewiesen.

Hier trafen wir auf die größte Schwierigkeit im Projekt, und ein Problem was wir bis heute nicht lösen konnten. Sobald einem Todo ein Kontakt oder Datum hinzufügt wird, erscheint diese Information nicht nur bei dem eigentlichen Todo, sondern auch bei (zufälligen?) bereits angelegten Todos, sowie jedem ab dem Zeitpunkt neu erstellten Todo. Da dieses Problem bereits auftrat als wir nur die Kontaktverküpfung implementiert hatten, vermuteten wir zuerst ein Problem mit dem Cursor der über die Kontaktresultate iteriert, oder ein Fehler bei der Übergabe des neuen Todos an die `TodoActivity`. Da das Problem allerdings auch beim Datum auftritt (welches lediglich über einen `DatePicker` Dialog festgelegt wird) kann man zumindest ersteres ausschließen. Alle anderen Eigenschaften wie Text oder Priorität sind nicht betroffen.

Nach etwas Debuggen und diversen Log-Ausgaben ist klargeworden, dass die Daten zumindest korrekt in der Datenbank abgespeichert werden, und auch in den Todo-Objekten welche der `TodosAdapter` verarbeitet noch die korrekten Eigenschaften haben. Der Fehler bleibt auch bestehen, wenn man die Applikation neustartet. Da die Daten an sich korrekt sind, muss der Fehler also irgendwo in der Darstellung liegen, vielleicht also im `TodosAdapter`. Wir konnten den Bug allerdings bisher nicht weiter eingrenzen. Auf einem anderen Testgerät als dem Samsung Galaxy S6 trat der Fehler nicht auf.

EditTodoActivity

In der `EditTodoActivity` können neue Todos angelegt werden und bestehende bearbeitet werden. Dafür gibt es ein `EditText` wo der Aufgabentext eingegeben werden kann, ein `Spinner` über den die Priorität ausgewählt wird, sowie zwei Buttons die das Hinzufügen von Kontakt oder Fälligkeitsdatum ermöglichen.

Zunächst wird das empfangene `Intent` ausgewertet. Falls ein bestehendes Todo als Extra empfangen wurde, werden die Eigenschaften des Todos in die entsprechenden Felder eingetragen.

Die zwei wichtigsten Funktionen dieser Activity sind das Hinzufügen eines Kontakts und eines Datums. Für das Verknüpfen mit einem Kontakt wird zunächst überprüft ob die nötige Permission (READ_CONTACTS) vorhanden ist. Wenn nicht, wird (zumindest in den neueren Android Versionen) ein Dialog gestartet der den Benutzer bittet die Permission auszuteilen. In `onRequestPermissionResult` wird das Resultat dieser Abfrage ausgewertet. Hat der Benutzer die Permission bestätigt, wird das Kontaktbuch über `startActivityForResult` geöffnet. Andernfalls wird eine Fehlermeldung ausgegeben.

Nachdem der Benutzer einen Kontakt ausgewählt hat, wird dies wieder über `onActivityResult` abgefangen. Als Resultat erhält man die `ContactURI` des ausgewählten Kontaktes. Da wir zusätzlich aber noch den vollen Namen des Kontaktes anzeigen möchten, wird über einen Cursor eine Auswahl aus dem `ContactsContract` gestartet und der *DISPLAY*_NAME abgefragt. Dieser wird anschließend zusammen mit der `ContactURI` in dem Todo Objekt abgespeichert.

Wir hatten hier zunächst überlegt ob es nicht sinnvoller wäre direkt nur die `ContactURI` abzuspeichern, und dann in der `TodoActivity` bzw. dem `TodosAdapter` bei der Anzeige für jedes Todo den Kontaktnamen abzufragen. Vorteil wäre hier, dass man flexibel gegenüber Änderungen in den Kontaktdaten wäre (also wenn der Benutzer bspw. den Namen des Kontaktes ändert). Nachteil wäre wiederum, dass für jedes Todo über einen Cursor eine Auswahl stattfinden müsste, was eventuell leichte Performance-Einbußen verursachen würde. Da wir ja lediglich den Namen des Kontaktes anzeigen wollen, und sich dieser normalerweise nicht mehr ändern sollte, haben wir uns entschlossen direkt den Namen des Kontaktes abzufragen und in der Datenbank zu speichern.

Um das Datum festzulegen benutzen wir wieder eine externe Library, den `MaterialDateTimePicker [3]`, bzw. hier lediglich den `DatePicker`.

Sobald der Benutzer das Todo abspeichert, werden alle Eigenschaften ausgelesen, in das Todo Objekt gespeichert, und dieses wieder an die aufrufende Activity (`TodoActivity`) übergeben.

AlarmManager

Nachdem der Benutzer ein Datum eingegeben hat, werden zunächst die aktuellen Settings abgefragt. Falls der Benutzer Notifications aktiviert hat, bzw. nur Notifications für Todos mit hoher Priorität empfangen möchte und es sich bei dem Todo um eins mit hoher Priorität handelt, wird in `setAlarm` das Datum ausgewertet und über den `AlarmManager` ein Alarm für 08:00 an dem entsprechenden Tag registriert.

AlarmReceiver

Der `AlarmReceiver` ist ein `BroadcastReceiver` welcher den ausgelösten Alarm empfängt und eine Notification an den Benutzer schickt. Die Notification enthält den Aufgabentext des Todos, und leitet bei Klick auf die zugehörige Todo-Liste (in der `TodoActivity`) weiter.

Im `AlarmReceiver` werden erneut die Einstellungen des Benutzers abgefragt, für den Fall dass der Benutzer zwischen dem Erstellen des Todos (und somit Registrierung des Alarms) und dem Fälligkeitsdatum an dem der `AlarmReceiver` startet, Notifications deaktiviert hat.

SettingsActivity

Die `SettingsActivity` verwaltet die Einstellungen des Benutzers. Es gibt momentan zwei Einstellungen:

- Notifications für Todos aktivieren Ja/Nein
- Nur Notifications für Todos mit hoher Priorität erhalten Ja/Nein

Die Einstellungen wurden in einer `PreferenceScreen` XML Datei definiert und werden über `addPreferencesFromResource` in `MyPreferenceFragment` eingefügt. Wir haben hier bewusst nicht die `PreferenceActivity` als Grundlage benutzt, sondern ein `PrefenceFragment`, damit die `SettingsActivity` einer normale `AppCompatActivity` ist und die Toolbar anzeigen kann. Sollten in Zukunft weitere Einstellungen hinzukommen, hätte dies natürlich auch den Vorteil, dass mithilfe des Fragments die Darstellungen auf großen Geräten (bspw. Tablets) responsiver ist.

Die Einstellungen des Benutzers werden automatisch als Key-Value Paare abgespeichert, und können von überall in der Applikation über den `PreferenceManager` abgefragt werden.

LOKALISIERUNG

Die Applikation wurde ursprünglich in Englisch geschrieben, inzwischen wurden alle Texte allerdings auch auf Deutsch übersetzt und befinden sich in der `de/strings.xml` Datei. Damit werden, wenn die Systemsprache des Benutzers Deutsch ist, automatisch die deutschen Texte angezeigt.

MOBILE SOFTWARE

Die Applikation befindet sich zurzeit in einer Alpha-Phase im Google Play Store. Falls wir sie später einmal öffentlich machen wollen gibt es noch einiges zu betrachten. So haben wir aufgrund eines Fehlers am Anfang der Entwicklung die App nur für API-Level 23 (Android 6.0) und höher entwickelt, was man nachträglich noch ändern müsste. Eventuell müssen einige Aspekte wie das Umgehen mit Permissions dann etwas umgeschrieben werden.

Außerdem wäre es vielleicht sinnvoll die Listen und Todo-Ansicht in Fragments auszulagern, und auf größeren Geräten eine parallele Ansicht zu ermöglichen. Um mit anderen Todo-Applikationen konkurrieren zu können, wäre es auch nötig die im Segment ‚Vergleich mit anderen Applikationen' genannten Features wie serverseitiges Speichern von Todos zur Synchronisation auf verschiedenen Geräten oder gar das Teilen von Listen/Todos mit anderen Benutzern zu implementieren.

FAZIT

Wir haben in diesem Projekt eine inkrementelle Vorgehensweise verfolgt, und die App immer weiter ausgebaut. Angefangen mit einem einfachen View wo Benutzer neue Todos in einem `EditText` eingeben konnten, wurde dieses Konzept prozedural erweitert. Todos wurden in die Datenbank eingefügt und in Listen zusammengefügt. Neue Eigenschaften kamen hinzu, und das Bearbeiten von Todos wurde in eine neue `Activity` ausgelagert.

Dabei haben wir es leider nicht geschafft alle in der Vorlesung vorgestellten Features und Funktionalitäten in unsere App aufzunehmen. So konnten wir beispielsweise die Features des `LocationManager` nicht implementieren, und auch jegliche serverseitige Kommunikation ist nicht vorhanden, weshalb auch die Aspekte der Mobility in dieser Dokumentation nur kurz angeschnitten wurden.

Dennoch haben wir viele Grundlagen der Android Entwicklung gelernt und diese auch (zumindest im Ansatz) erfolgreich umsetzen können. Zwar sind einige Funktionen unserer Applikation nicht optimal gelöst oder noch etwas verbugged, aber in ihren Grundlagen funktioniert die App mit den wichtigsten Features die wir von Anfang implementieren wollten.

REFERENCES

1. StickyListHeaders Library auf Github
   https://github.com/emilsjolander/StickyListHeaders
2. Android Dokumentation: Parcel
   https://developer.android.com/reference/android/os/Parcel.html
3. MaterialDateTimePicker Library auf Github
   https://github.com/wdullaer/MaterialDateTimePicker