Entwicklerhandbuch
- HfTL-APP -

Stand: 15. September 2015

INHALTSVERZEICHNIS

1 Allgemeines **3**

2 Verwendete Software **3**

3 Aufbau des Projekts **5**
3.1 Manifest.XML 5
3.2 Ordnerstruktur 6

4 Activities **10**
4.1 NewsActivity.java 10
4.2 EinstellungsActivity.java 11

5 NewsClickedActivity.java **12**
5.1 Allgemein 12
5.2 Klasse DetailHelper 12

6 Fragmente **14**
6.1 NewsFragment 14
6.2 Klasse NewsHelper 16
6.3 NavigationDrawerFragment 17
6.4 Notenfragment 19
6.5 StundenplanFragment 23

7 CustomAdapter **25**
7.1 Allgemein 25
7.2 CustomAdapterNews.java 27
7.3 CustomAdapterNoten.java 27
7.4 CustomAdapterStundenplan 28

8 XML-Dateien **29**
8.1 fragment.xml 29
8.2 _list.xml 29
8.3 activity_news_clicked.xml 29
8.4 settings_toolbar 29
8.5 impressum.xml & activity_impressum 29
8.6 colors.xml 30
8.7 array.xml 30
8.8 einstellung.xml 30
8.9 strings.xml 30

9 Layout **32**
9.1 Allgemeines 32
9.2 Schriftarten 32

9.3 Buttons 34

10 Erweiterungen und Verbesserungen für kommende Versionen 36
10.1 Pflege der array.xml 36
10.2 Raumplan 36
10.3 Portierung auf andere Systeme 37
10.4 Speiseplan 37
10.5 Login und Funktionen für Lehrkräfte 37
10.6 Unterstützung von Tabletts 37
10.7 Landscape-Modus 37
10.8 Bekannte Fehler 37

11 UML 38

1 Allgemeines

Dieses Dokument dient lediglich als Hilfswerkzeug zur (Weiter-)Entwicklung und Wartung der HfTL-App. In diesem Dokument ist der grobe Aufbau, sowie die wichtigsten verwendeten Funktion mitsamt zugehörigen Quelltext erklärt.
Dieses Dokument ist keine Programmieranleitung.
Es empfiehlt sich, gewisse Grundkenntnisse in objektorientierter Programmierung im Allgemeinen und in JAVA, XML und AndroidStudio im Speziellen mitzubringen, um dieses Dokument effizient nutzen zu können.
Standard-Methoden und Klassen sind nicht im Detail erklärt, da das den Rahmen dieses Dokuments übersteigen würde. Für tiefer greifende Informationen wird die Android-API empfohlen.

2 Verwendete Software

AndroidStudio

AndroidStudio ist die Standard-Entwicklungsumgebung für Android. Es bietet bereits ein fertiges Gerüst für eine funktionsfähige App an. Das Programm bietet Klassenbibliotheken, Debugger und selbst ein Emulator mit dessen Hilfe Android-Endgeräte auf dem PC virtuell dargestellt werden können.
Das Programm kann kostenlos unter developer.android.com heruntergeladen werden.

GitHub

GitHub ist ein onlinebasierter Dienst, der es einem ermöglicht, Dateien zu hosten und parallel im Team zu bearbeiten. Zudem bietet GitHub eine einfache und übersichtliche Form der Versionierung.
Durch des Erstellen unterschiedlicher Repositorys kann an mehreren Stellen eines Projekts gleichzeitig gearbeitet und getestet werden, ohne Datenverluste befürchten zu müssen.

Gimp

Gimp ist ein Open-Source Bildbearbeitungsprogramm. Es wurde in dem konkreten Fall genutzt, um Grafiken für die App zu erstellen.
Das Programm kann kostenlos auf der Seite des Herstellers (www.gimp.org) herunter geladen werden.

Microsoft Office

MS Office ist das Office Paket von Microsoft, welches bei der Entwicklung der App nur eine beiläufige Rolle spielte. Genutzt wurde insbesondere MS Word zur schnellen Erstellung von Fließtext bzw. der Dokumentation. Die Eigentliche Aufbereitung des Textes erfolgte dann in LaTex.
Die Erstellung der Organigramme bzw. Pläne erfolgte durch MS Excel.

LaTeX

ist ein Softwarepaket, das die Benutzung des Textsatzsystems TeX mit Hilfe von Makros vereinfacht. Die Projektdokumentation wurde mit TeX entwickelt

3 Aufbau des Projekts

3.1 Manifest.XML

Diese Datei ist für jede Android-App zwingend notwendig. Hier werden grundsätzliche Dinge definiert, z.B. Welche Berechtigung diese App benötigt und auf welche Hardware im laufenden Zustand zugegriffen werden muss.
Des Weiteren wird hier auch das package für den Javaquellcode definiert:

Listing 1: AndroidManifest.XML

```
<manifest xmlns:android="..." package="bkmi.de.hftl_app" >
```

Die Zugriffsberechtigungen sind wie folgt definiert:

Listing 2: AndroidManifest.XML

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.
    ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE
    " />
```

Bei der Installation der App wird der Nutzer entsprechend informiert, dass die App auf die jeweiligen Funktionen des Endgerätes zugreift.
In der manifest.xml ist ebenfalls eine Übersicht über die verwendeten Verzeichnisse und Komponenten hinterlegt, z.B. für die Activities.
Bei der HfTL-App ist der Verweis für die MainActivity (die Activity, mit der die App startet) für die NewsActivity gesetzt.
Die folgenden Tags wurden bei der App nicht verwendet, sind aber theoretisch für Erweiterungen möglich:

Listing 3: Zugriffsbeispiel

```
<uses-feature android:name="android.hardware.camera" />
```

Das wäre ein Tag, um im laufenden Betrieb auf die Kamera des Telefons zuzugreifen.

3.2 ORDNERSTRUKTUR

Database

- beinhaltet die NotenDB.java – Inhalt ist der Connector und die Kernfunktionen um Inhalte der eigentlichen Datenbank zu aktualisieren und zu modifizieren
- NotenTabelle.java – das ist die eigentliche Datenbank, bzw. die eigentliche Definition vom Aufbau der Datenbank

Fragmente

- beinhaltet die Fragmente, die von den Activties verwendet werden.

Service

- beinhaltet die Datei NotenService.java, die zum Erzeugen von Push-Nachrichten dient, sobald es in der Notenübersicht neue Noten für den jeweiligen Studenten gibt.

Help

- beinhaltet diverse Hilfsunktionen, die unter anderem zum Ausführen von Threads dienen.

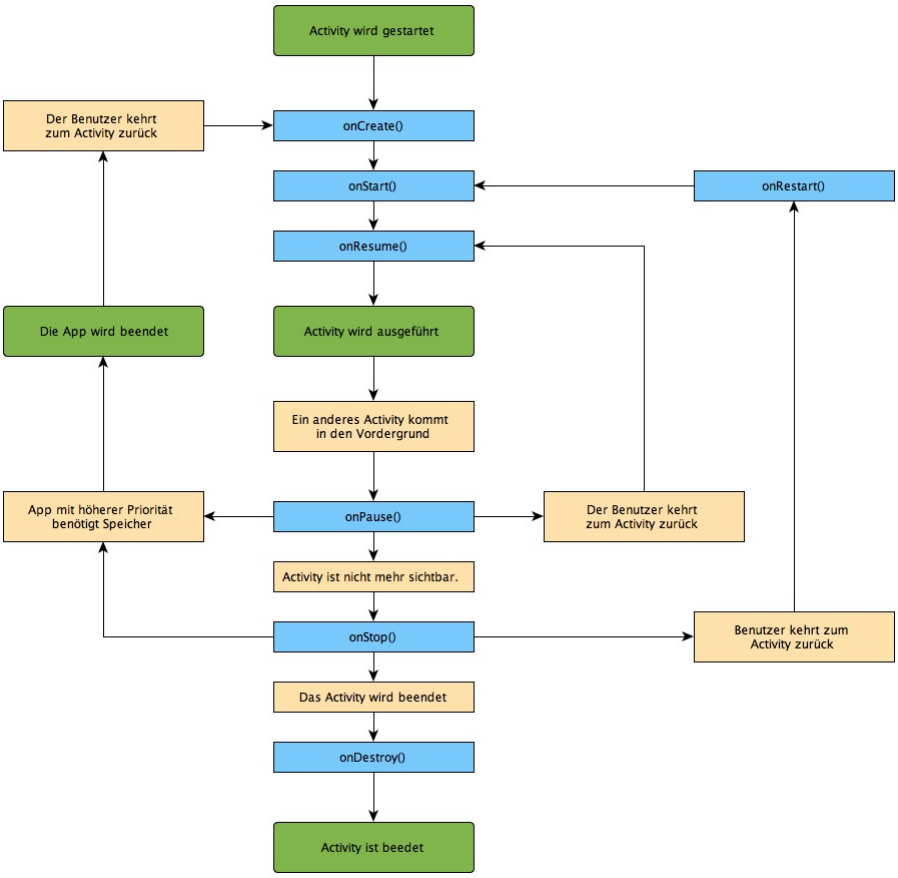
Ressourcen

- unter /src/main/res befindet sich eine Ordnerstruktur, welche XML- und Bilddateien für verschiedenste Anwendungszwecke beinhaltet. Diese werden beim Kompilieren des Projekts in die Ressourcendatei R.java geschrieben. Über diese Datei, wird dann auf die Ressourcen zugegriffen.
  Es wird im Weiteren nicht detailliert auf alle Dateien eingegangen. Es wird nur auf jene Dateien eingegangen, deren Inhalt zur Schilderung von wichtigen Kernfunktionen dienlich ist.

Activities

- Eine Activity stellt ein sichtbares Fenster dar, welches die eigentliche Interaktion mit dem Nutzer ermöglicht. Diese Interaktionen werden als Events bezeichnet bzw. behandelt. Durch sogenannte Intents ist es möglich andere Activitys zu starten. Der sichtbare Teil der Activity wird durch eine View definiert. Dort befinden sich dann die Eventauslöser, wie zum Beispiel Buttons etc.

- Folgende Grafik dient zur Übersicht:

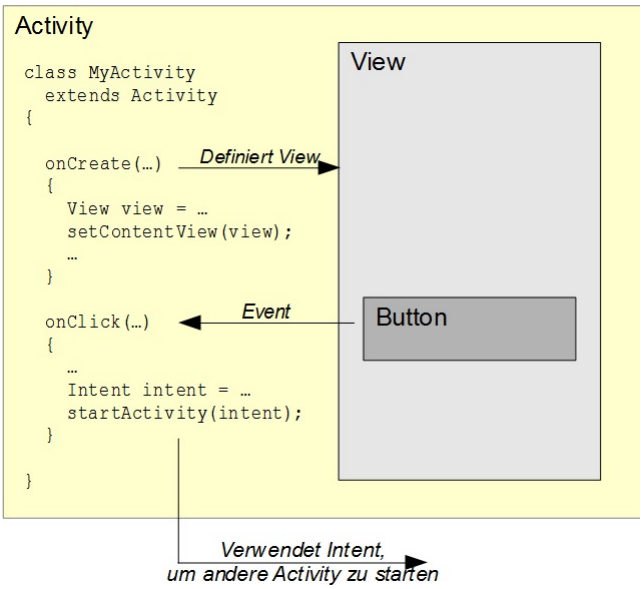


- Jede Activity hat einen sogenannten “Lifecycle“ (=Lebenszyklus), der von dem Betriebssystem gesteuert und verwaltet wird. Dem Entwickler ist es nun überlassen, welche Ressourcen er entsprechend sichert, sollten die Ressourcen auf dem Mobiltelefon knapp werden. Das passiert in den Events:

  - onPause(): die App ist noch sichtbar, aber nicht mehr im Vordergrund

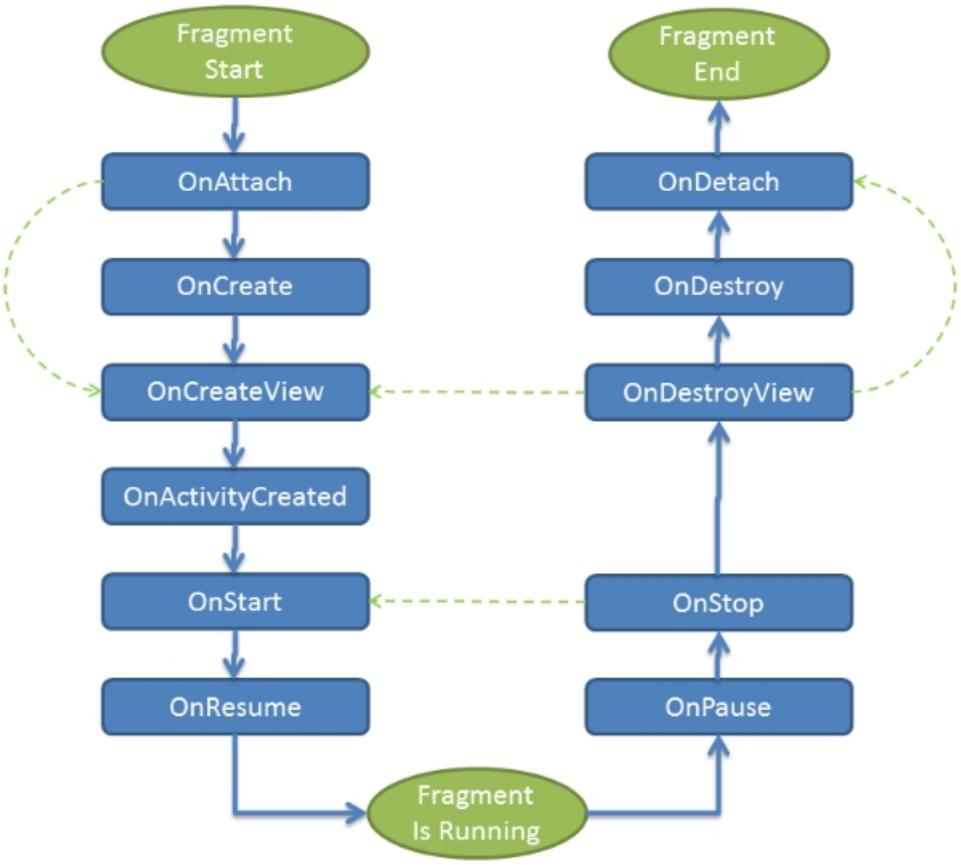
  - onStop(): die App ist nicht mehr sichtbar

- Lifecycle einer Activity:


Activity
class MyActivity
extends Activity
{
onCreate(…)
{
View view = …
setContentView(view);
…
}
onClick(…)
{
…
Intent intent = …
startActivity(intent);
}
}
View
Definiert View
Event
Button
Verwendet Intent,
um andere Activity zu starten

Fragmente

- In Android ist es nicht möglich, mehrere Activities gleichzeitig anzuzeigen, wenn es der Platz auf dem Display theoretisch erlauben würde. Um das dem Entwickler dennoch zu ermöglichen, bietet Android den Objekttypen “Fragment“ an.
- Fragmente sind UI-Container, die über laufende Activities gelegt werden. Dies macht besonders dann Sinn, wenn es um die Fragmentierung der App geht (verschiedene Displaygrößen) bzw. wenn man zwischen den Ansichten wechselt (Haltung des Telefons)
- Ebenso wie eine Activity hat ein Fragment ein Lifecycle, welcher in direkter Kommunikation mit dem Lifecycle der Activity steht. So werden zum Beispiel alle Fragmente einer Activity pausiert, wenn die Activity selbst pausiert wird. Folgendes Diagramm soll dies veranschaulichen:



4 Activities

4.1 NewsActivity.java

- Klasse wird mit Fragment extended
- NavigationDrawerFragment wird benötigt, um die Interaktion mit Fragmenten der Anzeige zu ermöglichen
- Intent wird benötigt, um andere Activities neben der NewsActivity zu verwalten.
- mTitle ist der Titel des zuletzt geladenen Screens

onCreate()

- Hier wird das Layout geladen, welches in der zugehörigen xml definiert wurde
- Laden des mNavigationDrawerFragment, zum Abbilden des Menüs am linken Seitenrand
- Laden des Layouts für das Fragment

  Durch das Laden des Layouts für das mNavigationDrawerFragment (durch setUp) wird die Funktion onNavigationDrawerItemSelected gerufen, weil in der Layoutdefintion Item 1 der Listview ausgewählt wird.

onNavigationDrawerItemSelected()

- Durch das Aufrufen der Funktion wird das eigentliche Fragment für den FragmentManager ausgewählt (durch die switch-case-Anweisung) und abschließend durch das commit aktiv geladen.

onSectionAttached()

- Je nach Auswahl wird hier mTitle aktualisiert.

restoreActionBar()

- Dient zum Aktualisieren der Titelleiste (durch setTitle und mTitle als Argument)

onCreateOptionsMenu()

- Erstellt das Menü oben rechts (drei Punkte) und befüllt es mit den Daten aus der /res/menu/news.xml

onOptionsItemSelected()

- Überprüft welches Element aus dem Menü ausgewählt wurde

4.2 EINSTELLUNGSACTIVITY.JAVA

onCreate()

- Einbinden der “einstellung.xml“ (beinhaltet Definitionen für Stringvariablen, Listen und Checkboxen)
- Setzen von Startwerten für shared, check und list
- Ausführen von registerPreferenceListener()

registerPreferenceListener()

- Es wird ein anonymer Listener erstellt, der auf Änderungen in der SharedPreference.xml reagiert
- In der Methode onSharedPreferenceChanged(SharedPreferences prefs, String key) werden die Änderungen abgefangen
- Listener wird am SharedPreferences Objekt registriert

testeBenutzerdaten()

- Funktion zum Überprüfen von Anmeldedaten
- via TextSecure ts wird Ver- und Entschlüsselung gewährleistet

keineBenutzerdaten()

- Funktion zum Ausgeben, dass die Anmeldeinformationen falsch eingegeben wurden

5 NewsClickedActivity.java

5.1 Allgemein

Diese Activity wird geladen, sobald in der News-Übersicht (NewsActivity.java) ein Eintrag geöffnet wird.
Der Inhalt wird durch NewsResolver und dessen Funktion getDetailsStringArray in das String-Array “s“ geladen.

onCreate()

- Laden des in der zugehörigen xml definierten Layouts (activity_ news_ clicked.xml)
- Einbinden der “Extras“ (Übergebene Variablen)
- Zuweisung der URL aus dem rufenden NewsFragment
- Aufruf des DetailHelpers durch

```
new Detailhelper().execute();
```

5.2 Klasse DetailHelper

onPreExecute()

- Anzeige eines ProgressDialogs (Ladebalken mit Hinweistext), um den Anwender zu informieren.

```
ladebalken = ProgressDialog.show(NewsClickedActivity.this, "
    Bitte warten", "Nachricht wird geladen", true, false);
```

onPostExecute()

- Ressourcen des ProgressDialogs werden freigegeben

```
ladebalken.dismiss();
```

- TextViews 1 bis 3 werden mit den Inhalten (Strings) befüllt.

```
public void run() {
    tv1=(TextView)findViewById(R.id.tv_news_date);
    ...
    tv1.setText(s[2]);
}
```

- Der User kann den Inhalt der TextViews kopieren und in die Zwischenablage des Endgeräts speichern.

```
tv1.setTextIsSelectable(true);
```

- Im TextView 4(*Inhalt der News*) werden einige besondere Anweisungen benötigt:
  - Darstellung und Verfolgung von (Hyper-)links.

```
tv4.setClickable(true);
tv4.setMovementMethod(LinkMovementMethod.getInstance());
```

  - Verarbeitung des vom NewsResolver übergebenen HTML-Strings

```
tv4.setText(Html.fromHtml(s[3]));
```

doInBackground()

- Neuinstanzierung eines NewsResolvers mit übergebener Url, um durch die Funktion getDetailsStringArray das Array "s" mit Daten zu füllen

```
 s[0]=elements.get(0).child(1).text()+"\n";    //Ueberschrift
 s[1]=elements.get(0).child(2).text()+"\n";    //Subhead
 s[2]=elements.get(0).child(0).text();         //Zeit
 s[3]=elements.get(0).child(0).outerHtml().replaceAll("<p>&
    nbsp;</p>", "");                    //Text
return s;
```

6 FRAGMENTE

6.1 NEWSFRAGMENT

Initialisierung erfolgt durch „NewInstance“, die aus der NewsActivity heraus gerufen wird. (siehe Funktionsaufruf -> onNavigationDrawerItemSelected) Durch das .commit wird diese neue Instanz der Klasse dann geladen.

Überschriebene Funktionen

OnAttach()

- Verknüpfung des NewsFragments mit der MainActivity

OnCreateView()

- Hier wird lediglich das Layout der View des Fragments geladen und angewendet

onActivityCreated()

- es wird geprüft, ob es SavedInstances (bereitsgeladene Inhalte) gibt
- wenn JA:
  - die Informationen werden aus dem ARRAYSPEICHER geholt und angewandt
- wenn NEIN:
  - die Funktion „zeigeNews“ wird gerufen, welche die aktuellsten News vom Server lädt

onStart()

- in dieser Funktion wird nur noch der Listener für den Aktualisierbutton mit der Schaltfläche verknüpft. Als onClick-Event wird dann lediglich die Funktion „zeigeNews“ gerufen.

standalone-Funktionen

isOnline()

- Diese Methode prüft durch einen Connectivity Manager, ob eine Verbindung zum Internet besteht

zeigeNews()

- Zunächst wird durch „isOnline“ geprüft, ob eine aktive Netzverbindung besteht
- wenn NEIN:
  - Es wird ein Hinweis an den Nutzer ausgegeben
- wenn JA:
  - Es wird eine Instanz der Klasse NewsHelper erstellt, welche dann als Hintergrund-Task ausgeführt wird, um ein Einfrieren der App zu verhindern.

6.2 KLASSE NEWSHELPER

- Klasse mit asynchroner Task-Ausführung
- Nach dem Aufruf der Methode *zeigenews* wird durch den Unterfunktionsaufruf *.execute* zunächst die Funktion *doInBackground* ausgeführt, wo eine neue Instanz des NewsResolvers erstellt wird.

```
private void zeigeNews(){
 NewsHelper nh = new NewsHelper();
 nh.execute();
}
```

- Durch *getTermineStringArray* des NewsResolvers wird ein String-Array zurückgegeben.
- Danach wird die Methode *onPostExecute* gerufen.

onListItemClick()

- Es wird ein intent verwendet um die *NewsClickedActivity* zu starten, als "Übergabeparameter" wird *putExtra* verwendet, im Falle der App die URL zu dem angeklickten Event.

```
 intent = new Intent(getActivity(), NewsClickedActivity.class)
    ;
 intent.puExtra(TERMINDETAIL, newsResolver.getURLAsString(
    position));
 startActivity(intent);
```

onPostExecute()

- Überprüfung ob das Fragment noch aktiv ist

```
if(getActivity()==null) return;
```

- einbinden des CustomAdapterNews in die ListView

```
setListAdapter(new CustomAdapterNews(getActivity(), ...);
```

6.3 NavigationDrawerFragment

- Dieses Fragment bildet das Menü auf dem linken Rand der App ab und aktiviert den Button mit dem man in die Einstellungen gelangt.

onCreate()

- In dieser Methode werden die Einstellungen der Activity übernommen und der Drawer ausgewählt.
- Außerdem wird geprüft ob eine gesicherte Instanz vorhanden ist, aus der dann das zuletzt ausgewählte Fragment ermittelt wird und über die Funktion *selectItem()* aufgerufen wird.
- Falls keine gespeicherte Instanz existiert, wird die "0" (*NewsFragment*) als Standardwert übergeben.

onActivityCreated()

- Hier wird das Menü für das aktuelle Fragment aktiviert indem an *setHasOptionsMenu()* "true" übergeben wird.

onCreateView()

- In dieser Funktion wird das Design für die Actionbar (als Listview) und die dazugehörigen Menüpunkte festgelegt.
- Zudem wird hier der ClickListener initialisiert, der dann den ausgewählten Eintrag an *selectItem()* übergibt.

isDrawerOpen()

- Diese Methode prüft ob der Drawer bereits offen ist und liefert einen Wahrheitswert zurück.

setUp()

- Es werden hier folgende Einstellungen vorgenommen:
  - Einstellungen zum Design
  - Aktivierung des HomeButtons und dessen Animation beim Draufklicken
  - Zusammenführung der *ActionBar* und des *NavigationDrawers*
  - weitere Einstellungen zum Drawer
- Dann wird die Konfiguration in *mDrawerToggle* abgespeichert

selectItem()

- Hier wird die Animation auf den angeklickten Menüpunkt ausgeführt.
- Zudem wird der Drawer geschlossen und die Position des angeklickten Punktes an *mCallbacks.onNavigationDrawerItemSelected()* übergeben.

onAttach()

- Diese Methode setzt den Zeiger mCallbacks auf die Activity.

onDetach()

- Hier wird der Zeiger *mCallbacks* auf "NULL" gesetzt.

onSaveInstanceState()

- Diese Funktion sichert die aktuelle Instanz.

onConfigurationChanged()

- Bei einer Änderung in den Einstellungen konfiguriert diese Methode *mDrawer-Toggle* um.

onCreateOptionsMenu()

- Diese Funktion legt das Design, aus einer XML-Datei für das Menü Einstellungen, fest.

onOptionsItemSelected()

- Hier wird der Button, über den man zu den Einstellungen gelangt, aktiviert und mit dessen Klasse verknüpft.

NavigationDrawerCallbacks()

- Hier wird die ausgewählte Menüpunkt-ID an die Activity übergeben.

6.4 NOTENFRAGMENT

onCreateView()

- Laden des entsprechenden XML-Layouts *fragment_ noten.xml*
- Laden und Zuweisung der Schriftart des TextViews für die Überschrift des Fragments

onAttach()

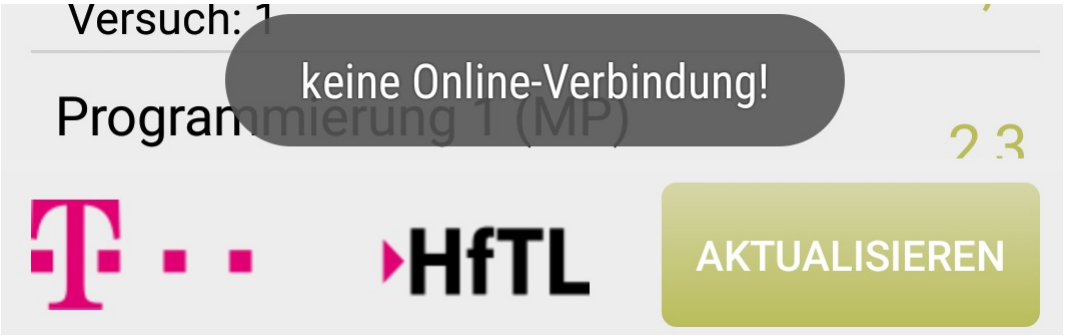
- Aufruf der Datenbank, in der die Noteneinträge abgelegt werden

onDetach()

- Schließen der Datenbank

onStart()

- Erstellen des Buttons zum Aktualisieren
- Mittels *OnClickListener* für den Button, wird über die Methode *testeBenutzerdaten()* überprüft, ob die Benutzerdaten für QiS eingetragen wurden. Andernfalls erfolgt die Ausgabe mittels der Methode *keineBenutzerdaten()*, dass diese nicht eingetragen wurden.
- Sofern die Benutzerdaten eingetragen wurden, und eine Onlineverbindung zu QiS besteht, werden die Noten erneut abgerufen.
- Ist keine Verbindung zu QiS vorhanden, wird der Nutzer mittels Toast-Benachrichtigung darüber informiert



getNoten()

- Liest in der Notendatenbank alle Werte in der Spalte NOTENABFRAGE aus und schreibt diese in das String-Array "*s*", welches auch übergeben wird.

getSemester()

- Liest in der Notendatenbank alle Werte in der Spalte SEMESTERABFRAGE aus und schreibt diese in ein String-Array "*s*", welches auch übergeben wird.

getFach()

- Liest in der Notendatenbank alle Werte in der Spalte FACHABFRAGE aus und schreibt diese in das String-Array "*s*", welches auch übergeben wird.

getVersuche()

- Liest in der Notendatenbank alle Werte in der Spalte VERSUCHABFRAGE aus und schreibt diese in das String-Array "*s*", welches auch übergeben wird.

Hinweis:

Die String-Arrays sind nur lokal in den jeweiligen Methoden definiert, weshalb diese - der Einfachheit halber - alle den Namen "s" erhielten. Tatsächlich werden hier vier unterschiedliche String-Arrays befüllt.

setzeListview()

- Hier werden zunächst entsprechende String-Arrays durch die jeweiligen Methoden zur Abfrage in der Datenbank befüllt:

```
notenList = getNoten();
semesterList = getSemester();
fachList = getFach();
versuchList = getVersuche();
```

  - Die Strings stehen entsprechend der alphabetischen Reihenfolge der Semesterbeschreibung in den Arrays.
  - Bsp: SoSe 12, SoSe 13, WiSe 11/12, WiSe 12/13, WiSe 13/14

- Um die Übersichtlichkeit zu wahren, werden die Werte in den Arrays nun sortiert, wobei die Werte vom aktuellsten Semester am Anfang stehen und anschließend chronologisch geordnet werden.
- Um dies zu realisieren wird zunächst mit Hilfe eines regulären Ausdrucks ein Suchpattern festgelegt, nach dem die zu sortierenden Elemente gefunden werden.
- Entscheidend für die Sortierung sind die letzten beiden Ziffern für die Jahreszahl.

```
Pattern p = Pattern.compile("\\d{2}$");
```

- Anschließend wird der Einfachheit halber ein Bubblesort verwendet (kann ggf. geändert werden), der durch das String-Array semesterList geht und bei benachbarten Einträgen den Suchpattern anwendet.

```
Matcher m = p.matcher(semesterList[j]);
Matcher n = p.matcher(semesterList[j+1]);
```

- Sofern die Pattern in den beiden Einträgen vorhanden sind (das ist eine zwingende Bedingung), werden diese jeweils in Hilfsvariablen geschrieben.

```
if((m.find())&&(n.find())){
 int x = Integer.parseInt(m.group());
 int y = Integer.parseInt(n.group());
 ...}
```

- Danach werden diese Hilfsvaribalen verglichen und ggf sortiert.
- Sofern die Einträge in (semesterList) sortiert werden, müssen auch die zugehörigen Werte in den anderen Arrays (notenList, fachList, (versuchList) entsprechend sortiert werden.

```
if(x<y){
  temps = semesterList[j];
  semesterList[j] = semesterList[j+1];
  semesterList[j+1] = temps;
  temps = notenList[j];
  notenList[j] = notenList[j+1];
  notenList[j+1] = temps;
  temps = fachList[j];
  ...
  }
```

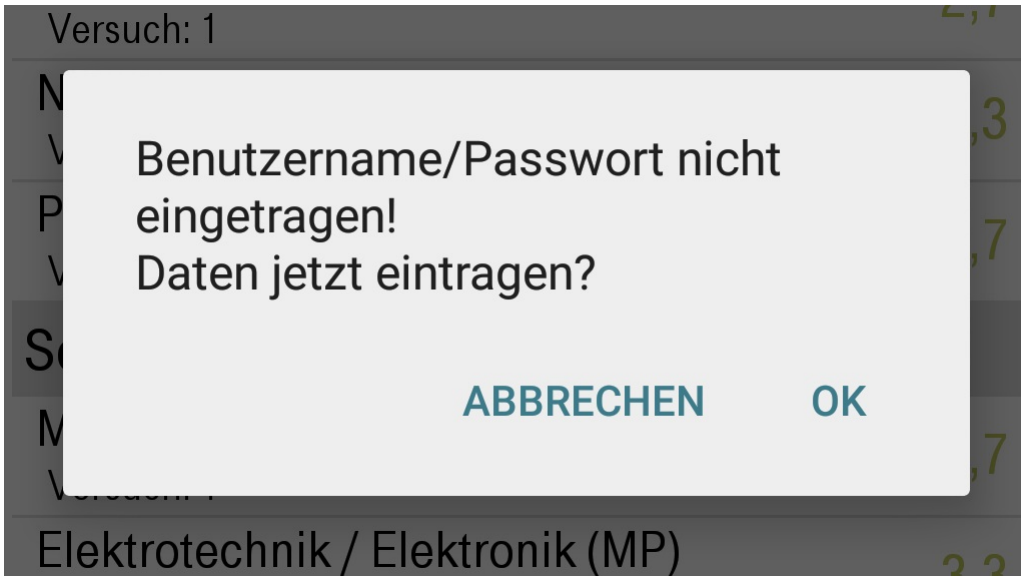
- Sobald die Sortierung erfolgt ist, müssen noch redundante Werte im Array (semesterList) gefunden und durch (NULL) ersetzt werden.
  - Dadurch wird eine Auflistung der Noten, Fächer und Versuche, aufgeschlüsselt nach Semester realisiert, ohne dass über jedem Eintrag die Semesterbeschreibung steht. (s. CustomAdapterNoten.java)

```
String[] vgl = new String[1];
vgl[0]="";
for(int k=0; k<semesterList.length;k++){
   if (semesterList[k].equals(vgl[0])) {
      vgl[0] = semesterList[k];
      semesterList[k] = null;
   }
   else{
      vgl[0] = semesterList[k];
   }
}
```

- Anschließend werden diese String-Arrays an den CustomAdapterNoten übergeben. Damit wird eine individuelle Befüllung und Formatierung der Liste mit den ausgelesenen Werten aus der Datenbank realisiert.

keineBenutzerdaten()

- Diese Methode prüft, ob in den Einstellungen Login und Passwort für QiS eingetragen wurden. Andernfalls wird eine Fehlermeldung ausgegeben:



– NotenHelper (Class)

6.5 StundenplanFragment

newInstance()

- Erstellt ein StundenplanFragment und “steckt“ die aktive Position (aus dem Navigation Drawer) in das “Bundle args“ welches als Argument im Fragment übergeben wird.

onCreateView()

- Laden des entsprechenden XML-Layouts fragment_ noten.xml
- Laden und Zuweisung der Schriftart für das TextView für die Überschrift des Fragments.

onViewCreated()

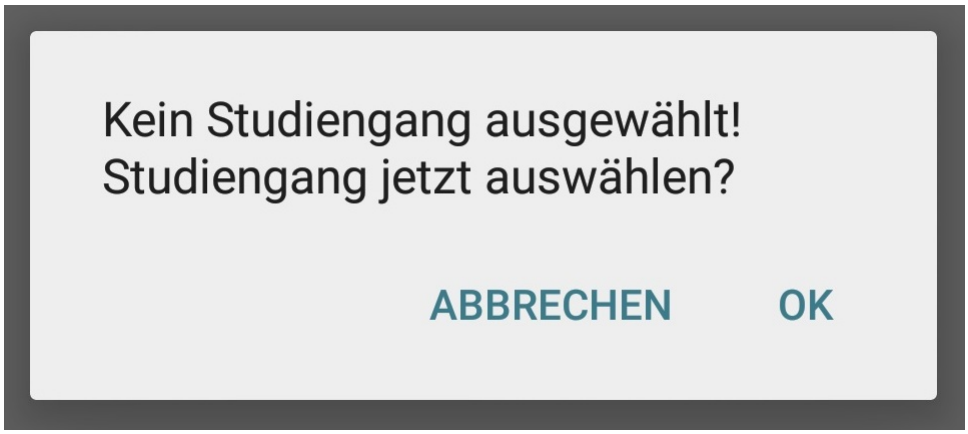
- Falls im “*Stundenplanspeicher*“ Daten vorhanden sind, werden diese geladen.
- Methode für das Dropdownmenü wird gerufen und Array für “events“ erstellt.
- Anonyme Listener für die Buttons werden erstellt.

erzeugeDropdown()

- Dropdown aus xml einem Objekt zuweisen.
- Listener für Dropdown (als anonymer Listener) wird erzeugt und registiert.
- Beim registieren wird die Methode *onItemSelected* aufgerufen und ein StundenplanHelper ausgeführt.
- Mittels eines “*Calendar*“, “*Date*“ und zwei “*SimpleDateFormat*“ wird das Dropdownmenü befüllt, indem die Daten in eine String-List eingefügt werden (*list.add(temp)*).
- Dropdown wird mit Liste verknüpft.

keinStudiengang()

- Prüft ob in den Einstellungen der Studiengang eingetragen wurde, ansonsten wird eine Fehlermeldung ausgegeben:



erstelleStundenplan()

- Erzeugt die Ausgabe des Stundenplans und fügt sie in den ListView ein.
- Falls keine Daten vorhanden sind, wird “keine Daten“ ausgegeben.

– StundenplanHelper (class)

onPreExecute()

- erzeugt einen Ladebalken.

onPostExecute

- falls das Fragment noch aktiv ist wird die Methode erstelleStundenplan() gerufen.
- Ladebalken wird entfernt.

doInBackground

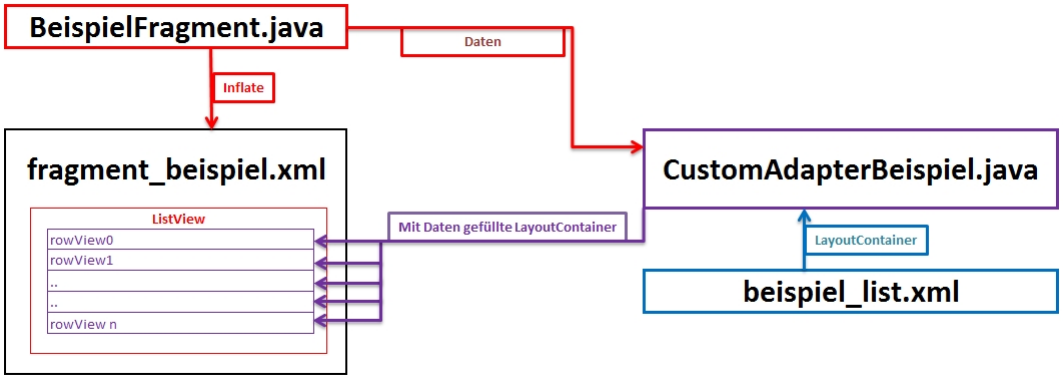
- Die Methode erzeugt einen *StundenplanResolver* und befüllt das “StundenplanEvents-Array *Events*“ mit Daten durch die Methode *erzeugeStundenplan(String woche)* des StundenplanResolvers.
- Falls ein Fehler auftritt, wird ein Event erstellt, in dem keine Daten sind.

7 CustomAdapter

7.1 Allgemein

Die CustomAdapter kommen da zum Einsatz, wo eine ListView genutzt und individuell befüllt werden muss.

1. Fragment.java wird mittels inflate der fragment.xml zugeordnet
2. Fragment.java übergibt die zuvor ermittelten Daten (mit Strings befüllte Arrays) an den CustomAdapter
3. Entsprechend der Länge x der Arrays wird x-mal die zugehörige _list.xml als LayoutContainer aufgerufen und ihre TextViews mit den Inhalten der Arrays befüllt.
4. Die befüllten LayoutContainer werden zeilenweise an die ListView in der fragment.xml übergeben.



class Holder

Dies ist ein Container mit TextViews

```
TextView tv_name;
```

Dieser Container wird dann als jeweilige Zeile in der ListView der zugehörigen Activity dargestellt.
Die Grundlage der Ausgabe bilden spezielle XML-Dateien. Diese dienen definieren die Output jeder einzelnen Zeile der ListViews. Hier sind Name, Position, Dimensionen, Farben, Schriftarten, Schriftgröße etc festlegt.
Für jeden Customadapter gibt es eine entsprechende _list.xml:

- news_list.xml
- noten_list.xml
- stundenplan_list.xml

public View getView()

erstellt einen neuen Container:

```
Holder holder = new Holder;
```

erstellt eine View rowView:

```
View rowView;
```

Das Layout der View und damit jeder Zeile in der ListView wird durch das Laden der entsprechenden XML namelist formatiert:

```
rowView = inflater.inflate(R.layout.name_list, null);
```

Die TextViews des Containers werden den entsprechenden TextViews in der XML zugeordnet:

```
holder.tv_name=(TextView)
rowView.findViewbyId(R.id.namelist_name.xml);
```

Das TextView wird mit dem entsprechenden Wert des zugehörigen String-Arrays befüllt:

```
holder.tv_name.setText(name[position]);
```

Der nun befüllte Container wird nun als rowView übergeben:

```
return rowView;
```

7.2 CustomAdapterNews.java

Dies ist der simpelste CustomAdapter der HfTL-App. Er wird verwendet um die News-Liste (also die Übersicht der News aus der HfTL-Homepage) zu formatieren.
Dabei werden drei String-Arrays (date, headline, content) in das zugehörige TextView der rowView übergeben. Der Inhalt dieser Arrays wird mittels der Klasse NewsHelper, die wiederum die Methoden der Klasse NewsResolver aufruft, befüllt.

7.3 CustomAdapterNoten.java

Dieser CustomAdapter wird für die Befüllung der ListView des Notenfragments benutzt.
Hier werden vier String-Arrays (subject, trys, mark, semester) mit den Daten aus der Notendatenbank befüllt.
Entsprechend des Inhalts, werden die zugehörigen TextViews noch gesondert formatiert.
Ist der Inhalt an der Position des String-Arrays "NULL", wird das zugehörige TextView auf "GONE" gesetzt. So wird realisiert, dass mehrere Fächer unter dem selben Semester gelistet sind, ohne dass ein leeres (dunkelgraues) TextView erscheint.

```
if(semester[position]==null){
 holder.tv_semester.setVisibility(TextView.GONE);
}
```

Auch für die jeweiligen Noten gibt es eine gesonderte Formatierung:

- Note schlechter als 5.0: magenta
- Note schlechter als 3,4: gelb
- Note besser als 3,5: grün

```
if (mark[position].equals("5,0" ))
 holder.tv_mark.setTextColor
  (context.getResources().getColor(R.color.magenta));
else if (mark[position].equals("4,0" ) |
         mark[position].equals("3,9" ) |
         mark[position].equals("3,8" ) |
         mark[position].equals("3,7" ) |
         mark[position].equals("3,6" ) |
         mark[position].equals("3,5" ) )
           holder.tv_mark.setTextColor
            (context.getResources().getColor(R.color.gelb));
    else
         holder.tv_mark.setTextColor
          (context.getResources().getColor(R.color.gruen));
```

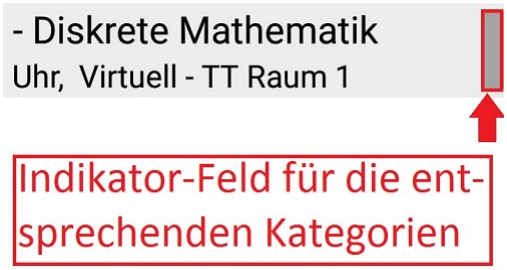
7.4 CustomAdapterStundenplan

Dieser CustomAdapter befüllt die ListView des StundenPlanfragments.
Hier werden fünf StringArrays(datum,fach,zeit,raum,kategorie) übergeben, die zuvor mittels der Methode *erstelleStundenplan()* des StundenplanFragments aus dem HTML-Code der QiS/HiS-Seite ausgelesen wurden.
Auch hier gibt es einige spezifische Formatierungen, abhängig vom übergebenen Inhalt.
Ist der Inhalt des StringArrays datum an einer Stelle "NULL", so wird die Sichtbarkeit des zugehörigen TextViews auf "GONE" gesetzt.
Analog zum CustomAdapterNoten werden so die einzelnen Fächer unter dem selben Datum gelistet. Andernfalls würde über jedem Fach ein dunkelgraues TextView stehen.

```
if(datum[position]!=null)
   holder.tv_date.setText(datum[position]);
else
   holder.tv_date.setVisibility(TextView.GONE);
```

Das StringArray kategorie wird, neben der reinen Ausgabe dazu verwendet, wichtige Ereignisse farblich kenntlich zu machen. Dabei wird der Inhalt an der jeweiligen Position des StringArrays geprüft und eine entsprechende (CI/CD-)Farbe für das Rechteck auf der rechten Seite gesetzt:

- Prüfung: magenta
- Praktikum: dunkelblau
- Rest: grau01




```
if (kategorie[position].equals("Pruefung"))
   holder.tv_category.setBackgroundColor
   (context.getRessources().getColor(R.color.magenta));
else if ((kategorie[position].equals("Praktikum"))
   holder.tv_category.setBackgroundColor
   (context.getRessources().getColor(R.color.dunkelblau));
else
    holder.tv_category.setBackgroundColor
   (context.getRessources().getColor(R.color.grau01));
```

8 XML-Dateien

8.1 fragment.xml

Die fragment.xml bilden das visuelle Rückgrat der App. Hier wird das Layout festgelegt und damit wo welcher Inhalt steht bzw. zu stehen hat. Statische Inhalte und Formatierungen(Farben, Dimensionen, Elementposition, Schriftarten etc.) werden bereits hier festgelegt.
Variable Formatierungen und Inhalte werden entweder direkt durch die zugehörige *Fragment.java* oder - wenn sie die ListViews betreffen - durch den zugehörigen CustomAdapter befüllt.
Auf den detaillierten Aufbau wird nicht näher eingegangen, da dieser mit einigen XML-Kenntnissen selbsterklärend sein dürfte.

8.2 _list.xml

Hier gibt es derzeit drei XML-Dateien:

- news_list.xml
- noten_list.xml
- stundenplan_list.xml

Diese dienen als LayoutContainer für die ListViews. Sie werden durch den entsprechenden CustomAdapter mit den Daten aus der Fragment.java befüllt und zeilenweise an das zugehörige ListView der Fragment.xml übergeben.

8.3 activity_news_clicked.xml

Dient als Designvorlage für eine aufgerufene News aus der Newsübersicht. Der Inhalt wird durch die Klasse NewsClickedActivity.java befüllt.

8.4 settings_toolbar

Definiert die Kopfzeile der einstellung.xml.

8.5 impressum.xml & activity_impressum

Hier steht das Impressum. Während das Layout in der activity_impresum.xml festgelegt wird, wird der Inhalt (*Ähnlich wie bei strings.xml*)zentral abgelegt.

8.6 COLORS.XML

Um den Aufruf der Farben nach CI/CD zu vereinfachen, wurden die RGB-Werte (in hexadezimal) zentral in der Datei main\res \values \colors.xml gespeichert.

```
<item name="magenta" type="color">#E20074</item>
<item name="grau01" type="color">#A4A4A4</item>
...
 <integer-array name="telekomcolors">
   <item>@color/magenta</item>
   <item>@color/grau01</item>
   ...
 </integer-array>
```

Die Farben können dann sowohl mittels JAVA als auch mittels XML aufgerufen werden:

- XML:

```
android:color="@color/magenta"
```

- JAVA:

```
getResources().getColor(R.color.magenta)
```

8.7 ARRAY.XML

Hier stehen alle aktuellen Studiengänge mit Matrikel und ihre zugehörigen Links zu QiS.
Ein Studiengang muss in den Einstellungen ausgewählt werden, damit der Stundenplan angezeigt werden kann.
Ferner sind hier die Werte für das Intervall der Notenabfrage abgelegt.
Die Pflege der Datei erfolgt derzeit manuell. (s. Pflege der array.xml)

8.8 EINSTELLUNG.XML

Layout-Definition des Einstellungsmenüs.

8.9 STRINGS.XML

Statische Strings, wie sie beispielsweise für die Fragmentüberschriften verwendet werden, werden nicht in den XML-Tag geschrieben.

```
<TextView
...
android:text="News aus der HFT-Leipzig"
.../>
```

Vielmehr werden diese zentral in der *strings.xml* abgelegt.

Listing 4: Ablage in strings.xml

```
<string name="News_headline">News aus der HFT-Leipzig</string>
```

Anschließend werden diese Strings dort abgerufen, wo sie benötigt werden.

Listing 5: Aufruf im fragment.xml

```
<TextView
...
android:text="@string/News_headline"
.../>
```

9 Layout

9.1 Allgemeines

Das Layout wird zum Großteil über die XML-Dateien realisiert. Um die Handhabung zu realisieren, wurden einige Elemente bzw. Bezeichnungen standardisiert.
Im Folgenden werden die wichtigsten Details rund um die Darstellung des Layouts erläutert. Mit einigen XML-Kenntnissen sollte der Großteil selbsterklärend sein.

9.2 Schriftarten

Da die Standardbibliothek von Android nur sehr wenige Schriftarten liefert, mussten die Schriftarten nach CI/CD nachträglich eingefügt werden. Diese befinden sich unter main\assets\fonts. Der Aufruf der Schriftarten erfolgt im Allgemeinen über folgende JAVA-Anweisung:

Listing 6: Beispiel: Einbinden der Schriftart Ocra im Notenfragment

```
View notenView = inflater.inflate(R.layout.fragment_noten,
    container, false);

TextView hl = (TextView) notenView.findViewById(R.id.hl_Noten);
Typeface headline = Typeface.createFromAsset(getActivity().
    getAssets(), "fonts/OCRA.TTF");
hl.setTypeface(headline);

return notenView;
```

In der Praxis hat sich jedoch herausgestellt, dass diese Art und Weise, Schriftarten einzubinden auf zwei Probleme trifft:

- Über das Fragment können die Schriftarten nicht den ListViews zugewiesen werden, da bei Erstellung der Fragmente die ListViews bzw. deren Inhalt (die durch die jeweiligen CustomAdapter befüllt werden) noch nicht existieren.
- Es wäre möglich die Schriftarten im jeweiligen CustomAdapter zuzuweisen. Jedoch hat die Praxis gezeigt, dass dadurch der Speicherverbrauch extrem ansteigt (200 bis 300 Prozent) und damit die Performance der App merkbar sinkt. Der Grund hierfür liegt daran, dass bei jedem TextView, das angezeigt wird, die Schriftarten aus den Assets geladen und im Cache abgelegt werden.

Das Problem wurde durch einige Anpassungen umgangen:

- Erstellen der Klasse FontCache
  - Diese Klasse prüft ob die geforderten Schriftarten bereits im Cache hinterlegt sind.

  - Sollten die Schriftarten bereits im Cache hinterlegt sein, dann werden diese genommen. Andernfalls werden sie aus den Assets geladen und im Cache abgelegt.

- Einbinden der Schriftarten in jeweilige Klassen
  - Die Klassen werden direkt durch die TextViews in den XML-Dateien aufgerufen.
  - Beim Aufruf wird die o.g. Klasse *FontCache* aufgerufen.
- Setzen der Schriftarten in den XML-Dateien
  - Um die benötigten Schriftarten zu laden, werden die XML-Tags geändert.
    * Ursprüngliche (native) XML-Tags:

Listing 7: Normales TextView

```
<TextView
  android:id="@+id/newslist_date"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  ... />
```

    * Pfadangabe zur jeweiligen Schriftart als neues XML-Tag:

Listing 8: Einbinden von TeleGrotNorm

```
<bkmi.de.hftl_app.help.Typefaces.TeleGrotNorm
  android:id="@+id/newslist_date"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  ... />
```

9.3 BUTTONS

Grundlegend werden vier Standardbuttons in der App verwendet. Sie alle wurden über eine eigene XML-Datei definiert und durch die XML-Dateien der jeweiligen Fragmente aufgerufen. Die Methoden, die bei Betätigung der Buttons aufgerufen werden, befinden sich in der JAVA-Datei des jeweiligen Fragments.
Durch die Auslagerung der Definitionen für die Buttons ist es u.a. möglich, Farbverläufe, abgerundete Ecken und interaktives Verhalten (bspw. Farbänderungen bei Betätigung) festzulegen. Im Grunde genommen wird nur ein *Item* generiert, das als Vorlage für einen Button herhalten muss.

Exemplarischer Button

- Es gibt drei Zustände:
  - gedrückt

```
<item android:state_pressed="true"> ... </item>
```

  - fokussiert

```
<item android:state_focused="true"> ... </item>
```

  - standard

```
<item> ... </item>
```

- Der Status *focused* wird derzeit nur als Dummy angesehen, da er derzeit nicht verwendet wird.
- Die Definitionen innerhalb der Items findet jeweils zwischen diesen Tags statt:

```
<item>
   <shape> ... </shape>
</item>
```

  - Farbverlauf mit Start- und Endfarbe, sowie dem Winkel des Farbverlaufs.

```
<gradient
  android:startColor="#77E20074"
  android:endColor="#E20074"
  android:angle="270"/>
```

  - Umrandung

```
<stroke
   android:width="3dp"
   android:color="@color/hellblau"/>
```

  - Rundung der Ecken

```
<corners
    android:radius="5dp"/>
```

Folgende Standardbuttons werden benutzt:

custom_button

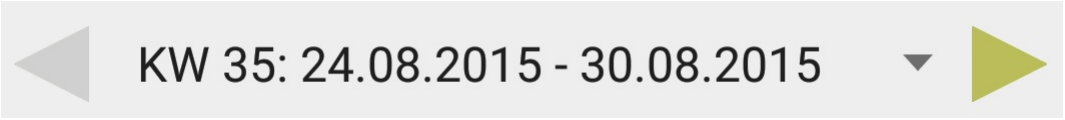
- Dies ist der Standard-Button.

custom_newsclick

- Hier werden die News-Einträge der News-Übersicht als Button verwendet, um ein optisches Feedback zu erhalten, wenn eine News ausgewählt wurde.

custom_vorbutton & custom_zurbutton

- Diese beiden Buttons werden beim Stundenplan verwendet. (Grüne Dreiecke um zwischen den Wochen vor- und zurückzublättern.)



- als Vorlage dienen jpg-Images
- Je nach Status werden die entsprechenden Images geladen:
  - inaktiv: grau
  - aktiv: grün
  - gedrückt (optisches Feedback): magenta

10 Erweiterungen und Verbesserungen für kommende Versionen

Da es aufgrund der Projektrahmenbedingungen nicht möglich gewesen ist, alle gewünschten Features zu implementieren, folgt hier eine Auflistung weiterer Funktionen oder Verbesserungen, die für die App denkbar wären.

10.1 Pflege der array.xml

Die Datei array.xml dient als Grundlage für das Einstellungsmenü.
Hier werden alle Studiengänge, mit den entsprechenden Matrikeln aufgelistet. Das Kernproblem ist, dass diese Liste manuell auf dem aktuellen Stand gehalten werden muss.
Denkbar wäre hier eine Methode, die zu gewissen Zeitpunkten (*Ende der Semesterferien*) die neuen Matrikel hinzufügt. Problematisch dabei sind jedoch mehrere Aspekte, die beachtet werden müssen:

- Festlegung der Zeitpunkte zur Aktualisierung der Daten
- Wegfall der alten Matrikel (Vermeidung von Datenmüll)
- Umbenennung von Studiengängen
- Hinzufügen neuer Studiengänge
- Wegfall alter Studiengänge

Die Problemstellung ist damit für das Projekt zu komplex und kann wahrscheinlich besser in Absprache und Zusammenarbeit mit dem Hochschul- und Prüfungsamt der HfTL gelöst werden.

10.2 Raumplan

Ein Kernfeature, das aber von der Komplexität her wohl ein eigenes Projektteam beschäftigen kann, ist die Raumplanung.
Hierbei soll es dem Nutzer bspw. möglich sein, freie Räume für selbst organisiertes Lernen zu finden.
Wichtig dabei ist auch zu wissen, wann welcher Raum durch wen belegt ist. Insofern müsste es abseits von der zentralen Raumplanung durch die HfTL möglich sein, dass der Nutzer sich einen freien Raum buchen kann und dieser Buchungswunsch dann auch für andere Anwesende zu sehen ist.
Idealerweise gäbe es auch einen Lageplan, wo welcher Raum zu finden ist.

10.3 Portierung auf andere Systeme

Die hier beschriebene App ist eine reine Android-App. Sofern es eine Nachfrage gibt, wäre eine Portierung auf andere Betriebssysteme (iOS, Windows Mobile, Ubuntu Touch und weitere) sicherlich sinnvoll.

10.4 Speiseplan

Eine Funktion zur Einsicht aktueller Kantinenangebote der HfTL und HTWK.

10.5 Login und Funktionen für Lehrkräfte

Eine Erweiterung des Funktionsumfangs der App für die Lehrkräfte der HfTL wäre denkbar, müsste aber noch näher spezifiziert werden.

10.6 Unterstützung von Tabletts

Die App ist gegenwärtig nur für Displaygrößen von Smartphones konzipiert. Eine Darstellung auf Android-Tabletts ist möglich, wurde jedoch nicht getestet und müsste gegebenenfalls in späteren Versionen mit bedacht werden.

10.7 Landscape-Modus

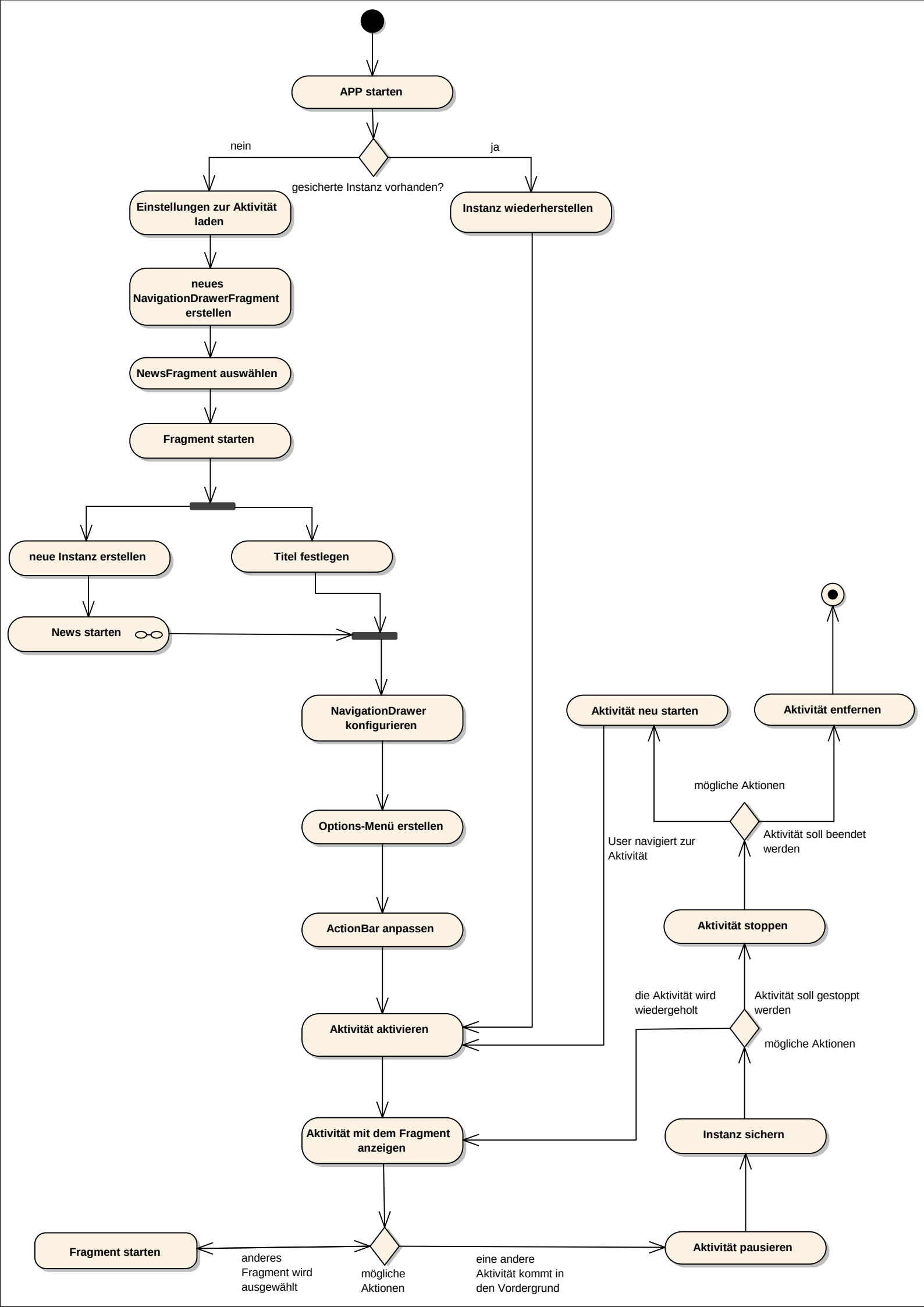
Der Landscape-Modus wird derzeit nicht unterstützt.

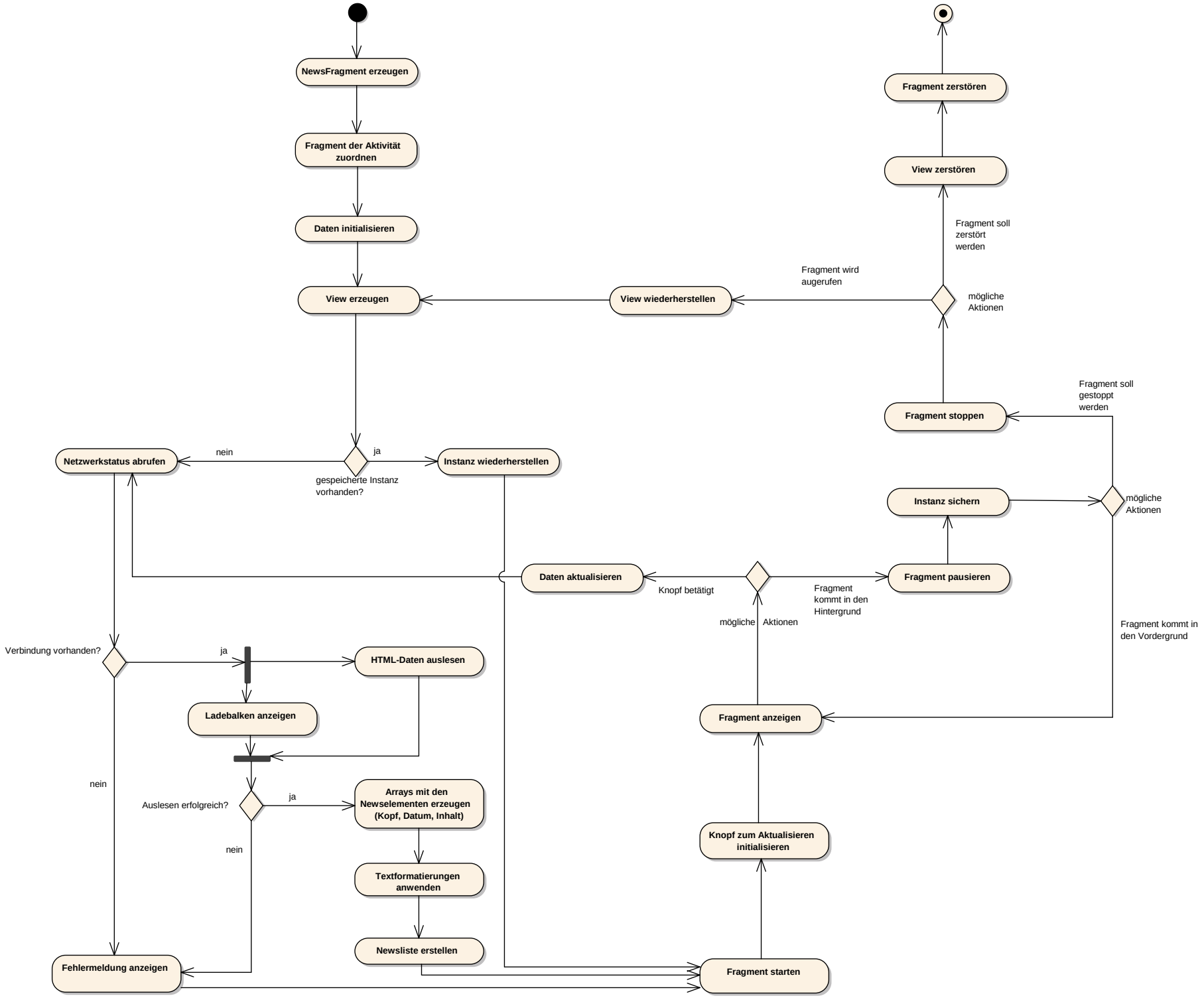
10.8 Bekannte Fehler

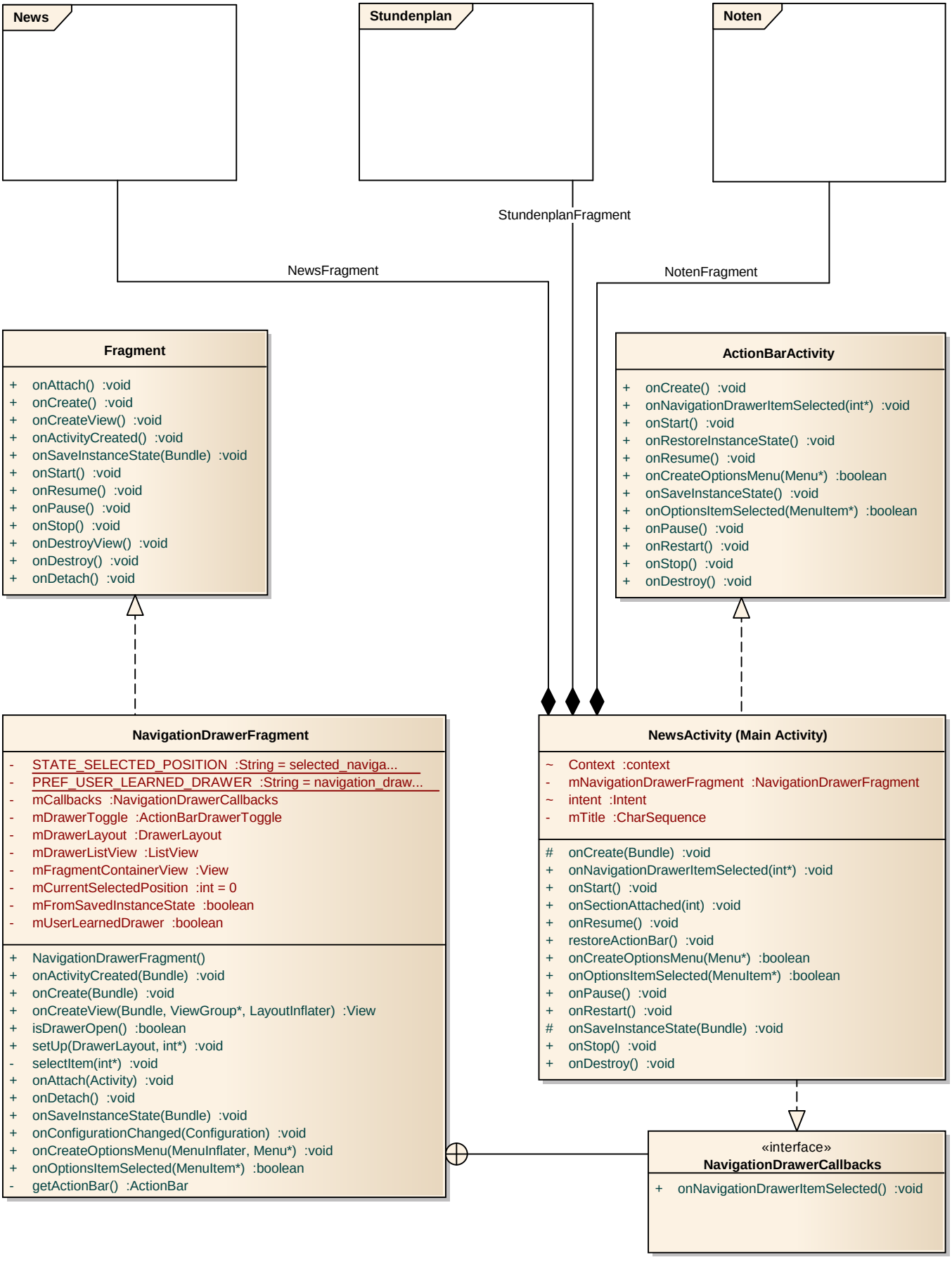
Es gibt einige Bugs bzw. Fehler die in zukünftigen Versionen behoben werden müssen:

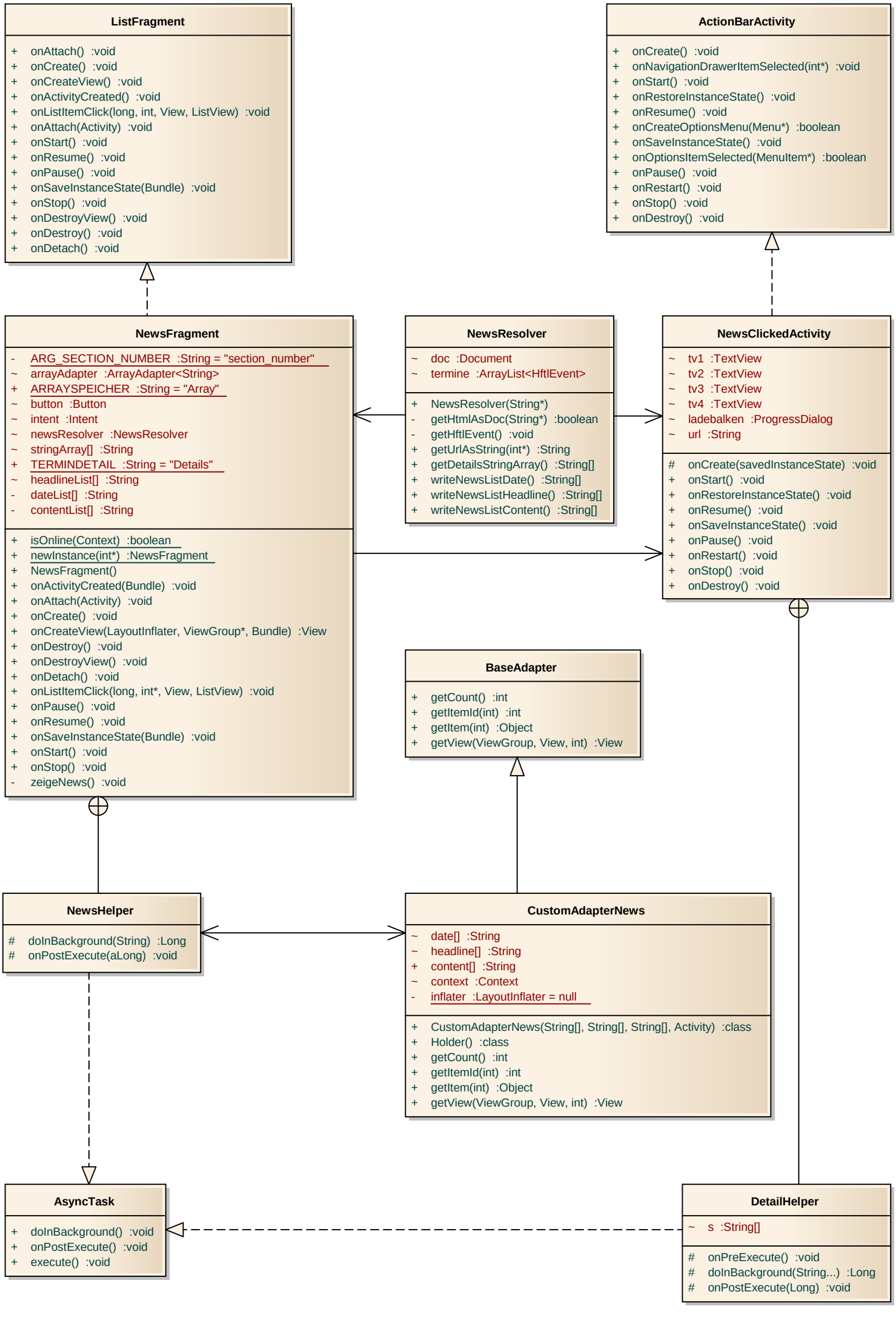
- Ist kein Email-Cient auf dem Smartphone installiert, kann womöglich die App abstürzen, wenn in den News ein Mail-Link aufgerufen wird.
- Sind in den News auf der Homepage der HfTL Links nicht absolut angegeben (*bspw. <a href="de/aktuelles/news.html">*), dann stürzt die App ab.

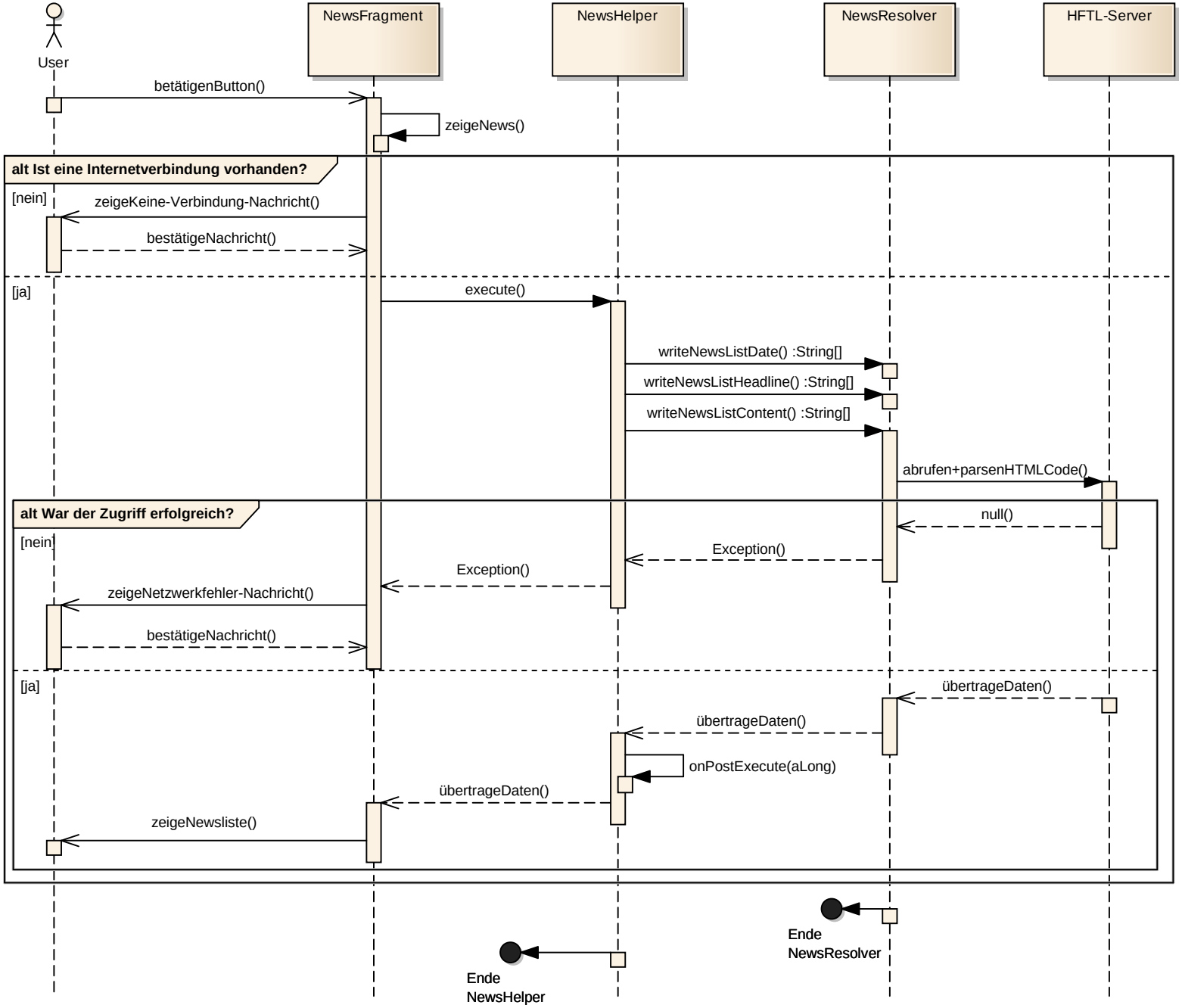
11 UML

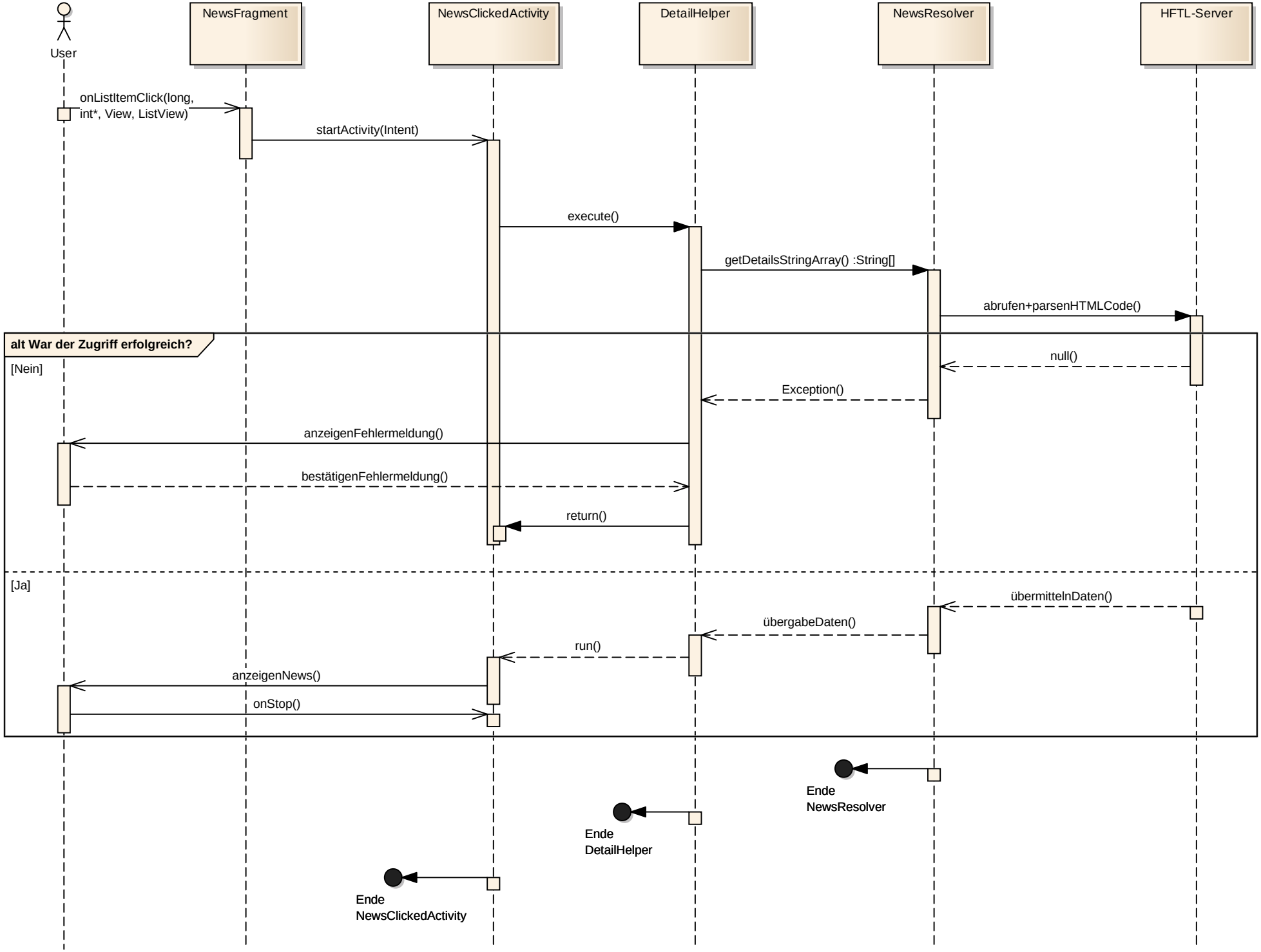

APP starten
nein
ja
gesicherte Instanz vorhanden?
Einstellungen zur Aktivität laden
Instanz wiederherstellen
neues NavigationDrawerFragment erstellen
NewsFragment auswählen
Fragment starten
neue Instanz erstellen
Titel festlegen
News starten
NavigationDrawer konfigurieren
Options-Menü erstellen
ActionBar anpassen
Aktivität aktivieren
Aktivität mit dem Fragment anzeigen
Fragment starten
anderes Fragment wird ausgewählt
mögliche Aktionen
eine andere Aktivität kommt in den Vordergrund
Aktivität pausieren
Instanz sichern
die Aktivität wird wiedergeholt
Aktivität soll gestoppt werden
mögliche Aktionen
Aktivität stoppen
User navigiert zur Aktivität
Aktivität soll beendet werden
mögliche Aktionen
Aktivität neu starten
Aktivität entfernen

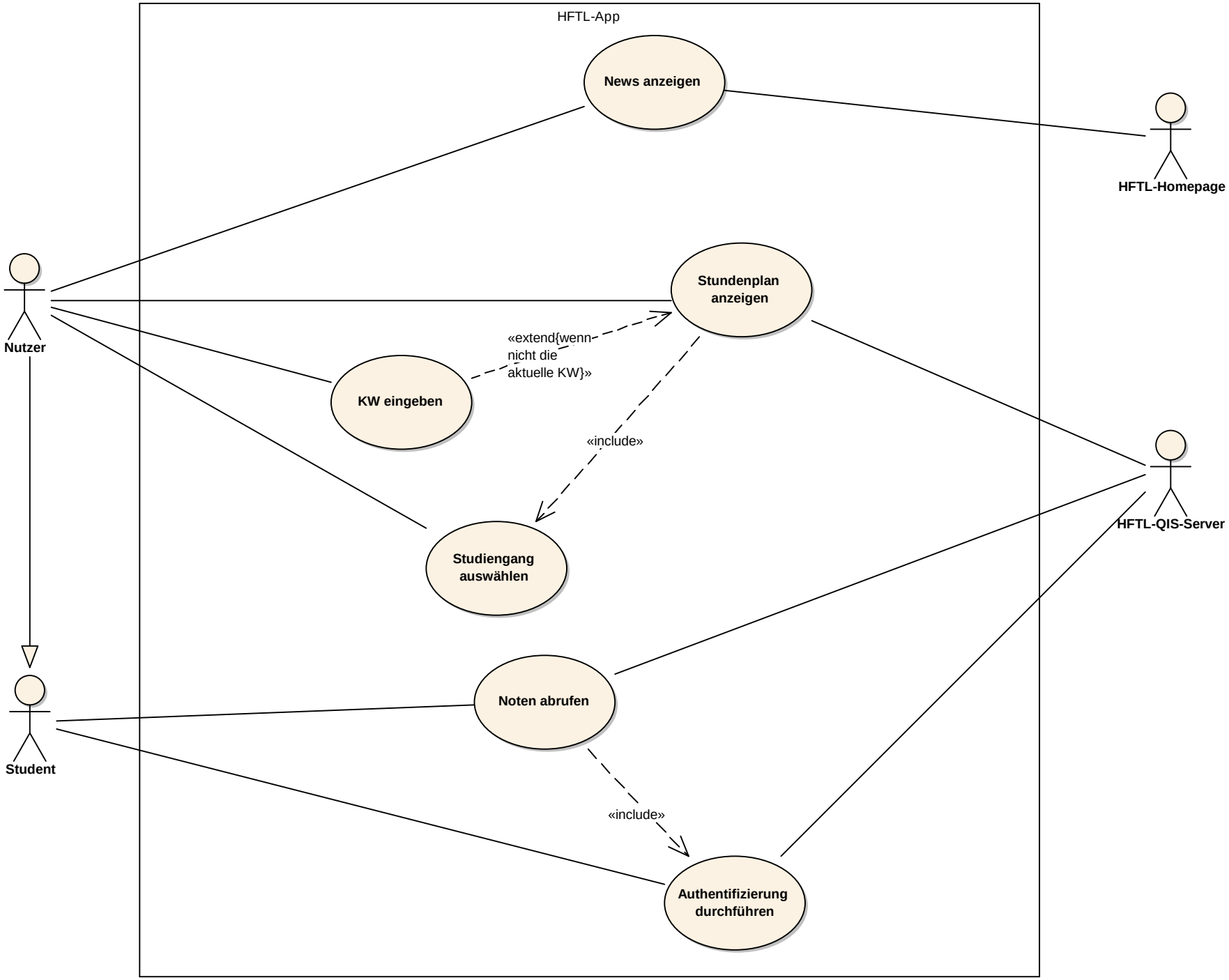

NewsFragment erzeugen
Fragment der Aktivität zuordnen
Daten initialisieren
View erzeugen
View wiederherstellen
Fragment wird augerufen
mögliche Aktionen
Fragment soll zerstört werden
View zerstören
Fragment zerstören
Fragment soll gestoppt werden
Fragment stoppen
Netzwerkstatus abrufen
nein
ja
gespeicherte Instanz vorhanden?
Instanz wiederherstellen
Instanz sichern
mögliche Aktionen
Daten aktualisieren
Knopf betätigt
Fragment pausieren
Fragment kommt in den Hintergrund
mögliche Aktionen
Fragment kommt in den Vordergrund
Verbindung vorhanden?
ja
HTML-Daten auslesen
Ladebalken anzeigen
Fragment anzeigen
nein
Auslesen erfolgreich?
ja
Arrays mit den Newselementen erzeugen (Kopf, Datum, Inhalt)
nein
Knopf zum Aktualisieren initialisieren
Textformatierungen anwenden
Newsliste erstellen
Fehlermeldung anzeigen
Fragment starten

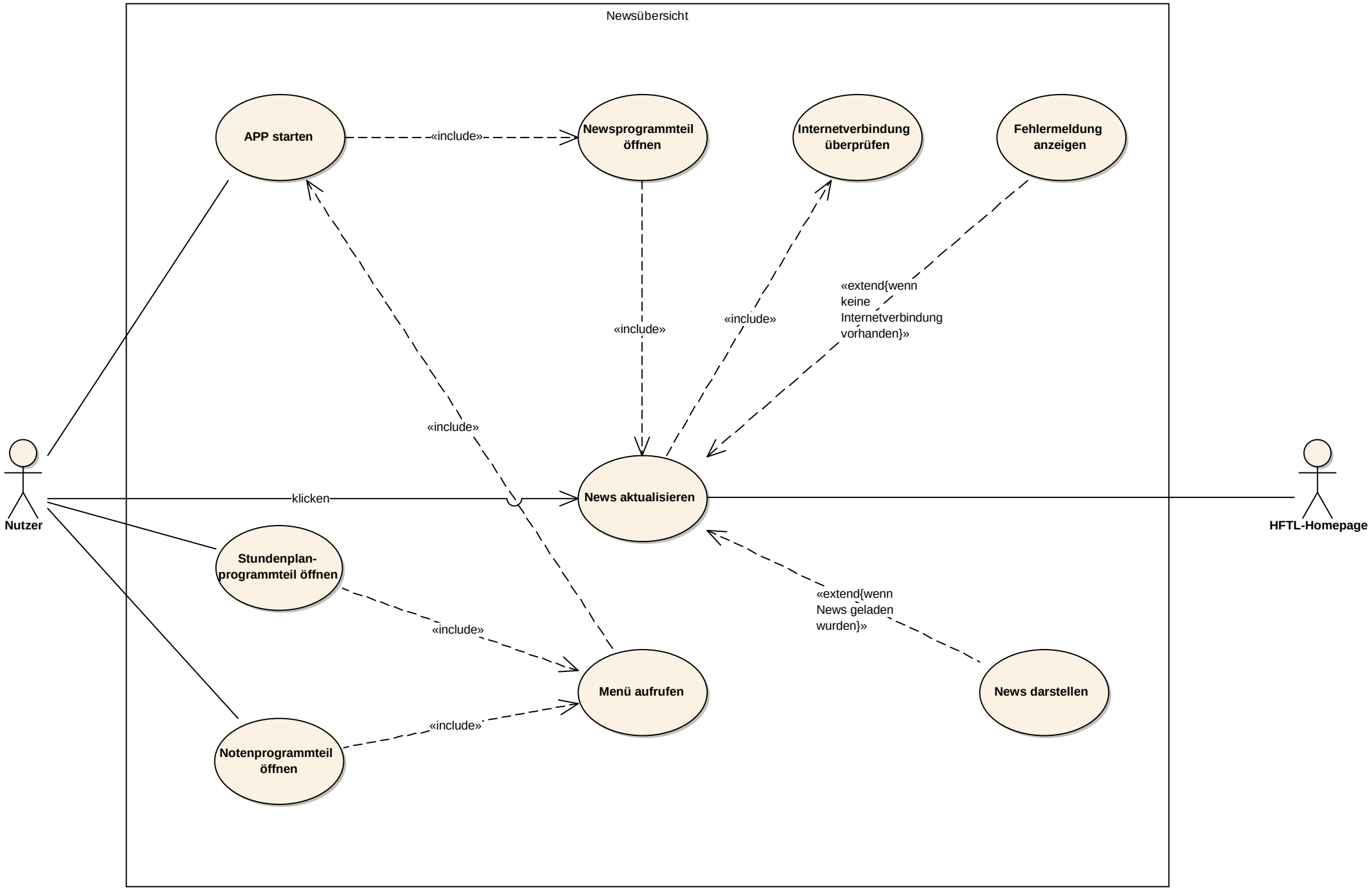

News
Stundenplan
Noten
StundenplanFragment
NewsFragment
NotenFragment
Fragment
+ onAttach() :void
+ onCreate() :void
+ onCreateView() :void
+ onActivityCreated() :void
+ onSaveInstanceState(Bundle) :void
+ onStart() :void
+ onResume() :void
+ onPause() :void
+ onStop() :void
+ onDestroyView() :void
+ onDestroy() :void
+ onDetach() :void
ActionBarActivity
+ onCreate() :void
+ onNavigationDrawerItemSelected(int*) :void
+ onStart() :void
+ onRestoreInstanceState() :void
+ onResume() :void
+ onCreateOptionsMenu(Menu*) :boolean
+ onSaveInstanceState() :void
+ onOptionsItemSelected(MenuItem*) :boolean
+ onPause() :void
+ onRestart() :void
+ onStop() :void
+ onDestroy() :void
NavigationDrawerFragment
- STATE_SELECTED_POSITION :String = selected_naviga...
- PREF_USER_LEARNED_DRAWER :String = navigation_draw...
- mCallbacks :NavigationDrawerCallbacks
- mDrawerToggle :ActionBarDrawerToggle
- mDrawerLayout :DrawerLayout
- mDrawerListView :ListView
- mFragmentContainerView :View
- mCurrentSelectedPosition :int = 0
- mFromSavedInstanceState :boolean
- mUserLearnedDrawer :boolean
+ NavigationDrawerFragment()
+ onActivityCreated(Bundle) :void
+ onCreate(Bundle) :void
+ onCreateView(Bundle, ViewGroup*, LayoutInflater) :View
+ isDrawerOpen() :boolean
+ setUp(DrawerLayout, int*) :void
- selectItem(int*) :void
+ onAttach(Activity) :void
+ onDetach() :void
+ onSaveInstanceState(Bundle) :void
+ onConfigurationChanged(Configuration) :void
+ onCreateOptionsMenu(MenuInflater, Menu*) :void
+ onOptionsItemSelected(MenuItem*) :boolean
- getActionBar() :ActionBar
NewsActivity (Main Activity)
~ Context :context
- mNavigationDrawerFragment :NavigationDrawerFragment
~ intent :Intent
- mTitle :CharSequence
onCreate(Bundle) :void
+ onNavigationDrawerItemSelected(int*) :void
+ onStart() :void
+ onSectionAttached(int) :void
+ onResume() :void
+ restoreActionBar() :void
+ onCreateOptionsMenu(Menu*) :boolean
+ onOptionsItemSelected(MenuItem*) :boolean
+ onPause() :void
+ onRestart() :void
# onSaveInstanceState(Bundle) :void
+ onStop() :void
+ onDestroy() :void
«interface»
NavigationDrawerCallbacks
+ onNavigationDrawerItemSelected() :void

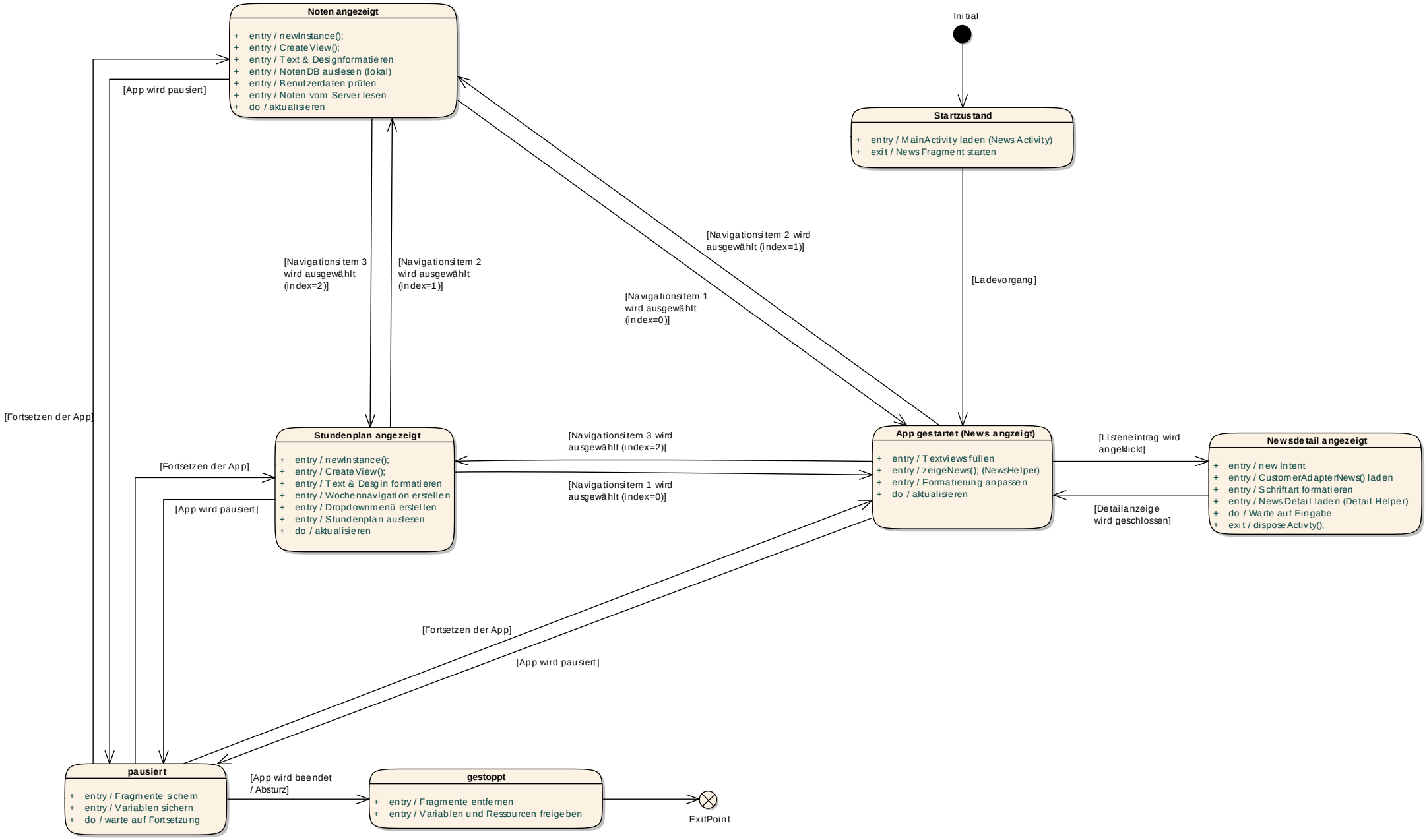

ListFragment
+ onAttach() :void
+ onCreate() :void
+ onCreateView() :void
+ onActivityCreated() :void
+ onListItemClick(long, int, View, ListView) :void
+ onAttach(Activity) :void
+ onStart() :void
+ onResume() :void
+ onPause() :void
+ onSaveInstanceState(Bundle) :void
+ onStop() :void
+ onDestroyView() :void
+ onDestroy() :void
+ onDetach() :void
ActionBarActivity
+ onCreate() :void
+ onNavigationDrawerItemSelected(int*) :void
+ onStart() :void
+ onRestoreInstanceState() :void
+ onResume() :void
+ onCreateOptionsMenu(Menu*) :boolean
+ onSaveInstanceState() :void
+ onOptionsItemSelected(MenuItem*) :boolean
+ onPause() :void
+ onRestart() :void
+ onStop() :void
+ onDestroy() :void
NewsFragment
- ARG_SECTION_NUMBER :String = "section_number"
~ arrayAdapter :ArrayAdapter<String>
+ ARRAYSPEICHER :String = "Array"
~ button :Button
~ intent :Intent
~ newsResolver :NewsResolver
~ stringArray[] :String
+ TERMINDETAIL :String = "Details"
~ headlineList[] :String
- dateList[] :String
- contentList[] :String
+ isOnline(Context) :boolean
+ newInstance(int*) :NewsFragment
+ NewsFragment()
+ onActivityCreated(Bundle) :void
+ onAttach(Activity) :void
+ onCreate() :void
+ onCreateView(LayoutInflater, ViewGroup*, Bundle) :View
+ onDestroy() :void
+ onDestroyView() :void
+ onDetach() :void
+ onListItemClick(long, int*, View, ListView) :void
+ onPause() :void
+ onResume() :void
+ onSaveInstanceState(Bundle) :void
+ onStart() :void
+ onStop() :void
- zeigeNews() :void
NewsResolver
~ doc :Document
~ termine :ArrayList<HftlEvent>
+ NewsResolver(String*)
- getHtmlAsDoc(String*) :boolean
- getHftlEvent() :void
+ getUrlAsString(int*) :String
+ getDetailsStringArray() :String[]
+ writeNewsListDate() :String[]
+ writeNewsListHeadline() :String[]
+ writeNewsListContent() :String[]
NewsClickedActivity
~ tv1 :TextView
~ tv2 :TextView
~ tv3 :TextView
~ tv4 :TextView
~ ladebalken :ProgressDialog
~ url :String
# onCreate(savedInstanceState) :void
+ onStart() :void
+ onRestoreInstanceState() :void
+ onResume() :void
+ onSaveInstanceState() :void
+ onPause() :void
+ onRestart() :void
+ onStop() :void
+ onDestroy() :void
BaseAdapter
+ getCount() :int
+ getItemId(int) :int
+ getItem(int) :Object
+ getView(ViewGroup, View, int) :View
NewsHelper
doInBackground(String) :Long
onPostExecute(aLong) :void
CustomAdapterNews
~ date[] :String
~ headline[] :String
+ content[] :String
~ context :Context
- inflater :LayoutInflater = null
+ CustomAdapterNews(String[], String[], String[], Activity) :class
+ Holder() :class
+ getCount() :int
+ getItemId(int) :int
+ getItem(int) :Object
+ getView(ViewGroup, View, int) :View
AsyncTask
+ doInBackground() :void
+ onPostExecute() :void
+ execute() :void
DetailHelper
~ s :String[]
# onPreExecute() :void
# doInBackground(String...) :Long
# onPostExecute(Long) :void

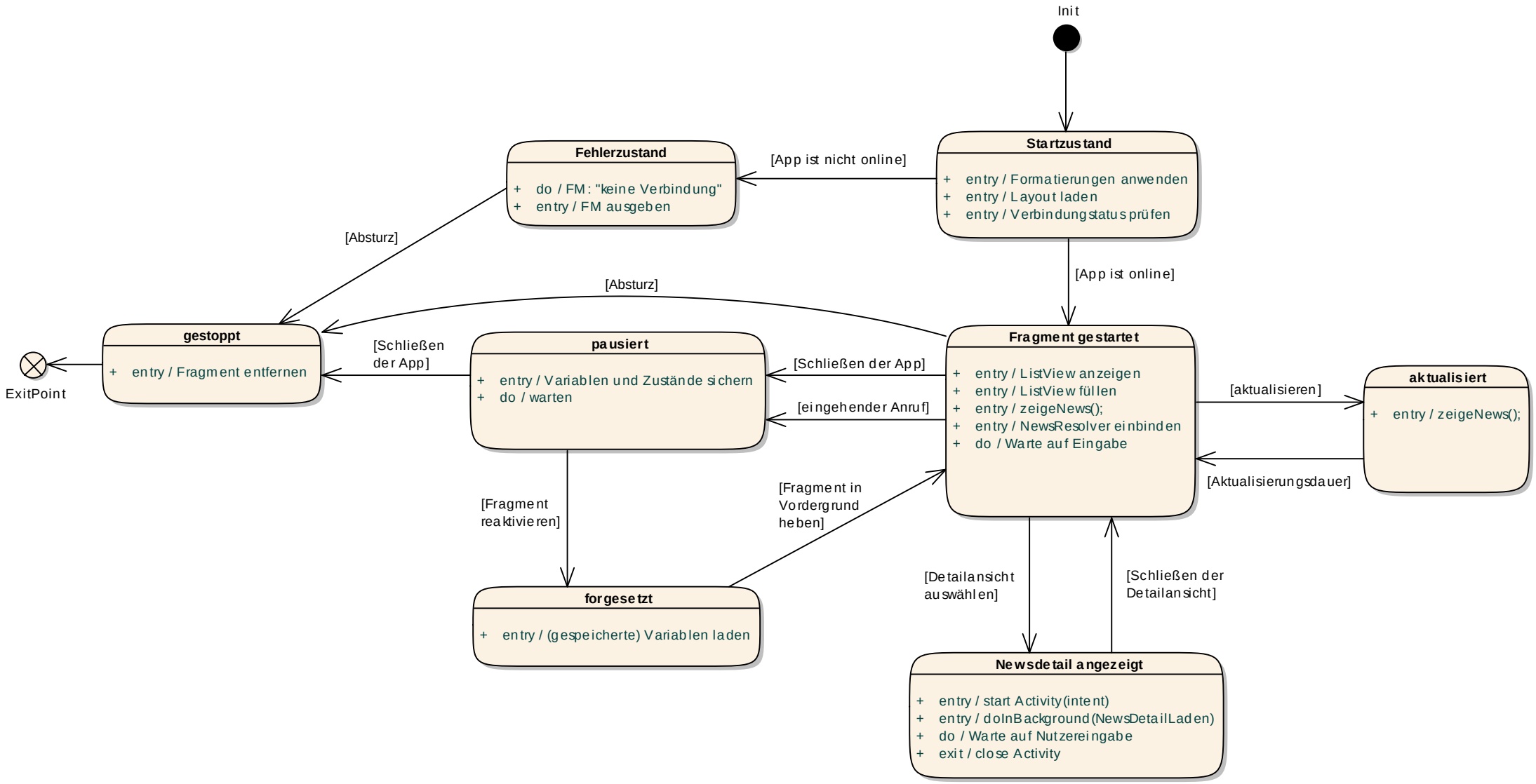

User
NewsFragment
NewsHelper
NewsResolver
HFTL-Server
betätigenButton()
zeigeNews()
alt Ist eine Internetverbindung vorhanden?
[nein]
zeigeKeine-Verbindung-Nachricht()
bestätigeNachricht()
[ja]
execute()
writeNewsListDate() :String[]
writeNewsListHeadline() :String[]
writeNewsListContent() :String[]
abrufen+parsenHTMLCode()
alt War der Zugriff erfolgreich?
null()
[nein]
Exception()
Exception()
zeigeNetzwerkfehler-Nachricht()
bestätigeNachricht()
[ja]
übertrageDaten()
übertrageDaten()
onPostExecute(aLong)
übertrageDaten()
zeigeNewsliste()
Ende
NewsResolver
Ende
NewsHelper


User
NewsFragment
NewsClickedActivity
DetailHelper
NewsResolver
HFTL-Server
onListItemClick(long, int*, View, ListView)
startActivity(Intent)
execute()
getDetailsStringArray() :String[]
abrufen+parsenHTMLCode()
alt War der Zugriff erfolgreich?
[Nein]
null()
Exception()
anzeigenFehlermeldung()
bestätigenFehlermeldung()
return()
[Ja]
übermittelnDaten()
übergabeDaten()
run()
anzeigenNews()
onStop()
Ende NewsResolver
Ende DetailHelper
Ende NewsClickedActivity


HFTL-App
News anzeigen
HFTL-Homepage
Nutzer
Stundenplan anzeigen
«extend{wenn nicht die aktuelle KW}»
KW eingeben
«include»
HFTL-QIS-Server
Studiengang auswählen
Noten abrufen
Student
«include»
Authentifizierung durchführen


Newsübersicht
APP starten
«include»
Newsprogrammteil öffnen
Internetverbindung überprüfen
Fehlermeldung anzeigen
«include»
«include»
«extend{wenn keine Internetverbindung vorhanden}»
«include»
Nutzer
klicken
News aktualisieren
HFTL-Homepage
Stundenplan-programmteil öffnen
«include»
«extend{wenn News geladen wurden}»
Menü aufrufen
News darstellen
Notenprogrammteil öffnen
«include»


Noten angezeigt
+ entry / newInstance();
+ entry / CreateView();
+ entry / Text & Designformatieren
+ entry / NotenDB auslesen (lokal)
+ entry / Benutzerdaten prüfen
+ entry / Noten vom Server lesen
+ do / aktualisieren
Initial
Startzustand
+ entry / MainActivity laden (News Activity)
+ exit / News Fragment starten
[App wird pausiert]
[Navigationsitem 3 wird ausgewählt (index=2)]
[Navigationsitem 2 wird ausgewählt (index=1)]
[Navigationsitem 2 wird ausgewählt (index=1)]
[Navigationsitem 1 wird ausgewählt (index=0)]
[Ladevorgang]
[Fortsetzen der App]
Stundenplan angezeigt
+ entry / newInstance();
+ entry / CreateView();
+ entry / Text & Desgin formatieren
+ entry / Wochennavigation erstellen
+ entry / Dropdownmenü erstellen
+ entry / Stundenplan auslesen
+ do / aktualisieren
[Navigationsitem 3 wird ausgewählt (index=2)]
[Navigationsitem 1 wird ausgewählt (index=0)]
App gestartet (News angzeigt)
+ entry / Textviews füllen
+ entry / zeigeNews(); (NewsHelper)
+ entry / Formatierung anpassen
+ do / aktualisieren
[Listeneintrag wird angeklickt]
[Detailanzeige wird geschlossen]
Newsdetail angezeigt
+ entry / new Intent
+ entry / CustomerAdapterNews() laden
+ entry / Schriftart formatieren
+ entry / News Detail laden (Detail Helper)
+ do / Warte auf Eingabe
+ exit / disposeActivty();
[Fortsetzen der App]
[App wird pausiert]
[Fortsetzen der App]
[App wird pausiert]
pausiert
+ entry / Fragmente sichern
+ entry / Variablen sichern
+ do / warte auf Fortsetzung
[App wird beendet / Absturz]
gestoppt
+ entry / Fragmente entfernen
+ entry / Variablen und Ressourcen freigeben
ExitPoint


Init
Startzustand
+ entry / Formatierungen anwenden
+ entry / Layout laden
+ entry / Verbindungstatus prüfen
[App ist nicht online]
Fehlerzustand
+ do / FM: "keine Verbindung"
+ entry / FM ausgeben
[Absturz]
[App ist online]
[Absturz]
gestoppt
+ entry / Fragment entfernen
ExitPoint
[Schließen der App]
pausiert
+ entry / Variablen und Zustände sichern
+ do / warten
[Schließen der App]
[eingehender Anruf]
Fragment gestartet
+ entry / ListView anzeigen
+ entry / ListView füllen
+ entry / zeigeNews();
+ entry / NewsResolver einbinden
+ do / Warte auf Eingabe
[aktualisieren]
aktualisiert
+ entry / zeigeNews();
[Aktualisierungsdauer]
[Fragment reaktivieren]
[Fragment in Vordergrund heben]
forgesetzt
+ entry / (gespeicherte) Variablen laden
[Detailansicht auswählen]
[Schließen der Detailansicht]
Newsdetail angezeigt
+ entry / start Activity(intent)
+ entry / doInBackground(NewsDetailLaden)
+ do / Warte auf Nutzereingabe
+ exit / close Activity