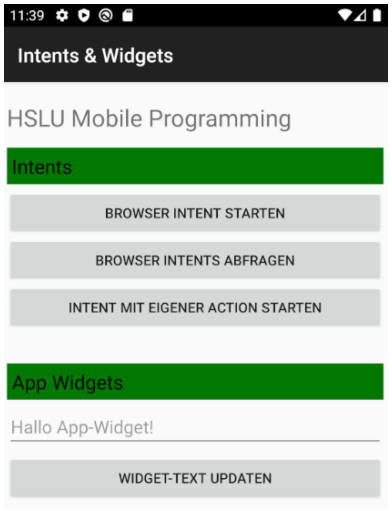


Übung 6: Intent-Filter und App-Widget

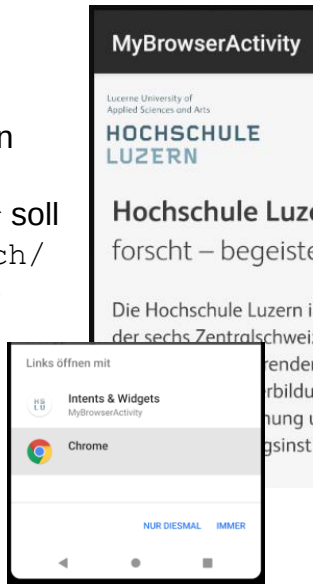
In dieser Übung geht es um implizite Intents, Intent-Filter und Intent-Auflösung. Sie werden dazu eine eigene Activity erstellen, die Browser-Intents empfangen kann. Weiter werden Sie beim System alle Komponenten abfragen, welche Browser-Intents empfangen können und diese auflisten, sowie Ihre App für eine eigene Intent-Action registrieren. Zusätzlich werden Sie ein eigenes App-Widget erstellen und dabei sehen, wie dieses beim System registriert ist und in welcher Form dafür Layout, Infos und Logik spezifiziert werden.

1. Neue App: Intents & Widgets

Erstellen Sie ein neues Android-Applikationsprojekt, wie gehabt mit einer "Empty Activity". Diese Übung besteht aus verschiedenen Teilaufgaben. Die `MainActivity` dieses Projekts soll am Schluss ungefähr so aussehen wie im Screenshot rechts oben. Verwenden Sie dazu eine `ScrollView` mit einem `Constraint-` oder `LinearLayout`, siehe Folien bzw. Übung 2 für Details dazu.

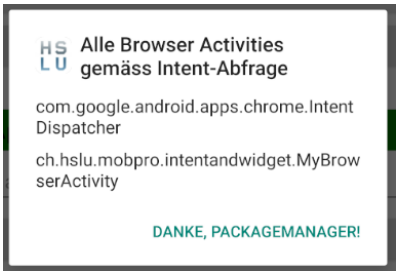
2. Browser Intent starten

Erstellen Sie eine neue Activity `MyBrowserActivity`, welche ganz banal und ohne irgendwelchen Komfort (wie etwa URL-Anzeige, Menü, Reload-Knopf, Bookmarks, ...) eine Webseite in einer `WebView` anzeigen können soll, siehe Screenshot rechts. Beim Klick auf „Browser Intent starten“ auf der `MainActivity` soll ein Browser-Intent für die URL `https://www.hslu.ch/de-ch/` abgesetzt werden. Deklarieren Sie im Android-Manifest für Ihre `MyBrowserActivity` einen passenden Intent-Filter, um Browser-Intents empfangen zu können. Wenn Sie dies korrekt getan haben, sollten Sie dann bei Knopfdruck vom System einen Dialog analog zu dem im Screenshot rechts gezeigten erhalten, wo Sie auswählen können, wie Sie diese Browser-Aktion gerne handhaben möchten. (Verwenden Sie dazu einen Emulator mit Android 11 (API Level <= 30), weil es da Anpassungen gab mit Android 12 (API Level 32) und dies jetzt nicht mehr so einfach ist)



3. Browser Intents abfragen und darstellen

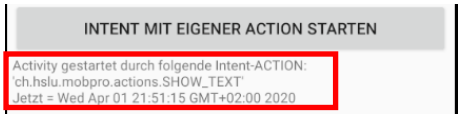
Erweitern Sie Ihre `MainActivity` um einen weiteren Knopf „Browser Intents abfragen“. Wird dieser gedrückt, sollen alle Activities angezeigt werden, welche einen Browser-Intent empfangen können. Verwenden Sie dazu wie in der Vorlesung gezeigt die System-Klasse `PackageManager` und stellen Sie die Namen aller gefundenen Activities (über das Attribut `ResolveInfo.activityInfo.name`) in einem Dialog dar. Das sollte in etwa so aussehen wie im



Screenshot rechts. Vergessen Sie nicht (wie in der Vorlesung gezeigt) die dazu notwendige Deklaration im Manifest vorzunehmen.

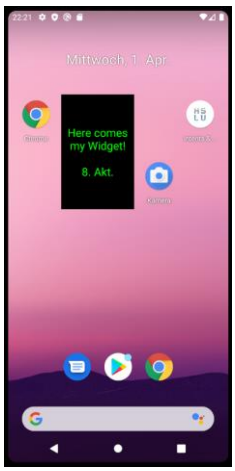
4. Intent mit eigener Action, inkl. Intent-Start aus anderer App

Erweitern Sie die `MainActivity` Ihrer App so, dass diese auf implizite Intents mit der durch uns definierten Intent-Action `ch.hslu.mobpro.actions.SHOW_TEXT` startet. Deklarieren Sie den dazu notwenigen Intent-Filter. Und stellen Sie sicher, dass auf der `MainActivity`, falls diese durch unsere eigene Intent-Action gestartet wurde, der Inhalt des Parameters `text` dargestellt wird, siehe den rot markierten Text im Screenshot rechts. Starten Sie dann einen entsprechenden impliziten `Intent` mit dieser Action und geben Sie diesem Intent einen Parameter (`Intent.putExtra(...)`) mit, in welchem Sie die Intent-Action sowie die aktuelle Uhrzeit mitgeben. Ergänzen Sie zum Testen dieser Funktionalität Ihre `MainActivity` um einen Knopf „Intent mit eigner Action starten“. Testen Sie diese Funktionalität ebenfalls auf einer App, d.h. starten Sie aus einer anderen App (z.B. aus Ihrer UI-Demo-App von SW03) einen äquivalenten Intent zu oben (d.h. mit unserer eigenen `SHOW_TEXT`-Action und inkl. `text`-Parameter). Damit haben Sie selbständig ein Beispiel für lose Kopplung zwischen zwei Apps implementiert und getestet. ☺



5. Ein eigenes App-Widget

Ergänzen Sie Ihre App um ein App Widget. Dieses App Widget soll einen Text programmatisch anzeigen. Erzeugen und implementieren Sie, wie in der Vorlesung gezeigt, eine eigene Klasse `MyAppWidgetProvider`, welche von `AppWidgetProvider` erbt, dazu eine XML-Layout-Datei `my_app_widget_provider.xml` und eine XML-Datei `xml/my_app_widget_provider_info.xml`. Registrieren Sie im Manifest Ihren `MyAppWidgetProvider` als Receiver-Komponente. Nachdem Sie Ihr App Widget implementiert haben, sollte dieses unter WIDGETS („langes Drücken“ auf Home-Screen oder auf App-Icon auf Home-Screen) auf Ihrem Android-System erscheinen. Ziehen Sie Ihr Widget auf den Home-Screen, siehe Screenshot unten rechts. Und wenn Sie es dort hinterlegt haben, bewegen Sie es danach an eine neue Stelle, wiederum durch „langes Drücken“ und dann „Ziehen“.



6. Widget-Daten on-demand aktualisieren inkl. Zähler

Ihr Widget soll nun einen konfigurierbaren Text erhalten, der über die App gesetzt werden kann. Ergänzen Sie dazu die `MainActivity` um einen `EditText`, um einen Text erfassen zu können und um einen `Button`, um diesen in den `SharedPreferences` zu speichern und Ihr Widget zu aktualieren. Passen Sie ihr Widget so an, dass in der `onUpdate()`-Methode dieser Text ausgelesen und angezeigt wird (anstelle eines statischen Texts). In derselben Methode soll der dargestellte Text ebenfalls um die Anzahl der bisher durchgeführten Updates angegeben werden, siehe Screenshot rechts unten. Verwenden Sie dafür eine statische (d.h. im `companion object`) Zähler-Variable `updateCount`. Da Widgets in der Regel maximal alle 30 Minuten aktualsiert werden, implementieren Sie als letztes noch einen Broadcast, der beim Updaten des Widget-Texts in der App ein Update des Widgets triggert (siehe Folien für ein Beispiel).