App Quest 2019

Memory

Am Treasure Hunt wird auf dem Gelände der HSR eine Menge an Bild-Paaren verteilt sein. Hier ein Beispiel eines Paars mit dem Android als Motiv:

Ein Bild-Paar hat immer dasselbe Motiv, aber die Bilder unterscheiden sich im enthaltenen QR-Code (hier zum Beispiel steht einmal „Hochschule“ und „Rapperswil“ drin, unser Server muss ja wissen, ob ihr auch wirklich

beide Bilder gefunden habt). Die App muss also in der Lage sein, Fotos aufzunehmen, den QR-Code auszulesen (keine Angst, wir verwenden hierfür eine Library) und diese irgendwie paarweise abzulegen. Zum Beispiel in einer Liste oder in einem Grid. Am Treasure Hunt wird die Aufgabe dann sein, möglichst viele solcher Bildpaare zu finden.

Android

Unter Android verwenden wir für das Aufnehmen des Fotos und gleichzeitige Auslesen des Barcodes die Library zxing-android-embedded. Diese können wir im build.gradle File bei den anderen Dependencies eintragen:

```
dependencies {
    ...
    implementation 'com.journeyapps:zxing-android-embedded:4.0.2'
}
```

build.gradle hosted with ❤ by **GitHub** **view raw**

Um ein Foto aufzunehmen und den Code auszulesen können wir die folgenden beiden Methoden in unserer Activity verwenden:

```
public void takeQrCodePicture() {
    IntentIntegrator integrator = new IntentIntegrator(this);
    integrator.setCaptureActivity(MyCaptureActivity.class);
    integrator.setDesiredBarcodeFormats(IntentIntegrator.QR_CODE_TYPES);
    integrator.setOrientationLocked(false);
    integrator.addExtra(Intents.Scan.BARCODE_IMAGE_ENABLED, true);
    integrator.initiateScan();
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == IntentIntegrator.REQUEST_CODE
        && resultCode == RESULT_OK) {

        Bundle extras = data.getExtras();
        String path = extras.getString(
            Intents.Scan.RESULT_BARCODE_IMAGE_PATH);

        // Ein Bitmap zur Darstellung erhalten wir so:
        // Bitmap bmp = BitmapFactory.decodeFile(path)

        String code = extras.getString(
            Intents.Scan.RESULT);
```

```
24        }
25    }
```

MyActivity.java hosted with ❤ by **GitHub** **view raw**

So erhalten wir zwei Strings: einen mit dem Pfad zum Foto, den anderen mit dem Inhaltes des QR-Codes. Die Klasse MyCaptureActivity muss angegeben werden, kann aber völlig leer gelassen werden:

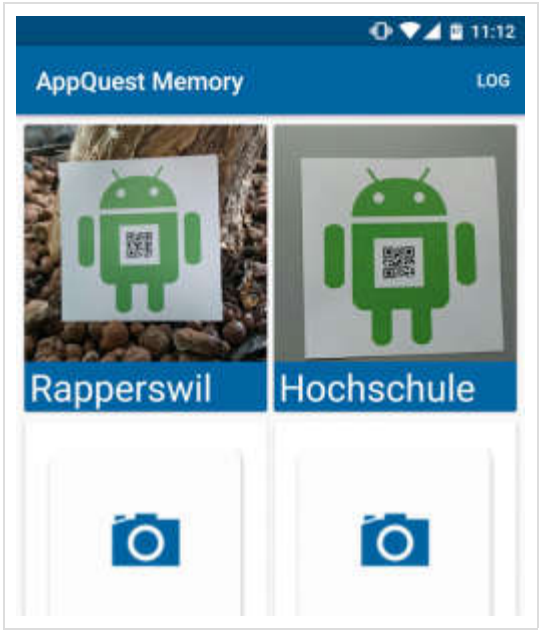
```
1  import com.journeyapps.barcodescanner.CaptureActivity;
2
3  public class MyCaptureActivity extends CaptureActivity { }
```

MyCaptureActivity.java hosted with ❤ by **GitHub** **view raw**

In unserer Implementierung der App legen wir die Fotos in einer RecyclerView ab, welche zwei Spalten hat:

```
RecyclerView rv = (RecyclerView)findViewById(R.id.recyclerView);
GridLayoutManager gridLayoutManager = new GridLayoutManager(this /* the activity */
rv.setLayoutManager(gridLayoutManager);
```

Hier gibt es ein weiteres gutes Tutorial das zeigt wie eine CardView mit Bild und Button gemacht wird (für unsere App könnte man einfach zwei Bilder und zwei Buttons für die Aufnahme des Fotos einsetzen). Wie gesagt sind hier beliebige Lösungen möglich. Am Ende müssen im Logbuch die zusammengehörenden Wörter eingetragen werden. Wie genau, ist in der Logbuch-Anleitung ersichtlich. Hier ein Screenshot unserer Implementation: ein Eintrag beinhaltet initial nur einen Button um ein Foto aufzunehmen (unten) und wir dann durch das Foto ersetzt:



iOS

Damit die erfassten Bilder und Codes beim Beenden der App nicht verloren gehen, müssen wir diese Daten

persistieren. Auf iOS kann man dazu das Core Data Framework verwenden. Wir haben eine kleine Projektvorlage vorbereitet, die bereits ein passendes Core Data Datenmodell enthält. Das Datenmodell besteht aus einer einzelnen Klasse ‚MemoryPair', welche Properties für die zwei Bilder und die zwei Codes eines Lösungspaars hat. Ausserdem beinhaltet die Vorlage auch bereits den SolutionLogger. Die Vorlage könnt ihr hier herunterladen.

Die folgenden Code-Beispiele helfen euch bei der Erfassung der QR-Codes und bei der Verwendung von Core Data.

Mit der Hilfsklasse SolutionLogger kann man einen QR-Code erfassen und gleichzeitig ein Foto davon machen:

```
@IBAction func captureImageWithQRCode(_ sender: UIButton) {
    let solutionLogger = SolutionLogger(viewController: self)
    solutionLogger.scanQRCodeAndCaptureImage { (qrCode: String, image: UIImage)  in
        // use qrCode and image
    }
}
```

Memory1.swift hosted with ❤ by **GitHub** **view raw**

Mit folgendem Code kann man alle MemoryPairs auslesen, die im Core Data Container gespeichert sind:

```
let fetchRequest: NSFetchRequest<MemoryPair> = MemoryPair.fetchRequest()
fetchRequest.sortDescriptors = [NSSortDescriptor(key: "creationDate", ascending: true)]

do {
  let pairs: [MemoryPair] = try coreDataStack.context.fetch(fetchRequest)
  // display the memory pairs
} catch {
  print(error)
}
```

Memory2.swift hosted with ❤ by **GitHub** **view raw**

In diesem Code-Beispiel wird ein neues MemoryPair erstellt:

```
let memoryPair = MemoryPair(context: coreDataStack.context)
memoryPair.creationDate = Date()
memoryPair.image1 = image1
memoryPair.text1 = code1
memoryPair.image2 = image2
memoryPair.text2 = code2
coreDataStack.saveContext()
```

Memory3.swift hosted with ❤ by **GitHub** **view raw**

Mit dem folgenden Code wird ein existierendes MemoryPair gelöscht:

```
1  coreDataStack.context.delete(memoryPair)
2  coreDataStack.saveContext()
```

Memory4.swift hosted with ❤ by **GitHub** **view raw**

Um die MemoryPairs im UI anzuzeigen, empfehlen wir euch die Verwendung einer UITableView.