User manual

OCR mode

On the app startup the user is presented with the camera view in the OCR mode.

At the top a user has three buttons: **(1) QR mode, (2) Manual mode and (3) Database update.**

Pressing (1) informs the camera that it should look for QR codes and hides the bottom shutter button; pressing (2) opens the manual entry screen (same as the manual fix screen in figure 2, without the loaded image); pressing (3) performs a forced update of the database.

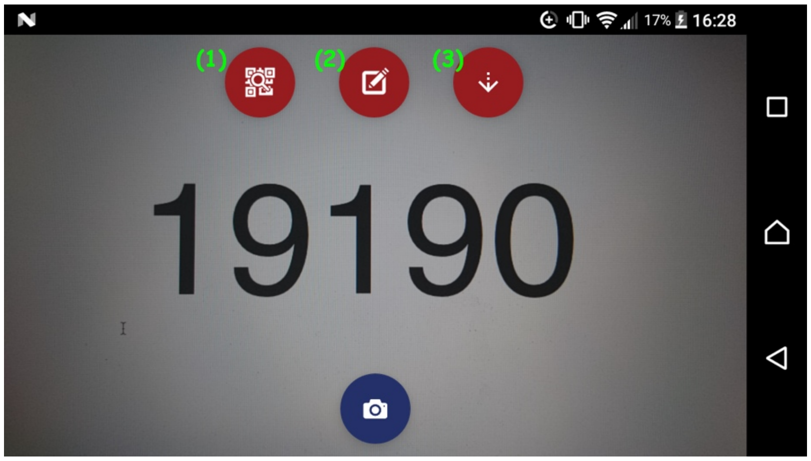
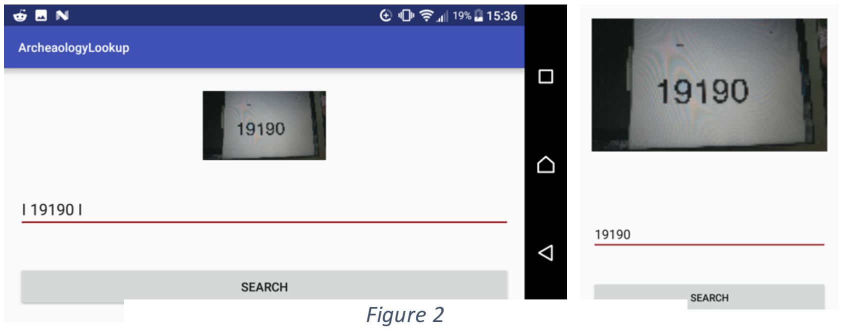


Figure 1

At the bottom of the screen there is a **camera shutter button** that, when pressed, analyzes the image and extracts the item ID; then, it opens the manual fix screen so the input can be fixed:



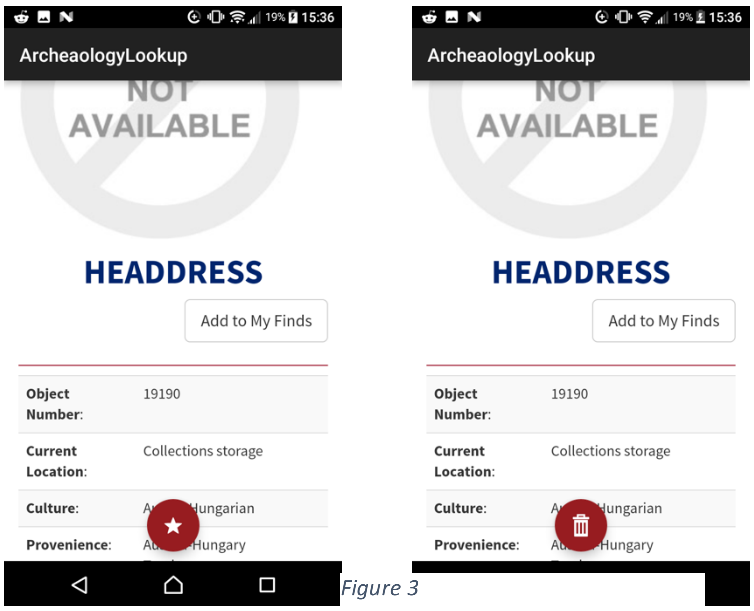
I 19190 I

19190

Figure 2

After the fixes to the OCR have been made by the user, they can perform a search on that ID, by pressing the search button.

That will open the object webpage, and enable the user to bookmark the content (by pressing on the star). If the object is already bookmarked the user is able to delete it from the bookmarks (by pressing on the trashcan):



Object Number:	19190
Current Location:	Collections storage
Culture:	A[illegible]Hungarian
Provenience:	A[illegible]Hungary

Figure 3

Going back to the main camera view (by pressing back button twice) and **swiping to the right** we can access the bookmarks list, and verify that the bookmark has indeed been added. By clicking on a bookmark it will open the browser with that URL. Similarly, if we **swipe to the left** we can access history. By clicking on the trashcan icon, we can clear history.

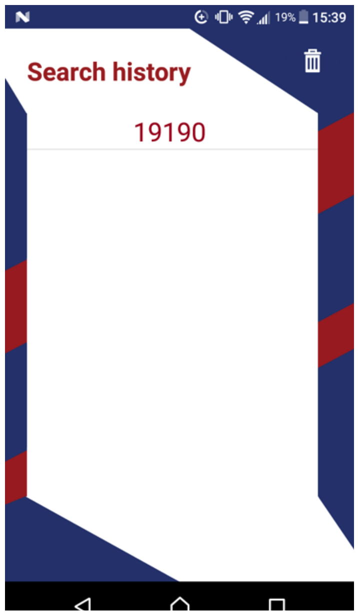


Figure 4

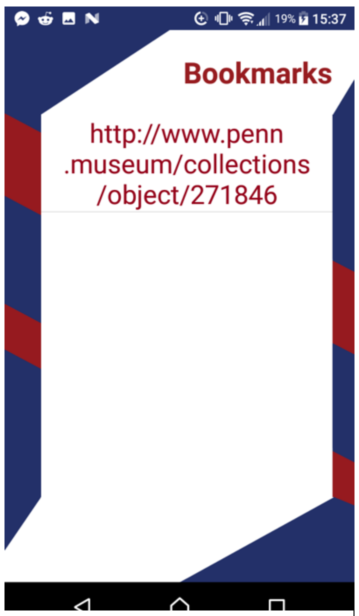
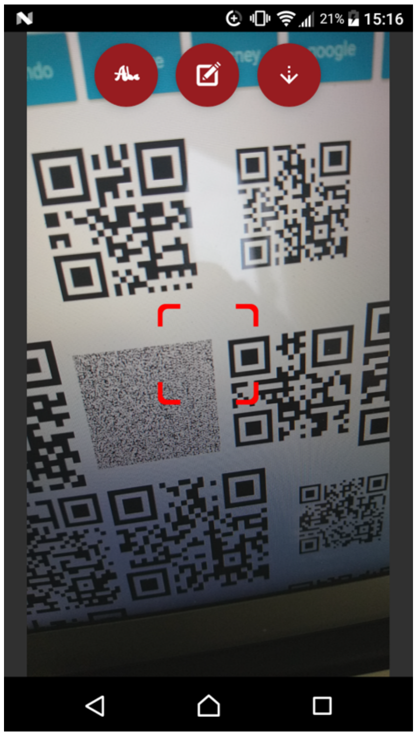


Figure 5

QR mode



If the QR mode has been selected, a red bracket (in which the user needs to place a QR code) will become visible. Once the code has been detected, web page with the corresponding object will be automatically loaded.

Figure 6

Technical manual

MainActivity

This activity is used as a opening screen. Here, the database is updated if necessary, Tesseract/OCR training data is loaded if necessary, Databases are initialized, and permissions are requested. After all of this is completed, the CamerUIActivity.class is called. The Main Activity, when not requesting permissions, should present itself as a blank screen.

CameraUiActivity

This activity displays the OCR view and serves as the main view of the app (*figure 1*). This class can call the HistoryActivity and FavoriteActivity by swiping left or right (*figures 4/5*), and can call ManualSearch or use QR (*figure 6*) to search the downloaded database and view the result.

<u>HISTORY/FAVORITE Activities:</u>

FavoriteActivity: Displays the favorite activity (*figure 5*). This class uses HistoryHelper to retrieve items from the history database and display them.
HistoryActivity: Displays the history activity (*figure 4*). This class uses HistoryHelper to retrieve items from the history database and display them.
HistoryHelper: This class creates and interacts with the history sql database and favorite sql database. The HistoryHelper has an insert function to insert new searches, a delete function to remove items, and a search function to retrieve information from the databases.

Database Schema (All database functions are located in the HistoryHelper.java class):
History Columns:
search: item ID being searched for
item: Object id returned
url: Url of object in Museum's database
name: Name of object
description: Description of object, ex: Small two-handed axe
provenience: Provenience of object, ex: Europe (uncertain) | United States (uncertain)
material: Material of object, ex: Wood
curatorial_section: Part of museum database located, ex: historic
Favorite Columns:
Same columns as History columns, but used to store favorites instead of history

<u>SEARCH Activities:</u>

ManualActivity: This activity is used to make manual searches and OCR picture searches (*figure 2*). It takes in a string based on whichever method of identification is used (OCR or manual), calls a retreiver to get the object's URL, then sends the display data to SearchActivity described below under VIEW Activities.
Retreiver: This is the abstract class that we use to implement modulare search and database functionality. New search classes can be built by implementing this retriever, then changing lines 62 and 64 in ManualLookup to incorporate your own implementation.
SqlRetreiver: This implementation of Retriever lets users access a SQL database.
WebRetreiver: This implementation of Retriever lets users access a web database.
LocalRetreiver: This implementation of Retriever lets users access the Penn museum's CSV archeology database. The CSV is downloaded from **http://www.penn.museum/collections/assets/data/all-csv-latest.zip.**

VIEW Activities:
SearchActivity: This class is responsible for performing artifact searches, setting the current artifact display strategy, and processing the database results in order to render them visible.
PennMuseumArtifactStrategy: An implementation of the Strategy interface, designed for the Penn Museum database artifacts. This Strategy takes in a bundle containing information about the artifact (e.g. material) and produces a WebView based on the object URL information (*figure 3*).
ExcavationDatabaseStrategy: Another implemenation of the Strategy interface, designed for an alternate database. Implemented mainly to prove the modular capabilities of the app.
Strategy: This interface specifies a method for the database administrator to input a custom display strategy, allowing for modularity of digital artifact storage. The main purpose of the Strategy is artifact view creation; given information about the artifact, this information is displayed in displayView() (most likely through the use of a WebView). In addition, the interface also specifies how the artifact should be saved as a favorite. An example implementation can be found in PennMuseumArtifactStrategy.

UPDATE DATABASE
updateDatabase - This is an abstract strategy that should be used in order to create new classes to retrieve different databases.
updateDatabaseMuseum.java

databaseUpdater - This class is used in order to check whether the database needs to be updated, and in order to instantiate the updating.

updateDatabaseMuseum - This class is used to asynchronously retrieve and unzip the database for usage in the rest of the program. On complete, the ui this was called from is toasted to.