

Android Application

The application was developed for android devices with an API of 28 and higher using [Android Studio](#). The project for the application is set up as described [here](#). The application makes use of the [Google Maps API](#) for which an [API key](#) is required.

REST configuration

The only part of the android project that differs from a standard project is the configuration of the REST API. For this the [Retrofit library](#) is used. The structure of the folders that belong to this section are given below. Folders and files are shown on the left, while a short explanation is given on the right if needed.

```
|- - app
|   |- - java
|   |   |- - com.example.senara
|   |   |   |- - models           class objects for data transfer
|   |   |   |- - remote
|   |   |   |   |- - APIUtils.java    defines the address of the backend server
|   |   |   |   |- - FileService.java  HTTP calls made by application
|   |   |   |   |- - RetrofitClient.java HTTP client
|   |   |   |- - ... rest of application functionality
```

Backend

The backend framework was developed using the [Django Web framework](#). To incorporate a REST architecture the [Django REST Framework](#) was added. All code for the backend is written using [Python 3.7](#). For the backend the following packages are used:

- [Django](#) – version 2.2.2 with the [Django REST Framework](#) enabled
- [OpenCV](#) – version 4.1.0
- [scikit-learn](#) – version 0.20.3
- [scikit-image](#) – version 0.15.0
- [numpy](#) – version 1.16.2
- [pandas](#) – version 0.24.1
- [imutils](#) – version 0.5.2
- [pyproj](#) – version 1.9.6

The folder structure for the most important elements of the backend is shown below:

```
|- - backend
|   |- - image upload           handles the image upload
```

- - images	<i>folder where uploaded images are stored</i>
- - processing	
- - image_processing	<i>image processing algorithm</i>
- - management	<i>admin functionality</i>
- - commands	<i>custom administrator commands</i>
- - waterMeterLocations.csv	
- - add_watermeters.py	<i>adds water meters to the database</i>
- - views.py	<i>backend functionality</i>
- - rest_api	<i>api-settings</i>

Database

The backend incorporates a [PostgreSQL](#) database, with the [PostGIS](#) database extender to handle geographical objects. The following python packages need to be installed for Django to function in combination with the database:

- [psycopg2](#)
- [GEOS](#)
- [PROJ.4](#)
- [GDAL](#)

Image Processing Algorithm

A separated python package containing the image processing algorithm implemented in the backend is added, including all intermediary results as well as the scripts used to generate the digit templates and the corresponding feature set, train the k-NN classifiers and the figures used for the result section of the thesis.

The following external python packages are used by the image processing package:

- [OpenCV – version 4.1.0](#)
- [scikit-learn – version 0.20.3](#)
- [scikit-image – version 0.15.0](#)
- [numpy – version 1.16.2](#)
- [pandas – version 0.24.1](#)
- [imutils – version 0.5.2](#)
- [pyproj – version 1.9.6](#)
- [openpyxl – version 2.6.3](#)
- [matplotlib – version 3.0.3](#)

Second Digit Template Set

Initially two methods were used to create the digit templates. The second method, not discussed in the thesis because of slightly disappointing results, focuses on a structured way to create the digit templates and goes as follows: At first two binary digit templates are selected for each digit class, with the template containing the whole number. As the digits pictured on the tally counters are all of the same font, the general shape of the unknown digit can be expected to be similar to the original font. For each of the whole digit templates a set of sub-templates is created, each template containing a possible occurrence of the digit in case it gets cut off from the left or right, top or bottom or a combination of both, for example, the top left of the digit. Within the package this method is referred to as the structured digit template set while the template set as discussed in the thesis is referred to as the random digit template set.

The file structure of the package is as follows:

- - image_processing	
- - data	
- - background	<i>background samples for random digit templates</i>
- - classifiers	<i>k-NN classifiers</i>
- - digits	<i>images of tally counter digits</i>
- - structured_templates	<i>structured digit templates</i>
- - knn_random_templates	<i>random templates training datasets</i>
- - knn_structured_templates	<i>structured templates training datasets</i>
- - result_labels	<i>class labels</i>
- - random_templates	<i>random digit templates</i>
- - templates	<i>base digit template set</i>
- - templates2	<i>base digit template set</i>
- - water_meters	<i>test images of water meters</i>
- - correct_results.csv	<i>correct results for each water meter image</i>
- - dial	<i>modules of the image processing algorithm</i>
- - digit_classification	
- - boundary.py	<i>boundary extraction</i>
- - classify.py	<i>k-NN classification</i>
- - fourier_descriptors.py	<i>Fourier descriptor calculation</i>
- - helpers	
- - adaptive_gamma_correction.py	
- - general_functions.py	
- - dial.py	<i>dial extraction</i>
- - pointer.py	<i>pointer reading</i>
- - pointer_red.py	<i>red pointer detection</i>
- - tally_counter.py	<i>tally counter reading</i>
- - zero.py	<i>zero point detection</i>
- - digit	<i>scripts used to create classifiers</i>
- - classifier	<i>k-NN classifier training</i>
- - dataset.py	<i>creation of training datasets</i>
- - train_knn_random.py	<i>random templates classifier training</i>

			<i>structured templates classifier training</i>
		- - train_knn_structured.py	<i>digit template creation</i>
	- -	digit_templates	<i>random digit templates</i>
		- - random_digit_templates.py	<i>structured digit templates</i>
		- - structured_digit_templates.py	
	- -	results	
		- - digit_bins	<i>binary images of each classified digit</i>
		- - digit_classification	<i>excel sheets and figures of classification results</i>
		- - image_processing	<i>intermediary results for image processing</i>
		- - make_results.py	<i>script used to generate excel sheets and figures</i>
	- -	helpers.py	<i>helper functions for main program</i>
	- -	main.py	<i>main program for image processing</i>

Running current architecture locally

To run the current set up locally, use the emulator provided by Android studio to simulate an android device with an API level of at least 28. Then start a development server with Django by opening the backend folder in a terminal and running the following command: "python manage.py runserver". This will start a local server to which the application on the simulated android device can connect.