1 Feedback Web Service

No changes.

```
1. Build rdf4j_server module
2. Run start_rdf4j_server.bat file
3. Open "localhost:8090/workbench/repositories" in the browser
4. Change RDF4J Server Url into "http://localhost:8090/rdf4j"
5. Run feedback_webservice module
```

2 Feedback Service App

Still requires feedback web service:

```
\begin{verbatim}
1. Find "feedback_web_service_url" keyword in strings.xml file in
   feedback-service-app project.
2. Replace the existing IP address with the IP address of local machine on which
   the feedback-web-service project is running.
3. Run project with virtual/physical device

strings.xml:
    <string name="feedback_web_service_url">http://192.168.0.102:8803/feedback</string>
```

Additionally specifies `location_web_service_url` for mode updates:

```
<string name="location_web_service_url">http://192.168.122.1:8903/location</string>
```

After logging in Every 5 seconds it sends the location JSON to the specified URL and expects a mode string as the reply.

Sample request body (originally without new formatting):

```
{
"altitude": 5,
"uid": "eiucOYz7DrVGjEbxsXfBksgjPUi2",
"bearing": 0,
"accuracy": 12.423,
"lon": -122.0841183,
"time": 1606818880649,
"lat": 37.4224,
"speed": 0,
"email": "a@a.com"
}
```

And the response is one of BICYCLE, BUS, CAR, TRAIN, WALK with status code 200.

3 Mode Detector Service

3.1 Compile

The project uses gradle build automation tool. The project will be automatically compiled before running. To build a separate executable jar use distZip or distTar tasks.

```
Linux:
./gradlew distZip
Windows:
gradlew.bat distZip
```

3.2 Run

Main class is dfki.mm.main.Main. To run the service execute the run task:

```
Linux:
./gradlew run
Windows:
gradlew.bat run
```

4 Mode Detector Service structure

Mode Detector Service has:

- A web service for Feedback Service App to get a mode
- A web interface for configuration

4.1 Mode Detector Service android API

The service starts by dfki.mm.wui.android.JettyMainSubmit.main() which returns a Jetty Server object.

It listens on port 8903. Currently there is only one request:

request A JSON object with location data, which must contain the following fields:

uid:string
email:string
lat:double
lon:double
time:long
altitude:double
bearing:double
speed:double
accuracy:double

response A response is with status code 200 containing a single word for the detected mode, one of: BICYCLE, BUS, CAR, TRAIN, WALK, UNDEF. On error:

- 400: mode cannot be detected
- 501: there was an exception (e.g. parsing json)

Mode detection is performed using the latest model (i.e. the latest one added in the Models page in web interface).

4.2 Mode Detector Service web API

The web service runs on port 12340 (e.g. `http://127.0.1.1:12340/`). The pages are:

Tracks Allows adding, removing and showing of the tracks

Models Allows to add a prediction model

Post-processing allows adding a post processing to any model (averaging over the mode with a moving window of 28)

PCA to create new PCA fields based on the existing ones

Normalize creates new fields that are scaled to fit from 0 to 1

Map Simple map