

ToTheLoo

Jamal, Jonas, Lena, Niklas und Patrick

Allgemeines zur App

Was soll sie können?

- Anzeige der Toiletten in der Umgebung
- Anzeige von Details einer ausgewählten Toilette
- Routing zu einer ausgewählten Toilette
- Toiletten Reviews erstellen

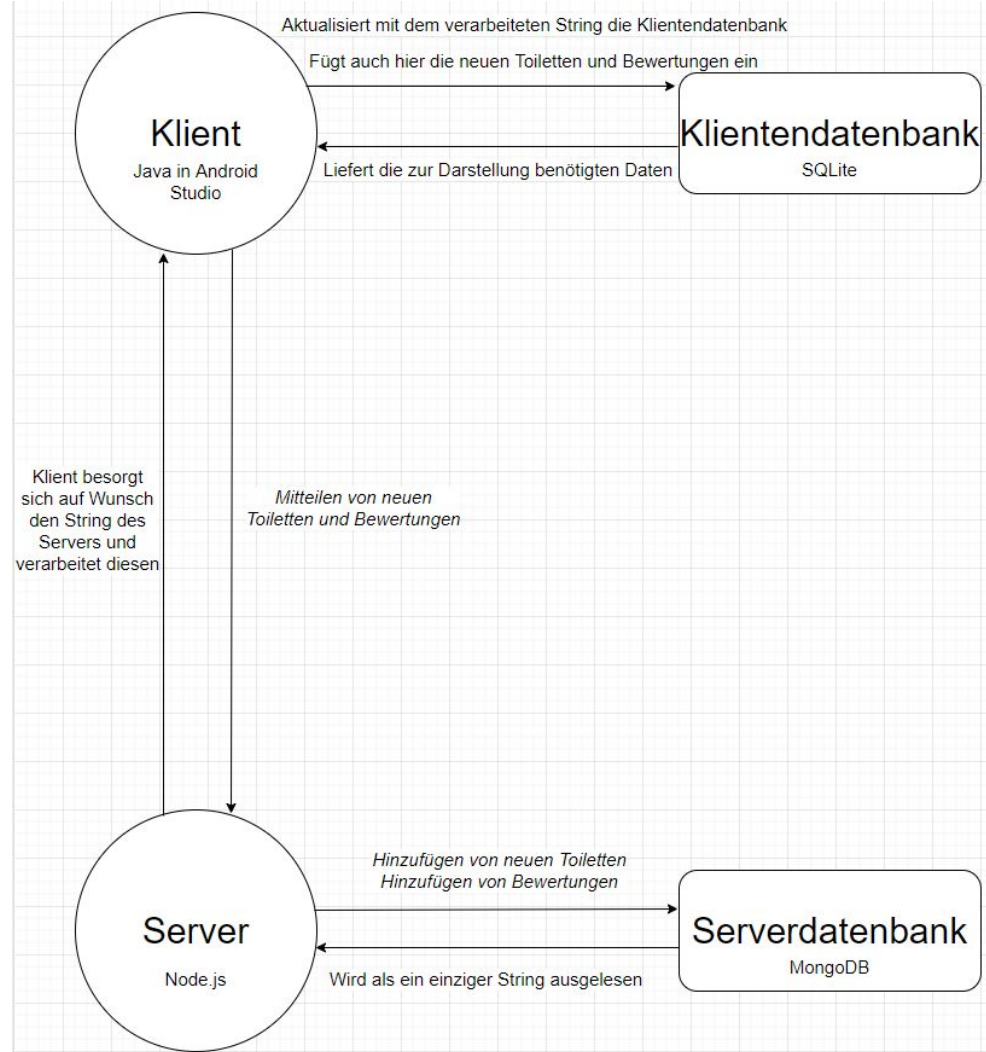
Nice-to-haves:

- Filteroptionen für die anzuzeigenden Toiletten
- Hinzufügen neuer Toiletten

Unsere verwendeten Kontexte

- Standort -> der Standort des Nutzers spielt eine zentrale Rolle
 - Um die Route anzuzeigen.
 - Um nur Toiletten in der Umgebung anzuzeigen.
- Zeit -> die allgegenwärtige Erbauung und Zerstörung von Toiletten über den Verlauf der Zeit
- Rating und Rezensionen sind an Toiletten ids gebunden.

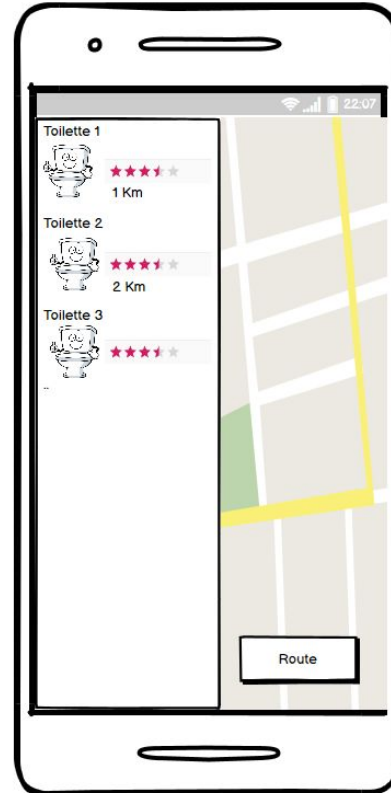
Kommunikation



Erste UI-Entwürfe

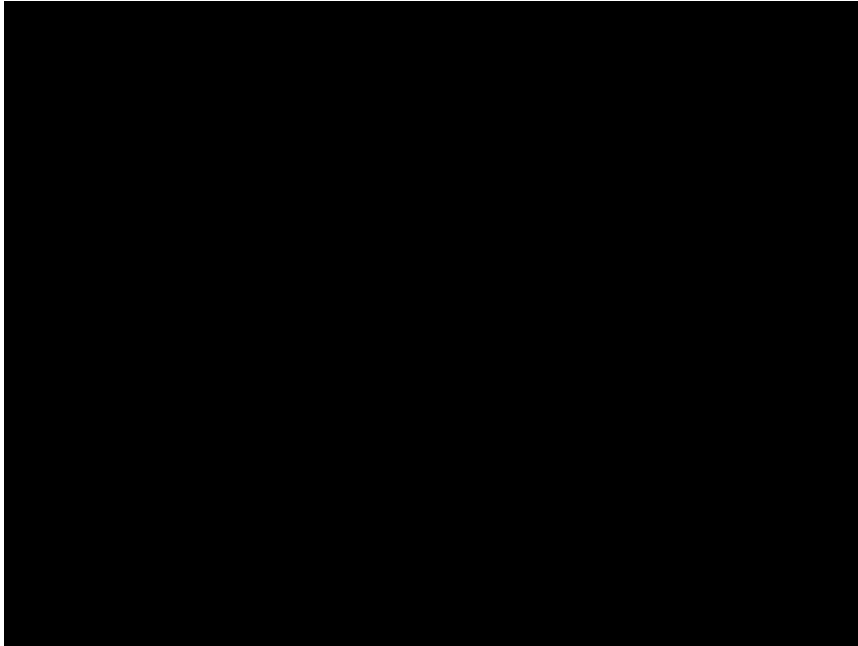


Erste UI-Entwürfe

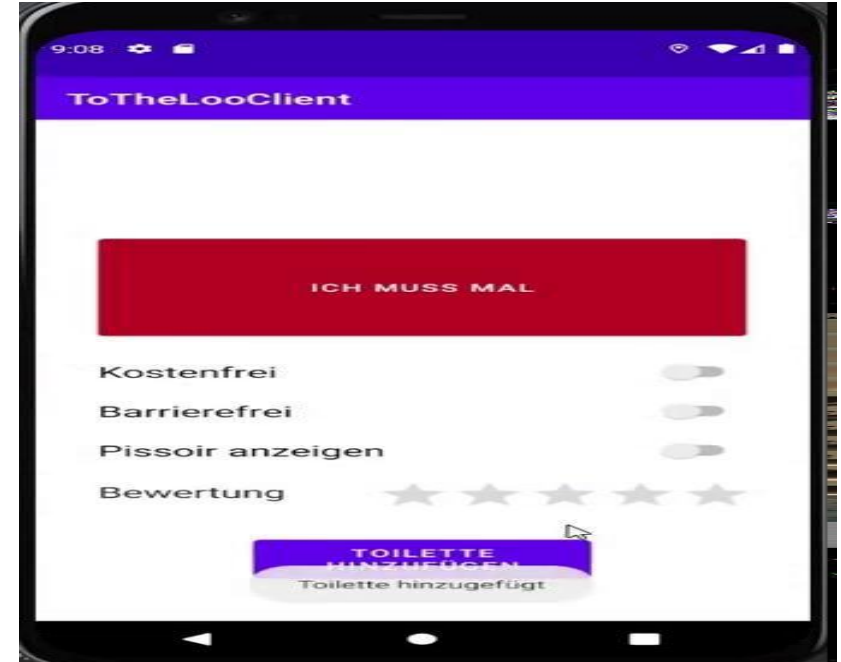


Kurze Präsentation des aktuellen UI

Anzeigen von Toiletten in der Umgebung



Hinzufügen von Toiletten



Evaluation



- Filterfunktion nicht eindeutig
- insgesamt auch zu wenig Filteroptionen
- keinen Hinweis über Standortzugriff
- Verortung und Bedeutung des “Plus”-Symbols nicht eindeutig
- wenn man eine neue Bewertung hinzufügt, kann man sie anschließend nicht überarbeiten

Map auf Codeebene

Die Informationen der Toiletten kommen in Stringform von der lokalen Datenbank.

```
'42;WonderLoo;53.5625;9.9573;4,5 \n 43;easyfalls;53.5725;9.9673;5 \n 44;Quite Place;53.5825;9.4573;3,7 \n 45;Sprinkler Anlage;56.5625;9.9373;1,1 \n 46;Das
```

Diese werden erst bei jedem Zeilenumbruch gesplittet und dann im loop in die einzelne notwendigen Attribute aufgeteilt. Diese gehen dann in die setMarker Methode.

```
String toilettenString = clientDatabase.getAllToiletsAsString(rating,kostenIsChecked);

String [] toiletten = toilettenString.split( regex: "\n");
for (String string : toiletten) {
    String [] parts = string.split( regex: ";");
    String id = parts [0];
    String name = parts [1];
    String Lat = parts [2];
    String Lng = parts [3];
    String Rating = parts [4];
    {
        setMarker(Double.parseDouble(Lat), Double.parseDouble(Lng), name, snip: "Bewertung:" + " " + Rating,id);
    }
}
```

In setMarker wird einmal der Marker erstellt und mit der Toiletten Id in eine Hashmap gepackt damit man für andere Operationen den Marker noch die passende Id zuordnen kann.

```
private void setMarker(double Latitude, double Longitude, String Title, String snip, String id) {  
    LatLng toilet = new LatLng(Latitude, Longitude);  
    Marker m = mMap.addMarker(new MarkerOptions().position(toilet).title(Title).icon(BitmapDescript  
    markerHashMap.put(m,id);  
}
```

Wenn auf ein Info Window geklickt wird, werden die nötigen Informatonen vom Marker und der Hashmap abgerufen und die Operation openMainActivity() Aufgerufen.

```
GoogleMap.OnInfoWindowClickListener MyOnInfoWindowClickListener  
    = new GoogleMap.OnInfoWindowClickListener(){  
    @Override  
    public void onInfoWindowClick(Marker marker) {  
        currentMarkerLat = marker.getPosition().latitude;  
        currentMarkerLng = marker.getPosition().longitude;  
        currentMarkerTitle = marker.getTitle();  
        currentMarkerId = markerHashMap.get(marker);  
        openMainActivity3();  
    }  
};
```

openMainActivity3() übergibt die nötigen Attribute und startet die nächste Activity.

```
public void openMainActivity3() {  
    Intent markerDetails = new Intent( packageContext: this, MainActivity3.class);  
    markerDetails.putExtra( name: "Latitude",currentMarkerLat);  
    markerDetails.putExtra( name: "Longitude",currentMarkerLng);  
    markerDetails.putExtra( name: "Title",currentMarkerTitle);  
    markerDetails.putExtra( name: "id",currentMarkerId);  
    startActivity(markerDetails);  
}
```

Nachdem die Permissions gecheckt wurden, wird alle 10000 MS die momentane Position abgefragt und als lokale Variable gespeichert und danach drawRoute(); ausgeführt.

```
if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_DENIED) {  
    mMap.setMyLocationEnabled(true);  
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, minTimeMs: 10000, minDistanceM: 0, mLocationListener);  
  
    if (mOrigin != null && currentDestination != null)  
        drawRoute();  
}
```

```
private void drawRoute(){  
  
    // Getting URL to the Google Directions API  
    String url = getDirectionsUrl(mOrigin, currentDestination);  
  
    DownloadTask downloadTask = new DownloadTask();  
  
    // Start downloading json data from Google Directions API  
    downloadTask.execute(url);  
}
```

drawRoute() besteht aus 3 Operationen, es holt sich erstens die nötige Url für die googledirectionsapi mit getDirectionsUrl() , erstellt eine neues downloadTask objekt und führt dieses dann mit . execute aus.

```

private String getDirectionsUrl(LatLng origin,LatLng dest){

    // Origin of route
    String str_origin = "origin="+origin.latitude+","+origin.longitude;

    // Destination of route
    String str_dest = "destination="+dest.latitude+","+dest.longitude;

    // Key
    String key = "key=" + getString(R.string.google_maps_key);

    // Building the parameters to the web service
    String parameters = str_origin + "&" + str_dest + "&" + key;

    // Output format
    String output = "json";

    // Building the url to the web service
    String url = "https://maps.googleapis.com/maps/api/directions/"+output+"?" +parameters;

    return url;
}

```

die googledirectionsApi braucht eine Url in der Form <https://maps.googleapis.com/maps/api/directions/outputFormat?parameters>, diese wird mit getDirections() und den Latitude/Longitude Daten des Zielmarkers und der eigenen Position die vorher als Lokale Variable gespeichert wurde generiert.

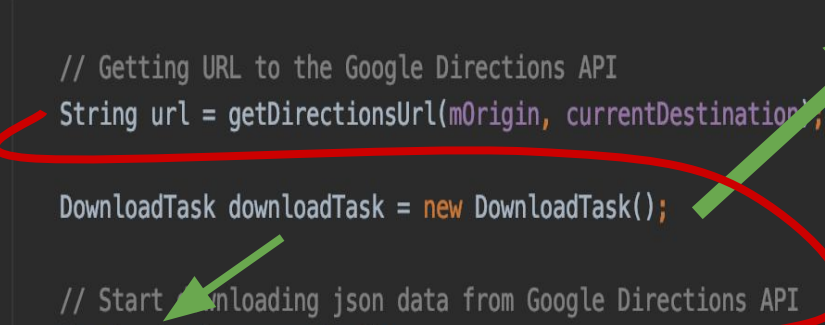
```
private void drawRoute(){

    // Getting URL to the Google Directions API
    String url = getDirectionsUrl(mOrigin, currentDestination);

    DownloadTask downloadTask = new DownloadTask();

    // Start downloading json data from Google Directions API
    downloadTask.execute(url);

}
```



Der DownloadTask macht 2 sachen, er nimmt die eben erstellte Url und holt sich die Daten von der GoogleApi mit der doInBackground() Methode über die downloadUrl() operation.

Und mit den erhaltenen Daten wird dann mit onPostExecute() ein neues ParserTask Object erstellt welchen dann die Polylines für die Route auf die Karte malt.

```
/** A class to download data from Google Directions URL */
private class DownloadTask extends AsyncTask<String, Void, String> {

    // Downloading data in non-ui thread
    @Override
    protected String doInBackground(String... url) {

        // For storing data from web service
        String data = "";

        try{
            // Fetching the data from web service
            data = downloadUrl(url[0]);
            Log.d( tag: "DownloadTask", msg: "DownloadTask : " + data);
        }catch(Exception e){
            Log.d( tag: "Background Task",e.toString());
        }

        return data;
    }

    // Executes in UI thread, after the execution of
    // doInBackground()
    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        ParserTask parserTask = new ParserTask();

        // Invokes the thread for parsing the JSON data
        parserTask.execute(result);
    }

}
```

```

/** A method to download json data from url */
private String downloadUrl(String strUrl) throws IOException {
    String data = "";
    InputStream iStream = null;
    HttpURLConnection urlConnection = null;
    try{
        URL url = new URL(strUrl);

        // Creating an http connection to communicate with url
        urlConnection = (HttpURLConnection) url.openConnection();

        // Connecting to url
        urlConnection.connect();

        // Reading data from url
        iStream = urlConnection.getInputStream();

        BufferedReader br = new BufferedReader(new InputStreamReader(iStream));

        StringBuffer sb = new StringBuffer();

        String line = "";
        while( ( line = br.readLine()) != null){
            sb.append(line);
        }

        data = sb.toString();

        br.close();

    }catch(Exception e){
        Log.d( tag: "Exception on download", e.toString());
    }finally{
        iStream.close();
        urlConnection.disconnect();
    }
    return data;
}

```

Über die `downloadUrl()` Operation wird eine Verbindung zu den GoogleApi servern aufgebaut.

Und nach erfolgreicher ausführung werden die Daten von der GoogleApi in Stringform zurück an den `DownloadTask` gegeben.

Klientendatenbank Implementation

- relationales Datenbankschema
- SQLite
- Vorteil: bereits in Android Studio implementiert
 - ⇒ Erstellung der Datenbank simpel
 - ⇒ Kommunikation mit Datenbank einfach
- Eingabe von Daten als String
- Ausgabe als Cursor Objekt

Klientendatenbank Schema

Attribute der Toiletten:

ToilettenID, Name, Breitengrad, Längengrad, Tag, Wegbeschreibung, Beschreibung

Attribute der Bewertungen: (ToilettenID als Fremdschlüssel gespeichert)

BewertungsID, ToilettenIDFK, BewertungsText, Sterne

Klientendatenbank Implementation

- Als Singleton implementiert
 - Es existiert zur gesamten Laufzeit nur ein Objekt der Klasse
 - Konstruktor ist private -> nur die Klasse selber ein Objekte erzeugen
 - Wird in einer lokalen Variable gespeichert
 - Ausgabe von Objekt der Klasse durch public Methoden (Da zur Erstellung der ersten Instanz ein Kontext benötigt wird muss das erste Objekt mit separater Methode erzeugt werden)

```
private static ClientDatabase INSTANCE = null;
```

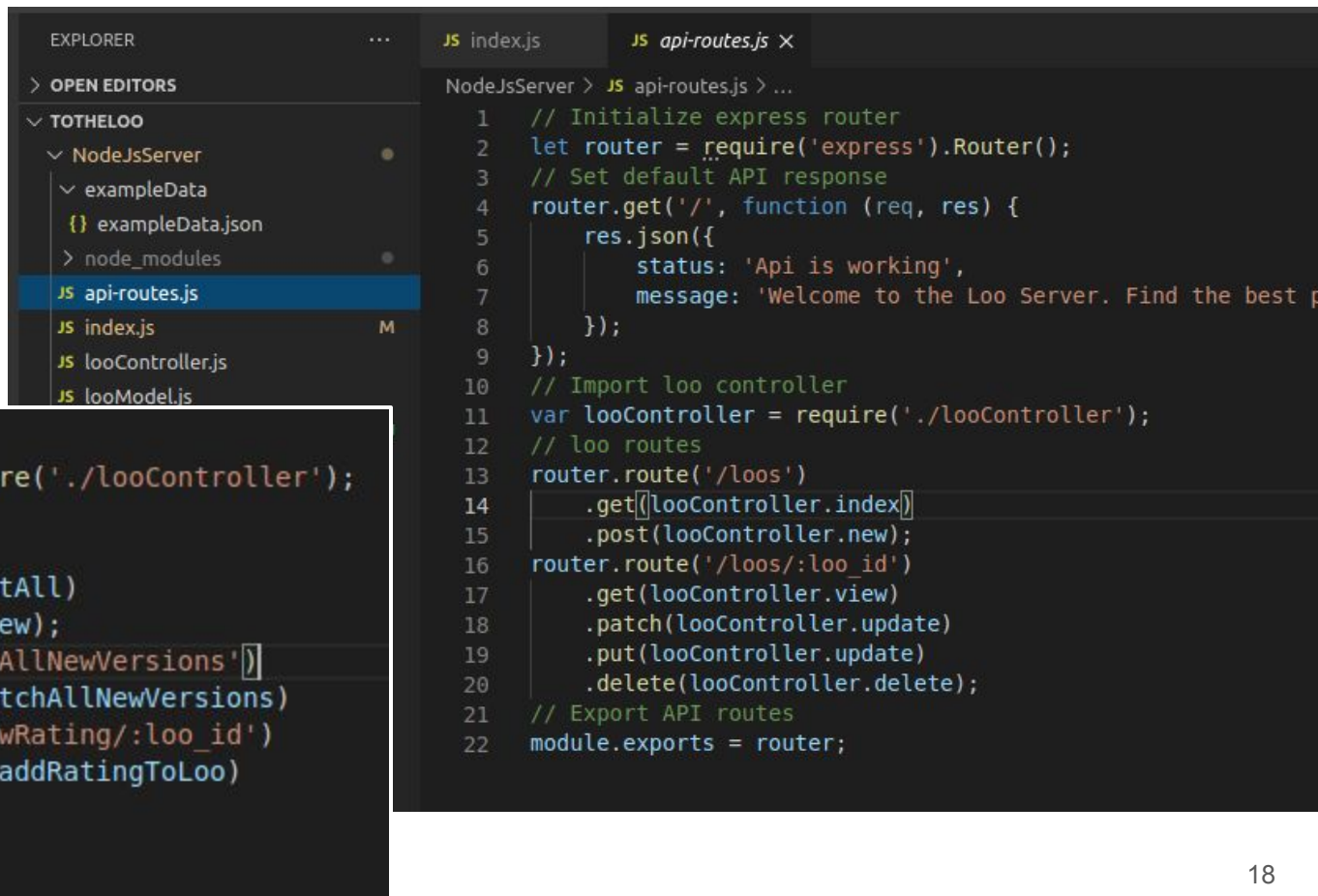
```
private ClientDatabase(@Nullable Context context) { super(context, DATABASE_NAME, factory: null, version: 1); }
```

```
public static ClientDatabase getFirstInstance(Context context) {  
    INSTANCE = new ClientDatabase(context);  
    return INSTANCE;  
}
```

```
public static ClientDatabase getInstance() { return INSTANCE; }
```

Schnittstellen des Backendservers

Erweiterung der
vorhandenen
Routen



```
EXPLORED
> OPEN EDITORS
TOTHESLOO
  NodeJsServer
    exampleData
      {} exampleData.json
    > node_modules
    JS api-routes.js
    JS index.js
    JS looController.js
    JS looModel.js

NodeJsServer > JS api-routes.js > ...
1 // Initialize express router
2 let router = require('express').Router();
3 // Set default API response
4 router.get('/', function (req, res) {
5   res.json({
6     status: 'Api is working',
7     message: 'Welcome to the Loo Server. Find the best p
8   });
9 });
10 // Import loo controller
11 var looController = require('./looController');
12 // loo routes
13 router.route('/loos')
14   .get(looController.index)
15   .post(looController.new);
16 router.route('/loos/:loo_id')
17   .get(looController.view)
18   .patch(looController.update)
19   .put(looController.update)
20   .delete(looController.delete);
21 // Export API routes
22 module.exports = router;
```

```
// Import loo controller
var looController = require('./looController');
// loo routes
router.route('/loos')
  .get(looController.getAll)
  .post(looController.new);
router.route('/loos/fetchAllNewVersions')
  .get(looController.fetchAllNewVersions)
router.route('/loos/addNewRating/:loo_id')
  .patch(looController.addRatingToLoo)
// Export API routes
module.exports = router;
```

Anmerkungen zu “Schnittstellen des Backendservers”

- Die Schnittstellen wurden für spezifische UseCases erweitert
 - es ist nicht nur möglich alle vorhandenen Toiletten abzufragen oder eine neue Toilette zu erstellen, sondern auch ein Rating zu einer Toilette hinzuzufügen und alle Toiletten mit einer neuen Versionsnummer abzufragen

Add new rating auf Serviceebene

Fehler
abfangen

Hinzufügen der
Bewertung

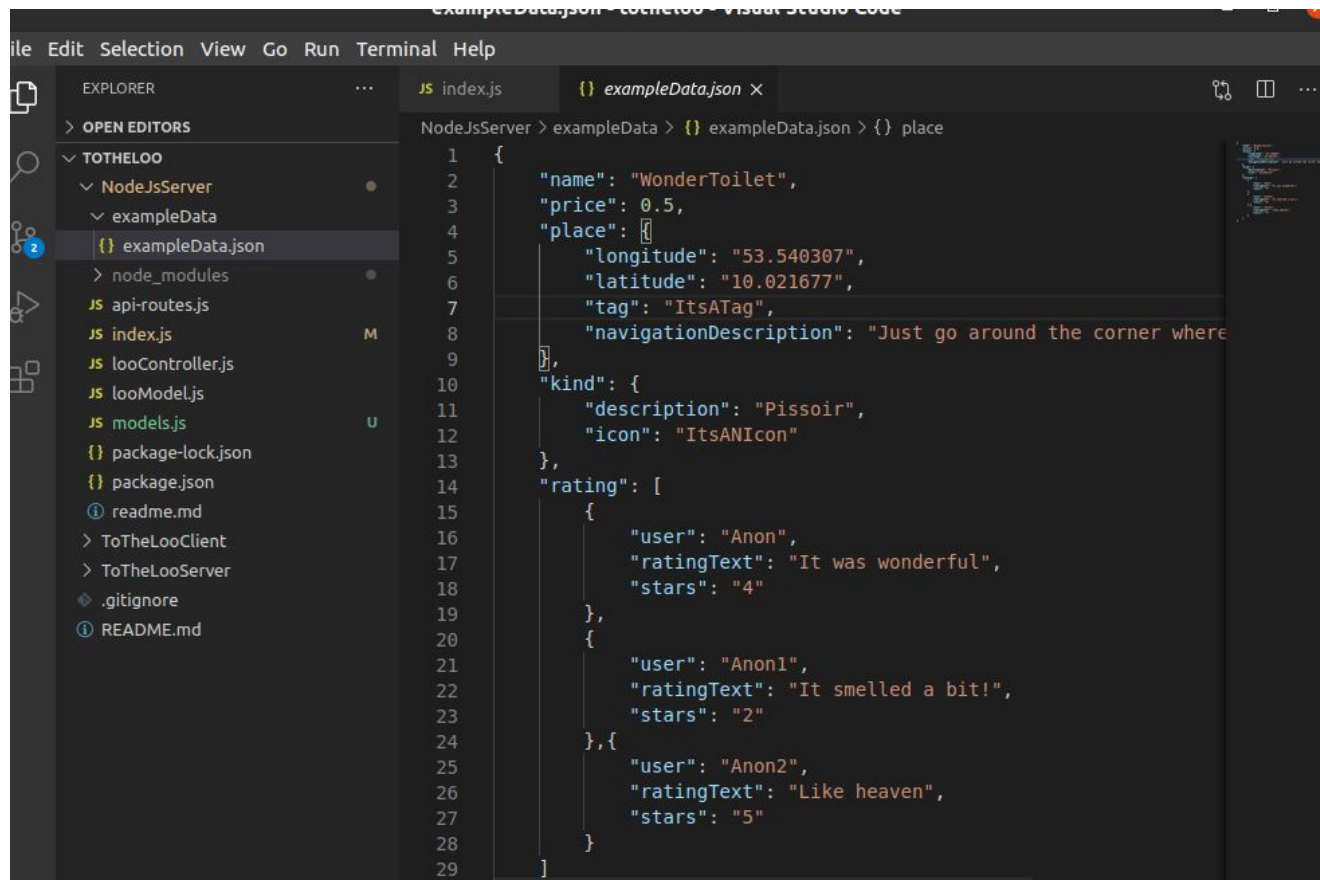
Updaten der
Versionsnummer

Update mit
Änderungen

Antwort bei
Erfolg

```
exports.addNewRatingToLoo = function(req, res) {  
  Loo.find({id: req.params.loo_id}, function (err, loo) {  
    if (err)  
      res.send(err);  
  
    loo[0].rating.push(req.body)  
    loo[0].version = loo[0].version + 1  
    Loo.update(  
      {id: req.params.loo_id},  
      {"rating": loo[0].rating,  
       "version": loo[0].version  
    }, () => {  
      if (err)  
        res.json(err);  
      res.json({  
        message: 'rating added to loo!',  
        data: loo  
      });  
    });  
  })  
}
```

Datenstruktur in der MongoDB (alter Stand aus der Zwischenpräsentation)



```
1 {
2   "name": "WonderToilet",
3   "price": 0.5,
4   "place": {
5     "longitude": "53.540307",
6     "latitude": "10.021677",
7     "tag": "ItsATag",
8     "navigationDescription": "Just go around the corner where"
9   },
10  "kind": {
11    "description": "Pissoir",
12    "icon": "ItsANIcon"
13  },
14  "rating": [
15    {
16      "user": "Anon",
17      "ratingText": "It was wonderful",
18      "stars": "4"
19    },
20    {
21      "user": "Anon1",
22      "ratingText": "It smelled a bit!",
23      "stars": "2"
24    },
25    {
26      "user": "Anon2",
27      "ratingText": "Like heaven",
28      "stars": "5"
29    }
30  ]
31 }
```

Datenbasis durch Dummydaten innerhalb Hamburgs erweitert

MongoDB Compass - localhost:27017

Connect View Collection Help

Local

4 DBS 2 COLLECTIONS

☆ FAVORITE

HOST
localhost:27017

CLUSTER
Standalone

EDITION
MongoDB 4.4.3 Community

Filter your data

ToTheLooDB

loos

admin

config

ToTheLooDB.loos Documents

ToTheLooDB.loos

DOCUMENTS 1 TOTAL SIZE 610B AVG. SIZE 610B INDEXES 1 TOTAL SIZE 20.0KB AVG. SIZE 20.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER OPTIONS FIND RESET ...

ADD DATA VIEW

Displaying documents 81 - 100 of 101

Refresh

```
{
  "_id": ObjectId("600b3ba91205e315ccb45415"),
  "place": Object,
  "kind": Object,
  "name": "LooLooLoo",
  "rating": Array,
  "create_date": 2021-01-22T20:55:05.873+00:00,
  "price": 0.4,
  "__v": 0
}
```

Beispieldaten

Anmerkungen zu DummyDaten

Es wurde ein Python-Script geschrieben, welches Loo-Entitäten generiert und in die Datenbank der EC2 Instanz eingepflegt hat. Diese Daten sind keine realen Daten, sondern erscheinen randomisiert in einem vordefinierten Radius innerhalb Hamburgs.

Datenabfrage Frontend - Backend

```
private void getDataFromBackend() {  
    HttpRequestThread httpThread = new HttpRequestThread();  
    httpThread.start();  
}  
  
class HttpRequestThread extends Thread {  
    @Override  
    public void run() {  
        try {  
            pullLoosFromServerToLocalDatabase();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

wird in der onCreate Methode der MainActivity aufgerufen

- mittlerweile erfolgt die Abfrage bei Aufrufen der MainActivity gestartet
- der Request muss in einen eigenen Thread ausgelagert werden, da Android keine http-requests auf dem MainThread erlaubt
-

- hier sieht man die Response des Servers als KonsolenPrint

```
app x  
D/EGL_emulation: eglCreateContext: 0xf3fae220: maj 2 min 0 rcv 2  
D/EGL_emulation: eglMakeCurrent: 0xf3fae220: ver 2 0 (tinfo 0xf42f9190) (first time)  
I/Gralloc4: mapper 4.x is not supported  
D/HostConnection: createUnique: call  
    HostConnection::get() New Host Connection established 0xf3faf250, tid 13401  
D/goldfish-address-space: allocate: Ask for block of size 0x100  
    allocate: ioctl allocate returned offset 0x3fd7a5000 size 0x2000  
D/HostConnection: HostComposition ext ANDROID_EMU_CHECKSUM_HELPER_v1 ANDROID_EMU_native_sync_v2  
I/System.out: {"status":"success","message":"Loos retrieved successfully","data":[{"place":{"lon
```


Ausblick - Was ist noch denkbar?

- Registrierung von Nutzern
- Den Radius der Toilettenanzeige beschränken
- Gamification: Medaillen für Bewertung oder Hinzufügen von Toiletten
- Informationen über Nutzung des Standorts
- Die DB mit realen Daten füllen