1Proof-Of-Concepts

1.1. GPS Lokalisierung gewährleisten

Für den Erfolg dieses Proof-Of-Concepts ist die Lokalisierung des aktuellen Standorts mit der Implementierung der Google Map Karte sehr bedeutsam. Durch die Überprüfung sollten diese Schnittstellen nicht mehr als Risiko gesehen werden.

Da wir uns in Köln befinden und keine Parkplatzdaten haben, haben wir uns nach der Lokalisierung des Standortes, für den Endziel in der Nähe Chlodwigplatz (Ecke Elsaßstraße, Brunostraße in Köln) entschieden, um die Navigation zu Testen. Das heißt bei jeder durchführen des Testes, wird das Programm uns von unserem Standort aus zum Chlodwigplatz (Ecke Elsaßstraße, Brunostraße) führen.

Eine Verbindung wird mit dem Location-Provider hergestellt um die Positionsdaten beziehen zu können. Der Code getSystemService(Context.LOCATION SERVICE) stellt eine Verbindung mit dem LocationManager dar, womit der Handle für den LocationManager abgerufen werden können. Mit der getLastKnownLocation(provider) stellt der LocationManager eine Anfrage nach dem aktuellen Standort.

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="25dp"
    android:onClick="driveToTest"
    android:text="@string/drivetotest" />
```

Abbildung1: „Android View Button“

```
manager = (LocationManager) getSystemService(LOCATION_SERVICE);
List<String> providers = manager.getAllProviders();

for (String name : providers) {

    LocationProvider lp = manager.getProvider(name);
    Log.d(tag,
            lp.getName() + " --- isProviderEnable(): "
                    + manager.isProviderEnabled(name));

    Log.d(tag, "requiresCell(): " + lp.requiresCell());
    Log.d(tag, "requiresNetwork(): " + lp.requiresNetwork());
    Log.d(tag, "requiresSatellite(): " + lp.requiresSatellite());

}

Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.POWER_LOW);

name = manager.getBestProvider(criteria, true);

Log.d(tag, name);

listener = new LocationListener() {

    @Override
    public void onLocationChanged(Location location) {

        Log.d(tag, location.toString());

    }

    @Override
    public void onProviderDisabled(String provider) {
        // Auto-generated method stub

    }
```

```
        @Override
        public void onProviderEnabled(String provider) {
            // Auto-generated method stub

        }

        @Override
        public void onStatusChanged(String provider, int status,
                Bundle extras) {
            // Auto-generated method stub

        }

    };

    manager.requestSingleUpdate(name, listener, looper);

    savedPosition = manager.getLastKnownLocation(name);
```

Abbildung2: „Code zum Lokalisieren der eigenen Position“

Um Google Maps zu starten erzeugen wir einen Link welcher wir mittels neuem Intent aufrufen. Da die Google Maps App automatisch auf diesen Link reagiert, erhält der User die Möglichkeit, diesen Link in Google Maps auszuführen und wird dort von der einen Koordinate zur anderen navigiert.

```
public void startMaps(Location pstart, Location pstop) {

    Intent intent = new Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://maps.google.com/maps?saddr="
                    + pstart.getLatitude() + "," + pstart.getLongitude()
                    + "&daddr=" + pstop.getLatitude() + ","
                    + pstop.getLongitude()));

    startActivity(intent);

}
```

Abbildung3: „Code zum Starten von Google Maps“

In dieser Methode wird die Zielposition definiert, in diesem Fall immer ein Ort in der Nähe des Chlodwigplatzes. Des Weiteren wird der aktuelle Standort übergeben.

```
public void driveToTest(final View view){

    Location target = new Location("");
    target.setLatitude(50.921301);
    target.setLongitude(6.9557633);

    startMaps(savedPosition, target);

}
```

Abbildung4: „Code zum Sammeln der Navigationsdaten“

8.2 Client-Server - Server

So werden die sich konnektierenden Clients verarbeitet. Der Listener wartet solange, bis sich ein neuer Client anmeldet und erzeugt dann ein neues Socket Objekt, welches dann dem verarbeitendem Objekt Handler mitgegeben wird.

```
private void listen() {

    try {

        //AbhÃ¶rer erstellen
        listener = new ServerSocket(portForClients);

        while(run){

            //jedes neue GerÃ¤t ein neuer Socket
            Socket client = listener.accept();

            System.out.println("New client noticed " + client.getInetAddress().getHostName());

            //pro neuem GerÃ¤t ein neuer Handler erzeugt
            Handler pHandler = new Handler(client);

            //Handler ans Ende der Liste
            addHandlerToLastHandler(pHandler);

        }

    } catch (IOException e) {

        //wenn nicht funktioniert, weinen und beenden
        e.printStackTrace();

        System.exit(-2);

    }

}
```

Abbildung5: „soe.boe.eis.server.Server.listen()“

Hier sieht man wie das Protokoll abgearbeitet wird, zu Testzwecken ist allerdings nur der Login und das Abmelden implementiert. Das Protokoll befindet sich im Anhang. Auch wird hier die Richtigkeit der Daten nicht auf Grundlage der Datenbankdaten überprüft, sondern es sind feststehende Testdaten.

```
public void run(){

    //Implementation des Protokolls
    conout.println("Hello");

    try {

        conin.readLine();

        conout.println("#001");

        String sParkingPlaceNumber = conin.readLine();

        //TODO Ab hier zu feactorisieren
        if (sParkingPlaceNumber.startsWith("#002")){

            int parkingPlaceNumber = Integer.parseInt(sParkingPlaceNumber.substring(5));

            //nur zum Testen
            if (parkingPlaceNumber == 13 || parkingPlaceNumber == 17){

                conout.println("#003");

                String sCheckCode = conin.readLine();

                if (sCheckCode.startsWith("#005")){

                    sCheckCode = sCheckCode.substring(5);

                    //nur zum Testen
                    if((parkingPlaceNumber == 13 && sCheckCode.equals("Schinken")) || (parkingPlaceNumber == 17 && sCheckCode.equals("Wurst"))){

                        conout.println("#006");

                        while(running){

                            String sCommand = conin.readLine();

                            if (sCommand.startsWith("#013")){
```

```
                            running = false;

                            conout.println("#013");

                            //TODO Hier Quelltext einfügen
                        } else {
                            conout.println("#012");
                        }

                    }

                } else {
                    conout.println("#007");
                }

            } else {
                conout.println("#007");
            }

        } else {
            conout.println("#004");
        }

    }

    conout.println("#012");
    conout.println("#013");

} catch (IOException e) {
    e.printStackTrace();
}

}
```

8.3 Client-Server - Client

Hier verbinden wir uns Testweise mit dem Server von einem "normalen" Client.
Sämtliche Benötigten Daten werden manuell eingegeben, um in Punkten wie der Adresse des Servers frei zu sein. Sobald der Server die Kommunikation beendet, wird auch das Programm beendet.

```
try {

    t = new Socket(ipAddress, port);
    conin = new Stream(t.getInputStream());
    conout = new PrintWriter(t.getOutputStream(), true);

    conin.readLine();
    conout.println("Hello");
    conin.readLine();

    System.out.print("ID des Parkplatzes: ");
    String pParkingPlaceNumber = scanner.nextLine();
    System.out.println();

    conout.println("#002 " + pParkingPlaceNumber);

    String pAnswer = conin.readLine();

    if(pAnswer.startsWith("#003")){

        System.out.print("Chackcode des Parkplatzes: ");
        String pCheckCode = scanner.nextLine();
        System.out.println();

        conout.println("#005 " + pCheckCode);

        pAnswer = conin.readLine();

        if(pAnswer.startsWith("#006")){

            // TODO Hier Quelltext einfügen
            System.out.println("Test erfolgreich");
            conout.println("#013");
            conin.readLine();

            System.exit(0);

        } else {

            System.out.println("Checkcode war falsch!");
```

```
                System.exit(0);

            }

        } else {

            System.out.println("ID war falsch!");

            System.exit(0);

        }

    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

}

public static void main(String[] args){
    new ParkingPlaceSoftware();
}

}
```

8.4 Client-Server - App

Hier ist dasselbe, nur für die App mit ThreadPolicy, um im Hauptthread ins Internet zu können (wurde nur zu Testzwecken gemacht). Des Weiteren ist die Eingabe der Daten angepasst.

```
public void connect(final View view){

    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);

    EditText edServerAddress = (EditText) findViewById(R.id.edserveraddress);
    EditText edServerPort = (EditText) findViewById(R.id.edserverport);
    EditText edParkingPlaceID = (EditText) findViewById(R.id.edparkingplaceid);
    EditText edSecCode = (EditText) findViewById(R.id.edseccode);

    String ipAddress = edServerAddress.getText().toString();
    int port = Integer.parseInt(edServerPort.getText().toString());
    int pParkingPlaceNumber = Integer.parseInt(edParkingPlaceID.getText().toString());
    String pCheckCode = edSecCode.getText().toString();

    System.out.println("ipAddress: " + ipAddress);
    System.out.println("port: " + port);
    System.out.println("pParkingPlaceNumber: " + pParkingPlaceNumber);
    System.out.println("edSecCode: " + pCheckCode);

    try {

        t = new Socket(ipAddress, port);
        conin = new Stream(t.getInputStream());
        conout = new PrintWriter(t.getOutputStream(), true);

        conin.readLine();
        conout.println("Hello");
        conin.readLine();

        conout.println("#002 " + pParkingPlaceNumber);

        String pAnswer = conin.readLine();

        if(pAnswer.startsWith("#003")){

            conout.println("#005 " + pCheckCode);

            pAnswer = conin.readLine();
```

```
            pAnswer = conin.readLine();

            if(pAnswer.startsWith("#006")){

                // TODO Hier Quelltext einfügen
                System.out.println("Test erfolgreich");
                conout.println("#013");
                conin.readLine();

            } else {

                System.out.println("Checkcode war falsch!");

            }

        } else {

            System.out.println("ID war falsch!");

        }

    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

8.5 Server-Datenbank

Im folgendem der Test zum Verbinden mit der Datenbank. Die Methode gibt einen boolschen Wert über den Erfolg/Misserfolg des Verbindungsaufbaus zurück.

```
public boolean connect() {

    try {

        Connection conn = DriverManager.getConnection(url, "eis", "Schinkenwurst");

        st = conn.createStatement();

        return true;

    } catch (SQLException e) {

        e.printStackTrace();

        return false;

    }


}
```

In der Testmethode wird ein Wert aus der Datenbank entnommen und um 1 erhöht und dann in der Datenbank erneut gespeichert. Zur Kontrolle wird auch dieser Wert nochmals heruntergeladen.

```
    public void test() {

        try {

            rs = st.executeQuery("SELECT * FROM poc");

            int i = 0;

            while (rs.next()) {
                i = rs.getInt(1);
            }

            System.out.println("i (Start): " + i);

            i++;

            System.out.println("i (Soll): " + i);

            st.executeUpdate("UPDATE poc SET `value`=" + i);

            rs = st.executeQuery("SELECT * FROM poc");

            while (rs.next()) {
                System.out.println("i (Ende): " + rs.getInt(1));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }

    }

}
```