**Projektarbeit  
E-Charge App**

NUTZERZENTRIERTE SOFTWAREENTWICKLUNG  
WS21/22  
–  
DOKUMENTATION

von

Herr Paul Bartelt (745280)

Herr Samuel Koob (769070)

Herr Sebastian Zill (769544)

Datum: 11.02.2022

**InhalT**

[1. Aufgabenbeschreibung 1](#_Toc95510832)

[2. Projektanforderungen 1](#_Toc95510833)

[2.1 User Stories 1](#_Toc95510834)

[2.2 Rahmenbedingungen 2](#_Toc95510835)

[3. Projektumfeld 2](#_Toc95510836)

[4. Ablauf und Terminplanung 3](#_Toc95510837)

[5. Funktionale Umsetzung der Projektanforderungen 4](#_Toc95510838)

[5.1 Screen Navigation 4](#_Toc95510839)

[5.2 Daten einlesen 5](#_Toc95510840)

[5.3 Design Konzept 8](#_Toc95510841)

[5.4 GoogleMap 9](#_Toc95510842)

[5.5 Umkreissuche 11](#_Toc95510843)

[5.6 RecyclerView 12](#_Toc95510844)

[5.7 Favoriten 14](#_Toc95510845)

[5.8 Defekte melden 14](#_Toc95510846)

[5.9 Filter Funktionen 15](#_Toc95510847)

[5.10 Persistente Daten 18](#_Toc95510848)

[5.11 Login methoden 20](#_Toc95510849)

[5.12 Mehrsprachigkeit 21](#_Toc95510850)

[5.13 Landscape Modus 22](#_Toc95510851)

[5.14 Feinschliff 23](#_Toc95510852)

[6. Ausblick 24](#_Toc95510853)

[7. Fazit 25](#_Toc95510854)

[8. Quellen 26](#_Toc95510855)

**Sonder-Inhaltsverzeichnisse**

**Anhang:**

[A.1 Eigenständigkeitserklärung: I](#_Toc95510856)

[A.2 App Durchlauf anhand Bilderreihe II](#_Toc95510857)

[A.3 Doxygen Java Dokumenation VII](#_Toc95510858)

**Darstellungsverzeichnis:**

[Tabelle 1: Praktikumstermine und -inhalte 3](#_Toc95510859)

[Abbildung 1: Arbeitsablauf 3](#_Toc95510860)

[Abbildung 2: Gegenüberstellung Activity und Fragment [3] 4](#_Toc95510861)

[Codebeispiel 1: DownloadThread.java 6](#_Toc95510862)

[Codebeispiel 2: StationAPI.java 7](#_Toc95510863)

[Abbildung 3: Erstes Design Konzept 8](#_Toc95510864)

[Codebeispiel 3: MainActivity.java 9](#_Toc95510865)

[Abbildung 4: GoogleMap mit Markierungen 10](#_Toc95510866)

[Codebeispiel 4: Berechnung Abstand zweier Geokoordinaten [5] 11](#_Toc95510867)

[Codebeispiel 5: RecyclerView.java 13](#_Toc95510868)

[Abbildung 5: Filter Funktionen Pop Up 15](#_Toc95510869)

[Codebeispiel 6: PopUpService.java 17](#_Toc95510870)

[Codebeispiel 7: Favoriten oder Defekte in Datenbank schreiben 18](#_Toc95510871)

[Codebeispiel 8: Favoriten oder Defekte aus Datenbank entfernen 18](#_Toc95510872)

[Codebeispiel 9: Favoriten oder Defekte aus Datenbank lesen 19](#_Toc95510873)

[Codebeispiel 10: LoginFragment.java 20](file:///F:\Dokumente\Programming\Uni\Semester-3\NZSE\nzse_2021_sa_se_pa\Do2x_Projektdokumentation_Bartelt_Koob_Zill.docx#_Toc95510874)

[Abbildung 6: Login Screen 20](#_Toc95510875)

[Abbildung 7: Tabelle Übersetzungen 21](#_Toc95510876)

[Codebeispiel 11: Multilanguage Support Strings 21](#_Toc95510877)

[Codebeispiel 12: AndroidManifest.xml Screen Orientation 22](#_Toc95510878)

[Abbildung 8: Landscape Screens 22](#_Toc95510879)

## Aufgabenbeschreibung

Gegenstand der Praktikumsaufgabe ist die Entwicklung einer App zur E-Mobilität. Nutzer der App sollen Zugang zu Informationen von Ladestationen für Elektrofahrzeuge erhalten. Dafür steht eine Datei der Bundesnetzagentur mit allen registrierten Ladesäulen und Informationen zu diesen (Steckertypen etc.) zur Verfügung. Darüber hinaus sollen nicht nur „normale“ Nutzer, sondern auch Servicemitarbeiter, die sich um die Instandhaltung der Ladesäulen kümmern, einen Zugang erhalten. Am Ende soll ein funktionsfähiger Prototyp entstehen, der möglichst intuitiv für Nutzer gestaltet ist. Dazu werden verschiedene Rahmenbedingungen durch das Praktikumsaufgabenblatt und Rücksprachen mit Prof. Dr. Wiedling gestellt. Diese werden in den folgenden Abschnitten aufgelistet und anhand von Erläuterungen, Codebeispielen und Bildern weiter beleuchtet.

## Projektanforderungen

### User Stories

Im 1. Praktikumstermin wurden User Stories formuliert, welche einen Teil der Rahmenbedingungen bilden und den Verlauf der App-Entwicklung maßgeblich beeinflussen. Diese sind folglich zusammengefasst, wobei User Stories wie zum Beispiel „Als Nutzer möchte ich eine Route planen können.“ nicht behandelt werden, da dies eine andere Aufgabenstellung verlangt:

**User Stories Nutzer:**

1. Als Nutzer möchte ich Ladesäulen in Umgebung und Ferne angezeigt bekommen,

um meine Route besser planen zu können.

1. Als Nutzer möchte ich diverse Informationen zu Ladesäulen angezeigt bekommen,

um meine Auswahl treffend wählen zu können.

1. Als Nutzer möchte ich defekte Ladestationen melden können,

damit diese schnellstmöglich repariert werden.

1. Als Nutzer möchte ich Favoriten hinzufügen können,

um meine Lieblingsstandorte schnell wieder finden zu können.

1. Als Nutzer möchte ich nach bestimmten Kriterien filtern können,

um passende Ladesäulen zu finden.

**User Stories Service:**

1. Als Service Mitarbeiter möchte ich defekte Ladesäulen angezeigt bekommen,

um diese schnell bearbeiten zu können.

1. Als Service Mitarbeiter möchte ich reparierte Ladesäulen wieder freigeben können,

um die Nutzer und meine Kollegen über die getätigte Arbeit zu informieren.

1. Als Service Mitarbeiter möchte ich Angaben zu den defekten Ladesäulen erhalten,

um meine Auswahl anzufahrender Ladesäulen passend treffen zu können.

1. Als Service Mitarbeiter möchte ich nach Ladesäulen in meiner Nähe filtern können,  
   um meine Arbeitsrouten besser planen zu können.

### Rahmenbedingungen

Aus den User Stories, dem Aufgabenblatt und Rücksprachen mit Prof. Dr. Wiedling lassen sich schließlich folgende Rahmenbedingungen zur Projektarbeit formulieren:

* Es soll ein funktionsfähiger Prototyp entstehen, den man als APK Datei exportieren und auf mobilen Geräten installieren und ausführen kann.
* In der Applikation sollen Bedienungen nach Nutzern und Service Mitarbeitern unterschieden werden können.
* Der Nutzer soll auf dem individuell verwendeten Endgerät über persistente und persönliche Daten, wie Favoriten und Standort, lokal verfügen.
* Für die große Anzahl an Ladesäulen soll die Funktionalität eines RecyclerViews verwendet werden, um nicht alle Ladesäulen auf einmal zu laden und somit die Auslastung des Gerätes und Netzwerkes zu verringern.
* Eine Umkreissuche, um nur Ladesäulen in bestimmten Entfernungen zum Nutzer anzuzeigen, soll implementiert werden.
* Diverse weitere Filterfunktionen nach den lokalen Präferenzen des Nutzers sollen für die Ladesäulen möglich sein. Unter diese fallen unter anderem favorisierte und als defekt markierte Standorte.
* Der Nutzer soll für Ladesäulen Defekte melden, und *nur* der Service Mitarbeiter diese werden entfernen können.
* Die App soll eine Funktionalität zur Mehrsprachigkeit anhand der eingestellten Systemsprache des Endgerätes haben.
* Die App soll in mindestens einem Screen im Landscape Modus benutzbar sein.

## Projektumfeld

Das Projektumfeld wird vorgegeben. Die verwendete Programmiersprache ist **Java** und die verwendete IDE **Android Studio**. Diese beinhaltet auch gleichzeitig einen **Emulator** zum Testen des Prototypen. Zusätzlich haben wir die Möglichkeit die App auch nativ auf einem **Android Gerät** zu testen. Zur besseren Teamarbeit wird **GitLab** zur Versionsverwaltung und **Discord** zurKommunikation verwendet. Um ein übersichtliches Design zu erstellen, wird das Programm **InVision Studio** (ähnlich AdobeXD) benutzt.

## Ablauf und Terminplanung

Die Terminplanung orientiert sich vor allem an dem Praktikumsblatt und den im vorherigen Abschnitt erwähnten User Stories und Anforderungen. In Tabelle 1 wird dies noch einmal zusammengefasst.



Tabelle 1: Praktikumstermine und -inhalte

Somit ist der Zeitraum festgelegt und die einzelnen Schritte können nacheinander abgearbeitet werden, wobei sich teilweise Features überschneiden oder voneinander abhängig sind, sodass andere Funktionen früher gebraucht werden als gedacht oder Klassen und deren Funktionen durch spätere Stände ergänzt wurden.

**In der folgenden Abbildung ist unser Arbeitsablauf grob skizziert:**

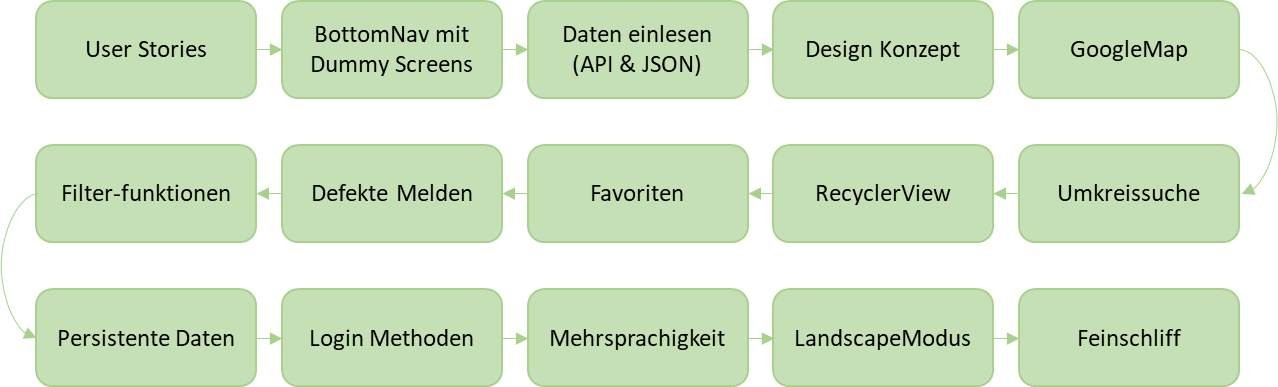


Abbildung 1: Arbeitsablauf

## Funktionale Umsetzung der Projektanforderungen

In diesem Abschnitt wird die Umsetzung der bereits in den vorherigen Abschnitten gestellten Projektanforderungen dargestellt, um in der funktionierenden Applikation anhand der User Stories zu resultieren. Es wird anhand des Arbeitsablaufes aus Abbildung 1 strukturiert, wobei einige Punkte parallel oder in Abhängigkeiten zueinander umgesetzt und ergänzt wurden.

### Screen Navigation

Nach der Entwicklung der User Stories und dem Kennenlernen der IDE Android Studio ging es zunächst darum die ersten Screens zu erstellen. Dies beinhaltet auch die Navigation zwischen den einzelnen Screens. Es gibt verschiedene Möglichkeiten zwischen zwei Screens hin und her zu wechseln. Die zwei am häufigsten verwendeten sind Tabs und Bottom Navigation. Tabs sind meist am oberen Bildschirmrand angesiedelt und vielen Usern bereits durch Verwendung eines Internet-Browsers bekannt. Die Bottom Navigation ist hingegen am unteren Bildschirmrand angesiedelt und vielen Usern durch die Navigation innerhalb eines Android oder IOS Betriebssystems bekannt.

Unsere Entscheidung fiel auf die Bottom Navigation, da diese aus der Sicht des Nutzers einfacher und intuitiver zu bedienen ist. Sie ist am unteren Bildschirmrand angesiedelt, wodurch es dem Nutzer einfacher fällt sie zu erreichen und hierdurch auch die Barrierefreiheit verbessert wird, da die Navigation auf jedem Screen gleich immer direkt sichtbar ist und die einzelnen Seiten durch ihre Icons erläutert sind. Da es sich außerdem um eine reine Android-App handelt und keine Web-App oder ähnliches vorgesehen ist, ist die Bedienung am unteren Bildschirmrand analog zur Bedienung des Smartphones und vielen anderen Apps.

Erstellt wurde die Bottom Navigation Bar mit Hilfe eines YouTube-Tutorials [1] und der Android Studio Dokumentation [2]. Auch hier muss wieder eine Entscheidung getroffen werden: Verwendet man Activities oder Fragments für die Screens?

Wir haben uns schließlich für Fragments entschieden, um eine übersichtlichere Projektstruktur zu gewährleisten. Eine Gegenüberstellung der beiden Möglichkeiten ist in Abbildung 2 zu sehen.

Ein Bild, das Text, Screenshot, Monitor, schwarz enthält.

Automatisch generierte Beschreibung

Abbildung 2: Gegenüberstellung Activity und Fragment [3]

### Daten einlesen

Nach dem Erstellen der Dummy Screens und der Bottom Navigation Bar ging es um das Einbinden der gegebenen Daten. Diese liegen in Form einer CSV-Datei und dank Herrn Quick auch als API vor. Wir haben uns dazu entschieden die API zu nutzen, da sich der offizielle Link der Bundesnetzagentur als externe Quelle [4] oder dessen interne Daten jederzeit verändern könnten und zudem dadurch das Dateiformat JSON vorliegt. Wir hatte ebenso bereits in letzten Semestern persönlichen Kontakt und Zusammenarbeit mit Herrn Quick als Lerngruppe und uns daher vor Projekt Kick Off bereits über die Vor- und Nachteile einer JSON gegenüber einer CSV ausgetauscht. Ein solcher Vorteil von JSON wäre zum Beispiel, dass sich die Dateistruktur generell und über Bibliotheken wie GSON besser eignet, um Objekte aus dessen Inhalten zu serialisieren und zu generieren.

Die Daten werden mittels der API von Herrn Quick [5] und dem Zusammenspiel mehrerer Klassen und Threads eingelesen:

**DownloadThread.java**: In einem weiteren Thread wird über die Implementation einer runnable die auf der API liegende JSON-Datei [5] heruntergeladen. Wir haben uns für die Verwendung eines parallelen Threads entschieden, da wir in ersten Tests festgestellt haben, dass der Download einige Zeit benötigt und in dieser Zeit die gesamte App stillsteht, was den Nutzer fälschlicherweise in die Vermutung eines Absturzes verleiten und generell irritieren könnte. Der Download und die Verarbeitung der JSON Datei wird in der Funktion getJson() abgewickelt. Anschließend wird der JSON-String über die Google - GSON Library mit der Funktion gson.fromJson() in Objekte konvertiert.

Diese Liste des DownloadThreads wird dann über die initialize() der **StationAPI.java** anhand der Funktion getStations() in den **GlobalStorage.java** übergeben, sodass die weiteren Fragmente auf diese Liste zugreifen und die Stationen verwenden können.

1. public class DownloadThread implements Runnable
2. {
3. private volatile ArrayList<Ladestation> stations;
5. /\*\*
6. \* Overrides the Runnable's run method
7. \*/
8. @Override
9. public void run()
10. {
11. Gson gson = new Gson(); // get gson object (generate objects from json format)  
     // set type of Ladestation object for gson return
12. Type return\_type = new TypeToken<ArrayList<Ladestation>>(){}.getType();
13. final String json = getJson(); // Load json from API url and save into var  
     // generate and return ArrayList from parsed json string with gson
14. this.stations = gson.fromJson(json, return\_type);
15. }
17. /\*\*
18. \* Fetches the stations as a JSON from the given URL
19. \* @return JSON-Formatted string
20. \*/
21. private String getJson()
22. {
23. BufferedReader reader = null;
24. try
25. {
26. try
27. {
28. URL url = new URL("https://api.aurora-theogenia.de/chargingstations/charging\_stations.json"); // read into buffer from url
29. // Create buffer to load giant string by chunks
30. reader = new BufferedReader(new InputStreamReader(read(url)));
31. StringBuffer buffer = new StringBuffer();
33. int read;
34. char[] chars = new char[1024]; // Read every 1024 chars into char array
35. while ((read = reader.read(chars)) != -1)
36. {
37. buffer.append(chars, 0, read); // Append those chars onto buffer
38. }
39. return buffer.toString(); // Convert buffer to string and return it
40. }
42. finally // once done close the reader again
43. {
44. if (reader != null)
45. {
46. reader.close();
47. }
48. }
49. }
50. catch (Exception e)
51. {
52. e.printStackTrace();
53. return null;
54. }
55. }
57. /\*\*
58. \* Reads a URL connection's stream and returns it
59. \* @param url The URL to get the stream from
60. \* @return The fetched InputStream
61. \* @throws IOException
62. \*/
63. private InputStream read(URL url) throws IOException
64. {
65. HttpURLConnection httpCon = (HttpURLConnection) url.openConnection();
66. httpCon.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36");
67. httpCon.setConnectTimeout(10000);
68. httpCon.setReadTimeout(10000);
69. return httpCon.getInputStream();
70. }
72. // return the stations ArrayList to save into ArrayList in StationsAPI
74. /\*\*
75. \* Getter for all stations
76. \* @return the stations ArrayList
77. \*/
78. public ArrayList<Ladestation> getStations(){return stations;}
79. }

Codebeispiel 1: DownloadThread.java

1. (…)
2. /\*\*
3. \* Initializes the Station API with default values
4. \* @param context
5. \*/
6. public static void initialize(Context context)
7. {
8. // generate downloadthread object
9. DownloadThread dl = new DownloadThread();
10. Thread thread = new Thread(dl);
11. thread.start();
12. try
13. {
14. thread.join();
15. }
16. catch (InterruptedException e)
17. {
18. e.printStackTrace();
19. }
20. // return and save downloaded ArrayList into api static
21. GlobalStorage.setAllStations(dl.getStations());
23. // fill fav array
24. sqLiteHelper = new SQLiteHelper(context);
25. GlobalStorage.setFavStations(sqLiteHelper.readFavouritesFromDB());
27. // fill defect array
28. GlobalStorage.setDefStations(sqLiteHelper.readFlagsFromDB());
29. GlobalStorage.setSaveCheckedState(new SaveCheckedState());
30. }
31. (…)

Codebeispiel 2: StationAPI.java

Die Struktur der Objekte sind identisch zur Klasse **Ladestation.java** und deren Daten können über Getter und Setter dieser Klasse abgefragt und bearbeitet werden.

### Design Konzept

Nachdem wir nun die Daten in unserer App zur Verfügung und bereit zur Darstellung hatten, wurde von uns als nächsten Schritt ein erstes Design Konzept mit Hilfe des UI/UX Tools InVision Studio entwickelt.

Wir entschieden uns für diesen Zwischenschritt, um uns vor Implementation einen Roten Faden zur Nutzerzentrierung zu bilden, um ein einheitliches und schlankes Design für unsere App zu erhalten.

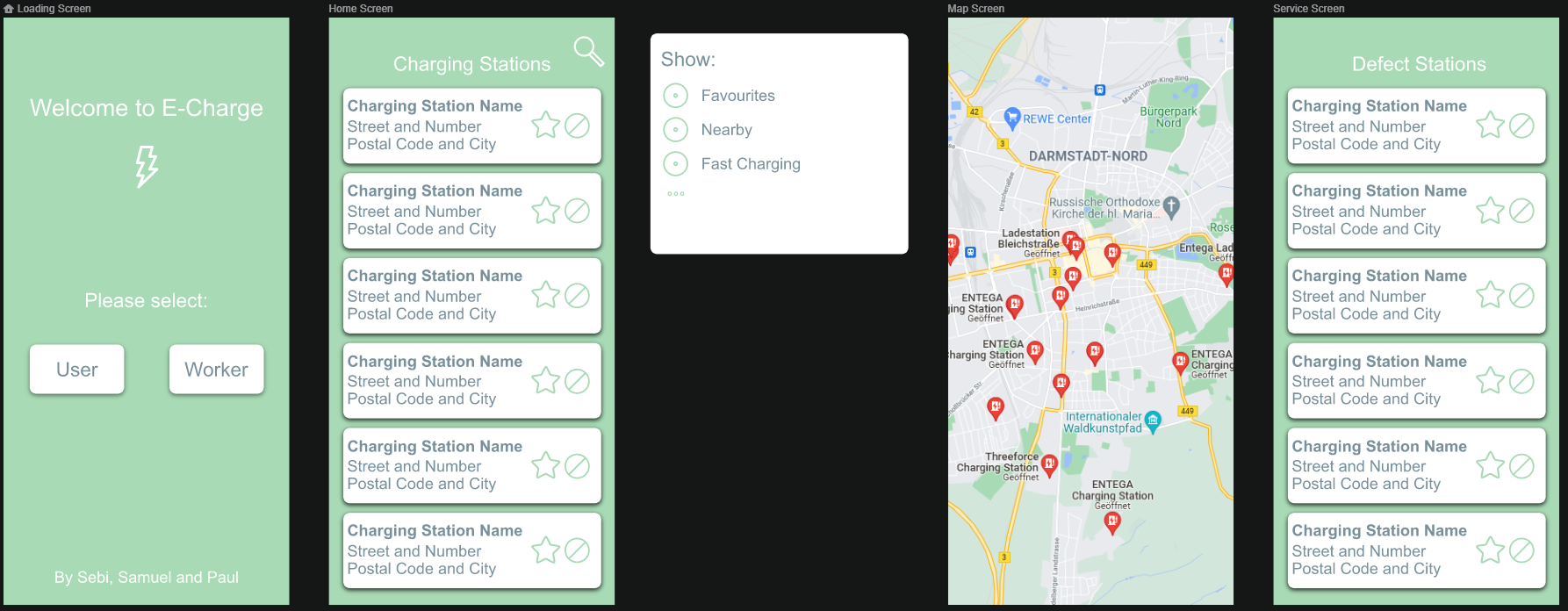


Abbildung 3: Erstes Design Konzept

Beim Erstellen des ersten Designs kam die Idee auf dem User Zugang zu zwei Screens zu ermöglichen, dem Home Screen und der GoogleMap. Der Service-Techniker erhält die Zugriffsrechte auf einen zusätzlichen Service Screen, um *nur* die defekten Ladestationen angezeigt zu bekommen und bearbeiten zu können.

Die entsprechende Bottom-Navigation des allgemeinen Layouts ergänzt diese Designs am unteren Bildschirmrand und wurde zur Visualisierung der restlichen Inhalte hier weggelassen.

### GoogleMap

Auch wenn es unter den Projektanforderungen nicht gelistet ist, haben wir uns dazu entschieden, dem Nutzer eine Karte zur Verfügung zu stellen, um neben dem RecyclerView noch eine weitere bekannte Anzeigemöglichkeit für Ladestationen im Umkreis zu haben. Der Nutzer kann dadurch selbst entscheiden welche Darstellungsweise ihm besser zusagt und für seine Zwecke passender ist. Zusätzlich bietet die GoogleMap weitere Möglichkeiten für zukünftige Implementierungen (siehe Kapitel 5).

Die GoogleMap wurde in die App mittels eines eigenen Fragments implementiert. Hierfür wurde für die Seite ein eigenes Layout-Fragment **fragment\_map.xml** erstellt, welches sich über die gesamte Höhe und Breite erstreckt und über die Fragmentklasse **MapFragment.java** die Karte per inflate reingeladen und asynchron betrieben wird.

Bevor die Karte allerdings geladen und dieser Screen aufgerufen wird, muss eine Funktionalität zur Bestätigung der Standortabnahme des Gerätes hinzugefügt werden. Diese Bestätigung wird bei jedem Start der App in der MainActivity von den Android-Permissions im Manifest angefragt. Insofern diese vorhanden ist, muss der Nutzer nichts weiter tun und die App startet ordnungsgemäß. Sollte aber noch keine Erlaubnis vorliegen, würden Map und Umkreissuche nicht funktionieren und die Bestätigung wird von der App vor dem eigentlichen Start der restlichen Fragmente in der **MainActivity.java** angefragt. Der User erhält dann ein Pop-Up mit der Aufforderung den Standort freizugeben.

1. /\*\*
2. \* Function triggers on create
3. \* @param savedInstanceState
4. \*/
5. @Override
6. protected void onCreate(Bundle savedInstanceState) {
7. super.onCreate(savedInstanceState);
8. setContentView(R.layout.activity\_main);
10. if(ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS\_FINE\_LOCA TION) != PackageManager.PERMISSION\_GRANTED && ActivityCompat.checkSelfPermis sion(this, Manifest.permission.ACCESS\_COARSE\_LOCATION) != PackageManager.PER MISSION\_GRANTED){
11. ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.AC CESS\_FINE\_LOCATION, Manifest.permission.ACCESS\_COARSE\_LOCATION}, request Code);
12. if(ActivityCompat.checkSelfPermission(this, Manifest.permission.AC CESS\_FINE\_LOCATION) != PackageManager.PERMISSION\_GRANTED && Activi tyCompat.checkSelfPermission(this, Manifest.permission.AC CESS\_COARSE\_LOCATION) != PackageManager.PERMISSION\_GRANTED){
13. Toast.makeText(this , this.getResources().getString(R.string.err Location), Toast.LENGTH\_LONG).show();
14. }
15. }
17. bottomNav = findViewById(R.id.bottom\_navigation);
18. bottomNav.setOnItemSelectedListener(navListener);
20. StationAPI.initialize(getApplicationContext());
21. Toast.makeText(this, Integer.toString(GlobalStorage.getAllStations().size()), Toast.LENGTH\_SHORT).show();
23. getSupportFragmentManager().beginTransaction().replace(R.id.fragment\_container,
24. new LoginFragment()).commit();
26. bottomNav.setVisibility(View.INVISIBLE);
27. }

Codebeispiel 3: MainActivity.java

Wenn der Standort also bestätigt ist, lädt die Karte per Inflater in dessen View hinein und weitere Funktionalitäten konnten ergänzt werden.

So wurde die im nächsten Punkt beschriebene Umkreissuche für die GoogleMap in das **MapFragment.java** hinzugefügt. Diese Umkreissuche fungiert mit einem so genannten setOnMapClickListener per Klick auf einen bestimmten Punkt der Karte. Für diesen Punkt werden dann alle Standorte der Ladestationen innerhalb einer bestimmten Entfernung (später in den Filtereinstellungen von Punkt 4.9 vom Nutzer einstellbar) angezeigt, wobei Favoriten auch außerhalb dieser Entfernungen als Sonderstandorte angezeigt werden.

Zusätzlich wurden für die Standorte nach deren Status eigene Markierungen auf der Karte ergänzt:

* Rot: Defekt
* Gelb: Favorit
* Blau: Normal funktionstüchtig

Zu guter Letzt wurde noch für jede Station ein Pop-Up hinzugefügt, sodass detailliertere Informationen zu den angewählten Stationen nach Klick auf deren Markierungen angezeigt werden.



Abbildung 4: GoogleMap mit Markierungen

### Umkreissuche

Um sich nicht alle Ladestationen, sondern nur die relevanten in der Nähe anzeigen zu lassen, wird eine Umkreissuche implementiert. Eine Funktion zur Berechnung des Abstands zwischen zwei Geokoordinaten wird im Praktikumsblatt bereits gegeben:

1. public static ArrayList<Ladestation> getProximityStations(ArrayList<Ladestation> stationsList, double lat1, double lon1, int dist){
2. ArrayList<Ladestation> filtered = new ArrayList<>();
3. stationsList.forEach(ladestation -> {
4. double lat2 = ladestation.getLat();
5. double lon2 = ladestation.getLon();
6. double R = 6371; // Erdradius in km
7. double dLat = (lat1 - lat2) \* Math.PI / 180.0;
8. double dLon = (lon1 - lon2) \* Math.PI / 180.0;
9. double a = Math.sin(dLat / 2.) \* Math.sin(dLat / 2.)
10. + Math.cos(lat1 \* Math.PI / 180.0)
11. \* Math.cos(lat2 \* Math.PI / 180.0)
12. \* Math.sin(dLon / 2.) \* Math.sin(dLon / 2.);
13. double c = 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1. - a));
14. double d = R \* c;
15. // Compare distance
16. if(d < dist)
17. {
18. filtered.add(ladestation);
19. }
20. });
21. return filtered;
22. }

Codebeispiel 4: Berechnung Abstand zweier Geokoordinaten [5]

Mit Hilfe der Funktion lässt sich das Array aller Ladestationen filtern und ein neues gefiltertes Array mit den Ladestationen in einem bestimmten Umkreis zurückgeben. Die Funktion wurde anschließend für den RecyclerView verwendet und nachträglich als eigene Filterfunktion in die GoogleMap ergänzt.

### RecyclerView

Der RecyclerView ist explizit gefordert. Am einfachsten lässt er sich mit einer dynamischen Liste beschreiben, die nur die angezeigten Elemente kreiert und somit auch für große Datenmengen gut einsetzbar ist. [7]

Für die Implementierung wird eine Adapter Klasse **StationAdapter.java** benötigt, der die Stationen in das  
RecyclerView fragment der Liste ergänzt:

1. public class StationAdapter extends RecyclerView.Adapter<StationAdapter.ViewHolder>{
2. private ArrayList<Ladestation> stationList;
3. private SQLiteHelper sqLiteHelper;
5. public StationAdapter(ArrayList<Ladestation> stationList, SQLiteHelper sqLiteHelper)
6. {
7. this.stationList = stationList;
8. this.sqLiteHelper = sqLiteHelper;
9. }
11. @NonNull
12. @Override
13. public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
14. Context context = parent.getContext();
15. LayoutInflater inflater = LayoutInflater.from(context);
16. View stationView = inflater.inflate(R.layout.list\_item, parent, false);
17. return new ViewHolder(stationView);
18. }
20. @SuppressLint("SetTextI18n")
21. @Override
22. public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
23. Ladestation ladestation = stationList.get(position);
24. holder.favouriteButton.setImageResource(R.drawable.ic\_baseline\_star\_25);
26. // Set top text
27. TextView stationTextTop = holder.stationTextTop;
28. stationTextTop.setText(ladestation.getOperator());
30. // Set middle text
31. TextView stationTextCenter = holder.stationTextCenter;
32. stationTextCenter.setText(ladestation.getStreet() +" "+ ladestation.getNumber());
34. // Set bottom text
35. TextView stationTextBottom = holder.stationTextBottom;
36. stationTextBottom.setText(ladestation.getPostalCode() + " " + ladestation.getLocation());
38. // Set station id
39. holder.stationID = ladestation.id;
41. // Buttons
42. ImageButton button = holder.favouriteButton;
43. if(GlobalStorage.getFavStations().contains(ladestation.id)){
44. button.setImageResource(R.drawable.ic\_baseline\_star\_24);
45. }
46. button.setOnClickListener(v -> toggleButtonFav(ladestation, button));
48. ImageButton buttonRep = holder.reportButton;
49. if(GlobalStorage.getDefStations().contains(ladestation.id)){
50. buttonRep.setImageResource(R.drawable.ic\_baseline\_flag\_25);
51. }
53. if(MainActivity.onServiceFragment){
54. buttonRep.setOnClickListener(v -> toggleButtonRepWorker(ladestation, buttonRep));
55. }
56. else{
57. buttonRep.setOnClickListener(v -> toggleButtonRep(ladestation, buttonRep));
58. }
59. }
60. @Override
61. public int getItemCount() {
62. return stationList.size();
63. }
65. public class ViewHolder extends RecyclerView.ViewHolder {
66. public TextView stationTextTop;
67. public TextView stationTextCenter;
68. public TextView stationTextBottom;
69. public ImageButton favouriteButton;
70. public ImageButton reportButton;
71. public int stationID;
73. public ViewHolder(View itemView) {
74. super(itemView);
75. stationTextTop = itemView.findViewById(R.id.text\_stationTop);
76. stationTextCenter = itemView.findViewById(R.id.text\_stationCenter);
77. stationTextBottom = itemView.findViewById(R.id.text\_stationBottom);
78. favouriteButton = itemView.findViewById(R.id.FavouriteButton);
79. reportButton = itemView.findViewById(R.id.ReportButton);
80. }
81. }
82. @SuppressLint("NotifyDataSetChanged")
83. public void setNewList(ArrayList<Ladestation> newList){
84. stationList = newList;
85. notifyDataSetChanged();
86. }
87. }

Codebeispiel 5: RecyclerView.java

Zunächst werden im Konstruktor des StationAdapters sowohl die in den vorherigen Schritten im GlobalStorage abgelegten Liste aller Stationen als auch die Anbindung an die persistenten Daten der SQLite (folgt in Kapitel 4.10) übergeben (siehe **RecyclerView.java** Zeile 2 - 9).

Anschließend wird im Abschnitt des onCreateViewHolder() das erstellte XML - Layout in den Screen geladen.

Der ViewHolder() bildet wiederrum ein Skelett für einen unbefüllten Listview, welcher dann anschließend mit Daten befüllt und eingeblendet wird.

Die eigentlich beschriebene Funktionalität des RecyclerViews, die Datensätze nach und nach anhand deren Position das Skelett zu befüllen und in den View zu laden, wird anschließend mit der Funktion onBindViewHolder() umgesetzt. Hier wird anhand der Scrollposition der zugehörige Datensatz in der Ladestationen-Liste mit .get(position) geladen und per Holder als Listenitem in der Anzeige angehängt.

### Favoriten

Eine weitere User Story ist die Möglichkeit Favoriten hinzuzufügen. Dadurch erkennt der Nutzer nicht nur seine persönlichen Lieblingsladestationen auf einen Blick, sondern hat zusätzlich noch die Möglichkeit die App zu seiner persönlichen zu machen. Jeder Nutzer kann seine eigenen Favoriten setzen, wodurch jeder auch seine App für sich gestaltet.

Der Button für das Hinzufügen der Favoriten wird bereits als „favouriteButton“ aus Codebeispiel 2 im RecyclerView (Zeile 42 - 46) implementiert. Gekennzeichnet ist dieser mit einem gelben Stern, um dem Nutzer auch ohne Beschreibung eine eindeutige Zuweisung zu geben.

Diese Einstellungen werden dann wie in Kapitel 4.10 beschrieben in einer App-internen SQLite Datenbank direkt auf dem Gerät gespeichert, sodass sie zu einem späteren App Start wieder zur Verfügung stehen.

### Defekte melden

Damit kein Nutzer eine Ladestation anfährt, die gerade außer Betrieb ist, wird eine weitere Funktion implementiert, um defekte Ladestationen zu melden. Zusätzlich bietet dies die Möglichkeit, dass Service Techniker defekt gemeldete Stationen anfahren und reparieren können.

Es entsteht ein Netzwerk an Nutzern, von dem jeder Nutzer anhand der Aktivität anderer profitiert.

Wie auch der Favoriten Button, wird der Button für das Melden als defekt bereits als „reportButton“ aus Codebeispiel 2 im RecyclerView (Zeile 48 - 51) implementiert. Gekennzeichnet ist dieser mit einer roten Flagge, um den Nutzer auch ohne Beschreibung eine eindeutige Zuweisung zu geben.

Wird der Button gedrückt, erscheint ein Pop Up, welches den Nutzer auffordert einen Grund für den Defekt anzugeben. Diesen Grund kann der Service Mitarbeiter für seine Problemanalyse angezeigt bekommen und nutzen.

### Filter Funktionen

Eine ungeordnete Liste aller Ladestationen bringt dem individuellen Nutzer nicht viel. Er möchte nur bestimmte für den eigenen Zweck angezeigt bekommen. Die wichtigsten Kriterien hierbei sind:

* Seine persönlichen Favoriten angezeigt zu bekommen.
* Nur Ladestationen in einem bestimmten Umkreis darstellen.

Der RecyclerView soll also gefiltert werden. Mit einem Klick auf die Lupe am oberen rechten Bildschirmrand erhält der Nutzer ein Pop Up mit den verschiedenen Filtermöglichkeiten (siehe Abbildung 5).

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 5: Filter Funktionen Pop Up

Die Favoriten und Schnelladesäulen lassen sich mit einer Checkbox anzeigen und der Umkreis lässt sich mit Hilfe eines Sliders (Seekbar) auf die gewünschten Kilometer einstellen. Dazu wird die bereits implementierte Umkreissuche aus Abschnitt 4.5 benutzt. Diese Filterfunktionen können auf die Liste beliebig kombiniert angewendet werden.

Um die Filterfunktionen als Pop-Up über dem eigentlichen Screen zu verwenden, wurde eine eigenes Layout und eine eigene Klasse **PopUpService.java** erstellt, die das Design dort drüber aufbaut. Zudem wurde im Zusammenspiel mit dieser Klasse eine weitere Klasse **SaveCheckedState.java** verwendet, um die getätigten Einstellungen in den Checkboxen und dem Slider vom Nutzer im Pop-Up auch nach schließen und öffnen dessen zu speichern.

Demnach sind in der **SaveCheckedState.java** lediglich drei Variablen (showFavourites, showFastCharging, searchRange) mit Gettern und Settern verwaltet. Diese Variablen speichern entsprechende Einstellungen im Pop-Up zwischen und werden im PopUpService bei jedem Aufruf entsprechend wieder hergestellt.

Die Klasse **PopUpService.java** schaut und verwaltet entsprechend die Filtereinstellungen und gleicht diese mit der SaveCheckedState Klasse in der Funktion showPopup() ab. Anschließend übergibt der Service an den RecyclerView eine neue Liste, die anhand der Filtereinstellungen neu gefiltert wurden. Dies passiert in der Funktion calculateThings().

1. public class PopUpService {
2. private final Context context;
3. private StationAdapter stationAdapter;
4. private AlertDialog dialog;
5. private SaveCheckedState saveCheckedState;
7. // set necessary vars within constructor
9. /\*\*
10. \* Constructor
11. \* @param context The context
12. \* @param stationAdapter the current station adapter
13. \* @param saveCheckedState the savestate
14. \*/
15. public PopUpService(Context context, StationAdapter stationAdapter, SaveCheckedState saveCheckedState) {
16. this.context = context;
17. this.stationAdapter = stationAdapter;
18. this.saveCheckedState = saveCheckedState;
19. }

22. // calculate filtered stations from popup to display in recyclerview list
24. /\*\*
25. \* calculate filtered stations from popup to display in recyclerview list
26. \*/
27. private void calculateThings(){
28. String favourites = this.context.getResources().getString(R.string.favourites);
29. String fastCharging = this.context.getResources().getString(R.string.fastCharging);
30. String range = this.context.getResources().getString(R.string.range);
32. // Ini list of filtered stations and deleted stations (those will be removed from full list later on, because java does not allow remove in for)
33. ArrayList<Ladestation> finalStations = new ArrayList<>();
34. ArrayList<Ladestation> toDel = new ArrayList<>();
36. // first add all stations to list
37. finalStations.addAll(GlobalStorage.getAllStations());
39. // check favourites filter
40. if(saveCheckedState.isShowFavourites()){
41. for(Ladestation ladestation : finalStations){
42. if(!GlobalStorage.getFavStations().contains(ladestation.id)){
43. toDel.add(ladestation);
44. }
45. }
46. }
47. // check fastcharging filter
48. if(saveCheckedState.isShowFastCharging()){
49. for(Ladestation ladestation : finalStations){
50. if(ladestation.getModuleType() != Ladestation.ModuleType.FAST\_CHARGING){
51. if(!toDel.contains(ladestation)){
52. toDel.add(ladestation);
53. }
54. }
55. }
56. }
57. // check seekbar range filter
58. if(saveCheckedState.getSearchRange() != 100){
59. ArrayList<Ladestation> rangeStations = StationAPI.getProximityStations(final Stations, MainActivity.userLocation.latitude, MainActivity.userLocation.longitude, saveCheckedState.getSearchRange());
60. for(Ladestation ladestation : finalStations){
61. if(!rangeStations.contains(ladestation)){
62. toDel.add(ladestation);
63. }
64. }
65. }
67. // delete stations outside of filters in delarray from finalarry
68. for(Ladestation ladestation : toDel){
69. finalStations.remove(ladestation);
70. }
72. // clear delarray for next filtering
73. toDel.clear();
75. // set filtered finalarray as new list in listview
76. stationAdapter.setNewList(finalStations);
77. }

80. /\*\*
81. \* function for popup toggle and listeners
82. \*/
83. void showPopup() {
84. String range = this.context.getResources().getString(R.string.range);
85. String unlimited = this.context.getResources().getString(R.string.unlimited);
87. // inflate popup above current listview
88. AlertDialog.Builder builder = new AlertDialog.Builder(context);
89. View view = LayoutInflater.from(context).inflate(R.layout.options\_popup, null);
91. // check favourites checkbox
92. CheckBox checkBoxFav = view.findViewById(R.id.FavToggleButton);
93. if(saveCheckedState.isShowFavourites()){
94. checkBoxFav.setChecked(true);
95. }
96. else {
97. checkBoxFav.setChecked(false);
98. }
100. // check fastcharging checkbox
101. CheckBox checkBoxFastCharging = view.findViewById(R.id.FastChargingToggleButton);
102. if(saveCheckedState.isShowFastCharging()){
103. checkBoxFastCharging.setChecked(true);
104. }
105. else {
106. checkBoxFastCharging.setChecked(false);
107. }
109. // check and handle seekbar range filters
110. SeekBar seekBar = view.findViewById(R.id.seekBar);
111. TextView tvProgressLabel = view.findViewById(R.id.seekBarText);
112. seekBar.setProgress(saveCheckedState.getSearchRange());
113. if(seekBar.getProgress() == 100){
114. tvProgressLabel.setText(range + ": " + unlimited);
115. }
116. else{
117. tvProgressLabel.setText(range + ": " + seekBar.getProgress() + "km");
118. }

121. // set listeners in according sequence
122. checkBoxFav.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
123. @SuppressLint("NotifyDataSetChanged")
124. @Override
125. public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
126. saveCheckedState.setShowFavourites(isChecked);
127. if(checkBoxFastCharging.isChecked()){
128. saveCheckedState.setShowFastCharging(true);
129. }
130. calculateThings();
131. }
132. });
133. checkBoxFastCharging.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
134. @Override
135. public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
136. saveCheckedState.setShowFastCharging(isChecked);
137. if(checkBoxFav.isChecked()){
138. saveCheckedState.setShowFavourites(true);
139. }
140. calculateThings();
141. }
142. });
143. seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
144. @Override
145. public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
146. if(seekBar.getProgress() == 100){
147. tvProgressLabel.setText(range + ": " + unlimited);
148. }
149. else{
150. tvProgressLabel.setText(range + ": " + seekBar.getProgress() + "km");
151. }
152. }
154. @Override
155. public void onStartTrackingTouch(SeekBar seekBar) {}
157. @Override
158. public void onStopTrackingTouch(SeekBar seekBar) {
159. saveCheckedState.setSearchRange(seekBar.getProgress());
160. calculateThings();
161. }
162. });
164. builder.setView(view);
165. builder.setCancelable(true);
167. dialog = builder.show();
168. dialog.getWindow().setBackgroundDrawableResource(R.color.transparent);
169. }
170. }

Codebeispiel 6: PopUpService.java

### Persistente Daten

Bisher ist nach dem Beenden der App alles wieder auf Anfang gesetzt. Um auch nach erneutem Starten der App die Einstellungen zu Favoriten und Defekten Stationen zu erhalten, wird eine SQLite Datenbank über die Klasse **SQLiteHelper.java** implementiert. Dabei wird sich an den Codebeispielen aus dem Moodle-Kurs von Prof. Dr. Wiedling und der Android Studio Dokumentation [8] orientiert.

Die Datenbank speichert die ID’s der Favoriten und defekten Stationen in jeweils eigenen Tabellen und lädt diese zu App Start aus. Hierfür wurde sowohl für Favoriten, als auch die Defekte ein trio an Funktionen implementiert:

* saveSingleFavoriteToDB(), saveSingleFlagToDB(): Speichert die ID der entsprechend als Favorit oder Defekt markierten Ladestation

1. public void saveSingleFavoriteToDB(int id) // save single value
2. {
3. SQLiteDatabase database = this.getWritableDatabase();
4. ContentValues values = new ContentValues();
5. values.put(ATTRIBUTE\_ID, id); // Write id to value set
6. database.insert(FAVOURITE\_TABLE\_NAME, null, values); // Write the valueset to DB
7. database.close();
8. }
9. public void saveSingleFlagToDB(int id) // save single value
10. {
11. SQLiteDatabase database = this.getWritableDatabase();
12. ContentValues values = new ContentValues();
13. values.put(ATTRIBUTE\_ID, id); // Write id to value set
14. database.insert(REPORTS\_TABLE\_NAME, null, values); // Write the valueset to DB
15. database.close();
16. }

Codebeispiel 7: Favoriten oder Defekte in Datenbank schreiben

* removeSingleFavoriteFromDB(), removeSingleReportFromDB(): Löscht die entsprechende ID der als Favorit oder Defekt entfernten Ladestation

1. public void removeSingleFavoriteFromDB(int id)
2. {
3. SQLiteDatabase database = this.getWritableDatabase();
4. database.delete(SQLiteHelper.FAVOURITE\_TABLE\_NAME, "id = ?",new String[] {String.valueOf(id)});
5. database.close();
6. }
7. public void removeSingleReportFromDB(int id)
8. {
9. SQLiteDatabase database = this.getWritableDatabase();
10. database.delete(SQLiteHelper.REPORTS\_TABLE\_NAME, "id = ?",new String[] {String.valueOf(id)});
11. database.close();
12. }

Codebeispiel 8: Favoriten oder Defekte aus Datenbank entfernen

* readFavouritesFromDB(), readFlagsFromDB(): Liest alle IDs von entsprechenden Favoriten oder als Defekt markierten Ladestationen in der lokalen SQLite Datenbank aus und gibt diese als Array zur Verarbeitung der Instanz zurück.

1. public ArrayList<Integer> readFavouritesFromDB() {
2. SQLiteDatabase database = this.getReadableDatabase(); // Get reference to db
3. String query = "SELECT \* FROM "+SQLiteHelper.FAVOURITE\_TABLE\_NAME;
4. Cursor res = database.rawQuery(query,null);
5. ArrayList<Integer> resultList = new ArrayList<Integer>();
7. if(res.moveToFirst())
8. {
9. do // Loop over all results
10. {
11. resultList.add(Integer.parseInt(res.getString(0)));
12. } while(res.moveToNext());
13. }
14. res.close();
15. database.close();
16. for(Integer i : resultList)
17. {
18. System.out.println(i);
19. }
20. return resultList;
21. }
22. public ArrayList<Integer> readFlagsFromDB() {
23. SQLiteDatabase database = this.getReadableDatabase(); // Get reference to db
24. String query = "SELECT \* FROM "+SQLiteHelper.REPORTS\_TABLE\_NAME;
25. Cursor res = database.rawQuery(query,null);
26. ArrayList<Integer> resultList = new ArrayList<Integer>();
28. if(res.moveToFirst())
29. {
30. do // Loop over all results
31. {
32. resultList.add(Integer.parseInt(res.getString(0)));
33. }while(res.moveToNext());
34. }
35. res.close();
36. database.close();
37. for(Integer i : resultList)
38. {
39. System.out.println(i);
40. }
41. return resultList;
42. }

Codebeispiel 9: Favoriten oder Defekte aus Datenbank lesen

### Login methoden

Nicht nur „normale“ Nutzer, sondern auch Service Techniker sollen Zugang zu der App haben. Da Service Techniker jedoch andere Rechte haben als Nutzer, benötigt man verschiedene Logins. Der Service Techniker erhält einen zusätzlichen Screen, auf dem *nur* defekte Ladestationen angezeigt werden. Zusätzlich hat er die Möglichkeit, sich den Grund für den Defekt anzuschauen und nach erfolgreicher Reparatur die Ladestation wieder freizugeben.

Umgesetzt wurde die Benutzerunterscheidung über ein eigenes Loginfragment, welches beim Appstart vor allen anderen Screens aufgerufen und angezeigt wird.

Je nach gewähltem Nutzer wird über die Klasse **LoginFragment.java** das Layout der Bottom-Navigation um den Punkt des Service Screens ergänzt oder dieser ausgeblendet, sodass er vom normalen Nutzer nicht angesteuert wird. Hierfür wurden im Menü zwei Layouts der Bottom-Navigation angelegt: **bottom\_navigation.xml** und **bottom\_navigation\_worker.xml**.

1. public class LoginFragment extends Fragment {
3. /\*\*
4. \* @param inflater
5. \* @param container
6. \* @param savedInstanceState
7. \* @return view
8. \*/
9. public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
10. View view = inflater.inflate(R.layout.fragment\_login, container, false);
12. // USER
13. Button userButton = view.findViewById(R.id.UserButton);
14. userButton.setOnClickListener(v -> {
15. MainActivity.getBottomNav().inflateMenu(R.menu.bottom\_navigation);
16. getParentFragmentManager().beginTransaction().replace(R.id.fragment\_container, new MapFragment()).commit();
17. MainActivity.getBottomNav().setSelectedItemId(R.id.nav\_map);
18. MainActivity.getBottomNav().setVisibility(View.VISIBLE);
19. });
21. // WORKER
22. Button workerButton = view.findViewById(R.id.WorkerButton);
23. workerButton.setOnClickListener(v -> {
24. MainActivity.getBottomNav().inflateMenu(R.menu.bottom\_navigation\_worker);
25. getParentFragmentManager().beginTransaction().replace(R.id.fragment\_container, new MapFragment()).commit();
27. MainActivity.getBottomNav().setSelectedItemId(R.id.nav\_map);
28. MainActivity.getBottomNav().setVisibility(View.VISIBLE);
29. });
31. return view;
32. }
33. }

Codebeispiel 10: LoginFragment.java

Ein Bild, das Text, Screenshot, Elektronik enthält.

Automatisch generierte Beschreibung

Abbildung 6: Login Screen

### Mehrsprachigkeit

Eine weitere gestellte Projektanforderung ist die Mehrsprachigkeit. Somit haben Nutzer die Möglichkeit die App in ihrer eingestellten Systemsprache zu verwenden. Realisiert wird dies mit **string.xml** Dateien für die jeweilige Sprache. Dazu wird zunächst jeder Text, der auf dem Bildschirm angezeigt werden kann als String Variable deklariert. Anschließend wird jeder Begriff übersetzt und in einer Tabelle abgespeichert. Bei App Start wird die Tabelle der eingestellten Systemsprache verwendet und jeder Text auf dem Bildschirm in der Systemsprache angezeigt. Ein Auszug davon ist in Abbildung 7 zu sehen.

Ein Bild, das Text, Gedenktafel enthält.

Automatisch generierte Beschreibung

Abbildung 7: Tabelle Übersetzungen

Ein Beispiel zur Verwendung der **string.xml** zur Generierung von Texten der entsprechenden Sprache können in der PopUpService Klasse eingesehen werden:

1. String range = this.context.getResources().getString(R.string.range);
2. String unlimited = this.context.getResources().getString(R.string.unlimited);
3. if(seekBar.getProgress() == 100){
4. tvProgressLabel.setText(range + ": " + unlimited);
5. }
6. else{
7. tvProgressLabel.setText(range + ": " + seekBar.getProgress() + "km");
8. }

Codebeispiel 11: Multilanguage Support Strings

Das Hinzufügen von weiteren Sprachen ist somit kein großer Mehraufwand. Es wird lediglich eine weitere Spalte mit der jeweiligen Sprache angelegt und die Übersetzungen in die Zeilen eingetragen.

### Landscape Modus

Der Landscape Modus ist ein Feature, das dem Nutzer mehr Möglichkeiten und Freiraum bietet, die App nach seinen persönlichen Präferenzen zu nutzen. Beim Drehen des Endgerätes schaltet die App automatisch in den Landscape Modus, sodass auch beim seitlichen Halten des Smartphones alle Elemente weiterhin auf dem Screen angezeigt werden und die App voll benutzbar bleibt.

Für die Implementierung wird ein Parameter in **AndroidManifest.xml** benötigt (Codebeispiel 3 Zeile 27):

1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3. package="com.example.myapplication">
5. <uses-permission android:name="android.permission.INTERNET"/>
6. <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>
7. <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>
8. <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />
9. <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />
11. <application
12. android:allowBackup="true"
13. android:icon="@mipmap/ic\_launcher"
14. android:label="E-Charge"
15. android:roundIcon="@mipmap/ic\_launcher\_round"
16. android:supportsRtl="true"
17. android:theme="@style/Theme.MyApplication"
18. >
20. <meta-data
21. android:name="com.google.android.geo.API\_KEY"
22. android:value="AIzaSyBtXnHz1RzepQ0B6tg4bbBP27UIl5aEgaw"/>
24. <activity
25. android:name=".MainActivity"
26. android:exported="true"
27. android:screenOrientation="sensor">
28. <intent-filter>
29. <action android:name="android.intent.action.MAIN" />
31. <category android:name="android.intent.category.LAUNCHER" />
32. </intent-filter>
33. </activity>
34. <service
35. android:name=".RequestService"
36. android:exported="false" />
37. </application>
38. </manifest>

Codebeispiel 12: AndroidManifest.xml Screen Orientation

Zusätzlich müssen für die verschiedenen Screens Landscape Varianten erstellt werden (siehe Abbildung 8).

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 8: Landscape Screens

Anschließend wählt die App automatisch das passende Layout je Drehrichtung des Smartphones.

### Feinschliff

Nach Implementierung der geforderten und festgelegten Features, wird die App mehreren Personen, die nicht an der Entwicklung beteiligt sind, zum Testen gegeben. Dafür werden den Personen die folgenden Aufgaben gegeben (als Nutzer):

* Logge dich als Nutzer ein.
* Finde Ladestationen in einem beliebigen Umkreis zu dir.
* Finde Ladestationen auf der Karte in einem Umkreis zu dir.
* Füge einen oder mehrere Favoriten hinzu.
* Melde eine Ladestation als defekt und gebe einen Grund an.

Und als Service Techniker:

* Logge dich als Service Techniker ein.
* Finde Ladestationen, die als defekt gemeldet wurden.
* Lasse dir nur defekte Ladestationen in deinem Umkreis anzeigen.
* Sehe dir den Grund für einen Defekt an.
* Gebe die Ladestation wieder frei.

Die Nutzer konnten zu großem Teil alle ihnen gestellten Aufgaben erledigen. Speziell der Login und das Filter Pop-Up über Lupen Icon wurden über die Aussagekräftigen Icons von allen Personen schnell gefunden.

Weiterhin konnten Nutzer im RecyclerView gut nachvollziehen, welche Icons für Favoriten- und Defekt-  
Markierungen zuständig sind und diese Einstellungen entsprechend setzen und filtern.

Ein Problem, welches sich im RecyclerView stellte war die ebenfalls von uns implementierte „Neu-Laden“-Funktion, wenn man über den obersten Eintrag der Liste hinausscrollt. Das Neu-Laden Icon, welches auftaucht, steht nämlich ohne Animation still, was dem Nutzer den Eindruck eines Absturzes vermittelt. Dieses Problem lässt sich beheben, indem man die „Neu-Laden“-Funktion in einen eigenen asynchronen Thread verlagert.

Ansonsten war der RecyclerView für alle Nutzer einleuchtend und gut zu bedienen.

Eine weitere Problematik ergab sich aus der Bedienung der Google Karte, da ein Zoom zu nah an die Karte auf den eigenen Standort eventuell darin resultiert, dass nach Klick zum Anzeigen von Ladestationen im eigenen Umfeld keine Ladestation nah genug ist, als dass sich auf der Karte etwas verändert und Stationen angezeigt werden. Hierfür könnte man als Lösung eine Funktion zum Rauszoomen implementieren, sollten keine Stationen im eigenen View liegen.

Zudem sollte es möglich sein über Anwählen einer Ladestationen diese direkt auf der Karte als Favorit oder Defekt markieren zu können.

Ein weiterer Feature-Wunsch, der sich implementieren lässt, war, dass Nutzer eine Verlinkung zur Routenführung über Google Maps von der ausgewählten Station erhalten.

Anhand der Weitergabe unserer App hatten wir uns also in der Orientierung und Umsetzung der User Stories bestätigt gefühlt, jedoch auch festgestellt, dass durch die Individualität der Nutzer immer neue Featureanfragen entstehen, sowie Weiterentwicklungen und Verbesserungen möglich sind.

## Ausblick

Beim Testen der App mit den Probanten wurde deutlich, dass die GoogleMap noch intensiver in die App eingebunden werden sollte. Insbesondere das Feature „Favoriten setzen“ auf der GoogleMap wurde vermisst.

Wie bereits in Abschnitt 4.14 aufgegriffen, sollte der Zoom automatisch angepasst werden, sobald der Nutzer zu nah herangezoomt hat, dass keine Ladestationen im Umkreis mehr sichtbar sind.

Die „Neu-Laden“-Funktion des RecyclerViews sollte künftig in einem asynchronen Thread verlagert werden, um nicht den Anschein zu erwecken, dass die App abgestürzt ist.

Es könnten weitere Filtermöglichkeiten, wie zum Beispiel Steckertyp, Hersteller etc. hinzugefügt werden, um die Erfahrung des Nutzers weiter zu verbessern. Durch die bereits vorhandenen Daten lässt sich dieses Feature leicht integrieren.

Des Weiteren lässt sich mit wenigen Handgriffen der Login verbessern. Da die Datenbank bereits angebunden wurde lassen sich Nutzerdaten verschlüsselt darin speichern, sodass jeder Nutzer einen eigenen Zugang hat und Service Techniker noch besser von normalen Nutzern getrennt sind.

## Fazit

Der Umgang mit Java und Android Studio war für uns alle zunächst neu. Daher wurden die ersten Wochen vor allem für die Grundlagen in der IDE genutzt. Das Verwenden von XML-Dateien, die Implementierung von Adaptern, Listenern, Layouts etc. war ungewohnt, wurde aber mit Hilfe des Skiptes von Prof. Dr. Wiedling, Internetquellen und YouTube-Videos verstanden. Sobald die Grundfunktionen verstanden wurden, ist der Workflow deutlich angenehmer, wodurch auch Umsetzung und Planung der gewünschten Funktionen besser werden.

Sowohl beim Design als auch bei Umsetzung der Projektanforderungen standen immer die User Stories im Vordergrund. So haben wir auf ein aufwendiges, verschachteltes Design verzichtet, um auch ohne Beschreibungen dem Nutzer die Möglichkeit zu geben alle Funktionen einfach zu finden und keine Fragen offenbleiben. Alle Buttons wurden entsprechend so gestaltet, dass sie ohne Erklärung aussagekräftig und eindeutig sind.

Alles in allem war es ein spannendes Projekt, bei dem wir viel gelernt haben. Generell die Gelegenheit an einem größeren Projekt, das über das gesamte Semester geht, zu arbeiten, gefiel uns sehr gut. Wir konnten uns in der Gruppe selbst organisieren und mit verschiedenen Tools wie Discord und GitLab super verständigen.

## Quellen

[1] <https://www.youtube.com/watch?v=tPV8xA7m-iw>

[2] <https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView>

[3] <https://www.geeksforgeeks.org/difference-between-a-fragment-and-an-activity-in-android/>

[4] <https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Energie/Unternehmen_Institutionen/E_Mobilitaet/Ladesaeulenregister_CSV.csv?__blob=publicationFile&v=27>

[5] <https://api.aurora-theogenia.de/chargingstations/charging_stations.json>

[6] <http://www.movable-type.co.uk/scripts/latlong.html>

[7] <https://developer.android.com/guide/topics/ui/layout/recyclerview>

[8] <https://developer.android.com/training/data-storage/sqlite>

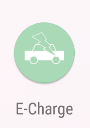
[9] Absprachen mit Prof. Dr. Wiedling und Herr Quick

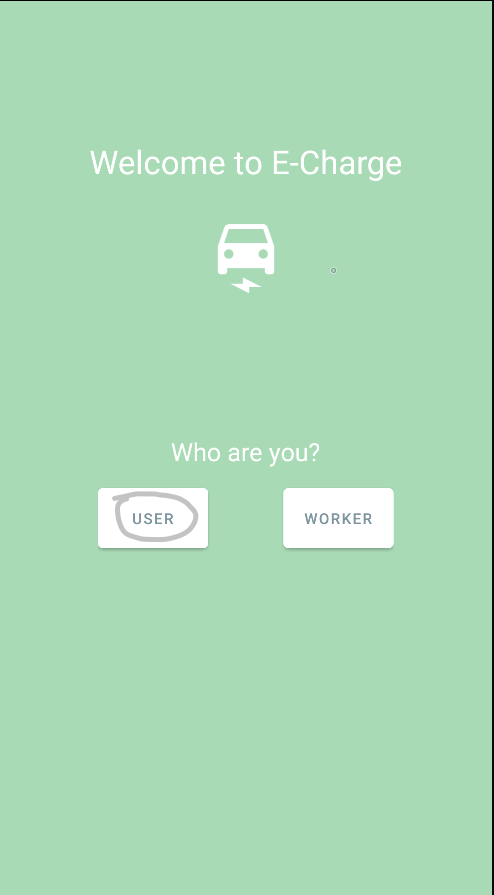
A.1 Eigenständigkeitserklärung:

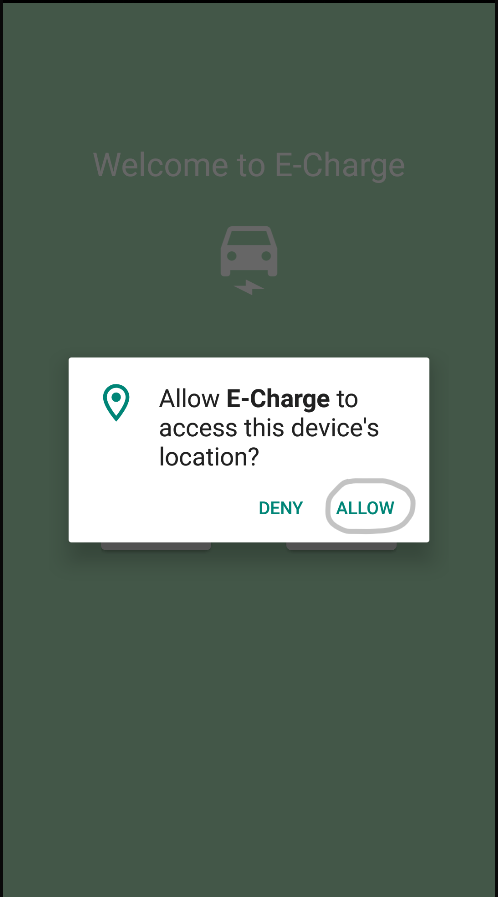
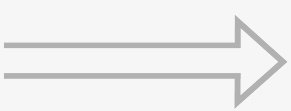
„Wir versichern hiermit, dass wir die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

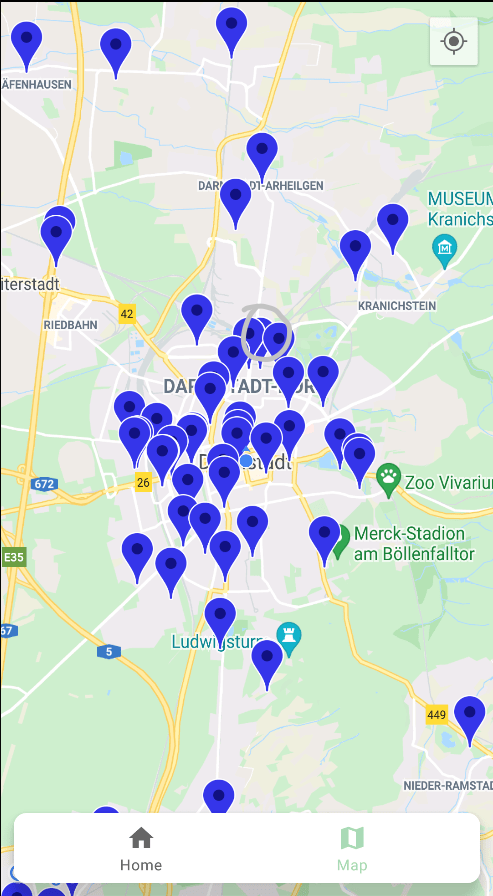
Die Zeichnungen oder Abbildungen in dieser Arbeit sind von uns selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.“

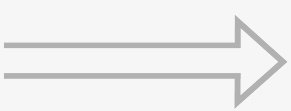
A.2 App Durchlauf   
anhand Bilderreihe

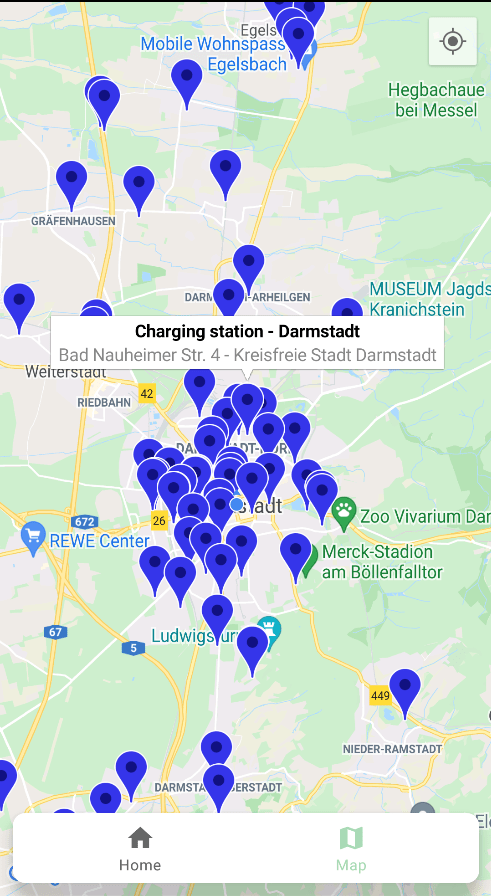
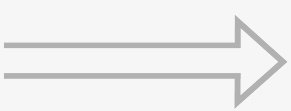




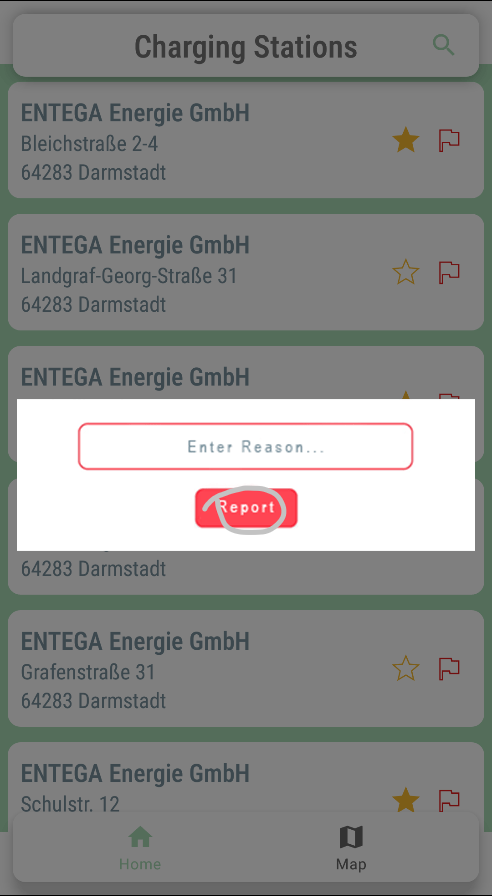
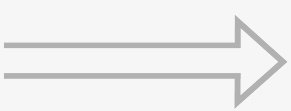


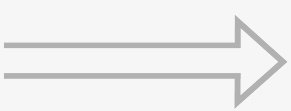




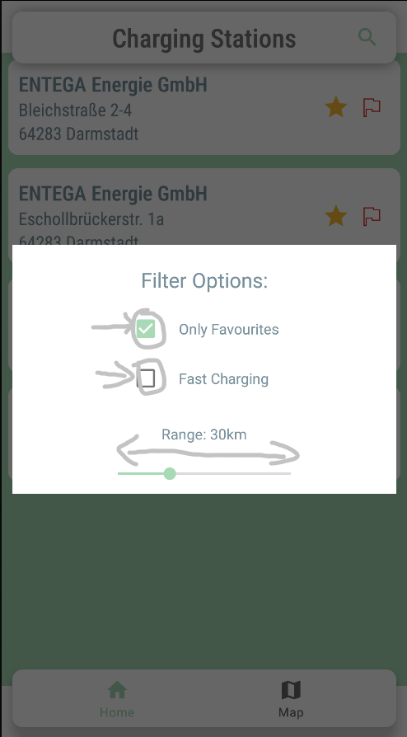
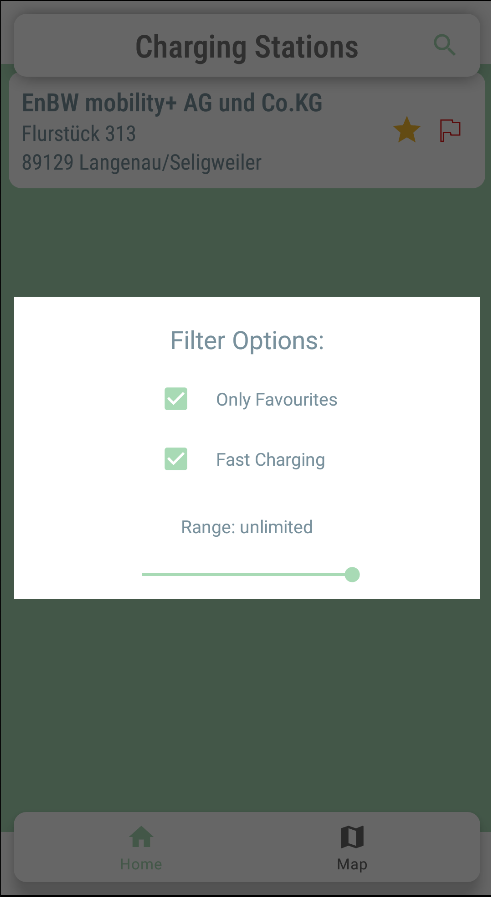
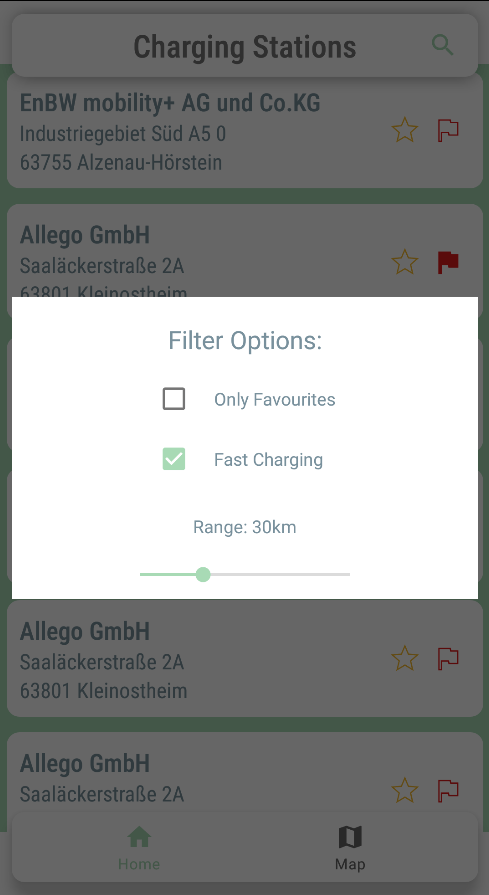


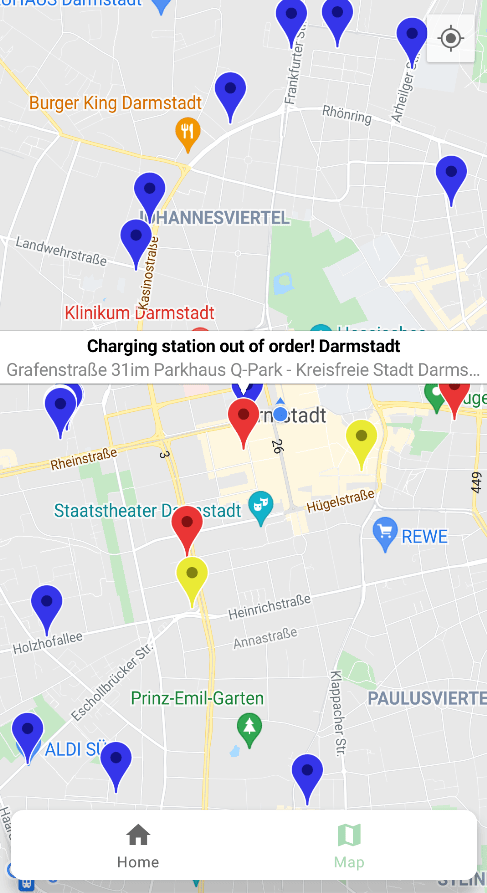
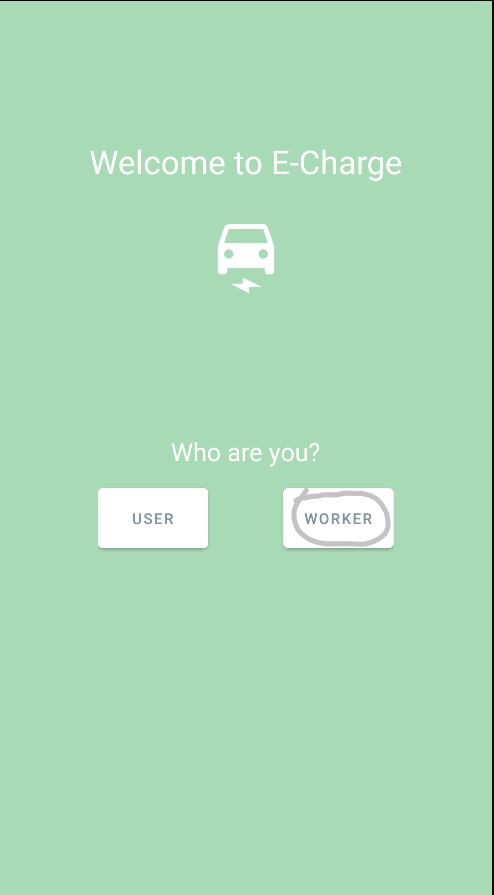
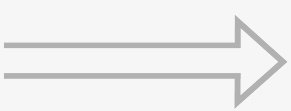




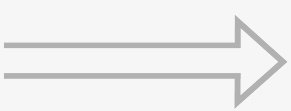


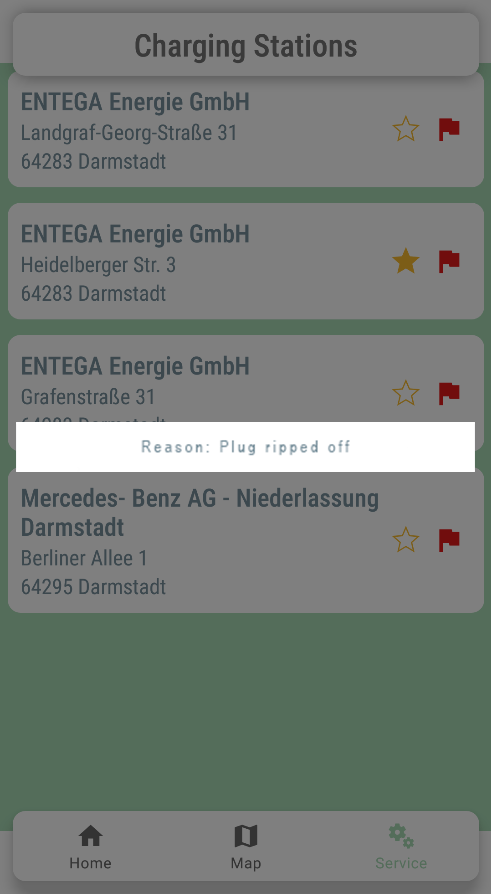
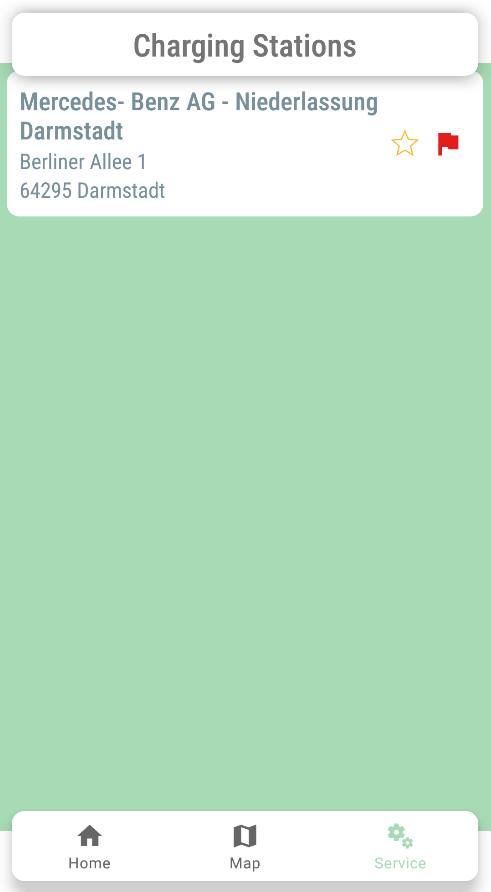
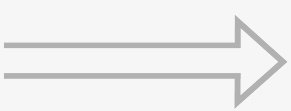


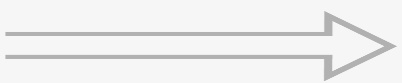


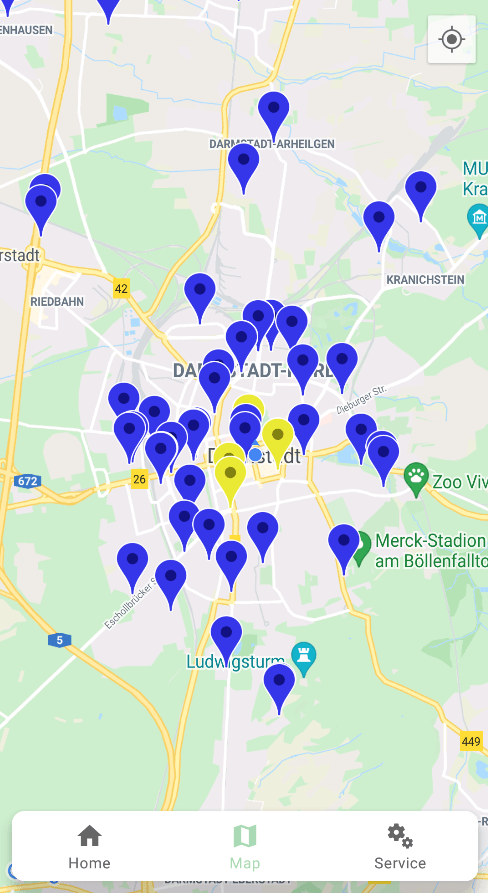












A.3 Doxygen Java Dokumenation

E Charge

Generated by Doxygen 1.9.3