

Projektdokumentation eFinder

im Rahmen des Fachs Nutzerzentrierte Softwareentwicklung
am Fachbereich Informatik
der Hochschule Darmstadt

von: Lukas Räßle & Etienne Gotha
Dozent: Prof. Dr. Hans-Peter Wiedling

Inhaltsverzeichnis

Abbildungsverzeichnis	III
1 Einführung	1
1.1 Projektbeschreibung	1
1.2 Vorgehensweise	1
2 Analyse	2
2.1 Heuristische Anforderungsanalyse	2
3 Design	4
3.1 User Story Mapping	4
3.2 Persona	4
3.3 Paper-Prototyping	4
4 Implementierung	4
4.1 Softwaredesign	4
4.1.1 Klassen & Kopplung	4
4.2 Umsetzung der Screens und Funktionalitäten	4
4.2.1 Datenbankintegration	4
4.2.2 Registrierung & Login	5
4.2.3 Navigationsleiste	7
4.2.4 Map-Screen	10
4.2.5 Suchen-Screen	10
4.2.6 Admin-Screen	10
4.2.7 Favoriten-Screen	10
4.2.8 Einstellungen-Screen	10
4.2.9 Mehrsprachigkeit & Landscape-Mode	10
5 Test	10
5.1 Funktionale Tests	10
5.2 Usability-Test	10
6 Fazit	11
7 Literaturverzeichnis	11
A Anhang	1
B Eigenständigkeitserklärung	2

Abbildungsverzeichnis

1	Generierung der Datenbankreferenzen	5
2	Login-Screen	5
3	Abfrage der existierenden Authentifizierungsinstanz	6
4	Authentifizierung des Benutzers	6
5	Registrierungs-Screen	7
6	Überprüfung des User-Input	8
7	Anlegen eines neuen Benutzers	8
8	Nutzer im Authentifizierungsbereich der Firebase	8
9	Nutzer in der Realtime-Database	9
10	Navigationsleiste	9
11	Initialisierung der Navigationsleiste in der Main-Activity	9
12	Host-Fragment in der XML der Main Activity	10

1 Einführung

Das vorliegende Projekt entstand im Rahmen des Praktikums zur Veranstaltung Nutzerzentrierte Softwareentwicklung im Wintersemester 2021/2022 an der Hochschule Darmstadt. Im Folgenden wird der Ablauf des Projektes von der Konzeption bis zur Auswertung der Ergebnisse dargestellt.

1.1 Projektbeschreibung

Gegenstand des Projektes ist die Entwicklung einer prototypischen Anwendung, die verschiedenen Nutzern den Zugang zu Informationen und Diensten bezüglich Ladestationen für Elektrofahrzeuge bereitstellt. Als Entwicklungsplattform wird Android Studio mit der Programmiersprache Java verwendet. Die Applikation soll sich dabei gleichermaßen an Fahrer von Elektrofahrzeugen als auch an Servicekräfte/Techniker, die für den Betrieb der Ladestationen zuständig sind, richten. Ziel des Projektes ist es einen durchgehend auf diese Nutzergruppen ausgerichteten Entwicklungsprozess zu durchlaufen, um ein möglichst hohes Maß an Usability zu erreichen.

1.2 Vorgehensweise

Die Durchführung des Projektes und somit auch der vorliegenden Dokumentation orientiert sich am Wasserfallmodell. Das Modell kann in die fünf Phasen Analyse, Design, Implementierung, Test und Betrieb unterteilt werden (Stober u. Hansemann, 2010, S. 16 ff.). Da es sich beim vorliegenden Projekt um die Erstellung eines Prototypen handelt, wird allerdings auf eine Betrachtung der Phase Betrieb verzichtet.

Im ersten Schritt des Projektes wird eine heuristische Anforderungsanalyse durchgeführt. Dabei werden begründete Annahmen darüber getroffen, in welchem Kontext die Anwendung vermutlich verwendet wird und welche Intentionen beziehungsweise Anforderungen die einzelnen Nutzergruppen mit sich bringen.

Die darauf folgende Designphase befasst sich zunächst mit den Fragen, wer der durchschnittliche Benutzer ist und welche Ziele er mit der Benutzung der Anwendung verfolgt. Dafür wird einerseits ein durchschnittliches Nutzerprofil in Form einer Persona für

jeweils den Elektroautofahrer und den Servicemitarbeiter erstellt. Außerdem werden User Stories zu den beiden Nutzergruppen verfasst, die in Form von kurzen Statements Bedürfnisse darlegen, die der Nutzer mit der Anwendung befriedigen möchte. Auf Basis dieser Ergebnisse wird dann ein Paper Prototyp erstellt, der den Aufbau der einzelnen Ansichten/Screens skizziert.

Für die Implementierung des zuvor erstellten Designs werden im ersten Schritt Überlegungen zum Aufbau der Software und der Kopplung einzelner Funktionalitäten aufgestellt. Danach wird die Anwendung agil entwickelt, indem möglichst früh ein fertiges Produkt geliefert wird. Dieses Produkt wird dann im Rahmen des vorhandenen Zeithorizonts iterativ durch weitere Komponenten ergänzt. Zur Verwaltung des kooperativ erstellten Codes wird ein Git Repository verwendet, wo die einzelnen Sprints als Issues angelegt sind.

In der letzten Phase des Projektes wird die Anwendung hinsichtlich ihrer Funktionalität und Usability getestet und bewertet.

2 Analyse

2.1 Heuristische Anforderungsanalyse

Betrachten wir zunächst die Umstände, die den Fahrer eines Elektroautos dazu verleiten könnten eine App für Ladestationen zu nutzen. Eine Intention, die als häufiger Kontext für die Verwendung der App angenommen werden kann, ist die Suche nach einer Ladestation in der näheren Umgebung des Nutzer. Eine leere Batterie, eventuell noch in einer unbekannten Umgebung, sind wieder auftretende Situationen, die ein schnelles Auffinden der nächsten Ladesäule nötig machen.

Am 01.01.2022 sind im Bundesgebiet 44.486 Ladesäulen gemeldet. Damit ist die Auswahl an Ladestationen in einem Jahr um 10.048 gestiegen. Allerdings ist der Ausbau der Infrastruktur im Vergleich der Bundesländer und vor allem im Vergleich zwischen Stadt und Land ungleich verteilt (Bundesnetzagentur, 2022). Diese inkonsistente Netzabde-

ckung bringt Unsicherheiten für den Fahrer eines Elektroautos mit sich. Somit wird ein potentieller Nutzer von der App außerdem erwarten, dass er sich die Ladesäulendichte in einem weiter entfernten Reiseziel anzeigen lassen kann.

Desweiteren gibt es bei den einzelnen Ladesäulen deutliche Unterschiede. Einerseits hinsichtlich des Preises und andererseits hinsichtlich angebotener Steckertypen und Ladegeschwindigkeit. Aus der wachsenden Anzahl an Optionen eine informierte Entscheidung treffen zu können, ist somit ein weiterer Anreiz für die Verwendung der App. Die Verschiedenheit der Ladesäulen und ihre relative Entfernung zum Wohnort des Nutzers führen unausweichlich dazu, dass einzelne Ladesäulen regelmäßig aufgesucht werden. Die präferierten Ladesäulen sollten daher vom Nutzer schnell aufgerufen werden können.

Eine relevante Information für einen Fahrer ist außerdem die momentane Verfügbarkeit einer Ladesäule. Das Aufladen eines E-Autos dauert deutlich länger als etwa die Tankdauer eines Autos mit Verbrennungsmotor. Der Ladevorgang bei einer herkömmlichen öffentlichen Ladesäulen dauert etwa 2 bis 4 Stunden und bei einer Schnellladesäulen immer noch 0.5 bis 1 Stunde (Entega, 2021). Ein Nutzer hat somit ein Interesse daran zu wissen, ob die Ladesäule gerade besetzt ist oder nicht.

Servicemitarbeiter/Techniker benötigen die App dagegen im Kontext ihres Anstellungsverhältnisses. Sie sind für die Reparatur und Wartung der Ladesäulen zuständig. Dafür müssen Sie einerseits ebenfalls den Standort einer Ladesäule einsehen können und andererseits wissen, welche Ladesäulen einen Defekt aufweisen. Daher wird ein Techniker erwarten, dass ein Kunde ein Problem mit einer Ladesäule melden kann und eine Liste dieser Meldungen für den Mitarbeiter ersichtlich ist.

Während jeder Kunde in der Lage sein sollte defekte Ladesäulen zu melden, sollte eine Reparatur nur von einem dafür dedizierten Mitarbeiter gemeldet werden können. Dafür sollte es einen abgetrennten Bereich in der Anwendung geben, der nur für Anwender mit bestimmten Administratorrechten zugänglich ist.

3 Design

3.1 User Story Mapping

3.2 Persona

3.3 Paper-Prototyping

4 Implementierung

4.1 Softwaredesign

4.1.1 Klassen & Kopplung

4.2 Umsetzung der Screens und Funktionalitäten

4.2.1 Datenbankintegration

Damit die Daten der Applikation persistent gemacht werden können und nicht bei jedem Abschalten der Anwendung verloren gehen, wird eine Datenbank mithilfe der Plattform Firebase angelegt. Dabei wird einerseits die dedizierte Authentifizierungsfunktionalität und andererseits die bereitgestellte NoSQL-Realtime-database verwendet. Die Datenbank kann über den integrierten Firebase Assistenten in Android Studio unter Verwendung des API-Schlüssels mit der App verbunden werden. So kann man sich über den Aufruf der Funktion `getInstance` eine Referenz zur vorhandenen Datenbankinstanz für den Authentifizierungsbereich und der Realtime Database holen (Abbildung 1). Wie die einzelnen Daten darin gespeichert werden, wird im weiteren Verlauf dargestellt.

```
//Get Firebase auth instance
private FirebaseAuth auth = FirebaseAuth.getInstance();
//Get Firebase Rtdb instance
DatabaseReference currentUserDb = FirebaseDatabase.getInstance("https://efinder-1640105181864-default-rtdb.europe-west1.firebaseio.com").getReference();
```

Abbildung 1: Generierung der Datenbankreferenzen

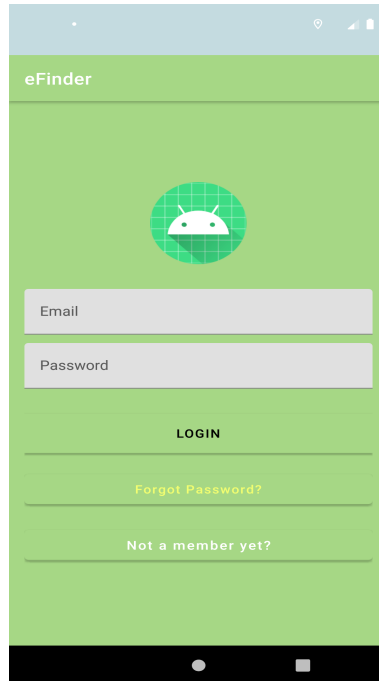


Abbildung 2: Login-Screen

4.2.2 Registrierung & Login

Beim erstmaligen Öffnen der App wird der Nutzer zum Login-Screen (Abbildung 2) weitergeleitet. Beim Aufruf der Login-Activity wird zunächst durch das Aufrufen der Funktion `isLoggedIn` überprüft, ob bereits eine Authentifizierungsinstanz existiert und wenn ja, wird der Nutzer direkt zur Main-Activity weitergeleitet (Abbildung 3).

Wenn bereits ein Konto existiert, aber keine offene Authentifizierungsinstanz gefunden wurde, kann der Nutzer sich mit seiner E-Mail und Passwort anmelden. Nach drücken des Buttons mit dem Label „Login“ wird die Funktion `authenticateUser` aufgerufen, die mit Hilfe der von Firebase bereitgestellte Funktion `signInWithEmailAndPassword` überprüft, ob die eingegebene E-Mail und das eingegebene Passwort korrekt sind (Abbildung 4). Erst dann wird der Nutzer zur Main-Activity weitergeleitet.


```

private void isLoggedIn() {
    if (auth.getCurrentUser() != null) {
        startActivity(new Intent( packageContext: LoginActivity.this, MainActivity.class));
        finish();
    }
}

```

Abbildung 3: Abfrage der existierenden Authentifizierungsinstanz

```

private void authenticateUser(String email, String password) {
    auth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener( activity: LoginActivity.this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                progressBar.setVisibility(View.GONE);
                if (!task.isSuccessful()) {
                    if (password.length() < 6) {
                        inputPassword.setError("Password must be at least 6 characters long!");
                    } else {
                        Toast.makeText( context: LoginActivity.this, "Authentication failed. Check your e-mail and password!", Toast.LENGTH_LONG).show();
                    }
                } else {
                    Intent intent = new Intent( packageContext: LoginActivity.this, MainActivity.class);
                    startActivity(intent);
                    finish();
                }
            }
        });
}

```

Abbildung 4: Authentifizierung des Benutzers

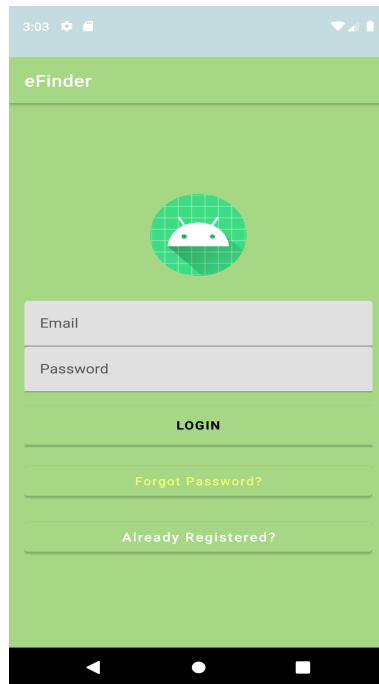


Abbildung 5: Registrierungs-Screen

Hat der Nutzer noch keinen Benutzer angelegt, kann er sich über den Button mit der Beschriftung „Not a member yet?“ zum Registrierungs-Screen (Abbildung 3) weiterleiten lassen. Wenn der Nutzer dort auf den Button mit der Beschriftung „Login“ drückt, wird zunächst getestet, ob er die nötigen Informationen in die dafür vorgesehenen EditText-Felder eingegeben hat und ob das Passwort eine Länge von mindestens 6 Zeichen aufweist. Ist dies nicht der Fall, wird der Nutzer darauf hingewiesen (Abbildung 4).

Wurden alle Eingabevoraussetzungen vom Nutzer erfüllt, wird durch das Aufrufen der Funktion `saveNewUser` ein neuer Benutzer in der Datenbank angelegt (Abbildung 5). Die Funktion `createUserWithEmailAndPassword` wird von Firebase bereitgestellt, um einen neuen Eintrag im Authentifizierungsbereich anzulegen (Abbildung 6). Gleichzeitig wird ein neues User-Objekt erstellt und als Node in der Realtime-Datenbank unter der ID des Benutzers angelegt. Wie das am Ende aussieht, kann in der Abbildung 7 gesehen werden. Nach erfolgreicher Registrierung wird der Nutzer zur MainActivity weitergeleitet.

4.2.3 Navigationsleiste

Nachdem der Nutzer zur MainActivity weitergeleitet wurde, befindet er sich zunächst auf dem Map-Screen. Von dort aus kann er sich mit Hilfe einer Navigationsleiste

```

if (TextUtils.isEmpty(email)) {
    Toast.makeText(getApplicationContext(), getString(R.string.Enter_eMail), Toast.LENGTH_SHORT).show();
    return;
}

if (TextUtils.isEmpty(password)) {
    Toast.makeText(getApplicationContext(), getString(R.string.Enter_password), Toast.LENGTH_SHORT).show();
    return;
}

if (password.length() < 6) {
    Toast.makeText(getApplicationContext(), getString(R.string.minimum_password), Toast.LENGTH_SHORT).show();
    return;
}

```

Abbildung 6: Überprüfung des User-Input

```

private void saveNewUser(String email, String password) {
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(activity, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Toast.makeText(context, "createUserWithEmail:onComplete:" + task.isSuccessful(), Toast.LENGTH_SHORT).show();
                progressBar.setVisibility(View.GONE);
                // If sign in fails, display a message to the user. If sign in succeeds
                // the auth state listener will be notified and logic to handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Toast.makeText(context, "Authentication failed." + task.getException(),
                        Toast.LENGTH_SHORT).show();
                } else {
                    String userID = auth.getCurrentUser().getUid();
                    String email = inputEmail.getText().toString();
                    currentUserDb.child("users").child(userID);
                    User newUser = new User(email);
                    Map<String, Object> userUpdates = new HashMap<>();
                    userUpdates.put("admin", false);
                    currentUserDb.setValue(newUser);
                    currentUserDb.updateChildren(userUpdates);
                    startActivity(new Intent(context, MainActivity.class));
                    finish();
                }
            }
        });
}

```

Abbildung 7: Anlegen eines neuen Benutzers


Kennung	Anbieter	Erstellt ↓	Angemeldet	Nutzer-UID
etiennegotha@gmx.de		26.01.2022	04.02.2022	cr24QJkpbzQ7hTja170UPw4UCXw2

Abbildung 8: Nutzer im Authentifizierungsbereich der Firebase



Abbildung 9: Nutzer in der Realtime-Database



Abbildung 10: Navigationsleiste

te(Abbildung 10) frei zwischen verschiedenen Screens(Map, Suchen, Admin, Favoriten, Einstellungen) bewegen. Die Navigationsleiste wird in der Main-Activity initialisiert und dort mit den einzelnen Fragmenten, welche die Views für die verschiedenen Screens enthalten, verbunden(Abbildung 11). Das Layout der Navigationsleiste wird über ein von Android bereitgestelltes AppBarConfiguration-Objekt gebildet. Über ein NavController-Objekt wird die Navigation zwischen den Fragmenten durch ein Host-Fragment in der XML der Main-Activity(Abbildung 12) organisiert.

```
BottomNavigationView navView = findViewById(R.id.nav_view);
// Passing each menu ID as a set of Ids because each
// menu should be considered as top level destinations.
AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(
    R.id.navigation_map, R.id.navigation_search, R.id.navigation_favorites,
    R.id.navigation_admin, R.id.navigation_preference)
    .build();
NavController navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment_activity_main);
NavigationUI.setupActionBarWithNavController( activity: this, navController, appBarConfiguration);
NavigationUI.setupWithNavController(binding.navView, navController);
```

Abbildung 11: Initialisierung der Navigationsleiste in der Main-Activity

```

<fragment
    android:id="@+id/nav_host_fragment_activity_main"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/main_green"
    app:defaultNavHost="true"
    app:layout_constraintBottom_toTopOf="@id/nav_view"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.52"
    app:navGraph="@navigation/mobile_navigation" />

```

Abbildung 12: Host-Fragment in der XML der Main Activity

4.2.4 Map-Screen

4.2.5 Suchen-Screen

4.2.6 Admin-Screen

4.2.7 Favoriten-Screen

4.2.8 Einstellungen-Screen

4.2.9 Mehrsprachigkeit & Landscape-Mode

5 Test

5.1 Funktionale Tests

5.2 Usability-Test

6 Fazit

7 Literaturverzeichnis

[Bundesnetzagentur 2022] BUNDESNETZAGENTUR: *Elektromobilität: Öffentliche Ladeinfrastruktur*. https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/E-Mobilitaet/start.html;jsessionid=DED56121F8DC1432BB5B1D52302BB226. Version: 2022. – zuletzt besucht am 05-02-2022

[Entega 2021] ENTEGA: *Ladezeit von Elektroautos: Wie schnell ist das Auto aufgeladen?* <https://www.entega.de/blog/elektroauto-ladezeit/>. Version: 2021. – zuletzt besucht am 06-02-2022

[Stober u. Hansemann 2010] STOBER, T. ; HANSEMAN, U.: *Best Practices for Large Software Development Projects*. Springer, 2010

A Anhang

B Eigenständigkeitserklärung

“Ich versichere, dass ich die Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen oder anderen Quellen entnommen sind, sind als solche kenntlich gemacht. Die schriftliche und die elektronische Form der Arbeit stimmen überein. Ich stimme der Überprüfung der Arbeit durch eine Plagiatssoftware zu.”

Ort, Datum

Unterschrift