

DiaMon

Ein Diabetes-Management System
für Kinder und Jugendliche

Meilenstein 4

EIS
ENTWICKLUNG INTERAKTIVER SYSTEME

ausgearbeitet von

Edgar Gellert
Marcel Holter

im Studiengang
MEDIENINFORMATIK

Dozenten : Prof. Dr. Gerhard Hartmann
Fachhochschule Köln

Prof. Dr. Kristian Fischer
Fachhochschule Köln

Betreuer : B. Sc. Robert Gabriel
Fachhochschule Köln

Gummersbach, 25. Juni 2015

Kurzfassung

Die Dokumentation zum Modul EIS - Entwicklung Interaktiver Systeme beschreibt das Vorgehen bei der Entwicklung des Diabetes Management Systems *Diagotschi*. Es werden MCI spezifische Vorgehensweisen, sowie WBA2 technische Aspekte des Projektes aufgelistet und erläutert.

Inhaltsverzeichnis

1 Meilenstein 4	6
1.1 Datenstrukturen	6
1.1.1 Profil der Kinder	6
1.1.2 Nahrungsmittel	7
1.1.3 Thread im Forum	7
1.1.4 Eltern	8
1.1.5 Google Cloud Messages	8
1.2 REST API Spezifikation	9
1.3 Anwendungslogik	11
1.3.1 Client	11
1.3.2 Server	18
1.4 Design, Testen und Entwickeln	22
1.4.1 Work Reengineering	22
1.4.2 Screen Design Standards	23
1.4.3 Prototypen UI	26
Abbildungsverzeichnis	36
Tabellenverzeichnis	37
Glossar	37
Literaturverzeichnis	38

1 Meilenstein 4

1.1 Datenstrukturen

Im Folgenden werden alle modellierten Datenstrukturen des Systems abgebildet. Im Kapitel Systemarchitektur wurde sich für das Datenformat JavaScript Object Notation (JSON) entschieden. Um Schemata für die Objekte zu definieren, wird das Modul Mongoose genutzt. Auf dem Client werden Klassen implementiert, deren Struktur equivalänt zu den Mongoose Schemata sind.

1.1.1 Profil der Kinder

Das zentrale Profil der Kinder beinhaltet alle wichtigen Informationen zu den Kindern inklusive aller Angaben bezüglich der Diabetes-Therapie. Das Logbuch als Array von ObjectIds, die auf die einzelnen Logbucheinträge refenzieren, zu modellieren, wurde verworfen. Da die Daten lokal auf dem Client, also auch ohne Verbindung zum Server erreichbar sein müssen und die ObjectIds erst beim Speichern in die Datenbank auf dem Server generiert werden, sind die Logbucheinträge nun direkt mit allen Informationen im Array "log" gespeichert.

```
1  {
2      _id: ObjectId,
3      gcm_id: String,
4      name: String,
5      age: Integer,
6      gender: String,
7      sports: [String],
8      diseases: [String],
9      log: [
10          date: { type: Date, default: Date.now },
11          bloodsugar: Number,
12          be: Number,
13          correctionValue: Number,
14          insulin: Number,
15          mood: String,
16          notes: String
17      ],
18      parent: ObjectId,
```

```

19   doc: ObjectId,
20   therapy: {
21     factor: {
22       morning: Number,
23       day: Number,
24       evening: Number,
25       type: String
26     },
27     type: String,
28     target: Integer,
29     correction: Integer
30   }
31 }
```

Listing 1.1: Kind

1.1.2 Nahrungsmittel

Vorhandene Nahrungsmittel-Datenbanken von Drittanbietern sind nicht kostenlos. Aus diesem Grund wird für den funktionalen Prototyp eine eigene Datenbank implementiert, die Testdatensätze zu Nahrungsmitteln mit den notwendigen Informationen enthält. Für die Berechnung der Insulindosis werden die Broteinheiten oder Kohlenhydrateinheiten in Lebensmitteln aus der Datenbank gelesen. Die jeweiligen BE bzw. KE werden pro 100 Gramm angegeben. Weiterhin werden Name, Hersteller und Kategorie des Produkts mitgegeben. Dadurch lassen sich Produkte unter verschiedenen Kriterien suchen und finden. Falls ein Bilderkatalog als Produktsuche implementiert werden sollte, wird ein Attribut mit der Quelladresse der Bildes hinzugefügt.

```

1 {
2   _id: ObjectId,
3   brand: String,
4   type: String,
5   description: String,
6   be: String,
7   ke: String
8 }
```

Listing 1.2: Produkt

1.1.3 Thread im Forum

Fragen, die Eltern an die Community stellen, werden als Threads in einem Forum modelliert. Dabei wird auf den Ersteller der Frage referenziert. Weiterhin besteht das Objekt aus dem Datum, dem Inhalt der Frage und den Kommentaren zu dieser Frage.

Die Kommentare werden in einem Array von Objekten modelliert. Stattdessen könnten die Kommentare auch selbstständig modelliert und nur per ObjectIDs referenziert werden. Da die Kommentare aber immer nur in Verbindung mit dem zugehörigen Thread und nicht eigenständig einsehbar sind, wird auf die Modellierung eigener Kommentar Objekte und deren Referenzierung über ObjectIDs in einem verzichtet.

```
1 {
2   _id: ObjectId,
3   author: ObjectId,
4   date: { type: Date, default: Date.now },
5   body: String,
6   topics: [String],
7   comments: [{  
8     body: String,  
9     date: { type: Date, default: Date.now },  
10    author: ObjectId }]  
11 }
```

Listing 1.3: Thread im Forum

1.1.4 Eltern

Das Objekt der Eltern beinhaltet die notwendigen Daten zur Authentifizierung und die Referenz auf die Objekte der Kinder über die ObjectIDs innerhalb des Arrays "children".

```
1 {
2   _id: ObjectId,
3   gcm_id: String,
4   email: String,
5   password: String,
6   name: String,
7   children: [ObjectId]
8 }
```

Listing 1.4: Eltern

1.1.5 Google Cloud Messages

Google Cloud Messages werden vom System in zwei Anwendungsfällen genutzt. Wenn ein Kind einen Eintrag im Logbuch gespeichert hat, soll der Client der Eltern darüber benachrichtigt werden. Der zweite Anwendungsfall ist die Weiterleitung der von Eltern im Forum gestellten Fragen an andere Eltern. Diese Nachrichten sollen per Google Cloud an die Empfänger gesendet werden. Da sowohl die Logbucheinträge als auch die Fragen im Forum personenbezogene Daten beinhalten, werden die Inhalte selbst

nicht über die Google Cloud gesendet. Stattdessen werden lediglich die ObjectIDs der Logbucheinträge bzw. Threads im Forum und der eindeutige Registration Token des Empfängers übertragen. Der Empfänger kann somit mit der empfangenen ObjectID den Inhalt beim Server direkt anfragen.

```

1 {
2   data: {
3     id: ObjectId
4   },
5   to: String
6 }
```

Listing 1.5: Google Cloud Messages

1.2 REST API Spezifikation

Alle Ressourcen der REST API werden mit den anwendbaren HTTP Methoden wie folgt beschrieben:

- /carbs
 - GET: Die Listenressource aller Nahrungsmittel in der Kohlenhydrate-Datenbank wird im Response gesendet.
 - POST: Eine neue Nahrungsmittel Ressource mit den im Request übergebenen Informationen (Bezeichnung, Hersteller, Kategorie, BE, KE) wird in der Datenbank unter einer eindeutigen, automatisch generierten ID angelegt.
 - PUT: -
 - DELETE: -
- /carbs?search=query
 - GET: Die Listenressource aller Nahrungsmittel, die in der Kohlenhydrate-Datenbank zum Suchwort {query} gefunden wurden, werden im Response gesendet.
 - POST: -
 - PUT: -
 - DELETE: -
- /children
 - GET: Die Listenressource aller Kind Ressourcen wird im Response gesendet.

- POST: Eine neue Kind-Ressource wird unter einer eindeutiger, automatisch generierten ID in der Datenbank gespeichert.
 - PUT: -
 - DELETE: -
- /children/{child}
 - GET: Die Kind-Ressource mit der eindeutigen ID {child} wird im Response gesendet.
 - POST: -
 - PUT: Die Kind-Ressource mit der eindeutigen ID {child} wird durch die im Request enthaltenen Informationen in der Datenbank aktualisiert.
 - DELETE: Die Kind-Ressource mit der eindeutigen ID {child} wird aus der Datenbank gelöscht.
 - /children/{child}/log
 - GET: Das Array "log" mit allen Einträgen im Logbuch der Kind-Ressource mit der eindeutigen ID {child} wird im Response gesendet.
 - POST: -
 - PUT: Der im Request enthaltene Logbucheintrag wird dem Array "log" der Kind-Ressource mit der eindeutigen ID {child} hinzugefügt.
 - DELETE: -
 - /forum
 - GET: Die Listenressource aller Thread-Ressourcen wird im Response gesendet.
 - POST: Eine neue Ressource mit einer eindeutig, automatisch generierten ID mit allen im Request enthaltenen Informationen in der Datenbank gespeichert.
 - PUT: -
 - DELETE: -
 - /forum/{thread}
 - GET: Die Thread-Ressource mit der eindeutigen ID {thread} wird im Response gesendet.
 - POST: -

- PUT: Dem Array "comments" der Thread-Ressource mit der eindeutigen ID {thread} wird ein Objekt mit den im Request enthaltenen Informationen hinzugefügt.
 - DELETE: Die Thread-Ressource mit der eindeutigen ID {thread} wird aus der Datenbank gelöscht.
- /parents/{parent}/children
 - GET: Die Listenressource aller Kind Ressourcen, die im Array "children" der Eltern-Ressource mit der eindeutigen ID {parent} enthalten sind, wird im Response gesendet.
 - POST: -
 - PUT: -
 - DELETE: -

1.3 Anwendungslogik

Da sowohl auf dem Server als auch auf dem Client Anwendungslogik modelliert und zum Teil implementiert wurde, werden Server und Client im Folgenden einzeln voneinander betrachtet.

1.3.1 Client

Struktur

Die einzelnen Java-Klassen wurden zur Übersicht in folgende *packages* unterteilt:

- activities
In diesem *package* sind alle Activities mit der darauf stattfindenden Anwendungslogik enthalten. Zur weiteren Übersichtlichkeit wurden die Activities für die Teilapplikation der Eltern und der Kinder in *children* und *parents* unterteilt
- util
In diesem *package* befinden sich Helferklassen für grundlegende Funktionalitäten wie zum Beispiel die Adapter zum Databinding von Daten an einen ListView oder der statische RestClient für die Kommunikation mit dem Server.

Insgesamt wurden viele Klassen implementiert, im weiteren Verlauf werden nur die Klassen mit den wichtigen Logiken beschrieben.

Berechnung der Insulineinheiten

Vorerst wird kurz der Ablauf für die Berechnung beschrieben, da das Verfassen des Logbucheintrags und die Ermittlung der Broteinheiten basierend auf einer nach den Nahrungsmitteln ein geschlossener Ablauf sind.

1. Kind gibt den Blutzuckerwert nach der Messung ein
2. Kind sucht nach Nahrungsmitteln, die es verzehren will
3. Client schickt die Suche an den Server und erhält die gewünschten Nahrungsmittel mit den Broteinheiten
4. Kind gibt das Gewicht der Nahrungsmittel an
5. Client berechnet die Insulineinheiten basierend auf den Nahrungsmitteln und deren Gewicht, dem aktuellen Broteinheiten-Faktor basierend auf der Tageszeit und der eventuellen Korrekturteinheiten basierend auf dem Blutzuckerwert und dem Zielwert im Profil des Kindes
6. Client speichert den Logbucheintrag lokal und sendet ihn gleichzeitig an den Server

```

1 void searchProducts(String search) {
2     RestClient.get("carbs?=" + search, null, new JsonHttpResponseHandler()
3     {
4         @Override
5         public void onSuccess(int statusCode, Header[] headers, JSONArray
6             response) {
7             for (int i = 0; i < response.length(); i++) {
8                 try {
9                     Product product = new Gson().fromJson(response.getJSONObject(i
10                         ).toString(), Product.class);
11                     products.add(product);
12                 } catch (JSONException e) {
13                     // TO-DO: Handle Exception
14                 }
15             }
16             ga.notifyDataSetChanged();
17         }
18     });
19 }
```

```

1 public void calculateUnits(View view) {
2     Calendar calendar = Calendar.getInstance();
3     Integer time = calendar.get(Calendar.HOUR_OF_DAY);
```

```

4  SharedPreferences settings = context.getSharedPreferences("sharedPreferences", 0);
5  String profile = settings.getString("profile", "");
6  Child child = new Gson().fromJson(profile, Child.class);
7  Double beFactor;
8  if (isBetween(time, 5, 12)) {
9      beFactor = child.therapy.factor.morning;
10 } else if (isBetween(time, 12, 17)) {
11     beFactor = child.therapy.factor.day;
12 } else {
13     beFactor = child.therapy.factor.evening;
14 }
15 Double total = 0.0;
16
17 for (int i = 0; i < lv.getCount(); i++) {
18     String item = pa.getProduct(i);
19     Double tmp = Double.parseDouble(item);
20     Double tmp2 = Double.parseDouble(pa.getItem(i).getBe());
21     Double result = (tmp / 100) * tmp2;
22     total += result;
23 }
24 correctionValue = child.therapy.correction;
25 targetValue = child.therapy.target;
26 Integer correctionUnits = (bsValue - targetValue) / correctionValue;
27
28 ArrayList<String> results = new ArrayList<>();
29
30 results.add(0, correctionUnits.toString());
31 results.add(1, total.toString());
32 results.add(2, beFactor.toString());
33
34 Intent intent = new Intent(context, AddEntry.class);
35 intent.putStringArrayListExtra("results", results);
36
37 startActivity(intent);
38 }

```

```

1 public void addEntry(View view) {
2     TimeZone tz = TimeZone.getTimeZone("GMT");
3     DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm'Z'");
4     df.setTimeZone(tz);
5     String nowAsISO = df.format(new Date());
6     Child.LogEntry logEntry = new Child.LogEntry();
7     logEntry.date = nowAsISO;
8     logEntry.bloodsugar = Integer.parseInt(bsValue.getText().toString());
9     logEntry.be = Double.parseDouble(beValue.getText().toString());

```

```

10    logEntry.beFactor = Double.parseDouble(beFactorValue.getText() .
11        toString());
12    logEntry.correctionValue = Integer.parseInt(correctionValue.getText() .
13        toString());
14    logEntry.insulin = Double.parseDouble(ieValue.getText().toString());
15    logEntry.notes = notes.getText().toString();
16    //logEntry.mood = mood.getText().toString();
17    SharedPreferences settings = context.getSharedPreferences(PREFS_NAME ,
18        0);
19    editor = settings.edit();
20    String profile = settings.getString("profile", "");
21    Child child = new Gson().fromJson(profile, Child.class);
22    child.log.add(logEntry);
23    String newProfile = new Gson().toJson(child);
24    editor.putString("profile", newProfile);
25    editor.apply();
26    RequestParams params = new RequestParams();
27    params.put("date", logEntry.date);
28    params.put("bloodsugar", logEntry.bloodsugar);
29    params.put("be", logEntry.be);
30    params.put("beFactor", logEntry.beFactor);
31    params.put("correctionValue", logEntry.correctionValue);
32    params.put("insulin", logEntry.insulin);
33    params.put("notes", logEntry.notes);
34    //params.put("mood", mood.getText().toString());
35    RestClient.post("children/exampleChild/log", params, new
36        JsonHttpResponseHandler() {
37            @Override
38            public void onSuccess(int statusCode, Header[] headers, JSONObject
39                response) {
40                Log.e("Response", response.toString());
41            }
42        });
43    }
44 }

```

Darstellung der Logbücher/Statistik

Für die Darstellung der Logbücher und der darin enthaltenen Einträge wurden verschiedene Arten geplant. Aufgrund mangelnder Zeit wird hier in Pseudocode lediglich die Darstellung eines Graphen in Pseudocode dargestellt. Basis ist ein Koordinatensystem mit dem Blutzucker auf der Y-Achse und Zeit auf der X-Achse. Jede Messung wird als Punkt im Koordinatensystem angezeigt und zur Darstellung des Verlaufs werden diese Punkte miteinander verbunden. Bei einem Klick auf einen der Punkte füllt

sich eine Tabelle unterhalb des Koordinatensystems mit allen Informationen, die der ausgewählte Beitrag beinhaltet:

```

1 FUER alle Eintraege im Logbuch
2   Punkt P(i) = (Zeitpunkt/Blutzuckerwert)
3   Zeichne P(i) in Koordinatensystem
4   Zeichne Linie von P(i) nach P(i-1)
5 ENDE
6
7 WENN Punkt angeklickt wird
8   DANN Tabelle.Datum = Eintrag.Datum
9     Tabelle.Ort = Eintrag.Ort
10    Tabelle.Blutzucker = Eintrag.Blutzucker
11    Tabelle.Broteinheiten = Eintrag.Broteinheiten
12    Tabelle.BEFaktor = Eintrag.BEFaktor
13    Tabelle.Korrekturinsulin = Eintrag.Korrekturinsulin
14    Tabelle.Insulineinheiten = Eintrag.Insulineinheiten
15    Tabelle.Gemuetszustand = Eintrag.Gemuetszustand
16    Tabelle.Notizen = Eintrag.Notizen

```

Listing 1.6: Darstellung des Logbuchs

RestClient zur Kommunikation der REST-API auf dem Server

Für die Kommunikation mit der REST-API des Servers wurde eine Helferklasse RestClient implementiert. Mit Hilfe dieser lassen sich die Methoden GET, PUT, POST und DELETE auf die Ressourcen anwenden. Der RestClient wurde statisch implementiert, da auch die Server-Adresse statisch ist. Man muss lediglich die Ressource bzw Route selbst beim Aufruf übergeben. Optional lassen sich request-Parameter in Form von Key/Value-Paaren übergeben. Weiterhin wird automatisch bei jedem Request der Token zur Authentifizierung im Header mitgegeben.

Diese Klasse nutzt die Library Android Asynchronous Http Client (LoopJ). Die Library sorgt für die asynchrone Verarbeitung der Requests, damit das Interface während dem Warten auf den Response vom Server nicht blockiert ist. Ein weiterer Vorteil dieser Library ist der zur Verfügung stehende JsonHttpResponseHandler. Da vom Server ausschließlich JSON zurückgegeben wird, lässt sich der Response sehr einfach weiter verarbeiten.

```

1 public class RestClient extends Application {
2
3     private static final String BASE_URL = "http://10.0.2.2:3000/";
4
5     private static AsyncHttpClient client = new AsyncHttpClient();
6

```

```

7     private static String token = Authentication.token;
8
9     public RestClient() {}
10
11    public static void get(String url, RequestParams params,
12                           JsonHttpResponseHandler responseHandler) {
13        client.addHeader("x-access-token", token);
14        client.get(getAbsoluteUrl(url), params, responseHandler);
15    }
16
17    public static void post(String url, RequestParams params,
18                           JsonHttpResponseHandler responseHandler) {
19        client.addHeader("x-access-token", token);
20        client.post(getAbsoluteUrl(url), params, responseHandler);
21    }
22
23    public static void put(String url, RequestParams params,
24                           JsonHttpResponseHandler responseHandler) {
25        client.addHeader("x-access-token", token);
26        client.put(getAbsoluteUrl(url), params, responseHandler);
27    }
28
29    public static void delete(String url, RequestParams params,
30                             JsonHttpResponseHandler responseHandler) {
31        client.addHeader("x-access-token", token);
32        client.put(getAbsoluteUrl(url), params, responseHandler);
33    }
34
35 }

```

Am Beispiel des Requests an den Server nach allen Forumthreads kann man sehen wie dies genau funktioniert:

```

1 private void getThreads() {
2     RestClient.get("forum", null, new JsonHttpResponseHandler() {
3         @Override
4         public void onSuccess(int statusCode, Header[] headers, JSONArray
5             response) {
6             for (int i = 0; i < response.length(); i++) {
7                 try {
8                     Thread thread = new Gson().fromJson(response.getJSONObject(i).
9                         toString(), Thread.class);
10                    threads.add(thread);
11                }
12            }
13        }
14    });
15 }

```

```

9         } catch (JSONException e) {
10             // TO-DO: Handle Exception
11         }
12     }
13     ta.notifyDataSetChanged();
14 }
15 );
16 }

```

Umwandlung von JSON in Java Objekte mittels GSON

Um mit den Daten von vom Server auf dem Client umgehen zu können, müssen sie von JSON in Java Objekte umgewandelt werden. Die Library GSON von Google eignet sich sehr gut dafür. Für diese Umwandlung müssen Klassen implementiert werden, die der Datenstruktur der JSON Objekte exakt entspricht.

```

1 Datenstruktur der Child Objekte in Java:
2
3 public class Child {
4     public String name;
5     public Integer age;
6     public String gender;
7     public List<String> sports;
8     public List<String> diseases;
9     public List<LogEntry> log;
10    public String parent;
11    public String doc;
12    public Therapy therapy;
13
14    public class Therapy {
15        public Factor factor;
16        public String type;
17        public Integer target;
18        public Integer correction;
19    }
20
21    public class Factor {
22        public Double morning;
23        public Double day;
24        public Double evening;
25        public String type;
26    }
27
28    public static class LogEntry {
29        public String date;
30        public Integer bloodsugar;

```

```

31     public Double be;
32     public Double beFactor;
33     public Integer correctionValue;
34     public Double insulin;
35     //public String mood;
36     public String notes;
37 }
38 }
39
40 Umwandlung vom JSON-String in Java Objekt:
41 Child child = new Gson().fromJson(profile, Child.class);

```

1.3.2 Server

Der Server wurde, wie im Kapitel Systemarchitektur beschrieben, in der NodeJS Umgebung mit dem Framework Express implementiert.

Struktur

Der Server ist wie folgt strukturiert:

- /bin
Hier ist die Konfiguration des Servers enthalten. Diese wurde von Express automatisch generiert
- /config
Hier befinden sich Konfigurationsdateien, wie die Adresse der Datenbank und dem Secret für die Tokens zur Authentifizierung
- /models
In diesem Ordner werden alle Schemata definiert. Diese bestimmen die erläuterten Datenstrukturen
- /routes
In diesem Ordner befinden sich alle in der REST-Spezifikation erläuterten API-Endpoints und deren Logik
- /util Hier befinden sich unterstützende Funktionen, wie der Überprüfung auf erfolgreiche Authentifizierung (selbst implementierte MiddleWare)

Token Authentication MiddleWare

Um die personenbezogenen Daten nur für Befugte zugänglich zu machen, müssen die Endpoints in der API mit einer MiddleWare geschützt werden. Diese überprüft, ob ein

Token vorhanden ist und ob dieser valide ist. Ist dies nicht der Fall, wird der StatusCode 403 mit einer Fehlermeldung zurückgegeben. Ist der Token valide gibt die MiddleWare die ID des Nutzers an die Route weiter, woraufhin dann überprüft werden kann, ob diese ID Zugriff auf die angeforderte Ressource hat.

```

1 var isAuthorized = function(req, res, next) {
2   console.log(chalk.yellow('Checking if user is authorized...'));
3   var token = req.headers['x-access-token'];
4   if (token) {
5     jwt.verify(token, secret, function(err, decoded) {
6       if (err) {
7         return res.status(403).json({
8           success: false,
9           message: 'Failed to authenticate token.'
10        });
11      } else {
12        req.decoded = decoded;
13        console.log(chalk.green(decoded.name + '(' + decoded._id + ')
14          has been verified!'));
15        next();
16      }
17    });
18  } else {
19    return res.status(403).json({
20      success: false,
21      message: 'No token provided'
22    });
23 }

```

Listing 1.7: Token Authentication MiddleWare

Suche nach Nahrungsmitteln

Die Suche nach Nahrungsmitteln wird mit einem GET auf die Ressource `/carbs` gestartet. Steht kein Suchbegriff im query (`/carbs?search={query}`), werden alle Nahrungsmittel der Datenbank zurückgegeben. Andernfalls wird mit einem Regulären Ausdruck nach dem Suchbegriff gesucht und alle Ergebnisse werden als Array von JSON Objekten zurückgegeben. Der reguläre Ausdruck sorgt dafür, dass beispielsweise beim Suchwort *Schokolade* auch *Alpenmilchschokolade* gefunden wird.

```

1 router.get('/', function(req, res, next) {
2   if (typeof req.query.search !== "undefined") {
3     console.log(chalk.yellow('Searching for: ') + chalk.blue(req.query.
4       search));
5     Product.find({

```

```

5     description: {
6       $regex: ".*" + req.query.search + ".*",
7       $options: 'i'
8     }
9   }, function(err, products) {
10    console.log(chalk.green('Found: ') + chalk.blue(JSON.stringify(
11      products, null, 2)));
12    res.json(products);
13  });
14 } else {
15  console.log(chalk.yellow('Requesting all products from database... '))
16 Product.find({}, function(err, products) {
17  console.log(chalk.green('Found: ') + chalk.blue(JSON.stringify(
18    products, null, 2)));
19  res.json(products);
20 });
21 }
22 });

```

MatchMaking bei Forenbeiträgen

Wenn Eltern im Forum einen Thread erstellen, wird das Profil des Kindes mittels eines MatchMaking-Algorithmus mit allen Kind Profilen in der Datenbank verglichen. Bei Übereinstimmung der Kind Profile werden die Eltern dieser Kinder per Google Cloud Messaging auf den Thread aufmerksam gemacht. Aufgrund mangelnder Zeit wurde diese Funktion nur per Pseudocode modelliert:

```

1 FUER jedes Kind in der Datenbank
2 WENN das Alter des Kindes mit einer Abweichung von +/-2 des zu
   vergleichendes Kindes uebereinstimmt
3   DANN erhoehe Uebereinstimmung +1
4 WENN eine der weiteren Krankheiten des Kindes mit einer der weiteren
   Krankheiten des zu vergleichenden Kindes uebereinstimmt
5   DANN erhoehe Uebereinstimmung +2
6 WENN eine der Sportarten des Kindes mit einer der Sportarten des zu
   vergleichenden Kindes uebereinstimmt
7   DANN erhoehe Uebereinstimmung +1
8 WENN der durchschnittliche Blutzucker des Kindes mit einer Abweichung
   von 25 mit dem durchschnittliche Blutzucker des zu vergleichenden
   Kindes uebereinstimmt
9   DANN erhoehe Uebereinstimmung +1
10 WENN Uebereinstimmung >= 3
11   DANN sende ID des Threads an Eltern des Kindes ueber Google Cloud
      Messaging

```

Listing 1.8: Pseudocode MatchMaking

Die Gewichtung der Übereinstimmungen wurden frei gewählt. Da dem Team nur Repräsentanten der Stakeholder und nicht die Stakeholder direkt zur Verfügung standen, war es schwierig zu erfahren, wie die Eltern diese Funktion bewerten. Als Alternative könnte man beim Erstellen der Threads den Eltern die Möglichkeit bieten die Gewichtung der einzelnen Vergleiche selbst zu bestimmen.

1.4 Design, Testen und Entwickeln

In der zweiten Phase des Usability Engineering Lifecycle werden die Prozesse Design, Testing und Entwicklung durchgeführt. Die gesamte Phase erstreckt sich über 3 Level. Das erste Level beginnt mit dem Work Reengineering.

1.4.1 Work Reengineering

Mit Beginn der zweiten Phase des Usability Engineering Lifecycle wird nach Mayhew ((Mayhew 1999), Seite 171) ein „Work Reengineering“ vollzogen. Dieser Prozess beschreibt eine Form von Neuausrichtung der momentanen Problem-, bzw. Arbeitsbewältigung. Hierfür werden die zuvor durchgeführten Arbeitsschritte genutzt, um darauf aufbauend ein „*Reengineered Task Organization Model*“ zu entwickeln. Dieses Modell beschreibt eine Aussicht auf einen möglichen Ablauf in dem zu entwickelnden System. Abbildung 1.1 zeigt das Reengineered Task Organization Model eines an Diabetes erkrankten Kindes. Aus den Task Szenarien und den Dokumentationen ergab sich, dass Eltern in der Regel keinen direkten Zugriff auf die Erfassungen der Kinder haben. Kontrolle, oder Einsicht in die Erfassungen der Kinder kann nur über ein Gespräch mit den Kindern und die darauffolgende Einsicht in das zumeist papierbasierte Logbuch erfolgen. Das zuvor in Kapitel ?? beschriebene Modell soll hier nun um ein weiteres ergänzt werden. Abbildung 1.2 auf Seite 24 zeigt ein Modell, das einen möglichen Systemablauf der Eltern beschreibt.

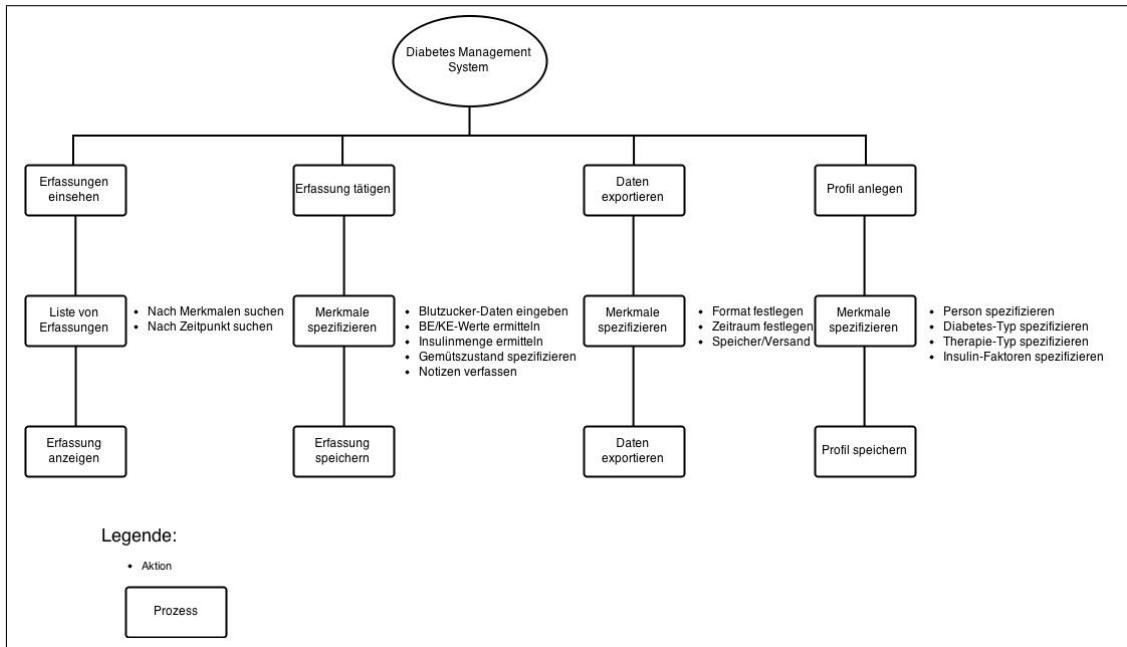


Abbildung 1.1: Reengineered Task Organization Model eines an Diabetes erkrankten Kindes

1.4.2 Screen Design Standards

Die Entwicklung eines eigenen Styleguides mag laut Mayhew für kleine Projekte nicht zwingend notwendig sein, die Screen Design Standards beschreibt Mayhew jedoch als unerlässlich. Sie liefert dazu keinen Shortcut im eigentlich Sinne, sondern sagt lediglich aus, dass die formale Dokumentation ausgelassen werden kann. Da dies nicht in Frage kommt, soll hier nochmals referenziert werden, auf welche Standards eingegangen wird und welche Control Standards Verwendung finden. Es sollen Standards festgelegt werden, die die von Mayhew beschriebenen Ease of Learning, Ease of Use, sowie die Grundsätze der Dialoggestaltung der EN ISO 9241-110:2006 (Committee 2008) erfüllen.

Dialog Box Standards

Im Folgenden sollen einige Dialog Box Standards spezifiziert werden:

- Vertikale Gruppierung von logisch verbundenen Feldern.
- Der Hintergrund der Dialoge soll in einem Grün-Ton gestaltet werden, der vom neuen Material Design (Google-Design) unterstützt wird. Grün deshalb, da die Farbe für Vitalität steht und somit eine positive User Experience ermöglicht. Für jene Textfelder, die einen Überblick über die eigenen Werte ermöglichen, soll eine blaue oder weiße Farbe verwendet werden. Blau repräsentiert den „Birds-Eye-

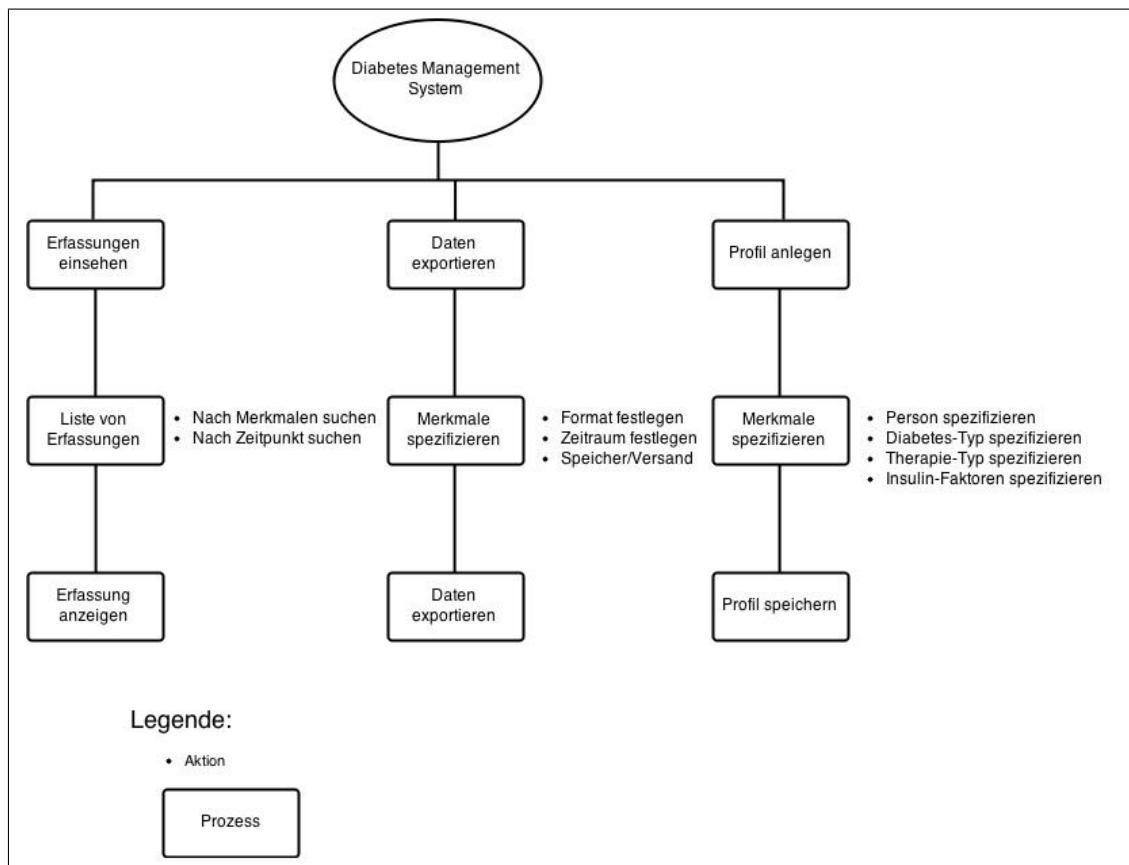


Abbildung 1.2: Reengineered Task Organization Model eines an Diabetes erkrankten Kindes

View“, es unterstützt also den Überblick zu behalten. Weiß steht für Fakten und Wissen und ermöglicht eine Form von Sachlichkeit. Insgesamt werden maximal drei Hauptfarben gewählt. Tests an der eigenen Sehkraft und der anderer ergaben, dass trotz Rot-Grün Schwäche die grüne Farbe als angenehm und perzipierbar erachtet wurde.

- White Space soll als Trennung zwischen Gruppierungen dienen.
- Der „Speichern“-, „Senden“-, oder „Exportieren“-Button eines Dialoges soll immer in der Mitte des unteren Rahmens abgebildet werden. Die Farbe der Buttons soll aus der *secondary palette* gewählt werden.
- Es sollen konsistente Labels für Buttons und Textfelder genutzt werden.
- Informationsflächen, die lediglich Mehrwissen transportieren, sollen in einer helleren Hauptfarbe dargestellt werden.
- Texte bzw. Schriften sollen vorzugsweise schwarz dargestellt werden.
- Es sollen keine Pop-Up Fenster oder Tab-basierte Dialogfenster zur Eingabe Benutzerspezifischer Daten verwendet werden.

Message Box Standards

Unterschieden wird bei Message Boxes zwischen einem **Alert** und einem **Toast**. Bei einem **Alert** ist der Benutzer gezwungen seine Interaktion zu unterbrechen und eine Bestätigung zu leisten. Bei einem **Toast** wird lediglich eine auf eine kurze Dauer beschränkte Nachricht auf dem Screen ausgegeben.

- Ein Alert, das die Interaktion des Benutzer unterbricht, soll so weit es geht vermieden werden.
- Ein Toast, das beispielsweise das erfolgreiche Speichern einer neuen Erfassung signalisiert, kann durchaus implementiert werden. Solch ein Toast kann als Feedback auf eine Interaktion des Benutzers angeführt werden.

Input Standards

Die Interaktion des Benutzer erfolgt über das Touch-Screen des Mobiltelefons. Zur Benutzung dienen die Tast- und Gestenbewegungen. Die Erfassung des Blutzuckers wird über ein externes Blutzucker-Messgerät vorgenommen, welches der Diabetiker in der Regel stets bei sich führt. Es existieren zwar Blutzucker-Messgeräte, die über den Audio-Ausgang des Handys angeschlossen werden können, solche Geräte sollen in erster Linie jedoch nicht berücksichtigt werden.

Output Standards

Abgesehen von der visuellen Ausgabe des Inhaltes über das Display, sollen die archivierten Erfassungen des Kindes über die Export-Funktion beispielsweise als PDF oder als MS Excel Sheet ausgegeben werden können. PDF/MS Excel?

Control Standards

Im Folgenden sollen, um das *Ease of Learning* und *Ease of Use* von Mayhew zu unterstützen, Standards der GUI spezifiziert werden.

Tabelle 1.1: Control Standards

Funktionen / Activities	Steuerung
Navigation action	Push Buttons mit Label
Auswahlmenüs / Listen	Drop-down Listen-Box, Recycle-View mit Card-View,
Action Bar	Icons und Activity Titel
Informationsfelder	ggf. Icons und textuelle Beschreibung
Input des Benutzers	Textfelder, Buttons, Image-Selection

1.4.3 Prototypen UI

In die Phase des Designs fließen alle Faktoren ein, die zuvor durch die Anforderungen, die Benutzer-Modelle und die Benutzungsmodellierung ermittelt wurden. Im Folgenden sollen einige Prototyp-Sketches dargestellt werden, die die ersten Eindrücke des User Interface visualisieren.

Wie zuvor bei den User Profiles, soll auch bei dem Design eine Unterscheidung zwischen den Benutzern erfolgen. Jüngeren Benutzern (zwischen 8 und 12 Jahren) soll ein Design geliefert werden, das sie in ihrer Motivation unterstützt und die nötigen Aufgaben absolvieren lässt. Dies geschieht durch das Einbinden einer Art Maskottchen. Benutzer im Jungendalter (zwischen 12 und 16 Jahren) hingegen haben vielleicht kein Interesse an einer verspielten Applikation. Für diese Benutzergruppe kann ein Design entwickelt werden, das ein minimalistisches Auftreten hat, die Aufgaben gleichwertig behandelt, jedoch nicht so verspielt ist.

Im Folgenden sollen einige prototypische User Interfaces vorgestellt werden. Abbildung 1.3 auf Seite 28 zeigt drei Home-Screens. Für das Design der oberen zwei wurde die Idee des Gamification-Paradigmas genutzt. Bild (a) aus Abbildung 1.3 zeigt eine Schei-

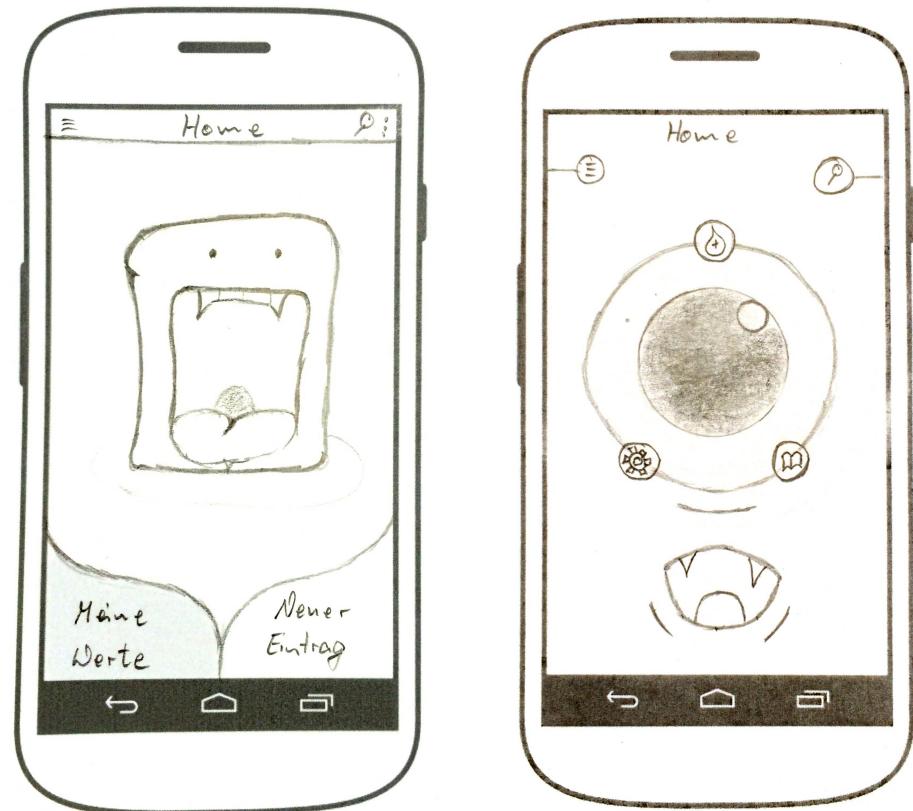
be Toast, das vom Kind mit dem Blutzucker gefüttert werden soll. Dieses „Maskottchen“ soll durch eine Kommunikation mit dem Benutzer diesen dazu animieren regelmäßig die Messungen durchzuführen. Belohnungen seitens des „Maskottchens“ können die Motivation weiter steigern. Die Funktion des „Neuen Eintrags“ und des „Logbuchs“ sind so angebracht, dass sie mit dem Daumen leicht erreichbar sind. Zudem ist die Größe der Buttons so gewählt, dass sie die Sehschwächen kompensieren, die mit einer Diabetes Erkrankung einhergehen.

Bild (b) aus Abbildung 1.3 zeigt einen Home-Screen, der im Ganzen als ein interaktives Medium fungiert. Dargestellt wird ein großes Auge und ein Maul. Durch eine Variation der Hintergrundfarbe, die sich je nach Niveau des Blutzuckerwertes ändert, und der Änderung des Auges und des Mundes, kann ein emotionales Signal an den Benutzer gesendet werden. Eine Abstimmung auf besondere Signalfarben und einer eventuellen Kommunikation mit dem Objekt kann dadurch eine Motivationsteigerung hervorgebracht werden.

Bild (c) zeigt eine Design-Idee für die Applikation der Jugendlichen. Sie ist weniger verspielt und verzichtet auf einen prägnanten Gamification-Aspekt. Dies bedeutet jedoch nicht, dass man diesen Aspekt ganz außer Acht lassen muss, da sicher auch ältere Personen vom Gamification-Prinzip mitgerissen werden können.

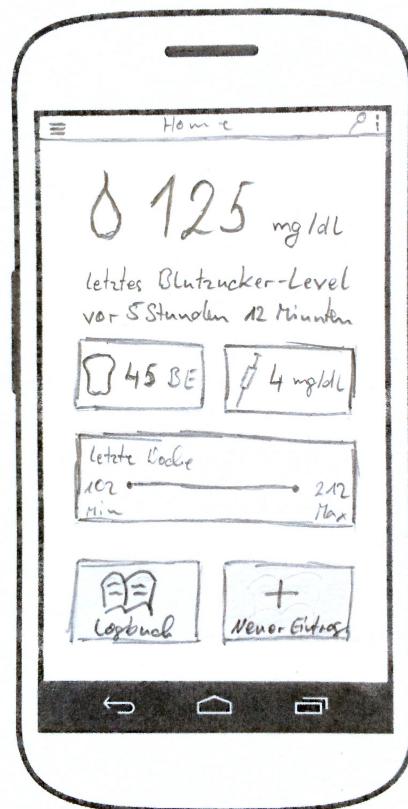
Viele Informationen bezüglich der letzten Erfassung werden direkt auf dem Home-Screen angezeigt. Darunter fällt der letzte Blutzuckerwert, wann die letzte Erfassung getätigkt wurde, der letzte BE/KE-Wert, die letzte Insulinmenge, sowie ein kleine Zusammenfassung der letzten Woche. Die Funktion des „Neuen Eintrags“ und des „Logbuchs“ sind so angebracht, dass sie mit dem Daumen leicht erreichbar sind. Auch in diesem Fall sind die Buttons so groß, dass sie vom Benutzer leicht perzipiert werden können.

Besonders die Interfaces (a) und (b) aus Abbildung 1.3 könnten unter Beachtung des Google Design Prinzips (Android-Design) „**Delight me in surprising ways**“ einen interessanten Mehrwert bei der Motivation des Kindes bringen.



(a) Home-Screen für die Applikation des Kindes.

(b) Alternatives Home-Screen für die Applikation des Kindes.



(c) Home-Screen für die Applikation des Jungendlichen.

Abbildung 1.3: Darstellung dreier Home-Screens.

Abbildung 1.4 auf Seite 29 zeigt zwei Menü-Screens. Für beide Menüs wurden große Buttons verwendet. Dies lässt sich auf die Erkenntnisse aus den User Profiles und der ARD-Dokumentation (ARD) zurückführen, in denen auf die mit Diabetes verbundenen Sehschwächen eingegangen wird.

Beide Menüs bieten den Zugang zu detaillierteren Einsichten, so z. B. zum Logbuch, zu den Statistiken und zu einem Graphen. Das Menü aus Bild (a) aus Abbildung 1.4 ermöglicht darüber hinaus das Einsehen, wann die letzte Erfassung getätig wurde, das Initiiieren einer neuen Erfassung und eine telefonische Direktwahl an die Eltern zu tätigen, falls irgendwelche Fragen bestehen.

Beide Interfaces zeigen Elemente, die auf das Google Design Prinzip (Android-Design) des „**Real objects are more fun than buttons and menus**“ zurückzuführen sind.

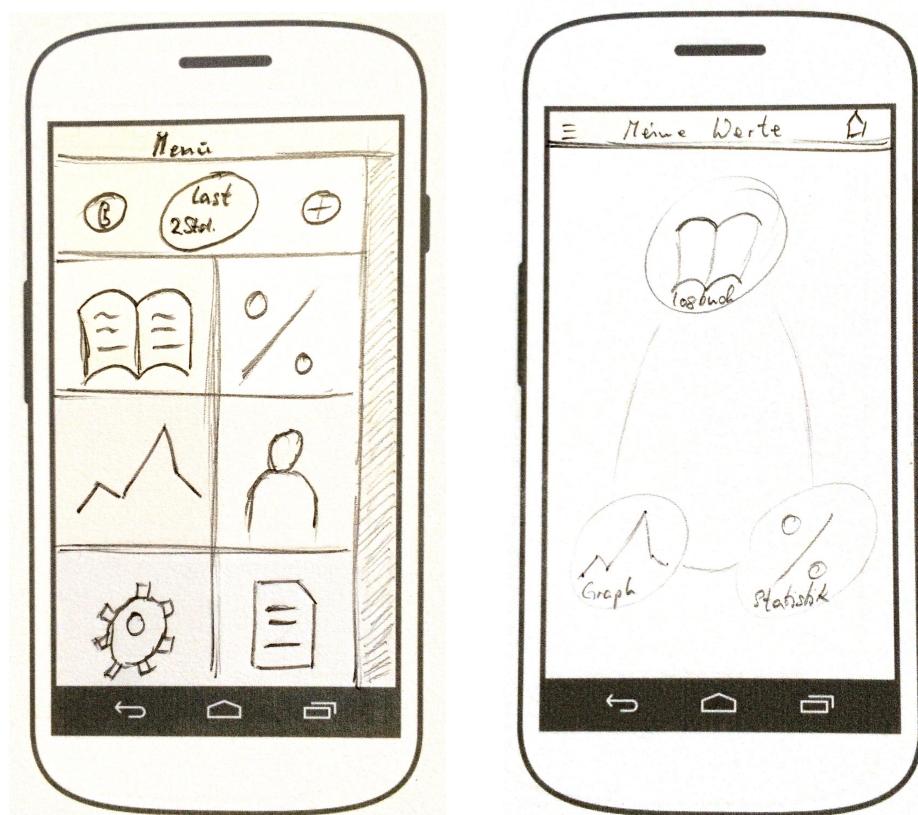
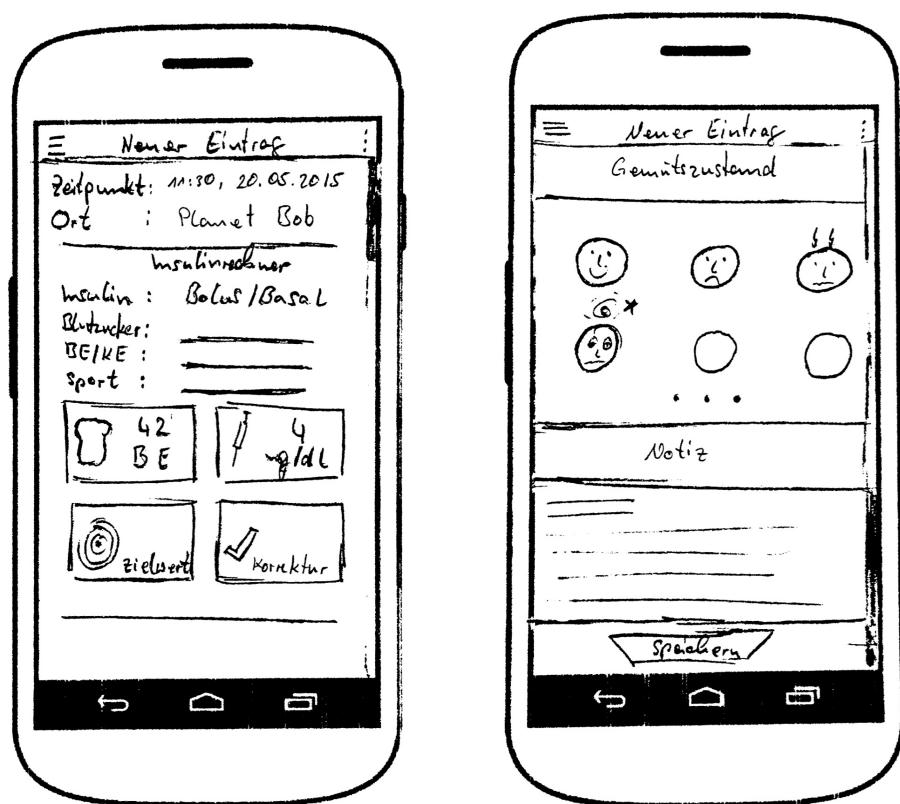


Abbildung 1.4: Darstellung zweier Menü-Screens.

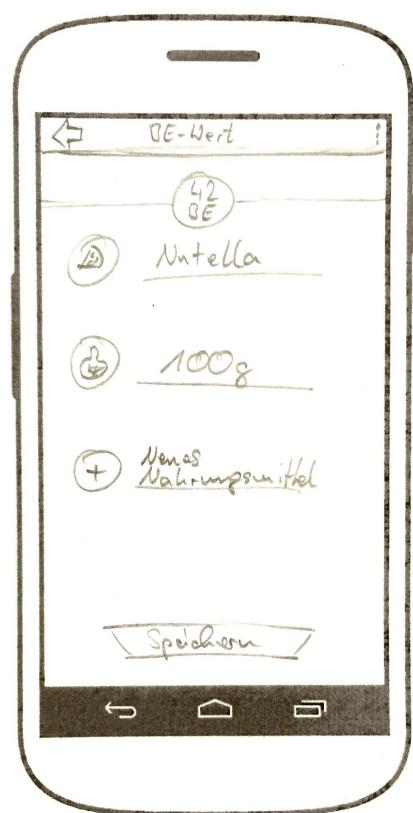
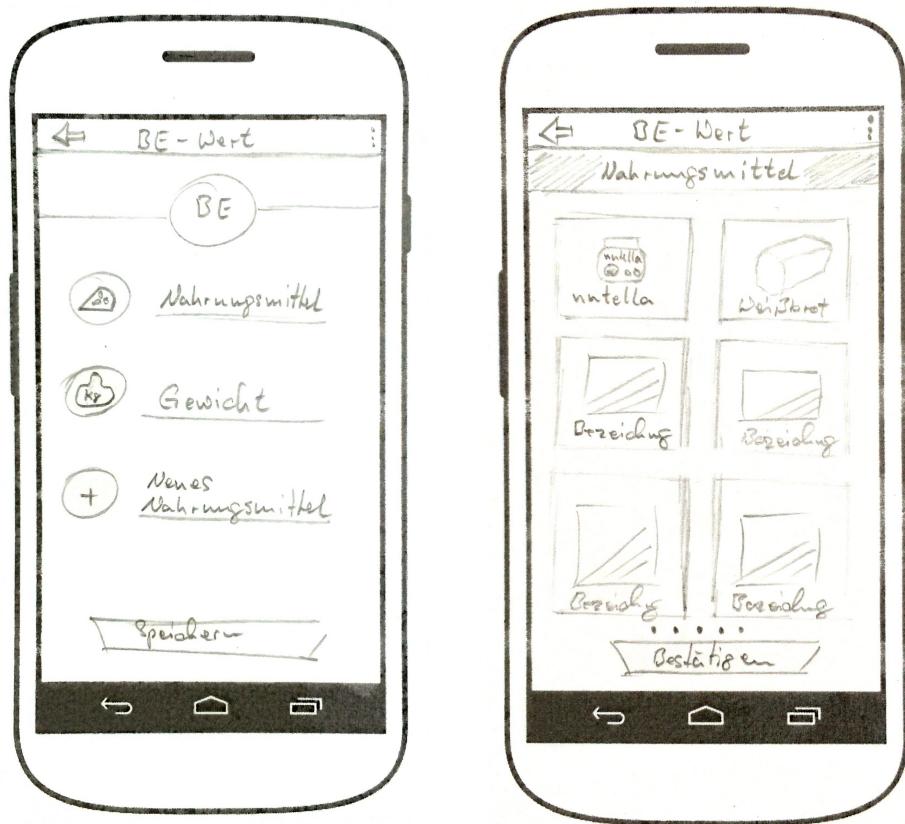
Das Tätigen einer neuen Erfassung wird mit Abbildung 1.5 dargestellt. In der abgebildeten Maske werden einige Werte abgefragt, die für eine Erfassung notwendig sind. Dazu zählen beispielsweise die Uhrzeit (soll automatisch vom System ausgefüllt werden), optional der Ort, der gemessene Blutzucker, die eventuelle sportliche Betätigung, der BE/KE-Wert, ein Gemütszustand und eigene Notizen. Das Notieren der Uhrzeit ist wichtig und muss immer wieder durchgeführt werden. Aus diesem Grund kommt hier das Google Design Prinzip (Android-Design) „**Decide for me but let me have the final say**“ zum Tragen. Die Uhrzeit wird automatisch vom System gesetzt, kann vom Benutzer aber jederzeit geändert werden. Das Bestimmen des BE/KE-Wertes wird durch einen Nebenschritt ausgeführt; dieser soll in Abbildung 1.6 auf Seite 32 dargestellt werden. Hierbei wird aus einem Katalog von Nahrungsmittel die gewünschte ausgesucht und das Gewicht bestimmt. Dieses Verfahren geht einher mit Googles Design Prinzip (Android-Design) „**Pictures are faster than words**“. Anschließend lässt sich die Maske verlassen oder weiter Nahrungsmittel auswählen. Der ermittelten BE/KE-Wert wird dann automatisch an die Erfassung übermittelt.



(a) Interface zur Erfassung eines neuen Eintrags.

(b) Interface zur Erfassung eines neuen Eintrags.

Abbildung 1.5: Darstellung des „Neuer Eintrag“-Screens.

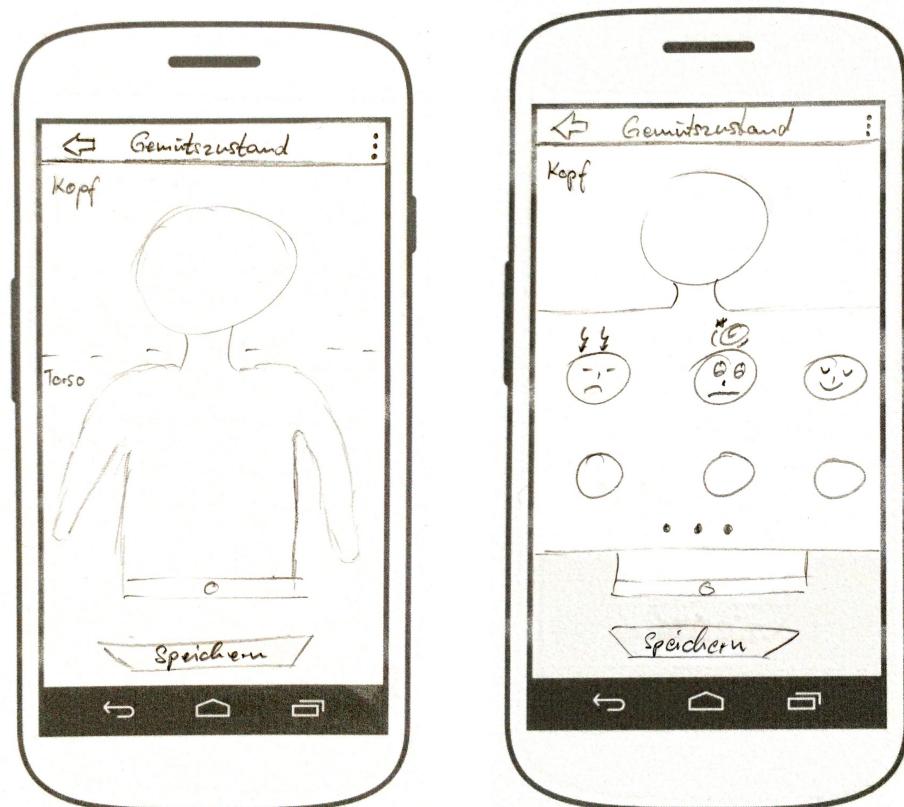


(c) Interface zur Bestimmung des BE/KE-Wertes.

Abbildung 1.6: Darstellung des Screens zur Bestimmung des BE/KE-Wertes.

Das Verstehen des Gemütszustandes ist für junge Diabetiker sehr wichtig, da der Körper auf den Zucker im Blut reagiert und Signale aussendet. Bei jeder Erfassung soll der Benutzer somit die Möglichkeit haben seinen Gemütszustand zu spezifizieren und Notizen zu verfassen. Dies erlaubt ihm bei erneutem Betrachten des Logbuchs das Rekapitulieren von Ereignissen, um so zu verstehen, welche Ereignisse zu möglichen Komplikationen führten. Abbildung 1.7 zeigt eine alternative Möglichkeit zur Bestimmung des Gemütszustandes und eventueller weiterer Beschwerden.

Auch hier lässt sich wieder das Google Design Prinzip (Android-Design) „**Real objects are more fun than buttons and menus**“ anwenden. Durch die Abbildung einer Person kann ein Kind direkt an der Quelle, und somit exakter Beschwerden spezifizieren.



(a) Alternatives Interface zur Bestimmung des Gemütszustandes.

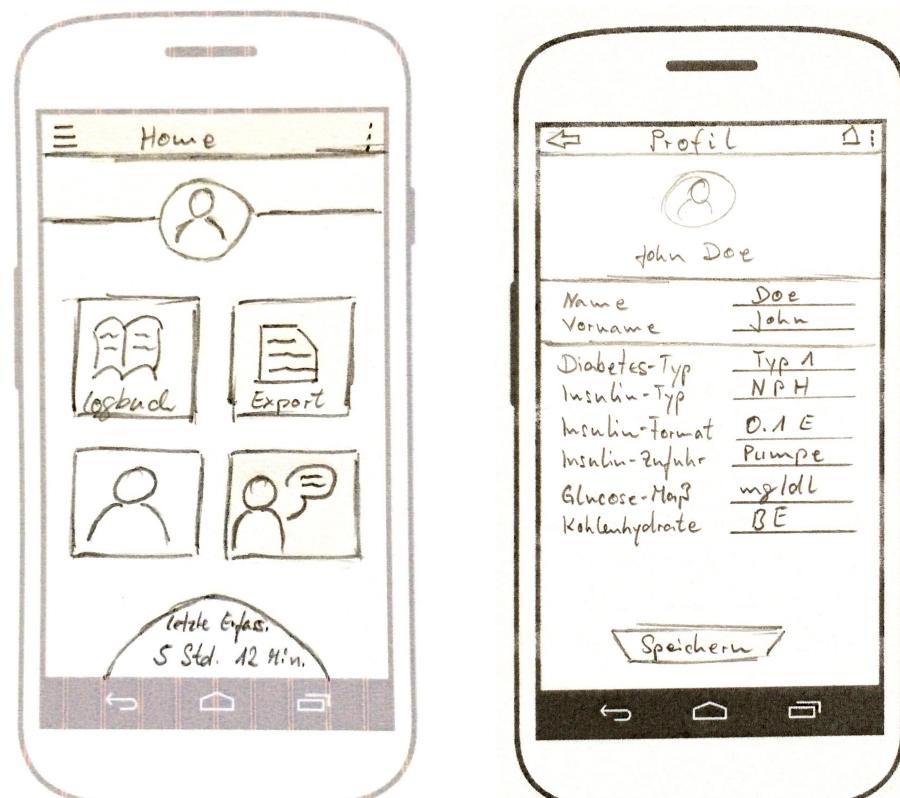
(b) Alternatives Interface zur Bestimmung des Gemütszustandes.

Abbildung 1.7: Alternative Darstellung des Screens zur Bestimmung des Gemütszustandes.

Die Applikation der Eltern dient dem Nachvollziehen und Kontrollieren der Werte ihrer Kinder, sowie dem Kontakt zu andern Eltern, um über wünschenswerte Themen zu diskutieren. Abbildung 1.8 visualisiert die Design-Idee der Eltern. Der Home-Screen (Bild a) liefert direkten Zugriff auf das Logbuch des Kindes, um so die genauen Werte einsehen zu können. Des Weiteren stehen die Funktionen „Exportieren“, „Profil Bearbeiten“ und „Forum“ über den Home-Screen zur Verfügung.

Bild (b) zeigt die Maske, in der das Profil des Kindes bearbeitet werden kann. Hier werden alle nötigen Werte spezifiziert, damit die Applikation des Kindes voll funktionsfähig ist und das Matchmaking einwandfrei funktioniert.

In beiden Interfaces lässt sich das Google Design Prinzip (Android-Design) „**Let me make it mine**“ anwenden, indem man dem Benutzer die Möglichkeit gibt beispielsweise das Bild seines Kindes in das System einzubetten.



(a) Home-Screens der Eltern.

(b) Profils der Kinder.

Abbildung 1.8: Darstellung zweier Screens, die den Home-Screen der Eltern und das Profil des Kindes zeigen.

Das Forum dient der Kommunikation zwischen den Eltern. Abbildung 1.9 visualisiert eine Design-Idee dieses Forums.

Bild (a) zeigt die Auswahl der Kategorien, in denen die jeweiligen Beiträge gepostet werden. Bild (b) zeigt die Themen innerhalb einer ausgewählten Kategorie. Bild (c) zeigt die Maske zur Erstellung eines neuen Beitrags mit der dazugehörigen Auswahl der Kategorie und Topics.

Alle drei Interfaces implementieren das Google Design Prinzip (Android-Design) „**Keep it brief**“, was durch kurze Phrasen dem Benutzer ein unnötig langes Aufhalten in bestimmten Bereichen ersparen soll.

Ebenfalls sollen alle Design Ideen dem Google Design Prinzip (Android-Design) „**I should always know where I am**“ Rechnung tragen, das einhergeht mit der ersten Heuristik von Jakob Nielson (Nielsen) „**Visibility of system status**“ und dem Benutzer Feedback zu seinem Aufenthaltsort innerhalb des Systems liefert.



Abbildung 1.9: Darstellung dreier Screens, die das Forum visualisieren.

Abbildungsverzeichnis

1.1	Reengineered Task Organization Model eines an Diabetes erkrankten Kindes	23
1.2	Reengineered Task Organization Model eines an Diabetes erkrankten Kindes	24
1.3	Darstellung dreier Home-Screens.	28
1.4	Darstellung zweier Menü-Screens.	29
1.5	Darstellung des „Neuer Eintrag“-Screens.	31
1.6	Darstellung des Screens zur Bestimmung des BE/KE-Wertes.	32
1.7	Darstellung des Screens zur Bestimmung des Gemütszustandes.	33
1.8	Darstellung zweier Screens, die den Home-Screen der Eltern und das Profil des Kindes zeigen.	34
1.9	Darstellung dreier Screens, die das Forum visualisieren.	35

Tabellenverzeichnis

1.1 Control Standards	26
---------------------------------	----

Literaturverzeichnis

Android-Design

ANDROID-DESIGN: *Android Design Principles*. <https://developer.android.com/design/get-started/principles.html>. – zuletzt gesichtet am 20.05.2015

ARD

ARD: *Die großen Volkskrankheiten (4): Diabetes - die unterschätzte Gefahr*. <http://www.ardmediathek.de/tv/Reportage-Dokumentation/Die-gro\T1\ssen-Volkskrankheiten-4-Diabete/Das-Erste/Video?documentId=7872680&bcastId=799280>. – zuletzt gesichtet am 27.05.2015

Committee 2008

COMMITTEE, Ergonomics S.: *Ergonomie der Mensch-System-Interaktion – Teil 110: Grundsätze der Dialoggestaltung (ISO 9241-110:2006)*. 09 2008

Google-Design

GOOGLE-DESIGN: *Material Design Color Palette*. <https://www.google.com/design/spec/style/color.html>. – zuletzt gesichtet am 18.06.2015

LoopJ

LOOPJ: *Android Asynchronous Http Client*. <http://loopj.com/android-async-http/>. – zuletzt gesichtet am 24.06.2015

Mayhew 1999

MAYHEW, Deborah J.: *The Usability Engineering Lifecycle: A Practitioner's Guide to User Interface Design*. 1999

Nielson

NIELSON, Jakob: *10 Usability Heuristics for User Interface Design*. <http://www.nngroup.com/articles/ten-usability-heuristics/>. – zuletzt gesichtet am 05.06.2015