InsuLink

Documentation

Produced by: Nicola Dean and Marco Fasanella

CP: 10674826,10617541

Table of Contents

SECTION 1

Idea

When Type 1 Diabetes[1] is diagnosed, a patient starts a new life with different eyes. From now on, the conception of food is completely different from the normal one, and the patient has to assimilate the big change and learn how to handle the disease. One of the most difficult but at the same time important things that the patient must learn, is the ***carbohydrates count*** and subsequently the correct insulin dose for a bolus [2]. InsuLink has been designed with the main purpose of giving an hand to Type 1 Diabetes patient with the calculation of the correct ***insulin doses*** and storing Glycemia values.

1.1 Main Goal

InsuLink main goal is to give a first support to the patient but only if combined with the doctor supervision. It is important to underline that this application is only defined by an algorithm, and in this kind of diseases ***each patient needs ad hoc treatments***.

```
\usepackage
```

or

```
\usepackage{package}
```

SECTION 2

Functionalities

Insulink offers some useful tools to keep track of the daily routine of a patient.

2.1 Food Scan

It is possible to to scan a given Food BarCode and be redirectd to the FoodDetails page with all necessary data.

2.2 Glycemia

Keep track of your daily Glycemia with intuitive charts and easly with the glycemia insertion tool.

2.3 Insulin Calculator

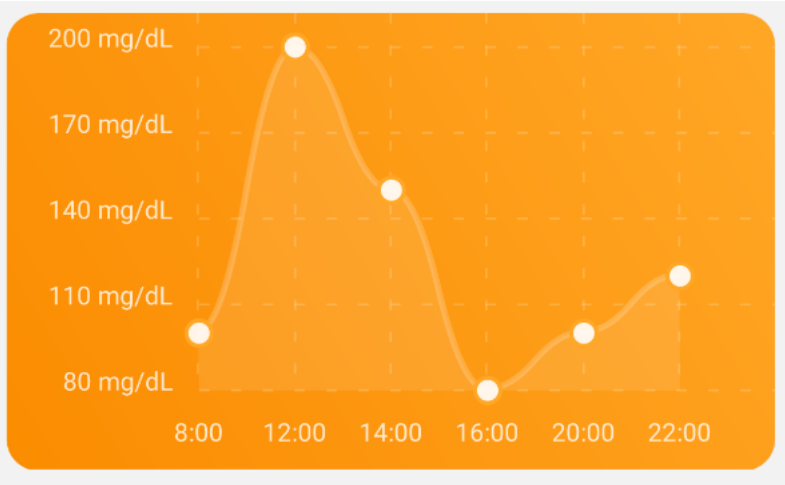
An algorithm (inside Insulin Calculator class) will retrieve last Glycemia, total amount of carbohydrates, sport activity and all essential data to calculate the optimal insulin dose for the given meal. A more detailed explaintation can be fount in the Insulin Calculator Section.

2.4 Calendar

The user can see a well detailed sight of all previous data, just choosing a date from the InsuLink calendar, that will retrieve all the informations about that day from the database.

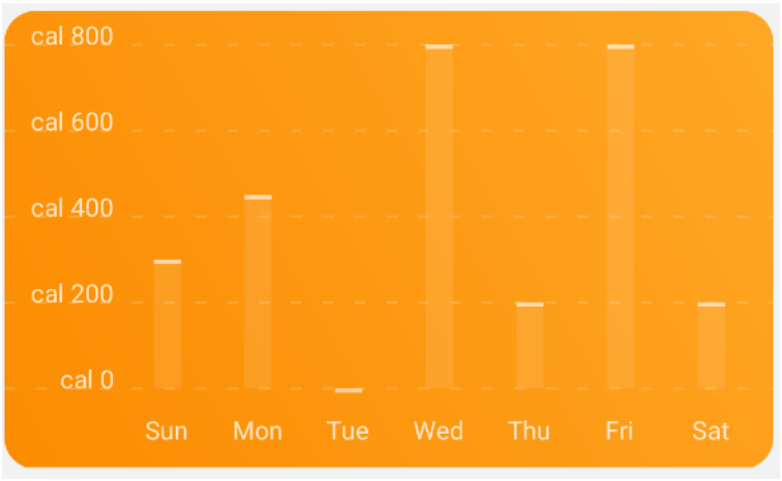
2.5 Charts

Charts are very useful to keep track of the glycemia but also of sport activities. InsuLink offers 4 different charts that allow the user to have a full experience with the app.

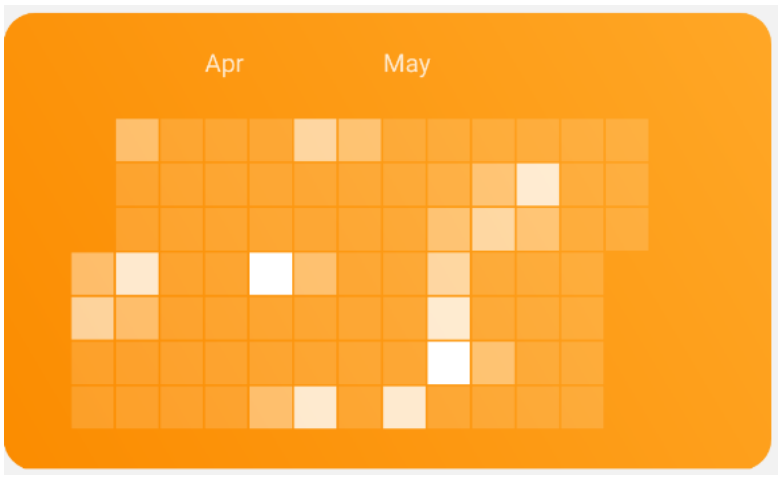


Graph with all bloodsugar misuration of the day, making it useful to see the trend of Glycemia.

Chart with the macro assimilated during the day (with respect to the limits chosen during registration).



Total calories burned per day during the last week.



Represents a heatmap of the sport done during the last months.

SECTION 3

Screens and Navigation

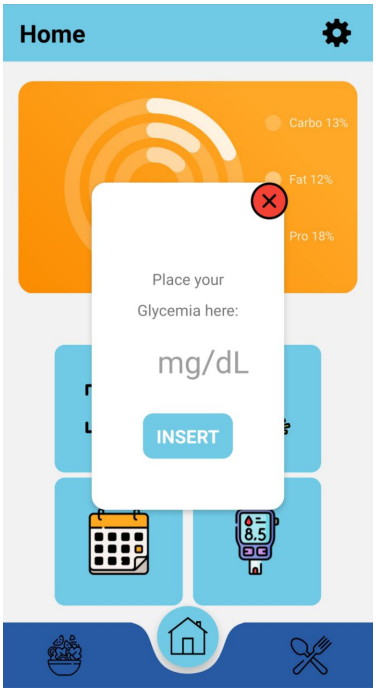
The following provides a screenshot of the pages with a brief description of their use.

3.1 Screens

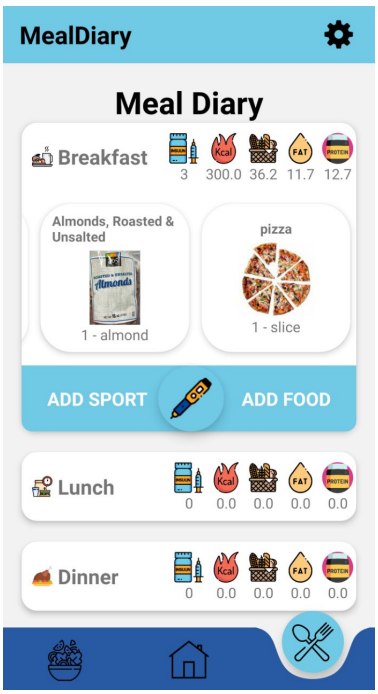
Home Home menu offers shortcuts to the main functionalities and a quick sight of the today glycemia with its intuitive charts.

Search Search food or recipe for nutritional details or to add it in meal diary. User can easly modify the unit measure and quantity of food.

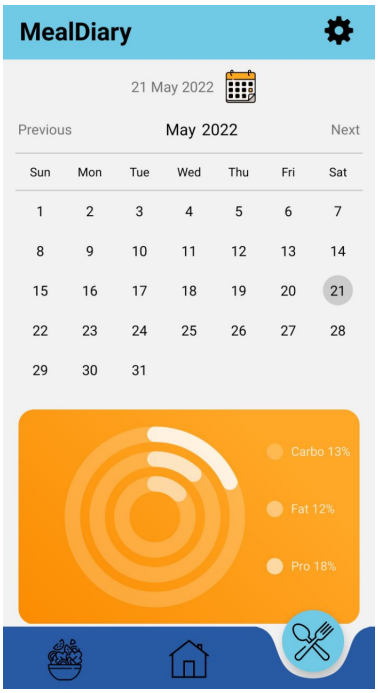
Glycemia PopUp Add glycemia quickly just using the menu shortcut or during the insulin caluclation procedure. The value will automatically stored in Firebase.



Meal Diary Meal Diary can be used for both calulating daily total macro nutrients and insulin dose of each meal.



Calendar In calendar it will be possible to retrieve historical data by clicking on a date.

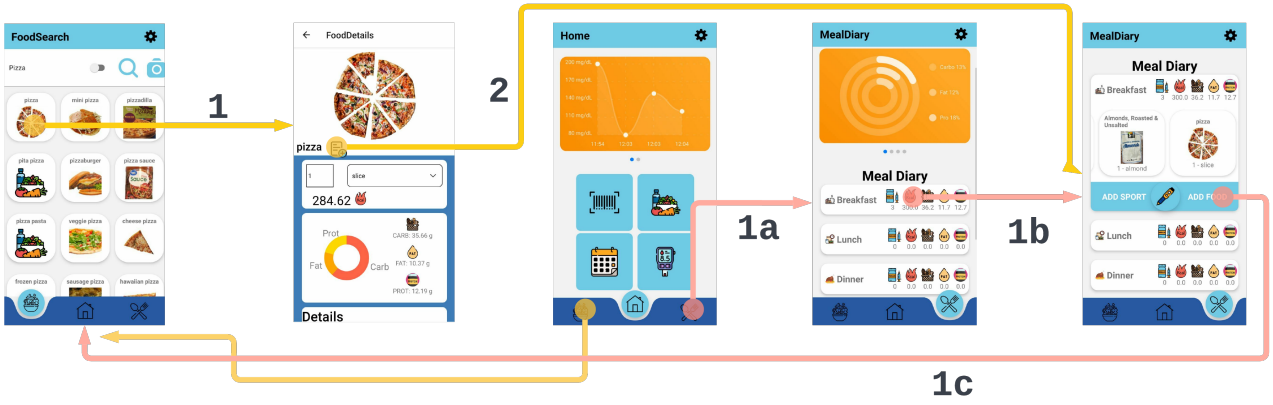


3.2 Navigation

Registration

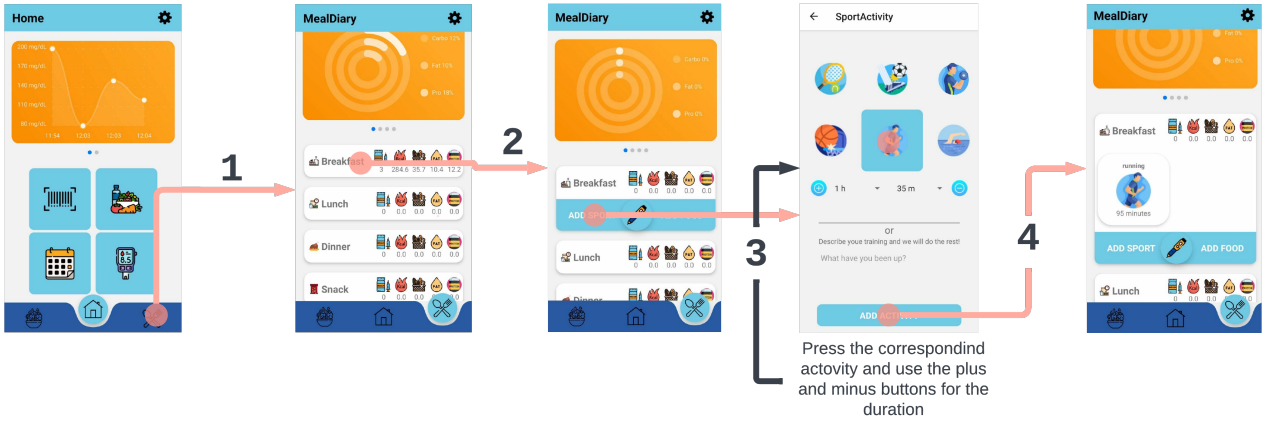
1. Fill Email and Password for registration
2. Then insert name and surname, and select gender and birthday
3. Phiscal informations: choose weight and height
4. Fitness informations: daily macro slider selection and (optionally) Insulin Sensitivity Factor and CHO Ratio

Search and Add Food



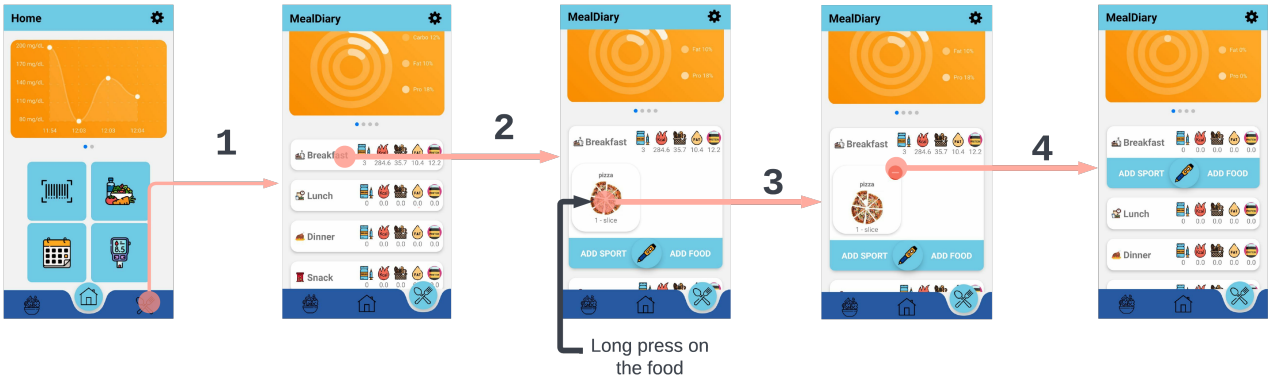
1. Starting from FoodSearch Page (Reachable following the yellow line or the red one 1a-1b-1c), insert a food or recipe name and click on the food item.
2. In FoodDetails page, all informations about food nutrients will be shown. Click on the add button to have the food inserted in the current meal

Add Sport



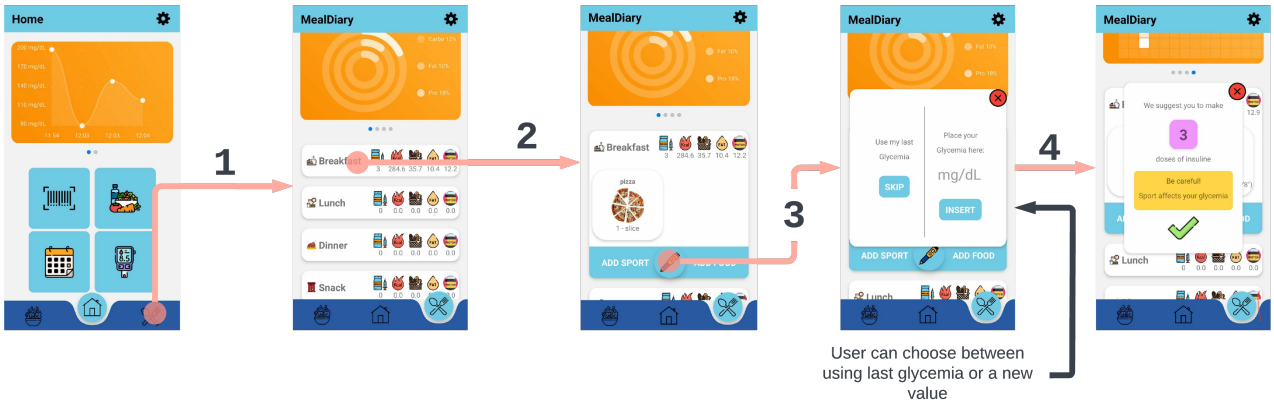
1. From Home page, click on Meal Diary navigation button
2. Expand the current meal and click on "Add Sport" button
3. Select a sport from default one and choose a duration (the user can also just write it on the bottom field)
4. Click on "Add Activity" button to insert it in current meal

Delete Food



1. Click on Meal Diary navigation button
2. Expand the current meal
3. Long press on the food item to delete
4. Click on the delete button to remove the food from current meal

Calculate Insulin



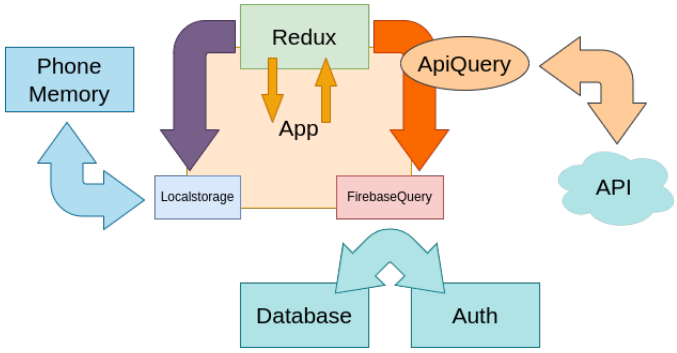
1. Click on Meal Diary navigation button
2. Expand the current meal
3. Click on Insulin Calculation PopUp button
4. Insert the current glycemia or skip this step and get the result

4 SECTION

Architecture

The technology used to make this app is react native [3]. In this chapter will be discussed the architecture of the app at an higher level (more details will be shown on next chapters).

4.1 UML app structure



4.2 Main components

The app is composed of 4 main internal components:

- **Localstorage** manage access of app to local memory
- **FirebaseQuery** manage access of app to firebase database and auth
- **ApiQuery** manage comunication of app with the API nutritionix
- **Redux** manage the interaction of user with app state

Those 4 components together allow the app state to be persistent both localy and remotly if the user close the app.

4.3 Ideal App Flow

- 1. User click on some buttons or search for something.
- 2. This trigger some internal functions.
- 3. Those function then can bring to 2 scenario (possibly mixed) :
  - Api query for some food,activity
  - Redux actions
- 4. Redux action then change the **runtime state** of app and, at the same time, change **asynchronously** the remote firebase/localstorage state.
- 5. Finaly usualy there is a navigation to other page or go back to **1.**

4.4 Folder Structure

`\assets`

Contains all images and component with a proper mapping.

`\constants`

All constants concerning the design and states of the app.

`\customComponents`

All buttons, charts and pickers specifically designed for the pages.

`\pages`

Folder with all pages of the app, using the custom components

`\stateManager`

Redux States for data managing with actions and reducers: macroTracker for meals and userReducer for the patient.

`\utils`

Logic of API, authentication, Firebase and Insulin Calculator

In the next 3 chapters all of those architectural components are explained more in details.

SECTION 5

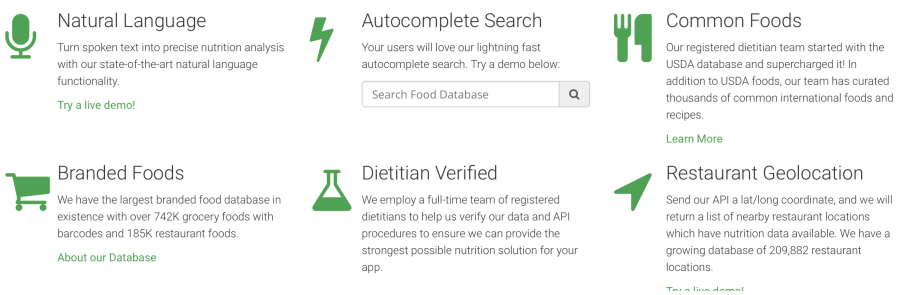
API and Services

InsuLink uses different APIs to get all informations about food and storing user data.

5.1 Nutritionix

Nutritionix [4] is the API used to have a well detailed food database, with all important informations to have a completely working application.

Nutritionix support Natural Language, BarCodes and has a huge dataset with food and recipes. Here there're some functionalities offered by the standard plan:



Following section is dedicated to explain the API and our helper class **/utils/ApiQuery.js**

5.1.1 Api requests

`\search/item GET (barrcode)`

Given upc code (standard use for barcodes) returns the food packaging details.

`\search/instant GET (food query)`

Given a user query (Food Search page) returns a list of possible foods and recipes.

`\natural/nutrients POST (food to search details)`

Given a food it retrive its details.

`\natural/exercise POST (sport query)`

Given a sport query it retrive the estimated calory.

5.1.2 Helper class

The technology used to execute API calls is **Axios**[5] helped by a custom method *doRequests(methods,path,parameter)* the used by other helper functions. Following a list of the api class methods:

```
1       class Api{
2         getSportCalories(userInput);
3         getFoodList(userInput);
4         getFoodListBarCode(barrcode);
5         getIngredientDetails(foodName)
6         doRequest(method,path,params);
7       }
8
```

Listing 1: Api Helper

5.2 Firebase

Firebase [7] is a Google serverless platorm for application development. It is a powerful tool to implement an app database and Google authentication, having many powerful tools to manage it.

All its requests and responses are implemented in the ***src/utils/firebaseQuery.js*** and ***src/utils/auth.js***, two custom helper class.

```
      class FirebaseQuery{
        this.users = firestore().collection(userTable);
        getUser(uid);
        editUser(uid,data);
        getGlicemy(uid,date);
        addGlicemy(uid,date,glicemy);
        getDiary(uid,date);
        saveDiary(uid,date,diary);
      }

```

Listing 2: Firebase Helper

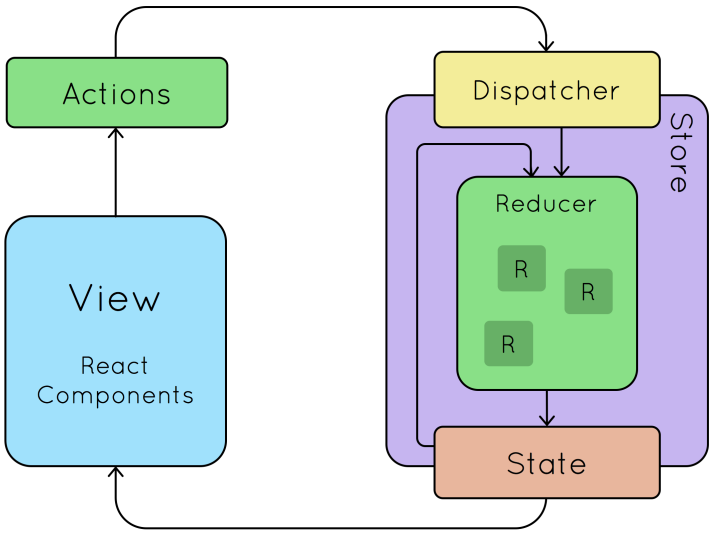
6

SECTION

Redux States

To manage app states we opted for Redux library [9] affiancated with Redux Thunk Middleware [10]

Redux is used mostly for application state management. In other words, Redux maintains the state of an entire application in a single immutable state tree (object), which can't be changed directly.
When something changes, a new object is created (using actions and reducers).



The main two reducers of the app are:

\userReducer

Contains all data about the user, including some additional fields such as Insulin Sensitivity Factor or CHO Ratio.
If not inserted these values will be calculated considering the patient weight.

```
const initialState = {
status:loginStatus.unlogged,
userId:"nullId",
mustCompleteReg:false,
userData:{
  email:"",
  password:"",
  name:"dummy",
  surname:"",
  weight:80,
  height:180,
  birthday:{seconds:0,nanoseconds:0},
  isf:0,
  choratio:0,
  glicemy:[],
  activitys:undefined,
  maxCarb:200,
  maxFat:100,
  maxProt:40,
}

```

Listing 3: UserReducer

\macroTracker

Stores informations about each meal and sport activity of the user. Additionally, cointains the sum of total macros and calories spent during the day.

```
    const initialDiaryState = {
    currentMeal:"breakfast",
    currentDate:FirebaseQuery.glicemyDateFormatter(new Date()),
    totMacro:{cal:0,carb:0,fat:0,prot:0},
    id:1,
    activity_id:1,
    meals:{
        breakfast:{foods:[],macro:{cal:0,carb:0,fat:0,prot:0}},
        lunch:{foods:[],macro:{cal:0,carb:0,fat:0,prot:0}},
        dinner:{foods:[],macro:{cal:0,carb:0,fat:0,prot:0}},
        snack:{foods:[],macro:{cal:0,carb:0,fat:0,prot:0}}
    },
    history:{
       ....
    },
    totCalBurned:0,
    activities:{
        breakfast:{sports:[],totCal:0},
        lunch:{sports:[],totCal:0},
        dinner:{sports:[],totCal:0},
        snack:{sports:[],totCal:0}
    }
}

```

Listing 4: MacroReducer

\errorReducer

Simpy store a lambda function that a custom "<ErrorPopup>" component subscribe

SECTION 7

Insulin Calculator

To understand the algorithm behind the Insulin Calculator, here're some basics:

- Approximately 40-50 percent of the total daily insulin dose is to replace insulin overnight, when you are fasting and between meals
- The other 50-60 percent of the total daily insulin dose is for carbohydrate coverage (food) and high blood sugar correction
- The bolus dose for food coverage is prescribed as an insulin to carbohydrate ratio. The insulin to carbohydrate ratio represents how many grams of carbohydrate are covered or disposed of by 1 unit of insulin.
- The bolus dose for high blood sugar correction is defined as how much one unit of rapid-acting insulin will drop the blood sugar.
- Sport Activity deeply influences the response to insulin and determine a less request of insulin in the patient.

Not considering sport activity, may lead the patient to a low bloodsugar value. Insulink checks if the user has done sport and suggests a new dose value with one unit less. This is not true for all patients, some may need even less amount of insulin.

7.1 Formulas

The main equation for the correct meal dose calculation is:

$$\text{CHO Insulin Dose} = \Sigma \text{ grams of CHO in the meal / grams of CHO disposed by 1 unit of insulin}$$

This happens in ideal conditions where glycemia is in the optimal interval.
Normally this is not the case: after a glucose misuration the patient could find an high bloog sugar. In this case there's a ***correction dose***:

$$\text{High Blood Sugar Correction Dose} = \Delta \text{ (Actual blood sugar - Target blood sugar)} \div \text{Correction Factor}$$

The result formula will be:

$$\text{CHO Total Dose} = \text{CHO Insulin Dose} + \text{High Blood Sugar Correction Dose}$$

Moreover, there's the sport activity to take into account and it will affect the value by decresing it.

8

SECTION

Testing

To perform automated and personalized testing it was used Jest [6] and React Test Renderer [12]. It is a JavaScript Testing Framework that supports React Native. To do so it was used the mocking technique to check the behaviour of components.
Tests were contained inside:

`\__tests__`
: This folder is automaticaly detected by the jest configuration as the test folder

`\__mock__`
: This folder, combined with a setup.js file contribute to the mocking system of the app

8.1 Testing Strategy

The testing strategy is composed on a mix of 3 main technique:

- **Snapshots** to detect when static pages or redux states dosn't behave correctly
- **Widget test** for buttons/events/redux actions
- **Unit Testing** for our helper classes

The next chapters will be dedicated on the description of this strategy.

8.2 Folder Structure

`\api-test`

Unit testing on all the components that access to **API,localstorage or firebase**.

`\redux-test`

Snapshot testing on the **reducers results**.

`\renders-test`

Widget and Snapshot testing on all the **main pages and components** (static screenshot).

`\utils`

Unit testing on the most importants **utils classes**

8.3 Test Suites

The tests are subdivided into 16 suites

Utils class testing

As explained before, the app contain 2 important utilities class, this suit aim to apply Unit Testing on them

insulin-calculator	1. Check correct calculation
input-checker	1. Actual Registration Errors 2. Personal Info Errors 3. Physical Info Errors

Firebase testing

An "Ad hock" class **CustomFirestoreMock** have been created to mock Firebase component. It contains a mock for doc,collection,get,set,data methods and allow to mock the return value of get() and data() functions.

getters	1. Get User Data 2. Get Food Diary 3. Add Glycemia Data
setters	1. Set Food Diary 2. Save User Data 3. Get Glycemia Data

Api Helper testing

Axios class have been mocked using an "ad hoc" jest.fn() that return a Promise. TODO put here CODE OF AXIOS MOCK

Food related query	1. Search for foods 2. Get Food Details 3. Get Food by barrcode
Sport related query	1. Get Activity calories

Local Storage testing

As for the previous classes, also AsyncStorage needed mock but it comes already distributed with the original module. It is called **Async Storage Mock**, we stored an automatic import/mock inside the **mock folder**

getters	1. Get User Data 2. Get Food Diary 3. Add Glycemia Data
setters	1. Set Food Diary 2. Save User Data 3. Get Glycemia Data

Redux testing

The redux testing was done by snapshotting the reducer output uppon a certain input.
Since reducers are **pure methods**, no mock was needed.

macroReducer	1. Add,Edit,Remove Food 2. Add,Remove Sport 3. Select Meal 4. Reset Diary
userReducer	1. Login 2. Update Data 3. Add Glycemia 4. Logout

Pages widgets and snapshots

For all main pages of our app we have a snapshot test + UI events testing. To make tests the mocking of firebase/ApiHelper and other components was neded. A mocked version of redux was provided to a "<Provider>" components to emulate state changing behaviour.

Food	1. Snapshot 2. Snapshot for deletable version 3. OnPress navigation
Food Details	1. Snapshot 2. Add Food 3. Edit Food 4. Delete Food (Checked both redux state and navigation)
Meal Diary	1. Snapshot 2. Click on meal(navigation) 3. Click on AddFood/Sport (Checked navigation)
Home	1. Snapshot 2. Click on the 4 Buttons (check navigation to correct page)
Login	1. Snapshot 2. Click on login button 3. Check if "notLogged" status is redirected to Login Page (check redux state changed)
Registration	1. Snapshot 2. Click on Reg button 3. Check on Errors wrong inputs

CustomComponents snapshots

All the static custom components of the app have been tested with react-test-renderer + snapshot. The following list show the tested components:

- WaitLoading

- CustomButton
- CustomImageButton
- CustomCalendar
- InputBlock
- InputContainer

SECTION 9

Future Implementations

Insulink has been structured with the possibility of implementing new technologies inside it.

9.1 API

The API utils section of the code is easly changeble from one provider to another. Using a premium API would affect the performance but also the usability of the app.

9.2 Machine Learning and AI

Calculating the correct insulin dose is a really difficult problem. The factor that influences the output is not only the amount of carbohydrates eaten, but many other features: fats, sport activity, emotional condition and sometimes even the weather.
Moreover each patient needs specific treatments, and has a different resistance to insulin.
Using Machine Learning in this field could be a smart way to correctly predict the optimal insulin dose.

9.3 NFC Glucose Meter

Some Glucose Meters use new technologies to simplify diabetic patients life. One famous example is FreeStyle Libre [8]. Using the NFC technology to retrieve glycemia helps not only in terms of time but also in visualization and store of data.
Once implemented, the user has just to bring the phone closer to the sensor and the app will check the glucose (and store it).

10

SECTION

References

[1] Diabetes Definition
https://en.wikipedia.org/wiki/Type_1_diabetes

[2] Bolus Definition
https://en.wikipedia.org/wiki/Bolus_(medicine)

[3] React Native
https://reactnative.dev

[4] Nutritionix
https://www.nutritionix.com

[5] Axios
https://axios-http.com/docs/intro

[6] Jest
https://jestjs.io

[7] Firebase
https://firebase.google.com

[8] FreeStyle Libre
https://www.freestyle.abbott/us-en/home.html

[9] Redux
https://redux.js.org

[10] Redux
https://redux.js.org

[11] Source for Insulin Calculator
https://dtc.ucsf.edu/types-of-diabetes/type1/treatment-of-type-1-diabetes/medications-and-therapies/type-1-insulin-therapy/calculating-insulin-dose/

[12] Source for React Test Renderer
https://it.reactjs.org/docs/test-renderer.html

Credits

Special thanks to Matteo Ciancio who designed the app logo. For any business contact you can reach him at: matteociancio99@gmail.com