BeFit

Calorie Tracker application

Software design & systems

Project designed and implemented by:

Tövisi Zoltán & Nyitrai Orsolya-Éva

Informatics class of 2020

2023

Table of contents:

1. Introduction

Our project’s purpose is to help people keep track of their calorie intake. Nowadays many people want to lose weight, and it is proven that the only way to do that is to be in a calorie deficit. So, we created BeFit, an app that helps people to count their calories easily.

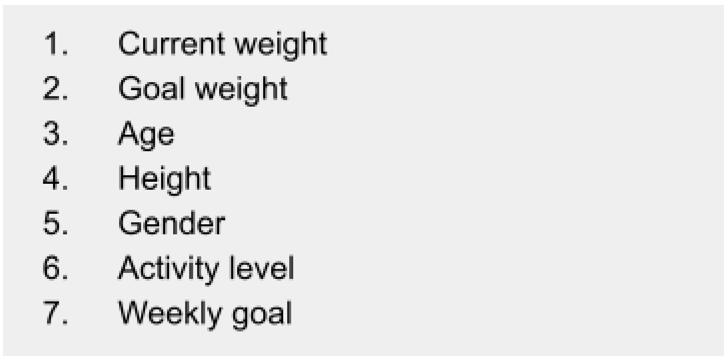
Furthermore, the app can keep track of the water that the individual drinks because it is known that hydration is the first step in the weight loss process. They can also add the amount of exercise they’ve done so that the app will increase their calorie budget for that day, as a result of exercise burning calories.

2. Requirements

2.1 User requirements

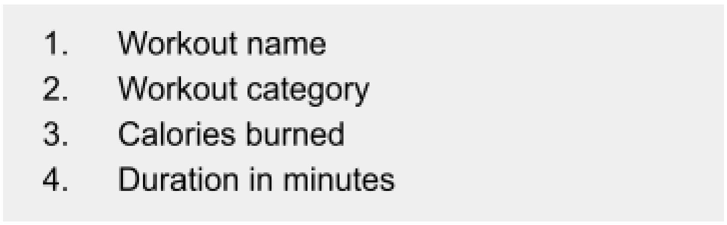
In order to use our app, which utilizes Google's Fitness API, every user must have a Google account.

After completing the initial sign-in process each user must fill out an onboarding form, to customize their account. This involves entering information about relevant health and fitness data:



1. Current weight
2. Goal weight
3. Age
4. Height
5. Gender
6. Activity level
7. Weekly goal

To create a new workout entry the users are required to enter the following information:

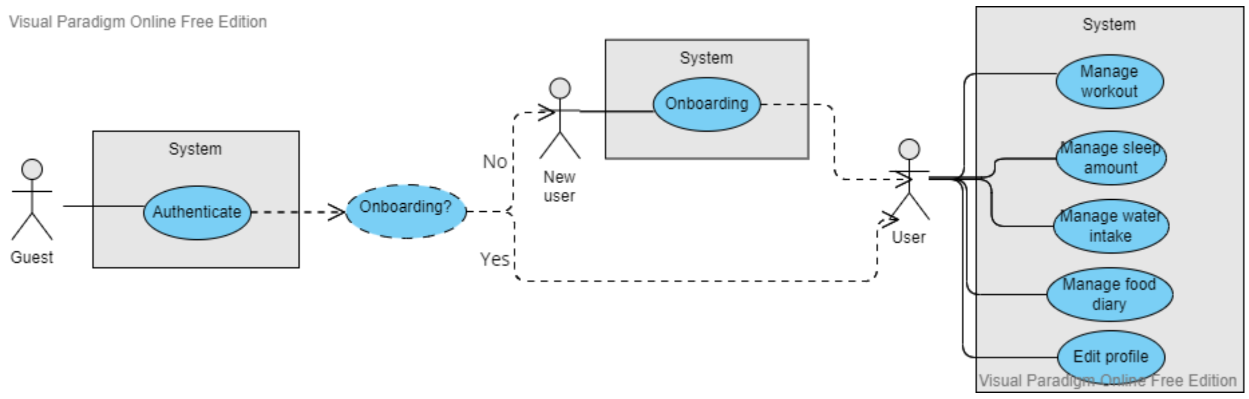


1. Workout name
2. Workout category
3. Calories burned
4. Duration in minutes

To create a new food entry the users must provide the following data:

1. Food name
2. Calories
3. Meal type
4. Image (optional)

2.2 Use cases



[Figure 2.2.1]

Figure 2.2.1 illustrates the flow o the application, below are listed and described the use cases in more detail:

UC-0001	**Sign-in**
Actors	Guest
Description	The guest can sign in with an existing account to use the rest of the application.
Preconditions	• The Guest isn't already signed in • The Guest has an existing Google Account
Postconditions	• The Guest is now a User • The User navigates to the dashboard page of the application
Normal course	• The user authenticates with a Google account • The User is signed in and sent to the dashboard layout

UC-0002	**Onboarding**
Actors	New User
Description	The new User must provide us with relevant health and fitness information.
Preconditions	● The User is signed in. ● This is the first time the user signed in to our page.
Postconditions	● The new User is now a normal User. ● The User navigates to the dashboard page of the application
Normal course	● The user fills out the onboarding form and submits it ● The User is sent to the dashboard layout ● The User can now view and edit his/her daily fitness information

UC-0003	**Manage workout**
Actors	User
Description	The User can add/edit/delete workout entries.
Preconditions	● The User is signed in. ● The User is on the dashboard layout page.
Postconditions	● The action updates the database. ● The action changes the daily calorie intake
Normal course	● The User presses the add/edit/delete workout button ● Database is updated with submitted data ● The daily calorie intake changes

UC-0004	**Manage sleep amount**
Actors	User
Description	The User can add/edit/delete sleep entries.
Preconditions	● The User is signed in. ● The User is on the dashboard layout page.
Postconditions	● The action updates the database.
Normal course	● The User presses the edit sleep amount button ● Database is updated with submitted data

UC-0004	**Manage water intake**
Actors	User
Description	The User can add/delete water entries.
Preconditions	● The User is signed in. ● The User is on the dashboard layout page.
Postconditions	● The action updates the database.
Normal course	● The User presses the add/remove water intake button ● Database is updated with submitted data

UC-0005	**Manage food entries**
Actors	User
Description	The User can add/edit/delete food entries.
Preconditions	● The User is signed in. ● The User is on the dashboard layout page.
Postconditions	● The action updates the database. ● The action changes the daily calorie intake
Normal course	● The User presses the add/edit/delete food entry button ● Database is updated with submitted data

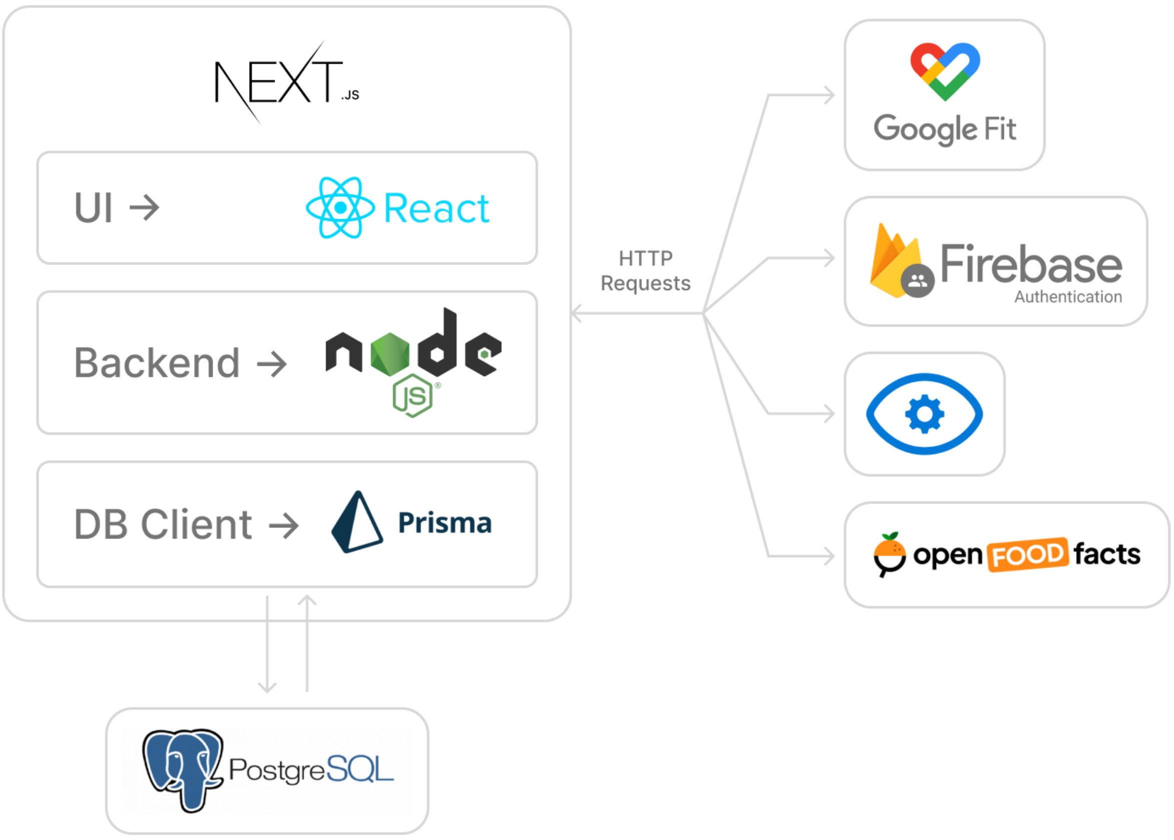
UC-0006	**Edit user profile**
Actors	User
Description	The User can edit his/her profile page.
Preconditions	● The User is signed in. ● The User is on the profile page.
Postconditions	● The action updates the database. ● The action changes the User’s fitness information and daily calorie intake
Normal course	● The User presses the Edit Profile button ● The User fills out the form with the updated information ● System will recalculate the daily calorie intake ● Database is updated with submitted data

2.3 System requirements

The system needs to have a web browser installed and it should be connected to the internet. To be able to use the image search features without having the image on the system, it has to be connected to a working camera.

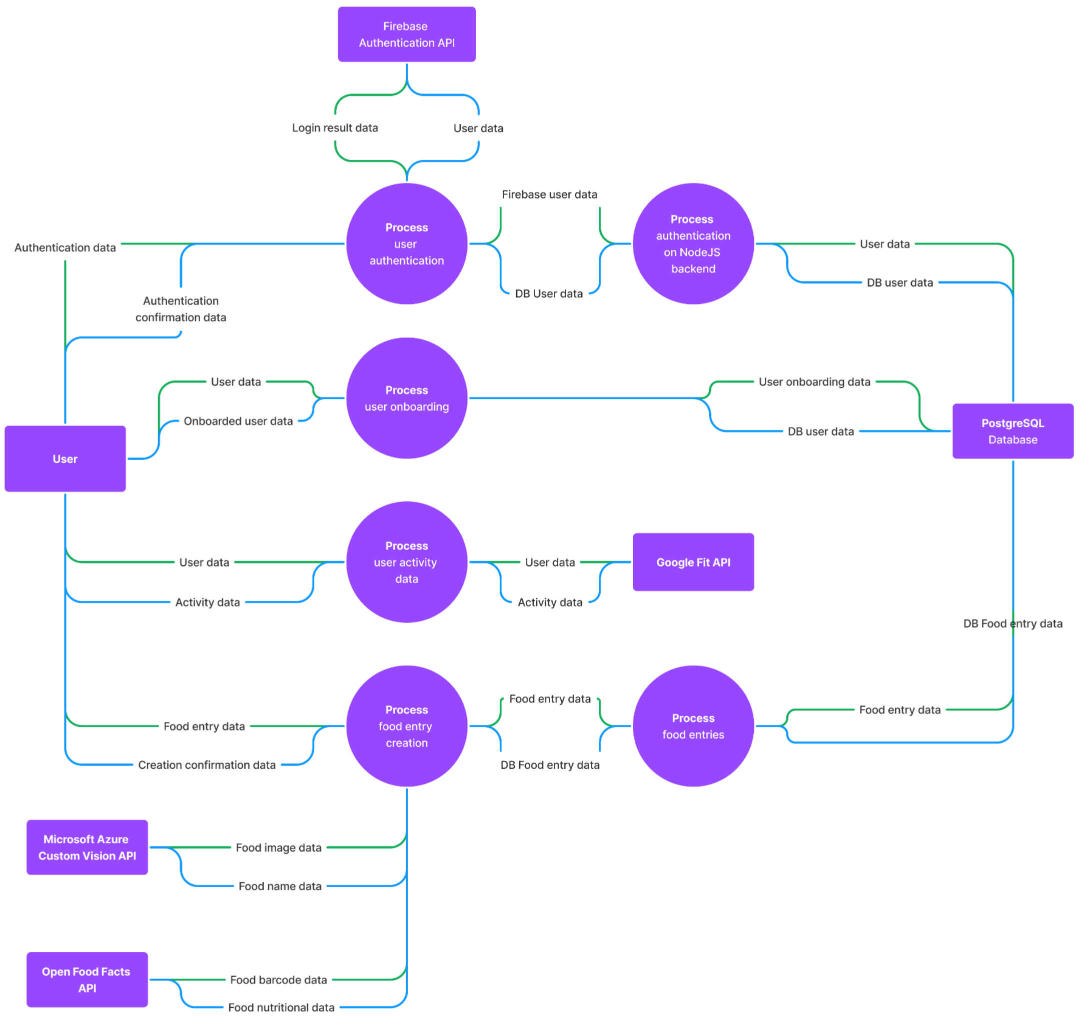
3. System design & architecture

3.1 System architecture



[Figure 3.1.1]

The application is built using a Microservices architecture. This is a design pattern in which a more extensive application is made as a suite of small, independent services that communicate with each other using well-defined APIs. The architectural diagram (*see Figure 3.1.1),* shows how our application is split into multiple services, some built by us and some external.

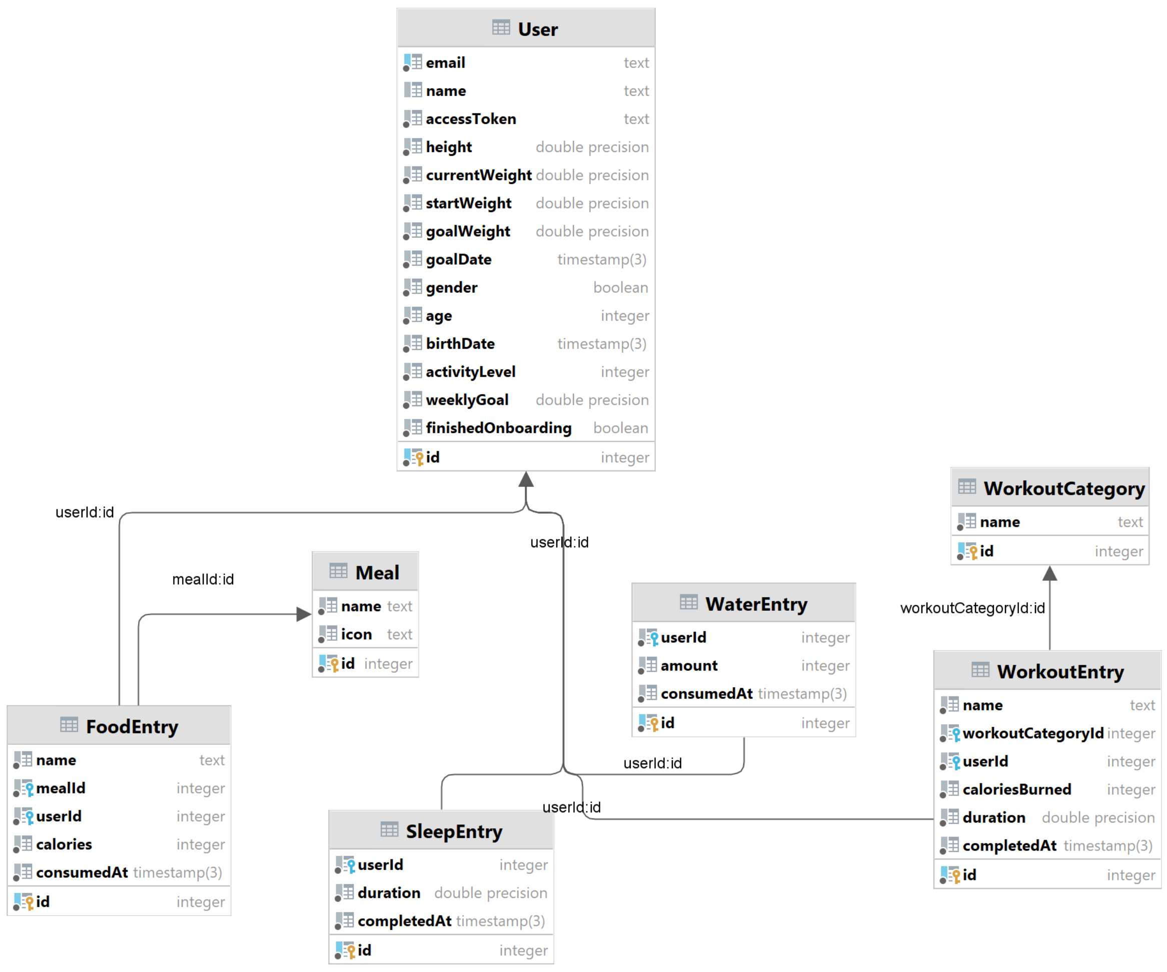


[Figure 3.1.2]

The diagram above (*see Figure 3.1.2*) shows how the data flows between the main parts of our application(only the important data flows were included for image clarity purposes).

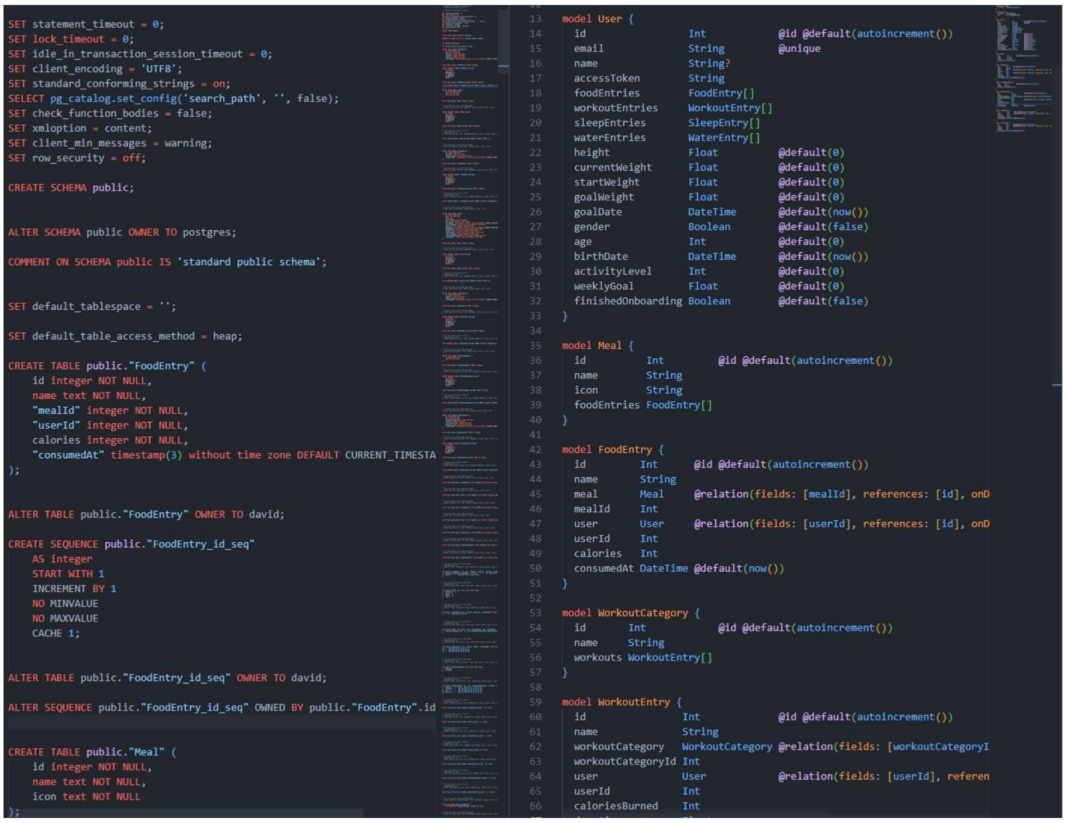
We have built the front end of our application using the Next.js framework, which allows the developers to create Single Page Applications using React.js and other tools built into the framework. React allows us to manage states using hooks, but it can get messy quickly if we have more complex states on a single page. For this reason, we decided to use Zustand, a lightweight, easy-to-use state management library for React. To improve the developer experience when developing the user interface we used TailwindCSS, a utility-first CSS framework for rapidly building custom designs.

The Next.js framework allowed us to easily build a NodeJS-based REST API, which we used to handle the data access and business logic related to the database. One of our objectives was to have a well-abstracted and easy to work with database handling method. For this purpose, we used Prisma, an open-source database toolkit that provides a feature-rich Object Relational Mapper and a very intuitive query builder and schema management method.



[Figure 3.1.3]

For the database itself, we used PostgreSQL, a very robust and powerful DBMS, that allows working with a wide range of data types.
You can see the diagram of our database in the figure above. (*see Figure 3.1.3*)



```sql
SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

CREATE SCHEMA public;


ALTER SCHEMA public OWNER TO postgres;

COMMENT ON SCHEMA public IS 'standard public schema';


SET default_tablespace = '';

SET default_table_access_method = heap;

CREATE TABLE public."FoodEntry" (
    id integer NOT NULL,
    name text NOT NULL,
    "mealId" integer NOT NULL,
    "userId" integer NOT NULL,
    calories integer NOT NULL,
    "consumedAt" timestamp(3) without time zone DEFAULT CURRENT_TIMESTA
);


ALTER TABLE public."FoodEntry" OWNER TO david;

CREATE SEQUENCE public."FoodEntry_id_seq"
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE public."FoodEntry_id_seq" OWNER TO david;

ALTER SEQUENCE public."FoodEntry_id_seq" OWNED BY public."FoodEntry".id


CREATE TABLE public."Meal" (
    id integer NOT NULL,
    name text NOT NULL,
    icon text NOT NULL
);
```

```prisma
model User {
  id                Int           @id @default(autoincrement())
  email             String        @unique
  name              String?
  accessToken       String
  foodEntries       FoodEntry[]
  workoutEntries    WorkoutEntry[]
  sleepEntries      SleepEntry[]
  waterEntries      WaterEntry[]
  height            Float         @default(0)
  currentWeight     Float         @default(0)
  startWeight       Float         @default(0)
  goalWeight        Float         @default(0)
  goalDate          DateTime      @default(now())
  gender            Boolean       @default(false)
  age               Int           @default(0)
  birthDate         DateTime      @default(now())
  activityLevel     Int           @default(0)
  weeklyGoal        Float         @default(0)
  finishedOnboarding Boolean      @default(false)
}

model Meal {
  id          Int         @id @default(autoincrement())
  name        String
  icon        String
  foodEntries FoodEntry[]
}

model FoodEntry {
  id         Int      @id @default(autoincrement())
  name       String
  meal       Meal     @relation(fields: [mealId], references: [id], onD
  mealId     Int
  user       User     @relation(fields: [userId], references: [id], onD
  userId     Int
  calories   Int
  consumedAt DateTime @default(now())
}

model WorkoutCategory {
  id       Int            @id @default(autoincrement())
  name     String
  workouts WorkoutEntry[]
}

model WorkoutEntry {
  id                Int             @id @default(autoincrement())
  name              String
  workoutCategory   WorkoutCategory @relation(fields: [workoutCategoryI
  workoutCategoryId Int
  user              User            @relation(fields: [userId], referen
  userId            Int
  caloriesBurned    Int
```

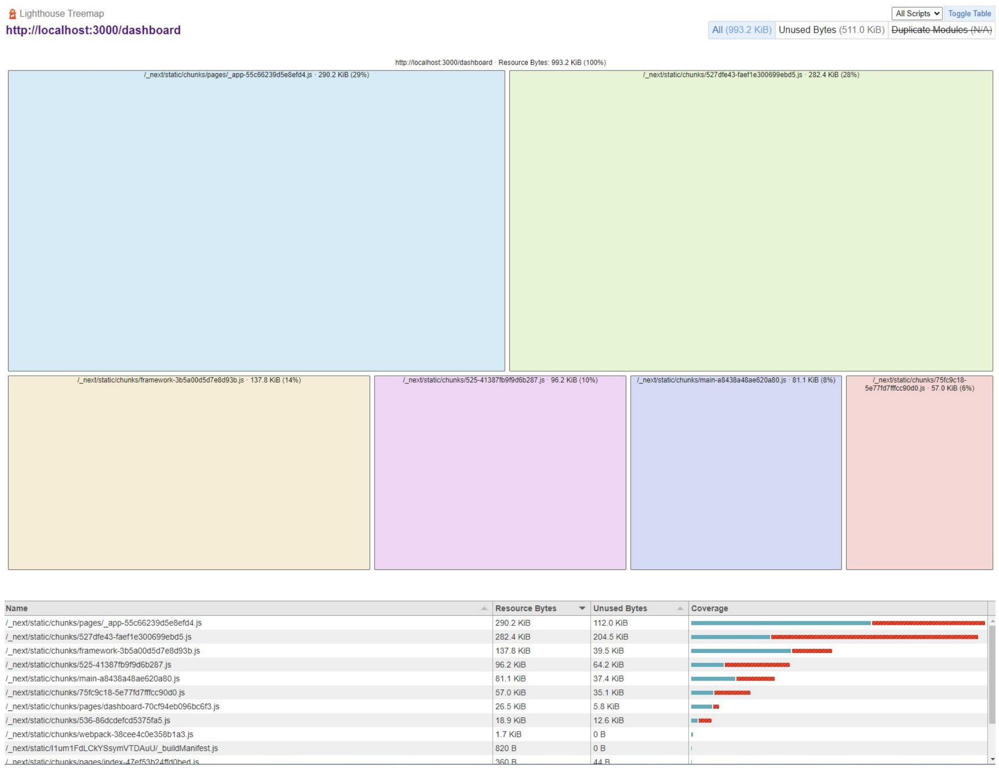
[Figure 3.1.4]

The image above (*see Figure 3.1.4*) shows the difference between creating the database schema using regular SQL queries or the Prisma schema builder. As you can see, Prisma has a more readable and object-oriented syntax, taking care of the relationships and keys in the background(it allows us to do it manually as well if we need to). Creating the same database with SQL queries allows for more developer errors, and it requires at least twice as much code as the Prisma method.

We decided not to handle the authentication ourselves because it wouldn't be as safe as a professional solution. So we decided to use the Firebase Authentication API, and we enabled authentication with a third-party provider, Google. For acquiring the user's activity data, like steps and traveled distance we decided to go with the Google Fit API. When signing in with Firebase the users are prompted to allow access to their fitness data. If they do the Auth API sends us an access token that we can use to retrieve the user data from the Fit API.

As it wouldn't be the best user experience to always have to write in the name and the calories of the foods that the users consume, we decided to implement two extra features that improved this experience. The first one was adding a QR Code reader for the user to scan the barcodes of the foods that they ate. After scanning a code we send a request to the Open Food Facts API, which provides nutritional information for more than 920,000 products based on their barcode. The second feature is for when the user consumes something that doesn't have a barcode. We have trained an AI image classifier using the Microsoft Azure Custom Vision(https://www.customvision.ai/) service so that the users can snap a photo of a food item and the classifier returns the name of the food item. A future improvement to this would be to train the AI based on the food entries on the application to try and approximate the calories as well, but for this lots of data would be necessary for training.

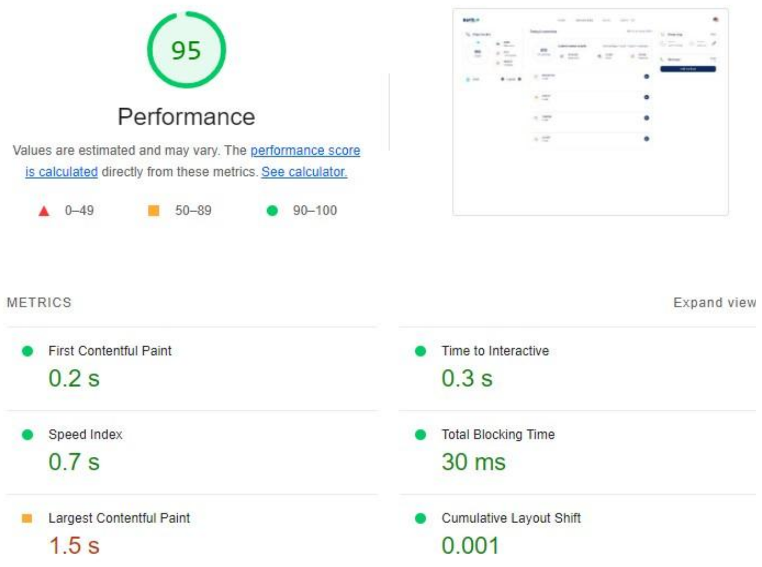
Last, but not least, we wanted to focus on code readability as well and have a very easy-to-navigate codebase, so we have used the ESLint and Prettier tools to make sure we adhere to specific coding standards and that our code is always well formatted.


Lighthouse Treemap
http://localhost:3000/dashboard
All Scripts
Toggle Table
All (993.2 KiB)
Unused Bytes (511.0 KiB)
~~Duplicate Modules (N/A)~~
http://localhost:3000/dashboard · Resource Bytes: 993.2 KiB (100%)
/_next/static/chunks/pages/_app-55c66239d5e8efd4.js · 290.2 KiB (29%)
/_next/static/chunks/527dfe43-faef1e300699ebd5.js · 282.4 KiB (28%)
/_next/static/chunks/framework-3b5a00d5d7e8d93b.js · 137.8 KiB (14%)
/_next/static/chunks/525-41387fb9f9d6b287.js · 96.2 KiB (10%)
/_next/static/chunks/main-a8438a48ae620a80.js · 81.1 KiB (8%)
/_next/static/chunks/75fc9c18-5e77fd7fffcc90d0.js · 57.0 KiB (6%)

Name	Resource Bytes	Unused Bytes	Coverage
/_next/static/chunks/pages/_app-55c66239d5e8efd4.js	290.2 KiB	112.0 KiB	
/_next/static/chunks/527dfe43-faef1e300699ebd5.js	282.4 KiB	204.5 KiB	
/_next/static/chunks/framework-3b5a00d5d7e8d93b.js	137.8 KiB	39.5 KiB	
/_next/static/chunks/525-41387fb9f9d6b287.js	96.2 KiB	64.2 KiB	
/_next/static/chunks/main-a8438a48ae620a80.js	81.1 KiB	37.4 KiB	
/_next/static/chunks/75fc9c18-5e77fd7fffcc90d0.js	57.0 KiB	35.1 KiB	
/_next/static/chunks/pages/dashboard-70cf94eb096bc6f3.js	26.5 KiB	5.8 KiB	
/_next/static/chunks/536-86dcdefcd5375fa5.js	18.9 KiB	12.6 KiB	
/_next/static/chunks/webpack-38cee4c0e358b1a3.js	1.7 KiB	0 B	
/_next/static/l1um1FdLCkYSsymVTDAuU/_buildManifest.js	820 B	0 B	
/_next/static/chunks/pages/index-47ef53b24ffd0bed.js	360 B	44 B	

[Figure 3.1.5]

The image above (*see Figure 3.1.5*) shows us one of the advantages of using the Microservices architecture: the small size of the application being served to the user. The whole application is just 900 Kbs, out of which 500 are unused. This means that realistically we could achieve an even smaller bundle size(approx 500 Kb) if we use a better JS bundler(for example Vite), or if we try and fine-tune our current bundler(Webpack).



95

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

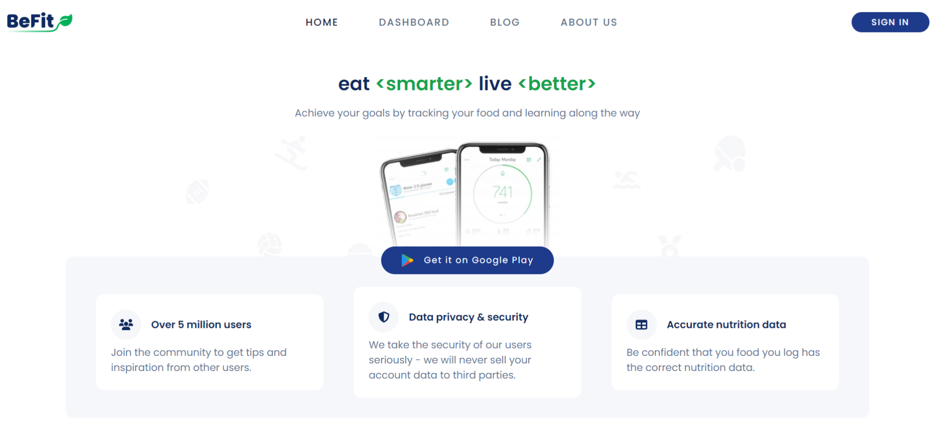
● First Contentful Paint 0.2 s	● Time to Interactive 0.3 s
● Speed Index 0.7 s	● Total Blocking Time 30 ms
■ Largest Contentful Paint 1.5 s	● Cumulative Layout Shift 0.001

[Figure 3.1.6]

The image above (*see Figure 3.1.6*) shows the performance of our application. This score is a very good one. We can notice that the LCP(Largest Contentful Paint) could be better, this is caused by one of the disadvantages of using a Microservices architecture: having to wait for other services to process a request. In our case, the Firebase authentication API takes a bit of time until it sends back a response, and until we don't get a response we can't show the unauthenticated user their data so we just display a loading screen.
The application is built as a Progressive Web Application. This allows the users to install the application on their smartphones just like an ordinary app. The application will look and feel just like a normal native app, but it really runs in the browser.

4. Application walkthrough

4.1 Landing page


BeFit
HOME
DASHBOARD
BLOG
ABOUT US
SIGN IN
eat <smarter> live <better>
Achieve your goals by tracking your food and learning along the way
741
Get it on Google Play

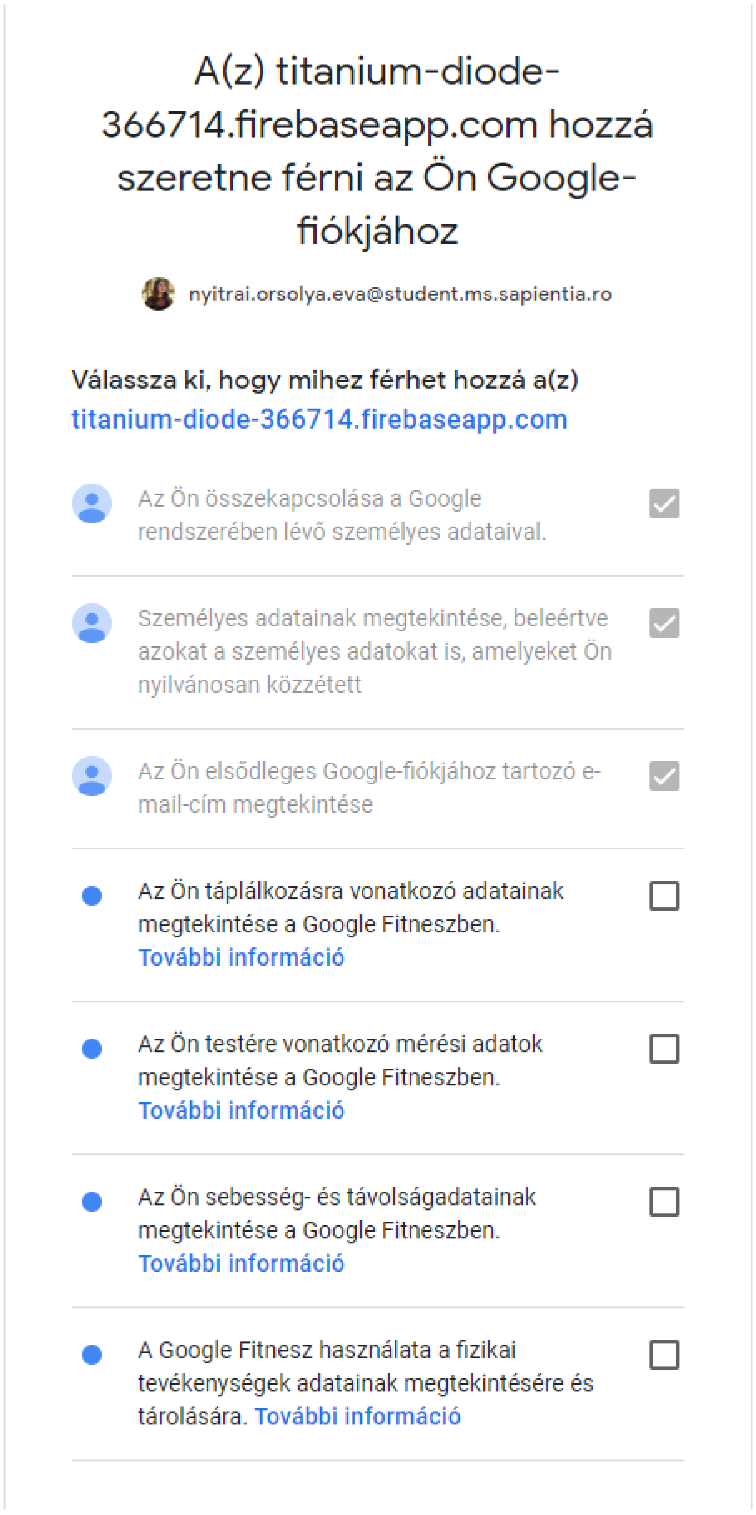
Over 5 million users
Join the community to get tips and inspiration from other users.

Data privacy & security
We take the security of our users seriously - we will never sell your account data to third parties.

Accurate nutrition data
Be confident that you food you log has the correct nutrition data.

[Figure 4.1.1]

4.2 Authentication


A(z) titanium-diode-366714.firebaseapp.com hozzá szeretne férni az Ön Google-fiókjához

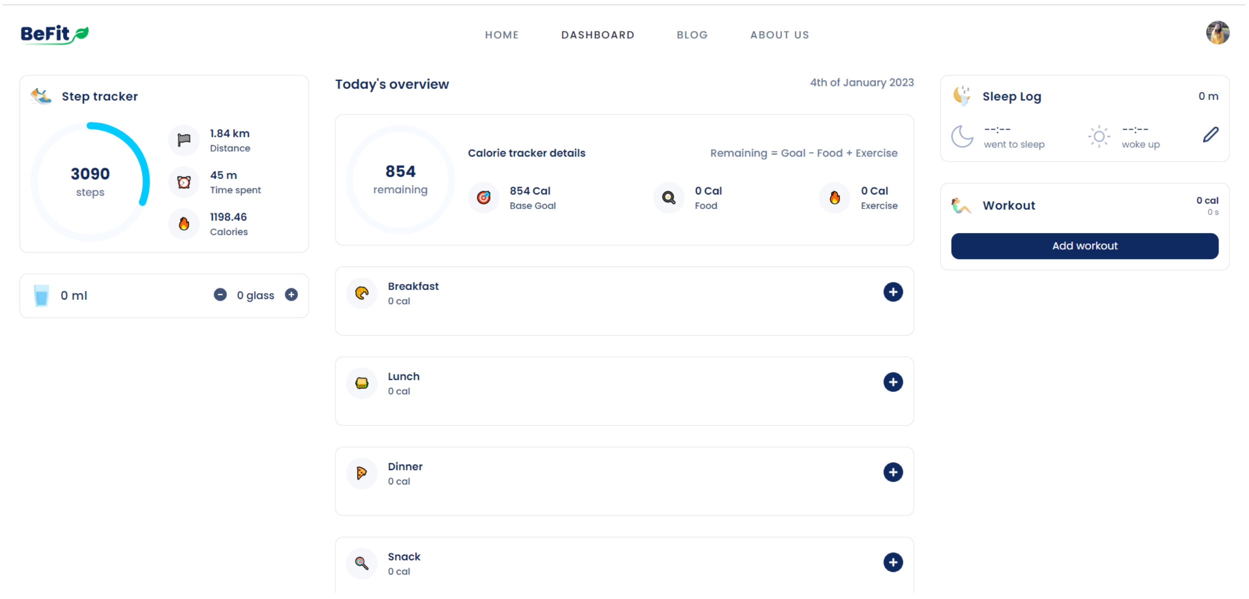
nyitrai.orsolya.eva@student.ms.sapientia.ro

Válassza ki, hogy mihez férhet hozzá a(z) titanium-diode-366714.firebaseapp.com

- Az Ön összekapcsolása a Google rendszerében lévő személyes adataival. ✓
- Személyes adatainak megtekintése, beleértve azokat a személyes adatokat is, amelyeket Ön nyilvánosan közzétett ✓
- Az Ön elsődleges Google-fiókjához tartozó e-mail-cím megtekintése ✓
- Az Ön táplálkozásra vonatkozó adatainak megtekintése a Google Fitneszben. További információ ☐
- Az Ön testére vonatkozó mérési adatok megtekintése a Google Fitneszben. További információ ☐
- Az Ön sebesség- és távolságadatainak megtekintése a Google Fitneszben. További információ ☐
- A Google Fitnesz használata a fizikai tevékenységek adatainak megtekintésére és tárolására. További információ ☐

[Figure 4.2.1]

4.3 Dashboard


BeFit

HOME DASHBOARD BLOG ABOUT US

Step tracker

3090 steps

1.84 km Distance

45 m Time spent

1198.46 Calories

0 ml — 0 glass

Today's overview — 4th of January 2023

854 remaining

Calorie tracker details — Remaining = Goal - Food + Exercise

854 Cal Base Goal

0 Cal Food

0 Cal Exercise

Breakfast 0 cal

Lunch 0 cal

Dinner 0 cal

Snack 0 cal

Sleep Log 0 m

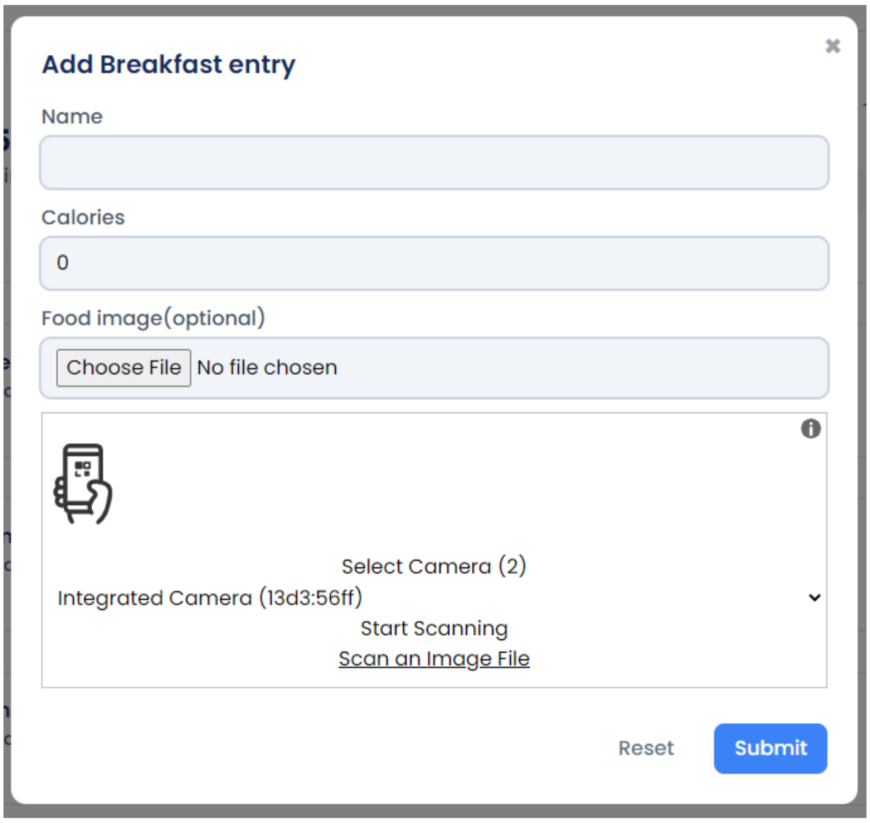
--:-- went to sleep

--:-- woke up

Workout 0 cal 0 s

Add workout

[Figure 4.3.1]


Add Breakfast entry

Name

Calories

0

Food image(optional)

Choose File No file chosen

Select Camera (2)

Integrated Camera (13d3:56ff)

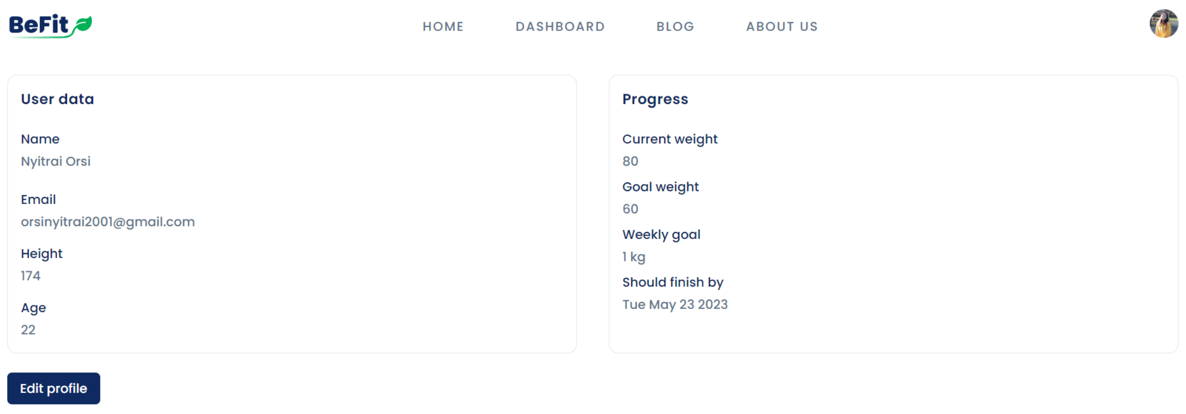
Start Scanning

Scan an Image File

Reset Submit

[Figure 4.3.2]

4.4 User profile



User data

Name
Nyitrai Orsi

Email
orsinyitrai2001@gmail.com

Height
174

Age
22

Progress

Current weight
80

Goal weight
60

Weekly goal
1 kg

Should finish by
Tue May 23 2023

Edit profile

[Figure 4.4.1]

5. Further development possibilities

1. Training an AI model to predict the possible calories based on food images
2. Implementing our own activity tracking system so we do not have to use Google's Fitness API for this
3. Adding a social component, such as a community forum/blog to share progress and achievements with friends
4. Gamify the application to improve user performance
5. Offering personalized meal plans or nutrition coaching
6. Training an AI model that learns from eating habits and creates a meal plan based on that data
7. Adding more in-depth nutritional tracking