Stredná priemyselná škola elektrotechnická, Brezová 2, 921 77 Piešťany

Inteligentná aplikácia na sledovanie kalórií

Ročníkový projekt

Autor: Filip Novák

Ročník štúdia: 4.

Piešťany, 2026

Stredná priemyselná škola elektrotechnická, Brezová 2, 921 77 Piešťany

Inteligentná aplikácia na sledovanie kalórií

Ročníkový projekt

Autor: Filip Novák

Ročník štúdia: 4.

Školiteľ: Ing. Michaela Matelová

Piešťany, 2026

Zriaďovateľ: Trnavský samosprávny kraj

Obsah

Úvod

Tento ročníkový projekt je zameraný na vývoj inteligentnej mobilnej aplikácie na sledovanie kalórií, vytváranie receptov a správu surovín. Cieľom našej aplikácie je podporiť používateľov pri vedení zdravého životného štýlu a efektívnom využívaní potravín, ktoré majú k dispozícii. Myšlienka projektu vznikla z potreby spojiť viaceré funkcionality, ktoré iné aplikácie ponúkajú iba samostatne. Väčšina existujúcich riešení umožňuje iba základné sledovanie kalórií, no neposkytuje správu potravín, upozorňovanie na trvanlivosť ani generovanie receptov na základe dostupných surovín. Našou snahou je vytvoriť komplexné riešenie, ktoré všetky tieto možnosti integruje do jedného prehľadného systému.

Aplikácia funguje na princípe prepojenia používateľského rozhrania, databázového systému a API. Po registrácii a prihlásení si používateľ nastaví svoje ciele, napríklad redukciu hmotnosti, priberanie alebo jej udržiavanie. Na základe zvoleného cieľa sa automaticky vypočíta odporúčaný denný príjem kalórií a živín. Potraviny je následne možné pridávať manuálne alebo prostredníctvom EAN kódov, ktoré aplikácia rozpoznáva s pomocou externej databázy. Každá potravina obsahuje nutričné hodnoty, ktoré sa automaticky započítavajú do denného súčtu, čo používateľovi poskytuje prehľad o jeho napredovaní.

Dôležitou súčasťou projektu je generovanie receptov prostredníctvom API napojeného na model GPT. Aplikácia na server odošle informácie, ako sú používateľské ciele, dostupné suroviny či maximálny počet kalórií receptu. Model GPT tieto dáta spracuje a vytvorí recept, ktorý spĺňa definované požiadavky. Týmto spôsobom aplikácia šetrí používateľom čas pri rozhodovaní, čo pripraviť na jedlo, a zároveň pomáha efektívne využívať potraviny.

Súčasťou aplikácie je aj modul špajze, v ktorom môže používateľ evidovať svoje potraviny - pridávať ich, sledovať množstvo, kontrolovať dátumy spotreby a prijímať upozornenia pri blížiacej sa exspirácii. Pri použití potravín v receptoch sa ich množstvo automaticky aktualizuje, čím má používateľ nepretržitý prehľad o svojich zásobách. Tento mechanizmus prispieva k znižovaniu plytvania potravinami.

Notifikácie slúžia ako pripomienky na pitný režim, zaznamenanie jedla alebo kontrolu stavu špajze. Chceme tým podporiť pravidelnosť používania aplikácie a pomôcť používateľovi dodržiavať jeho stanovené ciele. Cieľom projektu je vytvoriť modernú, praktickú a intuitívnu aplikáciu, ktorá podporí používateľov pri plánovaní jedál, správe potravín a znižovaní zbytočného plytvania. Veríme, že vznikne riešenie, ktoré bude funkčné, spoľahlivé a prínosné pre širokú skupinu požívateľov.

1. Problematika a prehľad literatúry

V tejto časti sme sa venovali problematike návrhu a realizácie mobilnej aplikácie, ktorá pracuje s používateľskými údajmi, komunikuje so serverom cez internet a ukladá časť údajov lokálne do zariadenia. Z praktického hľadiska sme sa zaoberali hlavne architektúrou aplikácie a rozdelením na obrazovky/taby, spôsobom komunikácie s backendom, a spracovaním všetkých potrebných používateľských dát. Tieto oblasti sme vybrali preto, lebo ich považujeme za najdôležitejšie pre funkčnosť aplikácie.

1.1. Mobilné aplikácie a ich architektúra

Pri návrhu aplikácie sme vychádzali z toho, že aplikácia typicky pozostáva z časti, s ktorou používateľ interaguje (UI), ďalej zo spracovania vstupov, stavov a dátovej vrstvy. V praxi sme sa snažili o také rozdelenie aplikácie, aby bola čitateľná, škálovateľná a aby sa minimalizovalo opakovanie kódu.

Za dôležité sme považovali aj riešenie navigácie medzi obrazovkami. Bežne sa používa zásobníková navigácia, prípadne kombináciu s tab navigáciou. V našom riešení sme použili stack navigáciu, keďže aplikácia prirodzene prechádza medzi úvodnými obrazovkami, autentifikáciou a následne hlavnou časťou aplikácie.

Významným aspektom aplikácie bola aj práca so stavom používateľa (napr. prihlásený/neprihlásený) a s uchovaním údajov. Počítali sme s tým, že používateľ musí mať možnosť len tak zavrieť aplikáciu, alebo môže byť dočasne bez internetu. Preto sme časť dát ukladali lokálne do zariadenia, aby sa aplikácia vedela spustiť rýchlejšie a aby sa obmedzilo opakované sťahovanie údajov z databázy.

1.2. React Native / Expo

Na tvorbu samotnej aplikácie sme zvolili React Native, pretože umožňuje vytvárať multiplatformové mobilné aplikácie pomocou JavaScriptu a React. Výhodu sme videli v rýchlom a jednoduchom tvorení, v širokej ponuke knižníc a v tom, že rovnaký kód je možné použiť na Android aj iOS bez ďalších úprav.

Ako vývojové prostredie sme použili Expo, vďaka ktorému vieme premietať náš kód naživo buď v mobilnom emulátore alebo na fyzickom zariadení bez žiadneho zdržania. Expo taktiež obsahuje rozsiahlu knižnicu jednoducho implementovateľných modulov. V našom projekte sa to prejavilo najmä pri využití knižníc ako expo-notifications (notifikácie) a expo-linking (deeplinky). Tieto moduly nám umožnili realizovať funkcie, ktoré by inak vyžadovali zložitejšiu konfiguráciu.

Ďalej sme pracovali s viacerými knižnicami pre UI a navigáciu. Základom bola navigácia cez @react-navigation (native stack, bottom tabs a drawer), čo nám umožnilo realizovať viacúrovňové členenie aplikácie. Pre lokálne ukladanie dát sme použili @react-native-async-storage/async-storage, ktoré sme využívali na ukladanie používateľských údajov a dát pre rýchlejší štart aplikácie a rýchlejšie načítanie údajov bez zbytočného sťahovania z databázy.

1.3. Backend pre mobilné aplikácie

Keďže aplikácia potrebovala pracovať s používateľskými dátami a s ich uložením, implementovali sme backend vo forme webovej služby. Zvolili sme platformu Node.js a framework Express, pretože umožňuje rýchlo a jednoducho vytvoriť REST rozhranie, ktoré aplikácia vie jednoducho volať pomocou HTTPS požiadaviek.

V aplikácii sme zrealizovali komunikáciu s backendom cez endpointy, na ktoré sa aplikácia obráti. Na strane používateľa sme použili volania cez fetch s metódami GET a POST. Tento prístup sme zvolili preto, že je priamo podporovaný v prostredí JavaScriptu a zároveň sa jednoducho implementuje.

Backend sme navrhli tak, aby zabezpečoval spracovanie autentifikácie, spracovanie profilu používateľa, prácu s položkami používateľa, a zabezpečenie zresetovania používateľského hesla.

V serverovej časti sme využili aj CORS, keďže aplikácia komunikuje cez sieť a pri vývoji/testovaní môže byť spúšťaná z rôznych prostredí.

1.4. Autentifikácia, databáza, notifikácie

V našej aplikácii sme realizovali autentifikáciu cez backendové API. Aplikácia posiela prihlasovacie údaje na server (POST /api/login). Po úspešnom prihlásení sme v aplikácii uložili základné údaje do lokálneho úložiska, aby sme umožnili automatické

prihlásenie pri ďalšom spustení aplikácie. Toto riešenie sme použili aby používateľ nemusel písať prihlasovacie údaje pri každom spustení aplikácie.

Pri registrácii sme implementovali aj povinný súhlas (GDPR) formou prepínača. Súhlas posielame na server spolu s časom udelenia súhlasu (gdprConsentAt). Rozhodli sme sa tak preto, aby bolo spracovanie osobných údajov transparentné a dohľadateľné.

1.4.1. Obnova hesla a deeplinky

Pre zabudnuté heslo sme použili spôsob, v ktorom používateľ zadal e-mail, kontaktuje sa endpoint (POST /api/forgot-password), pošle sa e-mail do schránky používateľa a následne aplikácia umožnila nastaviť nové heslo cez token získaný z odkazu v e-maile. Token sme získavali z deep linku pomocou knižnice expo-linking a následne sme ho posielali na server (POST /api/reset-password). Tento spôsob sme použili preto, že je bezpečnejší než resetovanie hesla priamo bez tokenu.

1.4.2. Databáza

Pre našu aplikáciu sme zvolili databázu MongoDB. Tento typ databázy sme zvolili preto, že sa dobre hodí na ukladanie dokumentovo orientovaných údajov a umožňuje jednoducho škálovať dátový model bez zbytočných komplikácií

1.4.3. Notifikácie

V aplikácii sme implementovali pravidelné notifikácie pomocou expo-notifications. Implementovali sme vyžiadanie povolení od používateľa, uloženie časov notifikácií do AsyncStorage, zrušenie a aktualizáciu existujúcich notifikácií, aby sa neduplikovali.

Notifikácie sme navrhli ako opakovateľné „daily“ trigger udalosti, čo sme zvolili preto, aby používateľ dostával pripomienky pravidelne.

2. Ciele práce

V práci sme si stanovili cieľ navrhnúť a implementovať kľúčové časti aplikácie, ktoré používateľovi zabezpečili plynulý vstup do aplikácie, prácu s obsahom, a mnoho dôležitých funkcií pre každodenné používanie aplikácie.

Za hlavný cieľ sme považovali nasledovné funkčné celky:

- úvodnú obrazovku s uložením stavu prvého spustenia

- obnovu hesla cez resetovací link v e-maile pomocou tokenu a resetu hesla cez deep link
- hlavnú obazovku aplikácie, Dashboard, so synchronýzaciou denných údajov
- modul receptov vrátane generovania receptov na servi, ukladania a možnosti započítať recept do dennej spotreby
- modul notifikácií s plánovaním denných pripomienok a uložením časov do lokálneho úložiska
- kamerový skener produktov, ktorý vie načítať produkt podľa EAN kódu, spracovať nutričné hodnoty a uložiť do špajze alebo priamo započítať do dennej spotreby

3. Materiál a metodika

3.1. Použité nástroje a technológie

Pri vývoji aplikácie sme použili:

- React Native a Expo ako základ vývoja mobilnej aplikácie
- React Navigation pre navigáciu medzi obrazovkami,.
- AsyncStorage pre lokálne ukladanie používateľských údajov.
- expo-linking pre spracovanie deep linkov.
- expo-notifications pre pravidelné notifikácie.

Na strane servera sme použili:

- Node.js ako runtime,
- Express ako webový framework pre REST API,
- mongodb pre komunikáciu s databázou MongoDB,
- cors pre nastavenie prístupov z klienta,
- bcryptjs pre bezpečnejšiu prácu s heslami (hashovanie).

3.2. Postup implementácie

Pri tvorení tohto projektu sme postupovali takým spôsobom, aby sme ako prvé vytvorili používateľsky zrozumiteľný a príjemný vstup do aplikácie so stručným opisom aplikácie. Následne sme doplnili funkcionality, ktoré používateľ bežne očakával po prihlásení a pri práci s účtom.

Postup práce sme zvolili v tomto poradí:

1. **WelcomeScreen** - najprv sme vyriešili to čo používateľ bude vidieť keď po prvý krát otvorí našu aplikáciu. Rozdelenie medzi registráciou, prihlásením a zabezpečenie aby sa úvodná obrazovka zobrazila len pri prvom otvorení aplikácie.
2. **Zabudnuté heslo a reset hesla** - implementovali sme kompletný postup na zresetovanie používateľského hesla. Od požiadania o reset cez prijatie resetovacieho linku až po nastavenie nového hesla cez token a deep link.
3. **Dashboard** - spravili sme hlavnú obrazovku aplikácie, v ktorej sa bude používateľ nachádzať a pracovať s ňou. Zhotovili sme všetku funkcionalitu ako napríklad načítanie lokálnych dát, synchronizácia so serverom a aktualizácie.
4. **Recepty** - doplnili sme obrazovku na generovanie receptov pomocou umelej inteligencie cez server. Ukladanie a mazanie receptov vrátane funkcie zjedenia receptu a započítania nutričných hodnôt do denného prehľadu používateľa.
5. **Notifikácie** – pripravili sme systém na doručenie pravidelných notifikácií, ktorý sa spustí po udelení povolenia používateľom pri prvotnom prihlásení do aplikácie. Taktiež podporuje upozornenie na koniec expiračného dátumu potravín uložených v špajzi.

Tento postup vypracovania sme zvolili preto, aby sme ako prvé zabezpečili plynulý tok aplikácie a jednoduchú navigáciu v aplikácii a až potom pridávali pokročilejšie funkcie, ktoré vyžadovali viac stavov, serverové endpointy a prácu s lokálnym úložiskom.

3.3. Návrh riešenia pre privítanie, reset hesla, Dashboard, recepty a notifikácie

Tieto časti sme navrhli ako kombináciu klienta resp. React Native a Expo prostredia a servera cez Node.js a Express, pričom sme sa snažili dodržať princíp, že:

- klient mal riešiť UI, lokálne stavy a prácu s AsyncStorage,
- server mal riešiť komunikáciu s databázou, komunikáciu s API, bezpečnostné citlivé operácie ako je prihlasovanie a generovanie obsahu pomocou AI agenta.

Tento návrh sme použili preto, aby boli citlivé operácie (reset hesla, hashovanie, tokeny) mimo mobilnej aplikácie a aby sa dali logicky kontrolovať a spravovať na serveri.

3.4. Privítanie používateľa (WelcomeScreen.js)

Privítanie sme implementovali ako jednoduchú úvodnú obrazovku, ktorá sa mala zobraziť iba pri prvotnom spustení aplikácie aby sa používateľ dozvedel základné informácie o našej aplikácii.

Túto funkciu sme zabezpečili takým spôsobom, že sme pri otvorení tejto obrazovky skontrolovali v lokálnom úložisku **AsyncStorage** stav **onboardingSeen**.

- Ak bola uložená hodnota nastavená na **true**, používateľa sme okamžite presmerovali priamo na obrazovku HomeScreen pomocou funkcie **navigation.replace("HomeScreen")**. Týmto sme zabezpečili, že sa úvodná obrazovka nezobrazí.
- Ak privítanie používateľa ešte nebolo dokončené, tak po stlačení jedného z tlačidiel sme uložili stav **onboardingSeen** do **AsyncStorage** a nastavili hodnotu na **true**. Ďalej sme presmerovali používateľa na ďalšiu obrazovku podľa toho aké tlačidlo si vybral.

Tento mechanizmus sme zvolili preto, aby privítanie neprekážalo používateľovi pri každom spustení aplikácie, ale zároveň aby nový používateľ hneď pochopil zmysel aplikácie.

3.5. Zabudnuté heslo a reset hesla

Táto časť zahŕňa implementovanie celej funkcionality zresetovanie používateľského hesla v prípade, že ho zabudol.

3.5.1. Požiadanie o reset hesla (ForgetPass.js + /api/forgot-password)

Na obrazovke HomeScreen sme vytvorili tlačidlo „Zabudnuté heslo?“, ktoré po kliknutí presmeruje používateľa na obrazovku **ForgetPass**. Na tejto obrazovke sa nachádza pole, do ktorého používateľ zadá svoj e-mail. Po potvrdení sme poslali požiadavku na resetovanie na serverový endpoint.

Doručenie používateľského e-mailu sme zabezpečili pomocou funkcie **fetch()** spolu s metódou **POST** a nasledovne kontaktovali serverový endpoint v tvare **/api/forgot-password** s telom **{email}**.

Taktiež sme pridali funkciu na odchytenie potenciálne vzniknutých chýb pri kontaktovaní servera. Pokiaľ komunikácia prebehne úspešne, používateľ je upozornení

pomocou funkcie **Alert.alert()** aby si skontroloval svoju e-mailovú schránku pre link na zresetovanie hesla.

Na serveri sme tento endpoint implementovali spôsobom aby:

1. server overil, či používateľ zadal svoj e-mail do poľa. Pokiaľ áno, server sa pokúsi vyhľadať používateľa v databáze pomocou funkcie **users.findOne({email})**. Pokiaľ používateľ nezadalplatný e-mail, je upozornení príslušným alertom.
2. server zakaždým vygeneroval dostatočne bezpečný token pomocou funkcie **crypto.randomBytes(32).toString("hex")**, aby token nemohol byť uhádnutý. Nasledovne je potrebné aby nastavil expiráciu tokenu na 15 minút. Vytvorí ju pomocou aktuálneho dátumu a času funkciou **Date.now() + 15 * 60 * 1000**.
3. server uložil vytvorený token a dátum exspirácie k používateľovi do databázy pomocou funkcie **users.updateOne()**.
4. server nastavil e-mailového klienta pomocou knižnice **Nodemailer** a funkcie **nodemailer.createTransport()**.

```
const transporter = nodemailer.createTransport({ // SMTP klient
  host: "smtp.gmail.com",
  port: 465, // SSL port
  secure: true, // šifrované pripojenie
  auth: {
    user: process.env.EMAIL_USER, // email účet
    pass: process.env.EMAIL_PASS, // heslo / app password
  },
});
```

Obrázok 1 Ukážka tela transportéra

Nasledne aby vytvoril resetovací link a odoslal link používateľovi pomocou funkcie **.sendMail()**.

```
    await transporter.sendMail({
from: `"Socka" <${process.env.EMAIL_USER}>`,
to: email,
subject: "Password Reset Request",
html: `
  <p>Požiadali ste o reset hesla.</p>
  <p>Kliknite na odkaz nižšie (platí 15 minút):</p>
  <a href="${resetLink}" style="color: blue; text-decoration: underline;">
    Resetovať heslo
  </a>
`,
);
```

Obrázok 2 Ukážka tela e-mailu

Tento prístup sme zvolili preto, aby sa heslo nedalo resetovať bez overenia cez e-mail a aby bol token časovo obmedzený, čím sme znížili riziko zneužitia.

3.5.2. Tvorba linku na reset hesla

Samotný link na zresetovanie hesla je zložený z viacerých dôležitých častí a je v tvare:

https://app.bitewise.it.com/reset-password?token=${token}.

- **app.bitewise.it.com** je doména, cez ktorú je možné kontaktovať náš server
- **/reset-password** je samotný endpoint kde sa overuje token a spracúva nastavenie nového hesla
- **?token=${token}** je časť kde sa pošle vygenerovaný token

Takýto link je nazývaný tzv. **deep link** alebo **app link.** Aby sme zabezpečili funkcionalitu presmerovania používateľa na aplikáciu po kliknutí linku bolo dôležité dodržať pár nevyhnutných požiadaviek:

- vo všetkých systémových súboroch sme museli zabezpečiť aby sa zhodoval **packageName** aplikácie.
- do súboru **AndroidManifest.xml** bolo potrebné pridať **intent-filter**, ktorý zabezpečuje aby aplikácia automaticky overila platnosť linku a aby systém vedel, že tento link má otvoriť našu aplikáciu.

```
<intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="https" android:host="app.bitewise.it.com"
    android:pathPrefix="/reset-password"/>
</intent-filter>
```

Obrázok 3 Ukážka intent-filtra

- nasadiť na server hostovanie priečinku **.well-known** so súborom **assetlinks.json**, pomocou ktorého systém skontroluje platnosť deep linku/app linku. Aby overenie prebehlo úspešne súbor musel obsahovať správny **package_name** a platný **sha256_cert_fingerprints**, ktorý je spojený s našou aplikáciou. Tento odtlačok je jedinečný a poskytovaný od spoločnosti **Cloudflare**.

3.5.3. Reset hesla cez token (ResetPass.js + /api/reset-password)

Po tom ako používateľ obdržal svoj resetovací link do svojej e-mailovej schránky a klikol naň sme na strane klienta sme potrebovali zabezpečiť, aby aplikácia vedela tento token získať z odkazu v e-maile. Ako riešenie sme preto implementovali nasledovný postup pomocou knižnice **expo-linking**, ktorá nám uľahčila celý tento proces:

- získanie linku na zresetovanie hesla sme zabezpečili pomocou funkcie **useEffect()**, ktorá obsahuje 2 časti. V prípade, že aplikáciu používateľ zatvoril alebo z iného dôvodu nezostala otvorená na pozadí pred kliknutím na link v e-maile, snažili sme sa token získať so samotného **deep linku** pomocou funkcie:

Linking.getInitialURL().then((url) => applyToken(getTokenFromUrl(url))).

Funguje to na princípe, že **Linking.getInitialURL()** získa URL z deep linku, ktorým bola aplikácia otvorená, **then((url)=>...)** vezme túto URL a vyparsuje z nej token cez funkciu **getTokenFromUrl(url)** a uloží ho do stavu cez **applyToken(...)**.

- v prípade, že aplikácia zostala bežať na pozadí v čase kliknutia na link sme zabezpečili aby aplikácia počúvala na nové **deep linky** pomocou funkcie **Linking.addEventListener("url", ...))**. Hneď ako používateľ úspešne zresetoval heslo a opustil obrazovku sme **listener** upratali cez funkciu **.remove()** aby nevznikali zbytočné úniky pamäte.

Po zadaní nového hesla sme posielali heslo pomocou funkcie **fetch()** na serverový endpoint v tvare **/api/reset-password** metódou **POST** a s telom **{ token, newPassword }**.

Na serveri sme endpoint implementovali takým spôsobom, aby:

1. overil, či token existoval a bol platný alebo nebol exspirovaný,
2. našiel používateľa v databáze podľa tokenu pomocou funkcie **.findOne()** s telom tokenu a času exspirácie tokenu.
3. nové heslo zahashoval pomocou funkcie **bcrypt.hash(newPassword, 10)** aby sme zachovali bezpečnosť údajov.
4. uložil nové heslo do databázy a zároveň odstránil token a čas expirácie tokenu pomocou funkcie **.updateOne()**.

Tento postup sme zvolili z dôvodu, aby sa heslo nikdy neukladalo v čistej čitateľnej forme a aby sa reset token po použití nedal znova použiť.

3.6.Dashboard (Dashboard.js)

Dashboard sme implementovali ako centrálnu obrazovku našej aplikácie tak, aby po prihlásení bol používateľ presmerovaný na túto obrazovku. Je to prostredie, ktoré spája všetky vedľajšie obrazovky do jedného centrálneho prostredia, v ktorom sa používateľ bude pohybovať po celú dobu používania aplikácie.

Rozhodli sme sa tak preto, aby sme zbytočne neustále neprepínali medzi viacerými súbormi a nezaťažovali systém, ale aby všetko bolo pekne pokope na jednom mieste.

Obrazovka Dashboard je rozdelená do nasledujúcich častí:

3.6.1. Horná lišta (TopBar)

Nachádza sa v hornej časti obrazovky, vytvorili sme ju tak, aby zobrazovala logo našej aplikácie a pozdrav pre používateľa. Pozdrav obsahuje **nick** používateľa, ktorý zadal pri registrácii do aplikácie.

Nick sme získali z lokálneho úložiska **AsyncStorage** pomocou funkcie **AsyncStorage.getItem(„userNick“)** a následne zobrazili v pozdrave. Tento proces sa vykonal automaticky pri zobrazení obrazovky pomocou funkcie **useEffect()**.

3.6.2. Hlavný obsah

Nachádza sa priamo v strede obrazovky a slúži na zobrazenie obsahu jednotlivých **tabov**/obrazoviek. Túto funkcionalitu sme dosiahli pomocou funkcie **renderContent** a stavu **activeTab**.

Stav **activeTab** služi na sledovanie aktuálne zakliknutého **tabu**. Implementovali sme to spôsobom, že sme každému tabu priradili číslo, ktoré ho identifikuje. Predvolená hodnota **activeTab** je **1**, čo znamená, že po prihlásení sa používateľovi ako prvé zobrazí prehľad (OverviewTab).

Samotné prepínanie tabov sme vyriešili funkciou **renderContent**, ktorá používa funkciu **switch**, vďaka ktorej vieme porovnávať hodnotu **activeTab** a následovne zobrazovať príslušný obsah tabu. Pokiaľ by došlo ku chybe pri prepínaní tabu, zobrazí sa text „Oops, niečo sa pokazilo“.

3.6.3. Spodná / Navigačná lišta (NavBar)

Navigačnú lištu sme vytvorili spôsobom, že sme si ju rozdelili na 5 častí. 4 taby medzi, ktorými vie používateľ prepínať a samostatné tlačidlo na prepnutie do **CameraScreen**.

Skladá sa z hlavnej časti ktorá slúži ako **flex kontainer** a vnútorných časti **flex items**. Všetky časti majú v sebe obrázok a príslušný názov aby používateľ vedel o aký tab sa jedná. Taktiež sme pridali ladný efekt aby používateľ vedel rozoznať, ktorý tab je aktuálne zakliknutý. Dosiahli sme toho takým spôsobom, že kontrolujeme stav **isActive**. Pokiaľ je hodnota **isActive** zhodná s číslom tabu, tak sa použije efekt zakliknutia.

Po kliknutí na jednotlivé taby sa nastaví hodnota **activeTab** na číslo zakliknutého tabu a v strednej časti obrazovky sa zobrazí príslušný obsah daného tabu.

3.7. Recepty (RecipesTab.js + serverové endpointy)

Recepty sme implementovali ako kombináciu statických / overených receptov, ktoré sú natvrdo vpísané v kóde aplikácie a sú to recepty, ktoré sme my osobne overili. A časť generovaných / uložených receptov, ktoré boli vytvorené cez OpenAI agenta a uložené používateľom.

Samotnú obrazovku sme teda rozdelili na časť overených receptov a časť uložených receptov. Pomocou vlastnosti **grid** sme vytvorili ako pre statické tak aj pre generované recepty bunky s názvom receptu a príslušným obrázkom receptu.

Bunky sme implementovali spôsobom, že používateľ si ich môže rozkliknúť aby sa dozvedel všetky potrebné informácie o recepte. Informácie sa zobrazia v podobe vyskakovacieho okna, ktoré sme vytvorili pomocou elementu **<Modal/>**.

Toto okno sme vytvorili z viacerých častí:

- úplne na začiatok sme umiestnili obrázok receptu, pričom pri statických používame overené obrázky pre dané recepty, no pri vygenerovaných receptov zobrazujeme **placeholder**, ktorý určujeme podľa toho akú kategóriu má recept pridelenú.
- ďalej sme vytvorili 2 menšie okienka umiestnené vedľa seba. Jedno okienko zobrazuje pridelenú kategóriu receptu a druhé okienko zobrazuje čas, ako dlho trvá pripraviť daný recept.
- nasledovne sme vytvorili priestor pre tabuľku na zobrazenie všetkých dôležitých nutričných hodnôt receptu. Každú položku v tabuľke sme vytvorili ako objekt s hodnotami **label, value** a **unit**. Každú položku zobrazujeme pomocou funkcie **.map(item, idx)...**.
- potom sme prirobili okno na zobrazenie všetkých ingrediencií potrebných na zhotovenie receptu. Ingrediencie zobrazujeme cez rovnakú funkciu ako hodnoty v tabuľke nutričných hodnôt. Kedže množstvo každej ingrediencie zobrazujeme v gramoch, pridali sme k názvu „Ingrediencie“ informačné tlačidlo v podobe písmena „i“ v zelenom krúžku. Po kliknutí sa zobrazí prirovnanie lyžičiek a pohára ku gramom (1 polievková lyžica = cca 15g), aby používateľ vedel, koľko približne má danej suroviny použiť a nemusel všetko vážiť na váhe. Prirobili sme aj tlačidlo na zavretie tohto okna.
- v poslednej časti sme prirobili okno na zobrazenie postupu na zhotovenie receptu. Na zobrazenie jednotlivých krokov sme opäť použili rovnakú funkciu ako v predošlých prípadoch.

Nakoniec sme implementovali v spodnej časti tlačidlo na zatvorenie okna, tlačidlo na zjedenie receptu a pokiaľ sa jedná o vygenerovaný recept aj možnosť na zmazanie receptu zo zoznamu uložených receptov.

- o tom či je okno viditeľné rozhoduje stav **showUnitInfo**. Po rozkliknutí receptu sa hodnota nastaví na **true** a zobrazia sa informácie o recepte. Tlačidlo „Zavrieť“ funguje na princípe, že nastaví hodnotu na **false** a okno sa zavrie.
- ďalej sme implementovali tlačidlo na zmazanie receptu pokiaľ sa jedná o vygenerovaný recept. Túto funkcionalitu sme zhotovil spôsobom, že keď používateľ klikne na tlačidlo zavoláme funkciu **deleteRecipe**, ktorá pomocou **fetch()** s metódou **DELETE** a s telom e-mailu a ID receptu kontaktuje serverový endpoint **/api/deleteRecipe**. Server vyhľadá používateľa a pomocou funkcie **updateOne** vymaže recept z poľa uložených receptov v databáze. Keď aplikácie dostane odpoveď od serveru tak aktualizuje zoznam aktuálne uložených receptov pomocou funkcie **.filter()**, ktorá porovná ID všetkých uložených receptov s ID aktuálne rozkliknutého receptu a vymaže ho. Tým zabezpečí aby sa vymazaný recept naďalej nezobrazoval. Potom aktualizuje lokálne úložisko pomocou **AsyncStorage.setItem()**.
- ako posledné sme vytvorili tlačidlo na zjedenie receptu. Tento princíp sme zabezpečili pomocou funkcie **consumeRecipe**, ktorá pomocou funkcie **fetch()** s metódou **POST** a s telom e-mailu, dnešného dátumu a nutričných hodnôt volá serverový endpoint **/api/consumeRecipe**. Server pomocou e-mailu vyhľadá používateľa, a podľa aktuálneho dátumu z databázy vytiahne jeho aktuálne nutričné hodnoty a uloží do konštanty **currentTotals**. Nasledovne vytvorí konštantu **updatedTotals**, v ktorej spočíta **currentTotals** s nutričnými hodnotami, ktoré dostal od aplikácie cez **fetch()**. Potom pomocou funkcie **updateOne()** uloží nové hodnoty do databázy.

3.7.1. Personalizácia receptu podľa požiadavkou

V hornej časti tabu receptov sme vytvorili tlačidlo na generovanie receptov pomocou AI agenta. Pripravili sme UI pre výber preferencií zapnutie voľby fitness cieľa a výber produktov zo špajze. Celkovú logiku generovania receptu sme rozdelili na viacero častí:

- najdôležitejšia časť generovania receptu je tá, kde si používateľ vyberá preferencie, podľa ktorých má AI vygenerovať recept. Používateľ ma dostupné veľké množstvo preferencií ako sú sladké, slané no aj komplexnejšie rozdelenie podľa druhu jedla, alergií a typu jedla podľa krajiny. Pokiaľ by nastala situácia, že používateľ nechápe čo znamená určitá preferencia, pridali sme informačné tlačidlo hneď vedľa názvu „Preferencie“, v ktorom sa nachádza vysvetlenie každej jednej preferencie. Princíp na vyberane preferencií sme vytvorili spôsobom, že každá zobrazená preferencia funguje ako tlačidlo. Po stlačení sa ID stlačenej preferencie presunie do poľa **selectedPreferences** a tlačidlo sa vizuálne presunie do okienka nad preferenciami aby používateľ mal istotu, že preferenciu vybral. Po presunutí pribudne tlačidlo na odstránenie preferencie pokiaľ by používateľ zmenil názor. Aby sa vybraté preferencie zobrazovali správne používame funkciu **.map()** pre každú položku v **selecedPreferences** a pomocou funkcie **.filter()** sme zabezpečili aby sme vymazali správnu preferenciu. Pokiaľ by chcel používateľ vidieť viac preferencií, pridali sme tlačidlo na zobrazenie zvyšných preferencií.
- ako ďalšiu funkciu sme prirobili možnosť generovať recept podľa nastaveného fitness cieľa používateľa. Túto funkcionalitu sme implementovali spôsobom, že sme do UI pridali pomocou elementu **<Switch>** páčku na zakliknutie. Keď používateľ páčku zaklikne, spustí sa funkcia **setUseFitnessGoal**, ktorá nastaví hodnotu **useFitnessGoal** na true.
- taktiež sme pridali možnosť používateľovi generovať recepty priamo podľa potravín, ktoré ma uložené v špajzi. Zabezpečili sme tak, že sme opäť pridali do UI páčku na prepínanie stavov. Pokiaľ používateľ vyberie túto možnosť, hodnota **usePantryItems** sa zmení na **true**. Keď je hodnota true, aplikácia kontaktuje server pomocou **fetch()** aby potiahla produkty z databázy a uloží ich poľa **pantryItems**. Nasledovne pomocou funkcie **.map()** mapujeme pole **pantryItems** a zobrazíme všetky produkty so špajze nižšie, kde dávame používateľovi vybrať jednotlivé položky, ktoré chce použiť alebo možnosť použiť všetky položky naraz. Keď používateľ vyberie položku meno položky sa uloží do poľa **selectedPantryItems**.
- ako poslednú funkciu sme pridali možnosť vybrať maximálny čas varenia pre vygenerovaný recept. Túto funkciu sme zhotovili pomocou elementu **<Slider>**, ktorému sme nastavili minimálnu hodnotu 15 minút, maximálnu 180 minút

a krokovanie po 5 minútach. Pri posúvaní slidera sa mení hodnota a tá sa automaticky ukladá do hodnoty **maxCookingTime**.

Na koniec sme prirobili tlačidlo na zresetovanie všetkých vybratých preferencií, vrátane generovania podľa fitnes cieľa a použitia produktov so špajze. Tlačidlo zavolá funkciu **resetState()**, ktorá vráti hodnoty všetkých stavov a polí na predvolenú hodnotu.

3.7.2. Generovanie receptu, prompt a server

Ako posledné sme vytvorili tlačidlo na zatvorenie okna na generovanie receptov a tlačidlo na vygenerovanie receptu. Po stlačení tlačidla sa zatvorí okno generovania pomocou **setGenerateModalVisivle(false)**, spustí sa funkcia **generateRecipe()** a resetujú sa stavy pomocou funkcie **resetState()**.

Funkcia **generateRecipe()** zapne stav načítania **setGenerating(true)** vďaka čomu sa zobrazí okno s textom „Vytváram recept“ a s animáciou načítavania. Ďalej sa zostaví text preferencií a text potravín so špajze , spolu s ktorými sa následne zostaví **userPrompt**, ktorý je tvorený vybranými preferenciami, pokynom na fitnes cieľ, limitom času varenia a potravín, ktoré sa majú použiť pri generovaní receptu.

Hodnotu **userPrompt** nasledovne posielam na serverový endpoint **/api/generateRecipe** pomocou funkcie **fetch()** spolu s ďalšími údajmi. Na servery sa **userPromtp** spojí spolu so systémovým promptom, ktorý je naprogramovaný priamo v serveri a udáva prísne pravidlá, podľa ktorých musí vygenerovať recept.

Pokiaľ nastanú problémy pri generovaní receptu, určili sme sa AI maximálny počet pokusov na 3. Pokiaľ 3 krát pokazí generovanie receptu proces sa preruší a používateľ je upozornení prostredníctvom vyskakovacieho okna.

Na generovanie receptov sme zvolili model **gpt-4o**, pretože je rýchly, spoľahlivý a na aplikáciu našej veľkosti postačujúci. Taktiež sme nastavili maximálnu dĺžku odpovede od agenta na **700 tokenov**, pričom 1 token sa rovná približne 7 znakom abecedy, číslic a špeciálnych znakov. Taktiež bolo potrebné nastaviť hodnotu **temperature**. Rozhodli sme sa hodnotu nastaviť na 0.9, čo dovoľuje agentovi byť viac kreatívny so svojimi odpoveďami ale zároveň dodržiava stanovené pravidlá v prompte. Samotného agenta voláme pomocou funkcie **openai.chat.completions.create()** kde vkladáme názov modelu aký používame, správu pre agenta, **max_tokens** a **temperature**.

Prompt sme nastavili tak, aby nám recepty generoval vo formáte JSON objektu, aby sme vedeli s vygenerovanými dátami jednoducho pracovať. Po obdržaní odpovede skontrolujeme či je JSON formát, ktorý agent vrátil platný pomocou viacerých pomocných funkcií a odpoveď sa následne server pokúsi premeniť na JSON objekt pomocou **JSON.parse**.

Pokiaľ všetky kroky prebehli úspešne, objekt pošleme naspäť na stranu klienta a zobrazíme všetky dáta rovnakým spôsobom ako pri statických a uložených receptoch.

S výnimkou pridaného tlačidla na uloženie receptu, kde sa používateľ rozhodne či chce okno len zavrieť alebo sa mu recept páči a chce si ho pridať medzi uložené recepty.

Pokiaľ sa používateľ rozhodne recept uložiť, pomocou funkcie **fetch()** s metódou **POST** kontaktujeme serverový enpoint **/api/addRecipe**. Server vyhľadá správne priradí hodnoty z vygenerovaného receptu hodnotám, ktoré budeme posielať do databázy a pomocou funkcie **.updateOne()** pošle nový recept. Po úspešnom uložení sa refreshne tab receptov a recept sa bude zobrazovať medzi uloženými receptami.

4. 3.8 Notifikácie (notifications.js)

Notifikácie sme implementovali v samostatnom module notifications.js, kde sme pripravili:

- vyžiadanie povolení (requestPermissions),
- plánovanie denných notifikácií na konkrétne časy (scheduleDailyNotifications),
- zrušenie všetkých naplánovaných notifikácií (cancelAllNotifications),
- načítanie uložených časov (loadNotificationTimes),
- aktualizáciu časov (updateNotificationTimes).

Pri plánovaní sme najprv zrušili staré notifikácie, aby sa neduplikovali, a následne sme naplánovali notifikácie s denným triggerom. Časy sme uložili do AsyncStorage pod kľúčom notificationTimes.

Tento postup sme zvolili preto, aby používateľ nemal viacnásobné notifikácie v rovnakom čase a aby sa nastavenie časov zachovalo aj po reštarte aplikácie.

---

5. 4 Výsledky práce (upravené – iba moje časti)

6. 4.1 WelcomeScreen – onboarding

Ako výsledok sme implementovali onboarding, ktorý sa zobrazil iba pri prvom spustení aplikácie. Stav sme uložili do AsyncStorage a pri ďalšom spustení sme onboarding preskočili. Týmto sme dosiahli, že používateľ nebol opakovane zdržovaný a zároveň sme mu pri prvom kontakte vysvetlili účel aplikácie.

7. 4.2 Zabudnuté heslo a reset hesla

Implementovali sme kompletný reset hesla:

- používateľ odoslal e-mail cez ForgetPass,
- server vygeneroval token, uložil ho s expiráciou a poslal reset link,

- aplikácia token spracovala cez deep link v ResetPass a odoslala nové heslo,
- server heslo zahashoval a reset token odstránil.

Výsledkom bolo bezpečné obnovenie účtu bez potreby zásahu administrátora.

8. 4.3 Dashboard – načítanie, ukladanie a synchronizácia dát

V Dashboarde sme dosiahli:

- načítanie používateľských údajov z AsyncStorage,
- zobrazenie obsahu cez taby,
- lokálnu prácu s dennými súhrnmi a ich synchronizáciu na server,
- pravidelné odosielanie zmien cez endpointy dennej spotreby.

Výsledkom bol prehľad, ktorý sa vedel rýchlo spustiť z cache a zároveň ostal synchronizovaný so serverom.

9. 4.4 Recepty – generovanie, ukladanie a „zjesť recept“

V receptoch sme dosiahli:

- generovanie receptu na serveri na základe preferencií a voliteľne podľa špajze,
- uloženie receptu do databázy používateľa,
- načítanie a zmazanie uložených receptov,
- započítanie receptu do dennej spotreby.

Výsledkom bol receptový modul s jasným prepojením na používateľský profil a na denný prehľad.

10. 4.5 Notifikácie – plánovanie a správa časov

Pripravili sme modul notifikácií, ktorý vedel:

- vyžiadať povolenia,
- naplánovať denné notifikácie,
- uložiť časy notifikácií a umožniť ich zmenu.

Výsledkom bola pripravená funkcionalita na pravidelné pripomienky.

---

11. 5 Diskusia (upravené – iba moje časti)

Pri mojich častiach sme identifikovali viacero silných stránok aj obmedzení.

Za silné stránky sme považovali:

- **reset hesla cez token** s expiráciou a hashovaním hesla, čo bolo bezpečnejšie než jednoduchá zmena hesla bez overenia,
- **deep link spracovanie** v ResetPass, vďaka čomu reset fungoval aj pri štarte aplikácie z e-mailu,
- **synchronizáciu dennej spotreby** medzi klientom a serverom, ktorá znižovala riziko straty dát,
- **AI generovanie receptov s validáciou JSON** na serveri, čo zlepšilo stabilitu výstupu pre klienta,
- **modul notifikácií** s rušením starých notifikácií, čím sme predišli duplicite.

Za obmedzenia sme považovali:

- notifikácie boli pripravené, ale ich automatické zapnutie po prihlásení sme mali v logike dočasne vypnuté, takže výsledok závisel od toho, či ich neskôr aktivujeme v UI,
- pri práci s lokálnou cache a serverom existovalo riziko dočasného nesúladu dát (najmä ak používateľ menil dáta pri nestabilnom pripojení),
- pri generovaní receptov sme boli závislí od dostupnosti externého AI modelu a od správnosti jeho JSON odpovede, preto sme museli zaviesť retry mechanizmus.

6 Záver

V práci sme navrhli a implementovali mobilnú aplikáciu, ktorá umožňovala používateľovi spravovať účet, doplniť profilové údaje a pracovať s denným prehľadom spotreby. V rámci riešenia sme vytvorili klientskú časť v React Native (Expo) a serverovú časť postavenú na Node.js/Express s databázou MongoDB. Týmto sme dosiahli funkčné klient-server riešenie, ktoré vedelo ukladať a načítavať dáta používateľa a súčasne ponúkalo rýchlu odozvu vďaka lokálnemu ukladaniu údajov.

Ako hlavné výsledky sme považovali:

- implementáciu registrácie, prihlásenia a automatického prihlásenia,

- implementáciu profilu používateľa a výpočtov cieľov v prehľade,
- správu špajze vrátane skenovania produktov cez externé API a ukladania do databázy,
- receptový modul so statickými aj AI generovanými receptami a možnosťou započítania do dennej spotreby,
- obnovu hesla cez e-mailový token a deep link,
- pripravenú implementáciu notifikácií pre budúce rozšírenie.

Zároveň sme identifikovali oblasti, ktoré by bolo vhodné v budúcnosti zlepšiť, najmä bezpečnosť ukladania prihlasovacích údajov, robustnejšiu synchronizáciu dát a plné zapojenie notifikácií do používateľského rozhrania. Napriek týmto obmedzeniam sme vytvorili ucelený základ aplikácie, ktorý bol pripravený na ďalší vývoj a rozširovanie podľa potrieb používateľov.