



Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

LUCRARE DE LICENȚĂ

Absolvent: Emil BOTEZATU

Coordonator: Lect. Dr. Anca VASILESCU

Brașov
Iulie 2023



Universitatea
Transilvania
din Braşov

FACULTATEA DE MATEMATICĂ
ŞI INFORMATICĂ

LUCRARE DE LICENŢĂ

SmartWash: soluție inteligentă de gestionare a spălătoriilor din
campusul UniTBv

Absolvent: Emil BOTEZATU

Coordonator: Lect. Dr. Anca VASILESCU

Braşov
Iulie 2023

Cuprins

1	Introducere	4
1.1	Motivație	4
1.2	Scopul și structura lucrării	5
2	Tranziția digitală în gestionarea spălătoriilor	6
2.1	Descrierea problemelor existente în gestionarea spălătoriilor universitare	6
2.2	Prezentarea soluției propuse: aplicația de spălătorie inteligentă	8
3	Tehnologii utilizate	10
3.1	Typescript	10
3.2	React Native	11
3.2.1	Elemente fundamentale ale programării în React Native	11
3.2.2	Arhitectura React Native	12
3.2.3	Mediul de dezvoltare Expo	15
3.2.4	React Navigation: Principiile de bază ale navigării . . .	15
3.2.5	Styling în React Native: CSS-in-JS	16
3.2.6	Design de interfață cu Tamagui	16
3.2.7	Gestionarea formularelor cu React Hook Form	17
3.2.8	TanStack Query: cache, performanță și interogarea da- telor	18
3.2.9	Zustand: gestionarea state-ului global	18
3.2.10	Comunicarea HTTP cu Axios	19
3.3	Node.js	21
3.3.1	Introducere în Node.js și rolul său în dezvoltarea server- side	21
3.3.2	Arhitectura de backend în cadrul Node.js	21
3.3.3	Express.js: Un cadru rapid, nelimitat și minimalist pentru Node.js	23
3.4	SQL	24
3.4.1	Introducere în SQL și rolul său în gestionarea datelor .	24

3.4.2	PostgreSQL	25
3.4.3	Supabase: Bază de date cloud în timp real	25
3.4.4	Prisma ORM	25
3.5	Priza inteligentă Gosund SP111	28
3.5.1	Caracteristicile și specificațiile tehnice	28
3.6	Tasmota: Controlul și monitorizarea dispozitivelor IoT	29
3.6.1	Introducere în Tasmota și rolul său în (IoT)	29
3.6.2	Interfața web Tasmota	29
3.7	Node-RED	31
3.7.1	Introducere în Node-RED	31
3.7.2	MQTT: Protocolul de mesagerie pentru sistemele de comunicare IoT	31
3.7.3	Configurarea și utilizarea MQTT în Node-RED	32
4	Funcționalitățile Aplicației	33
4.1	Noțiuni generale	33
4.2	Ecranul de autentificare	34
4.2.1	Validările Frontend	36
4.2.2	Fluxul cu JWT (JSON Web Token)	37
4.3	Conectarea prizei Gosund SP111 la aplicația SmartWash	39
4.3.1	Procesul de Tasmotizare a Prizei Inteligente	39
4.3.2	Configurarea setărilor de Wi-Fi și MQTT	41
4.3.3	Cum se stabilește conexiunea între priza inteligentă și Node-RED	42
4.3.4	Configurarea și utilizarea MQTT în Node-RED și Node.js	43
4.4	Sistemul de monitorizare în timp real a spălătoriei	44
4.5	Sistemul de rezervări	46
4.5.1	Crearea unei rezervări	46
4.5.2	Procesarea rezervărilor și stările mașinii	48
4.5.3	Prezentarea ecranului principal și opțiunile utilizatorului	49
4.5.4	Activarea mașinilor de spălat cu ajutorul codurilor QR	49
4.5.5	Comunicarea consumului de curent de către dispozitive	52
4.5.6	Algoritmul de decizie pentru oprire	53
4.5.7	Ecranul istoricului de rezervări	56
4.6	Sistemul de notificări	57
4.6.1	Prezentarea ecranului de inbox	57
4.6.2	Trimiterea de notificări folosind Expo	58
4.7	Chat-ul integrat	60
4.7.1	Prezentarea ecranului de conversații și chat	60
4.7.2	Conectarea la socket folosind Socket.io	61
4.8	Reprogramarea mai rapidă	62

4.8.1	Algoritmul de reprogramare: Problema rucsacului . . .	62
4.8.2	Detalii de implementare	63
4.9	Profilul personal al utilizatorului	65
4.10	Persistența datelor	66
4.10.1	Schema bazei de date	66
5	Concluzii și perspective de dezvoltare	68
5.1	Concluzii generale	68
5.2	Perspective de dezvoltare	69

Capitolul 1

Introducere

1.1 Motivație

În societatea contemporană, tehnologia joacă un rol esențial în a modela și îmbunătăți aspectele esențiale ale vieții noastre. Acest fapt este evident în special în sectorul educațional, unde tehnologia digitală a fost integrată în toate nivelurile pentru a eficientiza procesele și a îmbunătăți experiența de învățare a studenților.

În contextul universitar, era digitală a deschis noi orizonturi pentru învățare și a îmbunătățit experiența studenților din campus. Cursuri online, biblioteci digitale, aplicații pentru organizare și colaborare - toate acestea au devenit parte integrantă a vieții studenților. Cu toate acestea, există încă domenii ale vieții pe campus care pot fi îmbunătățite prin tehnologie. Unul dintre acestea este gestionarea spălătoriilor.

Chiar și în era digitală, gestionarea spălătoriilor în căminele studențești rămâne o problemă în mare parte nerezolvată. Procesul de rezervare și utilizare a mașinilor de spălat poate fi adesea ineficient, inconvenabil și departe de a fi sustenabil din punct de vedere energetic. Astfel, am ajuns la concluzia că există o nevoie și un potențial imens pentru o soluție digitală în acest domeniu.

Prin urmare, în acest sector există un potențial semnificativ de dezvoltare a unei soluții digitale. Aplicația „SmartWash” reprezintă o soluție la aceste necesități și vine în ajutorul studenților pentru a îmbunătăți atât gestionarea spălătoriilor din căminele universitare cât și pentru a elimina eventualele probleme apărute, în timp ce contribuie la o utilizare mai responsabilă a resurselor.

1.2 Scopul și structura lucrării

Scopul principal al acestei lucrări este de a documenta procesul de creare și implementare a aplicației „SmartWash”, de la ideea inițială până la finalizare. Aceasta își propune să evidențieze modul în care tehnologia poate fi folosită pentru a îmbunătăți experiența studenților și a contribui la un mediu universitar mai sustenabil și eficient din punct de vedere energetic.

Obiectivele lucrării sunt multiple. În primul rând, se urmărește identificarea și descrierea problemelor specifice cu care se confruntă studenții când vine vorba de gestionarea spălătoriilor din caminele universitare și descrierea unei soluții digitale, în acest caz, aplicația „SmartWash”.

Un alt obiectiv este prezentarea și explicarea tehnologiilor software folosite în dezvoltarea aplicației, precum și modul în care diferitele componente ale sistemului interacționează. În același timp, lucrarea își propune să evidențieze funcționalitățile cheie ale aplicației și modul în care acestea răspund nevoilor utilizatorilor săi.

În final, ne preocupă discutarea potențialelor îmbunătățiri și dezvoltări viitoare ale aplicației, oferind astfel o perspectivă asupra viitorului acesteia.

Capitolul 2

Tranziția digitală în gestionarea spălătoriilor

2.1 Descrierea problemelor existente în gestionarea spălătoriilor universitare

Unul dintre marile obstacole cu care se confruntă studenții este lipsa unui sistem eficient de programare a mașinilor de spălat. Aceasta duce adesea la aglomerații și așteptări neașteptat de lungi. Studenții pot petrece mult timp așteptând pentru o mașină disponibilă, ceea ce afectează atât timpul lor de studiu, cât și timpul lor liber.

În cazul spălătoriilor din campusul UniTBv, putem constata că atât în complexul Memorandului, cât și în complexul Colina, există o necesitate de organizare a spălătoriilor. În prezent, sistemul fizic de înscriere există doar în complexul Colina, în anumite spălătorii, după cum se poate vedea în tabelul de mai jos:

Interval orar 9:00-12:00			
Nume si prenume	Camera	Masina	Uscator
John Doe	500	1	U1
Jane Smith	501	2	
Alice Johnson	502	3	
Alex Brown	503	4	U2

Tabela 2.1: Tabelul de înscriere pentru intervalul orar 9:00-12:00.

Tabelul de mai sus prezintă câteva probleme și limitări în ceea ce privește utilizarea spălătoriilor. Mai jos sunt prezentate câteva dintre problemele cauzate de sistemul actual:

- **Ocuparea simultană a mașinilor de spălat și a uscătoarelor:** uscătoarele și mașinile de spălat sunt în aceeași secțiune ceea ce duce la confuzii și întârzieri.
- **Intervale fixe de timp:** Sistemul actual oferă doar intervale fixe de trei ore pentru utilizarea mașinilor de spălat. Dacă un student termină de folosit mașina mai devreme, intervalul rămâne nefolosit, ceea ce duce la o utilizare inefficientă a resurselor spălătoriei.
- **Absența unui sistem de rezervare online:** studenții pot întâmpina dificultăți în a se sincroniza cu colegii lor de cameră sau de pe etaj pentru a folosi mașina de spălat. Acest lucru poate conduce la tensiuni și neînțelegeri.
- **Necesitatea prezenței fizice pentru rezervare:** În sistemul actual, pentru a face o rezervare, studenții trebuie să fie prezenți fizic la spălătorie. Acest lucru poate fi neconvenabil, deoarece le-ar putea întrerupe alte activități și ar putea duce la cozi lungi la spălătorie.
- **Lipsa unui sistem de notificări:** În sistemul actual, studenții trebuie să își verifice în mod constant hainele pentru a vedea dacă ciclul de spălare s-a terminat. Aceasta poate duce la un timp de așteptare neașteptat de lung sau la ratări ale oportunităților de a folosi mașina de spălat, în special în perioadele de vârf, când cererea este mare. În plus, studenții pot ajunge să uite de hainele lor, blocând astfel mașina pentru alți utilizatori.
- **Lipsa unui sistem de control al accesului:** hainele studenților sunt supuse unui risc de furt sau deteriorare. O altă problemă o constituie accesul neautorizat la resursele căminului.

În concluzie, sistemul actual de gestionare a spălătoriilor universitare are mai multe limitări semnificative. Acestea includ lipsa unui sistem de notificări, necesitatea prezenței fizice pentru a face o rezervare, folosirea inefficientă a intervalelor de timp, și limitările în disponibilitatea resurselor. Toate acestea evidențiază necesitatea unei soluții digitale care să facă procesul de rezervare și utilizare a spălătoriilor mai eficient și mai ușor de gestionat de către și pentru studenți.

2.2 Prezentarea soluției propuse: aplicația de spălătorie inteligentă

În cadrul acestei lucrări, propunem o soluție software inteligentă pentru a răspunde provocărilor identificate în gestionarea spălătoriilor din cadrul căminelor universitare. Aceasta este prezentată sub forma unei aplicații de spălătorie inteligentă, denumită „SmartWash”.

Unul dintre elementele distinctive ale aplicației „SmartWash” este utilizarea unei prize inteligente, care poate monitoriza și raporta consumul de energie al mașinilor de spălat. Atunci când mașina de spălat a terminat ciclul și consumul de energie scade sub un anumit prag, priza inteligentă trimite o notificare către aplicație. Astfel, utilizatorii sunt informați în timp real când rufe lor sunt gata, permițându-le să-și planifice timpul mai eficient și să elibereze mașina de spălat pentru următorul utilizator.

În plus, SmartWash include o funcție de „rezervare flexibilă”. Aceasta permite utilizatorilor să aleagă dacă doresc ca rezervarea lor să fie reprogramată mai devreme, în cazul în care o rezervare anterioară este anulată sau încheiată mai devreme decât era planificat inițial. Prin această facilitate, studenții pot beneficia de un serviciu mai rapid și pot optimiza utilizarea mașinilor de spălat disponibile.

„SmartWash” asigură și că doar persoana care a făcut rezervarea poate folosi mașina de spălat. Acest lucru se realizează prin controlul accesului la priza inteligentă a mașinii. La începutul perioadei de rezervare, utilizatorul scanează un cod QR generat de aplicație, care îi autorizează utilizarea prizei inteligente și, implicit, a mașinii de spălat. Acest mecanism, nu doar că securizează accesul la mașina de spălat, dar asigură și rularea corectă a programărilor de rezervări.

„SmartWash” încurajează și comunicarea între studenți prin intermediul unui chat integrat. Acesta le permite studenților să interacționeze cu persoana care a folosit mașina de spălat înaintea lor. Astfel, dacă un student și-a uitat hainele în mașină sau dacă există alte informații relevante de transmis, acestea pot fi comunicate eficient și discret prin intermediul chat-ului. Această funcție nu numai că facilitează rezolvarea unor situații neprevăzute, dar contribuie și la crearea unei comunități de studenți care colaborează și se ajută reciproc.

În sinteză, „SmartWash” aduce un set valoros și pragmatic de caracteristici menite să eficientizeze utilizarea spălătoriilor în căminele universitare, eliminând problemele identificate în gestionarea tradițională. Prin digitalizare, flexibilitate, securitate și promovarea comunicării, aplicația oferă o soluție pentru îmbunătățirea experienței studenților în spațiile de spălare ale

UniTBv.

Capitolul 3

Tehnologii utilizate

3.1 Typescript

TypeScript este un limbaj de programare open-source dezvoltat și întreținut de Microsoft. Este un superset al JavaScript, ceea ce înseamnă că orice cod JavaScript valid poate fi, de asemenea, un cod TypeScript valid.[2]

Typescript se poate adopta gradual adăugând, tipuri unui proiect JavaScript incremental. Fiecare pas adaugă suport editor-ului și îmbunătățește calitatea codului sursă.

TypeScript introduce conceptul de tipuri statice în lumea JavaScript, permițând dezvoltatorilor să adauge tipuri de date la variabile, funcții, parametri și altele. Aceste tipuri statice sunt un instrument puternic care poate ajuta dezvoltatorii să scrie cod mai sigur și mai eficient, permițând unor editoare de cod inteligente și instrumentelor de timp de dezvoltare să ofere un feedback instant și precis.

TypeScript introduce interfețele, care sunt un mod de a defini "forme" pentru obiecte. De exemplu:

```
interface Account {  
    id: number  
    displayName: string  
    version: 1  
}  
  
function welcome(user: Account) {  
    console.log(user.id)  
}
```

Avantajele cheie ale utilizării TypeScript în acest proiect sunt:

- **Siguranța tipurilor:** TypeScript este un superset strict al JavaScript, adăugând un sistem puternic de tipuri. Acest lucru permite identificarea rapidă și eficientă a multor erori în timpul dezvoltării, făcând codul mai sigur și mai ușor de întreținut.
- **Autocomplete și refactorizare ușoară:** Cu TypeScript, tool-urile de dezvoltare pot oferi funcționalități mai puternice de autocomplete, ajutând la scrisul codului mai rapid și mai precis. De asemenea, refactorizarea devine mai simplă și mai sigură, deoarece putem schimba numele unei variabile sau a unei funcții și toate referințele vor fi actualizate corespunzător.
- **Productivitate sporită:** are suport pentru ultimele funcționalități ECMAScript, cum ar fi clasă, săgeți și promisiuni, plus altele, cum ar fi tipurile generice și modulele. Acest lucru îmbunătățește productivitatea dezvoltatorilor și face codul mai ușor de citit și înțeles.

În acest proiect, toate aceste avantaje ale TypeScript au contribuit la dezvoltarea unei aplicații robuste, sigure și ușor de întreținut. Prin urmare, utilizarea TypeScript a fost o decizie strategică cheie în realizarea acestui proiect.

3.2 React Native

React Native este o bibliotecă de programare open source dezvoltată de Facebook pentru a crea aplicații mobile. React Native permite dezvoltatorilor să creeze aplicații mobile natice cu JavaScript și React. Acesta combină elementele de bază ale JavaScript cu biblioteca React și permite crearea de aplicații iOS și Android cu o singură bază de cod, lucru care înseamnă că aplicațiile funcționează pe ambele platforme fără nicio modificare a codului.[13]

3.2.1 Elemente fundamentale ale programării în React Native

Programarea în React Native implică înțelegerea unor concepte și elemente fundamentale. Acestea includ:

- **Componente:** În React Native, toată interfața utilizatorului este compusă din componente. O componentă este o unitate independentă de

cod care reprezintă o parte din interfața utilizatorului. Fiecare componentă are propriile sale proprietăți (props) și stări (state), și pot fi compuse pentru a construi interfețe complexe.

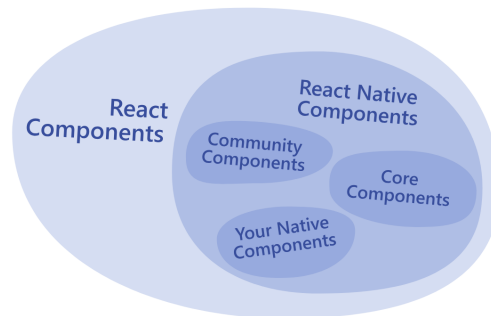


Figura 3.1: Arhitectura componentelor React Native

- **TSX:** este o extensie a limbajului TypeScript care adaugă sintaxa XML în TypeScript. TSX permite definirea structurii și aspectului componentelor direct în codul TypeScript. Similar cu JSX pentru JavaScript, TSX combină puterea JavaScript cu beneficiile suplimentare oferite de TypeScript, cum ar fi verificările de tip la compilare, pentru a ajuta dezvoltatorii să creeze aplicații mai robuste și mai ușor de întreținut.
- **Starea Componentei (State):** Starea este un obiect care stochează valori care pot influența comportamentul și aspectul unei componente. Atunci când starea unei componente se schimbă, componenta este re-renderizată pentru a reflecta noile valori.

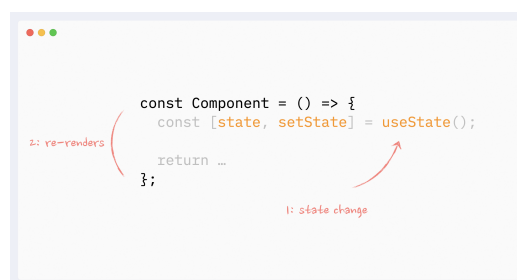


Figura 3.2: Re-renderarea componentelor React Native cu state

3.2.2 Arhitectura React Native

Arhitectura unei aplicații React Native poate varia în funcție de complexitatea proiectului, preferințele dezvoltatorului și necesitățile specifice ale

aplicației.

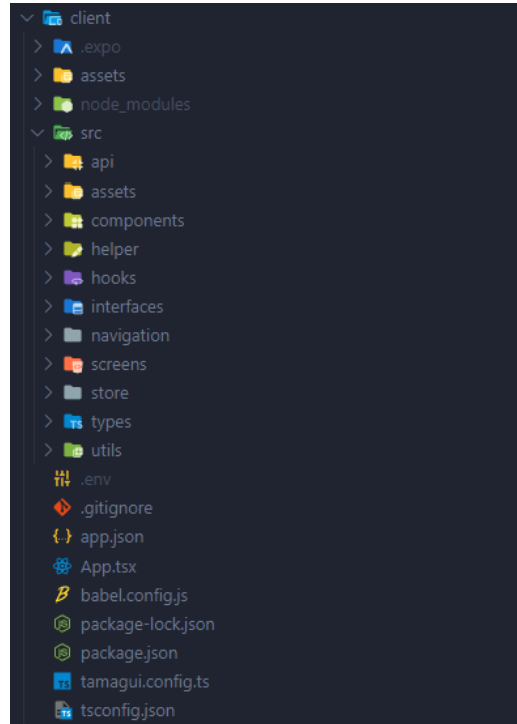


Figura 3.3: Structura client a aplicației „SmartWash”

Iată o scurtă prezentare a structurii de director a proiectului React Native pentru aplicația „SmartWash”:

- **node_modules:** este un director în care Node păstrează fișierele JavaScript pentru proiectele tale. Când instalezi un pachet folosind npm (Node Package Manager), acesta este descărcat și stocat în acest director.
- **src:** Acesta este directorul principal al codului sursă al aplicației. El conține mai multe subdirectoare:
 - **api:** găzduiește toate solicitările de rețea și funcțiile care interacționează cu backend-ul aplicație
 - **assets:** sunt stocate toate resursele statice ale aplicației, cum ar fi imagini, fonturi și stiluri globale.
 - **components:** conține toate componentele reutilizabile la nivel de aplicație.

- **hooks:** sunt definite hook-urile personalizate pentru a fi folosite în întreaga aplicație.
 - **interfaces:** toate tipurile și interfețele TypeScript utilizate în aplicație.
 - **navigation:** găzduiește toată logica de navigare a aplicației, inclusiv definirea stack-urilor de navigare și rutelor.
 - **screens:** conține toate ecranele sau paginile aplicației.
 - **store:** este gestionată starea globală a aplicației.
 - **utils:** conține diverse funcții utilitare care sunt utilizate în mai multe locuri în aplicație.
- **.env:** este folosit pentru a stoca variabilele de mediu specifice proiectului. Acesta este un loc sigur pentru a stoca date sensibile, cum ar fi cheile API, care nu ar trebui să fie hardcodate în codul sursă al aplicației.
 - **.gitignore:** este utilizat de Git pentru a determina care fișiere și directoare să ignore atunci când face commit-uri.
 - **tamagui.config.ts:** găzduiește configurația pentru Tamagui
 - **package.json:** conține metadate despre proiect, precum numele proiectului, versiunea, autorul și dependențele.
 - **App.tsx:** este punctul de intrare al aplicației. Aici este definită componenta rădăcină a aplicației.
 - **babel.config.js:** conține configurarea Babel, care este utilizat pentru a transforma codul JavaScript modern într-un cod care poate fi executat pe majoritatea browserelor și a platformelor de dispozitive mobile.

3.2.3 Mediul de dezvoltare Expo

Expo reprezintă un set de instrumente și servicii open-source, bazate pe React Native, care permit crearea și publicarea de aplicații pentru Android și iOS cu JavaScript și TypeScript. Este un cadru care oferă acces la o gamă largă de API-uri ale platformelor native, împreună cu un UI bine proiectat și componente reutilizabile.[22]

Expo Go este aplicația mobilă ce poate fi folosită pentru a rula aplicația React Native direct pe dispozitiv, în timpul dezvoltării. În mod tipic, pentru dezvoltarea unei aplicații, este nevoie de un emulator sau de un dispozitiv fizic conectat la calculator pentru a vedea schimbările. Cu Expo Go, tot ce este necesar este să se scaneze un cod QR din consola de dezvoltare și aplicația va fi lansată direct pe dispozitiv.

```
npx create-expo-app AwesomeProject

cd AwesomeProject
npx expo start
```

Listing 1: Exemplu de creare aplicație React Native cu Expo

3.2.4 React Navigation: Principiile de bază ale navigării

React Navigation este o bibliotecă extrem de importantă în cadrul unei aplicații React Native. Aceasta este responsabilă de gestionarea tranzițiilor între diferitele ecrane ale aplicației și oferă o serie de componente și funcționalități care permit crearea unui sistem de navigare complex și intuitiv.[13]

- *Stack Navigator* este un tip de navigator folosit în React Navigation care permite navigarea prin „împingerea” și „poparea” ecranelor într-un „stivă” de ecrane. În acest context, „împingerea” unui ecran înseamnă adăugarea acestuia în vârful stivei și afișarea lui, în timp ce „poparea” unui ecran înseamnă îndepărtarea lui din vârful stivei și revenirea la ecranul anterior.
- *Drawer Navigator* este un alt tip de navigator care implementează un model de navigare tip sertar. Acesta permite utilizatorilor să gliseze de pe marginea ecranului sau să apese pe un buton pentru a afișa un meniu lateral (sau „sertar”) care conține legături către diferite ecrane. Drawer

Navigator este frecvent utilizat în aplicații cu multe ecrane principale, cum ar fi aplicațiile de social media sau de știri.

- *Bottom Tab Navigator* este un tip de navigator care afișează o bară de navigare în partea de jos a ecranului, cu butoane care permit utilizatorilor să comute între diferite ecrane. Fiecare buton din bara de navigare este asociat cu un ecran specific și poate fi personalizat cu un titlu și o pictogramă. Bottom Tab Navigator este ideal pentru aplicațiile cu câteva ecrane principale de nivel superior care trebuie să fie accesibile în orice moment.

În proiectul „SmartWash”, aceste tipuri de navigatori au fost folosite pentru a asigura o experiență de navigare intuitivă și ușor de utilizat pentru utilizatori.

3.2.5 Styling în React Native: CSS-in-JS

În React Native, stilizarea componentelor se realizează folosind un sistem denumit CSS-in-JS, care este o abordare modernă pentru definirea stilurilor în aplicațiile JavaScript. Această abordare permite scrierea codului CSS direct în fișierele JavaScript.

Un avantaj major al acestei abordări este posibilitatea de a folosi toate capacitățile JavaScript atunci când se definește stilul, cum ar fi variabilele, funcțiile și condițiile. Acest lucru aduce un nivel de flexibilitate și putere în stilizare care nu este disponibil în CSS pur.

3.2.6 Design de interfață cu Tamagui

Tamagui este o bibliotecă de componente UI pentru React care se concentrează pe eficiență și pe implementarea unui design de interfață eficient și atrăgător. Acesta oferă un set de componente UI predefinite care pot fi utilizate pentru a construi rapid și eficient interfețe de utilizator.

Conform experienței cu diverse aplicații, se observă că între 30 și 50% din componente tind să fie "flattened" (platizate), adică transformate în structuri mai simple și eficiente. În plus, se constată că procentul poate fi crescut doar prin conștientizarea modului în care procesul de platizare optimizează componentele (adică adăugând comentariul `// debug` la începutul fișierului pentru a arăta un rezultat mai complet).[17]



Figura 3.4: Performanta librăriei pe web față de alte librări UI

Între timp, în ceea ce privește Native, deoarece nu se pot face optimizări dincolo de codul de bază React Native, beneficiile sunt mai puțin semnificative. Cu toate acestea, rezultatele sunt impresionante, având în vedere că se obține o performanță la distanță de doar 5% față de codul React Native optimizat manual. În plus, se obține un întreg set de funcții fără niciun cost adițional. Astfel, Tamagui îmbunătățește semnificativ eficiența și performanța în dezvoltarea aplicațiilor React Native.[17]

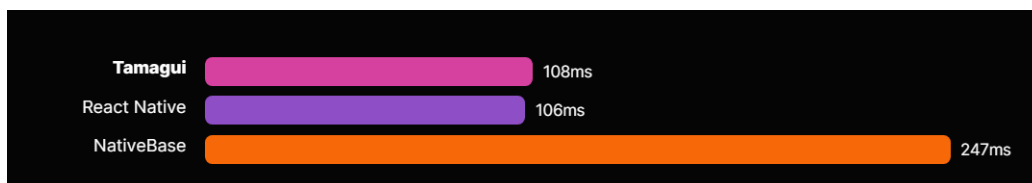


Figura 3.5: Performanța librăriei în React Native comparativ cu codul nativ

Prin urmare, utilizarea Tamagui în proiectul „SmartWash” nu aduce doar avantaje estetice și de eficiență în dezvoltare, ci îmbunătățește și performanța aplicației, oferind o experiență mai fluidă utilizatorilor.

3.2.7 Gestionarea formularelor cu React Hook Form

React Hook Form este o bibliotecă pentru gestionarea formularelor în aplicațiile React și React Native care încurajează performanța prin îmbinarea modelului de programare cu hook-uri și caracteristica de manipulare necontrolată a formularelor.

Acest model permite dezvoltatorilor să evite re-renderizările inutile și complexitatea asocierii locale a stării prin folosirea referințelor pentru colectarea valorilor formularelor.

În cadrul aplicației „SmartWash”, React Hook Form a fost folosit pentru a facilita procesul de înregistrare și autentificare a utilizatorilor, precum și pentru formularele de rezervare și gestionare a rezervărilor, asigurând un flux de utilizare fără probleme și o experiență de utilizare excelentă.

3.2.8 TanStack Query: cache, performanță și interogarea datelor

TanStack Query este o bibliotecă puternică pentru gestionarea și sincronizarea datelor la nivel de aplicație în aplicațiile JavaScript și TypeScript. Oferă mecanisme eficiente pentru preluarea, cache-ul, sincronizarea și actualizarea datelor dintr-un back-end, cu o concentrare deosebită pe performanță și experiența de utilizare.

Unul dintre principalele avantaje ale TanStack Query este capacitatea sa de a facilita gestionarea cache-ului de date. Cu un sistem de cache automat, TanStack Query păstrează datele în cache pentru a optimiza performanța și a reduce solicitările de rețea inutile. Acesta suportă, de asemenea, actualizarea în fundal a datelor, precum și refetching-ul la nivel de aplicație sau individual, pe baza dependențelor de date.[18]

TanStack Query oferă o multitudine de funcționalități care îmbunătățesc gestionarea și sincronizarea datelor în aplicațiile JavaScript și TypeScript. Iată câteva dintre aceste funcționalități:[18]

- *Backend agnostic*: TanStack Query nu este legat de niciun backend specific și poate fi folosit cu orice tip de API sau serviciu de date.
- *Auto Caching*: TanStack Query gestionează automat cache-ul datelor, îmbunătățind performanța și reducând solicitările de rețea inutile.
- *Auto Refetching*: TanStack Query oferă funcționalitatea de refetching automat, actualizând datele atunci când se schimbă dependențele.
- *Mutations API*: TanStack Query oferă o API pentru mutații, facilitând modificarea datelor de pe server.
- *Render-as-you-fetch*: Cu TanStack Query, datele pot fi preluate în timp ce componenta este randată, optimizând timpul de încărcare.

3.2.9 Zustand: gestionarea state-ului global

Zustand este o bibliotecă minimalistă, dar puternică, pentru gestionarea stării globale în aplicațiile React. Are o sintaxă simplă și oferă o soluție elegantă pentru partajarea și manipularea datelor între componente.

În esență, Zustand permite crearea unui „magazin” global care poate fi accesat de oricare dintre componente. Acest magazin conține starea aplicației și orice funcții pe care dorim să le utilizăm pentru a actualiza acea stare. Unul dintre principalele avantaje ale utilizării Zustand este că permite evitarea „prop drilling”, adică trecerea datelor prin multiple niveluri de componente.[23]

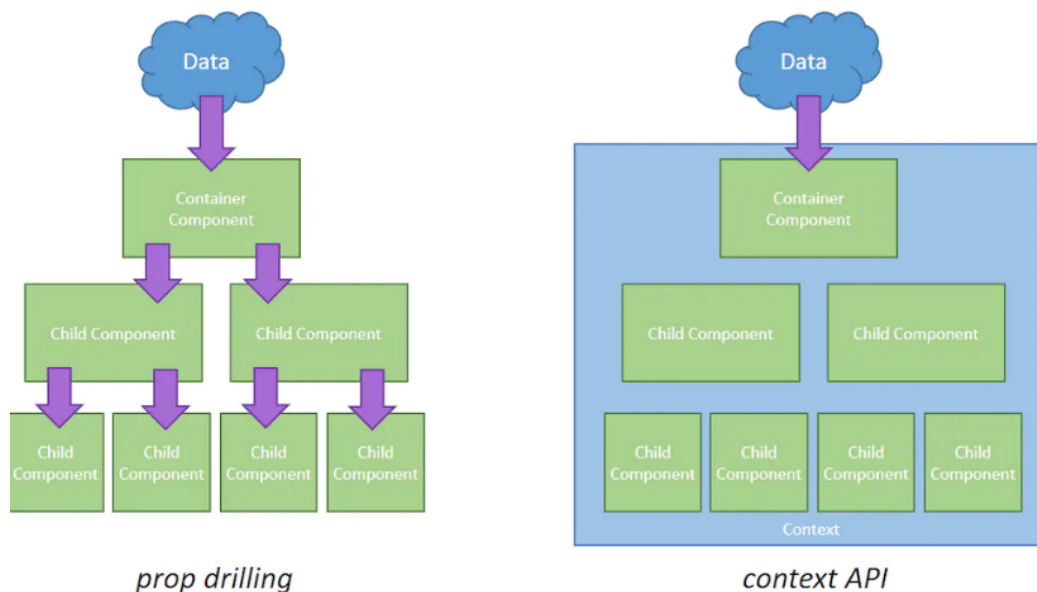


Figura 3.6: Prop drilling vs Zustand [12]

3.2.10 Comunicarea HTTP cu Axios

Axios este o bibliotecă populară pentru comunicarea HTTP în aplicații JavaScript, care poate fi utilizată într-un mediu Node.js sau într-un browser. Aceasta facilitează efectuarea de cereri HTTP către servere și tratarea răspunsurilor.[21]

Arhitectura protocolului de transfer de hipertext (HTTP) definește modalitatea de comunicare între client și server pe Internet. HTTP se bazează pe conceptul de cereri și răspunsuri, unde clientul trimite o cerere la server, iar acesta din urmă răspunde cu un răspuns corespunzător.[3]

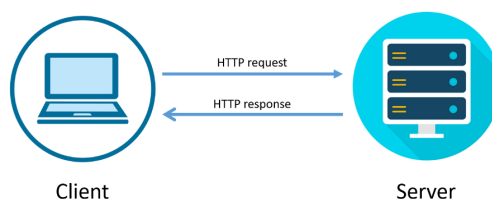


Figura 3.7: Comunicarea HTTP

Prin intermediul Axios, aplicația poate realiza cereri HTTP la server pentru a obține sau a trimite date. Fie că este vorba de obținerea de informații dintr-o bază de date, de efectuarea de acțiuni pe server sau de actualizarea

datelor, Axios se ocupă de trimiterea acestor cereri și de gestionarea răspunsurilor primite.

Utilizarea Axios în aplicația bazată pe React Native și Node.js în cadrul proiectului de spălătorie inteligentă pentru campus aduce multiple beneficii. Axios abstractizează complexitatea comunicării HTTP și oferă un set de funcționalități puternice pentru a comunica eficient între frontend (aplicația mobilă) și backend (serverul).

3.3 Node.js

Node.js este un mediu de execuție JavaScript bazat pe motorul V8 al Google, care permite rularea codului JavaScript pe partea server-ului. A fost creat pentru a extinde utilizarea JavaScript dincolo de mediu browser-ului, oferind posibilitatea dezvoltării aplicațiilor server-side, care rulează pe servere și gestionează cereri și răspunsuri.[26]

3.3.1 Introducere în Node.js și rolul său în dezvoltarea server-side

Unul dintre principalele avantaje ale Node.js este faptul că este bazat pe un model de evenimente și I/O non-blocant, ceea ce înseamnă că poate trata eficient multiple cereri simultan și poate face operații de I/O fără a bloca execuția. Aceasta permite o scalabilitate ridicată și o performanță excelentă în aplicațiile cu un număr mare de utilizatori sau cereri.[26]

3.3.2 Arhitectura de backend în cadrul Node.js

Arhitectura unei aplicații Node.js poate varia în funcție de necesitățile și complexitatea proiectului.

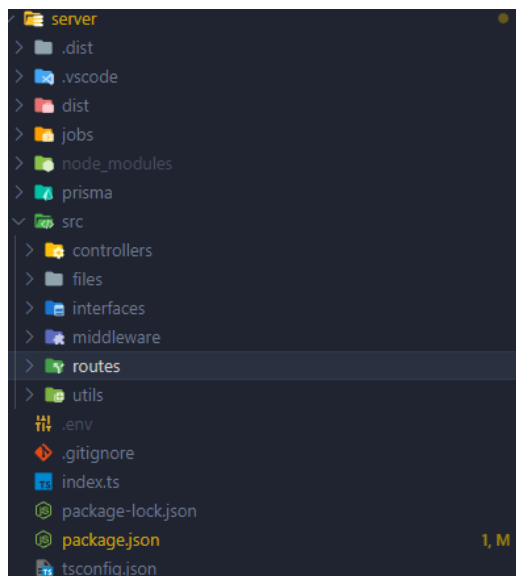


Figura 3.8: Structura aplicației în cadrul aplicației „SmartWash”

Structura proiectului Node.js pentru aplicația SmartWash poate fi descrisă astfel:

- **node_modules:** este un director în care Node păstrează fișierele JavaScript pentru proiectele tale. Când instalezi un pachet folosind npm (Node Package Manager), acesta este descărcat și stocat în acest director.
- **jobs:** conține job-urile care sunt executate utilizând biblioteca cron. Aceste job-uri sunt sarcini automate care sunt programate să ruleze la anumite intervale de timp.
- **prisma:** conține toate fișierele legate de Prisma, un ORM (Object-Relational Mapping) modern folosit pentru a interacționa cu baza de date.
- **src:** este directorul principal al codului sursă al aplicației. El conține mai multe subdirectoare:
 - **controllers:** conține funcțiile de manipulare pentru fiecare rută a aplicației. Aceste funcții, numite adesea „controlere”, sunt responsabile pentru preluarea datelor de intrare, efectuarea operațiilor necesare (cum ar fi interogări de baze de date sau apeluri la alte API-uri) și returnarea unui răspuns către client.
 - **interfaces:** conține toate interfețele TypeScript care sunt utilizate pentru a tipiza diferitele obiecte și funcții din aplicație.
 - **middlewares:** conține toate middleware-urile utilizate în aplicație. Middleware-urile sunt funcții care pot accesa obiectele de cerere și răspuns în Express.js și pot efectua operațiuni pe ele înainte ca cererea să fie redirecționată la următorul handler.
 - **routes:** conține toate rutele pentru aplicație. Fiecare rută reprezintă un punct de terminație API la care utilizatorii pot trimite cereri.
 - **utils:** conține diverse utilități și funcții ajutătoare care sunt utilizate în mai multe locuri din aplicație.
- **.env:** este folosit pentru a stoca variabilele de mediu specifice proiectului. Acesta este un loc sigur pentru a stoca date sensibile, cum ar fi cheile API, care nu ar trebui să fie hardcodate în codul sursă al aplicației.
- **.gitignore:** este utilizat de Git pentru a determina care fișiere și directoare să ignore atunci când face commit-uri.

- **package.json:** conține metadate despre proiect, precum numele proiectului, versiunea, autorul și dependențele.
- **index.ts:** este punctul de intrare principal al aplicației. Conține codul pentru pornirea serverului și conectarea la baza de date.
- **tsconfig.json:** conține configurația pentru compilatorul TypeScript.

3.3.3 Express.js: Un cadru rapid, nelimitat și minimalist pentru Node.js

Express.js este un framework web pentru Node.js care facilitează crearea de aplicații web și API. Este rapid, flexibil și minimalist, oferind un set robust de caracteristici pentru aplicații web și mobile.

În aplicația „SmartWash”, Express.js a fost utilizat pentru a crea API-ul de backend care gestionează toate cererile și răspunsurile serverului, inclusiv autentificarea utilizatorilor, gestionarea rezervărilor de mașini de spălat, notificări și multe altele. Cu ajutorul său, crearea și gestionarea acestor funcționalități complexe devine mult mai simplă și mai eficientă.

```
const router = express.Router();
router.get('/getStudents', getStudents);
export default router;
```

3.4 SQL

SQL sau Structured Query Language, este un limbaj de programare conceput special pentru gestionarea datelor în sisteme de gestionare a bazelor de date relaționale (RDBMS) sau pentru fluxul de procesare în sisteme de gestionare a bazelor de date relaționale.

3.4.1 Introducere în SQL și rolul său în gestionarea datelor

SQL este fundamental pentru gestionarea datelor în majoritatea bazelor de date relaționale, precum MySQL, Oracle, SQLite sau PostgreSQL. Abilitățile sale extinse de interogare și manipulare a datelor îl fac esențial pentru operațiunile cu baze de date. Prin interogări SQL, putem extrage, actualiza, șterge, crea și modifica date, precum și crea și manipula schema unei baze de date.

Există două tipuri principale de baze de date:

- Bazele de date relaționale (RDBMS), care organizează datele în tabele interconectate. Fiecare tabel are un set unic de câmpuri (coloane) și înregistrări individuale (rânduri). Acestea folosesc SQL ca limbaj principal pentru interogarea datelor.[14]
- Bazele de date non-relaționale, sau NoSQL, care nu se bazează pe tabele standard pentru stocarea datelor. Acestea pot folosi o varietate de modele de date, cum ar fi document, cheie-valoare, perechi cheie-valoare sau grafice, în funcție de natura și cerințele datelor.

Alegerea unei baze de date relaționale pentru acest proiect a fost determinată de o serie de factori:

- **Structura datelor:** Aplicația se bazează pe date care au o structură fixă și interconexiuni clare între ele, cum ar fi rezervările, mașinile de spălat și utilizatorii. Acesta este exact tipul de date pentru care bazele de date relaționale, cu modelele lor tabulare și relațiile bine definite, sunt proiectate.
- **Consistență și integritate a datelor:** În cazul unei aplicații de rezervare, este crucial să se mențină consistența și integritatea datelor. Bazele de date relaționale oferă o serie de mecanisme, cum ar fi tranzacțiile și constrângerile de cheie străină, care ajută la asigurarea acestui aspect.

- **Interogări complexe:** SQL, limbajul utilizat în bazele de date relaționale, este extrem de puternic atunci când vine vorba de interogări complexe. Acesta permite realizarea de interogări complicate pentru a extrage informații utile din date.

3.4.2 PostgreSQL

PostgreSQL este un sistem de baze de date relaționale open-source puternic, cu o lungă istorie și o comunitate activă de dezvoltare. Este cunoscut pentru fiabilitatea, integritatea datelor și conformitatea cu standardele SQL. În plus, PostgreSQL este extrem de extensibil și permite dezvoltatorilor să definească propriile tipuri de date, operatoare și funcții.

PostgreSQL este bine cunoscut pentru capacitatea sa de a gestiona cantități mari de date și pentru performanța sa de top, care poate fi mărită prin diferite tehnici de optimizare. De asemenea, PostgreSQL oferă suport pentru tranzacții, vizualizări, subinterogări, chei străine și proceduri stocate.

3.4.3 Supabase: Bază de date cloud în timp real

Supabase este o platformă de bază de date cloud care se distinge prin performanță excelentă și scalabilitate înaltă. Fiind construită pe PostgreSQL, una dintre cele mai puternice și mai sigure sisteme de gestionare a bazelor de date relaționale, Supabase îmbină robustețea și flexibilitatea acestei baze de date cu beneficiile aduse de cloud.[16]

Performanța este, de asemenea, un aspect crucial. Supabase promite timpi de răspuns rapizi și un nivel înalt de fiabilitate, îmbunătățind astfel experiența utilizatorilor finali. Datorită caracteristicilor sale, precum API-uri în timp real și funcționalități serverless, Supabase permite dezvoltarea de aplicații reactive și eficiente.

3.4.4 Prisma ORM

Prisma ORM (Object-Relational Mapping) este un instrument extrem de eficient care permite dezvoltatorilor să lucreze cu baze de date SQL într-un mod mai abstract și mai sigur. În loc de a scrie interogări SQL directe, acesta utilizează metodele și funcțiile Prisma pentru a interoga și a manipula datele.

Prisma este proiectat pentru a lucra cu TypeScript și JavaScript modern și oferă o experiență de dezvoltare foarte plăcută. Acesta oferă o verificare de tipuri puternică, care minimizează erorile în timpul fazei de dezvoltare. Cu Prisma, se poate genera în mod automat un client de bază de date tipizat, care este complet personalizat pentru schema bazei de date.[9]

Prisma ORM funcționează pe baza unei scheme, care este o reprezentare de nivel înalt a bazei de date. Această schemă este definită într-un fișier *schema.prisma* și conține modelele pentru fiecare tabel din baza de date. Aceste modele corespund tabelului din baza de date și conțin informații despre coloanele și relațiile tabelului.[9]

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model student {
  id          Int           @id @default(autoincrement())
  address_id  Int
  dorm_id     Int
  email       String
  first_name  String
  last_name   String
  registration_number Int
  password    String
  notification_token String
}
```

În această schemă, putem configura 3 elemente:

- **Sursa de date:** specifică conexiunea la baza de date (prin variabile de sistem)
- **Generator:** indică generarea clientului Prisma
- **Modelul de date:** definește modelele aplicației

Prin folosirea acestor modele, putem interoga și manipula datele din baza de date cu ajutorul clientului Prisma. De exemplu, pentru a obține toți utilizatorii din baza de date, am putea să facem ceva de genul:

```
const allStudents = await prisma.student.findMany()
```

Prisma ORM oferă flexibilitatea de a executa interogări SQL brute¹ direct pe baza de date. Acest lucru poate fi util atunci când este nevoie de o interogare complexă sau personalizată care nu poate fi ușor reprezentată prin intermediul API-ului Prisma:

```
const allStudents = await prisma.queryRaw(`SELECT * FROM student`);
```

¹Funcția *queryRaw* este concepută pentru a preveni SQL Injection prin utilizarea parametrilor legați (bind parameters) în locul concatenării string-urilor pentru a construi interogările.

3.5 Priza inteligentă Gosund SP111

3.5.1 Caracteristicile și specificațiile tehnice

Priza inteligentă Gosund SP111 este un dispozitiv de automatizare a locuinței care poate transforma aproape orice aparat electric obișnuit într-unul inteligent. Acest dispozitiv se conectează la rețeaua Wi-Fi de acasă și permite utilizatorilor să controleze și să programeze funcționarea aparatelor electrice. Această priză este capabilă de tasmotiz-are, un proces care va fi descris în paginile următoare și de asemenea crucial în dezvoltarea aplicației „SmartWash”.



Figura 3.9: Gosund SP111

Mai jos sunt prezentate specificațiile tehnice ale prizei Gosund SP111:

Utilizare	Interior
Tip montare	Aparent
Smart	Da
Tensiune alimentare	230 V
Amperaj	15 A
Putere	3450W

3.6 Tasmota: Controlul și monitorizarea dispozitivelor IoT

Tasmota este un firmware open-source pentru dispozitive IoT (Internet of Things) bazate pe chipsetul ESP8266 și ESP32. Acesta permite controlul și monitorizarea dispozitivelor IoT, cum ar fi prizele inteligente, întrerupătoare, becurile, senzorii și multe altele, printr-o interfață web simplă și ușor de utilizat.[19]

3.6.1 Introducere în Tasmota și rolul său în (IoT)

O caracteristică distinctivă a Tasmota este flexibilitatea sa în ceea ce privește conectivitatea. Firmware-ul suportă o gamă largă de protocoale de comunicare, inclusiv MQTT (Message Queuing Telemetry Transport), HTTP (HyperText Transfer Protocol) și KNX (un standard pentru automatizarea clădirilor). Aceasta înseamnă că dispozitivele care rulează Tasmota pot fi integrate cu o varietate de sisteme de automatizare a locuinței, precum Home Assistant, Domoticz sau Node-RED.

Un alt aspect important al Tasmota este faptul că respectă confidențialitatea utilizatorilor. Toate datele sunt stocate și procesate local pe dispozitiv, fără a necesita o conexiune la un server în cloud.

În cadrul proiectului „SmartWash”, Tasmota a fost folosit pentru a **tasmotiza** (adică a înlocui firmware-ul original cu Tasmota) priza inteligentă Gosund SP111. Acest lucru a permis controlul și monitorizarea prizei direct din aplicația „SmartWash”, oferind o integrare perfectă între dispozitivul fizic și interfața digitală a aplicației. Mai mult, datorită suportului Tasmota pentru MQTT, a fost posibilă implementarea unui sistem de notificări în timp real care informează utilizatorii atunci când mașina de spălat sau uscătorul este gata.

3.6.2 Interfața web Tasmota

Interfața web a Tasmota oferă un nivel de control și configurare profund, permițând personalizarea dispozitivelor la un nivel de detaliu rar întâlnit în alte soluții de firmware.

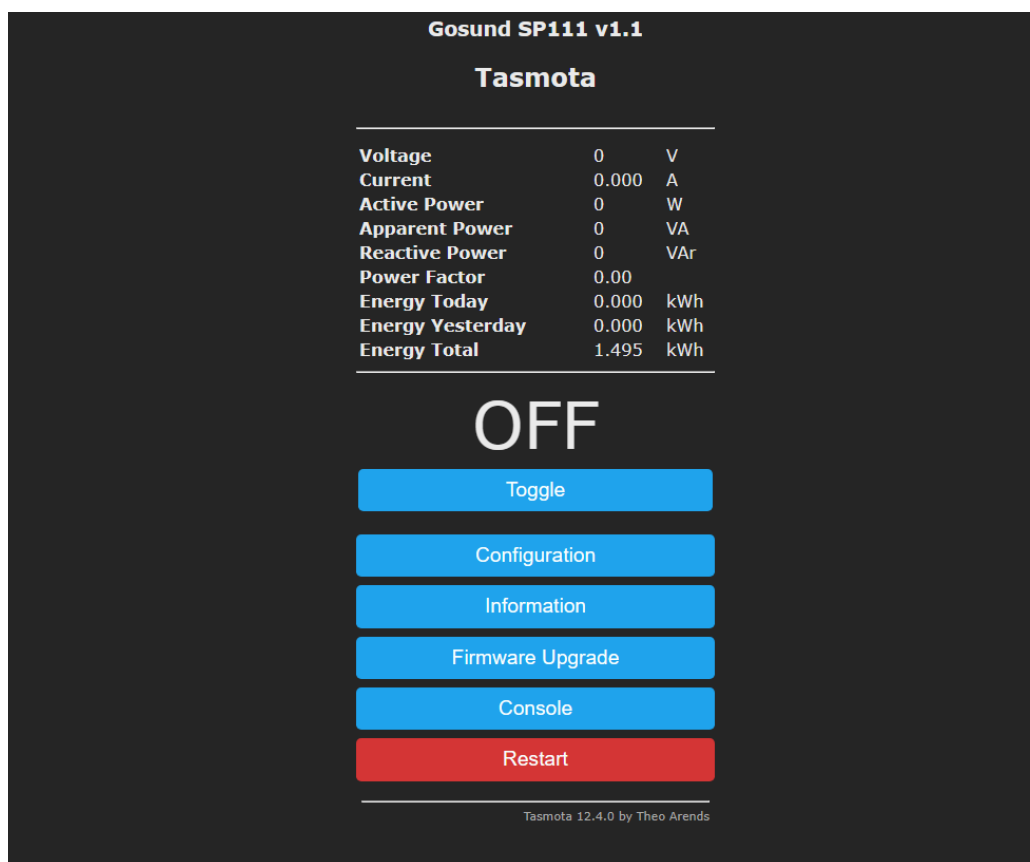


Figura 3.10: Interfața web Tasmota

O caracteristică centrală a interfeței web este capacitatea de a configura setările WiFi. Utilizatorii pot introduce detaliile rețelei lor WiFi, cum ar fi numele (SSID) și parola, permițând dispozitivului să se conecteze la Internet.

Pentru a permite comunicarea cu alte dispozitive și servicii, interfața web Tasmota oferă, de asemenea, opțiuni pentru configurarea MQTT (Message Queuing Telemetry Transport). Acest lucru include setarea adresei brokerului MQTT, a portului și a detaliilor de autentificare, precum și a subiectelor pe care dispozitivul le va utiliza pentru a publica și a primi mesaje.

Tasmota oferă, de asemenea, o gamă largă de informații despre starea și performanța dispozitivului, afișate în timp real pe interfața web.

O categorie importantă de informații este legată de parametrii electrici ai dispozitivului. În funcție de dispozitivul pe care rulează firmware-ul Tasmota, pot fi disponibile date despre tensiunea de alimentare (voltaj), curentul electric și consumul de energie.

3.7 Node-RED

Node-RED este un instrument de dezvoltare vizuală bazat pe fluxuri pentru programarea orientată pe evenimente, conceput inițial de IBM pentru a conecta dispozitive, servicii și API-uri hardware în cadrul Internetului obiectelor (IoT). Este construit pe Node.js, o platformă bazată pe JavaScript ce rulează pe server. [10]

3.7.1 Introducere în Node-RED

Node-RED oferă un editor de fluxuri bazat pe web care permite construirea de aplicații prin reuniunea de noduri de date, servicii și dispozitive într-un mod vizual și intuitiv. Aceste fluxuri pot fi apoi desfășurate pe serverul Node-RED cu un singur clic.

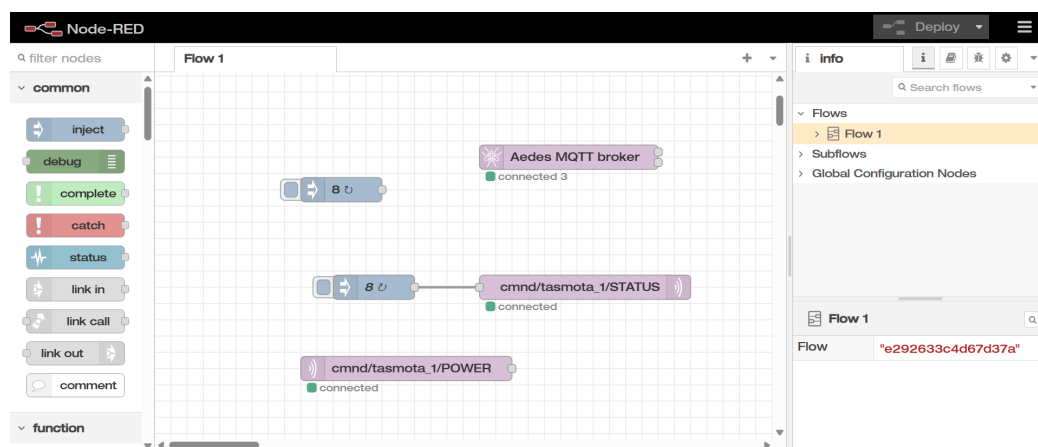


Figura 3.11: Interfata web Node-RED

Editorul de fluxuri permite conectarea de noduri prin trageri și plasări într-un panou de desen, facilitând configurarea lor și setarea de proprietăți. Nodurile pot reprezenta intrări, ieșiri, funcții sau dispozitive conectate. Datorită naturii sale vizuale și modulare, Node-RED facilitează dezvoltarea rapidă a aplicațiilor, încurajând prototiparea și testarea de idei în mod rapid și eficient.

3.7.2 MQTT: Protocolul de mesagerie pentru sistemele de comunicare IoT

MQTT, sau Message Queuing Telemetry Transport, este un protocol de comunicare pe subiect/abonat (topic-based publish/subscribe) care este con-

ceput pentru a fi ușor de implementat, eficient din punct de vedere al lățimii de bandă și al consumului de energie, și care oferă un nivel de serviciu fiabil. A fost conceput în special pentru dispozitivele de tip Internet of Things (IoT), având o amprentă redusă și fiind capabil să funcționeze pe conexiuni de rețea instabile sau cu lățime de bandă redusă.[8]

Comunicarea prin MQTT se bazează pe ideea de "subiecte" (topics) la care dispozitivele se pot abona sau pe care le pot publica. Când un dispozitiv publică un mesaj pe un subiect, toate dispozitivele abonate la acel subiect vor primi mesajul. Acest model de comunicare permite flexibilitate și scalabilitate în sistemele IoT.

În contextul acestei lucrări, MQTT este folosit pentru a permite comunicarea în timp real între aplicație și priza inteligentă, permitând monitorizarea și controlul prizei. MQTT este utilizat în combinație cu Node-RED, care servește ca broker MQTT și permite manipularea și rutarea mesajelor MQTT.

3.7.3 Configurarea și utilizarea MQTT în Node-RED

În această secțiune, vom discuta despre cum se configurează și se utilizează MQTT în Node-RED.

- **Configurarea unui broker MQTT:** În Node-RED, se poate configura un broker MQTT folosind nodul MQTT. Acest nod permite să specificați adresa și portul brokerului MQTT, precum și alte detalii, cum ar fi numele utilizatorului și parola, dacă sunt necesare pentru securitate.
- **Publicarea mesajelor:** pentru a publica un mesaj la un subiect specific, se utilizează nodul MQTT out. Acest nod permite specificarea subiectului și mesajul care urmează să fie publicat. Mesajul poate fi un simplu string sau un obiect JSON, în funcție de cerințe.
- **Abonarea la subiecte:** Pentru a primi mesaje de la un subiect specific, am utilizat nodul MQTT in. Acest nod permite configurarea subiectului la care ne abonăm. Când un mesaj este primit pe acest subiect, nodul va emite un mesaj în fluxul de date.

Capitolul 4

Funcționalitățile Aplicației

4.1 Noțiuni generale

Aplicația „SmartWash” reprezintă un proiect inovator conceput pentru a revoluționa gestionarea spălătoriilor universitare. A fost creată cu scopul de a elimina multe dintre problemele existente în sistemul tradițional de rezervări și utilizare a mașinilor de spălat din cadrul căminelor studențești.

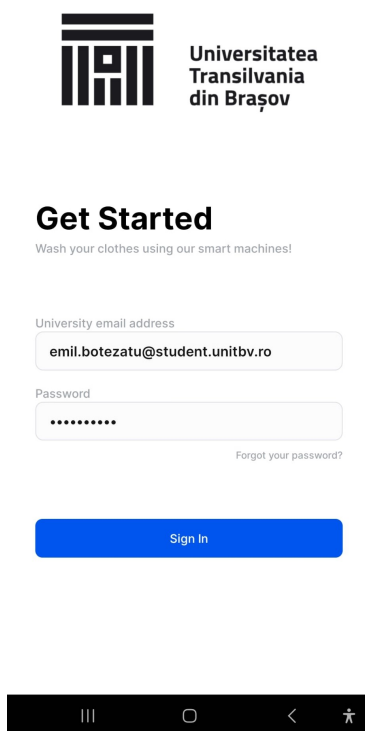
„SmartWash” este construită pe un fundament solid de tehnologii și framework-uri moderne, printre care se numără React Native pentru dezvoltarea interfeței cu utilizatorul, Node.js pentru back-end și SQL pentru gestionarea datelor. Toate acestea lucrează în tandem pentru a asigura o experiență fluidă și eficientă utilizatorilor.

Mai jos sunt prezentate câteva dintre cele mai importante funcționalități:

- notificări în timp real când mașina de spălat este gata
- posibilitatea de a programa rezervări mai devreme în cazul anulării sau încheierii rapide a unei rezervări
- un sistem de chat integrat care permite studenților să comunice între ei pentru a rezolva problemele legate de spălătorie.
- vizualizarea unui istoric de rezervări pentru a permite studenților modificarea lor
- introduce un sistem avansat de monitorizare în timp real a mașinilor de spălat și uscătorilor. Utilizatorii au posibilitatea de a verifica starea fiecărui echipament direct din aplicație, care afișează informații actualizate în mod constant

4.2 Ecranul de autentificare

Pagina de autentificare din aplicația „SmartWash” este simplă și ușor de utilizat, fiind alcătuită din două câmpuri de intrare: email și parolă. Acestea sunt gestionate utilizând biblioteca React Hook Form, o soluție modernă și eficientă pentru gestionarea formularelor în aplicațiile React.



The image shows a mobile application login screen. At the top, there is a logo for 'Universitatea Transilvania din Braşov' consisting of a stylized 'U' and 'T' icon followed by the university's name. Below the logo is the heading 'Get Started' with a subtext 'Wash your clothes using our smart machines!'. The main form contains two input fields: 'University email address' with the value 'emil.botezatu@student.unitbv.ro' and 'Password' with masked characters '*****'. A link 'Forgot your password?' is located below the password field. A blue 'Sign In' button is positioned below the form. At the bottom, there is a black navigation bar with four icons: a list icon, a home icon, a back arrow, and a star icon.

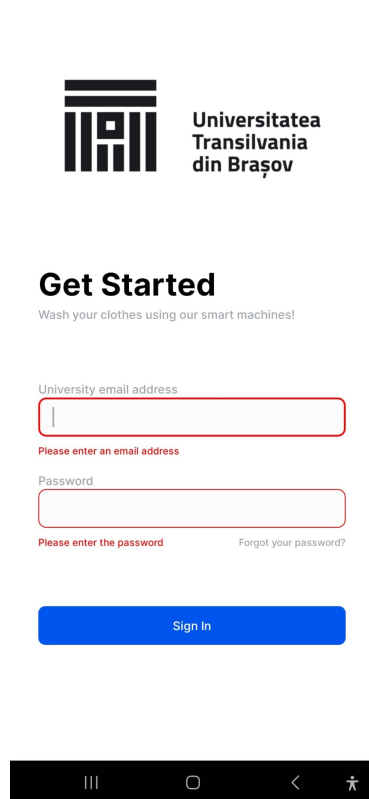
Figura 4.1: Ecranul de autentificare

„SmartWash” este concepută pentru a fi o aplicație dedicată studenților unei anumite instituții de învățământ. Astfel, procesul de înregistrare sau creare a unui cont nu este necesar, deoarece informațiile necesare pentru autentificare (adresa de e-mail a studentului și parola) sunt deja disponibile în baza de date a universității.

Codul de mai jos este un exemplu de cum se utilizează React Hook Form în combinație cu componenta Input pentru a gestiona valoarea și validarea unui câmp de intrare pentru email.

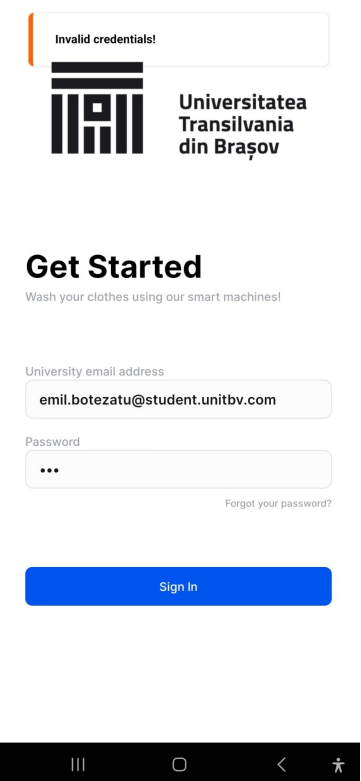
```
<Controller
  control={control}
  rules={{
    required: true,
  }}
  render={({ field: { onChange, onBlur, value } }) => (
    <Input
      fontFamily="InterSemi"
      borderColor={` ${errors.email ? "#FF0000" : "#DFE1E9"} `}
      selectionColor="#A0A7AE"
      onBlur={onBlur}
      onChangeText={onChange}
      value={value}
      focusStyle={
        errors.email && { borderColor: "#FF0000", borderWidth: 2 }
      }
    />
  )}
  name="email"
/>
```

4.2.1 Validările Frontend



The screenshot shows the login interface of the University of Transilvania in Braşov. At the top left is the university's logo and name. Below it, the text "Get Started" is followed by the tagline "Wash your clothes using our smart machines!". The login form consists of two input fields: "University email address" and "Password". Both fields have a red border around them, indicating a validation error. Below the email field, the text "Please enter an email address" is displayed. Below the password field, the text "Please enter the password" is displayed. To the right of the password field, there is a link that says "Forgot your password?". At the bottom of the form is a blue button labeled "Sign In". At the very bottom of the screen is a black navigation bar with four icons: a hamburger menu, a square, a left arrow, and a person icon.

Figura 4.2: Input incorect



The screenshot shows the login interface of the University of Transilvania in Braşov. At the top left is the university's logo and name. Below it, the text "Get Started" is followed by the tagline "Wash your clothes using our smart machines!". The login form consists of two input fields: "University email address" and "Password". The email field is filled with the text "emil.botezatu@student.unitbv.com" and the password field is filled with three dots. Above the email field, there is a message that says "Invalid credentials!". To the right of the password field, there is a link that says "Forgot your password?". At the bottom of the form is a blue button labeled "Sign In". At the very bottom of the screen is a black navigation bar with four icons: a hamburger menu, a square, a left arrow, and a person icon.

Figura 4.3: Parola sau email greşit

În ceea ce priveşte procesul de validare a datelor de intrare, avem două niveluri principale de verificare: validarea de pe partea de frontend şi validarea care vine de pe backend.

Validarea frontend-ului se concentrează pe verificarea integrităţii datelor de bază. În cazul nostru, aceasta înseamnă a verifica dacă utilizatorul a introdus ceva în câmpurile de intrare pentru email şi parolă. Dacă un câmp de intrare este lăsat gol, validarea frontend se va declanşa şi se va genera un mesaj de eroare care indică utilizatorului să completeze câmpul respectiv. Pentru a face acest lucru mai vizibil, bordura câmpului de intrare va deveni roşie.

Pe de altă parte, validarea care vine de pe backend se ocupă de verificarea exactităţii datelor introduse. În acest caz, serverul va verifica dacă emailul şi parola introduse de utilizator se potrivesc cu cele stocate în baza de date. Dacă nu, serverul va genera un mesaj de eroare care indică faptul că emailul sau parola este incorectă. Acest mesaj de eroare va fi afişat utilizatorului

printr-un „toast”, un mesaj pop-up care apare pe ecran pentru o perioadă scurtă de timp.

4.2.2 Fluxul cu JWT (JSON Web Token)

Procesul de autentificare cu JWT (JSON Web Token) este esențial pentru funcționalitatea corectă a aplicației și se întâmplă în următoarele etape:

- **Autentificare inițială:** Când utilizatorul apasă pe butonul de autentificare din aplicație, se trimite o cerere către server cu datele introduse de utilizator (email și parolă).

```
export const useLogin = (
  onSuccess: (data: string) => void,
  onError: (error: unknown) => void
) => {
  return useMutation(
    "login",
    async (input: LoginRequestType) => await login(input),
    {
      onSuccess,
      onError,
    }
  );
};

export const login = async (userCredentials: LoginRequestType) => {
  try {
    const {data} = await axios.post(url, userCredentials);
    console.log(data);
    return data;
  } catch (error) {
    const message = (error as AxiosError)?.response?.data as string;
    console.log(message)
    throw new Error(message)
  }
};
```

- **Verificarea emailului:** Serverul interceptează aceste date și începe prin a verifica dacă adresa de email introdusă de utilizator este înregistrată în baza de date. Dacă nu, serverul trimite înapoi un mesaj de eroare indicând că autentificarea a eșuat.

```

const result: student[] =
await prisma.$queryRaw`SELECT * FROM student WHERE email = ${email}`;
if (result.length === 0) {
    res.status(401).json("Invalid credentials!");
    return;
}

```

- **Verificarea parolei:** Dacă adresa de email este validă, serverul continuă prin a compara parola introdusă de utilizator cu cea stocată în baza de date. Acest proces implică decriptarea parolei hash-uite din baza de date folosind bcrypt¹ pentru a se asigura că parolele corespund.
- **Generarea token-ului JWT:** Dacă verificările de email și parolă sunt trecute cu succes, serverul generează un token JWT. Acest token conține informații despre utilizator care vor fi folosite pentru a-l identifica în toate cererile viitoare.

```

if (await bcrypt.compare(password, result[0].password)) {
    const token = sign(
    {
        user_id: result[0].id,
        first_name: result[0].first_name,
        last_name: result[0].last_name,
        expo_token: result[0].notification_token,
    },
    process.env.TOKEN_KEY as string
    );
    res.send(JSON.stringify(token));
} else {
    console.log("Password does not match");
    res.status(400).json("Invalid credentials!");
}

```

- **Decodarea token-ului:** Pe partea de frontend, token-ul primit este decodat printr-un hook personalizat. Acest hook extrage datele utilizatorului din token și le stochează în Zustand store. Aceasta permite accesul la datele utilizatorului în întreaga aplicație, fără a fi necesar să se facă cereri suplimentare la server.

¹Bcrypt este o bibliotecă de codificare a parolelor concepută pentru a asigura securitatea informațiilor de autentificare ale utilizatorilor. Bcrypt utilizează algoritmul Blowfish pentru criptare

- **Stocarea token-ului:** În final, token-ul JWT este stocat în Async Storage² pe dispozitivul utilizatorului. Acest lucru permite ca token-ul să fie accesat și utilizat pentru autentificare de fiecare dată când utilizatorul deschide aplicația, fără a fi necesar să se autentifice de fiecare dată.

```
const getAuthToken = async () => {
  try {
    const token: string | null =
      await AsyncStorage.getItem("token");

    if (token) {
      const decodedToken: {
        user_id: number;
        first_name: string;
        last_name: string;
        expo_token: string;
      } = jwt_decode(token);
      setToken(token);
      setFirstName(decodedToken.first_name);
      setId(decodedToken.user_id);
      setLastName(decodedToken.last_name);
      setExpoToken(decodedToken.expo_token);
    }
  } catch (error) {
    console.log(error);
  }
};
```

4.3 Conectarea prizei Gosund SP111 la aplicația SmartWash

4.3.1 Procesul de Tasmotizare a Prizei Inteligente

Procesul de tasmotizare implică instalarea firmware-ului Tasmota pe Priza Inteligentă Gosund SP111. Aceasta transformă priza într-un dispozitiv compatibil cu mult mai multe platforme și protocoale, inclusiv MQTT, oferindu-i utilizatorului un control sporit asupra acestuia.

²Async Storage este o bibliotecă persistentă, necriptată și asincronă de stocare locală pentru aplicațiile React Native. Este folosită pentru a stoca date între sesiunile de utilizare ale aplicației

Pași pentru procesul de tasmotizare sunt următorii:

- **Obținerea firmware-ului Tasmota:** Firmware-ul Tasmota poate fi descărcat de pe site-ul oficial. Există diverse versiuni disponibile, iar selecția depinde de nevoile și cerințele utilizatorului.
- **Pregătirea prizei pentru flash-uire:** Conectarea la interfața de programare serială a cipului ESP se realizează prin conectarea pinilor TX și RX ai convertorului nostru serial-la-USB la pinii RX și TX ai ESP și alimentarea cipului cu pinii de 3.3V și GND.

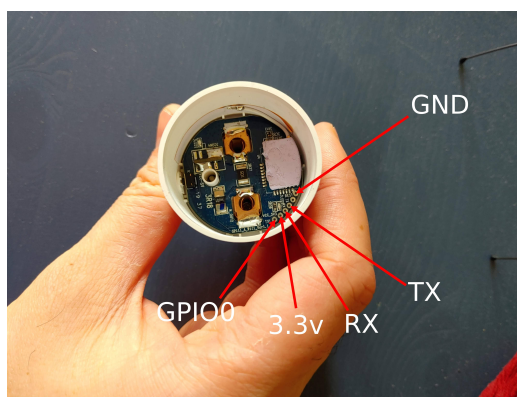


Figura 4.4: Procesul de conectare la pinii prizei [5]

- **Flash-uirea prizei cu firmware-ul Tasmota:** Aceasta este etapa în care firmware-ul Tasmota este efectiv instalat pe priză.

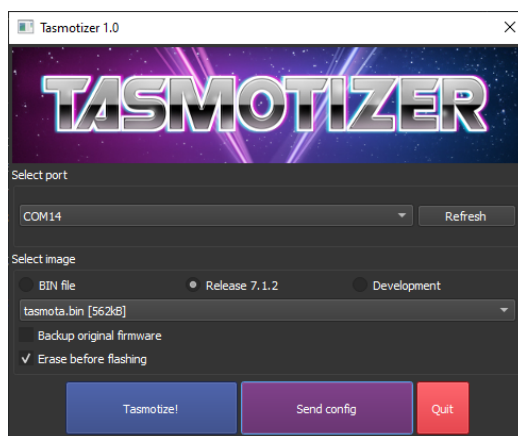


Figura 4.5: Procesul în curs de instalare a software-ului pe priză

4.3.2 Configurarea setărilor de Wi-Fi și MQTT

După finalizarea procesului de tasmotizare, următorul pas este configurarea setărilor pentru Wi-Fi și MQTT (Message Queue Telemetry Transport).

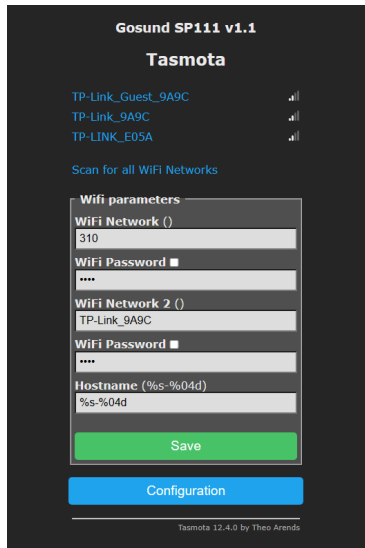


Figura 4.6: Configurare WIFI

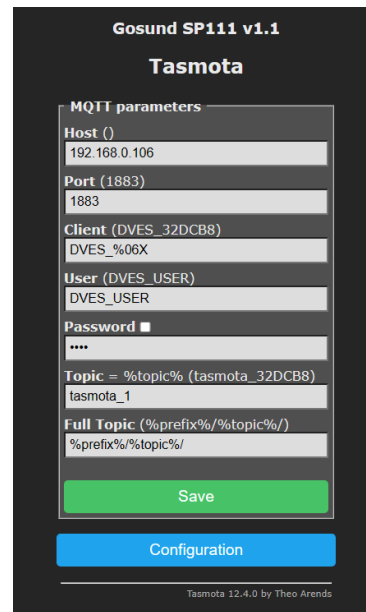


Figura 4.7: Configurare MQTT

Pentru configurarea WIFI, sunt necesare următoarele:

- *Numele rețelei WIFI* (SSID-ul)
- *Parola*

Pentru configurarea MQTT, sunt necesare următoarele informații:

- *Adresa serverului MQTT* (de exemplu, adresa IP a Node-RED dacă acesta rulează un broker MQTT)
- *Portul MQTT* (în cazul acesta 1883)
- *Numele utilizatorului și parola*
- *Topic-ul MQTT* (acesta este un identificator unic care permite identificarea mesajelor MQTT asociate cu priza inteligentă)

4.3.3 Cum se stabilește conexiunea între priza inteligentă și Node-RED

Node-RED este o unealtă care permite crearea de fluxuri pentru a conecta dispozitive hardware și API-uri. În acest caz, vom folosi Node-RED pentru a primi și a trimite mesaje către și de la priza inteligentă prin MQTT.

Pasul 1: Instalarea broker-ului MQTT în Node-RED

Înainte de a putea primi mesaje de la priza inteligentă, Node-RED trebuie să fie echipat cu un broker MQTT. Acest lucru se poate realiza prin instalarea nodului *node-red-contrib-aedes*. Acest nod va asculta mesaje de la priza inteligentă.

Pasul 2: Configurarea broker-ului MQTT

După instalarea nodului, acesta trebuie configurat cu aceleași setări care s-au folosit pe priza inteligentă. Aceasta include adresa serverului, portul, numele de utilizator și parola, dacă este cazul.

Pasul 3: Crearea unui flux în Node-RED pentru a primi mesaje

Cu broker-ul MQTT configurat, acum putem crea un flux în Node-RED care va asculta mesajele de la priza inteligentă. Acest lucru se face prin adăugarea unui nod de intrare MQTT la fluxul nostru și setarea acestuia pentru a asculta același topic care a fost setat pe priza inteligentă.

Acum, de fiecare dată când priza inteligentă trimite un mesaj MQTT, acesta va fi primit de Node-RED. Același proces poate fi folosit și pentru a trimite comenzi către priza inteligentă prin crearea unui nod de ieșire MQTT în fluxul creat.

Observăm în acest flux următoarele:

- Fluxul MQTT de intrare de pe priză va primi nodul *inject* cu mesajul 8, însemnând codul pentru consumul de curent, astfel nodul de intrare ascultă răspunsul dat, de la topicul *tasmota_1/STATUS8*
- Fluxul MQTT de ieșire va publica mesajul trimis de la server-ul Node.js către topicul *tasmota_1/POWER*. Acesta va fi un string de tipul *off* sau *on* ce va fi folosit pentru pornirea și stingerea prizei înainte și respectiv după utilizare

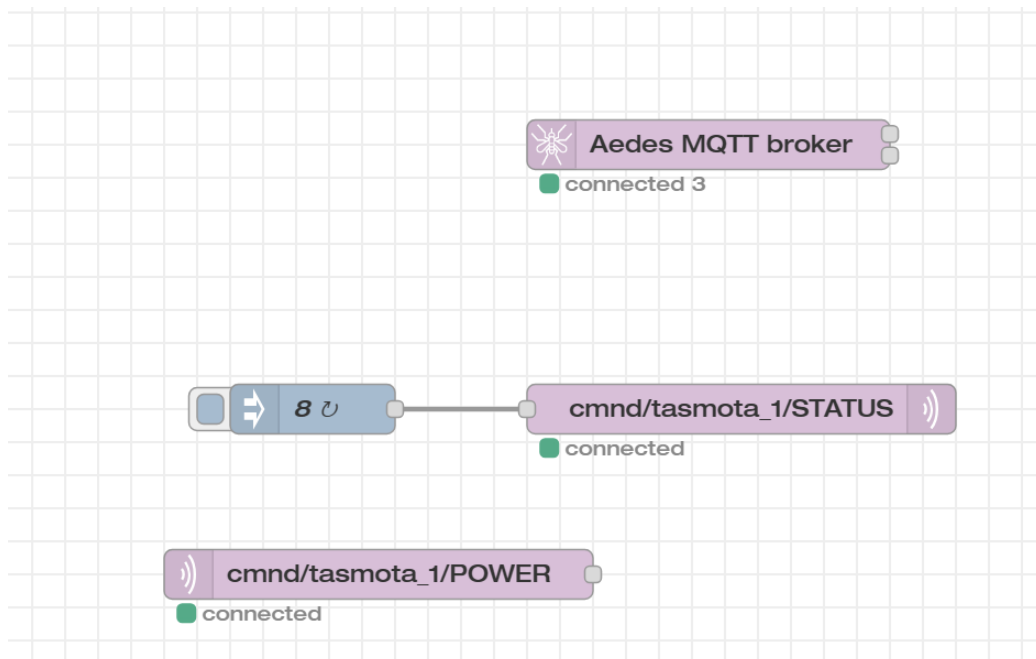


Figura 4.8: Fluxul Node-RED pentru primirea și trimiterea de mesaje de pe priză

4.3.4 Configurarea și utilizarea MQTT în Node-RED și Node.js

Configurarea și utilizarea MQTT în Node-RED și Node.js presupune câțiva pași:

Pasul 1: Instalarea pachetului MQTT

În cazul Node.js, acest lucru se face folosind managerul de pachete npm, cu comanda `npm install mqtt`. În cazul Node-RED, un nod MQTT este de obicei deja inclus în instalarea standard.

Pasul 2: Crearea unui client MQTT

În Node.js, acest lucru se face utilizând funcția `mqtt.connect()`, care primește ca argument URL-ul brokerului MQTT.

Pasul 3: Publicarea și abonarea la mesaje

În Node.js, metodele `client.publish()` și `client.subscribe()` pot fi folosite pentru a publica și respectiv abona la mesaje.

```

const connectUrl = `mqtt://${process.env.mqtt_host}:${process.env.mqtt_port}`;
const clientId = `mqtt_${Math.random().toString(16).slice(3)}`;
const client = mqtt.connect(connectUrl, {
  clientId,
  clean: true,
  connectTimeout: 4000,
  reconnectPeriod: 1000,
  username: process.env.mqtt_username,
  password: process.env.mqtt_password,
});

export const powerSmartPlug = (topic: string, payload: string, client: any) => {
  client.publish(topic, payload, (error: any) => {
    console.log(payload);
    if (error) {
      console.error(error);
    }
  });
};

```

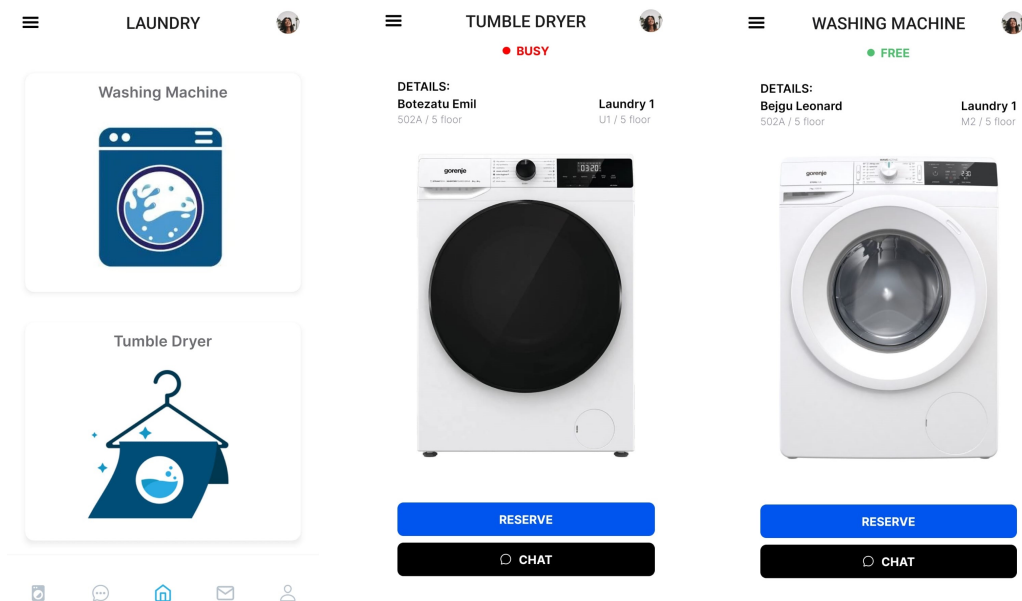
4.4 Sistemul de monitorizare în timp real a spălătoriei

Sistemul de monitorizare în timp real a spălătoriei permite utilizatorilor să vizualizeze în timp real starea fiecărei mașini de spălat sau uscător disponibil în spălătorie.

Datele sunt preluate din baza de date și sunt actualizate în timp real prin intermediul unui proces de refetch, care invalidează cache-ul și reîncarcă informațiile.

Pentru fiecare mașină, se poate vedea starea actuală și există 3 stări explicate în detaliu mai jos:

- **FREE**: atunci când mașina este liberă și poate să fie folosită la momentul respectiv. Aceasta se identifică prin mesajul *FREE* și codul de culoare verde
- **BUSY**: atunci când mașina este ocupată la momentul respectiv. Aceasta se identifică prin mesajul *BUSY* și codul de culoare roșu
- **NOT OPENED**: atunci când ciclul de spălare este încheiat, dar persoana respectivă nu a eliberat mașina de haine. Aceasta se identifică prin mesajul *NOT OPENED* și codul de culoare galben



Utilizatorii au și opțiunea de a rezerva o mașină direct din această interfață. Atunci când apasă pe butonul de rezervare, sunt redirecționați către ecranul de rezervare, unde campurile sunt deja umplute cu informațiile corespunzătoare mașinii de spălat selectate.

De asemenea, există și un buton de chat, care navighează către ecranul de chat al utilizatorului. În cazul în care nu există deja o conversație cu utilizatorul care utilizează mașina de spălat, apăsarea butonului de chat va crea o nouă conversație.

Pentru navigarea între ecrane s-a folosit React Navigation:

```
navigation.navigate("ChatStack");
```

4.5 Sistemul de rezervări

4.5.1 Crearea unei rezervări

Procesul de creare a unei rezervări în aplicația „SmartWash” este unul intuitiv și facil. Ecranul de rezervare este alcătuit din mai multe componente, care permit utilizatorilor să își personalizeze rezervarea în funcție de preferințele lor.

Mai jos este prezentat ecranul de rezervări în mai multe stări:

The image displays three sequential screenshots of the 'Reserve' screen in the SmartWash application, showing the process of creating a reservation. Each screen has a top navigation bar with 'Reserve', 'Processing', and 'History' tabs, and a bottom navigation bar with icons for home, messages, and profile.

- Left Screenshot (Initial State):** Shows the 'Reserve' tab selected. There are two buttons at the top: 'Wash' (highlighted with a blue border) and 'Dry'. Below these are five input fields: 'Select Laundry' (dropdown menu), 'Pick a washing machine' (dropdown menu), 'Choose an available date' (text field), 'Pick an available time slot' (dropdown menu), and 'Choose the start hour and end hour' (text field). A large blue 'START' button is at the bottom.
- Middle Screenshot (Intermediate State):** Similar to the first, but the 'Choose an available date' field now shows 'Mon Jun 12 2023'. Below this field, a section titled 'Available Laundries' is visible, showing 'Laundry 1 / floor 5'.
- Right Screenshot (Final State):** Similar to the middle one, but the 'Available Laundries' section is replaced by 'Available Hours', which lists two time slots: '07:00 - 15:00' and '18:00 - 00:00'.

Există cinci input-uri pe care utilizatorii trebuie să le completeze pentru a crea o rezervare:

- **Spălătoria:** Aici utilizatorul poate selecta din lista de spălătorii disponibile în sistem
- **Tipul de echipament:** Utilizatorii au la dispoziție două butoane, unul pentru mașina de spălat și altul pentru uscător. În funcție de opțiunea selectată, lista de echipamente disponibile va fi actualizată corespunzător.
- **Data:** Utilizatorul poate selecta data în care dorește să facă rezervarea.
- **Intervalele orare disponibile:** După ce a fost selectată data, sunt afișate intervalele orare disponibile pentru rezervare în acea zi.

- **Ora de start și finish:** După ce utilizatorul a selectat un interval orar, acesta poate specifica ora exactă de start și de finish a rezervării într-un modal.

Crearea unei rezervări în aplicația „SmartWash” presupune și respectarea unor condiții specifice care țin de modul în care este utilizat sistemul de rezervări. Acestea au fost stabilite pentru a asigura o utilizare eficientă și justă a echipamentelor de către toți utilizatorii.

Atunci când un utilizator alege ora de start și finish a rezervării în modal, sunt verificate următoarele condiții:

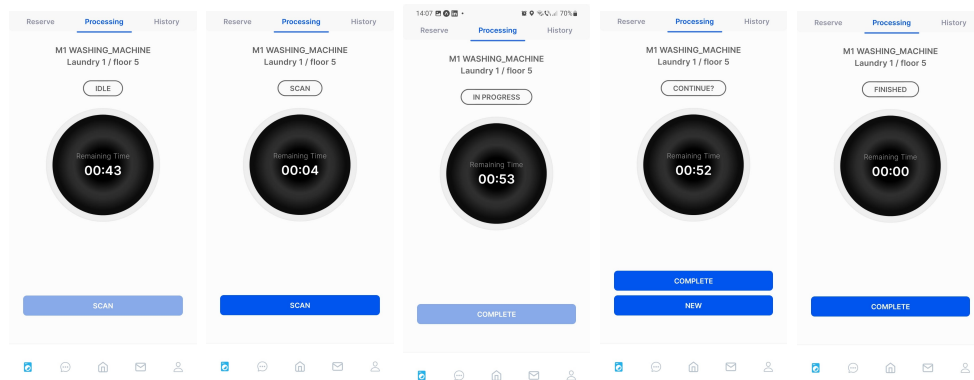
- **Durata rezervării:** Durata unei rezervări nu poate depăși 3 ore. Acesta este un limitator pus în loc pentru a preveni monopolizarea echipamentelor de către un singur utilizator pentru perioade lungi de timp.
- **Timpul de finalizare a rezervării:** Ora de încheiere a rezervării trebuie să fie mai mare decât ora de început. Acest lucru asigură că durata rezervării este pozitivă și că utilizatorul își rezervă o fereastră de timp utilă.
- **Intervalul orar disponibil:** Timpul selectat pentru rezervare trebuie să fie în cadrul intervalului orar disponibil ales. Aceasta este o condiție esențială pentru a evita suprapunerea rezervărilor și a asigura că fiecare utilizator își poate folosi timpul alocat pentru rezervare.

Figura 4.9: Exemplu de validare ce verifică durata rezervării

Figura 4.10: Exemplu de validare ce verifică orele rezervării

4.5.2 Procesarea rezervărilor și stările mașinii

În cadrul aplicației „SmartWash”, fiecare mașină de spălat poate avea unul dintre cele cinci stări diferite, fiecare reflectând starea curentă a rezervării și a mașinii:



- **Idle (Inactiv):** Această stare indică faptul că rezervarea este programată, dar nu a început încă. Mașina de spălat este liberă și așteaptă ca următoarea rezervare să înceapă.
- **Scan (Scanare):** În acest stadiu, utilizatorului i se permite să scaneze codul QR afișat pe mașina de spălat pentru a debloca aceasta și a începe procesul de spălare.
- **In Progress (În Desfășurare):** Mașina de spălat este în acest moment în funcțiune și procesul de spălare este în desfășurare. Această stare indică faptul că rezervarea este în curs.
- **Continue? (Continuare?):** Această stare este un punct de control care permite utilizatorului să decidă dacă dorește să încheie rezervarea mai devreme sau dacă dorește să continue să utilizeze mașina de spălat pentru restul perioadei de rezervare. Această opțiune poate fi folosită oricât de multe ori în cadrul perioadei de rezervare.
- **Finished (Terminat):** În această stare, rezervarea s-a încheiat. Utilizatorul trebuie să confirme că hainele au fost scoase din mașină apăsând pe butonul corespunzător în aplicație, iar rezervarea este încheiată cu succes.

4.5.3 Prezentarea ecranului principal și opțiunile utilizatorului

Ecranul principal al aplicației „SmartWash” este centrul activităților utilizatorului. El prezintă în timp real datele corespunzătoare spălătoriei și mașinii de spălat, precum și detalii despre starea curentă a rezervării. Iată ce informații sunt afișate pentru fiecare stare:

- **Inactiv:** Se afișează timpul rămas până la începerea rezervării. Acest timp de așteptare ajută utilizatorii să își planifice timpul în mod eficient înainte de începerea procesului de spălare. Butonul este dezactivat, indicând faptul că rezervarea nu a început încă și utilizatorul nu poate folosi mașina de spălat.
- **Scanare:** Aici, se afișează un contor invers care indică timpul rămas pentru scanarea codului QR și deblocarea mașinii de spălat. Utilizatorul are un termen limită de 10 minute pentru a începe procesul de spălare. Butonul devine activ, permițându-le utilizatorilor să deblocheze mașina prin scanarea codului QR.
- **În Desfășurare:** În această stare, utilizatorul poate vedea timpul rămas până la finalul perioadei de rezervare. Acest lucru îi permite să monitorizeze procesul de spălare și să știe când va fi gata. Butonul este inactiv, și nu se poate încheia rezervarea.
- **Continuare?:** În acest stadiu, se afișează *00:00*, indicând faptul că rezervarea s-a încheiat și nu mai este timp rămas. Pe ecran apar 2 butoane care permit utilizatorului să folosească mașina în continuare, scanând încă o dată codul QR sau să încheie rezervarea mai devreme.
- **Terminat:** În acest stadiu, se afișează *00:00*, indicând faptul că rezervarea s-a încheiat și nu mai este timp rămas. Butonul devine activ, iar utilizatorul poate încheia rezervarea cu succes, în momentul în care acesta a eliberat mașina de haine.

4.5.4 Activarea mașinilor de spălat cu ajutorul codurilor QR

Utilizarea codurilor QR în aplicație este menită să asigure prezența efectivă a utilizatorului în momentul începerii utilizării mașinii. Acest sistem previne astfel rezervările false sau neutilizate, care ar putea cauza ineficiențe și blocaje în utilizarea resurselor.

Iată cum funcționează acest proces:

Pasul 1: **Navigarea către ecranul principal**

Utilizatorul navighează către ecranul principal al aplicației unde sunt afișate detalii despre rezervarea curentă. În acest moment, mașina de spălat este în starea „Scanare”, indicând faptul că este pregătită pentru a fi utilizată.

Pasul 2: **Scanarea codului QR**

Utilizatorul apasă pe butonul de scanare prezent în aplicație. Acest lucru deschide camera dispozitivului pentru a citi codul QR afișat pe mașina de spălat.

Pasul 3: **Transmiterea și validarea datelor către backend**

Odată ce codul QR este scanat cu succes, informațiile sunt trimise către server prin intermediul unei cereri HTTP. Serverul validează codul QR și, dacă este valid, starea mașinii de spălat este schimbată în „În Desfășurare”.

Pasul 4: **Pornirea prizei smart**

În același timp, odată ce codul este valid, funcția care pornește priza smart, este apelată cu valoarea *on* și topicul corespunzător numărului prizei. Din acest moment consumul de curent este monitorizat.

```
powerSmartPlug("cmd/tasmota_1/POWER", "on", client);  
res.status(200).json("Washing machine unlocked successfully!");  
getPowerStatus(expoPushToken, client, "stat+/STATUS8", user_id, device[0].type);
```

Prevenirea rezervărilor *fantomă*:

Dacă o rezervare nu este confirmată prin scanarea codului QR într-un anumit interval de timp (în acest caz, 10 minute), un proces automat job³ care rulează pe serverul Node.js va anula rezervarea. Acest mecanism asigură o utilizare optimă a mașinilor de spălat și uscător, permițând accesul doar utilizatorilor activi și prezenți.

³ Aceste job-uri sunt sarcini automate care sunt programate să ruleze la anumite intervale de timp.

```

cron.schedule("* * * * *", async () => {
  try {
    const reservations = await prisma.$queryRaw<
reservation[]
>`select reservation.* from reservation
inner join washing_device on washing_device.id = reservation.washing_device_id
and reservation.start_hour::timestamp <
(NOW() AT TIME ZONE 'Europe/Bucharest' - INTERVAL '10' MINUTE)
and reservation.end_hour > NOW() AT TIME ZONE 'Europe/Bucharest'
and washing_device.status = true
`;
    if (reservations.length !== 0) {
      reservations.map(async (reservation) => {
        await prisma.$queryRaw`
delete from reservation where reservation.id = ${reservation.id}`;
        console.log("Found expired ", reservations);
      });
    } else {
      console.log("Every thing up to date!")
    }
  } catch (error) {
    console.log(error);
  }
});

```

Pe scurt, utilizarea codurilor QR pentru a activa mașinile de spălat ajută la asigurarea următoarelor:

- forțează utilizatorul să fie prezent la începutul perioadei de rezervare, ceea ce previne rezervările „fantomă” care ar putea bloca mașina de spălat pentru alți utilizatori.
- prin eliminarea rezervărilor neutilizate, acest sistem permite o rotație mai eficientă a utilizatorilor și o utilizare mai bună a mașinilor de spălat.
- limitează posibilitatea ca utilizatorii să rezerve și să blocheze mașinile de spălat fără a le folosi de fapt, asigurându-se că toți utilizatorii au un acces egal la resurse.

4.5.5 Comunicarea consumului de curent de către dispozitive

Comunicarea consumului de curent de către dispozitive este un aspect esențial în cadrul acestei aplicații.

Priza inteligentă trimite datele legate de consumul de curent către sistemul de monitorizare. Acest lucru se face prin intermediul unui obiect JSON care include toate informațiile și statisticile necesare:

```
{
  "StatusSNS":{
    "Time":"2023-06-13T18:58:05",
    "ENERGY":{
      "TotalStartTime":"2023-02-21T20:41:02",
      "Total":1.495,
      "Yesterday":0.000,
      "Today":0.000,
      "Power":0,
      "ApparentPower":0,
      "ReactivePower":0,
      "Factor":0.00,
      "Voltage":296,
      "Current":0.000
    }
  }
}
```

Aceste informații sunt transmise către Node-RED prin intermediul unei conexiuni MQTT. Pe Node-RED, există un nod Inject care trimite periodic un cod specific (în acest caz, codul 8) către un nod MQTT In, numit „tas-mota_1/STATUS”. Acesta reprezintă comanda pentru priza inteligentă de a trimite informațiile curente despre consumul de curent.

Într-o etapă ulterioară, serverul Node.js se abonează la acest topic MQTT pentru a primi și a procesa datele despre consumul de curent.

Aceste informații sunt folosite în algoritmul de decizie (care va fi discutat în continuare) pentru a gestiona eficient utilizarea mașinii de spălat și a uscătorului.

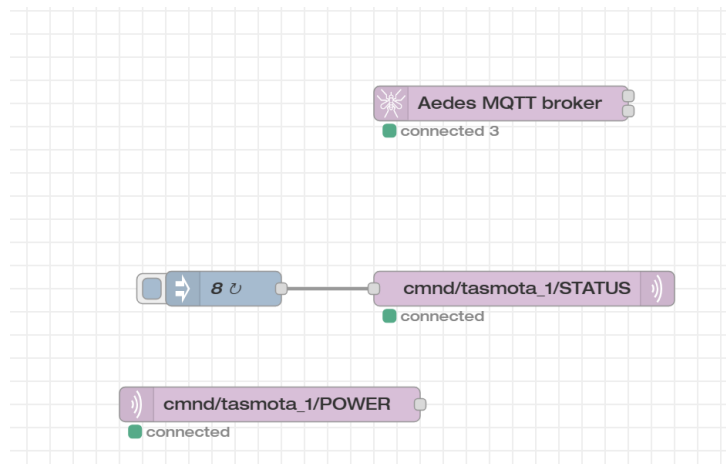


Figura 4.11: Fluxul Node-RED pentru primirea și trimiterea de mesaje de pe priză

4.5.6 Algoritmul de decizie pentru oprire

Algoritmul de decizie pentru oprirea dispozitivelor în aplicația este deosebit de important pentru eficiența și funcționalitatea sistemului. Acesta este bazat pe monitorizarea consumului de curent și luarea unei decizii în timp real.

Procesul este următorul:

1. **Monitorizarea consumului de curent:** Sistemul folosește datele de *Apparent Power* (Puterea Aparentă) primite de la priza inteligentă pentru a monitoriza consumul de curent al mașinii de spălat sau uscătorului. Acest lucru se face prin calcularea mediei consumului de curent pe un interval de un minut.
2. **Verificarea consumului:** Dacă media consumului de curent scade sub un anumit prag (threshold) pentru o perioadă de timp, algoritmul consideră că ciclul de funcționare al mașinii de spălat sau uscătorului s-a încheiat.
3. **Actualizare status și oprire priză:** Odată ce este identificată încheierea ciclului, algoritmul trimite o comandă către priza inteligentă pentru a opri alimentarea cu energie a dispozitivului. În același timp, face un update în baza de date pentru a marca status-ul mașinii ca fiind liberă.
4. **Notificări către utilizator:** După încheierea procesului, sistemul trimite notificări către utilizatorul care a folosit mașina de spălat sau

uscătorul pentru a-l informa că procesul este gata. Acesta se face prin intermediul unui sistem de notificări în aplicație.

5. **Crearea unei intrări în tabelul de notificări:** De asemenea, se creează o intrare în tabelul de notificări din baza de date, care va fi vizibilă pentru utilizator în Inbox-ul din aplicație. Acest lucru asigură o evidență a activităților și notificărilor pentru fiecare utilizator.

Aici este prezentată o secțiune din algoritm:

```
powerData.push({ powerConsumption, timestamp });

const thresholdData = powerData.filter(
  ({ timestamp }) =>
    timestamp.valueOf() >= new Date().valueOf() - 60 * 1000
);

const averagePowerConsumption =
  thresholdData.reduce(
    (sum, { powerConsumption }) => sum + powerConsumption,
    0
  ) / thresholdData.length;

if (averagePowerConsumption < 30) {
  if (thresholdStartTime === null) {
    thresholdStartTime = timestamp;
  }

  if (timestamp.valueOf() -
    thresholdStartTime.valueOf() >= 60 * 1000) {
    powerSmartPlug(`cmnd/tasmota_${smart_plug_id}/POWER`, "off", client);
    updateWashingMachineStatus(parseInt(smart_plug_id));
    const message =
      device === "WASHING_MACHINE"
        ? "Your machine has finished washing!"
        : "Your dryer has finished!";

    sendNotification(expoPushToken, message);
    const notification: Omit<notifications, "id"> = {
      student_id: user_id,
      title: message,
      timestamp: new Date(),
    }
  }
}
```



```
                subtitle: null,  
            };  
            createNotification(notification);  
            client.end();  
        }  
    } else {  
        thresholdStartTime = null;  
    }  
}
```

4.5.7 Ecranul istoricului de rezervări

Istoricul rezervărilor este organizat pe date, oferind o prezentare clară și structurată a tuturor rezervărilor viitoare. Acest lucru facilitează utilizatorilor planificarea și gestionarea eficientă a timpului și a rezervărilor lor, precum și vizualizarea rapidă a tuturor rezervărilor programate.

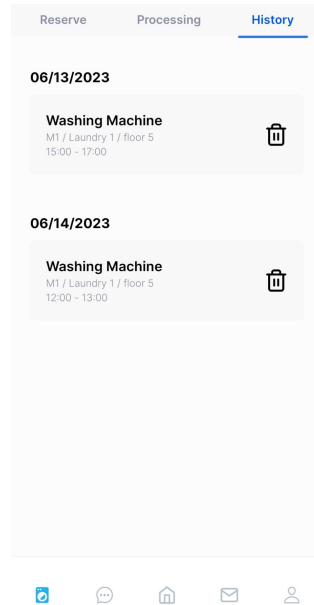


Figura 4.12: Vizualizarea istoricului rezervărilor din tab-ul de *History*

Butonul de ștergere (reprezentat prin pictograma unui coș de gunoi) este o funcție esențială în istoricul rezervărilor. Acesta le permite utilizatorilor să șteargă rezervările pe care nu le mai doresc sau care au devenit inutile.

Trebuie să se sublinieze faptul că ștergerea unei rezervări este definitivă și nu poate fi anulată. Prin urmare, utilizatorii sunt sfătuiți să folosească această opțiune cu grijă.

4.6 Sistemul de notificari

4.6.1 Prezentarea ecranului de inbox

Ecranul de *Inbox* este locul în care utilizatorii pot vizualiza toate notificările primite. Acestea sunt organizate într-un format ușor de citit și de înțeles, de la cele mai recente la cele mai vechi, permițând utilizatorilor să rămână la curent cu cele mai importante actualizări și informații legate de utilizarea mașinilor de spălat.

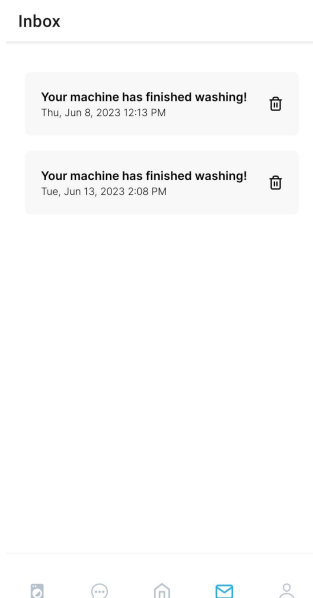


Figura 4.13: Vizualizarea notificărilor primite din ecranul *Inbox*

Fiecare notificare prezentată în *Inbox* include informații esențiale, cum ar fi ora și data la care a fost trimisă, precum și mesajul propriu-zis al notificării. Acest lucru asigură că utilizatorii primesc în timp real toate detaliile relevante, cum ar fi confirmarea rezervării, actualizările de stare a mașinii de spălat, notificările de terminare a ciclului de spălare și altele asemenea.

În plus, alături de fiecare notificare se găsește o pictogramă de coș de gunoi. Prin apăsarea acestei iconițe, utilizatorii pot șterge oricare notificare pe care o consideră inutilă sau irelevantă. Acest lucru îi ajută să mențină „inbox”-ul organizat și să își gestioneze eficient notificările, păstrând doar cele care sunt cu adevărat importante.

4.6.2 Trimiterea de notificări folosind Expo

Trimiterea de notificări este gestionată folosind Expo, un instrument de dezvoltare open-source pentru aplicațiile mobile React Native.

Pe partea de frontend, implică instalarea pachetului **expo-notifications**. Acest pachet oferă un set robust de instrumente pentru gestionarea și primirea notificărilor push.[4]

La autentificare, se apelează funcția **registerForPushNotifications()**. Această funcție are două responsabilități principale. În primul rând, solicită permisiunea utilizatorului pentru a primi notificări push. Acest lucru este necesar pentru a respecta reglementările privind confidențialitatea și pentru a asigura o experiență de utilizare optimă.[4]

Dacă utilizatorul acceptă să primească notificări, funcția va genera apoi un așa-numit „Expo Push Token”. Acest token unic este asignat dispozitivului specific al utilizatorului și va fi folosit pentru a trimite notificările push către acel dispozitiv.

După instalarea pachetului, pe frontend este implementat un listener de notificări. Acest listener va asculta în mod continuu notificările primite, permițând aplicației să răspundă corespunzător când se primește o notificare.

```
useEffect(() => {
  responseListener.current =
    Notifications.addNotificationResponseReceivedListener((response) => {
      if (response) {
        const receivedNotification =
          response.notification.request.content.data;

        if receivedNotification.id === id) {
          navigation.navigate("WashStack");
        }
        refetchNotifications();
      }
    });
  return () => {
    Notifications.removeNotificationSubscription(
      notificationListener.current
    );
    Notifications.removeNotificationSubscription(responseListener.current);
  };
});
```

```
    };
  }, [expoToken]));
```

Pe partea de backend, se folosește **expo-server-sdk**, o bibliotecă care facilitează trimiterea notificărilor push de pe server la dispozitivele utilizatorilor. Există două funcții utilitare în special care sunt folosite în acest scop. Prima funcție permite trimiterea unei notificări la o singură persoană, în timp ce cea de-a doua funcție permite trimiterea unei notificări la un grup de persoane, acceptând un vector de destinatari ca argument.

```
export const sendNotification = (
  expoPushToken: string | undefined,
  message: string
) => {
  const expo = new Expo({ accessToken: process.env.ACCESS_TOKEN });

  if (!Expo.isExpoPushToken(expoPushToken)) {
    console.error(
      `Push token ${expoPushToken} is not a valid Expo push token`
    );
    return;
  }

  const messageSend = {
    to: expoPushToken,
    body: message
  }

  let chunks = expo.chunkPushNotifications([messageSend]);
  let tickets = [];
  (async () => {
    for (let chunk of chunks) {
      try {
        let ticketChunk =
          await expo.sendPushNotificationsAsync(chunk);
        console.log(ticketChunk);
        tickets.push(...ticketChunk);
      } catch (error) {
        console.error(error);
      }
    }
  })();
};
```

Când o notificare este primită, aceasta poate fi vizualizată în bara de notificări a dispozitivului mobil. Apăsând pe notificare, utilizatorul poate deschide aplicația „SmartWash”, fiind astfel în mod direct redirecționat către conținutul relevant sau informațiile legate de notificare.

4.7 Chat-ul integrat

4.7.1 Prezentarea ecranului de conversații si chat

În aplicația „SmartWash”, ecranul de conversații active oferă un loc centralizat unde utilizatorii pot vedea și gestiona toate discuțiile lor în curs. Aceasta afișează o listă cu persoanele cu care utilizatorul a interacționat anterior, facilitând reluarea conversațiilor sau începerea unor noi discuții.

Mai jos este prezentată o posibilă discuție între 2 studenți pe tema unei probleme:

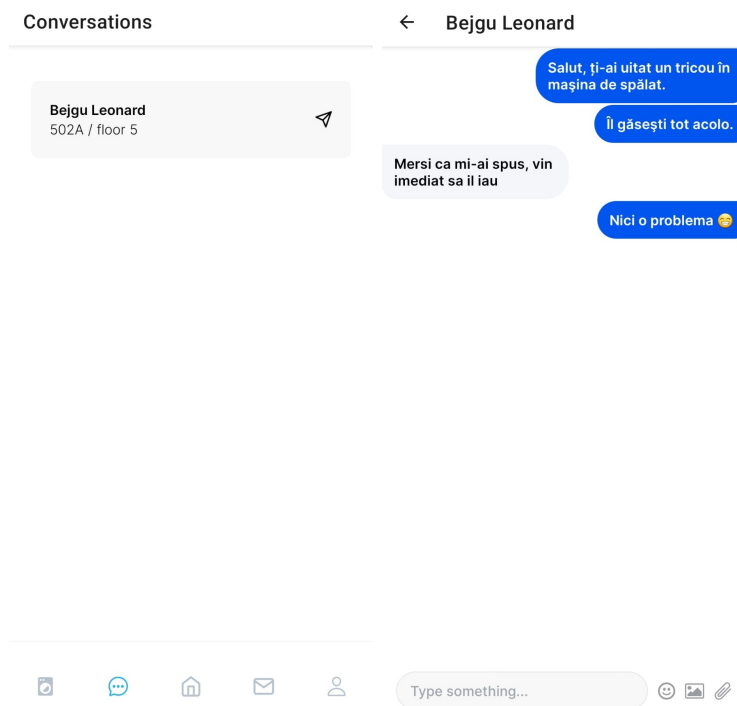


Figura 4.14: Simularea unei conversații între 2 studenți, realizată prin 2 instanțe de client, telefon și emulator Android

Atunci când un utilizator selectează o conversație din această listă, este deschisă fereastra de chat pentru acea conversație. Aici, utilizatorul poate vedea istoricul mesajelor schimbate cu persoana respectivă și poate trimite noi mesaje.

În plus, aplicația permite inițierea de noi conversații direct din meniul de mașini de spălat. Dacă un utilizator dorește să discute cu o anumită persoană - poate pentru a coordona utilizarea unei mașini de spălat sau pentru a discuta despre o problemă - poate selecta butonul de chat asociat cu acea persoană. Aceasta va crea o nouă conversație, permitându-le să înceapă imediat discuția.

4.7.2 Conectarea la socket folosind Socket.io

În cadrul aplicației, pentru a permite comunicarea în timp real între utilizatori, s-a folosit biblioteca socket.io, o bibliotecă JavaScript populară pentru comunicare în timp real între servere și clienți prin utilizarea de websockets.[15]

Procesul de comunicare cu Socket.io funcționează astfel:

1. Când un utilizator trimite un mesaj prin chat, frontend-ul trimite un obiect de tipul „mesaj” la server prin canalul (sau „eventul”) Socket.io numit „send message”. Acest obiect de mesaj conține detaliile mesajului, inclusiv conținutul acestuia și informațiile despre sender și receiver.
2. Serverul backend interceptează acest obiect de mesaj care vine pe canalul „send message”. Serverul apoi actualizează baza de date cu detaliile acestui nou mesaj.
3. După ce a înregistrat cu succes mesajul în baza de date, serverul trimite înapoi mesajul la aplicația frontend prin canalul Socket.io numit „new message”. Acesta este un eveniment care notifică frontend-ul că un nou mesaj a fost primit.
4. În frontend, după ce mesajul a fost primit pe canalul „new message”, este adăugat la store-ul Zustand. În acest caz, Zustand gestionează lista de mesaje din chat.
5. În cele din urmă, după ce mesajul a fost adăugat la store-ul Zustand, este afișat pe ecranul de chat al utilizatorului.

4.8 Reprogramarea mai rapidă

Respectarea corectă a programărilor utilizatorilor și a timpilor de rulare a mașinii de spălat este esențială pentru a asigura o funcționare eficientă și justă a sistemului de rezervare. Atunci când o mașină de spălat devine disponibilă mai devreme decât era prevăzut inițial (de exemplu, dacă utilizatorul curent își termină spălătoria mai devreme sau anulează rezervarea), este posibil să existe o fereastră de timp neutilizată care poate fi folosită de alți utilizatori.

Cu toate acestea, este important ca această disponibilitate suplimentară să fie gestionată într-un mod echitabil, astfel încât să nu se permită altor utilizatori să intre înaintea celor care sunt deja pe lista de așteptare. Pentru a rezolva această problemă, se utilizează un algoritm care mută rezervările în funcție de disponibilitatea timpului.

Prin utilizarea acestui algoritm, sistemul de rezervare poate să-și ajusteze în mod dinamic programările pentru a maximiza utilizarea mașinilor de spălat și pentru a asigura că toți utilizatorii sunt tratați în mod echitabil. Acest lucru îmbunătățește atât eficiența generală a spălătoriei, cât și satisfacția utilizatorilor.

4.8.1 Algoritmul de reprogramare: Problema rucsacului

Problema rucsacului este o problemă clasică de optimizare în care trebuie să alegem un subset de obiecte dintr-o mulțime dată, astfel încât suma valorilor lor să fie maximă, respectând un anumit criteriu de greutate.

Fie n obiecte, fiecare cu o valoare v_i și o greutate w_i . Trebuie să selectăm obiectele astfel încât suma valorilor să fie maximă, dar suma greutăților să nu depășească o capacitate dată W a rucsacului.[1]

Această problemă poate fi rezolvată utilizând tehnica programării dinamice. Putem defini o matrice dp , în care $dp[i][j]$ reprezintă valoarea maximă pe care o putem obține folosind primele i obiecte și având o capacitate maximă de j .

Algoritmul de programare dinamică pentru problema rucsacului este următorul [1]:

- Inițializăm matricea dp cu valori de 0.
- Pentru fiecare obiect i de la 1 la n și pentru fiecare capacitate j de la 1 la W , facem:
 - Dacă greutatea obiectului i este mai mică sau egală decât capacitatea j , calculăm valoarea maximă între a nu lua obiectul i și a

lua obiectul i :

$$dp[i][j] = \max(dp[i-1][j], v_i + dp[i-1][j - w_i])$$

- Altfel, copiem valoarea maximă de la rândul anterior din matricea dp :

$$dp[i][j] = dp[i-1][j]$$

- Valoarea maximă pe care o putem obține se află în celula $dp[n][W]$.

Această abordare de programare dinamică ne permite să rezolvăm problema rucsacului într-un mod eficient, evitând suprapunerea de subprobleme și reducând complexitatea algoritmului.

În această situație, „rucsacul” este intervalul de timp liber care a devenit disponibil, iar „obiectele” sunt rezervările care pot fi mutate în acest interval. Fiecare rezervare are o „greutate” (durata sa) și o „valoare” (beneficiul obținut prin mutarea acesteia în intervalul de timp liber). Scopul este de a maximiza „valoarea” totală (adică, a acoperi cât mai mult din timpul liber) fără a depăși „greutatea” totală (adică, durata intervalului de timp liber).

4.8.2 Detalii de implementare

Mai jos sunt explicați pașii realizați în procesul de reprogramare:

1. Identificarea ferestrei libere

Primul pas în procesul de reprogramare este identificarea unei ferestre libere de timp. Aceasta poate apărea în urma terminării mai devreme a unei rezervări sau ca urmare a anulării unei rezervări.

2. Crearea listei de rezervări care pot fi reprogramate

În continuare, se creează o listă cu toate rezervările care pot fi reprogramate. Acestea sunt rezervările care sunt programate pentru momente ulterioare și care ar putea fi mutate în fereastra liberă de timp, în acest caz sunt cele care au bifat opțiunea de a fi mutate mai devreme din ecranul de rezervări.

```
const intervals: reservation[] =  
await prisma.$queryRaw`  
SELECT * FROM reservation  
WHERE reservation.scheduled_early = true  
AND reservation.start_hour > ${startTime}::timestamp`;
```

3. Sortarea rezervărilor în funcție de durată

Lista de rezervări este sortată în funcție de durată, de la cea mai lungă la cea mai scurtă. Aceasta este o etapă crucială a algoritmului, deoarece rezervările mai lungi vor acoperi o mai mare parte din fereastra liberă de timp.

```
const sortedIntervals = intervals.sort((r1, r2) =>
  r1.end_hour.getTime() - r1.start_hour.getTime() <
  r2.end_hour.getTime() - r2.start_hour.getTime()
    ? -1
    : 1
  );
```

4. Mutarea rezervărilor

Se aplică algoritmul cu formula de mai sus, în felul următor:

```
for (let i = 1; i <= n; i++) {
  for (let j = 1; j <= intervalLength; j++) {
    if (getValue(sortedIntervals[i - 1]) <= j) {
      dp[i][j] = Math.max(
        dp[i - 1][j],
        dp[i - 1][j - getValue(sortedIntervals[i - 1])] +
        getValue(sortedIntervals[i - 1])
      );
    } else {
      dp[i][j] = dp[i - 1][j];
    }
  }
}
```

5. Confirmarea reprogramărilor

În final, toate reprogramările sunt confirmate, iar utilizatorii sunt notificați cu privire la noile lor orare de rezervare. Acestea pot fi vizualizate în secțiunea de istoric a rezervărilor din aplicație.

4.9 Profilul personal al utilizatorului

În cadrul aplicației, secțiunea de profil personal este esențială pentru gestionarea și personalizarea experienței utilizatorului. Această secțiune oferă o imagine de ansamblu asupra informațiilor personale ale utilizatorului și oferă funcționalități care pot îmbunătăți modul în care interacționează cu aplicația.

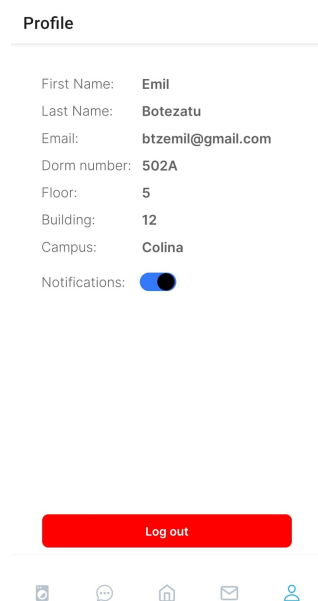


Figura 4.15: Vizualizarea detaliilor user-ului autentificat din ecranul *Profile*

În acest ecran se afișează toate detaliile relevante asociate cu contul utilizatorului. Aceasta include numele complet, adresa de e-mail, numărul camerei, etajul pe care se află camera, numărul clădirii și complexul din care face parte.

Aplicația oferă utilizatorului posibilitatea de a controla modul în care primește notificările. Aceasta poate fi o opțiune esențială, deoarece nu toți utilizatorii doresc să fie contactați constant cu actualizări. Astfel, utilizatorul are posibilitatea de a alege dacă dorește să fie abonat sau dezabonat de la notificări cu butonul de tip *switch*.

Pentru a asigura securitatea contului, este important ca utilizatorii să aibă posibilitatea de a se deconecta de la contul lor. Butonul de logout permite acest lucru, închizând sesiunea curentă și redirecționând utilizatorul înapoi la ecranul de autentificare.

4.10 Persistența datelor

Pentru păstrarea și gestionarea datelor într-o aplicație, se folosește de obicei o bază de date. Aceasta permite stocarea persistentă a informațiilor, ceea ce înseamnă că datele rămân disponibile și după ce aplicația a fost închisă sau chiar dacă dispozitivul este repornit. Schema bazei de date arată organizarea și structura acesteia.

4.10.1 Schema bazei de date

Schema bazei de date Supabase arată asemănător cu orice altă bază de date relațională. Mai jos este prezentată diagrama ERD a bazei de date:

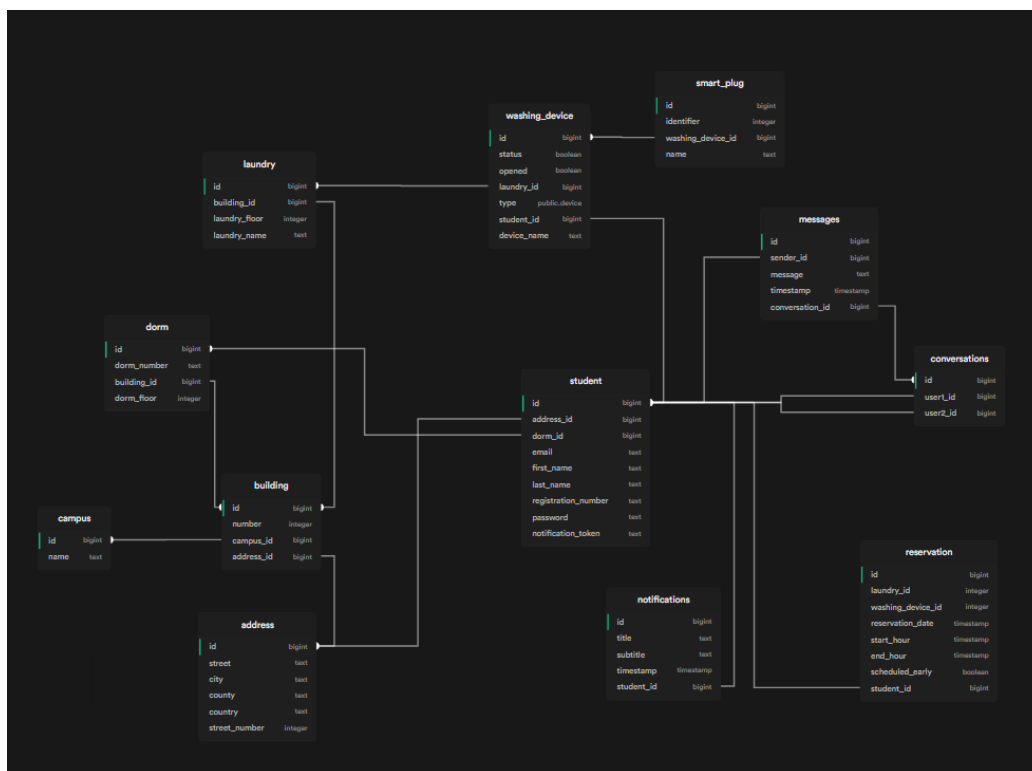


Figura 4.16: Diagrama ERD

Identificăm 12 tabele, fiecare având un rol bine definit:

- **student**: sunt stocate datele studentului împreună cu cheile străine specifice identificării acestuia în campus
- **address**: reprezintă adresele stocate

- **campus:** reprezintă campusul și numele acestuia
- **bulding:** reprezintă clădirile din campus. Acestea sunt legate cu cheie straină la tabela campus
- **dorm:** reprezintă camera din care face parte studentul
- **laundry:** reprezintă spălătoriile din căminul respectiv
- **washing_device:** reprezintă tabelul ce conține datele dispozitivului. Acesta poate să ia 2 valori pentru tip: Mașină de spalat sau Uscător și este identificat printr-un enum, denumit *device*. În acest tabel avem datele utilizatorului care a folosit dispozitivul ultima oară, precum și starea activă al acestuia
- **reservation:** sunt stocate datele rezervării
- **smart_plug:** tabelul ce reprezintă datele prizei smart și un identificator unic
- **conversations:** sunt stocate toate conversațiile începute
- **messages:** sunt stocate toate mesajele dintre 2 utilizatori
- **notifications:** tabelul unde apar notificările primite de un anumit student

Capitolul 5

Concluzii și perspective de dezvoltare

5.1 Concluzii generale

Această aplicație de gestionare a unei spălătorii este un exemplu excelent prin care arătăm cum tehnologia și dezvoltarea de software pot fi utilizate pentru a optimiza și digitaliza operațiunile zilnice, făcând viața utilizatorilor mai ușoară. Prin folosirea unei astfel de aplicații, studenții sau alți utilizatori pot rezerva și monitoriza în timp real mașinile de spălat, optimizându-și astfel timpul.

Folosirea unei arhitecturi de microservicii, a programării reactive și a comunicării în timp real prin websockets adaugă un nivel suplimentar de complexitate, dar și de funcționalitate și reactivitate aplicației. În plus, utilizarea bazelor de date relaționale cloud și a unui ORM precum Prisma facilitează manipularea și gestionarea datelor.

De asemenea, securitatea și confidențialitatea datelor utilizatorilor au fost luate în considerare, prin implementarea de module de autentificare și autorizare, precum și prin stocarea securizată a parolelor.

Această aplicație a fost dezvoltată ca răspuns la o problemă reală pe care am identificat-o în cadrul vieții mele de student și pe care o experimentează de asemenea mulți alți studenți. Gestionarea utilizării mașinilor de spălat într-un mediu universitar poate fi dificilă și consumatoare de timp. Probleme precum așteptarea la rând pentru o mașină disponibilă, lipsa de comunicare între studenți cu privire la starea mașinilor de spălat sau nefolosirea eficientă a mașinilor de spălat sunt situații comune care necesită o soluție.

Prin crearea acestei aplicații, obiectivul meu a fost de a aborda aceste provocări și de a îmbunătăți experiența studenților cu privire la utilizarea

spălătoriilor.

5.2 Perspective de dezvoltare

Pentru viitor, această aplicație are un mare potențial de extindere și îmbunătățire. Iată câteva dintre perspectivele de dezvoltare:

- **Integrarea cu alte aplicații sau servicii:** Aplicația poate fi integrată cu alte servicii sau aplicații relevante pentru studenți, cum ar fi baza de date a universității pentru unificare tabelor de utilizatori.
- **Predicția consumului de energie cu ML:** Modelul de ML poate fi antrenat să recunoască modelele de consum de energie asociate cu diferitele etape ale unui ciclu de spălare. Acest lucru ar putea permite o predicție mai precisă a momentului în care o mașină de spălat este gata și când este cel mai probabil să se oprească.
- **Detectarea deschiderii ușii prin senzori:** După terminarea unui ciclu de spălare, senzorul ar detecta dacă ușa mașinii de spălat a fost deschisă. Senzorul de ușă ar putea ajuta la verificarea disponibilității mașinii de spălat.
- **Implementarea unui cont de admin:** Administratorul aplicației ar putea avea capacitatea de a modifica starea mașinilor de spălat, de a gestiona utilizatorii sau de a vizualiza statistici legate de consumul de curent.

Bibliografie

- [1] *0/1 Knapsack Problem*. <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>. Accesat la: 2023-06-13.
- [2] *Adopt TypeScript Gradually*. <https://www.typescriptlang.org/>. Accesat la: 2023-03-24.
- [3] *Application programming interface (API)*. <https://en.wikipedia.org/wiki/API>. Accesat la: 2023-06-19.
- [4] *Expo Notifications*. <https://docs.expo.dev/versions/latest/sdk/notifications/>. Accesat la: 2023-06-05.
- [5] *Flashing custom firmware on a Gosund SP111*. <https://www.malachisoord.com/2019/11/24/flashing-custom-firmware-on-a-gosund-sp111/>. Accesat la: 2023-03-08.
- [6] *Git/GitHub branching standards & conventions*. <https://gist.github.com/digitaljhelms/4287848>. Accesat la: 2023-06-14.
- [7] *Making ‘dumb’ Dishwashers and Washing Machines Smart: Alerts When the Dishes and Clothes Are Cleaned*. <https://philhawthorne.com/making-dumb-dishwashers-and-washing-machines-smart-alerts-when-the-dishes-and-clothes-are-cleaned/>. Accesat la: 2023-02-18.
- [8] *MQ Telemetry Transport (MQTT)*. <https://en.wikipedia.org/wiki/MQTT>. Accesat la: 2023-04-10.
- [9] *Next-generation Node.js and TypeScript ORM*. <https://www.prisma.io/>. Accesat la: 2023-04-05.
- [10] *Node-RED. Get Started*. <https://nodered.org/#get-started>. Accesat la: 2023-04-12.

- [11] *Node.js with TypeScript*. <https://nodejs.dev/en/learn/nodejs-with-typescript/>. Accesat la: 2023-01-15.
- [12] *Past, Present, and Future of React State Management*. <https://leerob.io/blog/react-state-management>. Accesat la: 2023-06-19.
- [13] *React Native. Get Started*. <https://reactnative.dev/docs/getting-started>. Accesat la: 2023-01-18.
- [14] *Relational database*. https://en.wikipedia.org/wiki/Relational_database. Accesat la: 2023-05-10.
- [15] *Socket.IO*. <https://socket.io/>. Accesat la: 2023-03-02.
- [16] *Supabase Documentation*. <https://supabase.com/docs>. Accesat la: 2023-04-26.
- [17] *Tamagui. Performance tests and comparisons*. <https://tamagui.dev/docs/intro/benchmarks>. Accesat la: 2023-06-05.
- [18] *TanStack Query*. <https://tanstack.com/query/v3/>. Accesat la: 2023-02-10.
- [19] *Tasmota*. <https://tasmota.github.io/docs/>. Accesat la: 2023-03-08.
- [20] *Typescript. Type Manipulation*. <https://www.typescriptlang.org/docs/handbook/2/everyday-types.html>. Accesat la: 2023-02-27.
- [21] *What is Axios?* <https://axios-http.com/docs/intro>. Accesat la: 2023-06-18.
- [22] *Why choose Expo?* <https://docs.expo.dev/faq/>. Accesat la: 2023-06-18.
- [23] *Zustand library*. <https://github.com/pmndrs/zustand>. Accesat la: 2023-06-16.
- [24] Mario Casciaro. *Node.js Design Patterns*. 2014. Accesat la: 2023-03-24.
- [25] Abid Haleem, Mohd Javaid, Mohd Asim Qadri, and Rajiv Suman. *Understanding the role of digital technologies in education: A review*. <https://www.sciencedirect.com/science/article/pii/S2666412722000137>, 2022. Accesat la: 2023-06-01.

- [26] IEEE. *Research and Application of Node.js Core Technology*. <https://ieeexplore.ieee.org/document/9424850>, 2020. Accesat la: 2023-04-15.
- [27] Rishika Mehta, Jyoti Sahnia, and Kavita Khanna. *Internet of Things: Vision, Applications and Challenges*. <https://www.sciencedirect.com/science/article/pii/S1877050918307749>. Accesat la: 2023-06-20.
- [28] Dan Vanderkam. *Effective TypeScript*. Accesat la: 2023-01-02.